

# amira<sup>®</sup> 4.1

Amira Reference Guide

## **Copyright Information**

©1995-2005 Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany

©1999-2005 Mercury Computer Systems.

All rights reserved.

## **Trademark Information:**

Amira is being jointly developed by Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Mercury Computer Systems.

Amira® is a registered trademark of Konrad-Zuse-Zentrum für Informationstechnik Berlin.

HardCopy, MeshViz, VolumeViz, TerrainViz are trademarks of Mercury Computer Systems S.A.

Mercury Computer Systems S.A. is a source licensee of OpenGL®, Open Inventor® from Silicon Graphics, Inc. OpenGL® and Open Inventor® are registered trademarks of Silicon Graphics, Inc.

All other products and company names are trademarks or registered trademarks of their respective companies.

This manual has been prepared for Mercury Computer Systems licensees solely for use in connection with software supplied by Mercury Computer Systems and is furnished under a written license agreement. This material may not be used, reproduced or disclosed, in whole or in part, except as permitted in the license agreement or by prior written authorization of Mercury Computer Systems. Users are cautioned that Mercury Computer Systems reserves the right to make changes without notice to the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic or listing errors.

# Contents

|          |                                    |          |
|----------|------------------------------------|----------|
| <b>I</b> | <b>Alphabetic Index of Modules</b> | <b>1</b> |
| <b>1</b> | <b>Amira</b>                       | <b>3</b> |
| 1.1      | Access LargeDiskData . . . . .     | 3        |
| 1.2      | AffineRegistration . . . . .       | 4        |
| 1.3      | AlignPrincipalAxes . . . . .       | 7        |
| 1.4      | AlignSlices . . . . .              | 8        |
| 1.4.1    | Tool Bar . . . . .                 | 9        |
| 1.4.2    | Menu Bar . . . . .                 | 11       |
| 1.4.3    | Alignment Methods . . . . .        | 16       |
| 1.4.4    | Image Viewer . . . . .             | 17       |
| 1.4.5    | Key Bindings . . . . .             | 18       |
| 1.5      | Animate . . . . .                  | 22       |
| 1.6      | Annotation . . . . .               | 24       |
| 1.7      | AnonymizeImageStack . . . . .      | 25       |
| 1.8      | ApplyBSplineTransform . . . . .    | 25       |
| 1.9      | ApplyTransform . . . . .           | 26       |
| 1.10     | ArbitraryCut . . . . .             | 28       |
| 1.11     | Arithmetic . . . . .               | 30       |
| 1.12     | Axis . . . . .                     | 34       |
| 1.13     | BoundingBox . . . . .              | 35       |
| 1.14     | CalculusMatlab . . . . .           | 36       |

|      |                       |    |
|------|-----------------------|----|
| 1.15 | CastField             | 39 |
| 1.16 | ChannelWorks          | 41 |
| 1.17 | ClippingPlane         | 42 |
| 1.18 | ClusterDiff           | 42 |
| 1.19 | ClusterGrep           | 44 |
| 1.20 | ClusterLabel          | 45 |
| 1.21 | ClusterSample         | 46 |
| 1.22 | ClusterView           | 47 |
| 1.23 | ColorCombine          | 49 |
| 1.24 | Colorwash             | 51 |
| 1.25 | CombineLandmarks      | 52 |
| 1.26 | CompareLatticeData    | 53 |
| 1.27 | ComputeContours       | 54 |
| 1.28 | ConnectedComponents   | 54 |
| 1.29 | ContrastControl       | 56 |
| 1.30 | CorrelationPlot       | 59 |
| 1.31 | CreateCluster         | 62 |
| 1.32 | Curl                  | 63 |
| 1.33 | CurvedSlice           | 63 |
| 1.34 | Cutting Plane         | 66 |
| 1.35 | CylinderSlice         | 67 |
| 1.36 | DataProbe             | 68 |
| 1.37 | Delaunay2D            | 71 |
| 1.38 | DemoMaker             | 73 |
| 1.39 | Digital Image Filters | 83 |
| 1.40 | DisplayColormap       | 83 |
| 1.41 | DisplayLogos          | 85 |
| 1.42 | DisplayTime           | 86 |
| 1.43 | GetCurvature          | 87 |
| 1.44 | GridView              | 89 |

|        |                                                      |     |
|--------|------------------------------------------------------|-----|
| 1.45   | Grouping . . . . .                                   | 90  |
| 1.46   | HeightField . . . . .                                | 91  |
| 1.47   | Histogram . . . . .                                  | 92  |
| 1.48   | InterpolLabels . . . . .                             | 94  |
| 1.49   | InterpolateLabels . . . . .                          | 95  |
| 1.50   | Intersect . . . . .                                  | 95  |
| 1.51   | Isolines . . . . .                                   | 97  |
| 1.52   | Isolines (Surface) . . . . .                         | 98  |
| 1.53   | Isosurface (Regular) . . . . .                       | 99  |
| 1.54   | IvDisplay . . . . .                                  | 101 |
| 1.55   | IvToSurface . . . . .                                | 101 |
| 1.56   | LabelVoxel . . . . .                                 | 102 |
| 1.57   | LandmarkSurfaceWarp . . . . .                        | 104 |
| 1.58   | LandmarkView . . . . .                               | 104 |
| 1.59   | LandmarkWarp . . . . .                               | 105 |
| 1.60   | LatticeAccess . . . . .                              | 107 |
| 1.61   | LegoSurfaceGen . . . . .                             | 108 |
| 1.62   | LineProbe . . . . .                                  | 109 |
| 1.63   | LineSetProbe . . . . .                               | 109 |
| 1.64   | LineSetView . . . . .                                | 110 |
| 1.65   | Measuring . . . . .                                  | 114 |
| 1.66   | Merge . . . . .                                      | 115 |
| 1.67   | MovieMaker . . . . .                                 | 116 |
| 1.68   | MoviePlayer . . . . .                                | 118 |
| 1.68.1 | The amira Movie Format . . . . .                     | 119 |
| 1.68.2 | Optimized amira Movies . . . . .                     | 119 |
| 1.68.3 | Which movie or image format should I use ? . . . . . | 119 |
| 1.69   | Normalize Image . . . . .                            | 122 |
| 1.70   | ObliqueSlice . . . . .                               | 123 |
| 1.71   | OrthoSlice . . . . .                                 | 126 |

|        |                                       |     |
|--------|---------------------------------------|-----|
| 1.72   | OrthoViewCursor                       | 128 |
| 1.73   | PointProbe                            | 130 |
| 1.74   | PointWrap                             | 130 |
| 1.75   | ProbeToLineSet                        | 131 |
| 1.76   | ProjectionView                        | 131 |
| 1.77   | ProjectionViewCursor                  | 133 |
| 1.78   | RefineTetras                          | 134 |
| 1.79   | Relabel                               | 135 |
| 1.80   | RelabelTetras                         | 136 |
| 1.81   | Resample                              | 138 |
| 1.81.1 | Resampling non-labeled data fields    | 138 |
| 1.81.2 | Resampling labeled data fields        | 139 |
| 1.81.3 | Coordinates of the resampled data set | 139 |
| 1.81.4 | Resampling in progress                | 139 |
| 1.82   | Scale                                 | 140 |
| 1.83   | ScanConvertSurface                    | 142 |
| 1.84   | SelectLines                           | 143 |
| 1.85   | SelectRoi                             | 144 |
| 1.86   | Shear                                 | 145 |
| 1.87   | SmoothSurface                         | 145 |
| 1.88   | Sound                                 | 146 |
| 1.89   | SplineProbe                           | 147 |
| 1.90   | StandardView                          | 147 |
| 1.91   | SurfaceArea                           | 149 |
| 1.92   | SurfaceCut                            | 150 |
| 1.93   | SurfaceDistance                       | 151 |
| 1.94   | SurfaceField                          | 153 |
| 1.95   | SurfaceGen                            | 153 |
| 1.96   | SurfaceView                           | 155 |
| 1.97   | TimeHistoryPlot                       | 157 |

|          |                           |            |
|----------|---------------------------|------------|
| 1.98     | TimeSeriesControl         | 157        |
| 1.99     | TissueStatistics          | 159        |
| 1.100    | TriangleDistortion        | 161        |
| 1.101    | VRML-Export               | 162        |
| 1.102    | Vertex Morph              | 164        |
| 1.103    | VertexDiff                | 165        |
| 1.104    | VertexShift               | 165        |
| 1.105    | VertexView                | 166        |
| 1.106    | ViewBase                  | 170        |
| 1.107    | VolPro-1000               | 173        |
| 1.108    | Voltex                    | 178        |
| 1.109    | VolumeEdit                | 181        |
| 1.110    | VoxelView                 | 183        |
| <b>2</b> | <b>Molecular Pack</b>     | <b>185</b> |
| 2.1      | AlignMolecules            | 185        |
| 2.2      | AlignSequences            | 187        |
| 2.3      | BondAngleView             | 189        |
| 2.4      | BondCalculation           | 190        |
| 2.5      | CompMolInterface          | 191        |
| 2.6      | CompMolSurface            | 192        |
| 2.7      | ComputeHAtoms             | 195        |
| 2.8      | ComputeHBonds             | 195        |
| 2.9      | ComputeSecondaryStructure | 196        |
| 2.10     | ConfigurationDensity      | 197        |
| 2.11     | CreateSphere              | 199        |
| 2.12     | HBondView                 | 201        |
| 2.13     | MeanMolecule              | 202        |
| 2.14     | Measurement               | 203        |
| 2.15     | MolElectrostatics         | 204        |
| 2.16     | MolSurfaceView            | 205        |

|          |                            |            |
|----------|----------------------------|------------|
| 2.17     | MoleculeLabel              | 207        |
| 2.18     | MoleculeView               | 210        |
| 2.19     | Observables                | 212        |
| 2.20     | PrecomputeAlignment        | 213        |
| 2.21     | PseudoElectronDensity      | 214        |
| 2.22     | RankTimeStep               | 215        |
| 2.23     | SecStructureView           | 217        |
| 2.24     | TubeView                   | 221        |
| <b>3</b> | <b>VR Pack</b>             | <b>223</b> |
| 3.1      | VR Pack control module     | 223        |
| 3.2      | VR Pack show config module | 226        |
| <b>4</b> | <b>Large Data Pack</b>     | <b>231</b> |
| 4.1      | ConvertToLargeDiskData     | 231        |
| 4.2      | Isosurface (LDA)           | 232        |
| 4.3      | LDAExpertSettings          | 233        |
| 4.4      | ObliqueSlice (LDA)         | 236        |
| 4.5      | OrthoSlice (LDA)           | 238        |
| 4.6      | SelectRoi (LDA)            | 241        |
| 4.7      | Voltex (LDA)               | 243        |
| <b>5</b> | <b>Mesh Pack</b>           | <b>249</b> |
| 5.1      | AlignSurfaces              | 249        |
| 5.2      | AnalyticTensorField        | 250        |
| 5.3      | BoundaryConditions         | 251        |
| 5.4      | ComponentField             | 252        |
| 5.5      | ComputeTensor              | 253        |
| 5.6      | ComputeTensorOutOfCore     | 254        |
| 5.7      | ConePlot                   | 256        |
| 5.8      | ContourView                | 259        |
| 5.9      | Displace                   | 259        |



|      |                         |     |
|------|-------------------------|-----|
| 5.10 | DisplayISL              | 260 |
| 5.11 | Divergence              | 262 |
| 5.12 | DoseVolume (Tetrahedra) | 263 |
| 5.13 | DuplicateNodes          | 264 |
| 5.14 | EigenvectorToColor      | 264 |
| 5.15 | ExtractEigenvalues      | 266 |
| 5.16 | FiberTracking           | 266 |
| 5.17 | FieldCut                | 268 |
| 5.18 | FieldCut (Polyhedra)    | 270 |
| 5.19 | Gradient                | 271 |
| 5.20 | GridBoundary            | 272 |
| 5.21 | GridCut                 | 274 |
| 5.22 | GridCut (Polyhedra)     | 275 |
| 5.23 | GridVolume              | 276 |
| 5.24 | GridVolume (Polyhedra)  | 278 |
| 5.25 | HexToTet                | 279 |
| 5.26 | HexaView                | 280 |
| 5.27 | IlluminatedLines        | 281 |
| 5.28 | Interpolate             | 282 |
| 5.29 | Isosurface (Hexahedra)  | 283 |
| 5.30 | Isosurface (Polyhedra)  | 284 |
| 5.31 | Isosurface (Tetrahedra) | 285 |
| 5.32 | Lambda2                 | 286 |
| 5.33 | LatToHex                | 287 |
| 5.34 | LineStreaks             | 287 |
| 5.35 | MagAndPhase             | 288 |
| 5.36 | Magnitude               | 289 |
| 5.37 | ParametricSurface       | 290 |
| 5.38 | ParticlePlot            | 292 |
| 5.39 | PlanarLIC               | 295 |

|           |                                  |            |
|-----------|----------------------------------|------------|
| 5.40      | RateOfStrainTensor               | 297        |
| 5.41      | SeedSurface                      | 297        |
| 5.42      | Splats                           | 299        |
| 5.43      | StreamRibbons                    | 300        |
| 5.44      | StreamSurface                    | 302        |
| 5.45      | SurfaceLIC                       | 304        |
| 5.46      | TensorDisplay                    | 306        |
| 5.47      | TetToHex                         | 308        |
| 5.48      | TetraCombine                     | 308        |
| 5.49      | TetraGen                         | 309        |
| 5.50      | TetraQuality                     | 311        |
| 5.51      | TriangleQuality                  | 312        |
| 5.52      | VectorProbe                      | 313        |
| 5.53      | Vectors                          | 314        |
| 5.54      | Vectors (Tetrahedra)             | 316        |
| 5.55      | Vectors/Normals (Surface)        | 317        |
| <b>6</b>  | <b>Quantification+ Pack</b>      | <b>321</b> |
| 6.1       | Visilog                          | 321        |
| 6.2       | VisilogAnalyse                   | 326        |
| 6.3       | VisilogThreshold                 | 327        |
| <b>II</b> | <b>Alphabetic Index of Ports</b> | <b>329</b> |
| <b>7</b>  | <b>Amira</b>                     | <b>331</b> |
| 7.1       | Port                             | 331        |
| 7.2       | Connection                       | 333        |
| 7.3       | MasterConnection                 | 334        |
| 7.4       | Port3DPointList                  | 335        |
| 7.5       | PortButtonList                   | 337        |
| 7.6       | PortButtonMenu                   | 339        |

|      |                             |     |
|------|-----------------------------|-----|
| 7.7  | PortChannelConfig . . . . . | 340 |
| 7.8  | PortColorList . . . . .     | 341 |
| 7.9  | PortColormap . . . . .      | 342 |
| 7.10 | PortDoIt . . . . .          | 343 |
| 7.11 | PortDrawStyle . . . . .     | 344 |
| 7.12 | PortFilename . . . . .      | 345 |
| 7.13 | PortFloatSlider . . . . .   | 346 |
| 7.14 | PortFloatTextN . . . . .    | 348 |
| 7.15 | PortGeneric . . . . .       | 349 |
| 7.16 | PortInfo . . . . .          | 352 |
| 7.17 | PortIntSlider . . . . .     | 352 |
| 7.18 | PortIntTextN . . . . .      | 353 |
| 7.19 | PortMultiChannel . . . . .  | 353 |
| 7.20 | PortMultiMenu . . . . .     | 354 |
| 7.21 | PortRadioBox . . . . .      | 355 |
| 7.22 | PortSeparator . . . . .     | 356 |
| 7.23 | PortTabBar . . . . .        | 356 |
| 7.24 | PortText . . . . .          | 356 |
| 7.25 | PortTime . . . . .          | 357 |
| 7.26 | PortToggleList . . . . .    | 359 |

### III    **Alphabetic Index of File Formats** **361**

|          |                                               |            |
|----------|-----------------------------------------------|------------|
| <b>8</b> | <b>Amira</b>                                  | <b>363</b> |
| 8.1      | Amira Script . . . . .                        | 363        |
| 8.2      | Amira Script Object . . . . .                 | 363        |
| 8.3      | AmiraMesh Format . . . . .                    | 363        |
| 8.3.1    | Fields with uniform coordinates . . . . .     | 366        |
| 8.3.2    | Fields with stacked coordinates . . . . .     | 367        |
| 8.3.3    | Fields with rectilinear coordinates . . . . . | 368        |

|       |                                               |     |
|-------|-----------------------------------------------|-----|
| 8.3.4 | Fields with curvilinear coordinates . . . . . | 369 |
| 8.3.5 | Label fields for segmentation . . . . .       | 370 |
| 8.3.6 | Landmarks for registration . . . . .          | 371 |
| 8.3.7 | Line segments . . . . .                       | 372 |
| 8.3.8 | Colormaps . . . . .                           | 373 |
| 8.4   | AmiraMesh as LargeDiskData . . . . .          | 374 |
| 8.5   | Analyze 7.5 . . . . .                         | 374 |
| 8.6   | AnalyzeAVW . . . . .                          | 375 |
| 8.7   | BMP Image Format . . . . .                    | 375 |
| 8.8   | DXF . . . . .                                 | 375 |
| 8.9   | Encapsulated PostScript . . . . .             | 376 |
| 8.10  | HTML . . . . .                                | 376 |
| 8.11  | HxSurface . . . . .                           | 376 |
| 8.12  | Icol . . . . .                                | 381 |
| 8.13  | Interfile . . . . .                           | 381 |
| 8.14  | JPEG Image Format . . . . .                   | 382 |
| 8.15  | MATLAB Binary Format (.mat) . . . . .         | 382 |
| 8.16  | MATLAB M-files Format (.m) . . . . .          | 382 |
| 8.17  | Open Inventor . . . . .                       | 382 |
| 8.18  | PNG Image Format . . . . .                    | 383 |
| 8.19  | PNM Image Format . . . . .                    | 383 |
| 8.20  | PSI format . . . . .                          | 383 |
| 8.21  | Ply Format . . . . .                          | 385 |
| 8.22  | Raw Data . . . . .                            | 385 |
| 8.23  | Raw Data as LargeDiskData . . . . .           | 387 |
| 8.24  | SGI-RGB Image Format . . . . .                | 387 |
| 8.25  | STL . . . . .                                 | 387 |
| 8.26  | Stacked-Slices . . . . .                      | 387 |
| 8.27  | TIFF Image Format . . . . .                   | 388 |
| 8.28  | Tecplot . . . . .                             | 390 |

|           |                                 |            |
|-----------|---------------------------------|------------|
| 8.29      | VRML                            | 391        |
| <b>9</b>  | <b>Molecular Pack</b>           | <b>393</b> |
| 9.1       | AMBER                           | 393        |
| 9.2       | AMF                             | 394        |
| 9.3       | GROMACS                         | 397        |
| 9.4       | MDL                             | 398        |
| 9.5       | PDB                             | 398        |
| 9.6       | PSF/DCD (CHARMM)                | 400        |
| 9.7       | Tripos                          | 400        |
| 9.8       | UniChem                         | 400        |
| 9.9       | ZIB Molecular File Format       | 401        |
| <b>10</b> | <b>Large Data Pack</b>          | <b>409</b> |
| 10.1      | LDA                             | 409        |
| 10.2      | LargeDiskData                   | 409        |
| 10.3      | Stacked-Slices as LargeDiskData | 410        |
| <b>11</b> | <b>Mesh Pack</b>                | <b>411</b> |
| 11.1      | AVS Field                       | 411        |
| 11.2      | AVS UCD Format                  | 412        |
| 11.3      | FIDAP NEUTRAL                   | 412        |
| 11.4      | Fluent / UNS                    | 412        |
| 11.5      | Hypermesh                       | 412        |
| 11.6      | IDEAS universal format          | 413        |
| 11.7      | Plot 3D Single Structured       | 413        |
| <b>12</b> | <b>DICOM reader</b>             | <b>417</b> |
| 12.1      | ACR-NEMA                        | 417        |
| 12.2      | DICOM                           | 417        |
| <b>13</b> | <b>Madymo reader (TNO)</b>      | <b>423</b> |

|           |                                       |            |
|-----------|---------------------------------------|------------|
| 13.1      | Madymo                                | 423        |
| 13.2      | Madymo Time-history                   | 423        |
| <b>14</b> | <b>Mecalog Radioss reader</b>         | <b>425</b> |
| 14.1      | Radioss                               | 425        |
| <b>IV</b> | <b>Alphabetic Index of Data Types</b> | <b>427</b> |
| <b>15</b> | <b>Amira</b>                          | <b>429</b> |
| 15.1      | AnnaScalarField3                      | 429        |
| 15.2      | AnnaVectorField3                      | 430        |
| 15.3      | B-Splines                             | 432        |
| 15.4      | CameraRotate                          | 432        |
| 15.5      | Cluster                               | 433        |
| 15.6      | ColorField3                           | 435        |
| 15.7      | Colormap                              | 435        |
| 15.8      | Data                                  | 437        |
| 15.9      | Field3                                | 438        |
| 15.10     | IvData                                | 439        |
| 15.11     | LabelField3                           | 439        |
| 15.12     | LandmarkSet                           | 440        |
| 15.13     | LargeDiskData                         | 442        |
| 15.14     | Lattice3                              | 442        |
| 15.15     | Light                                 | 444        |
| 15.16     | LineSet                               | 446        |
| 15.17     | Movie                                 | 448        |
| 15.18     | MultiChannelField3                    | 450        |
| 15.19     | Object                                | 451        |
| 15.20     | ScriptObject                          | 453        |
| 15.21     | SpatialData                           | 457        |
| 15.22     | SpreadSheet                           | 459        |

|           |                                            |            |
|-----------|--------------------------------------------|------------|
| 15.23     | Surface                                    | 459        |
| 15.24     | TetraGrid                                  | 461        |
| 15.25     | Time                                       | 461        |
| 15.26     | VertexSet                                  | 462        |
| <b>16</b> | <b>Molecular Pack</b>                      | <b>465</b> |
| 16.1      | MolSurface                                 | 465        |
| 16.2      | MolTrajectory                              | 466        |
| 16.3      | Molecule                                   | 467        |
| <b>V</b>  | <b>Alphabetic Index of Editors</b>         | <b>473</b> |
| <b>17</b> | <b>Amira</b>                               | <b>475</b> |
| 17.1      | Advanced Colormap Editor                   | 475        |
| 17.1.1    | Description of the user interface elements | 476        |
| 17.1.2    | How to modify a colormap                   | 480        |
| 17.2      | CameraPath Editor                          | 481        |
| 17.3      | Color Dialog                               | 482        |
| 17.4      | Colormap Editor                            | 485        |
| 17.4.1    | Overview                                   | 486        |
| 17.4.2    | Menu Bar                                   | 486        |
| 17.4.3    | Global Settings                            | 486        |
| 17.4.4    | Histogram                                  | 487        |
| 17.4.5    | Gradient Editor                            | 487        |
| 17.4.6    | Color Chooser                              | 488        |
| 17.4.7    | Opacity Chooser                            | 488        |
| 17.4.8    | Control Buttons                            | 488        |
| 17.5      | Curve Editor                               | 489        |
| 17.6      | Digital Image Filters                      | 489        |
| 17.6.1    | Minimum Filter                             | 491        |
| 17.6.2    | Maximum Filter                             | 491        |

|         |                                                                             |     |
|---------|-----------------------------------------------------------------------------|-----|
| 17.6.3  | Unsharp Masking . . . . .                                                   | 491 |
| 17.6.4  | Laplacian Zero-Crossing Filter . . . . .                                    | 491 |
| 17.6.5  | Median Filter . . . . .                                                     | 491 |
| 17.6.6  | Gauss Filter . . . . .                                                      | 492 |
| 17.6.7  | Sobel Filter . . . . .                                                      | 492 |
| 17.6.8  | Histogram Filter . . . . .                                                  | 492 |
| 17.6.9  | Edge-Preserving Smoothing . . . . .                                         | 492 |
| 17.6.10 | Lanczos Filter . . . . .                                                    | 493 |
| 17.6.11 | Sigmoid Filter . . . . .                                                    | 493 |
| 17.6.12 | Brightness and Contrast Filter . . . . .                                    | 493 |
| 17.6.13 | Moments Filter . . . . .                                                    | 493 |
| 17.6.14 | Gamma Correction . . . . .                                                  | 494 |
| 17.7    | ImageCrop Editor . . . . .                                                  | 494 |
| 17.7.1  | Cropping an image by dragging and moving the box . . . . .                  | 495 |
| 17.7.2  | Cropping an image by setting values explicitly in the text fields . . . . . | 495 |
| 17.7.3  | Adding new slices . . . . .                                                 | 495 |
| 17.7.4  | Changing the size of the bounding box . . . . .                             | 496 |
| 17.7.5  | Flipping slices in one dimension . . . . .                                  | 496 |
| 17.7.6  | Exchanging two dimensions . . . . .                                         | 496 |
| 17.8    | LandmarkEditor . . . . .                                                    | 496 |
| 17.9    | LineSet Editor . . . . .                                                    | 497 |
| 17.10   | Parameter Editor . . . . .                                                  | 498 |
| 17.11   | Plot Tool . . . . .                                                         | 498 |
| 17.11.1 | Plot basics . . . . .                                                       | 499 |
| 17.11.2 | Editing parameters . . . . .                                                | 500 |
| 17.11.3 | Working with plot objects . . . . .                                         | 504 |
| 17.11.4 | Printing . . . . .                                                          | 504 |
| 17.11.5 | Saving data . . . . .                                                       | 504 |
| 17.11.6 | Saving the plot state . . . . .                                             | 505 |
| 17.12   | Segmentation Editor . . . . .                                               | 507 |



|                                                       |            |
|-------------------------------------------------------|------------|
| 17.12.1 Overview of the segmentation editor . . . . . | 508        |
| 17.12.2 Manipulating the material list . . . . .      | 510        |
| 17.12.3 Working in 4-viewer mode . . . . .            | 512        |
| 17.12.4 Selection tools . . . . .                     | 513        |
| 17.12.5 Segmentation tools . . . . .                  | 514        |
| 17.12.6 Selection Filters . . . . .                   | 517        |
| 17.12.7 Label Filters . . . . .                       | 520        |
| 17.12.8 Key bindings . . . . .                        | 521        |
| 17.13 Simplification Editor . . . . .                 | 523        |
| 17.14 Surface Editor . . . . .                        | 524        |
| 17.14.1 Menu Entries . . . . .                        | 527        |
| 17.14.2 Selectors . . . . .                           | 528        |
| 17.14.3 Tools . . . . .                               | 530        |
| 17.15 Transform Editor . . . . .                      | 532        |
| <b>18 Molecular Pack</b>                              | <b>535</b> |
| 18.1 Molecule Attribute Editor . . . . .              | 535        |
| 18.2 Molecule Editor . . . . .                        | 542        |
| <b>19 Mesh Pack</b>                                   | <b>549</b> |
| 19.1 Grid Editor . . . . .                            | 549        |

## **Part I**

# **Alphabetic Index of Modules**



# Chapter 1

## Amira

### 1.1 Access LargeDiskData

This kind of module is attached to a *LargeDiskData* object. provides an interface to select a subblock of the large volumen which is stored on disk only.

You may select the volume either by typing in numbers to the ports or by using the blue dragger in the viewer.

Additionally, it is possible to request a lower resolution of the data. Dependent on the file format other resolutions are stored on disk and might be accessed rather fast or they are calculated on the fly which might take some time.

After selecting a volume you might press the *Apply* button. An *amira* field object is created and filled with the data from disk. You can now use all standard *amira* visualization techniques on this subblock.

If you check *auto-refresh*, you might change the selected volumen and an immediate reload is started. This might be useful to easily explore a large volume.

### Connections

**Data** [required]

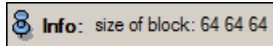
Connection to the *LargeDiskData* object.

**ROI** [optional]

If a *SelectRoi* is connected it will define the subvolume to be loaded.

## Ports

### Info



Indicates the size of the block to be loaded.

### BoxMin



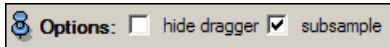
The position of the lower corner of the block. Printed as a lattice index.

### BoxSize



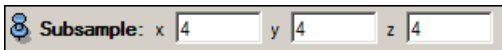
The size of the block to be loaded. The size is in lattice indices in the base resolution. The numbers will not change if you select subsampling but the real size of the block to be loaded will change. This is indicated in the *Info* port.

### Options



Hides the dragger and/or enables subsampling.

### Subsample



Type in the number of voxels to be averaged in each dimension.

## 1.2 AffineRegistration

This module computes an affine transformation for registration of two image data sets, using an iterative optimization algorithm. A hierarchical strategy is applied, starting at a coarse resampling of the dataset, and proceeding to finer resolutions later on. Different similarity measures like Euclidean distance, mutual information, and correlation can be chosen.

To use this module, you must connect it to two scalar fields. To follow the progress visually, a module like BoundingBox or Isosurface should be connected to each of them, but this may slow down the registration process. If the *Register* button of the *Action* port is pressed, the module starts by successively optimizing the transformation of the first input.

The optimization can be interrupted at any time. Interruption might take some seconds.

Note: In versions of *amira* prior to 4.0, this module was named *Registration*.

## Connections

### Model [required]

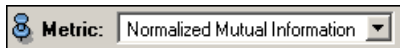
The model dataset to be transformed.

### Reference [required]

The reference dataset to which the model is registered.

## Ports

### Metric

A port control box for the Metric parameter. It features a blue icon with a question mark on the left, followed by the text "Metric:" and a dropdown menu currently showing "Normalized Mutual Information".

This port selects the similarity measure to be applied. *Euclidean* means the Euclidean distance, i.e. the mean squared difference between the gray values of model and reference. *Correlation* measures the correlation of the registered images. The *Mutual information* metrics, especially the normalized one, are recommended when medical images from different modalities, e.g., CT and MRT, are to be registered.

### Transformation

A port control box for the Transformation parameter. It features a blue icon with a question mark on the left, followed by the text "Transformation:". To the right are four checkboxes: "Rigid" (checked), "IsoScale" (unchecked), "AnisoScale" (unchecked), and "Shear" (unchecked).

At this port you can select the number of transformation parameters to be optimized. The numbers are 6 for *Rigid* (3 translations and 3 rotations), 7, 9, and 12 for *IsoScale*, *AnisoScale*, and *Shear*, respectively.

If you select for example *Rigid* and *AnisoScale*, on each resolution level the 6 parameters of a rigid transformation will be optimized first, followed by an optimization of all 9 parameters of an anisotransformation. In this way as much as possible is done using a rigid transformation.

If only *AnisoScale* is selected, all 9 parameters will be optimized at once. A larger contribution of the scaling parameters may be expected for this selection.

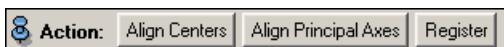
### Extended options

A port control box for the Extended options parameter. It features a blue icon with a question mark on the left, followed by the text "Extended options:" and a checked checkbox.

If this toggle is unset, only the most important ports are visible. The other ports are set to default values which should work well in most cases.

If the toggle is selected, the additional ports *Optimizer*, *Optimizer step*, *QuasiNewton Optimizer on finest N levels*, *CoarsestResampling*, *Histogram range reference*, *Histogram range model*, and *Options* become visible (see below).

### Action

A port control box for the Action parameter. It features a blue icon with a question mark on the left, followed by the text "Action:". To the right are three buttons: "Align Centers", "Align Principal Axes", and "Register".

If the *Align Centers* button is pressed, the centers of gravity of both datasets are computed, taking the image intensity as a mass density. The model dataset is translated in order to align both centers of gravity.

If the *Align Principal Axes* button is pressed, the centers of gravity and moments of inertia of both datasets are computed, again taking the image intensity as a mass density. The principal moments of inertia and corresponding principal axes are computed. The best of 24 possible alignments of the principal axes is determined according to the similarity measure as selected at the *Metric* port.

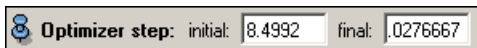
Pressing the *Register* button starts the actual registration process.

### Optimizer



At this port you can choose between different optimization strategies. The *ExtensiveDirection* or *BestNeighbor* optimizers are well suited for coarse resolution levels, the *QuasiNewton* or *Line-Search* optimizers for the finer resolution levels. The default strategy uses the *ExtensiveDirection* optimizer on the coarse levels and the *QuasiNewton* optimizer on the finest levels.

### Optimizer step



This port sets the initial and the final value for the stepwidth to be applied in the optimizations. These stepwidths refer to translations. For rotations, scalings, and shearings appropriate values are chosen accordingly.

The default value for the initial stepwidth is 1/5 of the size of the bounding box. If both datasets are already reasonably aligned, you may choose a smaller initial stepwidth.

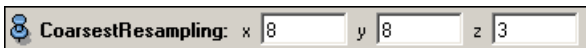
The default value for the final stepwidth is 1/6 of the voxel size.

### QuasiNewton Optimizer on finest N levels



At this port you can select the number of resolution levels (between 0 and 2) where the *QuasiNewton* optimizer is applied. On the coarser levels the optimizer as selected at port *Optimizer* is applied. If the number of levels is less than or equal to the number selected at this port, the optimizer as selected at port *Optimizer* is applied at least at the coarsest level.

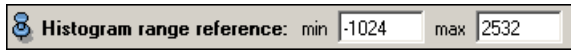
### CoarsestResampling



At this port you can define the resampling rate for the coarsest resolution level where registration starts. The resampling rate refers to the *reference* dataset. If the voxels of the reference dataset are *anisotropic*, i.e. have a different size in x-, y-, and z-direction, the default resampling rates are chosen in order to achieve isotropic voxels on the coarsest level. If the voxel sizes of *model* and

reference differ, the resampling rates for the model are chosen in order to achieve similar voxel sizes as for the reference on the same level.

### Histogram range reference

A control panel for the histogram range reference. It features a blue icon on the left, followed by the text "Histogram range reference:". To the right are two input fields: "min" with the value "-1024" and "max" with the value "2532".

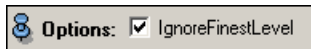
This port is only active if one of the *Mutual information* metrics has been selected. Here you can define the range of gray values for the histogram of the *reference* dataset. The essential information of the dataset should be within this range. It is a good idea to determine the range via a visualization of the dataset, e.g., using an *OrthoSlice* module.

### Histogram range model

A control panel for the histogram range model. It features a blue icon on the left, followed by the text "Histogram range model:". To the right are two input fields: "min" with the value "0" and "max" with the value "112".

The same as the previous port, now for the *model* dataset.

### Options

A control panel for options. It features a blue icon on the left, followed by the text "Options:". To the right is a checked checkbox followed by the text "IgnoreFinestLevel".

If toggle *IgnoreFinestLevel* is selected, registration will be performed on all but the finest (i.e. the original) resolution. In many cases a sufficient accuracy can be achieved in this way. Registration on the finest level may slightly improve the accuracy, but the computation time will typically increase by an order of magnitude.

## 1.3 AlignPrincipalAxes

This module computes a rigid transformation for a triangulated surface, such that its principal axes are aligned to a reference coordinate system. This can either be the standard basis in 3D or the principal axes of a reference triangulated surface.

### Connections

**Model** [required]

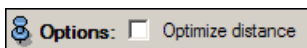
Model surface, that will be transformed.

**Reference** [optional]

Reference surface, whose principal axes will serve as a reference coordinate system.

### Ports

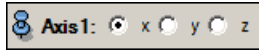
#### Options

A control panel for options. It features a blue icon on the left, followed by the text "Options:". To the right is an unchecked checkbox followed by the text "Optimize distance".



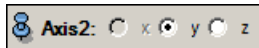
This port is only visible, when a reference surface is connected. For each possible solution (see explanation below at *Port Action*), the root mean square distance is computed, when the option 'Optimize distance' is chosen. The final transformation will be the one yielding the minimum value.

### Axis1



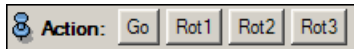
If no reference surface is connected, the principal axes with the largest moment of inertia will be aligned to the axis specified at this port.

### Axis2



If no reference surface is connected, the principal axes with the second largest moment of inertia will be aligned to the axis specified at this port.

### Action



The 'Go' button starts the alignment. Since the principal axes are only computed up to their orientation the model surface can subsequently be rotated 180 degree around each of its principal axes by pressing one of the 'Rot' buttons. The number corresponds the first, second or third largest moment respectively.

## Commands

### getRefSystem

Prints the center of gravity, the axes of inertia, and the moments of inertia in the console.

## 1.4 AlignSlices

This module allows you to interactively align 2D slices of a 3D image stack. Alignment is performed in a separate graphics window that is activated by pressing the *Edit* button of the module's *Action* port. To close the slice aligner, press the *Close* button of the module's *Action* port.

In addition to this documentation, there is a separate *tutorial about slice alignment* contained in the amira User's Guide.

The align window displays by default two consecutive slices using different draw styles that can be selected from the *View* menu. The two displayed slices can be chosen using the *slice slider* on the tool bar. Only one of these two slices is editable at a given time. The slice that can be edited is selectable and it can be either the lower or the upper one. By dragging the editable slice using the left mouse button, the slice will be translated. The slice can be rotated around its center by dragging it while

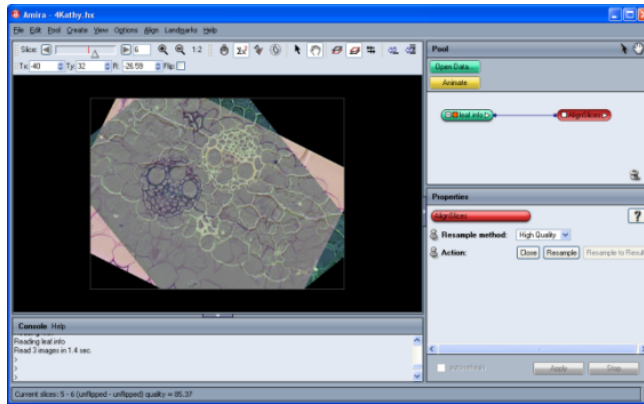


Figure 1.1: User interface of the align tool.

keeping the middle mouse button pressed. At any time the numbers of the current slice pair as well as the quality of the current alignment is displayed in the status bar at the bottom of the align window.

The documentation of the align tool is organized into the following sections (click on the links to open a section):

- *Tool bar* - describes the buttons of the tool bar.
- *Menu bar* - describes all entries of the menu bar.
- *Image viewer* - describes how to align images interactively.
- *Key bindings* - provides a list of all hot keys.

### 1.4.1 Tool Bar

- **Slice slider:** 

Allows the user to change the slices that are displayed and can be edited. The displayed number is the number of the editable slice. If this number is  $n$  and the editable slice is the upper one, slices  $n$  and  $n-1$  are shown. If the editable slice is the lower one, the displayed slices are slices  $n$  and  $n+1$ .
- **Zoom in button:** 

Increases the size of the image.
- **Zoom out button:** 

Decreases the size of the image.
- **Zoom label:**

Shows the current zoom factor. E.g., a zoom of 2:1 means that 2 pixels on the screen correspond

to one pixel of the original data set (magnification), while 1:4 would mean that four pixels of the data set correspond to one pixel on the screen.

- **Gravity centers button:**  Sets the gravity centers alignment as the current alignment algorithm.
- **Least squares button:**  Sets the least squares alignment as the current alignment algorithm.
- **Landmarks button:**  Sets the landmarks alignment as the current alignment algorithm. When this is the active align mode, the landmarks will be displayed for the current slices. If one of the other two algorithms is selected, the landmarks are not shown. The selected algorithm is used when the *Align current slices* or *Align all slices* button is pressed or the corresponding menu items are chosen.
- **Edge detection button:**  Sets the edge detection alignment as the current alignment algorithm.
- **Edit landmarks mode:**  Sets the active edit mode to be editing landmarks. When this is the active edit mode, the landmarks that have been already defined can be moved/removed and new landmarks can be defined (see *Image Viewer* section). This mode can be activated only when *Landmarks button* is ON (i.e. the active align mode is landmarks alignment).
- **Edit slice mode:**  Sets the active edit mode to be editing slices. This means that the editable slice can be translated/rotated.
- **Lower slice:**  This button forces the lower slice of the two displayed slices to become editable.
- **Upper slice:**  This button forces the upper slice of the two displayed slices to be editable (default). The editable slice can be translated and rotated and landmarks can be defined for it.
- **Mirror button:**  A mirroring transformation is applied to the editable slice.
- **Align current slices:**  The two currently displayed slices are automatically aligned using the selected algorithm.
- **Align all slices:**  All slices of the given 3D data are automatically aligned using the selected algorithm.

## 1.4.2 Menu Bar

### 1.4.2.1 View

- **Zoom In:** Increases the magnification factor of the image.
- **Zoom Out:** Decreases the magnification factor of the image.
- **Red/Green:** If this option is set, the two images are displayed in magenta and green, respectively. If both images match perfectly, a gray image is obtained. Color images are first converted to grayscale and then transformed to magenta and green.
- **Checkerboard:** If this option is set, one image is displayed in the white parts of a checkerboard pattern while the other image is displayed in the black parts. The size of the parts can be adjusted in the *Options* dialog (see below).
- **Average:** If this option is set the, two images are averaged or blended in the viewer window.
- **Invert:** If this option is set, the lower image is inverted. The inverted image is then blended with the upper image. If both images match perfectly, a constant gray image is obtained. This option is the default.
- **Transform parameters:** If enabled, an additional toolbar is added which displays the transformation and flip state of the currently editable slice.
- **Coronal view:** If enabled, displays a coronal view of the data and a toolbar for controlling the displayed slice number and its zoom factor.
- **Sagittal view:** If enabled, displays a sagittal view of the data and a toolbar for controlling the displayed slice number and its zoom factor.

If either or both of these views are enabled, an additional toolbar is displayed. Two text fields allow you to specify the min and max Z values of interest. Activate the *Auto update* toggle for the orthogonal views to be automatically updated each time you change a slice in the main align window. Press the *Update* button to request an explicit update.

### 1.4.2.2 Options

- **Undo:** This menu provides an undo feature that undoes the last operation. Successive invocation of *Undo* is possible, allowing several operations to be undone. Each operation made manually or automatically (translation, rotation, automatic alignment, landmarks editing) can be undone. The maximum number of operations that can be undone is 100.
- **Redo:** This menu provides a redo feature that redoes the last undone operation. Successive invocation of *Redo* is possible after several undo operations.
- **Reset all:** Set the translations on X and Y and the rotation to 0 for all slices.
- **Reset:** Set the translations on X and Y and the rotation to 0 for the current slice (the slice that is currently editable).
- **Transform all:** This is a toggle option. If the option is ON, the transformations made for the editable slice are also made for all slices above when the editable slice is the upper one. That means that the existing alignment of all other slices not involved in the current transformation

will be preserved.

- **Fix reference:** This option decides if a certain slice is used as a reference slice during the entire alignment procedure. The reference slice is marked by a red tag in the *Slice slider*. The other slices will be aligned according to this slice.
- **Read transformation:** Reads the *AlignTransform* and *AlignPoints* parameters from the input data and sets the transformation values accordingly. This is useful to return to a previous saved alignment. This can be seen as another kind of reset or undo.
- **Save transformation:** Save the actual translations, rotations, and landmarks as parameters in the input data.

### 1.4.2.3 Align

This menu offers several alignment algorithms that can be selected in order to obtain the best alignment. The performance of each algorithm is dependent on the data to be aligned. There are four alignment algorithms that can be chosen:

- **Gravity centers:** Align gravity centers and principal axes
- **Gray values:** Least squares algorithm based on gray values
- **Landmarks:** Align user-defined sets of landmarks
- **Edge detection:** Align the outer bounds of the objects

In addition, this menu allows you to specify which slices to align, as follows:

- **Align current pair:** Align just the current two slices
- **Align all slices:** Align all slices

Finally, the *Options* button of the *Align* menu opens a tabbed dialog window allowing you to define a variety of alignment-related options. Each tab will be handled separately below. The tabs are: *General*, *Output*, *View*, *Edge Detection*, and *Least Squares*.

### 1.4.2.4 Align/Options/General

- **Allow rotation in automatic alignment:** If this option is checked, rotations are considered during the alignment process. By default this option is checked. Translations are always considered.

### 1.4.2.5 Align/Options/Output

The *Output* dialog allows the user to give the output images of the image stack a different size than the input images. This is especially useful if the slices are rotated or moved during the alignment procedure and some areas of single slices are outside the borders of the image stack. Beside this, a

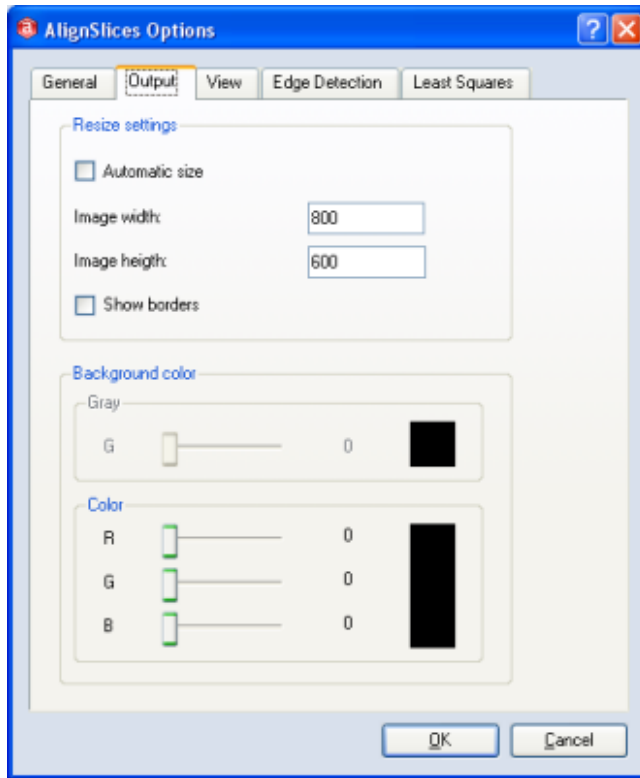


Figure 1.2: The *Output options* window.

background color for the new areas in the image can be determined. The following image shows the dialog box.

The following parameters can be set using this dialog:

- **Automatic size:** If this feature is activated, a minimal image size is calculated which contains all pixels of all slices. The dimensions and the position of the new bounding box are set such that all slices fit in this new bounding box.
- **Image width:** Sets the width of the output image.
- **Image height:** Sets the height of the output image.
- **Show borders:** Shows the new image borders of the output image in the *Slice Aligner*.
- **Background color:** The user can set the background color for the output image. The color can be set as a gray value or as an RGB color, depending on the type of the input image. The current background color is depicted in the box to the right.

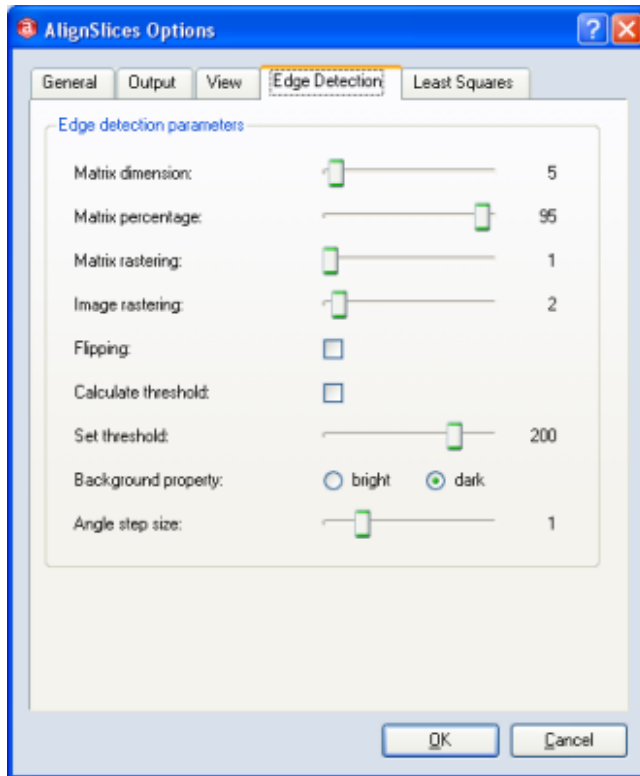


Figure 1.3: The Edge detection options window.

#### 1.4.2.6 Align/Options/View

- **Size of checkerboard:** Adjusts the size of the pattern if view mode is set to *checkerboard*.

#### 1.4.2.7 Align/Options/Edge Detection

The parameters for the *Edge detection* algorithm can be set using this dialog:

- **Matrix dimension:** This is the dimension of the matrix which scans the image and decides whether a pixel belongs to the object or the background.
- **Matrix percentage:** This percentage defines the amount of pixels in the surrounding of a pixel which have to belong to the object to indicate that this pixel belongs likewise to the object. The considered surrounding is defined by the matrix dimension.
- **Matrix rastering:** This indicates if the surrounding should be rasterized. If so, the alignment is done in a shorter time.

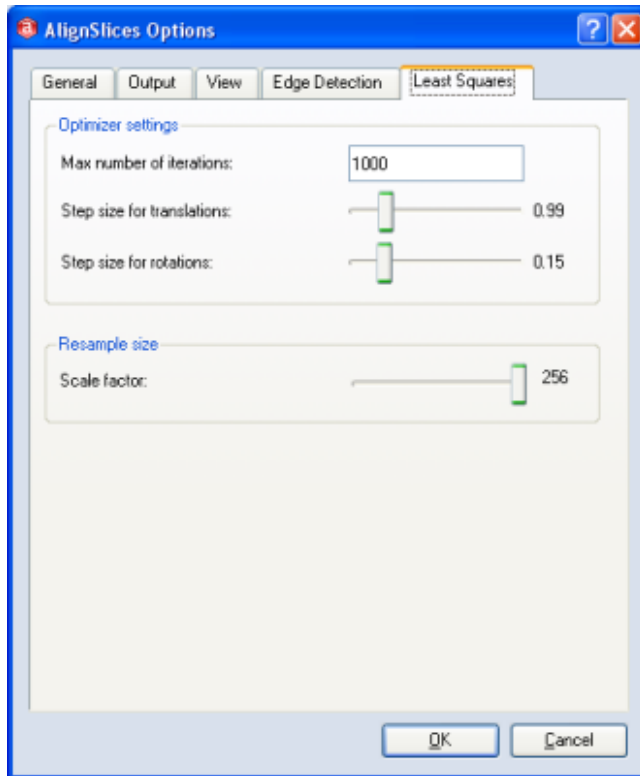


**Figure 1.4:** The *Least squares options* window.

- **Image rastering** This decides if the image should be downsampled during the alignment procedure. This speeds up the alignment.
- **Flipping:** This option indicates if a possible flipping of the images should be considered.
- **Calculate threshold:** This activates an automatic calculation of the threshold.
- **Set threshold:** This allows the threshold to be set manually.
- **Background property:** This must be set to describe the background of the image. According to this setting, the threshold is interpreted as an upper or a lower boundary of gray values which separate the object from the background.
- **Angle stepsize:** This sets the size of the search algorithm's angular steps (in degrees). A higher value speeds up alignment at the cost of accuracy.



#### 1.4.2.8 Align/Options/Least Squares

The following parameters for the gray values algorithm can be set using this dialog:

- **Max number of iterations:** This is only intended to prevent an infinite loop.
- **Step size for translations:** The step size used to search for better positions in X and Y. A small step size, though in general may lead to more accurate results, also slows down alignment.
- **Step size for rotations:** The step size used for rotations.
- **Scale factor:** During the alignment procedure, the image can be downsampled for the first iteration steps so that a better result can be reached in a shorter time.

More information about the align methods and the appropriate settings for the methods can be found in the *alignment section*.

### 1.4.3 Alignment Methods

The four different alignment methods are based on different concepts and thus the results may differ significantly. However, not only the method decides on the result but the right settings for the method. This page shall help to use the best alignment method and to use the right settings. Therefore, the algorithms behind the methods are briefly described. This requires a little background knowledge about mathematics.

- **Gravity centers:** This method calculates a center of gravity of the gray values and the orientation by means of the covariances of the image.
- **Gray values:** This method compares the single gray values of two images. The more pixels of two compared slices have the same gray values, the better the alignment will be valued for this method. That means that this method tries to move one slice and calculates the difference of the gray values of both images. If the quality is getting higher after the movement, this method will keep on changing the positions of the slices to each other until a maximum quality according to the above mentioned feature will be reached.
- **Landmarks:** This method is the one that requires most user interaction. The user can define some points, so called landmarks, which will be aligned to each other during the process. The method is mostly one of the best because the user decides about the alignment. The problem of this method is that the user has to set all of these landmarks so that this method is very time-intensive.
- **Edge detection:** This method is an outline-based method. It works in two steps. In the first step the method tries to identify the object and clears the surrounding of any noise. If the object is separated from the background, the outlines of the objects in two successive slices will be compared and aligned with each other.

Two methods allow to define some settings. These settings often decide on the quality of an alignment procedure. Sometimes, unintentional artifacts appear which result from wrong settings.

- **Gray values:** This is the first method that allows the user to set some parameters. As described above, the alignment compares the gray values of two successive slices. The possible settings are described in the *Least-squares options window*. The scale factor defines if an image is re-scaled before the entire process. This means that the image will be resampled in a *scale factor*-times coarser resolution. This speeds up the first steps of the process to find a first maximum. If this factor is too large, the editable slice could *flee* out of the canvas. This is a normal behavior since only points which overlap each other are considered for the calculation of the alignment quality, i.e., if one image moves out of the canvas the quality will be calculated as one hundred percent.
- **Edge detection:** This method has several settings which determine quality and speed of the alignment. As mentioned above, the alignment is divided into two phases. In the first phase, the object must be separated from the background, and in the second phase the rotation of two objects is calculated according to their outlines. The separation of image and background is achieved by using a matrix which decides according to a surrounding if one pixel belongs to the object or the background. The size of the surrounding region is set by the first parameter. The higher this value is, the slower the alignment procedure will be. If the background noise is rather small, the matrix size can be set small in order to speed up the alignment procedure. If an image has a high resolution, the image rastering can be set to a value greater than one. This speeds up the alignment procedure considerably. If the object has dis-symmetrical outlines, the angle-step-size can be set on a higher value. This also speeds up the alignment procedure. If the outlines are symmetrical, it is strongly recommended to set the step size to a small value.

#### 1.4.3.1 Landmarks

The buttons of this menu are only active if *Edit landmarks* mode is selected. The buttons have the following meaning:

- **Add:** set the shape of the mouse cursor to a black landmark and clicking on the slice a new landmark is created.
- **Remove:** delete the selected landmark. This menu item is active only when a landmark is selected and the number of landmarks is greater than 3.

### 1.4.4 Image Viewer

The *Image Viewer* represents the main part of the align tool. It allows you to align the slices manually using the mouse and/or the keyboard. While two slices are displayed simultaneously, only one slice can be edited at a time.

A slice can be translated over the other by keeping the left mouse button pressed and dragging the slice. Also, the slice can be moved using the cursor keys (see *Key Bindings* section). The rotation can be obtained by dragging the slice while keeping the middle mouse button pressed. All rotations are made around the middle of the slice.

If the *Transform all* option is ON (*Options* menu), the actual alignment will be preserved for all pairs of two consecutive slices (except the slices actually edited).

The *Image Viewer* allows you to edit the landmarks. In order to edit the landmarks, the *Edit landmarks* mode must be set.

A landmark can be **selected** by clicking inside the triangle that represents the landmark. Once selected, a landmark can be moved, removed or deselected. The landmarks can be also selected consecutively, by clicking anywhere on the slice (not on a landmark). After a so-selected landmark has been moved, the next landmark in the list can be selected in the same way.

In order to **move** a selected landmark, just click on the new position.

A selected landmark can be **removed** by choosing the *Remove* item of the *Landmarks* menu or pressing the **Del** key. A landmark can be removed only if the number of landmarks defined for each slice is greater than 2. The corresponding landmark will be removed from each slice so that the number of landmarks is the same for each slice.

You can create a **new** landmark using the *Landmark/Add* menu or pressing the **Ins** key and clicking on the desired position. A new landmark will be created for each slice. The creation of a new landmark can be canceled by pressing **Esc**.

### 1.4.5 Key Bindings

- **Changing the slice number**
  - Space** - Go to next slice
  - Backspace** - Go to previous slice
  - Home** - Go to first slice
  - End** - Go to last slice
- **Translating the editable slice**
  - CursorUp** - Translation one pixel up
  - CursorDown** - Translation one pixel down
  - CursorLeft** - Translation one pixel left
  - CursorRight** - Translation one pixel right
  - Shift+CursorUp** - Translation five pixels up
  - Shift+CursorDown** - Translation five pixels down
  - Shift+CursorLeft** - Translation five pixels left
  - Shift+CursorRight** - Translation five pixels right
- **Rotating the editable slice**
  - Ctrl+CursorUp** - 0.10 degree counter clockwise rotation
  - Ctrl+CursorDown** - 0.10 degree clockwise rotation
  - Ctrl+Shift+CursorUp** - 1.0 degree counter clockwise rotation
  - Ctrl+Shift+CursorDown** - 1.0 degree clockwise rotation

- **Changing the editable slice**

- 1 - Set the editable slice to be the lower slice. While you hold the key down, only the lower slice will be visible.
- 2 - Set the editable slice to be the upper slice. While you hold the key down, only the upper slice will be visible.

- **Changing the alignment algorithm**

- C - Set the gravity centers alignment as current alignment algorithm.
- G - Set the least squares alignment as current alignment algorithm.
- L - Set the landmarks alignment as current alignment algorithm.
- E - Set the edge detection alignment as current alignment algorithm.

- **Editing landmarks**

**Insert** - Set the shape of the mouse cursor to a black landmark. Clicking on the slice a new landmark is to be created.

**Delete** - Delete the selected landmark. This takes effect only when the number of landmarks is greater than 3.

**Escape** - Pressing Escape if a landmark is selected causes this landmark to be deselected. Pressing Escape after the Insert key was pressed, causes the insert operation to be aborted.

- **Changing the edit mode**

**Escape** - Pressing Escape allows you to switch between editing slices mode and editing landmarks mode. However, this doesn't happen if the current edit mode is editing landmarks and a landmark is selected. In this case the landmark will be deselected. Pressing Escape once more causes editing slices to become the new edit mode.

## Connections

### **Data** [required]

3D field for which the slices will be aligned. Currently, uniform Cartesian grids of type *HxUniformScalarField3*, *HxUniformStackedScalarField3*, and *HxUniformColorField3* are supported.

### **Reference** [optional]

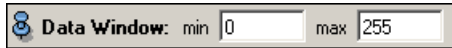
3D field from which the alignment informations can be read and transferred to the field that must be aligned (port *Data*). The reference field must have the same number of slices and the same dimensions as the data field.

### **Mask** [optional]

An additional *HxLabelLattice3* that can be connected in order to be used as mask during the alignment process.

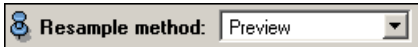
## Ports

### Data window



This port allows the user to restrict the range of visible data values. Values below the lower bound are mapped to black, while values above the upper bound are mapped to white. Only the values between the two bounds are used by the gray value-based alignment algorithm. This port is only active if a gray value image is used as input data.

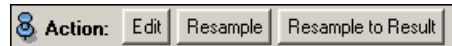
### Resample method



This port allows the user to decide which resample method will be used for the calculation of an output image. The calculation takes place when the *apply* or the *realign* button from the *Action* port is pressed. Two resample methods are possible:

- **Preview:** A very quick but not very precise method. It should only be used as solution preview.
- **High Quality:** A much slower but very accurate interpolation method.

### Action



Press the *Edit* button to open the align tool. Press the *Resample* button to create a new field according to the transformations made in the align tool. The new field will have by default the dimensions of the input image. The dimensions can be altered by the *resize dialog*. Press the *Resample to Result* button to overwrite an attached result. If a labelfield is attached to the existing result, the labelfield will also be transformed so that the former labels will also fit after the align procedure.

## Commands

performance

Computes the number of frames per second in the *Image Viewer*.

setSliceNumber <n>

Sets the current editable slice to be *n*.

getSliceNumber

Gets the current editable slice.

setEditableSlice {lower|upper}

Of the two currently displayed slices, sets the editable slice to be the lower/upper one.

getEditableSlice

Returns whether the editable slice is the upper or lower of the currently displayed slices.

**translate** <tx> <ty>

Translates the currently editable slice by *tx* on the X axis, by *ty* on the Y axis.

**rotate** <phi>

Rotates the currently editable slice by *phi*. Angle *phi* is considered to be in degrees and the rotation is made counterclockwise.

**setAlignMode** {0|1|2|3}

Changes the current alignment mode. The meaning of the parameter is 0 = principal axes alignment, 1 = least squares alignment, 2 = landmarks alignment, and 3 = edge detection alignment.

**getAlignMode**

Changes the current alignment mode. The meaning of the result is 0 = principal axes alignment, 1 = least squares alignment, 2 = landmarks alignment, and 3 = edge detection alignment.

**setViewMode** {0|1|2|3}

Changes the display mode. The view modes are encoded as follows: 0 = magenta/green view, 1 = checkerboard view, 2 = average view, 3 = invert view.

**align**

Aligns the current slices pair.

**alignAll**

Aligns all slices.

**setLandmark** <i> <k> <x> <y>

Sets the landmark with index *k* of slice *i* to be the point (*x*,*y*).

**getLandmark** <i> <k>

Returns the x and y coordinates of the landmark with index *k* of slice *i*.

**edit**

Shows the tool window (the same as the *Edit* button of the the *Action* port).

**showCoronalView** {0|1}

Toggles the display of a coronal cross-section through the aligned stack.

**showSagittalView** {0|1}

Toggles the display of a sagittal cross-section through the aligned stack.

**setCoronalViewSliceNumber** <n>

Sets the slice number of the coronal cross-section.

**getCoronalViewSliceNumber** <n>

Returns the slice number of the coronal cross-section.

**setSagittalViewSliceNumber** <n>

Sets the slice number of the sagittal cross-section.

`getSagittalViewSliceNumber <n>`

Returns the slice number of the sagittal cross-section.

`orthoViewZoomIn`

Zooms in on the sagittal and/or coronal cross-sections.

`orthoViewZoomOut`

Zooms out on the sagittal and/or coronal cross-sections.

`setUseMaxIntProjection {0|1}`

Enables or disables the use of maximum intensity projections of  $n$  slices instead of the real image data for the reference image and the transformed image. This is particularly helpful when aligning very small (point-like) objects, e.g., cross sections of neurons. If enabled, the automatic alignment method will use these projections instead of the real images. The number  $n$  is separately adjustable for the editable slice and the reference slice, using the commands below. Default is false (0).

`getUseMaxIntProjection {0|1}`

Returns whether maximum intensity projections are used for visualization.

`setMaxIntProjectionRefThickness <value>`

Specifies the number of slices to be used for the maximum intensity projection visualization of the reference.

`getMaxIntProjectionRefThickness <value>`

Returns the number of slices to be used for the maximum intensity projection visualization of the reference.

`setMaxIntProjectionSliceThickness <value>`

Specifies the number of slices to be used for the maximum intensity projection visualization of the currently transformed slice.

`getMaxIntProjectionSliceThickness <value>`

Returns the number of slices to be used for the maximum intensity projection visualization of the currently transformed slice.

## 1.5 Animate

The Animate module can be connected to any other object to animate its visibility or to animate the value of one of the following ports: *PortFloatSlider*, *PortIntSlider*, *PortFloatTextN*, or *PortIntTextN*. The animate module provides a time port which can be used to set the value of one of these ports of the other object. The time value is not used directly to set the other port's value but is modified by a user-defined expression. For example, if the time is running from 0 to 100 and you want to animate a slice slider from 0 to 50, you have to use the expression  $t/2$ . More complicated expressions involving functions such as *sin* or *cos* can be used as well. For a description of supported functions please refer to the *Arithmetic* module.

## Connections

### Object [required]

Connection to the object containing the port to be animated.

### Time [optional]

Optional connection to a time object. If the port is connected to a time object the time of that object is used. In this way multiple modules with a time port can be synchronized.

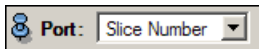
## Ports

### Time



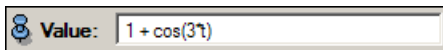
This slider allows you to set or animate the current time value. The time range and the increment used for animation can be adjusted in the configure dialog which can be popped up using the right mouse button.

### Port



Option menu listing all ports of the connected object which can be animated. The selected port actually will be animated. The special entry *visible* corresponds to the viewer mask rather than an actual port.

### Value



Use this text field to specify the expression used to compute the actual value of the port to be animated. The expression can include two different variables, namely  $t$  denoting the current time value, and  $u$  denoting the previous value of the port.

For the *visible* entry, the expression is interpreted as a boolean expression, where 0 means invisible, while a number different from 0 means visible. You can use expressions like  $t > 20$  or  $(t > 5) \&\& (t < 15) \ || \ (t > 100)$ .

### Value2



If the port to be animated has multiple text fields one expression port will be shown for each text field. If you don't want to change the value of a particular text field, use the expression  $u$ , which is the previous value of the port.



## Commands

### setMinMaxPortRange

Sets the minimum and maximum value of the time port to the minimum and maximum value of the chosen port.

## 1.6 Annotation

This module displays a 2D text in the 3D viewer. Position and color of the text can be specified by the user. This is useful for annotating snapshots. In order to create the module choose *Annotation* from the main window's *Create* menu.

For color legends refer to the module *Display Colormap*.

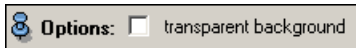
## Ports

### Color



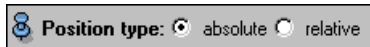
Text color. The color can be changed using the color editor, which is popped up when the color button is clicked.

### Options



If this toggle is off the annotation text is drawn inside a filled rectangle. The background color of this rectangle can be set via the Tcl command `setBackgroundcolor`.

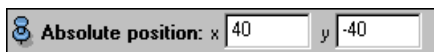
### Position type



Radio box defining whether the text position should be specified in *absolute* coordinates (screen pixels) or in *relative* coordinates ranging from (0,0) in the lower left corner to (1,1) in the upper right corner.

In both cases when the  $x$  coordinate is negative the right side of the text is placed relative to the right side of the viewer. Likewise, if the  $y$  coordinate is negative the top side of the text is placed relative to the top side of the viewer.

### Absolute position



Text position in screen pixels.

### Relative position

A control panel for relative position. It features a blue icon of a person with a speech bubble on the left. To its right is the text 'Relative position:'. Further right are two input fields: the first is labeled 'x' and contains the value '0.05'; the second is labeled 'y' and contains the value '-0.05'.

Text position in relative coordinates.

### Text

A control panel for text. It features a blue icon of a person with a speech bubble on the left. To its right is the text 'Text:'. Further right is a single-line text input field containing the value 'www.amiravis.com'.

The text to be displayed.

## Commands

`getFontSize`

Returns the current font size.

`setFontSize <points>`

Changes the font size. The default font size is 14 points.

`setBackgroundColor <color>`

Sets the color of the background rectangle which is drawn when the *transparent background* toggle is off.

## 1.7 AnonymizeImageStack

In the *Parameters* section of an image stack created from *DICOM* data the patient's name may occur at several locations:

- in the parameter bundle `DICOM-¿All, Parameter 'G0010-0010'`
- in the parameter bundles `DICOM-¿Slice...`, Parameters `'Filename'` and `'Name'`

Module *AnonymizeImageStack* creates a copy of the image stack and removes all occurrences of the patient's name from its parameter section.

Press the *Apply* button to initiate creation of the anonymized image stack.

## Connections

### Data

This port should be connected to the image stack to be anonymized.

## 1.8 ApplyBSplineTransform

Module for recovering and visualizing results of *elastic registration*.

## Connections

### Data [required]

The model image which was deformed during the registration. Alternatively, another data set can be connected. This will then be transformed accordingly.

### Control points [required]

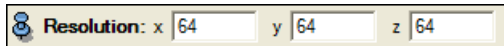
Control grid vectors storing the transformation computed as a result of an elastic registration.

### Reference [optional]

Optionally, the original reference image can be provided for determining the resolution of the output.

## Ports

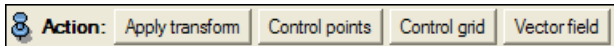
### Resolution



The image shows a port interface for 'Resolution'. It has a blue icon on the left. To its right is the text 'Resolution:'. Following this are three input fields: 'x' with the value '64', 'y' with the value '64', and 'z' with the value '64'.

The resolution of the output. Output can be the reformatted model image as well as the vector field giving the movement of each model voxel.

### Action



The image shows a port interface for 'Action'. It has a blue icon on the left. To its right is the text 'Action:'. Following this are four buttons: 'Apply transform', 'Control points', 'Control grid', and 'Vector field'.

*Apply transform* creates the reformatted and deformed model image. This might take some seconds because the reformatting is done using Lanczos interpolation. *Control points* creates the control points which induce the deformation as a landmark data set. *Control grid* shows the deformed control grid as a curvilinear data set, which allows for a combined visualization of the control grid using *LandmarkView* for the control points and *GridView* for the deformed grid data set.

*Vector field* creates a vector field storing the movement of each voxel. It allows for visualizing the deformation of space nicely if *GridView* or *Vectors* are chosen as display. To this end, it is recommended to create a vector field of reduced resolution using port *Resolution*.

## 1.9 ApplyTransform

Every data object in *amira* can carry a transformation as described in *Transformations*. But only the visualization of a field is changed not the representation in memory. That means if you scale a field with the *Transform Editor* dimensions and voxelsize will be retained though the size seems to be different on the screen. If you rotate a field the directions of the sampling planes will be still parallel to the local axis but no longer parallel to the global axis. The module *ApplyTransform* tries to create a new field in a way that it is displayed like the source field attached to the *Data* connection but without transformation.

You can e.g., align one field to another and apply the transformation. Afterwards you can directly work with the two fields without having to take into account any transformation.

There are two different ways to use the module:

- You can sample onto a given reference lattice.
- You can give a plane, e.g., an *ObliqueSlice*. The result is sampled in planes parallel to the given plane.

## Applying a transformation

Proceed as follows: Transform the field with the *Transform Editor*, the *Registration* module or any other way that creates a transformation. If you're satisfied with the transformation, attach an *Apply-Transform* module to the field. Now you have to choose some options.

The *Interpolation* method:

- *Nearest Neighbor* chooses for every new voxel the value of the voxel in the source field nearest to it.
- *Standard* linearly interpolates between the surrounding voxels.
- *Lanczos* is the slowest but most accurate method that tries to approximate a low pass filter that is in accordance with the sampling theorem. If you have time and want to get the best result, use this one.

The *Mode*:

- *cropped*: The new field has the same dimensions and size as the source field. It is adjusted to contain as much as possible of the source field.
- *extended*: The new field's size is adjusted to contain all of the original field. To do this, the voxelsize or the dimensions have to be changed. If you select this mode the *Preserve* port gets visible.

The *Preserve* port:

- *Voxel Size*: Adjust the result field's dimensions in a way that it has the same voxelsize as the source field. **Warning:** This may lead to huge dimensions.
- *Dimensions*: The result field will have the same dimensions as the source field.

After choosing the options press *Apply* and proceed.

## Sampling onto a reference lattice

Connect the source field to *Data*. Connect the reference field to *Reference*. Choose an *Interpolation* method. Press *Apply* and here we go.

## Sampling parallel to a given plane

Connect the source field to *Data*. Connect a plane (e.g., *ObliqueSlice*) to *Reference*. Choose an *Interpolation* method. Choose whether the result has the same dimensions (*cropped*) or the result is *extended* to contain all of the source field. Press *Apply* and the result is created. The result will get a transformation attached to in a way that the sampling planes are displayed parallel to the reference plane. Use the *Transform Editor* if you want to get rid of the transformation.

### Connections

#### Data [required]

The source field. The module operates on any regular 3D field with uniform coordinates and an arbitrary number of data components per node.

#### Reference [optional]

A regular field with uniform coordinates providing a reference lattice or a plane defining a sampling direction.

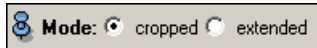
### Ports

#### Interpolation



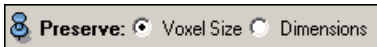
The interpolation method that will be used.

#### Mode



The result can be *cropped* to have the same properties as the source or it will be *extended* to contain all of the source field.

#### Preserve



If the result is extended should it keep *Voxel Size* or the *Dimensions* of the source.

## 1.10 ArbitraryCut

This module defines a plane which can be positioned and oriented arbitrarily inside the bounding box of a data object connected to port *Data*. Mostly, you will work with derived modules displaying some kind of geometry inside the plane defined by this base class. Examples of derived modules are *ObliqueSlice* or *PlanarLIC*.

However, this base class also serves as a background plane on which so-called *overlay modules* display their geometry. Examples of overlay modules are the *Isolines* module, the *Vectors* module, or the *Intersect* module. In fact, if you select an overlay module from a data object's popup menu an instance of *ArbitraryCut* called *EmptyPlane* will be created automatically.

You may also create an instance of *ArbitraryCut* by selecting *Clipping Plane* from the global *Create* menu. The module does not display useful geometric output by itself, but it can be used to clip the output of any other module in *amira*. In order to perform clipping you first have to connect port *Data* to the data object being investigated. Then you can simply click on the module's *clip button* located in the orange header block.

## Connections

### Data [required]

Connection to the data object from which the bounding box is taken.

## Ports

### Orientation



By clicking one of the three push buttons the plane's orientation is reset to the xy-, xz-, or yz-plane, respectively. The plane will be translated into the center of the bounding box.

### Options



This port provides three toggle buttons. If toggle *adjust view* is active then the camera of the 3D viewer will be reset whenever one of the orientation buttons is clicked.

If the *rotate* toggle is active then a virtual trackball is displayed. By picking and dragging the trackball you may change the orientation of the plane. Remember that the viewer must be in interaction mode in order to do so. The ESC-key inside the viewer window toggles between navigation mode and interaction mode. The trackball of the last active *ArbitraryCut* can also be turned on and off by pressing the TAB-key inside the viewer window.

The *immediate* toggle determines whether derived modules and downstream modules receive an update signal while the plane is being translated or rotated or not. This toggle might not always be visible.

The *fit to points* toggle lets you reset the plane by clicking on at least three different points in the scene. After enabling this toggle, you should switch to interaction mode by pressing the ESC-key inside the viewer. Then click three times at different points in the viewer window. The plane then will be automatically adjusted to match these points. Shift-clicking allows you to select more than three points. In this case plane will be computed that fits the selected points best.

## Translate



This port lets you translate the plane along its normal direction.

## Commands

`frame {0|1}`

This command lets you turn on or off the orange frame indicating the intersection of the plane with the bounding box of the data object connected to the data port.

`setFrameColor <color>`

Lets you change the color of the plane's frame.

`setFrameWidth <width>`

Lets you change the width of the plane's frame in pixels.

`getPlane`

Returns 9 numbers specifying the current plane settings. The numbers comprise the x, y, and z-coordinates of the plane's origin, the u-vector, and the v-vector.

`setPlane <origin uVec vVec>`

Adjust position and orientation of the plane. The command expects 9 number in the same format as returned by the `getPlane` command.

## 1.11 Arithmetic

The *Arithmetic* module performs calculations on up to three input data objects according to a user-defined arithmetic expression. The result is stored in a new data object called *Result*. The calculations are triggered by the *Apply* button. The arithmetic expression is evaluated either on the grid of the first data object (in case of regular, tetrahedral, or hexahedral grids or surfaces) or on a regular 3D uniform grid for which the number of points can be set by the *Resolution* port.

The module is able to process input fields with up to 6 different channels. Scalar input fields are referenced by the variables A, B, and C. The values of multi-component input fields are referenced for example by Ax, Ay, Az (if input A is a vector field) or Br, Bg, Bb, Ba (if input B is an RGBA color field).

In any case the expressions are evaluated per point, i.e., the result for a point (X,Y,Z) depends on input values at the same point only. If the resulting object is based on a regular grid, grid indices I, J, or K may also appear in the arithmetic expression. This means that for each grid point its associated I, J, or K index value will be substituted in the arithmetic expression on evaluation. This is useful in connection with comparison operators which produce a result of either zero or one. Computations may be confined to a specific sub-grid this way.

An expression consists of variables and mathematical and logical operators. The syntax is basically the same as for C expressions. The following variables are defined:

- **A**: The values of a scalar field at input A.
- **B**: The values of a scalar field at input B.
- **C**: The values of a scalar field at input C.
- **Ar**: The real part of a complex scalar field at input A.
- **Ai**: The imaginary part of a complex scalar field at input A.
- **Ax**: The x-component of a vector field at input A.
- **Ay**: The y-component of a vector field at input A.
- **Az**: The z-component of a vector field at input A.
- **Ar**: The red component of a color field at input A.
- **Ag**: The green component of a color field at input A.
- **Ab**: The blue component of a color field at input A.
- **Aa**: The alpha component of a color field at input A.
- **Arx**: Real part of the x-component of a complex vector field.
- **Aix**: Imaginary part of the x-component of a complex vector field.
- **Ary**: Real part of the y-component of a complex vector field.
- **Aiy**: Imaginary part of the y-component of a complex vector field.
- **Arz**: Real part of the z-component of a complex vector field.
- **Aiz**: Imaginary part of the z-component of a complex vector field.
- **Aii**: Index ii of a symmetric second order tensor.
- **Aij**: Index ij of a symmetric second order tensor.
- **Aik**: Index ik of a symmetric second order tensor.
- **Ajj**: Index jj of a symmetric second order tensor.
- **Ajk**: Index jk of a symmetric second order tensor.
- **Akk**: Index kk of a symmetric second order tensor.
- The same variables as above for fields at input B or C, but with A replaced by B or C.
- **I**: First index of a point (i,j,k) in a regular grid, or index of a point in an unstructured grid.
- **J**: Second index of a point (i,j,k) in a regular grid, undefined for unstructured grids.
- **K**: Third index of a point (i,j,k) in a regular grid, undefined for unstructured grids.
- **X**: The x-coordinate of the current point.
- **Y**: The y-coordinate of the current point.
- **Z**: The z-coordinate of the current point.
- **R**: The radius  $r = \sqrt{X^2 + Y^2 + Z^2}$ .



This is a list of available mathematical and logical operators:

- `+` `-` `/` `*` The basic mathematical operators.
- `!` Unary negation.
- `-` Unary minus.
- `%` The modulo operator.
- `>` `<` `<=` `>=` `!=` `==` The comparison operators *greater*, *less*, *less or equal*, *greater or equal*, *not equal*, and *equal*. If the comparison is true the result is 1, otherwise it is 0.
- `&&` `||` `^` The logical operators *and*, *or*, and *xor*. A non-zero operand is interpreted as true, while a zero operand is false. The result is either 1 (true) or 0 (false).
- `&` `|` The bitwise operations *and* and *or*. For bitwise *and*, the bits in the result are set to 1 if the corresponding bits in the two operands are both 1. For bitwise *or*, the bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.

There are also some built-in functions:

- `pow(x, a)` Power function. Note that there is no power operator (the `^` operator exists but means logical *xor*).
- `sin(x)` `cos(x)` `tan(x)` Trigonometric functions.
- `asin(x)` `acos(x)` `atan(x)` `atan2(x, y)` Inverse trigonometric functions.
- `sinh(x)` `cosh(x)` `tanh(x)` Hyperbolic trigonometric functions.
- `asinh(x)` `acosh(x)` `atanh(x)` Hyperbolic inverse trigonometric functions.
- `sqrt(x)` Square root.
- `floor(x)` `ceil(x)` Next largest or smallest integer number.
- `ln(x)` `log10(x)` `exp(x)` Logarithm and exponent.
- `rand()` Pseudo-random variable uniformly distributed between 0 and 1.
- `gauss()` Pseudo-random variable with Gaussian distribution (mean value 0, standard deviation 1).
- `erf(x)` `erfc(x)` Error function.
- `abs(x)` `fabs(x)` Absolute value.
- `min(x, y)` `max(x, y)` Minimum or maximum values.

For better understanding some examples of how to use the *Arithmetic* module follow:

Expression: A

The result is a copy of input object A. If A is defined on a tetrahedral or hexahedral grid and the result type is set to *regular* together with an appropriate resolution, the expression leads to a conversion from an unstructured grid to a structured regular grid. The same trick can also be used to resample a regular field with stacked coordinates onto a uniform grid.

Expression: A-B

The result is the difference between input objects A and B. This expression is sometimes useful in order to compare to different data sets.

Expression:  $255 * (A > 127)$

Simple thresholding: For every value of A the result is set to 255 if the value is greater than 127. Otherwise the result is 0.

Expression:  $A * (B > 0)$

Simple masking operation: If B is zero, the result is also set to zero. Otherwise, the result is set to A. For example, if A is a 3D image and B is a corresponding label field, the exterior parts of the object (where B is zero) are masked out by this expression.

## Connections

### InputA [required]

The input may either be a 3D data field with an arbitrary number of channels or a tetrahedral or hexahedral grid. The primitive data type of the result will be the same as this input, e.g., if input A contains bytes the result will also contain bytes.

The input may also be a surface field or a surface itself. For example, you may want to capture the values of a 3D data field at all points of a surface in a surface field. However, note that data on surfaces can only be used as input if the output is defined on the same surface. Trying to evaluate a surface field for points of a 3D grid or for points of some other surface is not possible, even if these points lie exactly on the source surface.

### InputB [optional]

Second input, can be any 3D data field or surface field.

### InputC [optional]

Third input, can be any 3D data field or surface field.

## Ports

### Result channels



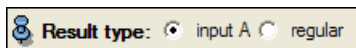
This port determines the number of channels of the result. By default, the result has the same number of channels as input A. Alternatively, you can specify that the result should have 2 channels (complex scalar field), 3 channels (vector field), 4 channels (RGBA color field), or 6 channels (either a complex vector or a symmetric tensor field of second order).

### Expr



This port specifies the mathematical expression used to compute the result. If the result has more than one channel, more expression ports will be shown.

## Result type



With this radio box the grid type of the result can be set either to either the same as input A or to a regular grid with uniform coordinates.

## Resolution



If port *Result Type* is set to *regular*, the resolution of the regular field to be generated is set to the values given here.

## MinBox



Port to set the minimum x-, y-, z-coordinates of a bounding box around the output data object. If one or more input data objects are connected, the bounding box of the first input data object is also taken as the output bounding box.

## MaxBox



Port to set maximum x-, y-, z-coordinates of the bounding box around the output data object.

## 1.12 Axis

The *Axis* module displays a coordinate frame. If the module is attached to an input data object, the coordinate frame is adapted to the bounding box of the input object. Otherwise, global axes centered at the origin of the world coordinate system are displayed. For convenience the *View* menu of the main window contains a toggle button labelled *GlobalAxis*. Activating this toggle automatically creates an *Axis* module in the Pool.

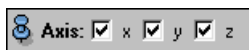
## Connections

**Data** [optional]

Data object the axes should be adapted to.

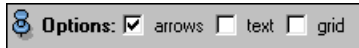
## Ports

### Axis



This port lets you select for which direction axes should be drawn.

## Options



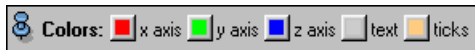
This port provides three toggles: If *arrows* is set arrows are drawn at the top of each axis. If *text* is set labeled ticks are drawn indicating the coordinate values. If *grid* is set a coordinate raster is drawn between every pair of visible axes.

## Thickness



Lets you adjust the thickness of the axes.

## Colors



Provides buttons for changing the colors of different parts of the geometry. By default, the x-, y-, and z-axes are drawn in red, green, and blue, respectively.

## Commands

`getFontName`

Returns the name of the text font. Default is *Helvetica*.

`setFontName <name>`

Lets you change the text font.

`getFontSize`

Returns the size of the text font. Default is 10 points.

`setFontSize <size>`

Lets you change the size of the text font.

`getBoundingBox`

Returns the bounding box of the volume enclosed by the axes.

`setBoundingBox <xmin> <xmax> <ymin> <ymax> <zmin> <zmax>`

Lets you change the bounding box of the axis module. Once the bounding box has been set automatically, the box will not be updated automatically anymore, unless the `setBoundingBox` command is issued again with no arguments.

## 1.13 BoundingBox

The *Bounding Box* module can be connected to any spatial data object in order to visualize its bounding box. The bounding box of a data object encloses its geometrical domain. More precisely, it is obtained by determining the minimum and maximum coordinates in x-, y-, and z-direction.

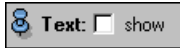
## Connections

**Data** [required]

Any spatial data object.

## Ports

**Text**



Shows or hides the coordinates of the lower left front and upper right back corner of the box.

## 1.14 CalculusMatlab

This module establishes a connection to MATLAB (The MathWorks, Inc.).

The module works like a compute module and allows the computation on **amira** data objects in the MATLAB workspace. **amira** data objects can be attached to the module and a MATLAB script file can be specified. The following actions are performed:

- all data objects attached are duplicated into MATLAB data objects
- the script is executed
- all variables defined are copied back into **amira**

There are limitations on the type of data objects used. Only lattice fields (scalar, complex, color and vector fields) can be copied to MATLAB. They are named  $A, A_1, \dots, A_n$  ( $n \geq 1$ ) in order of connection to the module. Only data matrices with 4 or fewer dimensions are copied back to **amira**.

- 1D - mapped to uniform scalar fields
- 2D - mapped to uniform scalar fields
- 3D - mapped to uniform scalar fields
- 4D - is interpreted as x-dim, y-dim, z-dim, X

where X may be one of the following:

- X=1 is mapped to a uniform scalar field
- X=2 is mapped to a uniform complex scalar field
- X=3 is mapped to a uniform vector field
- X=4 is mapped to a color field (value 0 to 255 per component)
- X=6 is mapped to a uniform complex vector field

In addition to lattice fields, time sliders can also be attached. They provide a way to define scalar variables that can be used to control the script (similar to parameters). In order to define a time slider,

use the *amiraCreate/Data/Time* menu entry and connect it as well to this module. If there are multiple time sliders, they will be named *t*, *t1*, *t2*, ... , *tn* where *n* is a number greater 0.

## Connections

### Data [required]

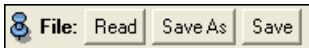
You can add any number of data objects to the module. Currently only uniform fields are allowed as valid data objects that can be transferred into MATLAB. Non-uniform coordinates such as curvilinear, stacked, tetra-grids, etc. are not supported.

### New port [optional]

After attaching a new data icon to the module, a new port is generated which can be used to attach additional data.

## Ports

### File



You should specify here a text file which contains a MATLAB script. Note that only scripts but not functions are allowed. An example script could look like this:

```
% A is the first connected data object (in double precision)
% X is newly created and will be sent back to amira
X = fftn(A); % use MATLAB function
magnitude = abs(X);
amplitude = unwrap(angle(X));
Z1 = randn(100,100,100,1); % Z1 is now a HxUniformScalarField3
Z2 = rand(100,100,100,2); % Z2 is now a HxUniformComplexScalarField3
Z3 = rand(100,100,100,3); % Z3 is now a HxUniformVectorField3
Z4 = rand(100,100,100,4).*255;% Z4 is now a HxUniformColorField3
Z5 = randn(100,100,100,5); % definition of Z5 will produce an error
Z6 = randn(100,100,100,6); % Z6 is now a HxUniformComplexVectorField3
clear A % clear all variables that you do not wish to export
```

Here is another example which implements an anisotropic diffusion:

```
% anisotropic diffusion in 2D (Perona Malik) adapted from Peter Kovess
erg = A;
for i=1:size(A,3),
    ima = double(reshape(A(:, :, i), size(A,1), size(A,2)));
    [r,c] = size(ima);
    diff = ima;
```

```

niter = 4;
kappa = 50;    % condition as function of gradient
lambda = .25; % speed of diffusion
if exist('t') == 1, % control by connected time slider
    niter = t; % number of iterations
end;
if exist('t1') ==1,% control by connected time slider
    kappa = t1;
end;
for j=1:niter,
    diff1 = zeros(r+2,c+2);
    diff1(2:r+1,2:c+1) = diff;
    deltaN = diff1(1:r,2:c+1) - diff;
    deltaS = diff1(3:r+2,2:c+1) - diff;
    deltaE = diff1(2:r+1,3:c+2) - diff;
    deltaW = diff1(2:r+1,1:c) - diff;
    cN = exp(-(deltaN/kappa).^2);
    cS = exp(-(deltaS/kappa).^2);
    cE = exp(-(deltaE/kappa).^2);
    cW = exp(-(deltaW/kappa).^2);
    diff = diff + lambda*(cN.*deltaN + cS.*deltaS + cE.*deltaE + cW.*deltaE);
end;
erg(:, :, i) = diff;
clear delta* c* diff* i j niter ima lambda kappa A r;

```

Here is an example of working with non-scalar data. In this case we have a colorfield attached as input A.

```

useimage=1; % which image from the stack
if exist('A') == 1, % is there an input?
    if size(size(A),2) == 4, % is the input 4D?
        if size(A,1) == 4, % the first dimension should be RGBA
            ima(:, :, 1) = reshape(A(1, :, :, useimage), size(A, 2), size(A, 3));
            ima(:, :, 2) = reshape(A(2, :, :, useimage), size(A, 2), size(A, 3));
            ima(:, :, 3) = reshape(A(3, :, :, useimage), size(A, 2), size(A, 3));
            mi = min(min(min(ima))); ma = max(max(max(ima)));
            ima = (ima-mi)./(ma-mi);
            imagesc(ima); % display the color image
        end;
    end;
end;
clear mi ma ima useimage A

```

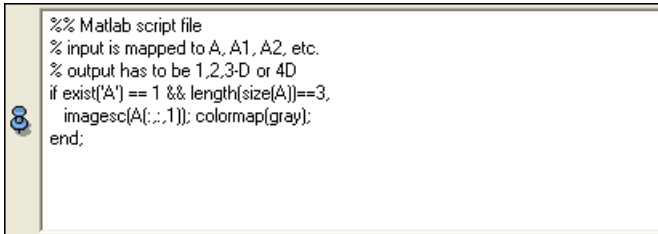
## Execute



Starts the execution of the MATLAB script. All generated messages will be printed in the **amira** console window.

You can either execute the contents of the text buffer below (*buffer*) or execute a selection of the text in the buffer (*selection*).

## MatlabBuffer



Enter or change a loaded script in this buffer.

## Options



This module starts a MATLAB session in the background. Its interface can be made visible, thus the user can interact with the running MATLAB process and inspect variables in its workspace. Note: Even if the MATLAB interface is visible, it may be minimized.

Because there is a running MATLAB process, you can use its window to inspect values in its workspace. This can be used as an effective tool for developing or debugging your scripts.

## 1.15 CastField

This is a computational module that allows you to change the primitive data type of a regular 3D scalar field or of an RGBA color field. A new scalar field will be created having the same dimensions and the same coordinates as the incoming one, but the data values will be shifted, scaled, and cast to a new data type.

As an additional feature, *CastField* is also able to convert a uniform scalar field of bytes into a *label field*, which is commonly used to store segmentation results in **amira**. For each value or label occurring in the incoming field a corresponding material will be created. Material colors may be defined via an optional colormap.

Press the *Apply* button to start the computation.



## Connections

### Data [required]

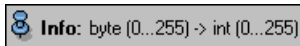
The scalar field or color field to be converted.

### Colormap [optional]

Optional colormap specifying material colors if a uniform scalar field of bytes is to be converted into a label field. Only visible, if *LabelField* has been selected for output.

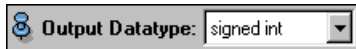
## Ports

### Info



Shows how the input data will be mapped during conversion. If no clipping occurs the input range covers all values between the minimum and maximum data value of the incoming scalar field. This range will be mapped as indicated. If clipping occurs, the minimum or maximum value of the output range or both will be equal to the smallest respectively biggest value which can be represented by the selected output data type. In this case the input range shows which values correspond to these limits. Data values below the lower limit or above the upper limit will be clamped.

### Output Datatype



Lets you select the primitive data type of the output field. The item *LabelField* is special. If this is selected the incoming scalar field is converted into a *label field*.

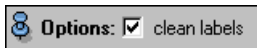
### Scaling



Defines a linear transformation which is applied before the data values are clamped and cast to the output datatype. The transformation is performed as follows:  $\text{output} = \text{SCALE} * (\text{input} + \text{OFFSET})$

If you want to convert data with a range  $\text{inmin} \dots \text{inmax}$  into a range  $\text{outmin} \dots \text{outmax}$ , SCALE and OFFSET are determined like this:  $\text{SCALE} = (\text{outmax} - \text{outmin}) / (\text{inmax} - \text{inmin})$  and  $\text{OFFSET} = \text{outmin} / \text{SCALE} - \text{inmin}$

### Options



This port is only shown if you want to convert the incoming scalar field into a label field. If option *clean labels* is set then the materials found in the input data set will be relabeled so that the first material is 0, the second is 1, and so on. If *clean labels* is not set then the resulting label field will contain exactly 256 materials and no check is performed if a material actually can be found.

## Colormap



Port to select a colormap.

## Color Channel



This option menu will only be shown if an RGBA color field is to be converted. It allows you to specify which channel of the color field should be regarded. In addition to the four RGBA channels also *Gray* and *Alpha\*Gray* can be selected. The gray channel is computed on-the-fly from the RGB values of the color field according to the NTSC formula, i.e.  $I = .3 * R + .59 * G + .11 * B$ .

## 1.16 ChannelWorks

This module allows to convert between scalar and vector fields. You can e.g., combine three scalar fields into one vector field, or extract one of the six channels of a complex valued vector field to obtain a scalar field.

Press the *Apply* button to start the computation.

## Connections

### Input1 [required]

Connects to a field, e.g., a uniform vector field, a uniform scalar field, 3D image volume, or field defined on a surface.

### Input2 [optional]

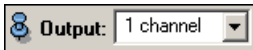
Optional second input, which accepts the same data types as the first input.

### Input3 [optional]

Optional third input, which accepts the same data types as the first input.

## Ports

### Output



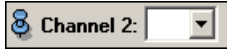
Select the desired output type: scalar field (1 channel), complex valued scalar field (2 channels), vector field (3 channels), color field (4 channels) or complex vector field (6 channels).

### Channel 1



Select which of the input channels is used as first output channel.

### Channel 2



Select which of the input channels is used as second output channel. This port is only present if the output has more than one channel.

### Channel 3



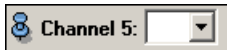
Select which of the input channels is used as third output channel. This port is only present if the output has more than two channels.

### Channel 4



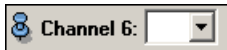
Select which of the input channels is used as fourth output channel. This port is only present if the output has more than three channels.

### Channel 5



Select which of the input channels is used as fifth output channel. This port is only present if the output has six channels.

### Channel 6



Select which of the input channels is used as sixth output channel. This port is only present if the output has six channels.

## 1.17 ClippingPlane

The clipping plane module is an instance of the *Arbitrary Cut* module (for a description see there). It can be created by selecting *Clipping Plane* from the main window's *Create* menu.

## 1.18 ClusterDiff

This module displays displacement vectors between corresponding points in two different data objects of type *Cluster*. The displacement vectors may be colored according to their lengths or according to any data variable defined in one of the input clusters. Like in the *ClusterView* module subsets of points may be selected by drawing a contour in the viewer window or by specifying an arithmetic filter expression.

## Connections

### Data

The first point cluster.

### Data2

The second point cluster. Initially, this port will not be connected to any data set. In order to use the module you have to establish a connection manually by activating the popup menu over the white square on the left side of the module's icon.

### Colormap

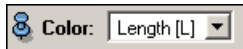
The colormap used for pseudo-coloring. The alpha channel of the colormap will be correctly taken into account unless the option *opaque* has been selected. However, note that many default colormaps are completely opaque. In this case, use the *colormap editor* to make them transparent.

## Ports

### Colormap



### Color



An option menu containing all data variables which can be used for pseudo-coloring. The data variables' symbols are displayed in brackets behind the names.

### Options



The following two options can be set:

*Opaque* influences the way how the displacement vectors are drawn. By default, anti-aliased semi-transparent lines are rendered. If the opaque option is set transparencies will be ignored. Opaque rendering is faster but gives less nice results. In addition, you can't suppress certain lines by making them more transparent.

*Filter* causes a text field to be shown, which can be used to define an arithmetic expression for selecting a subset of points.

### Filter



This port allows you define an arithmetic filter expression. The filter expression is evaluated for every pair of points. A displacement vector will only be drawn if for both points a non-zero result is obtained. The filter expression may contain any arithmetic and logical operator defined in the C programming language. All symbols listed in the color menu may be used in the filter expression. For example, in order select all vectors shorter than 0.2 and with an energy larger than -4 use  $(L < 0.2) \&\& (E > -4)$ .

### Action



The button labeled *Export A* creates a new point cluster containing all selected points of the first input cluster together with the corresponding data variables. Similarly, the button labeled *Export B* creates a new point cluster containing all selected points of the second input cluster.

After pressing the *Select* button you may draw a contour in the viewer window in order to deselect certain points. By default, all points inside the contour are deselected. If you keep the Ctrl-key pressed down when starting to draw the contour all points outside the contour are deselected. Using the Alt-key allows you to define straight-line segments.

The *Reset* button causes all points matching the filter expression to be shown again.

The *Undo* button undoes the effect of the last interactive contour selection operation.

## Commands

`setLineWidth <width>`

Specifies the width of the displacement vectors. By default, the vectors are one pixel wide.

`setLineSmooth {0|1}`

Enables or disables line smoothing. By default, smoothing is on unless the *opaque* option has been selected.

## 1.19 ClusterGrep

This module identifies common points in two different data objects of type *HxCluster* and copies them into new cluster objects. Using this module you may for example extract the same set of points from multiple clusters representing different time steps in a dynamic simulation.

### Connections

#### A

The first point cluster.

## B

The second point cluster. Initially, this port will not be connected to any data set. In order to use the module you have to establish a connection manually by activating the popup menu over the white square on the left side of the module's icon.

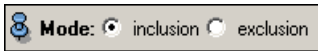
## Ports

### Points



This port displays the number of points in the two input clusters. If both input ports are connected also the number of common points is displayed.

### Mode



This port provides two radiobox buttons allowing you to specify which points are copied into the output clusters. If *inclusion* is selected only common points are copied. If *exclusion* is selected only points not present in the other cluster are copied.

### Action



Button *Export A* copies points from the first input cluster together with associated data values into a new cluster. Button *Export B* copies points from the second input cluster together with associated data values into a new cluster.

## 1.20 ClusterLabel

This module visualizes string labels associated with data objects of type *Cluster*. For information on labeling cluster objects, see the *Cluster* documentation.

## Connections

**Data** [required]

Cluster object.

## Ports

### Labels



This port displays the list of all labels stored in the cluster object. The selected item is the label being displayed.

### Color



Displayed text color. To change the color, click on the *Color* button. A color editor will appear which you can use to choose the desired color.

### Font size



Displayed text size.

## 1.21 ClusterSample

This module converts a set of points with associated data values into a uniform scalar field. The conversion is performed by determining for each node of the uniform grid the nearest point of the point set. The data value of that point will be taken at the grid node. The input data object must be of type *Cluster*.

Press the *Apply* button to start the computation.

## Connections

### Data

Point cluster to be processed.

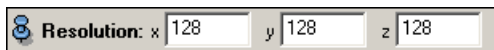
## Ports

### Variable



Option menu determining which variable should be converted into an uniform scalar field.

## Resolution

A control panel for resolution settings. It features a blue pushpin icon on the left, followed by the label "Resolution:". To the right of the label are three text input fields labeled "x", "y", and "z". Each field contains the number "128".

This port provides three text field specifying the resolution of the resulting scalar field in x-, y-, and z-direction. The default resolution is chosen so that a sufficiently high sampling rate is obtained provided the input points are distributed homogeneously.

## 1.22 ClusterView

This module visualizes data objects of type *Cluster*. The vertices of the cluster may be rendered using semi-transparent points, or using textured plates. Both, points and plates may be colored according to any data variable defined in the cluster. Vertices of the cluster may be selected or deselected by drawing a contour in the viewer window or by specifying an arithmetic filter expression. Finally, it is possible to display bonds between neighboring points. This is useful, for example, if the points represent atoms in a crystal lattice.

In plates mode (see below) individual points may be deselected by shift-clicking them. Clicking them without the shift-key pressed causes their id to be printed in the console window.

## Connections

### Data

The point cluster to be visualized.

### Colormap

A control panel for colormap settings. It features a blue pushpin icon on the left, followed by the label "Colormap:". To the right of the label is a color gradient bar transitioning from blue to red. On either side of the bar are text input fields containing the numbers "0" and "1" respectively.

The colormap used for pseudo-coloring. The alpha channel of the colormap will be correctly taken into account unless the option *opaque* has been selected. However, note that many default colormaps are completely opaque. In this case, use the *colormap editor* to make them transparent.

## Ports

### Color

A control panel for color selection. It features a blue pushpin icon on the left, followed by the label "Color:". To the right of the label is a dropdown menu with a small arrow pointing downwards.

An option menu containing all data variables which can be used for pseudo-coloring. The data variables' symbols are displayed in brackets following the names.

### Options

A control panel for visualization options. It features a blue pushpin icon on the left, followed by the label "Options:". To the right of the label are four checkboxes with corresponding labels: "plates", "filter", "opaque", and "bonds". The "opaque" checkbox is checked, while the others are unchecked.

The following four options can be set:



*Plates* activates the textured plates mode. If this mode is active each selected vertex is represented by a little sphere. For large numbers of points a significant amount of time may be required each time new colors are set. By default, the plates mode is deactivated. Instead, simple points are rendered. The point size is constant regardless of the distance of a point from the camera.

*Filter* causes a text field to be shown, which can be used to define an arithmetic expression for selecting a subset of points.

*Opaque* influences the way point primitives are drawn. By default, anti-aliased semi-transparent points are rendered. If the opaque option is set, transparencies will be ignored. Opaque rendering is faster but gives less attractive results. In addition, you can't hide certain points by making them more transparent.

*Bonds* enables the display of bonds between neighboring points. If necessary, connectivity information is computed automatically. This may take some time, especially for large data sets.

### Filter



This port allows you define an arithmetic filter expression. The filter expression is evaluated for every point. Only points for which a non-zero result is obtained are drawn. The filter expression may contain any arithmetic and logical operator defined in the C programming language. All symbols listed in the color menu may be used in the filter expression. In addition, the symbols  $x$ ,  $y$ , and  $z$  are defined. These symbol indicate the coordinates of a 3D point. For example, to select all points with positive  $x$ -coordinates and with an energy

### Point size



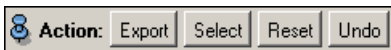
Specifies the point size in pixels. Only visible if point mode is activated, i.e., if no spheres are shown.

### Sphere scale



Allows you to adjust the size of spheres shown if option *plates* has been selected. The default sphere radius is computed from the bounding box of the data set.

### Action



The *Export* button creates a new point cluster containing all selected points of the input cluster together with the corresponding data variables.

After you press the *Select* button, you can draw a contour in the viewer window in order to deselect certain points. By default, all points inside the contour are deselected. If you keep the Ctrl-key pressed down when starting to draw the contour, all points outside the contour are deselected. Using

the Alt-key allows you to define straight-line segments. In order to be visible, a point must pass the filter test (see above) and the contour selection test. Both options are independent of each other, i.e., if you change the filter the current contour selection remains valid.

The *Reset* button resets contour selection mode. All points passing the filter test become visible.

The *Undo* button undoes the effect of the last interactive contour selection operation.

## Commands

`setBondColor <color>`

Specifies the color used for drawing bonds between neighboring points. The color may be specified by either an RGB triple in the range 0...1 or by a common X11 color name, e.g., *red* or *blue*.

`setBondWidth <width>`

This command allows you to change the width of the lines representing the bonds between neighboring points. By default, lines are drawn one pixel wide.

## 1.23 ColorCombine

This module combines up to three source fields into an *RGBA color field*. The resulting field has the same dimensions as the field connected to *source1*. Therefore it is imperative that there be a *source1*. *ColorCombine* connects to color fields, scalar fields, and scalar fields with a colormap. The relative scale of the input fields is determined by their bounding boxes. The module supports three different ways of merging the input data. "Alpha Weighted" weights the different inputs according to their alpha values, i.e., inputs with a large alpha value become more prominent in the resulting field. "Average" is the simple geometrical average of the inputs. "Add & Clamp" adds the values, making sure they do not exceed 255. If all connected source fields are scalar byte fields of the same dimensions, the module offers two more options: "colormaps", which does exactly the same as the procedures above, yet implemented in a much more efficient way, and "RGB planes", which interprets *source1*, *source2*, and *source3* as the R, G, and B values of the resultfield, respectively.

Press the *Apply* button to start the computation.

## Connections

**Source1** [required]

A scalar or color field. This one determines the size of the result field.

**Source2** [optional]

Another scalar or color field.

**Source3** [optional]

And yet another one.

### Colormap1 [optional]

The colormap for source field 1.

### Colormap2 [optional]

The colormap for source field 2.

### Colormap3 [optional]

The colormap for source field 3.

## Ports

### Colormap1



Colormap input port for source field1.

### Colormap2



Colormap input port for source field2.

### Colormap3



Colormap input port for source field3.

## Mode



These radio buttons let you choose the combining algorithm. *alpha weighted* weights the different inputs according to their alpha values, i.e., inputs with a large alpha value become more prominent in the resulting field. *average* is the simple geometrical average of the inputs. *add & clamp* adds the values, making sure they do not exceed 255. This port is only visible if the *ColorMode* port is set to *colormaps*.

## ColorMode



This port is only visible if all source fields are of equal dimensions and if they are all byte fields. It means that faster algorithms are now in use and offers the special choice *RGB planes*, which makes source1, source2, and source3 the R, G, and B components of the result respectively.

## 1.24 Colorwash

The *Colorwash* module helps you to visualize two scalar fields in combination, e.g., medical CT data and a dose distribution. The module is attached to an *OrthoSlice* module visualizing the first field, e.g., medical CT data. The image of the *OrthoSlice* is modulated so that it also encodes the second field, e.g., a dose distribution. The standard modulation technique is to multiply color into an underlying gray scale image. This explains the name of the module.

In order to use the module, first select the scalar field to be colorwashed, e.g., a dose distribution. Then choose *Colorwash* from the popup menu of an existing *OrthoSlice* modules. The new *Colorwash* module automatically connects to the selected scalar field. Alternatively, of course you can also connect the *Data* port of the module by hand.

### Connections

**Data** [required]

The 3D scalar field to be colorwashed.

**Module** [required]

Connection to the underlying *OrthoSlice* module.

**Colormap** [required]

Connection to a *Colormap*.

### Ports

**Fusion Method**



Specifies how the underlying *OrthoSlice* image is modulated. In any case the scalar field connected to the *Data* port is first mapped to a color image using the colormap connected to the *Colormap* port. This color image is then combined with the *OrthoSlice* background. The following modes are supported:

*Multiply*: The colors (scaled to the range 0...1) are multiplied. A colormap containing bright colors or white should be used, such as *amira's temperature.icol*. [Click here for an example.](#)

*Add*: The colors (scaled to the range 0...1) are added and clamped. Both the background image and the colormap should not be too bright in order to avoid overflows.

*WeightedSum*: The background image and the color image are blended using a weight factor which can be adjusted in a separate port (see below). If the weight factor is 0.5 the two images are just averaged.

*MagicLens*: A checkerboard pattern is generated, displaying the two images in the different squares. The square size can be adjusted in a separate port (see below).

*Overlay*: In this mode a data range can be specified. Inside this range the background image is replaced by the color image.

*AlphaBlending*: The background image and the color image are blended depending on the alpha values stored in the colormap. If the colormap contains no semi-transparent values at all, opacity is computed from the luminance of the color values.

### WeightFactor



Interpolation factor used in *WeightedSum* mode.

### LenseWidth



Width of the checkerboard tiles in *MagicLens* mode.

### OverlayRange



Range of data values where the overlay image replaces the background in *Overlay* mode.

## Commands

### setNearestNeighbor

Enables nearest neighbor interpolation for regular scalar fields. By default regular fields are interpolated trilinearly, except for label fields, which are evaluated using nearest neighbor interpolation in any case.

## 1.25 CombineLandmarks

This module merges an arbitrary number of *vertex set* objects into a single *landmark set*.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

Accepts a *vertex set* object. As soon as this port gets connected, an additional source port will be created. This allows to merge a practically unlimited number of data objects.

### Source2 [optional]

See above.

## Ports

### Unify sets



If this option is chosen, the resulting landmark set will contain only one set of landmarks, i.e., all vertices will be added to the same set. Otherwise, the vertices of the different input objects will be added to different sets. In order to ensure that all sets have the same number of markers the input object with the maximum number of vertices is determined and all other sets are filled with (0,0,0) if necessary.

## 1.26 CompareLatticeData

This module takes to regular fields with the same dimensions and the same number of data variables per voxel as input and computes the average difference per voxel between both. In addition, a new field with the point-wise difference of both inputs can be generated.

Press the *Apply* button to start the computation.

## Connections

### InputA [required]

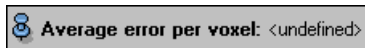
A 3D regular field. Any coordinate type, primitive data type, or number of data values per voxel are accepted.

### InputB [required]

A 3D regular field with the same dimensions and the same number of data values per voxel as input A.

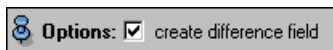
## Ports

### Average error per voxel



Displays the difference between the two input data sets.

### Options



If this toggle is set a new field with the point-wise difference between both input data sets is created.

## 1.27 ComputeContours

This module compute contours for a 3D label set using 2D cutting planes orthogonal to the selected axis (x,y or z). There is one cutting plane per each slice of the set; the plane has the same coordinate as the slice. If the labeled set contains subvoxel accuracy informations (weights) the contours can be computed using this information too (this is an option and by default is on). It is possible to compute only the contours that separate one certain material; this can be done by selecting a material from a menu. The default settings compute contours between all the materials.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

The label set to extract contours from.

### Ports

#### Orientation



Controls the direction used for contours computing. The cutting planes will be orthogonal to this axis. For example, if z axis is selected, all the contours will be contained in xy planes.

#### Options



There is only one check button which controls the use of subvoxel accuracy information, if present.

#### Materials



Selects a material; only contours separating this material from others will be computed. By default, all the materials and implicitly all the contours are considered.

## 1.28 ConnectedComponents

This module searches connected regions in a 3D image volume. The regions are detected based on thresholding, i.e., a region is a set of adjacent voxels with intensity values lying inside a user-defined range. The module is useful for counting cells on a dark background.

Press the *Apply* button to start the computation.

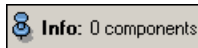
## Connections

### Data [required]

Image dataset to be analyzed. Only byte fields are supported.

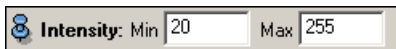
## Ports

### Info



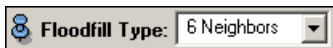
After pressing *Apply*, this port will display the number of detected regions, the volume of the smallest and the largest region, and the average volume.

### Intensity



Voxels with values outside this intensity range are considered to be part of the background.

### Floodfill Type



Defines the connectivity type. In case of *6 neighbors* voxels with a common face are considered to be connected. In case of *18 neighbors* voxels with at least one common edge are considered to be connected. Finally, in case of *26 neighbors* voxels with at least one common vertex are considered to be connected.

### Size



Minimal and maximal size in voxels of the regions. Regions smaller or larger than this range are considered part of the background. If *Max* is zero no upper size limit will be implied.

### Output



If *Region field* is checked, a scalar field of type byte will be created. Voxels which were considered to belong to a region are set to some non-zero value, while background voxels are set to zero. Different connected regions will be assigned different values, so that the regions can be visualized using color coding. However, since the output type is byte the region labels are not unique if more than 255 components are found. The regions can be visualized in 3D by using e.g., an *Isosurface* module with threshold *0.5*. Alternatively, the result can be cast to a label field using *CastField* and then the *SurfaceGen* module can be applied.

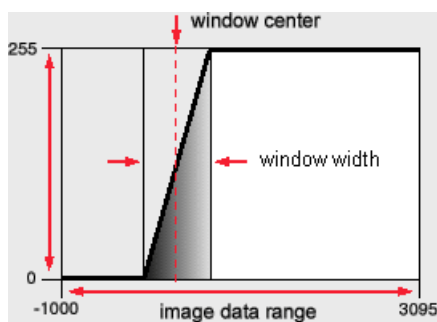
If *Spreadsheet* is checked, a *spreadsheet* object will be created, containing a list with the size and position of all detected regions.



## 1.29 ContrastControl

The *ContrastControl* module (*Contrast* in short) is an extension to the *OrthoSlice* and *ObliqueSlice* modules. It is particularly useful for the gray or color value analysis of scalar fields like medical image data, as well as a preliminary stage for image segmentation. It allows a fast and intuitive adjustment of the transfer function, mapping scalar values stored with the image data to those values used for visualization.

With the help of *ContrastControl* the linear mapping of the data values to a subset of scalar values or indices to color values, defined by a *Colormap*, can be modified. The term *windowing* is often used within this context. The *window* defines a possibly narrowed view to the data's scalar values, for enhancing the contrast of certain structures within the image data. The width of the *window* defines the range of data values between a lower and an upper threshold which shall be mapped to a range of values specified by the height of the *window*. In terms of gray value mapping a subset of thousands of possible data values can be mapped to i.e. 256 gray values, linearly distributed on a ramp, varying from black (0) to white (255). Black values are represented by the lower border of the *window* and white values by the upper border. The center of the *window* might be located at any value of the image data, thus the entire window can be shifted to various points for data evaluation.



The *window's* center and width can be modified with the two sliders in the Properties Area of the *Contrast* module. These values can either be adjusted by shifting the sliders to the left or right, varying the values by a certain percentage of the image data range, or they can be specified numerically via the appropriate entry fields.

For an overview of the current mapping, a graphical representation of the mapping function can be displayed within the *viewer window*. In order to do so the toggle button in the Properties Area with the *Window Show* label must be activated. The horizontal extent of the *window graph* specifies the image data range from the lowest to the highest value. The linear ramp indicates the mapping function where the area below or above the ramp represents the currently chosen mapping *window*.

Experienced users can directly use the mouse for a fast and simultaneous modification of the *window's* center and width. The contrast can be adjusted via the left mouse button while the shift button is pressed. Moving the mouse to the left or right moves the center of the *window* accordingly. Up and

down movements increase or decrease the width of the *window*. A vertical line (that means lower and upper threshold are identical respectively the *window* width equals zero) indicates a binary mapping of the image data values. All values below the *window*'s center will be mapped to black and all other values will be mapped to white. Moving the mouse further down will invert the mapping, thus exchanging black and white values.

## Connections

### Module [required]

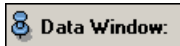
Connection to an *OrthoSlice* or *ObliqueSlice* module.

### Data [required]

Connection to the scalar field is automatically established via the *OrthoSlice* module.

## Ports

### Data Window



This port displays information on the image data range. In verbose mode the current data window will be displayed, too.

### Center



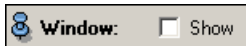
The window center can be positioned between the minimum and maximum value of the image data range. The increment value for slider movement is 1 percent of the image data range. More accurate values can be specified via the numeric input field.

### Width



The window width can be adjusted between 0 and the total image data range. The increment value for slider movement is 1 percent of the image data range. More accurate values can be specified via the numeric input field.

### Window



This port enables or disables the display of a graphical representation of the linear transfer function within the viewer. If it is enabled an additional port appears, which allows the specification of the graph's position. The graph will be updated synchronously with the contrast adjustment.

### Position



This port is only visible when *Window Show* is enabled. The input values are the X- and Y position of the graph within the viewer. The lower left corner of the viewer has the coordinates (0, 0). Negative coordinates are relative to the right respectively upper border of the viewer. Both values can also be modified by moving the mouse pointer into the appropriate entry field, pressing the shift key and moving the mouse.

## Commands

`verboseMode {0|1}`

Setting `verboseMode` to a value not equal to zero leads to a permanent refresh of the current data window settings via the mapping port. Due to the possibilities of changing the window settings quickly this leads to flickering results and furthermore reduces the interactive adjustment speed. A value of 0 is the default.

`showWindowGraph`

This is just a command line interface for the window port, bringing the window graph up front.

`hidewindowGraph`

This is the counterpart to `showWindowGraph`, removing the window graph.

`info`

Prints out a short info to the *ContrastControl* module.

`setLineColor <r> <g> <b>`

Using `setLineColor` you can change the visual appearance of the window graph. This is useful when the currently chosen background color interferes with the line color of the graph. The RGB triple specifies how much a certain color component contributes to the resulting color. These values are clipped to 0 and 1.

`setMouseSensitivity [<value>]`

Direct contrast adjustment with the mouse (Shift - Left Mouse Button) in interactive mode can be amplified or damped using this command. The default value is 0.5 times 1 percent of the total image data range. The value is clipped to 0.1 and 1

`setPosition <x> <y>`

The position of the window graph can be modified with `setPosition`. Negative values are relative to the right and upper border of the viewer. If no window graph appears with `showWindowGraph` try `setPosition 0 0` which should set the display to the lower left corner.

`setScaleFactor <factor>`

The size of the window graph can be modified using `setScaleFactor`. This might be useful if the window graph is disturbing the visual output of the viewer.

## 1.30 CorrelationPlot

This module computes a 2D correlation histogram of two uniform scalar fields (e.g. image stacks). It detects regions of correlated intensity in two images of the same object. The result of correlation analysis can be applied for an image segmentation because regions of high correlation are likely to represent a unique material or tissue type. The module can be connected to a *MultiChannelField* or to two separate images of the same dimensions. The correlation histogram is computed as follows:

- The data values of each input image are subdivided into a user defined number of intervals.
- Element  $(i, j)$  of the 2D histogram contains the number of all voxels where the data values fall into interval  $i$  for the first image and interval  $j$  for the second image.

Typically, *CorrelationPlot* is used for co-localization analysis in fluorescence microscopy, where the spatial distribution of two fluorochromes has been recorded in two different channels. There, the module finds the regions where the fluorescence signal is high in both images.

A further application of *CorrelationPlot* is the segmentation of multi-modal medical images such as CT and MRT. In this case, the images must be registered, i.e., the object position must be identical in both images. Use module *Registration* for achieving this alignment, followed by module *Resample* for resampling the model dataset to the lattice of the reference dataset. For the correlation analysis of medical image data from different modalities, *CorrelationPlot* lets you define arbitrary intensity ranges.

Besides the 2D histogram, the module computes basic statistics for the image pair such as the number of co-localized voxel pairs, the Peak-Signal-to-Noise-Ratio, and the correlation coefficient.

A graphical representation of the correlation histogram is shown in an extra 2D plot window using a user-defined colormap. The objective of a correlation analysis is to find and select regions of high correlation, which occur as local maxima in the 2D plot. Such regions can be selected either by manually drawing in the 2D plot window or by numerically entering subrange limits.

The selected regions can be used for a segmentation of the image datasets. For this, *CorrelationPlot* optionally generates a *LabelField*. The number of labels is given by the number of selected regions in the 2D plot. For each selected region, all voxels of the image datasets with corresponding data values are assigned to the respective label.

### Connections

#### Source1 [required]

First input, must be a scalar field with regular coordinates or a *MultiChannelField* field.

#### Source2 [optional]

Second input, must be a scalar field with regular coordinates or a *MultiChannelField* field.

The module takes the first two fields that it can find in the input objects connected to the two source ports. If, for instance, a *MultiChannelField* field with three channels is connected to source1

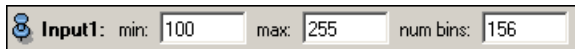
and a scalar field to source2, the module takes the lattices from channel1 and channel2 of the *MultiChannelField* as input lattices, ignoring the third channel and the scalar field connected to source2.

### Colormap [optional]

Optional colormap used to map the values of the correlation histogram. The range values of the colormap are ignored by this module.

## Ports

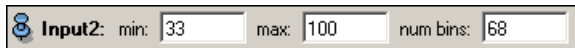
### Input1



The **Input1** port control interface consists of three text input fields. The first field is labeled "min:" and contains the value "100". The second field is labeled "max:" and contains the value "255". The third field is labeled "num bins:" and contains the value "156".

The first two text fields define the data range of *Input1* used for computing the correlation histogram. Typically, the number of background voxels (low intensity or 0) that co-localize in two images overwhelms those of the structures of interest. Therefore, *min* and *max* can be set to exclude certain intensity ranges from the analysis. The text field labeled *num bins*: adjusts the number of bins and thus the coarseness of the histogram. In the case of input fields with byte or short as primitive data type the number of bins is internally adjusted such that each bin has the same integer width.

### Input2



The **Input2** port control interface consists of three text input fields. The first field is labeled "min:" and contains the value "33". The second field is labeled "max:" and contains the value "100". The third field is labeled "num bins:" and contains the value "68".

Defines data range as well as the number of bins for the 2nd input. See description above for details.

### Gamma correction



The **Gamma correction** port control interface features a slider control. The slider has a triangular handle positioned towards the left. To the right of the slider, a numerical value "0.05" is displayed.

This port defines the value of an exponent *gamma* used to specify the mapping of the histogram entries to color values. First, the histogram entries are normalized to their maximal value. Then, before the color is looked up, an element  $x$  of the correlation histogram is replaced by  $x^{gamma}$ . Choosing a gamma value less than 1 emphasizes small values. This is useful for similar reasons as described for port *Input1*.

### Colormap



The **Colormap** port control interface includes a color bar and two numerical input fields. The color bar is a horizontal rectangle with a checkerboard pattern transitioning from dark on the left to light on the right. The first input field is labeled "Colormap:" and contains the value "0". The second input field contains the value "1".

Before plotting them, the histogram entries are normalized to values between 0 and 1. The colormap connected to this port is used for the histogram look-up. The range of the colormap will be ignored. If no colormap is connected, a gray map is used.

### Selection by



The **Selection by** port control interface has two radio button options. The first option is labeled "manual draw" and is currently unselected. The second option is labeled "subrange" and is selected, indicated by a filled circle.

Two methods are supported for selecting regions of the histogram. In *manual draw* mode one can

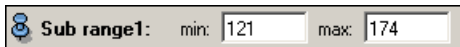
use one of the draw tools from the tool bar at the upper border of the plot window. In the second mode labeled *subrange*, one can define a range of values for each of the two input fields. This selects a rectangular subregion of the histogram. In *subrange* mode, two additional numbers are computed as described below.

### Region action



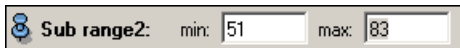
Press this button to remove the last selected region. This port is only shown in manual drawing mode.

### Sub range1



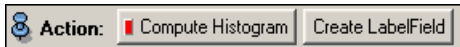
This port is only available in *subrange* mode. Here the boundaries of the subrange for Input1 are defined. To change the subrange with a visual feedback in the plot window, use the mouse wheel to single step or hold down the shift key while moving the left pressed mouse for making bigger steps.

### Sub range2



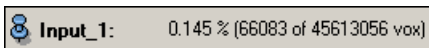
Like the above port for Input2.

### Action



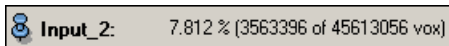
The *Compute Histogram* button recomputes the correlation histogram and pops up the plot window. The *Create LabelField* button computes a *LabelField* using the current selection.

### Input\_1



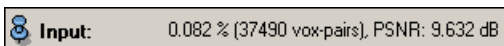
Prints relative (%) and absolute (n of  $n_{tot}$ ) numbers of voxels specified in the *min* and *max* fields of port *Input1*.

### Input\_2



Prints relative (%) and absolute (n of  $n_{tot}$ ) numbers of voxels specified in the *min* and *max* fields of port *Input2*.

### Input



Prints the fraction (relative to total number of voxel pairs) and absolute number of voxel pairs within

specified interval. Only these pairs are considered for the histogram generation. The second value is the Peak-Signal-to-Noise-Ratio [dB], which is calculated according to

$$\text{PSNR} = 10 * \log_{10}\left(\frac{1}{\text{MSE}}\right), \quad (1.1)$$

with MSE being the (normalized) mean square error.

### Correlation

 **Correlation:** -0.035

Prints the correlation coefficient of the voxel pairs considered for histogram generation.

### Selection

 **Selection:** 17.839 % of Input (6688 vox-pairs) in sub range

Prints the relative (% , relative to Input) and absolute numbers of selected voxel pairs.

### Sub range\_1

 **Sub range\_1:** 37.339 % of Input\_1 (24675 vox)

This port is available only in *subrange* mode. It prints the number of voxel pairs inside the set specified by the first subrange. The final selection is the intersection of the voxel pair sets defined by the first and the second subrange.

### Sub range\_2

 **Sub range\_2:** 47.067 % of Input\_2 (1677191 vox)

This port is available only in *subrange* mode. It prints the number of voxel pairs inside the set specified by the second subrange.

## 1.31 CreateCluster

This module converts a vertex-set object to a point cluster object. The point cluster object stores the coordinates of the vertices but does not keep any other information such as the connectivity of a surface.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

The vertex-set object to be converted.

## 1.32 Curl

The *Curl* module computes the curl of a vector field consisting of floats defined on a uniform grid. The output is another uniform vector field.

$$\text{curl}V = \left( \frac{\partial V_z}{\partial y} - \frac{\partial V_y}{\partial z}, \frac{\partial V_x}{\partial z} - \frac{\partial V_z}{\partial x}, \frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y} \right)$$

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

Vector field defined on a uniform grid (*UniformVectorField3*). The vector components must be floats.

## 1.33 CurvedSlice

The *CurvedSlice* module lets you display arbitrarily curved slices through a 3D scalar field or along a curve for a 2D field. A 3D scalar field having a dimension less than 2 along one axis is treated like a 2D field. Three mapping methods are available for mapping data values to colors or gray levels.

### Connections

**Data** [required]

The 2D or 3D field to be visualized. Currently, regular scalar fields and RGBA color fields with uniform or stacked coordinates are supported. For 3D fields, the slice is curved through the field. For 2D fields, the whole field is extruded along the curve.

**Curve** [optional]

Curve defining a fence. The curve can be either a BSpline or a LineSet. When using a LineSet, a BSpline is automatically created and connected.

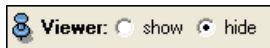
**Colormap** [optional]

The colormap used to map data values to colors. This port is ignored when *linear* or *histogram* equalized mapping is selected.



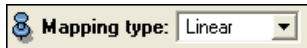
## Ports

### Viewer



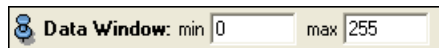
Extra viewer representing the texture.

### Mapping type



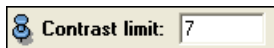
This option menu controls how scalar values are mapped to screen colors. For *linear* mapping, a user-defined data window is mapped linearly to black and white. If *histogram* is selected, an adaptive histogram equalization technique is applied. This method attempts to show all features of the data, even if a wide range of values is covered. Finally, *colormap* can be used to activate pseudo-coloring.

### Data Window



This port is displayed if *linear* mapping is selected. It allows you to restrict the range of visible data values (brightness and contrast). Values below the lower bound are mapped to black, while values above the upper bound are mapped to white.

### Contrast limit



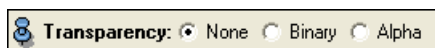
This port is only visible if *histogram* equalization is selected. The number determines the contrast of the resulting image. The higher the value, the higher the contrast in the resulting image. A value of zero means that contrast will not be limited at all.

### Colormap



This port is displayed if *colormap* is selected. Choose a colormap to map data to colors.

### Transparency



This radio box port determines the transparency of the slice. *None* means that the slice is fully

opaque. *Binary* means that black parts are fully transparent while other parts are opaque. *Alpha* means that opacity is taken as is. If a colormap is used for visualization, opacity values are taken from there.

### Sampling



This port provides four choices for specifying the texture resolution:

*coarse*: lowest resolution. For 3D fields, 128x128. For 2D fields, size-of-field divided by 8.

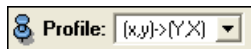
*medium*: medium resolution. For 3D fields, 256x256. For 2D fields, size-of-field divided by 4.

*fine*: intermediate resolution. For 3D fields, 512x512. For 2D fields, size-of-field divided by 2.

*finest*: highest resolution. For 3D fields, 1024x1024. For 2D fields, size-of-field.

For 2D data fields, selecting *finest* resolution ensures that the texture and the field have the same resolution.

### Profile

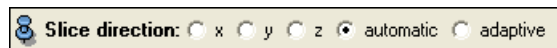


This port is only shown for 2D fields, and specifies the coordinate relation between the profile and the 2D field.

$(x,y) \rightarrow (X,Y)$ : coords of profile correspond to  $(x,y)$  values of field.

$(x,y) \rightarrow (Y,X)$ : coords of profile correspond to  $(y,x)$  values of field.

### Slice direction



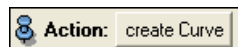
This radio box port specifies the slice orientation.

*x, y, or z*: slice is extruded along the corresponding axis.

*automatic*: slice direction is computed using all points of the curve. The computed slice direction is used for the entire profile.

*adaptive*: slice direction on each profile point is computed with the normal at the point.

### Action



Creates a new curve object.

## Commands

`createImage <image-base-name>`

This command creates a 2D image from the slice and adds it to the Pool. The image base name can be omitted.

## 1.34 Cutting Plane

The *Cutting Plane* module can be connected to any display module derived from *ViewBase*. The module provides a special cutting plane of finite size which can be rotated, translated, and scaled interactively. The module then computes the intersections of all triangles shown by the *ViewBase* module with the cutting plane. The resulting line segments are stored in a *LineSet* object. Optionally, the module can be used without any input. In this case the intersection of any surface-like geometry shown in the main viewer is computed.

*Note:* Surfaces and tetrahedral grids can be more easily intersected with a plane using the *Intersect* module. This module also provides a Tcl command which can be used to export a *LineSet* object.

## Connections

**Data** [optional]

A display module derived from *ViewBase*, e.g., *Isosurface* or *SurfaceView*. If no input is specified the entire scene will be cutted with the plane.

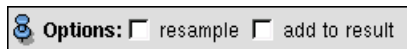
## Ports

### Orientation



These three buttons reset the orientation of the cutting plane to the xy, xz, or yz plane, respectively.

### Options



If *resample* is checked, the resulting lines are resampled so that line segments of equal length are obtained. The length of the resampled line segments can be adjusted using a separate slider (see below). If *add to result* is checked the resulting lines will be added to an existing result object. Otherwise, for each cutting operation a new result will be created.

### SampleDist



This port is only visible if the *resample* option is checked (see above). It specifies the preferred length of the resampled line segments.

## Action



The *Cut* button actually computes the intersecting lines. The *Clear result* button removes all line segments from the current result.

## 1.35 CylinderSlice

This module displays the values of any scalar field on a cylinder. The cylinder surface is mapped on a planar slice and shown in an extra viewer. The cylinder is specified by the dragger position. The height is computed by the length of the diagonal of the bounding box of the scalar field. The module provides the same ports as *ObliqueSlice*. Additionally, it allows to save the cylindrical slice to an image. The image format is selected by the file name extension.

## Connections

**Data** [required]

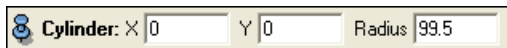
The scalar field.

**Module** [required]

The *OrthoSlice* or *ObliqueSlice* that specifies the dragger plane.

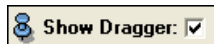
## Ports

### Cylinder



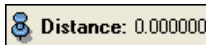
The position of the cylinder center with respect to the base plane and the cylinder radius.

### Dragger



Switches the dragger on and off.

### Distance



Display the distance between the cylinder center and the the center of the slice.

## Commands

`createImage <image base name>`

This command creates a 2D image from the cylindrical view and adds it to the Pool. The image base name can be omitted.

## 1.36 DataProbe

The three data probing modules *PointProbe*, *LineProbe*, *SplineProbe* are used to inspect scalar or vector data fields. They have many features in common so they are described in one section. The probes are taken at a point (*PointProbe*) or along a line (*LineProbe*, *SplineProbe*) which may be arbitrarily placed. The controlpoints of the data probing modules determine the locations where the samples are to be taken. *PointProbe* has one controlpoint which is the samplepoint, *LineProbe* has two controlpoints which are the endpoints of a line which is subdivided into a number of segments given by the Samples port or by the Distance port, and *SplineProbe* has at least three controlpoints that define a spline that goes through the endpoints and approximates the points in between. The spline curve is subdivided into a number of segments given by the Samples port or by the Distance port by which the sample points are obtained.

To place the controlpoints within the bounding box of the given geometry you can either type in the coordinates in the port Points (see below) or you can shift the points interactively with the mouse. The latter can be done by turning the 3D viewer into interactive mode and picking and moving the crosshair dragger of the current point. The plot immediately changes if the immediate mode toggle is set. Usually the sampled values are plotted against the length of the probe line or as a bar in case of *PointProbe* in an extra *plot window*. If you use more than one data probing module of the same type all plotted curves will be shown in one plot window.

*PointProbe* displays the value at the sample point and the material if material values are set. *LineProbe* and *SplineProbe* display the length of the line resp. spline.

A module of type *ProbeToLineSet* can be connected to a *LineProbe* or a *SplineProbe* module in order to save the probe line and the sampled values.

### Connections

#### Data [required]

Can be connected to arbitrary 3D fields. The *LineProbe* module can also be attached to 3D input objects other than fields such as surfaces or Open Inventor geometry. In this case the *LineProbe* doesn't sample any data but simply measures distances.

### Ports

#### Orientation



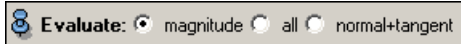
This port which is used in *LineProbe* and *SplineProbe* provides three buttons to specify the probe line orientation. Axial probe lines are perpendicular to the x-y-plane, frontal probe lines are perpendicular to the x-z-plane, and sagittal probe lines are perpendicular to the y-z-plane. If one of these buttons is pressed the start and end coordinates of the probe line are set to the minima resp. maxima of the corresponding axis and to the center of the two axes perpendicular to the probe line.

## Options



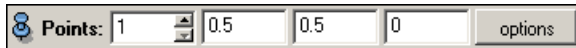
The *immediate* toggle determines whether the samples are taken while the controlpoints are being moved or when the motion is finished. If the *orthogonal* toggle is on all points are moved in sync, when the coordinates of one point are changed with the dragger. This port is not shown for module *PointProbe*.

## Evaluate



If the probe is connected to a vector field, these radiobuttons are shown. If the *magnitude* button is set the magnitude of the vectors is shown in the plot window. With the *normal+tangent Comp.* button set you get the normal and tangential components as two curves. Setting the *all* button shows all components of the vector field as separate curves.

## Points



Here you can see resp. type in the coordinates of all controlpoints the data probing module makes use of. The options menu lets you toggle whether a dragger or a sphere is shown for the controlpoints. You can also append, insert or remove controlpoints if this module is a *SplineProbe* module.

## Control



With this radio buttons you can choose which of the following two ports are taken to compute the sample points. This port is not shown for module *PointProbe*. If you have chosen the *adaptive* button, the module tries to keep the number of samples which can be seen in the plot window as close as possible to the number given with the appropriate slider. I.e. the more you zoom into the plot window the more exact are the curves in the plot window.

## Samples



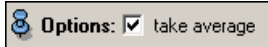
This slider allows you to choose the numbers of samples along the probe line. This port is not shown for module *PointProbe*.

## Distance



This slider allows you to choose the distance between two consecutive sample points along the probeline.

## Options



The *average* option averages the probe values by taking samples on a disk perpendicular to the sampling points and smoothes the sampled values along the sampling line(s). This button is only available for a LineProbe resp. SplineProbe module.

## Radius



The radius of the sampling disk. This slider is only shown if the above average option is chosen.

## Longitudinal Width



The width determines how many sampling values are used for smoothing. This slider is only shown if the above average option is chosen.

## Plot



If the *Show* button is pressed a *plot window* appears where the sampled values are plotted against the length of the probe line. *Note:* There will be only one plot window regardless of how many Line Probe modules there are in your setup. Every line probe is represented in that plot window by a curve bearing the name of the corresponding module.

## Commands

`getInterpol`

Returns the currently used interpolation method.

`setInterpol {none|linear|spline}`

Sets the interpolation method. *none* means no interpolation at all and the sample values are taken at the controlpoints only.

`getOrder`

Returns the order of the spline probe.

`setOrder <value>`

Sets the order of the spline probe.

`getPoint [<index>]`

Returns the coordinate of the requested controlpoint. Index defaults to 0.

`setPoint [<index>] <x> <y> <z>`

Sets the coordinates of controlpoint *index*. Index defaults to 0.

**getSamplePoints**

Returns the coordinates of all points where samples are taken.

**getSampledValues**

Returns all sampled values.

**setImmediate {0|1}**

Switches the immediate mode on or off, i.e. data is shown while the probe line resp. point is being moved.

**setOrtho {0|1}**

Switches the orthogonal mode on or off, i.e. all controlpoints of a probe line are moved in sync or not.

**getNumPoints**

Returns the number of control points of a probe line. In case of a point probe 1 is returned.

**getLength**

Returns the length of the probe line. 0 in case of a point probe.

**appendPoint <x> <y> <z>**

Appends a controlpoint with the given coordinates.

**insertPoint <index> <x> <y> <z>**

Inserts a controlpoint with the given coordinates as the  $\text{index}^{th}$  point.

**removePoint <index>**

Removes the given controlpoint.

## 1.37 Delaunay2D

This module takes a set of 3D vertices and produces a Delaunay triangulated surface with 2D topology. In order to do so the vertices are projected either into the xy-, xz-, or yz-plane or alternatively onto a cylinder or sphere whose axis is parallel to the x-, y-, or z-axis. By default the cylinder or sphere is automatically positioned so that it matches the center of the vertices. However this *projection center* can be defined by the user.

The incoming data object must be derived from a vertex set, cf. Section ???. Note that the Delaunay algorithm may take some time and possibly memory, especially for large data sets. The closed sphere or closed cylinder projection mode may imply significant additional computation time. If the input is not of planar topology, like, e.g., a sphere, the result will probably not be useful. If multiple 3D vertices project to the same 2D vertex, the result is undefined. In the latter case, it might be help to issue a `jitterPoints 0.001 0.001 0.001` command on the command line.

If the input data set is of type *Cluster* and if this cluster has data associated to it than these data values are converted into one or more surface scalar fields.



Press the *Apply* button to start the computation.

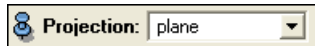
## Connections

**Data** [required]

The vertex set to be triangulated.

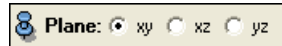
## Ports

### Projection



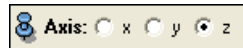
Lets you select whether the 3D points should be projected into a plane or onto a cylinder or a sphere. If *closed cylinder* or *closed sphere* is selected then a closed surface will be created.

### Plane



This port lets you select on which plane the 3D vertices should be projected on in case of a planar projection.

### Axis



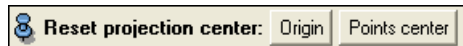
This port lets you select the orientation of the cylinder or sphere axis in case of a cylindrical or spherical projection.

### Projection center



This port let you change the center for cylindrical or spherical projections . By default this is the center of vertices of input data.

### Reset projection center



Lets reset the center coordinates to origin (0,0,0) or vertices center.

### Parameters



The first number defines an optional internal scaling applied to the 3D vertices. The direction in which the scaling is applied depends on the selected projection type. The option is useful if the 3D points are distributed non-isotropically, i.e., if the average feature size is different in the three

coordinates. *Note that an external scaling defined with the transformation editor is currently not taken into account by this module.*

The second number lets you specify a maximum edge length in 3D space. Delaunay triangles with edges longer than this limit are discarded. When computing the 3D edge length the scaling factor defined by the first parameter is taken into account. When a new input is connected and no other input was connected before the max edge length field is initialized with half of the average edge length of the bounding box of the data set. A maximum edge length of 0 indicates that no triangles should be discarded.

## Commands

`setTriangulator {0|1}`

Lets you select the triangulation algorithm. `McDelaunay` is provided for compatibility with `amira 3.1` and earlier. `MeshVizDelaunay` (0:default) is a faster implementation.

## 1.38 DemoMaker

Using the *DemoMaker* module, you can create an animated sequence of operations, e.g. for automatically running demonstrations or for advanced movie recording. Select *Create/Animation/Demos/DemoMaker* from the menu to add a *DemoMaker* module to the Pool.

### Defining and storing a demo sequence

The demo sequence is a set of *actions* over a time range as represented by the module's *Time* port. A wide range of such actions can be defined, e.g. switching a toggle value on or off, varying the numerical value of some other module's port over time, and defining breaks to interrupt the automatic demo sequence.

Actions are defined by first selecting an entry from the *GUI elements* port. This port lists all GUI elements of all modules that currently exist in the Pool, except the ports of the active *DemoMaker* module itself.

Once you have selected the right element (or a different action type like a pause, break, go-to, or an arbitrary Tcl command) from the *GUI elements* menu, additional ports will show up below the *GUI elements* port, depending on the type of the selected element (e.g. numeric, toggle, button, ...). Using these ports, you can define the time (or time range) of the action on the *DemoMaker Time* line as well as the value to which the selected GUI element will be set at that time (see *example* below).

When action time and values have been defined, press the *Add* button in the *Event List* button line. This adds the desired action to the *DemoMaker*'s list of events as represented by the *Event List* menu. Multiple sequential, overlapping, or parallel actions may be contained in one demo sequence.

Once a part of the desired action sequence is defined, simply store it by choosing *File / Save Network...* from the menu. *DemoMaker* in its current state will be saved along with the rest of the Pool. When saving the network, make sure the *Time* slider is in the desired starting position.

## Playing a demo sequence

For playing the demo sequence as defined, simply press the *Time* slider's play button (triangle pointing to the right) or press the corresponding function key F4. The sequence will run from the current time step to the end of the demo, or to the next break as defined by the user (see *Defining a break* below). The play button will change into a stop button, and pressing that button or pressing the F3 function key immediately stops playing. If the demo stops at a user-defined break, continue by again clicking the play button or pressing F4.

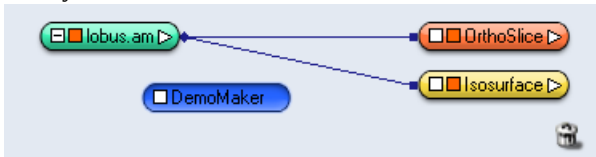
If the demo is stopped at the beginning or at some user-defined break, you can press the function key F10 to jump to the point of the next break, or to the end of the demo. Similarly, pressing F9 jumps to the previous break or to the very beginning of the action sequence. Once you have jumped to the right step of the demo, simply press play or F4 again to play it from there.

If the action sequence is running too fast or too slow, you can adjust its speed using the *Time* port's *Configure* option (right-click onto the time port, see *time port documentation*). Note that the playback speed can only be adjusted for the complete time range at once. If you want only to change the speed of only part of the sequence, you must do so by adjusting the time range of the corresponding action entries.

## Usage Example

Here is a small example to demonstrate the way *DemoMaker* works. Load *lobus.am* from the *tutorials* subdirectory of the *amira* installation directory. Connect an *OrthoSlice* module and an *Isosurface* module. Create a *DemoMaker* by selecting it from the *Create/Animation/Demo* menu.

Now your network should look similar to this:



All available user interface ports of the modules in the Pool should now appear in the *GUI element* port of the *DemoMaker* module. If not, press the *Update* button.

Now select *OrthoSlice/Slice Number* from the *GUI elements* menu. Fields for specifying start/end values and start/end time appear. Enter 86 and 0 as start and end values and 0 and 0.4 as start/end time:

Now press the *Add* button to add this action event to the event list. Press the time slider's play button

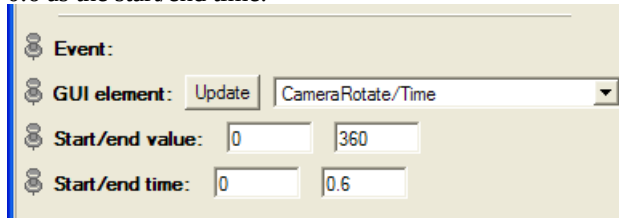
to see the demo sequence just specified. You should see the *OrthoSlice* plane moving during the time range from 0 to 0.4, but nothing happening in the time range from 0.4 to 1.0.

Now select a threshold of 70 in the *Isosurface* module and press *Apply* to make the surface appear in the viewer. Now go back to *DemoMaker*, select *OrthoSlice/Viewer mask/Viewer 0* from the *GUI elements* menu, select *on* as the toggle value and 0.4 as the trigger time:



Now press *Add* again to add this action to the list. Jump back to 0 by clicking on the time slider. You should see that when jumping back (from later than 0.4 to earlier than 0.4) the *Isosurface* is switched off. Then play the demo from the beginning, and you see that the surface is switched on right after the *OrthoSlice* is moved.

As another exercise you can create a camera path (select *Create/CameraPath* or *Create/CameraRotate* from the menu and edit the camera path to your liking. Then press *Update* in *DemoMaker* and select *CameraPath/Time* or *CameraRotate/Time* as GUI element. Now the start/end value refers to the value of the camera time slider. Enter the minimum and maximum values of that time slider. Enter 0.0 and 0.6 as the start/end time:



Now you see that in parallel to the actions defined before, the camera path is applied to the scene. Note that you can execute only parts of a camera path, or concatenate multiple paths this way. However, multiple camera paths cannot be used in parallel.

## Connections

### Data [not used]

The *data* port is a default script object port, but is not used by *DemoMaker*.

### Time [optional]

The *Time* port is a quick way for synchronizing other modules with *DemoMaker*'s *Time* port, e.g. for using the *MovieMaker* module or camera path modules like *CameraRotate* or *CameraPath*. Alternatively, you can use camera path objects with *DemoMaker* by selecting these objects' time sliders from the *GUI element* port and making it part of the *DemoMaker* event list.

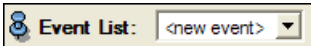
## Ports

### Time



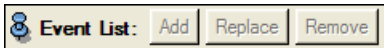
The time slider defines the time range in which the demo sequence is played, and by clicking on the slider, one can jump to an arbitrary point of the demo. Clicking one of the play buttons (triangles pointing to the left/right for backwards/forwards) will start playing the demo. Right-clicking on the port brings up a dialog box allowing you to change the range of the slider (min/max value) as well as the increment determining the playback speed (smaller increment means slower playback). For more information, see documentation of the *time* port.

### Event List (Selection)



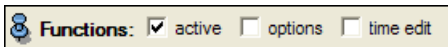
This menu contains the list of currently defined demo actions that make up the demo sequence. The buttons below this menu always refer to the currently selected menu entry.

### Event List (Buttons)



Manipulate the event list menu. *Add* adds the event defined in the lower part of the *DemoMaker* module to the list of demo action events. When selecting one of the action entries from the *Event List* menu, the *Remove* button deletes this entry from the list, and the *Replace* button replaces the selected entry by the event as defined in the lower part of the *DemoMaker* module.

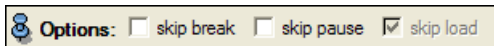
### Functions



Activate/deactivate this *DemoMaker* instance, and show optional parts of the *DemoMaker* user interface, such as some option ports and some ports for editing the time line.

- *active*: activate/deactivate this module. When deactivated, the module will not define any function keys and not manipulate objects in the Pool. Deactivating is especially important when using multiple alternative *DemoMaker* objects.
- *options*: display option ports (see *Options (1)* - *Options (3)* below).
- *time edit*: display ports for editing the time line (see *Move from* - *Time edit* below).

### Options (1)

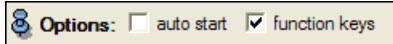


This port is only shown when the *options* toggle of the *Functions* port is enabled.

- *skip break*: do not stop at user-defined breaks when playing the demo sequence.

- *skip pause*: do not execute user-defined pause events (no waiting time).

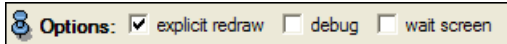
## Options (2)



This port is only shown when the *options* toggle of the *Functions* port is enabled.

- *auto start*: lets the demo start automatically when the network containing the *DemoMaker* module is loaded.
- *function keys*: when toggled off, no function keys (F3/F4/F9/F10) will be defined when the network containing the *DemoMaker* module is loaded. This is especially important when combining multiple *DemoMaker* modules, since only one module can define the function keys.

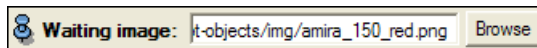
## Options (3)



This port is only shown when the *options* toggle of the *functions* port is enabled.

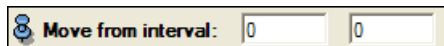
- *explicit redraw*: when toggled on, for each time step the active viewers are explicitly called to perform a redraw. If toggled off, *amira*'s *auto redraw* feature is used. If in some cases the desired action sequence is not properly displayed in the viewer, try toggling *explicit redraw* on.
- *debug*: when toggled on, the actual Tcl commands that are executed during the demo sequence are output to the *amira* console. This will significantly slow down the graphical display, but can be used for understanding what exactly happens in the demo sequence.
- *wait screen*: enable a waiting image to be displayed during jumps on the time line. This is only useful if jumping takes considerable amount of time, e.g. for very long event sequences. If enabled, the *Waiting image* port will appear below this option for specifying the waiting screen image file.

## Waiting image



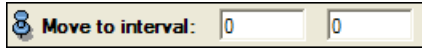
If the *wait screen* option is enabled, this port is used to specify the file name of the waiting screen image. The image will be displayed during jumps on the time slider.

## Move from interval



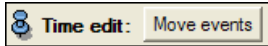
This port is only shown when the *time edit* toggle of the *Functions* port is enabled. Specify the time interval to be relocated on the time line (see *Moving events on the time line* below).

### Move to interval



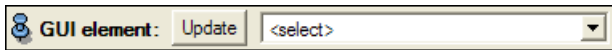
This port is only shown when the *time edit* toggle of the *Functions* port is enabled. Specify the target time interval for relocation on the time line (see *Moving events on the time line* below).

### Time edit



This port is only shown when the *Time edit* toggle of the *Functions* port is enabled. *Move events*: move all events within the *Move from* interval to the *Move to* interval on the time line (see *Moving events on the time line* below).

### GUI element



The selection menu contains all GUI elements in the current Pool that can be manipulated using *DemoMaker*. If the Pool is modified (e.g. by loading a new dataset or connecting a new module), press *Update* to update the contents of the selection menu.

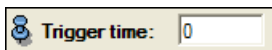
### Additional Ports

Additional ports will appear in the *DemoMaker* module, depending on the type of *GUI element* the user has chosen. These ports are listed in the following section, grouped by GUI element types.

## Defining a break

Selecting *\*Break, continue on keypress* from the GUI element menu lets you insert a break in the demo sequence. When playing, the demo will stop at that point. See *playing a demo sequence* above.

### Trigger time

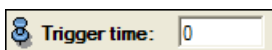


Defines the point in time (on the time slider) at which the break is inserted.

## Defining a pause

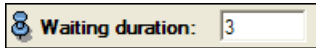
Selecting *\*Pause, waiting time* from the GUI element menu lets you insert a pause, where the demo will stop and wait for a user-defined amount of time. Please note that such a pause will not be executed if *DemoMaker* is driven by a *MovieMaker* object. In such a case, you must insert a pause by adjusting the time slider range and adapting the time range of the action entries.

### Trigger time



Defines the starting point of the pause on the time slider.

## Waiting duration

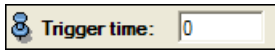


Defines the waiting time in seconds. Please note that this time is taken in addition to the time defined by the time slider range.

## Defining a go-to

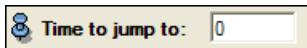
Selecting *\*Go-to, jump to user-specified time step* from the GUI element menu lets you jump from the one point in the demo sequence to another point. For example, you can simply define an endless loop by jumping back to some previous time step. You can end the loop by pressing the stop button or F3 (see *playing a demo sequence* above).

### Trigger time



This defines the point in time (on the time slider) at which the go-to is inserted.

### Time to jump to

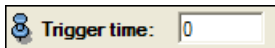


This defines the point in time which the go-to will jump to. For example, to create a loop between time 0.2 and 0.4, insert a go-to with *Trigger time* 0.4 and *Time to jump to* 0.2.

## Defining a toggle

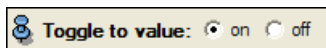
Toggle actions can toggle certain GUI elements between the two states *on* and *off*. Examples for toggle values are the different options in a *ToggleList* port<sub>i</sub>/a<sub>z</sub>, or the orange *viewer mask* toggle(s) present in every data object or display module (to switch the display of the module in the viewer on and off).

### Trigger time



This defines the point in time (on the time slider) at which the value is toggled.

### Toggle to value



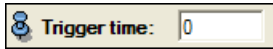
This defines the value (on or off) to which the selected GUI element is set at the defined trigger time, if the demo sequence is played forwards. When playing or jumping backwards, the inverse value is used.



## Defining a button press

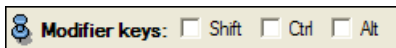
Button actions emulate the pressing of a button or executing certain events with no parameters. Examples for buttons elements are the *ButtonList port*, or inverting the orientation of a clipping plane.

### Trigger time

A control box with a blue icon on the left, followed by the text "Trigger time:" and a text input field containing the number "0".

This defines the point in time (on the time slider) at which the button is pressed or other action is taken.

### Modifier keys

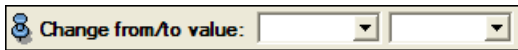
A control box with a blue icon on the left, followed by the text "Modifier keys:". To the right are three checkboxes: "Shift", "Ctrl", and "Alt", all of which are currently unchecked.

This defines whether *DemoMaker* should emulate that **Shift**, **Ctrl**, or **Alt** is held down at the time when the button is pressed. The meaning of these modifier keys depends on the module defining the button to be pressed.

## Defining a select action

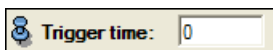
Select actions are used to set a port's value to one of a set of choices as offered by a *selection menu* or a *radio box*.

### Change from/to value

A control box with a blue icon on the left, followed by the text "Change from/to value:". To the right are two dropdown menus, both currently showing a downward-pointing arrow.

This defines from which old value to which new value the port will be switched at the specified trigger time. When playing or jumping backwards, the two values will be used inversely.

### Trigger time

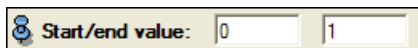
A control box with a blue icon on the left, followed by the text "Trigger time:" and a text input field containing the number "0".

This defines the point in time (on the time slider) at which the port is set to a new selection value.

## Defining a numeric action

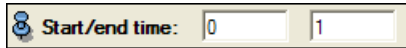
Numeric actions are used to vary a port's numerical value from a start value to an end value over time. Examples for numeric ports are *numerical sliders* or *numerical text fields*.

### Start/end value

A control box with a blue icon on the left, followed by the text "Start/end value:". To the right are two text input fields; the first contains the number "0" and the second contains the number "1".

The two fields of this port define from which start value to which end value the value of the GUI element will be varied. The values of this port are constrained to the range of the corresponding GUI element.

### Start/end time

A GUI element for setting a time range. It consists of a blue icon with a play symbol, followed by the text "Start/end time:", and two adjacent input boxes. The first box contains the number "0" and the second box contains the number "1".

This defines the time range (on the time slider) during which the port value will be varied from the start value to the end value specified above. When playing or jumping backwards, the value will be varied from the end value to the start value.

## Defining a Tcl command

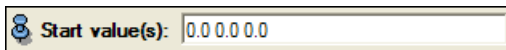
When selecting *\*Tcl command* from the GUI element menu, you can insert arbitrary Tcl commands into the demo sequence. Furthermore, like for *numeric events*, the command can be parameterized by one or more numeric values that will be varied between user-defined start and end values.

### Tcl command

A GUI element for entering a Tcl command. It features a blue icon with a play symbol, followed by the text "Command:", and a single, wide text input box.

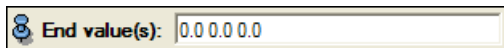
Text field for typing the desired Tcl command. Within the command, special placeholders %0%, %1%, %2%, ... can be used that will be replaced by numeric values. The start and end values for these placeholders are defined in the ports described below.

### Start value(s)

A GUI element for specifying start values. It includes a blue icon with a play symbol, the text "Start value(s):", and a text input box containing the value "0.0 0.0 0.0".

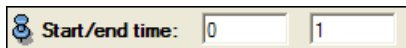
If the numeric placeholder %0% is used in the *Command* field, specify the start value for this placeholder. If multiple placeholders are used (e.g. %0% and %1%), specify space-separated start values for all of the employed placeholders.

### End value(s)

A GUI element for specifying end values. It features a blue icon with a play symbol, the text "End value(s):", and a text input box containing the value "0.0 0.0 0.0".

If the numeric placeholder %0% is used in the *Command* field, specify the end value for this placeholder. If multiple placeholders are used (e.g. %0% and %1%), specify space-separated end values for all of the employed placeholders.

### Start/end time

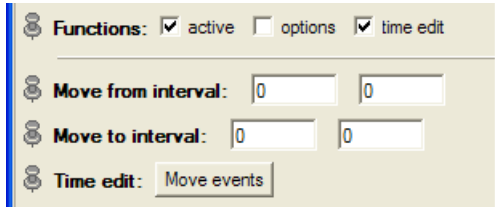
A GUI element for setting a time range. It consists of a blue icon with a play symbol, followed by the text "Start/end time:", and two adjacent input boxes. The first box contains the number "0" and the second box contains the number "1".

This defines the time range (on the time slider) during which the specified Tcl command will be executed, with the placeholders replaced by values between the specified start and end values.

## Moving events on the time line

After you have defined some events on the time line, you may want to move a part of this event sequence to a different time, e.g. for inserting more events at a certain point, or for stretching or compacting part of the demo sequence.

This can be easily done if you switch on the *Time edit* toggle of the *Functions* port:



Simply enter the time interval that you want to relocate in the *Move from* port, enter the destination time interval in the *Move to* port, and press the *Move events* button. If the new time interval is outside of the current min/max time, the time slider configuration will be updated accordingly.

## Commands

The following commands are methods of the *DemoMaker* script object. They can be called externally by first specifying the name of the script object (e.g. *DemoMaker*), followed by a space and the name of the method. Example: *DemoMaker play* starts playing the demo sequence.

### *play*

start playing the time slider at its current time step. This function is called when pressing F4.

### *stop*

stop playing the time slider. This function is called when pressing F3. Also calls the *stop callback* (see below).

### *jumpNext*

jump to the next user-defined break, or to the end of the demo. This function is called when pressing F10.

### *jumpPrev*

jump to the previous user-defined break, or to the start of the demo. This function is called when pressing F9.

### *jump <breakidx>*

jump to user-defined break. If *;breakidx;* is 0, this command will move to the beginning of the demo. Otherwise it will be jumped to the *;breakidx;*'th break.

### *setEndCallback <cmd>*

sets an internal callback function to the specified Tcl code *<cmd>*. This code will be executed only once when *DemoMaker* reaches the end of the time slider. Call with an empty string to disable the callback.

### *writeDescriptionFile*

opens a file dialog, where you can specify a file to which the description file should be saved. The description file is an xml-file containing several predefined xml-tags describing the demo. Most

important of these tags are the commands to step through the demo. The description file is used to automatically generate demosequence files and html pages for steering the demo.

## DemoMaker tips and limitations

Although the *DemoMaker* module is quite powerful, some things should be noticed, and there are some known limitations that are listed here for convenience. If you find further important limitations or bugs, please report them to Mercury Computer Systems hotline as a bug or feature request.

- The *\*Load network* action type for loading a different network file is currently not implemented.
- For *button action*, "snapping" a button to automatic update mode is not supported yet.
- GUI elements that are part of a *Generic port* are not properly represented in the *GUI elements* menu of the *DemoMaker* module.
- If a *DemoMaker* module is renamed (e.g. by selecting *Edit / Rename...* from the menu), the function keys F3/F4/F9/F10 will no longer work. However, simply disabling and re-enabling the *function keys* option in *DemoMaker* helps.
- If you rename or remove modules after defining events with *DemoMaker*, then events involving these modules will generate errors. Please properly arrange your network before using *DemoMaker*.

### Further links

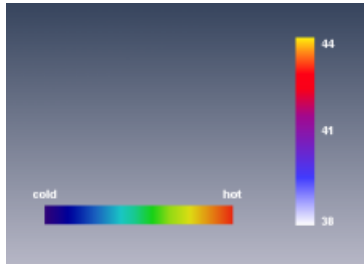
*script object documentation* for script programming.

## 1.39 Digital Image Filters

Encloses the *ImageEditor* in a compute module.

## 1.40 DisplayColormap

This module allows you to position a colormap icon in the 3D viewer. This is useful, for example, to produce snapshots that contain an explanation of what specific colors mean. Note that although colormaps are ordinary data objects in *amira*, they often are hidden by default. Use the *Pool/Show Object* menu of the main window to display their icons in the Pool.



## Connections

**Data** [required]

The colormap to be displayed.

## Ports

### Options

 **Options:** ☐ custom text ☒ vertical ☒ rel. size ☒ transp. bg

This port provides the following four toggles:

- **custom text:** Enable or disable custom text (see below).
- **vertical:** Vertical orientation instead of horizontal.
- **relative size:** Change size of colormap when viewer size changes.
- **transparent background:** If off, render background rectangle.

### Position

 **Position:** x  y

Position of colormap in viewer relative to lower left corner. If one of the numbers is negative, it is interpreted relative to upper right corner.

### Size

 **Size:** length  thickness

Length and width of the colormap in pixels. If the option *relative size* has been selected, the length and width are interpreted relative to a window size of 1000x800. If the actual viewer window is smaller (or larger), then the displayed colormap will be smaller (or larger).

### Custom Text

 **Custom Text:**

This is only available if the *custom text* option is selected. You may specify a space-separated list

of values to be displayed. In addition, you may specify a text string that is displayed instead of the number by using the `'/'` character. The example in the image above has been generated using this text: `-8/cold 50/hot`, which displays "cold" at value -8 and "hot" at the value 50 (the colormap in this case ranges from -8 to 50). If the label text contain blanks, you must enclose the text in double quotes, e.g., `100/"very hot"`.

## Commands

**getFontSize**

Returns the current font size.

**setFontSize** <points>

Changes the font size. The default font size is 14 points.

**setBGColor** <color>

Sets the color of the background rectangle which is drawn when the transparent background toggle is off.

**setColor** <color>

Sets the color of the text.

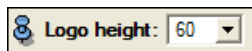
## 1.41 DisplayLogos

This module displays the product logo (amira) as well as the Mercury Computer Systems logo in the viewer window. In order to do this, a specific Open Inventor file is loaded that contains transparent raster graphics files. The user can choose between different versions, each having a different size.

Internally, the *DisplayLogos* script object loads the Open Inventor file and creates an *IvDisplay* module. Choose *Pool/Show All* to see these hidden modules in the Pool.

## Ports

**Logo height**



Choose the desired size of the logo.

## Related Topics

The resource file for this module also defines the global Tcl command

```
createLogos [size]
```

to easily display the logos. For convenience, this command hides the newly created *DisplayLogos* module in the Pool. The optional argument **size** allows you to specify the size of the logo, where only those values offered in the module's *Logo height* port are valid.

## 1.42 DisplayTime

This module displays the value of a time object in the 3D viewer using a textual or iconic representation. This is useful for animations which shall contain an illustration of time. Four major styles are available: plain text, a round clock, a horizontal time bar, or a vertical time bar.

This module should be connected to a *Time* object. In this case the current time is visualized. Besides that it is possible to visualize an arbitrary value which can be set using the commands of the invisible port *value*.

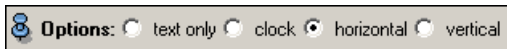
### Connections

**Time** [optional]

The time object to be displayed.

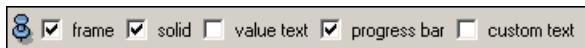
### Ports

#### Options



This port chooses between the four major draw styles.

#### Options2



This port adjusts additional settings influencing the drawing. The last three options are disabled for the *text only* draw style.

- **frame** Draw a frame around the time icon.
- **solid** Fill the icon's background.
- **value text** Show value together with time icon.
- **filled value** Fill space between zero and value.
- **custom text** Whether or not custom text should be displayed (see below).

#### Position



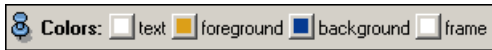
Position of the time icon with respect to the lower left corner of the screen. If negative values are entered the icon is positioned relative to the right and/or top of the screen.

#### Size



Specifies the length of the horizontal or vertical time bar (in pixels) and relative thickness of the bars. If a clock is drawn *length* refers to the diagonal, while *thickness* is ignored. For text only display both values are ignored.

### Colors



The colors of the various parts of the icon can be set here.

### Custom Text



This is only available if *custom text* option is selected. You may specify a space separated list of values, that shall be displayed. Alternatively you may specify a text, which is displayed instead of the number, by using the *'*' character. The syntax is the same as for the *Display Colormap* module.

### Format



The format for displaying the time value (printf syntax of the C programming language).

### Value

This port is always hidden. It can be used to specify a time value when no time object is connected (via the Tcl interface).

## Commands

**setFontSize <points>**

Changes the font size. The default font size is 28 points for text only displays and 14 points otherwise.

## 1.43 GetCurvature

This module computes curvature information for a discrete triangular surface of type *Surface*. Either the maximum principal curvature value, the reciprocal curvature value, or the direction of maximum principal curvature can be computed. The algorithm works by approximating the surface locally by a quadric form. The eigenvalues and eigenvectors of the quadric form correspond to the principal curvature values and to the directions of principal curvature. Note that the algorithm does not produce meaningful results near locations where the input surface is not topologically flat, i.e., where it has non-manifold structure.

Press the *Apply* button to start the computation.



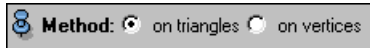
## Connections

### Data [required]

The surface for which curvature information should be computed.

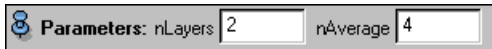
## Ports

### Method



Radio box allowing the user to select between two different computational algorithms. Choice *on triangles* produces a surface field with curvature values or curvature vectors being defined on the surface's triangles. Alternatively, by selecting *on vertices* a surface field with data being defined on the vertices can be generated.

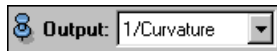
### Parameters



The first input, denoted *nLayers*, determines which triangles are considered to be neighbors of a given triangle and which points are considered to be neighbors of a given point. If the value of this input is 1, then only triangles sharing a common edge with a given triangle are considered to be neighbors of this triangle and only points directly connected to a given point are considered to be neighbors of this point. For larger values of *nLayers* successively larger neighborhoods are taken into account.

The second input, denoted *nAverage*, determines how many times the initial curvature values computed for a triangle or for a point are being averaged with the curvature values of direct (first-order) neighbor triangles or points. The larger the value of *nAverage* the smoother the curvature data being obtained. Note that averaging only applies to the scalar curvature values, not to the directional curvature vectors which are computed when port output is set to *max direction*.

### Output



This menu controls the output of the curvature module. If *curvature* is selected then a surface scalar field is generated containing the maximal principal curvature of a triangle or of a point. If *1/curvature* is selected then the curvature values are inverted. In this case the output values have the dimension of a length, indicating the radius of a sphere locally fitting the surface.

The *mean curvature* and *1/mean curvature* are similar to the first two options. Here, however, the mean value of the two principal curvature values is computed. This quantity will be negative in strictly concave regions and positive in strictly convex regions. It can be zero in regions where a positive and negative principal curvature cancel each other.

The *Gauss curvature* is the product of the two principal curvatures. It is negative in surface areas with hyperbolic geometry (convex-concave, like near saddle points) and positive in areas with elliptic geometry (strictly convex or strictly concave).

If *max direction* is selected then a surface vector field is computed indicating the direction of maximum principal curvature. The length of a directional vector is equal to the corresponding curvature value.

## 1.44 GridView

This module allows you to visualize the grid structure of a regular 3D scalar or vector field with stacked, rectilinear, or curvilinear coordinates. The nodes of the underlying grid can be addressed using an index triple (i,j,k). The *GridView* module extracts slices of constant i, j, or k index out of the grid and displays them as a wireframe model. In principle, the module may also be connected to a field with uniform coordinates. However, it will not be listed in the popup menu of such fields.

### Connections

**Data** [required]

The regular field to be investigated.

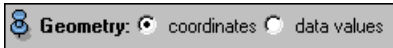
### Ports

#### Orientation



Specifies whether slices in the ij, ik, or jk plane should be displayed. For stacked as well as rectilinear coordinates the indices i, j, and k refer to the x, y, and z direction, respectively.

#### Geometry



This port is only visible if the module is connected to a real-valued 3D vector field. In this case the grid may be built using the *data values* instead of the node's *coordinates*. This is a useful option if the vector field contains displacement vectors.

#### Slice



Allows you to select the constant node index, e.g., the k index if orientation is set to ij.

## 1.45 Grouping

This module allows you to define arbitrary groups or surface triangles and/or elements of tetrahedral or hexahedral grids. The groups can be used to easily display parts of more complex objects. The grouping module can be connected to one or more ordinary display modules like *SurfaceView*, *GridVolume*, or *HexaView* (in fact any module derived from *ViewBase* will work). In order to define a group first select the parts of the object you are interested in using the selection mechanisms provided by the input modules. For example, in case of a *SurfaceView* module different parts can be selected via the materials menu, via the selection box (buffer show/hide), via 2D lasso selection (buffer draw), or by selecting or deselecting individual triangles with the mouse. Once the parts of the object you want to be in a group are visible, select the *New group* button. Later, this particular display state can be restored by choosing the group from the *Groups* combo box again.

Internally groups are defined in a bitfield (one bit for each selectable element). The bitfields are stored in the parameter section of the corresponding data objects. If the objects are saved in the AmiraMesh or HxSurface file format after the groups have been defined the group definitions are saved too. In addition, the group definitions will also be included in network files. Note that group definitions may become invalid if the data objects are modified, e.g., if the number of triangles of a surface is reduced using the simplification editor.

### Connections

#### Data [required]

Connection to a display module like *SurfaceView*, *GridVolume*, or *HexaView*.

#### Module2 [optional]

Connection to an additional display module connected to some other data object than the first one. If multiple inputs are connected to the grouping module elements of all input objects can be grouped together.

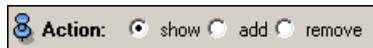
### Ports

#### Groups



This combo box lists all groups currently being defined. Selecting a group from the box causes the elements contained in that group to be shown, to be added to the view, or to be removed from the view, depending on the value of the *Action* port (see below).

#### Action



This radio box defines what happens when a new group is chosen in the *Groups* menu. If *show* is selected, the elements of the new group become visible and all other elements become invisible.

If *add* is selected the elements of the new group are added to the view. If *remove* is selected the elements of the new group are removed from the view. If *nop* is selected nothing happens. This last option allows you to change the *Groups* port without changing the view.

### Rename



This text port allows to rename the current group. By default new groups are called *Group1*, *Group2*, etc.

### Edit



This port provides three buttons for creating a new group containing all elements currently being visible (selected, not highlighted), for deleting the current group without changing the selection, and for replacing the contents of a group by the elements currently being visible.

## 1.46 HeightField

The *HeightField* module offers another method for the visualization of scalar data fields defined on regular grids, such as stacks of tomographic images. The data are visualized by extracting an arbitrary axial, frontal or sagittal slice out of the volume. This slice is represented as a surface for which the heights of the vertices are mapped to the data, using a tunable scale parameter.

### Connections

#### Data [required]

The scalar or color field to be visualized. If an RGBA color field is connected the vertices of the height field are colored as in the color field. The displacement is computed from the fields grayvalue intensity which is computed using the formula  $0.3*R+0.59*G+0.11*B$ .

#### Colormap [optional]

The colormap used to map data values to colors.

### Ports

#### DrawStyle



This port is inherited from *ViewBase*. For a description see there.

#### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

### Slice Number



This slider allows you to select different slices.

### Scale



This slider allows you to select the desired scale for the data-heights mapping. This should be in the range [-1,1].

## 1.47 Histogram

This module computes the histogram of the data values of a scalar field and plots it in a separate plot window. In addition, the mean value and the standard deviation are printed.

Press the *Apply* button to start the computation of the histogram and pop up the plot window containing it.

### Connections

#### Data [required]

The scalar field to be investigated. Regular fields, fields defined on tetrahedral or hexahedral grids, and fields defined on surfaces are supported.

#### ROI [optional]

Connection to a module providing a region-of-interest *SelectRoi*. If such a module is connected, only the volume within the ROI is evaluated in the histogram.

#### Labels [optional]

A uniform label field which can be used to restrict the histogram to a certain material.

### Ports

#### Info



Prints the mean value and the standard deviation of all input data values once the histogram has

been computed. Both info ports are only shown when the *Apply* button has been pressed, i.e. the histogram has been computed.

### Range

A control interface for the 'Range' parameter. It features a blue circular icon with a question mark on the left. The text 'Range:' is followed by 'min' and a text box containing '21', then 'max' and a text box containing '243'. To the right is a 'Reset' button.

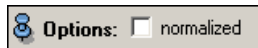
Defines the data range over which the histogram is to be computed. The *Reset* button adjusts the data range so that it completely covers the values of the input object.

### NumBins

A control interface for the 'Num bins' parameter. It features a blue circular icon with a question mark on the left. The text 'Num bins:' is followed by a slider control with a triangle marker in the middle, and a text box on the right containing the value '55'.

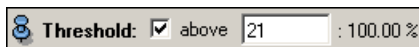
Defines the number of bins (intervals) the data range is divided into. The histogram counts the number of data values falling in each bin. In the case of integer input values, i.e., bytes, shorts, or ints, the number of bins is internally adjusted so that all bins have the same integer width. This is required in order to avoid aliasing effects.

### Options

A control interface for the 'Options' parameter. It features a blue circular icon with a question mark on the left. The text 'Options:' is followed by a checkbox labeled 'normalized'.

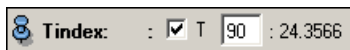
If the *normalize* toggle is set, the histogram is scaled so that its area equals one. Otherwise the total number of counts is shown.

### Threshold

A control interface for the 'Threshold' parameter. It features a blue circular icon with a question mark on the left. The text 'Threshold:' is followed by a checked checkbox, the word 'above', a text box containing '21', and a percentage display showing ': 100.00 %'.

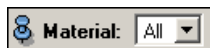
Defines a threshold value and displays the percentage of all data values which lie above the given threshold. The percentage corresponds to the given range. The threshold value is shown as a yellow vertical markerline in the plot window. This markerline can be moved within the plot window by pressing the left mouse button over the markerline and moving it to the new position. The new position is then taken as the new threshold value. The computation and display of the threshold value can be toggled on or off. This port is only shown when the histogram has been computed.

### Tindex

A control interface for the 'Tindex' parameter. It features a blue circular icon with a question mark on the left. The text 'Tindex:' is followed by a colon, a checked checkbox, the letter 'T', a text box containing '90', and a percentage display showing ': 24.3566'.

The tindex value defines a reversed threshold value, i.e., a tindex of 90 returns the value where 90% of the data values lie above that value. It is denoted by the reddish markerline in the plot window. The computation and display of the tindex value can be toggled on or off. This port is only shown when the histogram has been computed.

### Material

A control interface for the 'Material' parameter. It features a blue circular icon with a question mark on the left. The text 'Material:' is followed by a dropdown menu currently showing 'All'.

If a label field is connected to the port *Labels* and the input data field is a regular scalar field, this port provides the opportunity to compute the histogram for one material only. Otherwise it is hidden.

## 1.48 InterpolLabels

The computational module *InterpolLabels* interpolates between the slices of a labelfield parallel to the z-axis. The input must be a uniform label field.

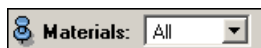
### Connections

**Data** [required]

Input data object of type *Uniform label field*.

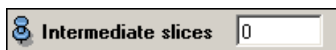
### Ports

#### Material

A dropdown menu with a blue icon on the left, the label "Materials:", and a selection box showing "All".

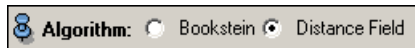
Choose if you want to interpolate all the labels or just a particular one. Note that 'Exterior' cannot be selected.

#### Intermediate slices

An input field with a blue icon on the left, the label "Intermediate slices", and a text box containing the number "0".

Choose how many intermediate slices should be interpolated. The default is computed such that a voxel is as homogeneous as possible.

#### Algorithm

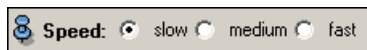
A selection control with a blue icon on the left, the label "Algorithm:", and two radio buttons. The first is labeled "Bookstein" and is unselected. The second is labeled "Distance Field" and is selected.

Two methods are implemented: The computation of an implicit function that minimizes a thin-plate-spline functional is used to interpolate between slices (Bookstein).

The other method computes the 2D distance field of each slice and interpolates between them.

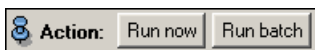
The Bookstein method takes much longer than the distance field method. It is recommended to run this computation as a batch job (see below).

#### Speed

A selection control with a blue icon on the left, the label "Speed:", and three radio buttons. The first is labeled "slow" and is selected. The second is labeled "medium" and is unselected. The third is labeled "fast" and is unselected.

The slow options tries to interpolate every material by the Bookstein method. If this fails, the distance field method is used. The medium option tries to use the Bookstein method only if the dimension of the problem is small enough. The fast option uses only the distance field method.

#### Action

Two buttons with a blue icon on the left, the label "Action:", and two buttons labeled "Run now" and "Run batch".

Pushing these buttons triggers the computation. You can chose between an online computation or a batched computation. The last is advisable for the computation of the Bookstein method. You can also just compute the contours for debugging.

## 1.49 InterpolateLabels

This module adds intermediate slices to a label field, thereby interpolating the labels. This is useful e.g., to avoid stair-case effects in data sets with a large slice distance compared to the slice resolution.

Press the *Apply* button to start the computation.

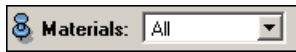
### Connections

**Data** [required]

Connects to a label field.

### Ports

#### Materials

A port control for 'Materials' featuring a blue icon with a plus sign, the label 'Materials:', and a dropdown menu currently showing 'All'.

You can interpolate for all or for one material only. Multiple selected materials can be successively added to the result.

#### Intermediate slices

A port control for 'Intermediate slices' featuring a blue icon with a plus sign, the label 'Intermediate slices', and a text input field containing the number '2'.

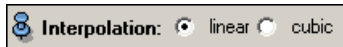
For a uniform label field, this port specifies the number of intermediate slices. e.g., if your input data set had 5 slices and you would specify 2 here, your result would have 13 slices.

#### Attempted slice distance

A port control for 'Attempted slice distance' featuring a blue icon with a plus sign, the label 'Attempted slice distance:', and a text input field containing '0.01'.

For a stacked label field, this port specifies the attempted slice distance for the interpolated field. Depending on the actual slice distance in the stacked field, an appropriate number of intermediate slices is inserted.

#### Interpolation

A port control for 'Interpolation' featuring a blue icon with a plus sign, the label 'Interpolation:', and two radio buttons. The 'linear' radio button is selected, and the 'cubic' radio button is unselected.

Cubic interpolation will in general generate smoother shapes.

## 1.50 Intersect

The *Intersect* module shows the intersection of a surface or of the boundaries of a tetrahedral grid and a cutting plane. For each triangle intersecting the plane a line segment is drawn. The plane description is taken from a slicing module such as *OrthoSlice* or *ObliqueSlice* connected to port *Module* (any



other orange amira module can be used as well). *Intersect* can be chosen from the popup menu of an existing slicing module. It can also be chosen directly from the popup menu of a surface or of a tetrahedral grid. In this case an *empty slicing module* is created and *Intersect* is automatically attached to it.

The line segments drawn by this module can be exported into a *line set* data object via the Tcl command `createLineSet` (see below).

## Connections

**Data** [required]

Surface or tetrahedral grid to be intersected.

**Module** [required]

Slicing module defining the cutting plane.

## Ports

**Line Width**



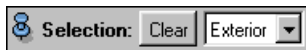
Defines the width of the intersection lines in pixels.

**Line Color**



This port defines the color of the intersection lines.

**Selection**



This port allows you to restrict the number of triangles being intersected with the cutting plane. By default all triangles of a surface or all boundary triangles of a tetrahedral grid are considered. With the *Clear* button the list of possible triangles can be resetted. Afterwards triangles adjacent to a particular material can be added again by choosing that material from the port's option menu.

**Lines**



This port is only available if the module is connected to a tetrahedral grid. It specifies whether only intersections with boundary triangles will be shown or intersections with all triangles of the grid.

## Commands

`setColor <r> <g> <b>`

Sets the color of the intersection lines.

## createLineSet

This command exports the set of currently visible line segments into a *LineSet* data object. The *LineSet* object will be added to the Pool. It then can be written into a file or can be processed further in some other way. Individual line segments will be sorted in such a way that polylines consisting of as many vertices as possible are obtained.

## 1.51 Isolines

This module computes isolines for an arbitrary 3D scalar field on a 2D cutting plane. The plane itself is defined by another module which must be connected to port *Module*. Isolines are computed using two different algorithms depending on the type of the incoming scalar field. If the scalar field is defined on a tetrahedral grid then all tetrahedra intersecting the plane are determined first. The straight isoline segments are generated for these tetrahedra according to the requested threshold values. If the incoming scalar field is not a tetrahedral one then the field is sampled on a regular 2D raster first. Each rectangular cell of the raster is then checked for isolines. Note, that this approach may lead to artifacts if the sampling density is too low.

To execute a demo script illustrating the isoline module click [here](#).

### Connections

#### Data [required]

Scalar field for which isolines are to be computed.

#### Module [optional]

Module defining cutting plane used for isoline computation.

#### Colormap [optional]

Colormap used for pseudo-coloring. If no colormap is connected all isolines have equal color.

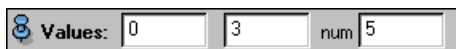
### Ports

#### Spacing



Control how isoline thresholds are being defined. In *uniform* a certain number of isovalues are distributed equidistantly within a user-defined interval. In *explicit* mode the threshold for each isoline has to be set manually.

#### Values



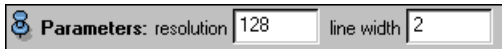
In *uniform* mode this port contains three text fields allowing the user to define the lower and upper bound of the isoline interval as well as the number of isolines being computed in this interval.

### Values

A port interface labeled 'Values' with a blue icon. It contains a single text field with the value '0.75 2.25'.

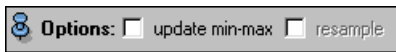
In *explicit* mode an arbitrary number of blank-separated threshold values can be defined in this text field. For each threshold a corresponding isoline is displayed.

### Parameters

A port interface labeled 'Parameters' with a blue icon. It contains two text fields: 'resolution' with the value '128' and 'line width' with the value '2'.

The *resolution* value determines the resolution of the sampling raster used for computing isolines. This parameter is ignored if the incoming scalar field is defined on a tetrahedral grid and the *resample* option is off. The second input of this port determines the width of the isolines in pixels.

### Options

A port interface labeled 'Options' with a blue icon. It contains two toggle buttons: 'update min-max' and 'resample', both of which are currently unchecked.

Two toggle buttons are provided. If *update min-max* is set then the isoline thresholds are automatically reset to some default values whenever the data range of the incoming scalar field changes. The toggle called *resample* allows you to compute isolines using the resampling approach even if the incoming scalar field is defined on a tetrahedral grid.

### Colormap

A port interface labeled 'Colormap' with a blue icon. It contains a color bar with a gradient from blue to red, and two text fields: '0' on the left and '1' on the right.

Port to select a colormap.

## 1.52 Isolines (Surface)

This *Isolines* module computes isolines for a scalar field defined on a surface made of triangles. Inside the triangles the scalar field is linearly interpolated if the scalar field contains values for every node. If the field contains values for every surface triangle only, the isolines will follow the triangle edges if the values of the neighboring triangles are appropriate.

### Connections

#### Data [required]

A scalar field defined on a *Surface* can be connected to this port.

#### Colormap [optional]

A colormap is used to map the values represented by the isolines to a corresponding color. If no colormap is connected, a constant color is used. See also *Colormap*.

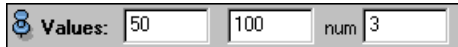
## Ports

### Spacing



Controls how isoline thresholds are being defined. In *uniform* mode a certain number of isovalues are distributed equidistantly within a user-defined interval. In *explicit* mode the threshold for each isoline can be set manually.

### Values



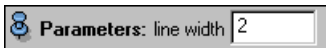
In *uniform* mode this port contains three text fields allowing the user to define the lower and upper bound of the isoline interval as well as the number of isolines being computed in this interval.

### Values



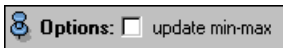
In *explicit* mode an arbitrary number of blank-separated threshold values can be defined in this text field. For each threshold a corresponding isoline is displayed.

### Parameters



The input of this port determines the width of the isolines in pixels.

### Options



If *update min-max* is set then the isoline thresholds are automatically reset to some default values whenever the data range of the incoming scalar field changes.

### Colormap



Port to select a colormap.

## 1.53 Isosurface (Regular)

This module computes an isosurface within a three-dimensional scalar field with regular Cartesian coordinates.

Very high-resolution datasets can be downsampled to reduce the number of polygons being produced. In addition, an internal polygon reduction method is provided that merges certain triangles of the original triangulation. This way, the number of triangles can be reduced up to 50%. A second independent

scalar field may be connected to the module. This field determines how the isosurface is colored. If no color field is connected to the module the isosurface has constant color.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

The scalar field defined on a regular 3-dimensional grid.

### ColorField [optional]

Arbitrary scalar field which is mapped onto the isosurface using pseudo-coloring.

### Colormap [optional]

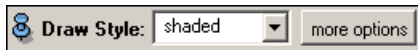
The colormap is used for pseudo-coloring the isosurface.

### PointProbe [optional]

If this port is connected to a *PointProbe* module as isovalue the value at a certain point within the scalar field will be chosen. For details see *PointProbe*.

## Ports

### Draw Style



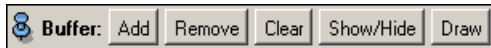
The draw style port is inherited from class *ViewBase*. For a description of this port see *there*.

### Colormap



In case a colormap is connected to the isosurface module, this colormap will be shown here. If no colormap is connected to the module the port's default color is used. To change this color, click into the color bar with the left mouse button. This will bring up the color selection dialog. To connect the port to a colormap, use the popup menu under the right mouse button. See also *Colormap*.

### Buffer



This port must be enabled explicitly with the command `Isosurface showBuffer`. For a detailed description see the *ViewBase* documentation.

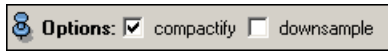
### Threshold



Determines the value used for isosurface computation. The slider is automatically adjusted to cover

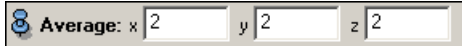
the whole range of data values. For CT data, use a value of -200 to extract the skin surface or a value of about 150 to extract the bone structures.

### Options



If the *compactify* toggle is set, up to 50% less triangles will be produced, but it results in a slightly shifted surface. This port further allows you to enable *downsampling* (see below).

### Sample Factor



This port only appears if *downsampling* is enabled. It allows you to specify a sample factor for each of the three main axes. The factors determine how many of the original grid points in each direction will be merged into one.

## 1.54 IvDisplay

The *IvDisplay* module renders an object containing *Open Inventor* geometry data.

### Connections

**Data** [required]

Connect a data object of type *IvData* to this port.

### Ports

**Draw Style**



This port provides a menu to select one of three draw styles to render the *Open Inventor* geometry.

## 1.55 IvToSurface

The *IvToSurface* computational module parses the connected *Open Inventor scene graph* (*Open Inventor* geometry data) and converts it into the surface format. This is done when you press the *Apply* button. A green icon named *GeometrySurface* representing the generated surface should appear in the Pool.

If no data object is connected, all geometry currently displayed in the first viewer is converted.

Duplicated points are removed during the conversion. Duplicated points are points with a distance less than the diameter of the bounding box times the given *Relative Tolerance* factor.

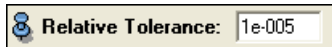
## Connections

**Data** [required]

Connect a data object of type *IvData* to this port.

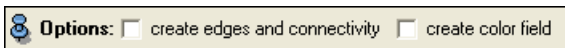
## Ports

### Relative Tolerance



Factor to determine duplicated points.

### Options



If the *create edges and connectivity* option is set, the triangle connectivity of the Surface data format is filled.

The *create color field* option controls whether an additional vector field containing color values is created from the Open Inventor data that approximates the diffuse material properties. This data can be used as a *ColorField* when displaying the geometry.

## 1.56 LabelVoxel

The *LabelVoxel* module provides a simple threshold segmentation algorithm applicable to CT or MR image data. The method is also suitable for binary segmentation of other gray level images. Up to five different regions separated by four different thresholds can be extracted. For CT images the four regions *Exterior*, *Fat*, *Muscle*, and *Bone* are predefined. However, the name of these regions as well as the corresponding thresholds may be reset by the user.

In order to find suitable thresholds an image histogram showing the (absolute) number of occurrences of voxel values and the current partitioning of the gray value range into the segments is provided. In this histogram several peaks can be identified more or less clearly, e.g., the ones for fat and muscle. The corresponding threshold or segment boundary should be set at the minima between successive peaks. The histogram window pops up by clicking on the *Histo* action button, adjustments made by moving the value sliders are shown (almost) immediately. You may change the histogram layout (scales, plots of lines and curves, colors) using the *Object Editor* which pops up when you select *Edit Objects* in the *Edit* menu of the histogram window. For more details please refer to the *Plot Tool* description.

Segmentation starts when you click on the *Apply* button. Before doing so several options may be set. These options are described in detail below. The segmentation results are stored as a *LabelField* data object. You may use the *Image Segmentation Editor* to further process this object.

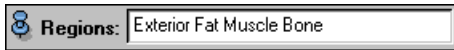
## Connections

**Data** [required]

Image data to be segmented.

## Ports

### Regions



This port lets you specify the names of the different regions to be segmented. Between two and five regions may be specified. The individual region names must be separated by blanks.

### Exterior-Fat



Lets you set the threshold separating the first and second region.

### Fat-Muscle



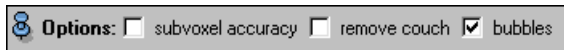
Lets you set the threshold separating the second and third region.

### Muscle-Bone



Lets you set the threshold separating the third and fourth region.

## Options



Toggle *subvoxel accuracy* causes certain weights to be computed, indicating the degree of confidence of the assignment of a voxel to a particular region. This information is used by the surface reconstruction algorithm to create smooth boundary surfaces, c.f. the *SurfaceGen* module. If no weights are present quite blocky surfaces occur. Note that weights can also be defined using the smoothing filter of the *Image Editor*.

The second option *remove couch* may be used for medical CT images if parts of the couch the patient is lying on are falsely classified as muscle or bone. In particular, the biggest connected component of voxels not assigned to the first region, i.e. *Exterior*, will be detected. Then all voxels not contained in this component will be assigned to *Exterior*.

Finally, if option *bubbles* is set an algorithm similar to *remove couch* is applied in order to detect bubbles or lung tissue inside the patient. Because of their low intensity values otherwise these regions would be assigned to *Exterior*.



## Action



The *Histo* button makes a window pop up showing a histogram for the input image data.

## 1.57 LandmarkSurfaceWarp

This module deforms a vertex set object using a number of pairs of corresponding points, which are represented by a *LandmarkSet*. The warping methods and the meaning of the ports are the same as in the *LandmarkWarp* module. Please refer to its documentation for details.

## 1.58 LandmarkView

This module displays a *landmark set* as small spheres, or in case of medical markers shows a specific geometry.

### Connections

**Data** [required]

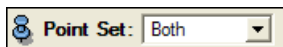
The landmark set to be displayed.

**Box** [optional]

The bounding box of the associated image data. The box is used to determine the projected positions of the landmarks.

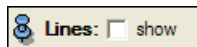
### Ports

**Point Set**



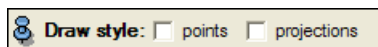
You may select whether only the first, only the second, or all point sets are shown.

**Lines**



Display lines connecting corresponding points in the first and second set. This option is only present if the input contains two or more point sets.

**DrawStyle**



Landmarks can be displayed either as spheres or as simple points. This is selected in the first radio box. If the input contains a large number of points (several hundred, or thousands), point style can significantly improve display performance.

In the second box, the projected positions are displayed as circles on the bounding box boundary. The box is either taken from the connection *Box* or computed from the landmarks.

#### Size



Size of displayed spheres or points.

#### Complexity



The larger this value is, the nicer the spheres look, but the lower the rendering speed is.

## Commands

`setColor <set> <r> <g> <b>`

Set rgb color for one landmark set (all color values in the range between 0 and 1).

`setColor <set>`

Get rgb color for one landmark set.

`setLineColor <r> <g> <b>`

Set rgb color for connecting lines.

`getLineColor`

Get rgb color for connecting lines.

`getLineWidth`

Get line width of connecting lines.

`setLineWidth`

Set line width of connecting lines.

## 1.59 LandmarkWarp

This module deforms a 3D uniform scalar field (e.g., image data) using a number of pairs of corresponding points, which are represented by a *LandmarkSet*.

Three different transformation modes are offered: *Rigid* transforms the input image by applying a global translation and rotation. *Bookstein* uses so called *thin plate splines* proposed by Bookstein.

*Flow* uses scattered data interpolation. *Bookstein* mode guarantees that all landmarks will be transformed exactly to their corresponding points. This is not the case for the two others. In all three modes nearest neighbor interpolation is used for resampling.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

The *LandmarkSet*, which defines corresponding points. It must contain at least 2 sets of landmarks.

### ImageData [optional]

The data set (3D uniform scalar field), which is to be transformed.

### Master [optional]

If this port is connected to a uniform scalar field, the output field will have the same bounding box and resolution as the master.

## Ports

### Direction



Select whether points in the first set are to be moved towards their corresponding points in the second set, or the other way round.

### Method



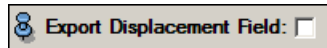
See description above.

### Premultiply Rigid



Perform a *Rigid* transformation followed by the nonrigid *Flow* transformation.

### Export Displacement Field



Export a displacement field which is defined on the output grid ( $x_{output} - x_{input}$ ). The data type of the output is given by the input data type. Usually you will need to convert the input to float first (use *CastField*).

### Beta



Only available in Flow mode: The larger this value is chosen, the more local and the less smooth the resulting transformation becomes. On the other hand for  $\beta=0$ , the transformation is constant. In detail the basis function used in the flow method is  $f(p1,p2) = \exp(-d(p1,p2) * \beta)$ .

#### Norm



The norm to measure the distance in the *flow* algorithm. Either the L1 norm  $d(p1, p2) = |x1 - x2| + |y1 - y2|$  is used or the L2 norm  $d(p1, p2) = \sqrt{|x1 - x2|^2 + |y1 - y2|^2}$ .

## 1.60 LatticeAccess

This module can be attached to *amira* data objects providing a *Lattice* interface. The *Lattice* interface is designed to access large data sets in a block-by-block fashion on different resolution levels. However, many *amira* modules require data sets to be stored completely in memory. The *LatticeAccess* module makes it possible to use such modules by converting a subblock of a large volume into a traditional memory-resident data object.

You may select the subblock either by typing numbers in the ports or by using a dragger in the viewer. Additionally, it is possible to request a subsampled version of the data. Depending on the actual data object providing the *Lattice* interface lower resolution levels are stored on disk and might be accessed rather quickly or they are calculated on the fly, which might take some time. After selecting a subblock you should press the *load* button. A memory-resident data object is created and filled with the data from disk. All standard visualization techniques may now be applied on the subblock.

Press the *Apply* button to load the subblock. If you check the *auto-refresh* box, the subblock will be immediately reloaded once the selection has been modified. This is useful for interactively exploring a large volume.

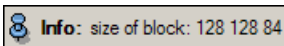
### Connections

**Data** [required]

A data object providing a *Lattice* interface.

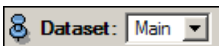
### Ports

#### Info



This port displays the size of the subblock to be loaded.

#### Dataset



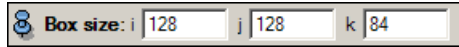
If the input object provides multiple data sets this port allows you to select the data set to be converted into a memory-resident data object.

### Box min

A control panel for 'Box min' with a blue icon on the left. It contains three input fields labeled 'i', 'j', and 'k', each with the value '0'.

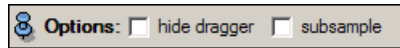
The lattice index of the lower corner of the subblock.

### Box size

A control panel for 'Box size' with a blue icon on the left. It contains three input fields labeled 'i', 'j', and 'k' with values '128', '128', and '84' respectively.

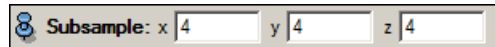
The size of the subblock to be loaded. The size is in lattice indices in the base resolution. The numbers will not change if you select subsampling but the real size of the subblock to be loaded will change. This is indicated in the *Info* port.

### Options

A control panel for 'Options' with a blue icon on the left. It contains two checkboxes: 'hide dragger' and 'subsample', both of which are currently unchecked.

Hides the dragger and/or enables subsampling.

### Subsample

A control panel for 'Subsample' with a blue icon on the left. It contains three input fields labeled 'x', 'y', and 'z', each with the value '4'.

The number of voxels to be averaged in each dimension, if subsampling is enabled.

### Timestep

A control panel for 'Timestep' with a blue icon on the left. It features a slider control and an input field on the right showing the value '0'.

If the input object provides time-dependent data, this port allows you to select the time step to be loaded.

## 1.61 LegoSurfaceGen

This module reconstructs a surface from a label field like SurfaceGen, but the surface exactly matches the voxel boundaries. Useful for special purpose applications and for development.

Press the *Apply* button to start the computation.

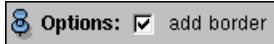
### Connections

**Data** [required]

Connection to input label field.

## Ports

### Options



Same behavior as the corresponding option in the SurfaceGen module.

## 1.62 LineProbe

See Section *Data Probing* for details.

## 1.63 LineSetProbe

This module samples an arbitrary 3D field at the vertices of a *LineSet*. Since line sets can consist of many lines every line of a line set is represented as at least one curve in the plot window according to the dimension of the inspected field resp. to the state of the *Evaluate port* (see below).

The sampled values can also be saved into a copy of the input line set.

## Connections

### Data [required]

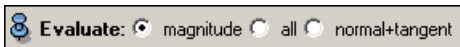
Can be connected to arbitrary 3D fields.

### LineSet [required]

The connected line set contains the probe line(s) where the samples are taken.

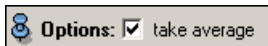
## Ports

### Evaluate



If the probe is connected to a vector field, these radiobuttons are shown. If the *magnitude* button is set the magnitude of the vectors is shown in the plot window. With the *normal+tangent Comp.* button set you get the normal and tangential components as two curves. Setting the *all* button shows all components of the vector field as separate curves.

### Options



The *average* option averages the probe values by taking samples on a disk perpendicular to the sampling points and smoothes the sampled values along the sampling line(s).

## Radius



The radius of the sampling disk. This slider is only shown if the above average option is chosen.

## Longitudinal Width



The width determines how many sampling values are used for smoothing. This slider is only shown if the above average option is chosen.

## Samples



If the *Show* button is pressed a *plot window* appears where the sampled values are plotted against the length of the probe line(s). *Note:* There will be only one plot window regardless of how many LineSet Probe modules there are in your setup. Every lineset probe is represented in that plot window by at least one curve bearing the name of the corresponding module.

Pressing the *Store* button stores the sampled data in a copy of the input lineset.

## 1.64 LineSetView

This module visualizes data objects of type *LineSet*. Individual lines may be displayed in wireframe mode. Alternatively, simple shapes like triangles or squares may be extruded along the lines. In this way true three-dimensional tubes are obtained. Additional features of *LineSetView* are pseudo-coloring and the display of little spheres at each line vertex.

In order to colorize, scale and twist the displayed lines, it is possible to use one of the *data variables*: those are all additional data values of the line set object given per vertex and the connected scalar fields. The latter will be sampled at each line vertex when it comes to displaying it.

## Connections

### Data [required]

The line set to be displayed.

### ROI [optional]

Optional connection to an object providing a region-of-interest. Only line segments inside this region will be visualized.

### ScalarField1 [optional]

Optional connection to a scalar field which can be used to influence coloring, scaling, twisting etc.

### ScalarField2 [optional]

Optional connection to a scalar field which can be used to influence coloring, scaling, twisting etc.

### ScalarField3 [optional]

Optional connection to a scalar field which can be used to influence coloring, scaling, twisting etc.

### Colormap [optional]

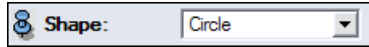
Colormap used for pseudo-coloring.

### Alphamap [optional]

Alphamap used for assigning transparency values.

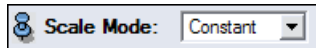
## Ports

### Shape



Determines how the lines will be displayed. If *Lines* is selected a simple wireframe model will be created. The other menu entries denote 2D objects which will be extruded along the lines in order to obtain three-dimensional tubes.

### ScaleMode



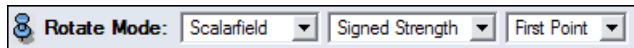
If data variables are available this menu allows you to select one such variable which will be used to scale the diameter of the three-dimensional tubes. If no additional data values are present only *Constant* scaling will be available.

### ScaleFactor



Additional factor used to adjust the diameter of three-dimensional tubes. This factor has no effect if shape has been set to *Lines*.

### RotateMode



Controls the way how the three-dimensional tubes can be rotated around their axis. This can only be adjusted if shape has not been set to *Lines*.

The first menu is set to *No Rotation* by default. Setting it to *Constant* will allow to rotate all lines with the same speed. Following, there will be one entry for each available data variable. Selecting such an entry influences the rotation as defined with the following two menus.

The second menu controls the direction and strength of the rotation. With the third menu one can define at which line point the value shall be evaluated.



## RotateFactor



Changing the value of this slider causes the lines to rotate. Animating this slider gives best results.

## TwistMode



Controls the way how the three-dimensional tubes can be twisted around their axis. This can only be adjusted if shape has not been set to *Lines*.

The first menu is set to *No Twist* by default. Setting it to *Constant* will allow to twist all parts of the line with the same amount. Following, there will be one entry for each available data variable. Selecting such an entry influences the strength of the twist.

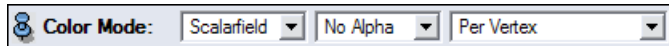
Selecting *Accumulate Twist Values* in the second menu causes the selected twist variable to be accumulated along the line before usage.

## TwistFactor



Changing the value of this slider adjusts the amount of twist.

## ColorMode



If data variables are available this menu allows you to select one such variable which will be used to lookup vertex or segment colors. If *No Color* has been selected the lines or tubes will be displayed in uniform default color. This color may be changed using the command `setLineColor`. The following two menus will only be available if coloring has been chosen.

The second menu controls transparency mapping. Select *No Alpha* for opaque lines. Select *Colormap* to use the alpha values of the colormap. If data variables are present, the following entries refer to the alphamap.

The third menu controls how to assign color values.

If you select *Per Vertex* color mode, additional data values are assigned to each vertex. This will generate a smooth coloring along the line in contrast to the following two options of this menu.

If you select *Per Segment Use First* color mode, each line segment will be assigned the value of its first vertex, and the last vertex value will be unused.

If you select *Per Segment Use Last* color mode, each line segment will be assigned the value of its second vertex, and the first vertex value will be unused.

## Spheres



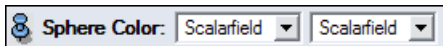
By default *No Spheres* is selected. Changing this selection causes a sphere to be displayed at each line vertex. *Constant* will display the spheres with a constant radius. Following, there will be one entry for each available data variable. Selecting such an entry causes the sphere radii to be scaled according to the selected variable.

### SphereScale



Additional factor used to adjust the size of the spheres. This factor has no effect if *No Spheres* is selected.

### SphereColor



Similar to port *Color Mode*, but affects the sphere colors instead of the line colors.

### Colormap



Optional colormap.

### Alphamap



Optional alphamap.

## Commands

`setLineColor <color>`

Lets you adjust the default line color which is used if color mode is set to *No Color*. The color may be specified as an RGB color triple or as a X11 color name.

`setSphereColor <color>`

Lets you adjust the default sphere color which is used if sphere color mode is set to *No Color*. The color may be specified as an RGB color triple or as a X11 color name.

`setLineWidth <width>`

Lets you adjust the width of the lines when selected shape is *Lines*.

`setStripeColorMapping <mode>`

This has only effect if selected shape is not *Lines*. Three-dimensional tubes can have stripes with different coloring on them. This enhances the perception of twist and rotation. Setting this mode to 1 enables red stripes along the tubes. Setting this mode to 2 maps the color values according to the colormap onto the stripes.

## 1.65 Measuring

This module provides access to two-dimensional and three-dimensional measuring tools. The idea is to click directly into the rendered scene and draw. The 2D tools draw anywhere in the viewer window, whereas the 3D tools work only on geometry displayed in the viewer.

You can access the tools via the *View/Measuring* menu or via the measuring tool button at the top of the viewer. Click on the tiny triangle at the lower right corner of the tool icon to display a pulldown menu from which you can select the desired measuring tool: (2D or 3D line, 2D or 3D angle, and 2D or 3D text). To change the position of a selected tool, click on one of its red handles and drag it to a new location.

Every tool has attributes (visible, locked, render in front, text, color, line width, and font size) that are accessible via the *Tools* port of the *Measuring* object which was created in the Pool.

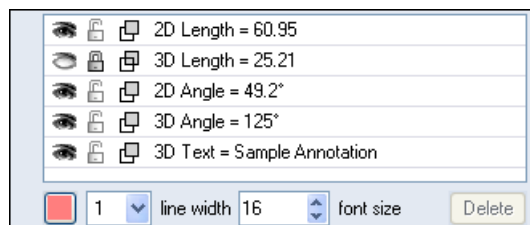
### Ports

#### Add



You can add a *line*, an *angle*, or a *text* string in 2D or 3D.

#### Tools



Each tool is displayed as a row in the toolbox. The columns have the following meaning (from left to right):

- *visible*: Click on the eye icon to toggle visibility of the tool in the viewer. When the eye is empty, the tool is invisible. Click and drag over multiple rows to toggle the visibility of multiple tools.
- *locked*: Click on the padlock icon to lock/unlock the tool. When the padlock is locked, the tool is locked and can't be modified in the viewer. Click and drag over multiple rows to lock/unlock multiple tools.
- *render in front*: Click on the render-in-front icon to toggle whether the tool is always rendered in front of the scene and no longer hidden by objects that are nearer to the camera. It also is always pickable. Click and drag over multiple rows to change the icon for multiple tools.
- *type*: The fourth column indicates the type of the tool (2D length, etc.).

- *text*: The fifth column displays the value of the measured quantity. Double clicking on it allows you to change the display format for numerical values, and the text string for text tools.

If you select a tool in the toolbox, it is highlighted in the viewer with red squares (handles) that you can drag using the mouse. You can select more than one tool at the same time. With the color button, the line width dropdown menu and spin box, and the font size spin box, you can change the appearance of the selected tools. The line width affects the size the of red tool handles as well.

With the *Delete* button you can delete the currently selected tool(s).

## 1.66 Merge

This module works on any 3-dimensional field on rectilinear coordinates and merges the input data by interpolation.

### Merging

Proceed as follows: Attach a *Merge* module to an input field. For additional inputs, right-click on the white square on the left side of the *Merge* module's icon, select an input item, and drag a connection to a data icon. Input fields can have an arbitrary transformation in which case the result field will have an axis aligned that spans the bounding boxes of the input fields. The voxel size of the result field is determined by the first input. At locations within the resulting volume where none of the input fields are defined voxels are initialized with zeros.

For image data, you have the choice among various *interpolation* methods:

- *Nearest Neighbor* chooses for every new voxel the average value of the input voxels nearest to it.
- *Standard* linearly interpolates between the surrounding voxels.
- *Lanczos* is the slowest but most accurate method and approximates a low pass filter. If you have time and want to get the best result, use this one.

For label fields, this port is hidden and always *Nearest Neighbor* is used.

Choose an additional *option* and press the *Apply* button.

### Merging in progress

While merging, the progress bar at the bottom of the *Properties Area* indicates the percentage of merging that is done. You may cancel the merging any time by pressing the stop button on the right of the progress bar.

## Connections

**Data** [required]

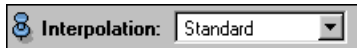
The first data field to be merged.

**Lattice\*** [optional]

Additional input data.

## Ports

### Interpolation



The interpolation method that will be used.

### Options



Option *blend* controls the merging process. If this toggle is disabled, a voxel value of the *first data field* is used if such exists at a given location. Otherwise, a result value will be calculated from the additional input data. If this toggle is enabled, result values are always interpolated.

Use the second option *use existing result* to overwrite an existing result. If *use existing result* is not set, a new result object is generated.

## 1.67 MovieMaker

This module can be used to create an MPEG 1 movie or an animation consisting of a series of 2D images in TIFF, JPEG, or PNG format. The module requires a *time object* as input. When creating the movie, the module iteratively sets the time value, fires the network so that other objects can adjust themselves, and takes a snapshot of the viewer window. The snapshots are appended on-the-fly to the specified MPEG file. In order to create an initial time object that a movie maker module can be attached to, choose *Time* from the main window's *Create* menu.

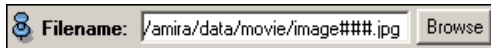
## Connections

**Data** [required]

Connection to a time object. It is not required that the connected time is defined in seconds or in any particular unit. Instead the number of frames and the frame rate are specified (see below). For example, if 100 frames are to be recorded, the connected time is moved from its minimum values to its maximum values in 100 steps.

## Ports

### Filename



The name of the resulting movie or image file(s). In order to write a series of TIFF, JPEG, or PNG files, specify a filename with the suffix *.tif*, *.jpg*, or *.png*. To create an MPEG movie specify a file with the extension *.mpg*. On Microsoft Windows it is also possible to specify *.avi* for creating AVI movies. The filename should contain a series of hash marks *####*. The hash marks will be replaced by the actual frame number. You can also choose the output format via the file dialog which pops up when the *Browse* button is pressed. If no filename is specified, no movie can be created.

### Frames



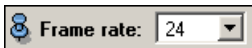
The number of frames to be recorded. This value has nothing to do with the range of the connected time object. If the input time is already defined in seconds and if you don't want any scaling, then the number of frames should be set to the number of seconds of the time object times the desired frame rate.

### Frame rate



Specifies the desired frame rate. It depends on the particular movie player if the specified frame rate is actually achieved when playing the *Movie*.

### Frame rate



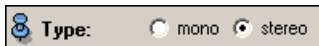
Some movie file formats are only capable of storing movies in specific frame rates. In that case this port is shown with all possible rates of the requested format.

### Compression quality



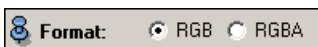
This port is shown if a lossy output file format was chosen. It allows you the adjustment of the degree of compression. Small values indicate high compression and low quality, high values indicate low compression and high quality.

### Type



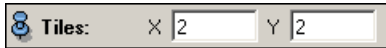
The moviemaker is able to create monoscopic and stereoscopic movies. Set up the movie type here.

### Format



This port specifies the pixel format of the screenshots. If *RGBA* is chosen, the screenshots are generated with an 8-bit alpha channel blending out the scene background. Using the alpha feature has the advantage of smaller image files on complex scene backgrounds (gradient). The second advantage of movies containing an alpha channel is that one can choose another scene background during movie playback. Note that not all image or movie file formats are capable of storing a fourth channel. To create alpha image files choose the suffix *.tif* since TIFF is capable of storing alpha channels. The disadvantages of alpha movies are slower playback speed on some architectures and larger data files on non-complex backgrounds (uniform).

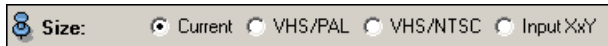
### Tiles



The screenshot shows a control panel with a blue icon on the left and the label "Tiles:". To the right are two input fields: "X" followed by a box containing the number "2", and "Y" followed by a box containing the number "2".

In order to create movies with a higher resolution than the screen, specify the number of tiles to render in each direction. Since the anti-aliasing feature of the former MovieMaker is gone, you can create high resolution single image files and downsample them in a batch to get smooth edges.

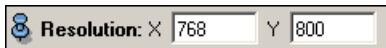
### Size



The screenshot shows a control panel with a blue icon on the left and the label "Size:". To the right are four radio buttons: "Current" (which is selected), "VHS/PAL", "VHS/NTSC", and "Input XxY".

Radio box allowing you to adjust the size of the recorded images. If *current* is specified, the current viewer size adjusted to multiples of 16 pixels will be used. *VHS/PAL* means 720 by 576 pixels. *VHS/NTSC* means 640 by 480 pixels. Finally, if *Input XxY* is specified, the size can be specified explicitly.

### Resolution



The screenshot shows a control panel with a blue icon on the left and the label "Resolution:". To the right are two input fields: "X" followed by a box containing the number "768", and "Y" followed by a box containing the number "800".

This port will only be shown if *Input XxY* is chosen for the *Size* port. It allows you to set the image size of the resulting movie.

### Action



The screenshot shows a control panel with a blue icon on the left and the label "Action:". To the right is a button labeled "Create Movie".

Starts to create the movie. Since the movie is generated from snapshots, the viewer window must not be hidden by other windows while the movie is being created. Furthermore, the *Movie* data with the module connected to its master connection port is altered with the settings of the newly created movie. If there is not such a *Movie* data object connected to the module, a new one is created.

## 1.68 MoviePlayer

This module allows you to play back a sequence of single images, a movie. These images can be stored as separate files in a 2D image format supported by *amira* (for example *JPEG*, *TIFF*, *PNG*, or *PPM*) or they may be packed together in one or more movie data files (movienamename.amovstream). To create movies using *amira*, use the *MovieMaker*.

### 1.68.1 The amira Movie Format

This module is capable of playing back a sequence of images. This sequence is loaded from one or more so called streams. Each stream consists of an image sequence. During playback this module retrieves images from all streams and meshes them together for the final image sequence which gets then rendered to the screen. Imagine there are N streams. The first image comes from the first stream, image 2 from stream 2, image N from stream N, image N+1 again from stream 1, image N+2 from stream 2 and so on. (Expert note: this behavior can be changed by editing the *amira* movie info file). After specifying the streams, the module needs to know what type of movie this image sequence forms, e.g., mono, stereo, stereo interlaced and so on. The structure of an *amira* movie, i.e. the number and specification of streams, the type and the preferred compression type, is stored in an *amira* movie info file (moviename.amov), which is a human readable and editable text file. In *amira*, movies are specified in separate data objects connectable to this module. See *Movie* for details.

### 1.68.2 Optimized amira Movies

This module is capable of converting a movie to an optimized format, i.e. a format that comes up with faster playback speed and sometimes smaller but mostly bigger but fewer data files. To perform this conversion, connect a *Movie* to the input connection. That's the source movie. Then connect a *Movie* to the Result connection port. That will be the destination movie. In the stream specification for that destination movie only *amira* movie data files are allowed. The movie conversion is not able to write into single image files (which would not make sense anyway). After this, select *convert (overwrite)* for overwrite mode or *convert (append)* for append mode. To perform the conversion press one of the play buttons. Seeking does not affect the destination movie, so it is possible to seek to the part of interest in the source movie and start the conversion from there.

### 1.68.3 Which movie or image format should I use ?

To maximize the playback speed:

- Store many images into one or a few big *amira* movie data files by performing the conversion process. This avoids the file searching overhead especially if there are thousands of single image files in a single directory.
- If the target system has limited CPU bandwidth, avoid using CPU-intensive operations like the post compression feature or a CPU-intensive image reader. To find out which image reader is the fastest you have to experiment, since it depends on the individual hardware and software configuration. If the systems *OpenGL* implements the extension *GL\_ARB\_texture\_compression*, you can compress the images using *OpenGL* texture compression. This has the advantage, that the image size is reduced by factor 4 to 6 without the need for decompression by the CPU during playback, since the decompression is performed by the GPU.
- If the target system has limited harddisk bandwidth, compress the data by using high compressing single image formats like JPEG or the post compression feature in combination with the *OpenGL*



texture compression feature if available. If the bandwidth of a single harddisk limits the playback speed, but there are more harddisks available (like on many PCs), store the movie in multiple streams (described above), each of them stored on a different harddisk. On systems with RAID the movie should be stored in a single stream, since the RAID should distribute the movie data file over many disks, which should already increase the retrieval bandwidth. Using more than one stream jams the RAID strategy used for fast retrieval of large disc files, which results in a slowly and bumpy playback.

To maximize the playback image quality:

- Choose a loss-free image format during movie creation outside of this module and also during movie conversion with this module. RGB(A) and gzip postcompression are loss-free. *OpenGL* texture compression is lossy.

#### Known Issues

- When playing single images with readers that are not thread safe, *amira* may crash. In this case one can set the value in the *movies* MaxThreads-port to 1 which sets the maximum number of reader-threads to one, what makes the playback speed slow but allows a conversion to the optimized format. On most platforms the JPG-reader should be thread-safe. The BMP-reader is known to be not thread-safe.

## Connections

### Data

Connect the *Movie* the module is going to play or convert.

## Ports

### Action



Press this button to create a new *Movie* data object connected to this player via the movie's Master port. Filename and type of this newly created movie are initialized according to the properties of the input movie. The default number of streams is one. Alter the settings within the input movie object to match your needs before converting data into that movie.

### Mode



Set this to specify the action the module should perform. If set to *play*, the module will play the movie connected to the data connection port. If set to *play result*, the module will play the *Movie* connected with its *Master* port. If set to *convert (overwrite)*, the module performs the movie conversion from the source *Movie* to the destination *Movie*. The old contents of the destination

*Movie* are deleted first. If set to *convert (append)*, the module appends the contents of the source *Movie* to the destination *Movie*.

### Position



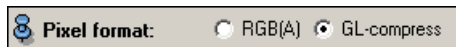
This slider displays the current position in the movie during playback. By pulling the slider it's possible to seek in the movie. With the two upper adjustable sub-range buttons one can limit the movie presentation to a smaller scene of interest. With the two outside buttons one can step through the movie, image by image, making it possible to use this module to present a series of single high resolution slides. By specifying an integer value in the text field one can jump directly to the specified frame. Note that the frame count begins with zero.

### Playback



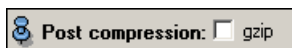
Push these buttons to seek to the beginning of the scene of interest, to its end, to play the movie forward or backward, and to pause or unpause playback.

### Pixel format



This setting gets evaluated during the movie conversion process. If set to *RGB(A)* the images are stored loss-free to the *amira* movie data files, 24 bits per pixel if RGB and 32 bits if RGBA. Note the alpha blending feature of this module which is enabled if the input image contains an alpha channel (*amira* is capable of creating screenshots with an alpha channel). If *GL-compress* is enabled and the systems *OpenGL* is capable of compressing textures, this module compresses the images using *OpenGL*. Note that this kind of compression is lossy. If it is not available on the current system, this option is disabled.

### Post compression



This setting is evaluated during the movie conversion process. Specify this for the destination movie. If enabled, the *MoviePlayer* module compresses the image, independent of the selected pixel format, with *GZIP*. The result is smaller movie data files that need more CPU power during playback. Note that only parts of the movie stream file are going to be compressed, so it is not possible to compress the whole file with an external zip encoder application.

### Chunk size



This setting is evaluated during the movie conversion process. Sometimes the resulting *amira* movie data file gets too large. With this option one can request the module to split this file into

pieces during creation. The fragments are named filename.amovstream, filename.00000001, filename.00000002, ..., filename.0000000N. A value of 0 disables the split feature. A value greater 0 specifies the maximum fragment size in megabytes (each 1,048,576 bytes). One can split an amira movie stream file with external tools into fragments with arbitrary sizes. If the naming scheme matches the one mentioned above, the module will play it correctly. Note that there is an amira movie index file maintained by this module, which holds file numbers and byte offsets for every image on this stream. After manual splits or to force a recreation of this index file, simply remove it. At the next attempt to play or seek this stream, this module will scan the amira movie stream file and recreate the index file.

## 1.69 Normalize Image

A module to normalize volumetric images by computing an approximation to the background and subtracting it from the image. The image is scaled later to the original input data range.

The background image can be computed in different ways dependent on the actual data. The `minimum` option computes the background by using the minimum value in a region of interest defined by the `Filter size` port. This mode is especially useful for images with non-uniform illumination and a larger number of objects. The `maximum` and the `mean` mode compute the corresponding maximum and mean of minimum and maximum operation. The `maximum` is therefore useful if the background contains larger values than the objects.

The `BSpline` mode uses bi-cubic splines to first down-sample the image to the resolution given in the `Filter size` port. This is usually giving a smoother approximation of the background. The modules `minimum`, `maximum`, `mean`, and `BSpline` use a bi-cubic spline to up-sample the low-resolution version back to the original image size before subtracting the background from the data thus correcting for non-uniform illumination.

The `Whitening` mode aims to normalize the data to zero mean and unit variance. This will set the contrast to be close to uniform throughout the image. You should provide the data as either float or double in this case.

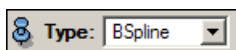
### Connections

**Data** [required]

The data set the correction is performed on.

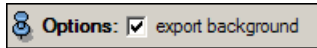
### Ports

**Type**



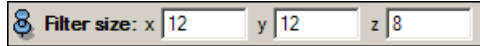
Select a mode for the normalization. Different modes make sense for different images.

## Options



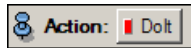
You can export the calculated background image as well.

## Filter size



This is either the resolution of the down-sampled image (for minimum, mean, maximum, and BSpline) or the size of the moving filter that is used for the whitening operation.

## Action



Perform the required operation.

# 1.70 ObliqueSlice

The *ObliqueSlice* module lets you display arbitrarily oriented slices through a 3D scalar field of any type, as well as through an RGBA color field or a 3D multi-channel field. In the medical context such slices are known as multi-planar reconstructions (MPR).

The module is derived from *ArbitraryCut*. See the documentation of the base class for details about how to adjust position and orientation of a slice. Like in the *OrthoSlice* module three different methods are supported to map scalar values to screen colors, namely linear mapping, monochrome mapping based on adaptive histogram equalization, and pseudo-coloring.

## Connections

### Data [required]

The 3D field to be visualized. 3D scalar fields, RGBA color fields, or 3D multi-channel fields are supported. The coordinate type of the input data doesn't matter. The data will be evaluated using the field's native interpolation method (usually trilinear interpolation).

### Colormap [optional]

The colormap used to map data values to colors. This port is ignored when linear or histogram equalized mapping is selected, or when the module is connected to an RGBA color field or to a 3D multi-channel field.

## Ports

### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

### Options



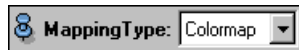
If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when the mouse button is released.

### Translate



This slider allows you to select different slices. The slices may also be picked with the mouse and dragged directly in the 3D viewer.

### Mapping Type



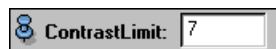
This option menu controls how scalar values are mapped to screen colors. In case of a *linear* mapping a user-defined data window is mapped linearly to black and white. If *hisoram* is selected, an adaptive histogram equalization technique is applied. This method tries to show all features of the data, even if a wide range of values is covered. Finally, *colormap* can be used to activate pseudo-coloring. The port is hidden if the the module is connected to a color field or a multi-channel field.

### Data Window



This port is displayed if linear mapping is selected. It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, while values above the upper bound are mapped to white.

### Contrast Limit



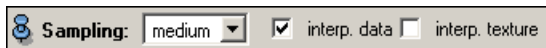
This port is only visible if *histogram* equalization is selected. The number determines the contrast of the resulting image. The higher the value, the more contrast is contained in the resulting image.

### Colormap



This port is displayed if *colormap* is selected. Choose a colormap to map data to colors.

## Sampling



This port controls the way how oblique slices are reconstructed. First, an option menu listing different sampling resolutions is provided. The four choices *coarse*, *medium*, *fine*, and *finest* correspond to an internal resolution of the underlying texture map of 128, 256, 512, and 1024 square pixels, respectively.

The next toggle, denoted *interpolate data*, will only be active if the scalar field to be visualized is defined on a uniform grid. In this case, if the toggle is off nearest neighbor interpolation is used. If it is on or if the data is not defined on a uniform grid, then the field's native interpolation method is used, e.g., trilinear interpolation on regular grids.

Finally, the toggle labeled *interpolate texture* controls how the slice is texture-mapped by the underlying OpenGL-driver. If the toggle is off, nearest-neighbor sampling is used. Otherwise, bilinear filtering is applied.

## Overlay



This port determines how optional *ColorWash* modules are mapped onto this slice.

## Transparency



This radio box port determines the transparency of the slice. The option is only available if the module is connected to an *RGBA color field*. *None* means that the slice is fully opaque. *Binary* means that black parts are fully transparent while other parts are opaque. *Alpha* means that opacity is taken as is.

## Channels



This port is shown if the module is attached to a 3D multi-channel object. It allows you to toggle individual channels on or off. Each channel is mapped using a linear intensity ramp. The data window for each channel can be adjusted in the multi-channel object itself.

## Commands

All commands described for *ArbitraryCut* can be applied to *ObliqueSlice* as well.

`createImage <image base name>`

This command creates a 2D image from the oblique slice and adds it to the Pool. The image base name can be omitted.

## 1.71 OrthoSlice

The *OrthoSlice* module is an important tool for visualizing scalar data fields defined on uniform Cartesian grids, e.g., 3D image volumes. Such data are visualized by extracting an arbitrary axial, frontal, or sagittal slice out of the volume. The data values can be mapped to colors or gray levels by one of three mapping methods. The most simple mapping technique uses a linear gray ramp together with two threshold values. This range determines which data values are mapped to black and which are mapped to white. Alternatively, a contrast limited histogram equalization technique may be applied. With this method, there is no unique correspondence between gray levels and data values any more. The method tries to visualize all features of an image. As a third mapping method, an external colormap can be used in *OrthoSlice*.

The *OrthoSlice* module is also capable of extracting slices of an RGBA color field or of a 3D multi-channel field. In these cases the slices are displayed as is and no mapping method need to be chosen.

### Connections

#### Data [required]

The 3D field to be visualized. Currently, regular scalar fields, multi-channel fields, and RGBA color fields with uniform or stacked coordinates are supported.

#### Colormap [optional]

Optional colormap used to map scalar data to colors. This port is hidden when *linear* or *histogram* mapping is selected. See also *Colormap*.

### Ports

#### Orientation



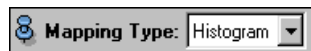
This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

#### Options



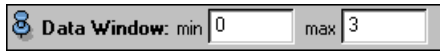
If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. The *bilinear* toggle determines how color is interpolated within a slice. By default, the toggle is off and constant interpolation is used. With constant interpolation individual pixels become visible. Bilinear interpolation often produces somewhat blurry results.

#### Mapping Type



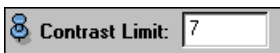
This option menu lets you select between the three different mapping methods, namely a *linear* gray ramp, *histogram* equalization, and *colormap* mode. The port is not available if the *OrthoSlice* module is connected to an RGBA color field or to a 3D multi-channel field.

### Data Window



This port is displayed if *linear* mapping is selected. It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, values above the upper bound are mapped to white. In order to quickly change the data window a *ContrastControl* module can be attached to the *OrthoSlice*.

### Contrast Limit



This port is displayed if *histogram* equalization is selected. The number determines the contrast of the resulting image. The higher the value, the more contrast is contained in the resulting image. A value of zero means that contrast will not be limited at all.

### Colormap



Choose colormap if *mapping type* is set to *colormap*.

### Slice Number



This slider allows you to select different slices. The slices may also be picked with the mouse and dragged directly in the 3D viewer.

### Transparency



This radio box port determines the transparency of the slice. *None* means that the slices are fully opaque. *Binary* means that black parts are fully transparent while other parts are opaque. *Alpha* means that opacity is proportional to luminance. If a colormap is used for visualization opacity values are taken from there.

### Channels



This port is shown if the module is attached to a 3D multi-channel object. It allows you to toggle individual channels on or off. Each channel is mapped using a linear intensity ramp. The data window for each channel can be adjusted in the multi-channel object itself.



## Commands

`frame {0|1}`

This command lets you turn on or off the orange frame indicating the intersection of the plane with the bounding box of the 3D field.

`setFrameColor <color>`

Lets you change the color of the plane's frame.

`setFrameWidth <width>`

Lets you change the width of the plane's frame in pixels.

`createImage <image base name>`

This command creates a 2D image and adds it to the Pool. The image base name can be omitted.

## 1.72 OrthoViewCursor

This module controls two perpendicular *ObliqueSlices* to inspect scalar image data. Two draggers control these slices, one controls the translation, the other one controls the rotation.

## Connections

**Data** [required]

The scalar image data to be visualized.

**Module** [required]

The *OrthoSlice* or *ObliqueSlice* that specifies the dragger plane.

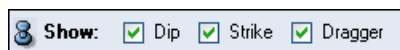
**Colormap** [optional]

Optional colormap used to map scalar data to colors. This port is hidden when linear or histogram mapping is selected. See also Colormap.

**Cylinder slice** [optional]

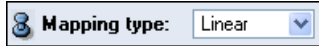
## Ports

**Show**



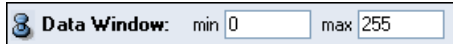
Show or hide the slices or the dragger.

## Mapping type

A control box with a blue icon on the left, the text "Mapping type:", and a dropdown menu showing "Linear".

This option menu lets you select between the three different mapping methods, namely a linear gray ramp, histogram equalization, and colormap mode. The port is not available if the OrthoSlice module is connected to an RGBA color field or to a 3D multi-channel field.

## Data Window

A control box with a blue icon on the left, the text "Data Window:", and two input fields labeled "min" and "max". The "min" field contains the value 0 and the "max" field contains the value 255.

This port is displayed if linear mapping is selected. It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, values above the upper bound are mapped to white. In order to quickly change the data window a ContrastControl module can be attached to the OrthoSlice.

## Contrast limit

A control box with a blue icon on the left, the text "Contrast limit:", and a single input field containing the value 7.

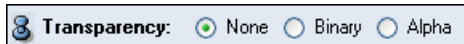
This port is displayed if histogram equalization is selected. The number determines the contrast of the resulting image. The higher the value, the more contrast is contained in the resulting image. A value of zero means that contrast will not be limited at all.

## Colormap

A control box with a blue icon on the left, the text "Colormap:", an input field containing 0, a horizontal color gradient bar, and an input field containing 255.

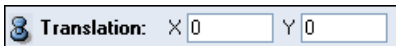
Choose colormap if mapping type is set to colormap.

## Transparency

A control box with a blue icon on the left, the text "Transparency:", and three radio buttons labeled "None", "Binary", and "Alpha". The "None" radio button is selected.

This radio box port determines the transparency of the slice. None means that the slices are fully opaque. Binary means that black parts are fully transparent while other parts are opaque. Alpha means that opacity is proportional to luminance. If a colormap is used for visualization opacity values are taken from there.

## Translation

A control box with a blue icon on the left, the text "Translation:", and two input fields labeled "X" and "Y". Both fields contain the value 0.

The translation of the dragger center with respect to the center of the base plane.

## Rotation

A control box with a blue icon on the left, the text "Rotation:", a slider control, and an input field containing the value 0.

The rotation of the dragger within the base plane.

Additionally, the color-mapping ports of *ObliqueSlice* are provided.

## 1.73 PointProbe

See Section *Data Probing* for details.

## 1.74 PointWrap

This algorithm performs a surface reconstruction from a set of unorganized points. It models a *probe sphere*, that is being 'dropped' onto and then 'rolled over' the set of points. Every three points the sphere rests on during this tour become a triangle in the resulting surface. The result is (almost) guaranteed to be an oriented manifold.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

The input point set.

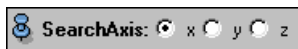
### Ports

#### Probe Radius



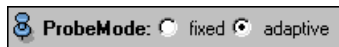
This slider specifies the radius of the probe sphere. It is only relevant if the module is run in the fixed radius mode and when looking for an initial triangle. If the algorithm is unable to find an initial triangle try increasing this value.

#### Search Axis



The direction the probe is initially dropped from to find a starting triangle. If no such triangle can be found, try a different axis.

#### Probe Mode



Here you can choose the probe radius to be fixed throughout the whole computation or to adapt it to local feature size. *Adaptive* probe size is faster than *fixed* probe size and gives better details on point sets that are relatively 'well behaved'. However, it is less robust.

#### Enlargement



If the algorithm is run in *adaptive* probe size mode the local probe size might be too small to find any more triangles. If that happens, the size is enlarged by this factor and the local search is restarted.

### MaxIterations



The iterative process described under *Enlargement* is repeated no more times than the number specified with this slider.

## 1.75 ProbeToLineSet

This module can be connected (tight connection) to a *LineProbe* or *SplineProbe* module. It saves the probe line resp. spline into a *LineSet* data object. The *LineSet* contains all points where samples are taken as coordinates and the sampled values as data. Furthermore the spline controlpoints are saved in the parameters of the *LineSet*. Press the *Apply* button to save the current probe line into the *LineSet*. Since a *LineSet* can hold more than one line additional probe lines will be saved whenever the *Apply* button is pressed.

If a *LineSet* is connected to the data port (optional) this *LineSet* is then copied and the probe lines are saved into the copy.

### Connections

#### **SplineProbe** [required]

This is the controlling *SplineProbe* resp. *LineProbe* module.

#### **Data** [optional]

Can be optionally connected to a *LineSet*. A copy of that *LineSet* is used to save the probe lines.

### Commands

## 1.76 ProjectionView

This module computes a shadow projection of a 3D uniform scalar field (an image volume) onto the three major planes (xy, xz, yz). Color fields and multi-channel fields are also supported. Maximum intensity projection or averaging can be chosen as projection method. This module is especially useful for data containing line-like structures like neurons or angiographic data.

If attached to a 1-component scalar field, the location of voxels containing maximal values can be investigated interactively using the module *Projection View Cursor*.

Press the *Apply* button to recompute the projection.

## Connections

### Data [required]

The dataset to be projected. Uniform scalar fields, uniform color fields, and uniform multi-channel fields are supported.

### ROI [optional]

Connection to a module providing a region-of-interest *SelectRoi*. If such a module is connected, only the volume inside the ROI is projected. Alternatively, this port can be connected to *Cylinder-Slice*, which defines a cylindrical ROI.

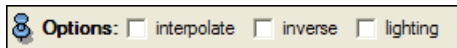
### Colormap [optional]



An optional colormap which is used when the mapping type is set to *colormap* (see below).

## Ports

### Options

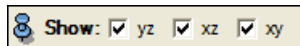


*interpolate*: Bilinearly interpolate between pixels on the projection planes. This in general gives better image quality and less aliasing. However sometimes the resulting image may appear somewhat blurred.

*inverse*: Invert gray values. Inversion is done *after* mapping data values to intensity values via the selected range.

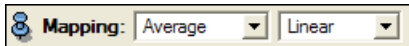
*lighting*: Specifies whether the slices should be illuminated or not. If lighting is on the luminance of the slices changes when the scene is rotated.

### Show



The three toggles can be used to turn off the yz-, xz-, or xy-slice.

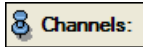
### Mapping



The first menu specifies the projection mode. *Maximum* searches the largest data value in each projection ray, *Minimum* for the smallest one. *Average* computes the average data value for each projection ray. The options *depth* and *depth + max* are only available for scalar fields. If *depth* is chosen on each slice the depth of the pixel containing the maximum value is depicted, instead of the maximum value itself. If *depth + max* is chosen, first an ordinary maximum intensity projection is computed and mapped to a gray image. Then the gray image is multiplied with a pseudo-color image of the depth values. This allows to visualize both projected structures and depth information at once.

The second menu specifies how the projected data should be mapped to color or intensity. Three modes are available, *linear*, *histogram*, and *colormap*. These modes are identical to the ones described for the *OrthoSlice* module. *Colormap* is not available for color fields and for multi-channel fields.

### Channels



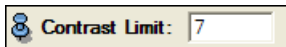
This port is shown if the module is connected to a multi-channel field. It allows you to turn individual channels on or off.

### Range



This port is only available in *linear* mapping mode. All data values smaller than the specified minimum are mapped to black, all values larger than the maximum are mapped to white, and the values in between are mapped linearly.

### Contrast Limit



This port is only visible if *histogram* equalization is selected. The number determines the contrast of the resulting image. The higher the value, the more contrast is contained in the resulting image.

## Commands

`createImage <image base name>`

This command creates 2D images from the projections in the Pool. Between one and three images are created depending on the port *Show*. The image base name can be omitted.

`setTransparency <alpha>`

Set transparency of projections. A value 0 makes it completely opaque, 1 completely transparent.

## 1.77 ProjectionViewCursor

This module can be attached to a *Projection View* module. It visualizes a 3D location on all three planes. The location is specified by picking (i.e. clicking on) any geometry, like an isosurface, in the viewer. If one of the three planes of the *Projection View* itself is picked the module will search for the brightest pixel to determine the missing coordinate. However, this will only work if a scalar field is connected to the *Projection View*, but not for color fields or multi-channel fields.

### Connections

**Module** [required]

This module is attached directly to a *ProjectionView* module.

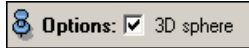
## Ports

### Color



Color of sphere and circles.

### Options



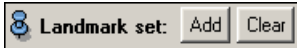
If the 3D sphere option is selected, a sphere is shown at the 3D position of the selected location. Otherwise only the three projections of the sphere are shown.

### Size



Radius of sphere and circles in percent of the bounding box diagonal length.

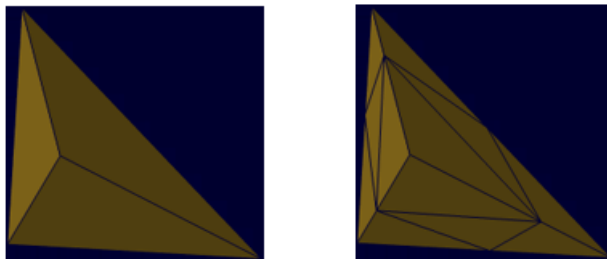
### LandMarkSet



This port can be used to export the current cursor position into a *landmark set data object*. Such an object will be created automatically the first the the *Add* button is pressed. After that you can modify the cursor position and append the new position to the landmark set by pressing the *Add* button again. The *Clear* button removes all landmarks stored in the landmark set data object.

## 1.78 RefineTetras

Connecting *RefineTetras* to a *TetraGrid* allows for a global or local refinement, producing a refined copy of the input grid. For each refinement level a tetrahedron is subdivided by eight tetrahedra via edge bisection.



In case global refinement was chosen the entire tetrahedral grid is successively refined according to the refinement level. The amount of tetrahedra is always increasing by a factor of 8, thus resulting in

huge meshes for higher refinement levels (1, 8, 64, 512, 4096, etc.). Due to the refinement strategy the element quality remains unchanged. Hence the element quality slider will not influence the resulting grid.

Local refinement will be applied to the currently selected (i.e. visible) tetrahedra of the input grid. Therefore a *GridVolume* has to be connected to the grid. In case all tetrahedra are selected *RefineTetras* issues a warning, asking for a global refinement. Having only a subset of the grid selected (e.g. using the Buffer or the Draw Selection), all selected tetrahedra are refined according to the above scheme. In order to keep the mesh consistent adjacent tetrahedra have to be refined as well. The strategy for improving the resulting element quality can be controlled via the slider.

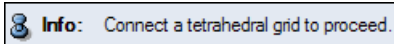
## Connections

**TetraGrid** [required]

The input grid that is to be refined.

## Ports

### Info



This port becomes only visible in case no input grid is connected.

### Refinement Level



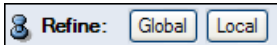
The refinement level (i.e. the amount of iterations).

### Element Quality



Quality measure for local grid refinement ranging from 0 (low quality) to 1 (high quality).

### Refine



Triggers the appropriate refinement strategy.

## 1.79 Relabel

This module sorts the materials in a *LabelField* according to the material list of a template. Materials which are only present in the template are added. If materials are found in the input, which are not present in the template, a warning is issued.



The module is also capable of merging multiple label fields of the same size. This is useful if different parts of the same data set have been segmented independently. If corresponding voxels in the input are assigned different materials, which are both different from Exterior, the last input will be used. If the option *Conflict Material* is checked, such voxels will be assigned to a new material named *MergeConflict*.

This module can be used to fill a new, empty label field with materials from a previous segmentation. Press the *Apply* button to start the computation.

## Connections

**Data** [required]

Input LabelField.

**Data2** [optional]

Additional LabelField to be merged with the first one.

**Data3** [optional]

Additional LabelField to be merged with the first two ones.

**Template** [optional]

LabelField with the template material list.

## Ports

### Options



The **verbose** flag produces some information on the actual mapping. If the **Modify Input** option is chosen, the input data set itself is modified instead of duplicating it, do not use this option on a data set, while an editor is active. If the **Conflict Material** option is chosen in a merging action, voxels with different materials in the input data sets are assigned to a special material called *MergeConflict*.

## 1.80 RelabelTetras

Similar to a *LabelField* each tetrahedron of a *TetraGrid* has a unique material type assigned. Connecting *RelabelTetras* to a *TetraGrid* allows for a global or local reassignment of materials to a copy of the input grid that is automatically generated.

In an interactive mode only materials, that are defined within the parameter section of a *TetraGrid* may be used. In case new materials are to be introduced, they have to be defined with the *ParameterEditor* before connecting *RelabelTetras* to the grid. The assignment of materials operates on the selected (i.e. visible) tetrahedra. Therefore a *GridVolume* has to be connected to the output grid. A subset

of tetrahedra may be chosen using the Buffer or the Draw selection tool of GridVolume. In case all tetrahedra are selected, relabeling leads to a homogeneous material grid.

Another option to relabel tetrahedral grids is to connect a *LabelField* to the second input of *RelabelTetras*. The *BoundingBox* of the TetraGrid and the LabelField must intersect each other. Typically this is the case for tetrahedral grids that have been constructed from label fields via *SurfaceGen*. Otherwise use the *TransformEditor* to align the two. Now the materials of the label field's parameter section are copied to the output grid, and the input field's labels are transferred to the tetrahedra according to their locations. Using a sampling level of one, assigns the material label that is located at the barycenter of a tetrahedron. Choosing a higher sampling level (up to three) assigns the dominating material label with regard to the sample points. A sampling level of 2 takes the material labels at eight inner points. These locations coincide with the barycenters of the eight tetrahedra that will result by a first level refinement using *RefineTetras*. That way up to 512 samples per tetrahedron are evaluated.

## Connections

### **TetraGrid** [required]

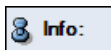
The input grid that is to be relabeled.

### **LabelField** [optional]

A label field.

## Ports

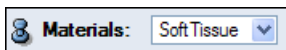
### Info



Info:

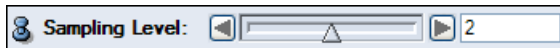
This port becomes only visible in case no input grid is connected, or a label field is connected.

### Materials



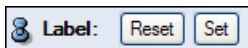
A list of materials that is defined within the parameters of the input grid. This port is only visible in case no label field is connected.

### Sampling Level



The sampling level will choose the number of samples per tetrahedron. This port is only visible in case a label field is connected.

### Label



Triggers the relabeling or resets all labels to the values of the input grid.

## 1.81 Resample

This module works on any 3-dimensional field with regular coordinates, e.g., complex and non-complex scalar or vector fields or RGBA color fields. It lets you *resample* the data, i.e., enlarge or shrink the dimensions of the regular grid while recalculating the data according to it. The first port displayed is *Input Resolution* which indicates the resolution of the incoming data field to be resampled. If the input field has uniform coordinate the *voxelsize* is also displayed. The other ports depend on whether the input field contains ordinary numbers or labels as used in image segmentation (c.f. *LabelField*).

For non-labeled data fields you see the *Filter*, the *Mode*, the *Resolution* port and if the input field has uniform coordinates you will see the *Voxel Size* port. In case of a labeled data field you see the *Average* port. Depending on the existence of labeled data the resampling operation is performed differently.

### 1.81.1 Resampling non-labeled data fields

For non-labeled input data the ports denoted *Filter*, *Mode*, *Resolution* and *Voxelsize* will be shown.

*Filter* provides an option menu allowing you to specify the filter kernel for the resampling operation. Usually the default kernel, *Triangle*, yields sufficiently good results for both minifications and magnifications. The following filters are supported:

- *Box* - simple replication of scalar values, shows considerably tiling or jaggies
- *Triangle* - computationally simple, still sharp transition lines
- *Bell* - smoothing filter
- *B-Spline* - no sharp transitions, but its width causes excessive blurring
- *Lanczos* - excessive "ringing" effect
- *Mitchell* - no sharp transitions, good compromise between "ringing" and "blurring"
- *Minimum* - useful for down-sampling, preserves tiny dark features on a bright background
- *Maximum* - useful for down-sampling, preserves tiny bright features on a dark background
- *Cubic, width 6* - similar to Lanczos
- *Cubic, width 8* - performs similar to Lanczos

The *Minimum* and *Maximum* filters can only be used to downsample non-complex data fields.

Port *Mode* gives you choices how to specify the resolution of the output field. If you select 'dimensions' the port *Resolution* will be enabled to take your input. if you select 'voxel size' port *voxel size* is enabled.

If you connect a second field to the *Reference* connection the behavior will be a little bit different. If you select 'dimensions' the output field will have the same dimensions as the reference field. If you select 'voxel size' the voxel size will be taken from the reference. Additionally the choice 'reference' is enabled which resamples directly onto the lattice that is provided by the reference field.

Press the *Apply* button to start the resampling.

### 1.81.2 Resampling labeled data fields

Labeled data fields can only be down-sampled. Instead of *Filter* and *Resolution* a port denoted *Average* appears. This port allows you to enter the number of cells to average in every dimension. Note, that no enlargement is possible.

As above, the *Apply* button initiates the resampling.

### 1.81.3 Coordinates of the resampled data set

Resampling can be performed on any 3-dimensional field with either uniform, stacked, rectilinear, or curvilinear coordinates. The resampling operation does not change the coordinate type. If you want to convert a data set with stacked, rectilinear, or curvilinear coordinates into one with uniform coordinates you should use the *Arithmetic* module instead of *Resample*. The coordinates of the resampled data set are obtained by a resampling operation on the coordinates of the input data set.

Note that in general the bounding box of the resampled data set will be different from the one of the input data set. In particular, for uniform coordinates the bounding box will extend from the center of the first voxel to the center of the last one.

### 1.81.4 Resampling in progress

While resampling, the progress bar at the bottom of the *Properties Area* indicates the percentage of resampling that is done. You may cancel the resampling calculation any time by pressing the stop button on the right of the progress bar.

## Connections

#### **Data** [required]

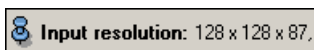
The underlying data field to be resampled.

#### **Reference** [optional]

A reference lattice. It can provide the new resolution, the new voxelsize or the new lattice.

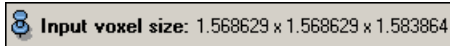
## Ports

#### **Input Resolution**



Displays the resolution of the input data set.

### Input Voxel Size



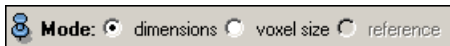
Displays the voxelsize of the input data set (if available).

### Filter



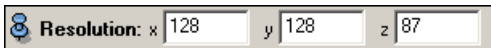
This lets you select one of the resampling filters mentioned above. This will only be visible if the input data set does not contain labels.

### Mode



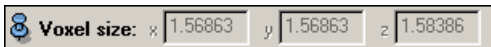
This lets you select whether you want to specify the new resolution or the new voxelsize. If a reference lattice is connected the values are taken from there. Additionally there is the option enabled to sample directly onto the points of the reference lattice.

### Resolution



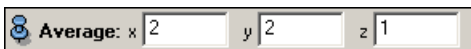
Specifies the resolution of the output data set. This port will only be visible if the input data set does not contain labels.

### Voxelsize



Specifies the voxelsize of the output data set. This port will only be visible if the input data set does not contain labels and has uniform coordinates.

### Average



Specifies how many labels should be averaged during down-sampling. This port will only be visible if the input data set contains labels.

## 1.82 Scale

The Module displays a 2D coordinate System on top of the rendering area. The coordinate is calculated on a plane perpendicular to the viewing direction located at the focal distance of the camera. The focal distance is the point about which rotations are performed. In orthogonal projections the displayed distances are valid for all parts of the picture, in perspective projection only for the mentioned plane. The location and size of the coordinate system can be controlled by the ports. The module scales things to get "nice" numbers (no fractionals). Create an instance with the *Create* menu.

## Ports

### PosX

A slider control with a blue icon on the left, the label "PosX:", a horizontal slider bar with a triangle in the center, and a text box on the right containing the value "0.05".

The position of the origin (fraction of horizontal size of the viewing area).

### PosY

A slider control with a blue icon on the left, the label "PosY:", a horizontal slider bar with a triangle in the center, and a text box on the right containing the value "0.05".

The position of the origin (fraction of vertical size of the viewing area).

### SizeX

A slider control with a blue icon on the left, the label "SizeX:", a horizontal slider bar with a triangle in the center, and a text box on the right containing the value "0.9".

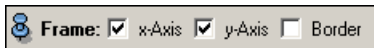
Maximal size of the coordinate system. (fraction of viewport size).

### SizeY

A slider control with a blue icon on the left, the label "SizeY:", a horizontal slider bar with a triangle in the center, and a text box on the right containing the value "0.9".

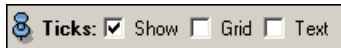
Maximal size of the coordinate system. (fraction of viewport size).

### Frame

A form control with a blue icon on the left, the label "Frame:", and three checkboxes: "x-Axis" (checked), "y-Axis" (checked), and "Border" (unchecked).

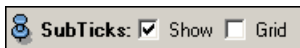
Select the parts of the frame which should be drawn.

### Ticks

A form control with a blue icon on the left, the label "Ticks:", and three checkboxes: "Show" (checked), "Grid" (unchecked), and "Text" (unchecked).

Select whether Ticks, Grid and Text should be drawn.

### SubTicks

A form control with a blue icon on the left, the label "SubTicks:", and two checkboxes: "Show" (checked) and "Grid" (unchecked).

Select whether SubTicks and SubGrid should be drawn.

### Unit

A text control with a blue icon on the left, the label "Unit:", and a text box containing the value "cm".

A unit that is displayed at the axes.

### Color

A color control with a blue icon on the left, the label "Color:", and a small square color selection box.

Color of the coordinate system.

## 1.83 ScanConvertSurface

This module computes a volumetric representation of closed manifold and non-manifold surfaces. The result is a 3D uniform *LabelField*, whose bounding box and dimension can either be specified by filling in the corresponding ports or by a connecting a reference field. Each voxel on the inside of the surface is labeled according to material index of the surface. The user can choose to label all or pick out single materials.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

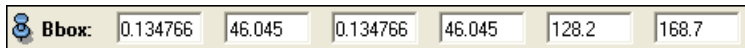
Surface to be converted.

**Field** [optional]

Reference regular 3D field, whose dimensions and bounding box are used to define the resulting field.

### Ports

**Bbox**

A port control for the Bbox parameter. It features a blue icon with a red 'X' on the left, followed by the label 'Bbox:'. To the right are six input fields containing the values: 0.134766, 46.045, 0.134766, 46.045, 128.2, and 168.7.

|       |          |        |          |        |       |       |
|-------|----------|--------|----------|--------|-------|-------|
| Bbox: | 0.134766 | 46.045 | 0.134766 | 46.045 | 128.2 | 168.7 |
|-------|----------|--------|----------|--------|-------|-------|

Bounding box of the resulting field.

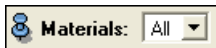
**Dimensions**

A port control for the Dimensions parameter. It features a blue icon with a red 'X' on the left, followed by the label 'Dimensions:'. To the right are three input fields containing the values: 258, 258, and 136.

|             |     |     |     |
|-------------|-----|-----|-----|
| Dimensions: | 258 | 258 | 136 |
|-------------|-----|-----|-----|

Dimensions of the resulting field.

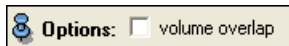
**Materials**

A port control for the Materials parameter. It features a blue icon with a red 'X' on the left, followed by the label 'Materials:'. To the right is a dropdown menu currently showing 'All'.

|            |     |
|------------|-----|
| Materials: | All |
|------------|-----|

Material(s) of the surface to be converted.

**Options**

A port control for the Options parameter. It features a blue icon with a red 'X' on the left, followed by the label 'Options:'. To the right is a checkbox labeled 'volume overlap', which is currently unchecked.

|          |                                         |
|----------|-----------------------------------------|
| Options: | <input type="checkbox"/> volume overlap |
|----------|-----------------------------------------|

If a reference *LabelField* was connected, the volume overlap of the materials of the resulting field with the reference field can optionally be computed. Its value will be written to the console window.

## 1.84 SelectLines

This module extracts a subset of lines or line segments from a given line set object. The lines can be selected by one or more regions of interest *SelectROI*, by one to three materials in a connected label field, or by an expression which is evaluated on the data values of the vertices of the lines.

Press the *Apply* button to start the computation and generate a new line set object.

### Connections

#### Data [required]

The lines set to be filtered.

#### Label field [optional]

A label field with a number of materials. You can select up to three materials from the label field and only lines that connect all three materials will be copied to the output.

#### Region of interest01 [optional]

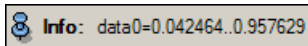
A region of interest. Lines need to go through this region to be copied to the output.

#### Region of interest02 [optional]

A second region of interest. More region of interest ports will be added to the model at run-time as long as new *SelectROI*'s are connected.

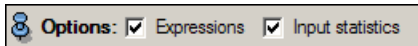
### Ports

#### Info



This port appears if the *statistics* check box is marked. The port presents minimum and maximum values for each data dimension per vertex.

#### Options



Select *Expressions* to make the *Expression* port visible. Select *Input statistics* to make the *Info* port visible. The values displayed in the *Info* port will be evaluated each time, so disabling this option will speed up the computation of the line sets.

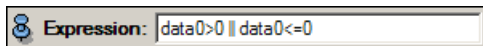
#### Materials



This port is visible if a label field is connected to the module. It allows the user to select up to three materials from the label field. Only lines that touch each material are copied to the output.



## Expression



Enter an expression which is evaluated on the data per vertex of each line. Only vertices for which the expression returns true are copied to the output. This will increase the number of lines as one line of the input may be split into several line fragments of the output.

The variable for each data dimension is named ;dataN<sub>i</sub> where N is the number of dimension starting with 0. An expression like data0<sub>i</sub>0.5 will therefore copy only line segments to the output where the first data value of a point along the line is greater than 0.5.

## 1.85 SelectRoi

This module defines a region-of-interest with the shape of an axis-aligned 3D box. This box can be used to restrict the output of many visualization modules like *Voltex*, *ProjectionView*, or all modules derived from *ViewBase*, e.g., *Isosurface* or *HxSurfaceView*.

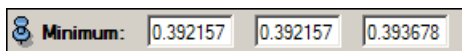
### Connections

#### Data [required]

Connection to a 3D data object which defines the maximum size of the region-of-interest. Only the bounding box of the data object but not the data itself is interpreted by this module.

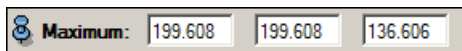
### Ports

#### Minimum



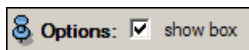
Minimum x-, y-, and z-coordinates of the region-of-interest.

#### Maximum



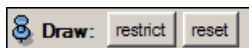
Maximum x-, y-, and z-coordinates of the region-of-interest.

#### Options



If the option *show dragger* is enabled a tab-box dragger is shown which allows you to interactively adjust the region-of-interest in the 3D viewer.

#### Draw



After pressing the *restrict* button you can draw a contour in the viewer window in order to restrict the region-of-interest. However, note that although you can draw an arbitrary contour the region-of-interest itself still remains a box. The *reset* button resets the box so that it covers the full bounding box of the connected data set.

## 1.86 Shear

This module shears a uniform scalar field by shifting each xy-slice in y direction. The shift is proportional to the z-coordinate of the slice, resulting in a shear-operation. The angle between the original and the sheared z-axis can be specified.

Press the *Apply* button to start the computation.

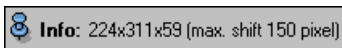
### Connections

**Data** [required]

Connects to uniform scalar fields.

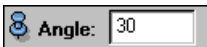
### Ports

#### Info



The output field will be larger than the input field. This port gives the details.

#### Angle



Specifies the angle between the input z-axis and the sheared z-axis. Positive and negative values are allowed.

## 1.87 SmoothSurface

This module smoothes a surface by shifting its vertices. Each vertex is shifted towards the average position of its neighbors. Special care is taken in the case of boundary vertices, for which not all the neighbors are considered, but only those that are also on the boundary. In this way sharp boundaries are preserved.

The smoothing operation is controlled by two tunable parameters: the number of iterations to be performed and a *lambda* coefficient which should be in the range 0...1 and describes the step for each iteration.

Press the *Apply* button to start the smoothing operation. Pressing *Apply* button once again causes the result to be smoothed further. The input is only accessed if no result is present. This behavior is different from most other modules but it allows better interactive control of the smoothing operation.

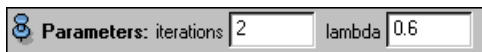
## Connections

**Data** [required]

The surface to be smoothed.

## Ports

### Parameters



A widget with a blue icon and the label "Parameters:". It contains two input fields: "iterations" with the value "2" and "lambda" with the value "0.6".

Contains two parameters: *Iterations* specifies the number of steps for the smoothing, *lambda* is a shifting coefficient which should be in the range (0..1)

### Action



A widget with a blue icon and the label "Action:". It contains a single button labeled "Reset".

Pressing the *Reset* button restores the original surface, i.e., the input is copied to the output.

## 1.88 Sound

- On Microsoft Windows, the underlying multimedia system is used; only WAVE format sound files are supported.
- On X11, the Network Audio System is used if available, otherwise all operations work silently. NAS supports WAVE and AU files.
- On Macintosh, all QuickTime formats are supported.

## Ports

### Sound file



A widget with a blue icon and the label "Sound file:". It contains a text input field and a button labeled "Browse".

Browse the file to be played.

### Sound



A widget with a blue icon and the label "Sound:". It contains two buttons: "Play" and "Stop".

Starts or stops the sound playing. Depending on the platform audio facilities, other sounds may stop

or may be mixed with the new sound. The sound can be played again at any time, possibly mixing or replacing previous plays of the sound.

### Loop



Sets the sound to loop indefinitely.

## 1.89 SplineProbe

See Section *Data Probing* for details.

## 1.90 StandardView

The *StandardView* module displays a 3D image data set or, more precisely, a 3D scalar field with either uniform or stacked coordinates in three different 2D windows at once. The windows show coronal, sagittal, and axial views of the data, respectively. These views correspond to standard xy-, yz, and xy-orientations. Note, that for the axial (xy) view the origin is in the upper left corner. This is the standard orientation used in radiology. In each 2D view the position of the two other slices is indicated by a colored cross hair. You may click at any point in a 2D view in order to reposition the two other slices. The upper left part of the viewer window shows the usual 3D view.

### Connections

#### Data [required]

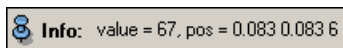
The 3D scalar field to be visualized.

#### OverlayData [optional]

If this port is connected to a second 3D image data set, an overlay of both images is shown in the 2D windows.

### Ports

#### Info



This port gives some information about the value of the 3D scalar field at the point where the three 2D slices intersect, as well as its coordinates.

#### Range



Controls the mapping of input data to gray values. Values below *min* are mapped to black, values above *max* are mapped to white. Values between *min* and *max* are mapped linearly.

### SliceX



Number of slice currently displayed in sagittal (yz) window.

### SliceY



Number of slice currently displayed in coronal (xz) window.

### SliceZ



Number of slice currently displayed in axial (xy) window.

### Zoom



Allows you to decrease, reset, or increase the current image magnification factor. You may also press [Ctrl][Z] to zoom down or [Ctrl][Shift][Z] to zoom up.

### Overlay mode



This port is only visible if a second image dataset has been connected to input port *OverlayData*. You can choose one of four overlay modes: in *blend*, *add*, and *maximum* mode a blending, the sum, or the maximum of both images is shown, respectively. In *checkerboard* mode the 2D windows are divided like a checkerboard showing both image datasets alternately.

### Overlay range



This port is only visible if a second image dataset has been connected to input port *OverlayData*. It controls the mapping of the second image dataset to gray values, similar to port *Range* for the first image dataset.

### Blend factor



This port is only visible if overlay mode *blend* has been selected. It controls the blending of both image datasets. Blend factors 0 and 1 mean that only the first or only the second dataset is shown, respectively. For values between 0 and 1 a linear interpolation is done.

### Pattern size



This port is only visible if overlay mode *checkerboard* has been selected. It controls the size of the checkerboard tiles.

## 1.91 SurfaceArea

This module calculates the area of the individual patches of a surface (*patch mode*). The results are stored in a *spread sheet* data object.

In an alternative mode the area and the enclosed volume of the different regions defined in the surface are computed (*material mode*). In this mode the total surface area is twice as large since every triangle contributes to two regions. Note that for this mode it is required that the surface be closed, i.e., that all regions are completely enclosed by triangles. Otherwise, the computed volumes will be incorrect.

Actually, the volume computation simply sums the signed volumes of all of the tetrahedra joining surface triangles to the origin (point with coordinates  $x=0$ ,  $y=0$ ,  $z=0$ ). The sign of the volumes depends on triangle orientation relative to the origin, so volumes are computed correctly even for non convex surfaces. In the case of a closed surface, the result is the volume of the enclosed surface. If the surface is not closed, then the result can be considered as the volume comprised between the surface (triangles) and the origin, which can be still useful depending on your purpose.

Press the *Apply* button to start the computation.

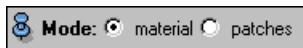
### Connections

**Data** [required]

The surface to be investigated.

### Ports

#### Mode



Toggles between *material mode* and *patch mode*.

In *material mode* the resulting spread sheet object contains one row for every non-empty region of the surface. Each row contains the name of the region, the number of triangles of the region's boundary, the surface area of the boundary, as well as the enclosed volume. Usually, for the *exterior* region the volume is negative.

In *patch mode* the resulting spread sheet object contains one row for every patch of the surface. Each row contains the patch id, the patch's inner region name, the patch's outer region name, the number of triangles of the patch, and the surface area of the patch. In the following columns the

total number of triangles and the total surface area are printed. If the surface data structure also contains the surface contour, the surface perimeter is displayed. Note that you might have to use the **recompute** command of the *Surface* module, to obtain the contour information.

## 1.92 SurfaceCut

The *SurfaceCut* displays a filled cross-section through a surface generated by the *SurfaceGen* module. The surface is supposed to separate different volumetric regions from each other without any holes. Within the cross section the different materials are indicated by their respective colors. If the surface does not form closed loops in the intersection plane these parts will not be shown. The module is derived from *ArbitraryCut* and thus provides the same methods for manipulating the position and orientation of the cross section as this base class. An analog module *GridCut* exists for displaying cross-sections in a tetrahedral finite-element grid.

### Connections

**Data** [required]

The surface to be visualized.

### Ports

#### Orientation



This port provides three buttons to specify the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

#### Options



If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch on the rotate handle for the cutting plane and off again. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

#### Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

## Selection



This port maintains a list of materials to be displayed within the cross section. With the selection menu one can select a single material. The *Add* button adds the currently selected material to the list so that it becomes visible and the *Remove* button removes the material so that it becomes invisible.

## Commands

Inherits all commands of *ArbitraryCut*.

`selectMaterial <id1> [<id2> ...]`

Selects the materials with the specified ids so that intersections of these materials with the cutting plane will be shown. You need to call `fire` before changes take effect.

`unselectMaterial <id1> [<id2> ...]`

Unselects the materials with the specified ids so that intersections of these materials with the cutting plane will not be shown. You need to call `fire` before changes take effect.

## 1.93 SurfaceDistance

This module computes several different distance measures between two triangulated surface. For each vertex of one surface it computes the closest point on the other surface. From the histogram of these values the following measures are computed:

- mean distance
- standard deviation from the mean distance
- root mean square distance
- maximum distance (Hausdorff distance)
- medial distance
- area deviation: percentage of area that deviates more than a given threshold

By using an octree structure the computation is sped up, but may fail, when two surfaces are too different in their shape or location.

Press the *Apply* button to start the computation.

## Connections

**surface1** [required]

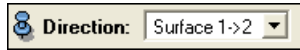
any triangulated surface



**surface2** [required]  
any other triangulated surface

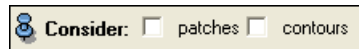
## Ports

### Direction

A control panel for the 'Direction' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Direction:' and a dropdown menu currently showing 'Surface 1->2'.

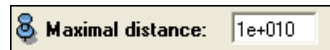
The distance measures are non-symmetric. Thus one can compute them from surface 1- $\rightarrow$ 2 or vice versa 2- $\rightarrow$ 1. One way to get symmetric measures is to join the histograms from both directions into a single histogram (Two-sided option). Note that the two-sided option takes twice as long to compute as their one-sided counterparts.

### Consider

A control panel for the 'Consider' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Consider:' and two checkboxes: 'patches' (checked) and 'contours' (unchecked).

When computing the closest points, one can chose to respect either patches or patches and contours. This means, that closest points from any patch/contour will be restricted to lie on the same patch/contour of the other surface.

### Maximal distance

A control panel for the 'Maximal distance' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Maximal distance:' and a text input field containing '1e+010'.

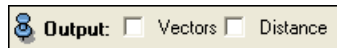
Value for the distance to be used when no closest point could be computed. This can happen when surfaces are too far apart or too different in their shape.

### AboveThreshold

A control panel for the 'AboveThreshold' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'AboveThreshold:' and a text input field containing '0.5'.

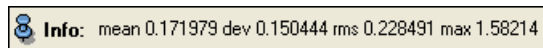
Threshold value for the computation of the area deviation.

### Output

A control panel for the 'Output' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Output:' and two checkboxes: 'Vectors' (unchecked) and 'Distance' (checked).

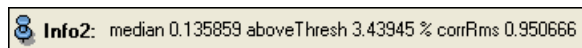
Optional output of the vectors from each point on one surface to their associated closest points on the other surface, or only their magnitude (distance option).

### Info

A control panel for the 'Info' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Info:' and a text input field containing the statistics: 'mean 0.171979 dev 0.150444 rms 0.228491 max 1.58214'.

Resulting distance measures will be displayed here: mean, standard deviation, root mean square and maximum.

### Info2

A control panel for the 'Info2' parameter. It features a blue icon with a circular arrow on the left, followed by the text 'Info2:' and a text input field containing the statistics: 'median 0.135859 aboveThresh 3.43945 % corrRms 0.950666'.

Resulting distance measures will be displayed here: median, area deviation. If the connected surfaces have the same number of nodes. The root mean square distance between two vertices with the same index will be computed as well. This is of interest, when you have computed correspondences by some other method than the closest points computation.

## 1.94 SurfaceField

This module computes a *SurfaceField* from a *Surface* and a *UniformScalarField*. One can choose between encoding on nodes, on triangles, or on triangle nodes.

Press the *Apply* button to start the computation.

### Connections

**Surface** [required]

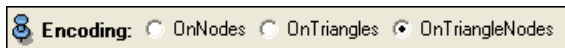
A surface that is the domain of the surface field to be generated.

**Data** [required]

A 3D scalar field.

### Ports

#### Encoding



The encoding defines how the data is stored:

- On the the nodes of the surface.
- On the triangles of the surface.
- Three data values per triangle, one for each node.

## 1.95 SurfaceGen

This module computes a triangular approximation of the interfaces between different tissue types in a *LabelField* with either uniform or stacked coordinates. The resulting surfaces can be non-manifold surfaces. In earlier releases of *amira* this module was called GMC.

Depending on the resolution of the incoming *LabelField* the resulting triangular surface may have a huge amount of triangles. Therefore it is often recommended to start with a downsampled version of the *LabelField*, e.g., with a resolution of 128x128 pixels per slice. During the resampling process the *Resample* module will create or update the probability information of the *LabelField*. This way the

loss of information caused by the resampling process will be minimized. The SurfaceGen module can use the probability information to generate smoother surfaces.

Press the *Apply* button to start the surface extraction. Depending on the size of the LabelField the algorithm requires up to a minute to finish.

## Connections

### Data [required]

LabelField from which the interfaces should be extracted.

## Ports

### Smoothing

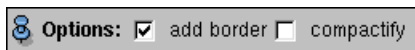


This port controls the way in which the module generates a smooth surface. If set to *none*, no sub-voxel weights are used and the resulting surface will look staircase-like. If set to *existing weights*, then pre-computed weights are used; such weights can be generated with the resample module or the smoothing filter in the image segmentation editor. The options *constrained smoothing* and *unconstrained smoothing* generate sub-voxel weights, such that the surface is naturally smooth; in the *constrained smoothing* mode, the module guarantees that no label be modified: any two voxel centers that have been labeled differently before the smoothing are separated by the generated surface afterwards. This is not necessarily the case for every small detail in the unconstrained case. The amount of smoothing can be controlled via the Tcl interface. Type

```
SurfaceGen setVar SmoothKernelSize <value>
```

into the amira console, to change the default, which is 5 for the unconstrained and 4 for the constrained case. The values currently cannot be larger than 9 but are not limited to integer variables. Note that setting this variable only applies to the actual SurfaceGen module. For further modules SurfaceGen2 etc the command has to be repeated.

### Options



This port provides two toggles.

The first toggle, *add border*, ensures that the resulting surfaces will be closed, even if some regions extend up to the boundary of the LabelField. Closed surfaces are required if a tetrahedral grid is to be generated later on.

If *compactify* is on then a specialized post-processing edge-contraction technique is used to reduce the number of triangles in the resulting surface. See the description of the port *Minimal Edge Length* below for a more detailed explanation.

## Border



This port will only be visible if *add border* has been selected. It provides two toggle buttons labeled *adjust coords* and *extra material*.

If the first option is selected points belonging to triangles adjacent to boundary voxels will be moved exactly onto the nearest boundary face of the bounding box. In this way the resulting surface appears to be sharply cut off at the boundaries.

The second toggle indicates that triangles adjacent to boundary voxels will be inserted into separate patches. The outer region of these patches will be called *Exterior2*. If the toggle is off no such extra material will be created. Instead, the boundary is assumed to be labeled with 0, which usually corresponds to *Exterior*.

## Minimal edge length



A non-vanishing value indicates that short edges of the final surface should be contracted in order to increase triangle quality as well as to decrease the number of triangles. The value of the port indicates the minimal allowed edge length relative to the size of a unit grid cell. By default, values between 0 and 0.8 can be entered. Typically, a value of 0.4 already yields good results. However, note that intersections may be introduced during edge contraction. If you want to avoid this try to use the *simplification editor*. The editor applies some special strategies in order ensure topological consistency.

## 1.96 SurfaceView

This module allows you to visualize triangular surfaces, i.e., data objects of type *Surface*. Derived from the generic *ViewBase* class, the module provides an internal buffer of visible triangles. You can add triangles to this buffer by means of two special option menus. For example, if your surface contains regions *FAT* and *MUSCLE*, you may first highlight all triangles separating these two regions by choosing *FAT* and *MUSCLE* in port *Materials*. Highlighted triangles are displayed in red wire-frame. By pressing button *Add* of port *Buffer* highlighted triangles can be added to the internal buffer, which causes them to be displayed in their own colors. You may restrict highlighting by means of an adjustable box. In order to resize the box pick one of the green handles at the corners of the box. Highlighted triangles may also be removed from the buffer by pressing button *Remove*. In addition, individual triangles may be removed from the buffer by *shift-clicking* them.

## Connections

**Data** [required]

The surface to be visualized.

### ColorField [optional]

In conjunction with a colormap an optional field can be visualized on top of the surface in pseudo-color mode. The field may be either of type *HxScalarField3* or of type *HxSurfaceScalarField*. In addition to scalar surface fields also 3 and 4-component surface fields are supported. In this case, the field components are directly interpreted as RGB or RGBA values. The values should range from 0 to 1.

### Colormap [optional]

The *Colormap* is used to visualize the data values of a scalar field connected to port *ColorField*.

## Ports

### Draw Style



This port is inherited from the *ViewBase* class and therefore the description will be found there.

### Colormap



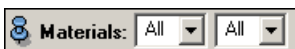
This port becomes visible only if a scalar field has been connected to the *ColorField* port.

### Buffer



This port lets you *add* and *remove* highlighted triangles (being displayed in red wireframe) to an internal buffer. For a further description and for the functionality of each of the port buttons see *ViewBase*.

### Materials



This port provides two option menus listing all regions defined in the surface. By setting the menus properly, you can highlight all triangles separating two different regions. Highlighted triangles are displayed in red wireframe. You may restrict the set of highlighted triangles by means of an adjustable box. Use port *Buffer* to add or remove highlighted triangles to the internal buffer.

### Colors



There are four different color modes:

*Normal:* Each side of a triangle is colored according to the color of the opposite region.

*Mixed:* Both sides of a triangle are colored in the same color, which is a mixture of the colors of the two sides in *normal* mode.

*Twisted:* Each side of a triangle is colored according to the color of the adjacent region.

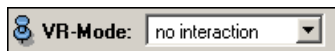
*Boundary id:* Color denotes the boundary ids of the triangles. For each boundary id a separate color can be defined in the surface's parameter section. Boundary ids can be set and removed using the *surface editor*.

### BaseTrans



This port is visible if *transparent* has been chosen as the draw style. It allows you to modify the transparency of the surface. For each material an own transparency value can be defined in the surface's parameter section. If base transparency is 0.5, exactly this default transparency is taken. Larger values make all parts of the surface more opaque, smaller values the surface more transparent.

### VR-Mode



This port is only available in VR mode (see VR Pack documentation). It allows you to choose how to interact with the surface using the 3D wand. In *query* mode information about the clicked point is displayed similar to what is displayed in standard amira when clicking onto a surface with the middle mouse button. The other modes are *select patches*, *highlight patches*, *select triangles*, and *highlight triangles*.

## 1.97 TimeHistoryPlot

This module can be used to display time-history data stored in a Spreadsheet data object. The spreadsheet column "time" is used for x coordinate. This module creates a plot window displaying one curve for each other column. Only columns containing float values are considered. The *Time* port can be used to control a marker line showing the current time step. The plot buffer and curve port can be used to add or remove curves from the plot.

## 1.98 TimeSeriesControl

This module is created automatically if files are imported via the *Open Time Series Data...* option of the main window's *File* menu. The module assumes that all files selected in the file browser represent the same data object at different time steps. Instead of loading all files at once, a slider is provided allowing the user to select the current time step. Whenever a new time step is loaded data, objects associated with a previous time step are replaced. The replacement is performed in such a way that connections to down-stream modules are retained.

The time series module is also able to linearly interpolate between subsequent time steps. However, this only works for certain types of data objects, namely surfaces, tetrahedral grids, hexahedral grids,

fields defined on surfaces, tetrahedral or hexahedral grids, and data objects derived from the vertex set base class. In addition, it is required that for each time step the same number of data objects be created, that corresponding data objects are created in the same order in all time steps, and that corresponding data objects have the same number of vertices or elements. For example, you cannot easily interpolate between surfaces with a different number of triangles (amira provides other modules to support this). If multiple data objects are created for each time step, interpolation of particular objects can be suppressed by switching off the orange viewer toggle of these objects.

In addition to the step number, the module can also display the physical time associated with each time step, provided the physical time is specified as a *Time* parameter for each data object. It is required that the physical times of subsequent time steps be monotonically increasing. In order to use physical time mode, the relevant time steps must be loaded in index mode first.

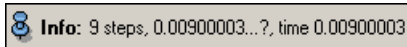
## Connections

**Time** [optional]

Connection to a global time object.

## Ports

### Info



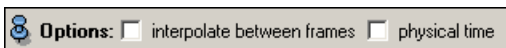
This port displays the total number of time steps. If the data objects provide a physical time in a *Time* parameter, the physical time range is displayed too. If the last time step has not yet been loaded, a question mark is printed instead of the maximum time value. In addition, if a physical time is provided, the current physical time or the current time step index is displayed, depending on the value of the *physical time toggle*.

### Cached steps



Allows you to adjust the cache size. By default every time step is cached, i.e., the number of cached steps is equal to the total number of time steps. If the number of cached steps is zero the cache is disabled. Data objects associated with a previous time step are deleted before a new time step is loaded. Interpolation mode requires a cache size of two, i.e., in addition to the current interpolated time step, at least two additional time steps have to be stored in memory.

### Options



The first toggle is used to activate *interpolation mode*. In this mode, fractional time steps can be specified and a linear interpolation between two subsequent time steps is performed. Note that interpolation can only be performed if certain requirements are met (details are described above).

The second toggle is used to activate *physical time mode*. Physical time mode is only available if the loaded data objects provide a *Time* parameter. In physical time mode the time slider displays the physical time instead of the current time step.

### Time



Specifies the current time step or the current physical time. The port provides a popup menu (right mouse button click) which can be used to configure settings like animation mode or subrange interval. The two outer buttons allow you to automatically animate the time step or the physical time forwards or backwards. Animation speed in physical time mode can be controlled via the increment value in the configure dialog.

## 1.99 TissueStatistics

This module takes a uniform or stacked *label field* as well as an optional scalar field as input and computes some statistical quantities for the regions defined in the label field.

If the quantities are computed per material or per connected component, the different columns of the spreadsheet have the following meaning:

- **Number:** Enumerates all materials (regions) of the label field.
- **Material:** Denotes which material (region) is stored in a row.
- **Count:** Number of voxels contained in a region.
- **Volume:** Number of voxels times size of a single voxel.
- **CenterX:** X-coordinate of the region's center.
- **CenterY:** Y-coordinate of the region's center.
- **CenterZ:** Z-coordinate of the region's center.

If an additional scalar field is connected to the module, then five more columns will be generated:

- **Mean:** Mean value of the field in a particular region.
- **Deviation:** Standard deviation of the field in a region.
- **Min:** Min value of the field in a particular region.
- **Max:** Max value of the field in a particular region.
- **CumulativeSum:** The cumulative sum of the values of the field in a particular region.

The scalar field will be evaluated at the center of each voxel.

Press the *Apply* button to start the computation.



## Connections

### Data [required]

Label field defining the regions.

### Field [optional]

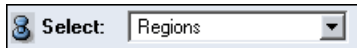
Optional scalar field, for example the image data the label field is associated with. If the scalar field has the same dimensions as the label field, the voxels are accessed directly. Otherwise, a lookup using the field's native interpolation method is performed.

### Voi [optional]

Optional label field used as a volume of interest. It is only used if the statistics are set to *Volume per VOI*.

## Ports

### Select



The quantities can be computed in different modes:

The results are stored in a *spreadsheet* object.

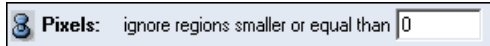
For example, in a medical application you may have two kidneys both assigned to the same material. In the *Material* mode, the two kidneys will be interpreted as one object, in the *Region* mode as two objects.

In mode *Volume per slice* the slices are stored as rows in the spreadsheet. The columns are labeled by the materials and the cells contain the volume of these materials in each slice. The scalar field is ignored. The same output is produced by the *Area per slice* mode. In both cases the voxel size is used to calculate the value. Because of this, the area value corresponds to the volume value when multiplied by the voxel size in the z-direction.

In mode *Volume per VOI* (Volume of Interest) another label field must be attached to the connection named Voi. The rows of the spreadsheet are labeled with the materials in this label field. The columns are labeled with the materials of the main label field. For each voxel of the main label field it is determined to which VOI region the voxel center belongs to. The voxel volume is then added to the particular row of the spreadsheet. If the voxel center is outside of the bounding box of the VOI, it is ignored.

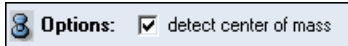
In mode *Polar moment of inertia* the moment of inertia about the z axis  $J_z$  is computed which describes the ability of the per-slice cross section to bend. The larger the moment of inertia, the less the section will bend. Additional values that are computed are the center of mass  $R_{x,y,z}$  and the area- and mass-radius of gyration about the z axis ( $\sqrt{J_z/A}$ ,  $\sqrt{J_z/m}$ ). If the optional scalar field is connected, the real center of mass will be used, otherwise all voxels are assumed to count equally (mass = 1.0). The user may also provide a center of rotation which will be shared for all slices.

### Pixels

A screenshot of the 'Pixels' port control. It features a blue icon with a gear and a lock symbol on the left. To its right is the text 'Pixels: ignore regions smaller or equal than' followed by a text input field containing the number '0'.

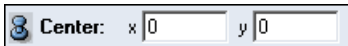
This port will only be visible if *Volume per slice* or *Area per slice* is selected in the *Select* port. It allows you to exclude small regions from the statistics.

### Options

A screenshot of the 'Options' port control. It features a blue icon with a gear and a lock symbol on the left. To its right is the text 'Options: ' followed by a checked checkbox and the text 'detect center of mass'.

This port will only be visible if *Polar moment of inertia* is selected in the *Select* port. It allows you to specify if the center of mass should be automatically detected.

### Center

A screenshot of the 'Center' port control. It features a blue icon with a gear and a lock symbol on the left. To its right is the text 'Center: ' followed by 'x' and a text input field containing '0', then 'y' and another text input field containing '0'.

This port will only be visible if *detect center of mass* is unchecked in the *Options* port. It allows you to specify the center of rotation.

## 1.100 TriangleDistortion

This module computes metric distortions of triangles of two different surfaces, both having with the same connectivity (same number of points/triangles). Metric distortions quantify the amount of deformation of two triangles in terms of angles, areas or lengths. The result output is a surface scalar field.

### Connections

**Data** [required]

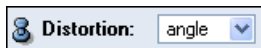
any triangulated surface of type *HxSurface*

**Template** [optional]

any other triangulated surface of type *HxSurface*

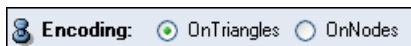
### Ports

#### Distortion

A screenshot of the 'Distortion' port control. It features a blue icon with a gear and a lock symbol on the left. To its right is the text 'Distortion: ' followed by a dropdown menu showing 'angle' and a downward arrow.

Type of distortion measure: choose from angle, area or stretch (= length distortion)

#### Encoding

A screenshot of the 'Encoding' port control. It features a blue icon with a gear and a lock symbol on the left. To its right is the text 'Encoding: ' followed by two radio buttons. The first radio button is selected and is labeled 'OnTriangles'. The second radio button is labeled 'OnNodes'.

Encoding of the output field. Values on nodes are averages over all triangles adjacent to the node.

## Options



Automatically adjusts the colormap used in the connected SurfaceView module. Distortion values can optionally be weighted by the area of the triangle.

## 1.101 VRML-Export

This module enables you to export a *triangular surface* and optionally the 3D dataset from which the surface is derived into a VRML scene. The scene consists of a coordinate system and some animations: the 3D dataset is displayed by three orthoslices. To get an idea how the basic 3D dataset is related to the derived surface you can set the orthogonal slices via sliders, scrolling through the triangular surface. Moreover, the materials of the surface are clickable objects, which means that they can be hidden or shown by a mouse click.

You can publish such a VRML scene directly on the web, just put a link to it into your homepage. To view the file by yourself, make sure that a VRML viewer plugin like *Cosmo Player (version 2.1 or newer)* from SGI is installed for your web browser.

If connected to the SurfaceView module, a special simple export mode becomes available which produces a VRML scene containing only a single IndexedFaceSet node. This is necessary for some 3D printers with limited VRML parsing abilities.

If no surface is connected, only the slices may be exported to VRML.

Press the *Apply* button to start the export.

## Connections

### Data [required]

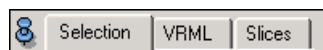
The surface to be inserted into the VRML scene. It can either be an *amira Surface* dataset or the *SurfaceView* module.

### Image [optional]

A scalar field of type *UniformScalarField3* or of type *UniformColorField3* is needed to export the orthoslices. If omitted, no slices are displayed in the VRML scene.

## Ports

### Tabbar



Tab bar to navigate through the user-interface sections. The *Slices* tab remains disabled as long no image data is connected.

## Info

 **Info:** 141776 triangles, 70619 vertices, twosided triangle colors

In simple export mode, the coloring mode and the number of primitives is shown here. (only available in simple mode).

## Selected

 **Selected:** Lobula, Medulla, Lobula-Plate, Exterior2

This port shows the selected materials (not available in simple mode).

## Material

 **Material:**

With this port you can choose a material in order to add or to remove it from the current selection (not available in simple mode).

## Buffer

 **Buffer:**

With this port the selection of materials can be modified. To remove a material from the list, choose this material from the *Material* menu and press *Remove*. Similarly you can use *Add*. *Clear* clears the selection list. Initially, all materials are selected (not available in simple mode).

## Mask

 **Mask:** ☒ All ☐ Selected in SurfaceView

If the simple export mode was chosen, no selection by material is provided. Instead one can use the selection mechanism of the connected *SurfaceView* module. Toggle *Selected* to export only the primitives selected there, i.e., the visible ones (only available in simple mode).

## Simple mode

 **Simple mode:** ☐ (export as a single IndexedFaceSet-Node)

If connected to the *SurfaceView* module, a special simple export mode becomes available which produces a VRML scene containing only a single *IndexedFaceSet* node. This is necessary for some 3D printers with limited VRML parsing abilities.

## Render specular

 **Render specular:** ☐ (set a specular color)

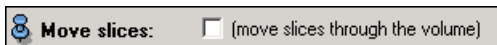
Adding a specular color to the scene which makes the surface more look like plastic.

## Render smooth

 **Render smooth:** ☐ (use vertex normals)

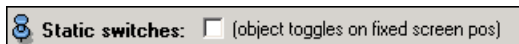
Gives the scene a smoother appearance by using vertex normals.

## Move slices

A control box with a blue icon on the left. It contains the text "Move slices:" followed by an unchecked checkbox and the text "(move slices through the volume)".

If clicked, the slices will actually be moved through the volume when the sliders are operated. Otherwise, only the textures mapped onto the slices are cycled.

## Static switches

A control box with a blue icon on the left. It contains the text "Static switches:" followed by an unchecked checkbox and the text "(object toggles on fixed screen pos)".

If clicked, the small objects used to switch on and off the surface parts will be fixed on the screen. Note that this requires at least *Cosmo Player 2.1* or similar.

## Labels

A control box with a blue icon on the left. It contains the text "Labels:" followed by an unchecked checkbox and the text "(draw object labels)".

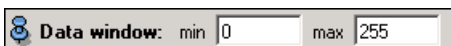
Toggle this for displaying the material names near the material toggles in the VRML scene.

## Export slices

A control box with a blue icon on the left. It contains the text "Export slices:" followed by a checked checkbox.

If toggled, the slice images taken from the image data are exported. If no image data is connected, no slices are exported regardless of this toggle.

## Data window

A control box with a blue icon on the left. It contains the text "Data window:" followed by "min" and a text field with "0", then "max" and a text field with "255".

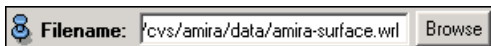
Needed to map data values of input field to gray values. See module *OrthoSlice* for details.

## Slice numbers

A control box with a blue icon on the left. It contains the text "Slice numbers:" followed by "x" and a text field with "10", then "y" and a text field with "10", then "z" and a text field with "10".

Enter the numbers of slices per direction you want to be selectable in the VRML scene. Only available on SGI and when image data is connected.

## Filename

A control box with a blue icon on the left. It contains the text "Filename:" followed by a text field with "/cvs/amira/data/amira-surface.vrl" and a "Browse" button.

Name of the file the VRML code is written to. If export of slices is selected, the slice images are stored to the same directory.

# 1.102 Vertex Morph

This module takes two *Vertex Set* objects as input, e.g. two surfaces or two tetrahedral grids, and computes an output surface by linearly interpolating the vertex positions. This can be used to create a

smooth transition between the two objects. Note, that the two input data sets must be related in order to produce meaningful results. For example, the second input could actually be a copy of the first one with an applied transformation.

## Connections

**Input1** [required]

First input data set. This data set is duplicated to produce the output.

**Input2** [required]

Second input data set, must have at least as many points as the first input.

## Ports

**t**



Interpolation parameter. For  $t=0$  the output will be identical to the first input. For  $t=1$  output will be second input.

## 1.103 VertexDiff

The module computes the displacement field. A vector field on a surface is computed by the difference of the vertex positions of corresponding vertices in both surfaces.

Press the *Apply* button to start the computation.

## Connections

**Data** [required]

Surface 1

**Surface2** [required]

Surface 2

## 1.104 VertexShift

The module displaces the vertices of a surface. The displacements are given by one or more vector fields. The computed vector fields automatically connect to the domain surface.

Press the *Apply* button to start the computation.

## Connections

**Data** [required]

The surface to be displaced

**Vector field** [required]

The displacement field

## Ports

### Info

 **Info:** : Connect a vector field to move vertices.

### Lambda 1

 **Lambda 1:** :  0.371429

For each displacement field, a parameter *lambda* controls the displacement. When the mouse is moved over the slider, an icon indicates which field the slider belongs to.

## 1.105 VertexView

This module allows you to visualize arbitrary *vertex sets*. Vertex sets occur as part of objects of other types, such as *Surfaces*, Tetrahedral Grids, Line Sets, or Molecules. The vertices can be displayed in three different modes: *spheres*, *plates*, and *points*. The vertices may be colored according to a scalar field and a colormap. Alternatively, colors may also be defined via the *command interface* of the *VertexView* module. Furthermore an internal buffer exists that allows you to view only those vertices that are of interest to you.

## Connections

**Data** [required]

The data object from which the vertex set is read.

**ColorField** [optional]

3D scalar field which is used along with a colormap to color the vertices according to the value of the scalar field at the position of a vertex.

**Colormap** [optional]

Used to color the vertices in connection with the *ColorField*.

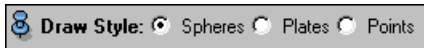
## Ports

### Colormap



Port to select a colormap.

### Draw Style



Vertices may be drawn in three different styles:

- *Spheres*: Points are drawn as triangulated spheres with equal radius.
- *Plates*: Points are drawn as quadrats with mapped-on image of a sphere.
- *Points*: Points are drawn as points. The size of the points does not differ according to the distance of the vertex from the viewer; they all have the same size.

### SphereRadius



Specifies unique radius for all spheres. This port is only visible if *Spheres* or *Plates* is chosen as the draw style.

### Point Size



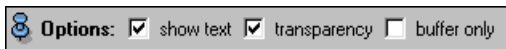
Size of points for draw style *Points*. Only visible in *Points* mode.

### Complexity



Set complexity of displayed spheres. Reducing the complexity leads to coarser spheres and improved rendering performance. If draw style is set to *plates*, the complexity controls the size of the texture maps containing the sphere's images. The smallest texture size is 32x32, the biggest is 512x512. The port is not visible for draw style *points*.

### Options:



A toggle list of options.

- *Show text*: If selected, vertex numbers are drawn. This might be quite slow, especially for large vertex sets. Alternatively, you may select a vertex by clicking on it. Then its number is printed in the console window.



- *Transparency*: Optionally, the spheres and plates may be drawn transparent. Note that the vertices are not sorted along the z-axis. Thus the appearance of the vertex set may look incorrect.
- *Buffer only*: The default is to display all spheres of the vertex set. By clicking this toggle you can display only those vertices that were previously added to the buffer.

### Text Size



Specify text size if *show text* is active.

### Transparency



Allows you to specify the degree of transparency of the spheres.

### Buffer:



A list of buttons for manipulating the internal buffer. In order to actually see the buffer content, activate the *buffer only* option.

- *Add*: adds selected vertices to buffer.
- *Remove*: removes selected vertices from buffer.
- *Clear*: removes all vertices from buffer.
- *Show/Hide Box*: controls box to select vertices.
- *Invert*: exchanges buffer content.

## Commands

`setHighlightColor <red> <green> <blue>`

Set the color that is used to highlight selected spheres.

`setDefaultColor <red> <green> <blue>`

Set default color.

`setColorHighlighted <red> <green> <blue>`

Color all highlighted spheres.

`setColor [<first-vertex-number> [<last-vertex-number>]] <red>  
<green> <blue>`

Set color for all spheres from *first-vertex-number* to *last-vertex-number*. If *last-vertex-number* is omitted, the color is set for the sphere with index *first-vertex-number*. If *first-vertex-number* is omitted too, the color for all spheres is set. *Red, green, and blue* range from 0 to 1.

**highlight** <first-vertex-number> [<last-vertex-number>]

Highlight spheres from *first-vertex-number* to *last-vertex-number*.

**getHighlighted**

Returns a list of all currently highlighted spheres.

**unhighlightAll**

Unhighlight all spheres.

**addToBuffer** <first-vertex-number> [<last-vertex-number>]

Add all spheres ranging from *first-vertex-number* to *last-vertex-number* to buffer. If *last-vertex-number* is omitted, only the sphere with index *first-vertex-number* is added.

**removeFromBuffer** <first-vertex-number> [<last-vertex-number>]

Remove all spheres ranging from *first-vertex-number* to *last-vertex-number* from buffer. If *last-vertex-number* is omitted, only the sphere with index *first-vertex-number* is removed.

**getNumVertices**

Print number of vertices.

**getCoords** <vertex-number>

Print coordinates of vertex with number *vertex-number*

**setTextSize** <size>

Set size with which the text is displayed.

**setTextOffset** <x> <y> <z>

Set offset which is added to the text position.

**setTextColor** <red> <green> <blue>

Set text color.

**updateTextures**

To both *plates* and *spheres* textures are applied to make the spheres look smoother. If the direction from which the light comes changes, those textures need to be recomputed. Updating is invoked by this command only.

**setPickCallback** <proc>

Registers a pick callback. The specified Tcl procedure is called whenever a vertex is selected by clicking on it with the mouse. The procedure is assumed to takes two arguments, the name of the *VertexView* module and the id of the point which has been clicked. To unset the pick callback, call this method without an argument.

## 1.106 ViewBase

This module is the base class for several other *amira* modules displaying a set of triangles, like *Iso-surface*, *SurfaceView*, *GridVolume*, and others. *ViewBase* is not useful on its own but provides special features common to all derived modules. In particular, these features comprise the following:

- A dedicated port allowing the user to modify the *draw style* of a triangular surface in an easy and consistent way. All modules derived from *ViewBase* thus have a similar GUI. Among the supported draw styles is a physically correct transparency mode.
- A generic *buffer* concept allowing the user to select triangles by means of an Open Inventor tab-box dragger. Only triangles added to the internal buffer will be displayed. Thus, complex surfaces may be decomposed into smaller pieces.
- An arbitrary 3D scalar field may be visualized on top of the triangular surface by means of pseudo-coloring. Pseudo-coloring may be achieved via Gouraud shading (mapped vertex colors will be interpolated) or, more accurately, via texture mapping (vertex values will be interpolated linearly and mapped to color afterwards).
- The set of triangles currently being visible can be converted automatically into a *Surface* object. This is useful for example in order to post-process isosurfaces or to extract parts from a bigger surface.
- Common *Tcl commands* allow control of many parameters of modules derived from *ViewBase*. This includes line width, outline color, highlight color, specular color, shininess, alpha mode, normal binding, and more.

## Connections

### Color Field

Arbitrary scalar field to be visualized via pseudo-coloring. The color field will be evaluated at the surface's vertex positions and the vertex color will be set appropriately.

### Colormap

Colormap used for pseudo-coloring. To connect the port to a colormap use the popup menu under the right mouse button. To change the port's default color click it with the left mouse button. See also *Colormap*.

## Ports

### Draw Style



This port determines the draw style of the surface. Five major styles may be selected via an option menu:

*Outlined:* Opaque shaded display with edges superimposed.

*Shaded*: Opaque shaded display without edges being visible.

*Lines*: Shaded wireframe display.

*Points*: Triangle vertices only.

*Transparent*: Semi-transparent display.

Transparent mode implies physically correct transparencies, i.e., triangles appear more opaque if they are viewed under a small angle. It also implies approximate depth-sorting, i.e., triangles are roughly rendered from back to front in order to obtain correct blending results. Note that in some cases visual artifacts may occur for long and thin triangles, for self-intersecting surfaces, or for multiple semi-transparent surfaces being displayed simultaneously.

The drawstyle may be fine-tuned by means of an additional popup menu which is activated by clicking on the button labeled *more options*. This menu contains the following items:

*Shaded*: Enables or disables specular highlights. Specular color and shininess may be changed via the Tcl command interface.

*Gouraud*: Indicates that if a color field is connected pseudo-coloring is performed via Gouraud shading. First, colors are looked up at the triangles' vertices, then these colors are interpolated inside the triangles.

*Texture*: Indicates that if a color field is connected pseudo-coloring is performed via texture mapping. The color field's values are interpolated linearly before color lookup is performed. In this mode no specular colors can be used.

*Opaque*: All triangles will be rendered opaque.

*Const alpha*: Opacity values of a triangle will be taken as is. Usually, if no pseudo-coloring is done, all parts of the surface will have equal opacity.

*Fancy alpha*: Enables physically correct transparencies. The triangle's opacity values will be modified according to their orientation with respect to the viewing direction. Causes the silhouette of the surface to be fully opaque, thus enhancing perception for very transparent surfaces.

*Depth sorting*: Enables approximate depth sorting. The triangle's centers are presorted along the major coordinates axes.

*Create surface*: Creates a *Surface* object containing the set of currently visible triangles.

The following items are only present for a subset of derived modules, e.g., *Isosurface* or *Surface-View*:

*Both faces*: Indicates that triangles are rendered both from back and front.

*Front face*: Enables back face culling. Increases rendering speed but may lead to artifacts for non-closed surfaces.

*Back face*: Enables front face culling. Increases rendering speed but may lead to artifacts for non-closed surfaces.

*Triangle normals*: Enables per-triangle normals. Shading will be discontinuous at the triangles' edges.

*Vertex normals*: Enables per-vertex normals. An average normal is computed for all triangle vertices.

*Direct normals*: An averaged normal is computed for every vertex a triangle. No averaging is performed if two neighboring triangles form an angle bigger than the crease angle set via the Tcl command `setCreaseAngle`. The default is 30 degrees.

## Buffer



This port can be used to modify the list of currently visible triangles. All triangles being visible are stored in an internal buffer. You may add or remove triangles from this buffer via an Open Inventor tab-box dragger. Triangles selected by this dragger will be highlighted, i.e., displayed in red wireframe. If the dragger is not visible, click on one of the buttons to activate it.

*Add*: Adds highlighted triangles to the buffer. If the Shift key is held down while pushing the button the buffer will be cleared before adding. The Shift key is especially useful in conjunction with the *Draw* selection.

*Remove*: Removes highlighted triangles from the buffer.

*Clear*: Removes all triangles from the buffer.

*Show/hide*: Shows or hides the tab-box dragger without modifying the internal buffer.

*Draw*: Activates a lasso style selection mechanism: Using the mouse you can draw a curve in the viewer. All triangles within this curve will be highlighted. If CTRL is pressed while drawing, the triangles within the curve are un-highlighted.

## Commands

`createSurface [name]`

Converts set of visible triangles into a *surface*.

`setAlphaMode {opaque|constant|fancy}`

Triangles may be drawn either opaque or transparent. Two transparent modes are possible: with a constant alpha value (*constant*) or an alpha value varying according to triangle normal (*fancy*).

`setNormalBinding {perTriangle|perVertex}`

Normals can be bound either per triangle or per vertex. In the first mode the triangles appear flat.

`setPointSize size`

Sets the size of *points*.

`setLineWidth width`

Sets the width of *lines*.

`setOutlineColor color`

Sets color of lines in *outlined* mode.

**setHighlightColor** color

Sets wireframe color of selected triangles.

**setEmissiveColor** color

Sets the emissive color of the surface.

**setSpecularColor** color

Sets the specular color of the surface. This will only take effect if specular lighting has been enabled in port *Draw Style*.

**setShininess** shininess

Sets the shininess of the surface.

**showBox**

Shows box that is used for selecting triangles.

**hideBox**

Hides box that is used for selecting triangles.

## 1.107 VolPro-1000

This module is available for Windows only. If you need it on other platforms, contact us at [3d\\_hotline@mc.com](mailto:3d_hotline@mc.com) or [3d\\_hotline\\_europe@mc.com](mailto:3d_hotline_europe@mc.com).

**Note that this module requires special hardware.**

This module provides high quality real time volume rendering by exploiting the TetraRecon VolumePro 1000 board (compare [www.terarecon.com](http://www.terarecon.com)). The volume rendering algorithm is implemented in hardware on the VolumePro board. Hence, the rendering can be performed at interactive frame rates even on low-end machines. For general information about volume rendering see the description of the *Voltex* module.

The *VolPro-1000* module exploits most of the features provided by the VolumePro 1000 system, such as transfer functions, lighting, super sampling, several blend and modulation modes, cropping and cut planes in the three main orientations. Rendering can be combined with other polygonal geometry, as long as no transparent polygons are involved.

Press the *Apply* button to reload the volume to the VolumePro 1000 board(s). This may be necessary after changing some crucial ports. As long there is no need to reload the volume, the button is disabled. The default behavior of this module at its creation time is to load the volume into a single board. This can be stopped by pressing the stop button in the work area beside the progress bar. Then one could change the multiboard parameters and press this button to reload the volume.

### Connections

**Data** [required]

The scalar field to be visualized. The module is also able to handle objects of type *LargeDiskData*.

## Colormap [optional]

The transfer function represented by a colormap.

## Ports

### Tabbar



This port facilitates the navigation in the user interface by grouping the ports.

### Colormap



This port allows you to select one of the predefined colormaps and to specify a range of the scalar values which the colors of the colormap should be mapped to.

### Alpha scale



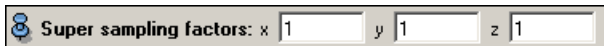
This value scales the opacity of the object.

### Super sampling space



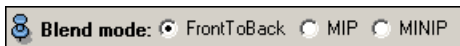
Supersampling is a technique for improving the quality of the rendered image. This port allows you to choose among two supersampling spaces, i.e. camera and object space. Supersampling in object space is especially useful if the distance between the voxels varies strongly along the three main axes. At rendering time object space supersampling factors are transformed into camera space factors. Supersampling in camera space in the x and y directions results in more samples in the base plane. This is done by multipass rendering, which drastically decreases the performance. Supersampling in the z direction results in more slices in the viewing direction, which is supported by the hardware and, hence, hardly influences performance.

### Super sampling factors



This port allows to specify the degree of supersampling in the x, y and z direction.

### Blend mode



Three blending modes are supported:

- *FrontToBack*: Going along the ray from the front of the volume to the back, the final color value is accumulated from all the samples.

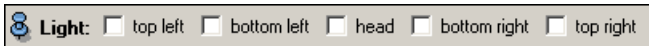
- *MIP (Maximum Intensity Projection)*: Chooses the voxel with the greatest intensity along the ray.
- *MINIP (Minimum Intensity Projection)*: Chooses the voxel with the lowest intensity along the ray.

## Modulation



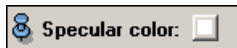
- *GMOM (Gradient Magnitude Opacity Modulation)*: The original opacity is scaled by the magnitude of the sample gradient. Thus only surfaces will be perceived.
- *GMIM (Gradient Magnitude Illumination Modulation)*: Multiplies the computed diffuse or specular color with the magnitude of the sample gradient. As a result the illumination from non-surfaces, which have small gradients, is reduced. GMIM is only effects the image if light is applied to the scene.

## Light



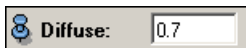
Up to five lights can be applied to the scene at the same time.

## Specular color



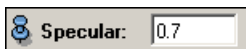
Specify the color of the light(s).

## Diffuse



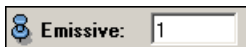
Material coefficient.

## Specular



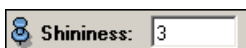
Material coefficient.

## Emissive



Material coefficient.

## Shininess



Material coefficient.

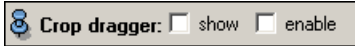


## Light options



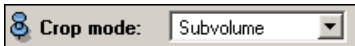
Light intensity.

## Crop dragger



A dragger box can be used to crop parts of the volume. With the *show* toggle you hide or show the box, which can then be scaled and translated in the viewing window. With the *enable* toggle cropping is switched on or off.

## Crop mode

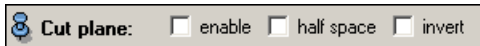


There exist six different crop modes:

- *SubVolume*: Only the volume within the box is shown.
- *3DCross*: The sides of the box are projected to the bounding box and everything within within this 3D cross is displayed.
- *3DFence*: All voxels within the x-, y-, OR z-range of the box are displayed.

The results of these three modes can also be inverted.

## Cut plane



With the help of the cut plane slices of varying thickness can be cut out of the volume. Either the part of the volume defined by this slice or the rest of the volume except this slice might be visualized separately. Apart from slices half spaces can be clipped. The *Cut Plane* port has three toggle buttons, which determine how clipping is done. The first button enables or disables the cut plane. If the second toggle button, *halfspace*, is pushed, not a plane is selected, but a whole halfspace, i.e. the volume on one side of the plane is clipped, the other one is shown. The toggle *invert* inverts clipping, e.g., if it is off and *halfspace* is off too, a slice with a certain thickness is shown. If you then switch the *invert* toggle on, the slice that could formerly be seen is now clipped and the rest of the volume can be seen.

## Slice number



Determines the number of the cut plane slice.

## Slice thickness



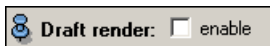
Determines the thickness of the cut plane. Notice, that the thickness grows only into one direction.

## Slice orientation



With this port you can specify the plane which the cut plane should be parallel to.

## Draft render



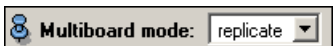
If checked the rendering process gets simplified during object animation. The desired effect is to increase the rendering speed. The loss of quality is the price for the more of interactivity. If the objects stop to transform a maximum quality still picture is rendered.

## Draft render quality



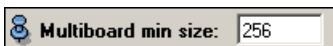
With this slider one can adjust the amount of simplicity or quality loss when draft rendering is enabled. The lower the value the faster but messier the render result. No matter which value was chosen, the depth buffer feature (providing correct occlusion with polygonal objects) is always turned off during draft rendering.

## Multiboard mode



A volume buffer can be replicated or spread over multiple VolumePro 1000 boards. This port can be either set to *single*, *replicate* or *split*. The default is *single* disabling multiboard rendering for that volume. Multiboard mode *replicate* replicates the volume data onto all boards in the system. Each board will render the entire volume for one horizontal band of the output image buffer. This will speed up rendering when the slowest part of the rendering process is the ray casting process, but will slow down volume updates because each update must go to all boards. Multiboard mode *split* splits the volume into pieces and loads one piece onto each board in the system. This allows very large volumes to be rendered, since two boards will be able to support a volume greater than one board. However, the images from each hardware rendering operation are full size images that must be blended together in a separate rendering process; this adds extra overhead for rendering.

## Multiboard min size



Minimum size to enable multiboard capabilities for a volume.

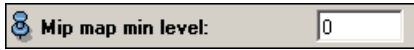
## Multiboard overlap



For multiboard rendering mode *split*, this specifies how much overlap there will be between the split volume data. The minimum is 2, which is fine for non-mipmap volumes. The larger the overlap,

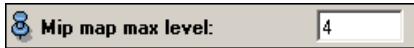
the more mipmap levels can be generated correctly. For example, an overlap of 64 will allow levels 1 through 5 even with the mipmap shrink factor set to 50

### Mip map min level



Range of mipmap levels to use when rendering. Valid values are 0 to 32.

### Mip map max level



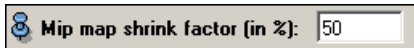
Range of mipmap levels to use when rendering. Valid values are 0 to 32.

### Mip map min volume size



Minimum size to enable mipmap capabilities for a volume.

### Mip map shrink factor (in %)



Shrink factor (in percent) to be used for automatic mipmap level generation. The valid range is 50

## 1.108 Voltex

Direct Volume Rendering is a very intuitive method for visualizing 3D scalar fields. Each point in a data volume is assumed to emit and absorb light. The amount and color of emitted light and the amount of absorption is determined from the scalar data by using a *colormap* which includes alpha values. Default colormaps for volume rendering are provided with the distribution and can be edited using the *colormap editor*. Then the resulting projection from the "shining" data volume is computed.

This module provides you with a hardware accelerated implementation, which uses 2D or 3D texture hardware, to allow for real-time rendering. Note that this currently is not supported by all graphics hardware. Currently hardware acceleration for 2D and 3D textures is available on e.g., SGI Octane, SGI Reality and InfiniteReality, SGI High/Max Impact, HP fx/4, fx/6, and fx/10. The SGI O2 supports 2D texturing. Most PC graphics cards support 2D texture mapping. Older SGI systems, like Indigo2 Extreme, and many Linux drivers currently do not offer hardware texture acceleration. Using this module on the latter platforms can be extremely slow.

Note that on some systems a significant slowdown can occur if the data set is larger than the available texture memory (which typically is 4 - 16 MB).

Press the *Apply* button to start the computations necessary to display the volume. Most parameter changes require pressing this button again.

## Connections

### Data [required]

The 3D scalarfield to be visualized. Alternatively an RGBA data volume (Colorfield) can be connected. In this case no colormap is used, but the color and opacity values are taken directly from the data. As a third mode the module can operate on *multi-channel fields*. Here the transfer function for each channel is computed automatically based on the channels native color, the channels data range, and the value of the Gamma port (see below).

### ROI [optional]

Connection to a module providing a region-of-interest, like *SelectRoi*. If such a module is connected, only the selected part of the volume will be displayed.

### Colormap [optional]

Colormap used to visualize the data.

## Ports

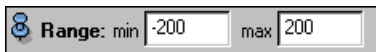
### Options



*mip* stands for *maximum intensity projection*. When this option is selected, the brightest data value along each ray of sight is displayed instead of the result of the emission absorption model described above. This mode is especially useful for very "sparse" data sets, for example: angiographic data or images of neurons.

*color table* enables a transfer function lookup for monochrome data when *RGBA* lookup mode is on. When this option is activated only a quarter of the texture memory is needed for *RGBA* rendering and the color table and its range can be modified in real-time (i.e. without pressing *Apply*). Note that due to incomplete OpenGL implementations some graphics boards which claim to support color tables, they do not. If you see artifacts or only plain white cubes, disable this option.

### Range



This port is only available if the module operates on a 3D scalar field and no colormap is connected. In this case data values are mapped according to this range. Values smaller than the minimum are mapped to completely transparent (no absorption and no emission). Values larger than the maximum appear completely opaque and emit the maximum amount of light. Values in between are mapped proportionally.

### Lookup



Only available if a colormap is connected. In *Alpha* mode, the colormaps alpha value is used for

both absorption and emission. In *LumAlpha* mode, the colormaps alpha value is used for absorption, while the luminance is taken for (uncolored) emission. In *RGBA* mode, colored images are generated by using all four channels of the colormap.

### Colormap



Port to select a colormap.

### Gamma



Controls the shape of the transfer function when multi-channel fields are visualized. The opacity value is taken to be  $\alpha = x^\gamma$ ,  $x = 0 \dots 1$  (proportional to data values). The smaller the gamma value is, the more prominent regions with small data values will be.

### Alpha scale



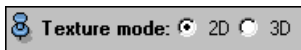
A global factor to change the overall transparency of the object independent of the data value.

### Number of slices



Only available in 3D texture mode. The larger this number, the better the image quality and the less the rendering performance.

### Texture mode



2D texture mode requires some precomputation time but also works on machines which do not support hardware accelerated 3D texturing, e.g., SGI O2. 3D mode needs less setup time and sometimes provides superior quality on high-end machines.

### Downsample



You can specify integer downsample factors to reduce the size of the data set on-the-fly. e.g., downsampling by 2 in each direction would decrease the size of the data set by a factor of 8. This can dramatically improve rendering performance.

## Commands

`showSlices {0|1}`

If set to 1, the slices used to display the volume are drawn outlined instead of full textured. This mode is mainly useful for debugging.

**setInterpol {0|1}**

Enables or disables linear interpolation of texture values. If linear interpolation is disabled a nearest neighbor lookup is performed. By default, linear interpolation is enabled.

**getInterpol**

Checks if linear texture interpolation is enabled or not.

**setColorTableInterpol {0|1}**

Enables or disables linear interpolation within the color table. The setting will be ignored if color table mode is off or if color table rendering is done using paletted textures. The default value is on.

**getColorTableInterpol**

Checks if linear color table interpolation is enabled or not.

**setColorTableMode {0|1|2|3|4}**

Sets the type of OpenGL extension used for color table mode. The modes are encoded as follows:

0 = Don't use any extension, turning off color table mode.

1 = GL\_SGI\_texture\_color\_table.

2 = GL\_EXT\_paletted\_texture.

3 = GL\_NV\_fragment\_program.

4 = GL\_ARB\_fragment\_program.

**getColorTableMode**

Returns the type of OpenGL extension used for color table mode.

**doBricking {0|1}**

Enables or disables bricking in 3D texture mode. If bricking is enabled the volume is rendered in smaller blocks even if it's size exceeds predefined size. The predefined size is by default 0.8 times the assumed texture memory size which depends on the hardware and operating system and is obtained by **amira** on program startup. The texture memory size assumed by **amira** can be modified by setting the environment variable **AMIRA\_TEXMEM** (in megabytes). To alter the predefined size factor of 0.8, one may set a different value in the Tcl variable *voltexTextureAmount*.

**verbose {0|1}**

If value is 1 additional message for debugging are printed.

## 1.109 VolumeEdit

This module provides tools for the interactive modification of 3D image volumes. This is particularly useful for removing noise or undesired objects in a 3D image before applying isosurfaces, volume rendering or other image segmentation tools.

The module takes a scalar field as input and produces a new data set as output which can be modified iteratively. The module does not display any geometry in the viewer. It is typically being used in conjunction with a Voltex or an Isosurface module.

## Connections

### Data [required]

The input dataset to be edited (uniform scalar field).

## Ports

### Tool

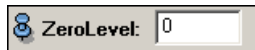


The module provides two different types of tools: a lasso or draw tool and 3D dragger tools.

The lasso or draw tool lets you encircle a specific region in the 3D viewer which then can be cleared in the output data set. Alternatively the part not encircled (exterior) can be cleared (cut away), or the original data values can be restored in the encircled region. In order to use the draw tool, first press one of the action buttons *cut interior*, *cut exterior*, or *restore*. Then draw a line around the specific region in the 3D viewer.

The dragger tools let you specify a region to be modified by dragging, rotating and resizing a 3D shape (box, ellipsoid, cone, or cylinder). A cut or restore operation can then be applied to the interior or exterior part of that shape.

### ZeroLevel



This port specifies by which data value voxels in selected regions are replaced when either *cut interior* or *cut exterior* is pressed.

### Cut



If the button *interior* is pressed, voxels inside the region selected by a dragger shape are replaced by the zero level port. If the button *exterior* is pressed, voxels outside this region are replaced. If the *draw tool* is active, you have to encircle the region to be replaced after pressing one of the buttons.

### Restore



If the button *interior* is pressed, voxels inside the region selected by a dragger shape are replaced by the original data values. If the button *exterior* is pressed, voxels outside this region are replaced. If the *draw tool* is active, you have to encircle the region to be replaced after pressing one of the buttons.

If the button *all* is pressed, the entire volume is reset to its original state.

### Edit



This port provides two buttons for undoing or redoing the last cut or restore operation. The *create mask* button creates a binary label field in which all voxel with a modified data value are set.

## 1.110 VoxelView

This module can be attached to an *OrthoSlice* module. It allows you to visualize contiguous 3D regions of a *LabelField* or of some other uniform scalar field with integer values. The regions are selected by clicking onto the *OrthoSlice* with the middle mouse button. Starting from the selected pixel a 3D flood fill process is performed. Multiple regions can be selected by shift-clicking multiple seeds.

By default the regions to be visualized are taken from the same input object the *OrthoSlice* module is attached to. However, optionally an independent scalar field may be connected to the *VoxelView* module. For example, an *OrthoSlice* module may be used to visualize a stack of CT images, while a *VoxelView* attached to it is used to display segmented regions defined in a label field.

Press the *Apply* button to hide all selected 3D regions.

### Connections

#### Slice [required]

The *OrthoSlice* module which provides the slice where seed points have to be selected using the middle mouse button.

#### Data [optional]

Optional scalar field. If set contiguous regions of this field will be displayed instead of regions of the field the *OrthoSlice* module is attached to.

#### Colormap [optional]

The colormap used to color the 3D regions.

### Ports

#### Colormap



Port to select a colormap.

#### Max Dist



This port limits the region growing process. At most the given number of slices are considered in upward or downward direction. May be useful on slow machines in order to limit the number of triangles.

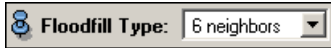


## Draw Style



Three different draw styles are provided, *filled*, *lines*, and *points*. Due to the regular structure of the voxel data only three different face orientations occur. Thus only three different colors will be used to render the voxel regions.

## Floodfill Type



Specifies the kind of flood fill algorithm to be applied (neighbors at faces, faces and edges, or faces, edges, and corners). If *take all* is selected all voxel with the same value as the seed voxel will be selected and displayed.

## Chapter 2

# Molecular Pack

### 2.1 AlignMolecules

This module enables you to align two arbitrary molecules to each other. However, the molecules need to fulfill one precondition: they both must have (at least) one level, the *align level*, with the same name and an equal number of groups. The algorithm works as follows: for each group of the first molecule all its atoms attract all atoms of its corresponding group of the other molecule, whereby the correspondence is established by the index of the group in the *align level*. One possibilities to create such levels is by using the module *AlignSequences*.

The algorithm works iteratively. Starting from an arbitrary position, a force and a rotation are calculated at each step resulting from the attraction forces of the atoms. The algorithm stops if the transformation from one step to another is smaller than some epsilon. The alignment found is not necessarily the global minimum of the sum of distances between the corresponding atoms. However, it will always be a local minimum.

If the global minimum is prominent, it can be assumed that this minimum will be found from each starting position. In order to find out whether there exist other equally good local alignments, it is possible to compute alignments starting from 20 different positions. The different alignments will be stored in a list. You can then compare the alignments visually as well as by the distance between the two molecules.

Press the *Apply* button to compute the alignment and transform the second molecule.

### Connections

**MoleculeA** [required]

First molecule to which the second molecule will be aligned.

### MoleculeB [required]

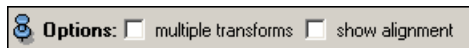
Second molecule to be aligned to the first one.

### Slave [optional]

You can connect any data object to this port. The same transformation that is applied to the second molecule will be applied to this data object as well.

## Ports

### Options



- *multiple transforms*: If this toggle is *not* selected, a local minimum will be found starting from the current positions of the molecules. If the toggle is selected, molecule B will be translated such that the centers of the molecules according to the groups found in the *Align Level* are at the same position. Starting from this position, 20 rotations will be applied to molecule B, and these new positions will be used as starting points for the alignment.
- *show alignment*: If this option is set, you can observe the alignment process visually. Otherwise, only the final alignment will be shown.

### Transforms



If you compute the alignment with the *multiple transforms* toggle switched on, this slider appears. Once the computation is finished, i.e., for all 20 starting positions at which the alignment was computed, you can step through all alignments that differ by an epsilon. The epsilon can be set by the command *setEpsilon*. Notice that if there is only one local alignment, it can be assumed that it is also the global one. However, there is no guarantee.

### Align Level



Select the level which is used for alignment.

## Commands

### getEpsilon

Prints the current epsilon that is used for comparison of the transformations.

### setEpsilon <value>

Set epsilon to value, which should be smaller than 1.0. Check the current epsilon before setting a new one with the command *getEpsilon*.

## 2.2 AlignSequences

This module aligns the sequences of two molecules to each other. Proteins as well as nucleic acids can be aligned. Currently, it is not possible to align t-RNA sequences because they contain modified bases deviating from the standard data format. The module will be adjusted and extended in the near future. The output of the alignment is displayed in a separate window. Currently you may choose between *local*, *semiglobal*, and *global* alignment. The algorithms are based on the Waterman algorithms for pairwise sequence alignments using a *Blosum* weighting matrix for protein sequences and a *Transition/Transversion* weighting matrix for nucleic acids. Consider the following three cases to decide which algorithm to use:

- *local alignment*

The local alignment algorithm is the best algorithm to use if you want to match a short sequence, possibly a subsequence or motif of some molecule, against a long one. You can specify a motif (e.g., a promoter sequence) and search for it in the molecule of interest. Beware, the local alignment algorithm is not useful for aligning two long sequences to each other, e.g., two related proteins. You might end up with a very large number of equally evaluated sequences, fragmenting the molecules in many pieces. For alignment of two long sequences, you should rather use one of the next two algorithms instead.

- *global alignment with gap function*

In this case, you want to match two related long sequences to each other, e.g., two related enzymes. The algorithm matches the whole sequences penalizing long gaps proportionally less than many short gaps.

- *semiglobal alignment*

If you have no idea about the relationship of your two test molecules, it is best to use the semiglobal algorithm first because it will scan for the possibly best matching parts of your molecules without having to align the whole sequences as with the global algorithm.

Press the *Apply* button to compute the alignment and display it.

### Connections

**MoleculeA** [required]

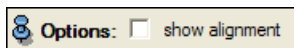
First molecule to be aligned.

**MoleculeB** [required]

Second molecule to be aligned.

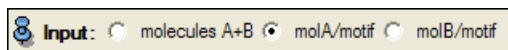
### Ports

#### Options



Toggle the button to show/hide the alignment. This toggle is sensitive if an alignment exists, i.e., if it has previously been computed.

### Input



Which sequences should be taken as input for the alignment. Three options exist:

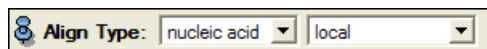
- molecule A and B.
- molecule A and the specified motif (see below).
- molecule B and the motif.

### Motif



You can specify a motif which will be searched for in a longer sequence. This port appears if one of the last two options is chosen in the *Input* port.

### Align Type



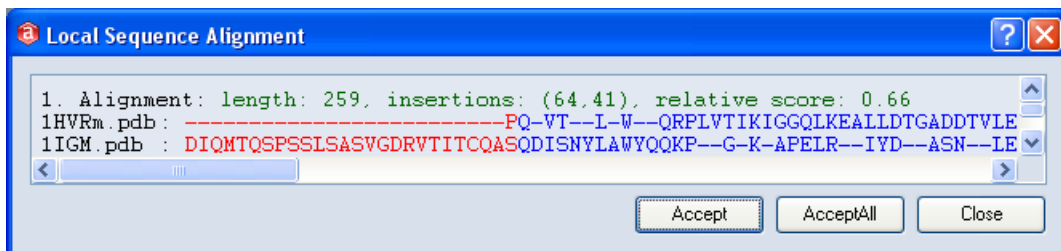
You can align proteins as well as ribonucleic acids. Specify the type of molecule in the first pop-up menu. In the second menu you can select one of three algorithms as explained above: *local*, *semiglobal*, or *global* alignment.

## Commands

`setLimit <value>`

With this command you restrict the number of alignments computed from one start position of the alignment matrix. The default is 10.

## AlignView



For each alignment, its number and additional information are displayed in the first line. Currently the information includes the length of the alignment, the number of insertions for both sequences, and the relative score, i.e., the score divided by the length of the alignment. The latter might be interesting when comparing alignments between homologous sequences.

The *AlignView* displays corresponding residues in blue. To provide an idea about where in the sequences similarities have been found, the rest of the sequence is displayed as well (in red). To find out where in the sequence you are, just click at the position within the sequence you are interested in. Immediately, the positions for both sequences will be shown in the lower-left corner of the window. The clicked position will be marked by a black rectangle.

Apart from just showing alignments, the *AlignSequences* module allows you to add alignments as levels to the aligned molecules. However, since there might be a large number of alignments and you might only be interested in a few, the window allows you to select alignments. In order to select a single alignment, left-click on the row with the label *i. Alignment*. The label will turn black. If you want to select more than one alignment, use the **Ctrl** key. An alignment can be unselected by just clicking on it again. All currently selected alignments are highlighted by a black frame around the label. Those alignments can be written to the molecules by pressing the *Accept* button. If all alignments are desired, press the *AcceptAll* button.

When an alignment is written to a molecule as a new level, all letters of the sequence that are not matched, i.e., that correspond to a blank, are left out. Both molecules will have a new level with exactly the same name and length. Thus, those levels can be used by the *AlignMolecules* module.

## 2.3 BondAngleView

The *BondAngleView* offers an alternative to the *MoleculeView*. This viewing module, however, does not display atoms and bonds but, as the name suggests, bond angles, i.e., for every three atoms connected by two bonds a triangle will be shown. The vertex colors are linearly interpolated across the triangle. Apart from coloring the molecule as is done in the *MoleculeView*, the bond angles can be colored according to a specified color field, e.g., the electrostatic potential, and a colormap. Here, in addition to interpolating the colors, texture mapping can be applied which interpolates the texture coordinates.

### Connections

**Data** [required]

*Molecule* to be visualized.

**ColorField** [optional]

Scalar field that can be displayed on the bond angle by pseudo-coloring.

**Colormap** [optional]

Colormap needed in conjunction with the color field.

### Continuous CM [optional]

Colormap that is used to map values of float attributes to colors.

### Discrete CM [optional]

Colormap that is used to map values of discrete attributes, like integers and strings, to colors.

## Ports

### Draw Style



This port is inherited from the *ViewBase* module.

### Colormap



This port becomes visible only if a scalar field has been connected to the color field port.

### Transparency



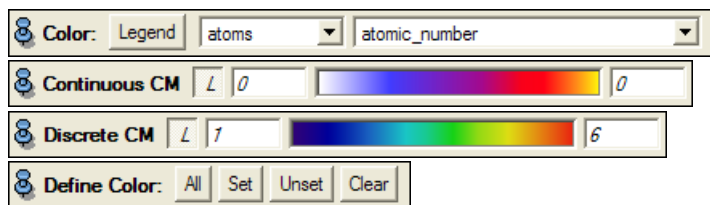
If the draw style is set to *transparent*, this port lets you configure the degree of transparency.

### Color Mode



Allows you to switch between *normal* coloring mode and coloring according to a scalar field.

### Coloring



For an explanation of the four ports above, see the description in the general section on *display of molecules*.

## 2.4 BondCalculation

The module calculates bonds in a given molecular structure by analyzing the distances between the individual atoms of the structure. A maximal distance value will be used to determine whether two atoms are connected or not.

This module is especially useful when being used with a molecule that is derived from a trajectory. Each newly displayed time step will cause the bonds to be recalculated. A similar module for bond calculation is available via the *molecule editor*. This editor module offers advanced functionality but cannot be used dynamically in the work flow of the Pool.

## Connections

**Data** [required]

Molecule for which a new molecule with a complete bond structure should be computed.

## Ports

**Maximal atom distance**



The value determines the maximal distance between two atoms (in Å) below which they are considered to be connected by a bond.

**Action**



Starts the computation of the new molecule.

## 2.5 CompMolInterface

This module generates a distance-based interface between two molecules or between groups of a chosen level of a single molecule. As results of the computation a *MolSurface* and a distance field are generated. Each point on this surface has the same distance to the closest atom of both groups or molecules. Viewing the surface with the *MolSurfaceView* module allows to access additional information contained in the surface (e.g., kind of closest atom or kind of group it belongs to). The distance field can be used to color the surface.

Press the *Apply* button to start the computation.

## Connections

**Molecule A** [required]

Molecule for which interfaces should be computed. In case of the computation of intermolecular interfaces, this is one of the molecules for which the interfaces should be generated.

**Molecule B** [optional]

Second molecule for the computation of the intermolecular interface.



## Ports

### Voxel size

A control port for 'Voxel size' featuring a small blue icon with a question mark, the text 'Voxel size:', a slider bar with a triangle marker, and a numeric input field containing '0.3'.

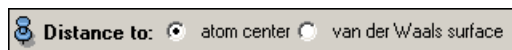
Voxel size of the *distance field*, i.e., measure of discretization. The smaller the voxel size, the longer the processing time.

### Cutoff distance

A control port for 'Cutoff distance' featuring a small blue icon with a question mark, the text 'Cutoff distance:', a slider bar with a triangle marker, and a numeric input field containing '2'.

Largest distance from an atom that should be considered in the algorithm. Points further away than the cutoff distance to any atom will not be considered in the computation.

### Distance to

A control port for 'Distance to' featuring a small blue icon with a question mark, the text 'Distance to:', and two radio button options: 'atom center' (which is selected) and 'van der Waals surface'.

Select the type of interface. Choosing the first option the distance to the van der Waals surface will be considered, otherwise the distance to the atom center, which results in an approximation of the Voronoi diagram.

### Levels

A control port for 'Levels' featuring a small blue icon with a question mark, the text 'Levels', and a dropdown menu currently showing 'atoms'.

This option menu allows you to choose the level of molecular structure for which the intramolecular interfaces should be generated. For example, you might want to compute atomic interfaces on a very low level, or you might just want to generate interfaces between functional groups on a higher level or even between secondary structures. This port disappears if two molecules are connected to the module.

## 2.6 CompMolSurface

This module computes molecular surfaces. Three types of molecular surfaces can be generated: the *van der Waals (vdW) surface*, the *solvent accessible surface (sas)*, and the *solvent excluded surface (ses)*. The *vdW* surface encloses all van der Waals spheres of the molecule's atoms. The *sas* encloses all van der Waals spheres extended by some probe radius. Finally, the *ses* encloses the subspace which is not accessible to a probe sphere in the presence of the molecule represented by its van der Waals spheres.

The implemented algorithm allows the computation of the full molecular surface as well as of partial surfaces. If the partial surface option is selected, the surface will only be computed for all highlighted atoms (see, e.g., the description of the *Molecule Selection Browser*). This module will register itself at the Selection Browser, which allows you to restrict the computation of the molecular surface to a

subset of its atoms. For example, if the molecule consists of several chains, you can compute the molecular surface of a single chain by deselecting the other chains.

Apart from the option to compute either the full or a partial surface, you can also choose between two algorithms that differ in the *quality* of the surface generated, and also in speed. If time does not matter, you should always use the default option, *correct*. However, if you are interested in the dynamic behavior of the molecular surface, time does matter and you might want to use the second algorithm. This algorithm works as follows. We start with an arbitrary atom contributing to the surface. For this we compute the atom's surface contribution. All atoms adjacent to this atom's surface are stored in a list. Next, we take the first atom from the list and deal with it in the same fashion as for the very first atom. Thus, in the case of a *vdW* surface or an *sas* we end up with the same surface as with the *correct* algorithm if the surface consists of one component. In case of the *ses* we run into problems if two components are further away from each other than the probe's diameter. Furthermore, the *faster* algorithm might not compute all cavities, whereas the *correct* algorithm will.

A maximum of two molecules may be connected to the module, thus enabling the surface computation of a complex consisting of two molecules. This can be interesting, for example, if you want to compute the surface before and after a docking.

As a result of the computation a new object, i.e., the molecular surface, will appear in the Pool. To visualize the surface, attach the *MolSurfaceView* module to it.

Press the *Apply* button to start the computation.

## Connections

### Molecule [optional]

The molecule for which the molecular surface should be computed.

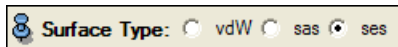
### Molecule2 [optional]

If you want to compute the surface of a complex of two molecules, you must connect this port to a second molecule.

Either of those two ports needs to be connected to some molecule in order to compute a molecular surface.

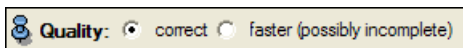
## Ports

### Surface Type

A control panel for Surface Type. It features a blue icon of a person on the left. To its right is the text "Surface Type:". Further right are three radio buttons. The first is labeled "vdW", the second "sas", and the third "ses". The "ses" radio button is selected, indicated by a black dot inside the circle.

Choose between *van der Waals*, *solvent accessible*, and *solvent excluded* surface.

### Quality

A control panel for Quality. It features a blue icon of a person on the left. To its right is the text "Quality:". Further right are two radio buttons. The first is labeled "correct" and is selected, indicated by a black dot inside the circle. The second is labeled "faster (possibly incomplete)".

The default option computes the molecular surface correctly, i.e., with all, possibly disconnected, components and all enclosed cavities. The second algorithm is faster because it does not compute the molecular surface contribution for each atom, but only for surface atoms. If the surface is not connected, it might miss parts of the surface.

### Probe Radius

A control box with a blue icon on the left, the text "Probe Radius:" followed by a slider and a text field containing the value "1.4".

The radius of the solvent probe sphere. This port is only visible for the surface types *sas* and *ses*.

### Edge length

A control box with a blue icon on the left, the text "Edge length:" followed by a slider and a text field containing the value "0.480562".

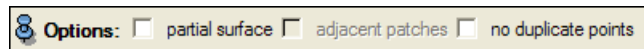
Choose the approximate edge length of the surface's triangles. This port corresponds to the number of points per  $\text{\AA}^2$ , and vice versa. Thus, if you modify the value of this port, the value of the other port will be changed too.

### Number of points

A control box with a blue icon on the left, the text "Number of points per A^2:" followed by a slider and a text field containing the value "5".

With this port you define the approximate number of points per  $\text{\AA}^2$  and thereby the granularity of the surface.

### Options

A control box with a blue icon on the left, the text "Options:" followed by three checkboxes: "partial surface", "adjacent patches", and "no duplicate points".

If the option *partial surface* is selected, only the surface of the currently highlighted atoms will be computed. The second option, which is only active if the first option is checked and the surface type *ses* is selected, initiates the computation of the highlighted atom's adjacent patches, i.e., toroidal and spherical concave patches.

### Surface area

A control box with a blue icon on the left, the text "Surface area:" followed by three checkboxes: "atom area", "residue area", and "specify attribute name".

If either the first or the second option is selected, the exact size of the current molecular surface will be computed and the size per atom or per residue, respectively, will be written to the molecules topology. For the respective level, i.e. *atoms* or *residues*, you will find a new attribute *area*, which contains the contribution of each item to the overall surface area. This allows you to color the molecule according to the surface contribution. It further enables you to select all surface atoms or residues (see *atom expressions* for details). If the last option is selected, a text field will appear allowing you to specify an arbitrary name for the attribute into which the surface areas will be written.

### Attribute name

A control box with a blue icon on the left, the text "Attribute name:" followed by a text field containing the value "area".

Specify an attribute name here, if you do not want to save the surface area in the attribute *area*. In order to do so, the third option of the *Surface area* port must be selected.

## 2.7 ComputeHAtoms

This module returns a new molecule, into which new hydrogen atoms have been added based on geometrical and basic chemical rules. These rules include: distance, connection angle and chemical possible connectivities. Press the *Apply* button to start the computation, returning a new molecule with possibly found hydrogen atoms.

### Connections

**Data** [required]

The molecule for which new hydrogen atoms should be computed

## 2.8 ComputeHBonds

Using various methods for hydrogen bond detection, this module returns a molecule filled with a new level (the hydrogen bond level) and possibly found hydrogen bonds. Distance, angle, atomic number, and other parameters may affect whether or not a hydrogen bond exists between two atoms. Chemistry only allows hydrogen bonds between oxygen and nitrogen atoms that are closer than 3.5 Angstroms to each other. The hydrogen bond calculation represents the base for further computations, such as the secondary structure detection.

Press the *Apply* button to start the calculation of hydrogen bonds.

### Connections

**Molecule** [required]

A molecule for which the hydrogen bond level should be generated.

### Ports

**Minimum Angle**



This angle determines the minimum angle a hydrogen bond must have. The angle of each bond is defined as the angle between the three atoms involved in the bond (alpha carbon - oxygen - nitrogen). The ideal value is about 180 degrees, representing a perfectly aligned hydrogen bond. Realistically this value may range from 120 to 240 degrees.

### Remove Residues of Type

 Remove Residues of Type: ☒ none ☐ H2O ☐ HET ☐ both

Specifies restrictions that may be set for the hydrogen bond calculation. Removing atoms from HET residues and/or water molecules may produce a cleaner result.

### Allow inner-residual HBonds

 Allow inner-residual HBonds: ☒ Yes ☐ No

Toggle to allow inner-residual hydrogen bonds.

### Maximum HBonds per Atom

 Maximum HBonds per Atom:

Specifies the maximum allowable number of hydrogen bonds per atom.

## 2.9 ComputeSecondaryStructure

Using the Kabsch-Sander algorithm to detect secondary structures, this module creates a new molecule filled with generated hydrogen bonds and possibly found secondary structures, such as helices, strands, sheets, and turns. Chains are not detected.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

A molecule for which the secondary structure should be generated.

### Ports

#### Minimum Angle

 Minimum Angle:

This angle determines the minimum angle a hydrogen bond must have. The angle of each bond is defined as the angle between the three atoms involved in the bond (alpha carbon - oxygen - nitrogen). The ideal value is about 180 degrees, representing a perfectly aligned hydrogen bond. Realistically this value may range from 120 to 240 degrees.

#### considered atoms

 Considered Atoms: ☒ backbone-enriched ☐ all

This port allows you to select which atoms will be considered during the computation. Better results may be obtained when only backbone-enriched atoms are used for the computation.

### Write HBonds

A control panel for 'Write HBonds' with a blue icon on the left. It contains three radio buttons: 'all' (unselected), 'secondary structure' (selected), and 'none' (unselected).

This port lets you specify which hydrogen bonds to write out.

### Maximum HBonds per Atom

A control panel for 'Maximum HBonds per Atom' with a blue icon on the left. It features a slider bar and a text input field on the right containing the number '2'.

Specifies the maximum allowable number of hydrogen bonds per atom.

## 2.10 ConfigurationDensity

This module enables you to compute a probability density for the positions of atoms and bonds within a molecular trajectory. The input is an object of type *MolTrajectory*. As output either a scalar field or a color field is generated. In order to visualize the computed density you can, for example, apply the volume rendering module, *Voltex*.

Since the time steps of the molecular trajectory can be arbitrarily rotated and translated, we must perform an alignment for each time step to fit it best to some chosen reference. This is done internally, but you must specify how the molecules should be aligned to each other. There are four options from which to choose. The first uses all atoms for alignment, which is the recommended option. In the second you can select a few atoms (see below for more information). The third and fourth use *none* and *center of gravity* alignment, respectively.

For the representation of the molecule two kinds of geometric objects can be used: spheres and sticks. Here, spheres represent the positions of the atomic nuclei and the sticks the existing bonds within the molecule. You can compute the density for sticks, atoms, or sticks and atoms.

### Connections

#### Data [required]

The molecular trajectory for which the density should be computed.

#### AlignMaster [optional]

The molecule to which each time step of the trajectory will be aligned.

#### PrecomputedAlignment [optional]

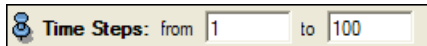
Instead of aligning all time steps of a trajectory to the *AlignMaster* molecule, you can use a pre-computed alignment to align the time steps.

#### Continuous and DiscreteColormap [optional]

These two colormaps are used for the color management of the computed volume, in case a color field should be generated.

## Ports

### Time Steps

A port control box with a blue icon on the left. It contains the text "Time Steps: from" followed by a text input field containing "1", then "to" followed by a text input field containing "100".

Specifies range of time steps for which the density should be computed.

### Voxel Size

A port control box with a blue icon on the left. It contains the text "Voxel Size:" followed by a text input field with a triangle icon and a value of "0.1".

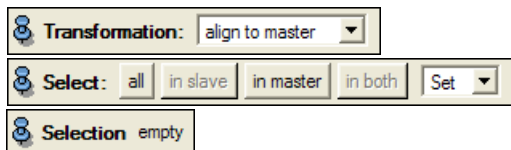
This value determines the size of a voxel, i.e., its height, width, and depth, of the field storing the probability density.

### Grid Dimension

A port control box with a blue icon on the left. It contains the text "Grid Dimension:".

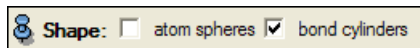
The dimension of the field grid resulting from the specified voxel size.

### Alignment

Three stacked port control boxes. The first has a blue icon and "Transformation:" with a dropdown menu showing "align to master". The second has a blue icon and "Select:" with four buttons: "all", "in slave", "in master", and "in both", followed by a "Set" button and a dropdown arrow. The third has a blue icon and "Selection" followed by the text "empty".

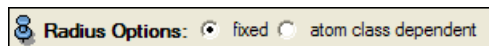
The three ports above specify how the molecules are aligned to each other. See the description in the general section on *alignment of molecules* for a detailed explanation.

### Shape

A port control box with a blue icon on the left. It contains the text "Shape:" followed by two checkboxes: "atom spheres" (unchecked) and "bond cylinders" (checked).

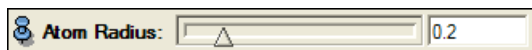
Choose the geometry used to represent the molecule. If you have a highly varying trajectory, the recommended shape is *cylinders*.

### Radius Options

A port control box with a blue icon on the left. It contains the text "Radius Options:" followed by two radio buttons: "fixed" (selected) and "atom class dependent".

Use a unique radius for all atom spheres (*fixed*) or take for each sphere the van der Waals radius scaled by the value of port *Atom Radius* (*atom class dependent*).

### Atom Radius

A port control box with a blue icon on the left. It contains the text "Atom Radius:" followed by a text input field with a triangle icon and a value of "0.2".

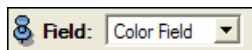
Radius of the atom spheres.

### Bond Radius

A port control box with a blue icon on the left. It contains the text "Bond Radius:" followed by a text input field with a triangle icon and a value of "0.1".

Radius of cylinders representing bonds.

## Field



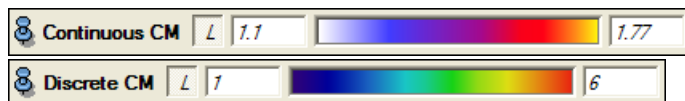
This menu contains two options, *Scalar Field* and *Color Field*. Choose the first if you only want the density. If you choose the second option, in addition to the density, the color information will be stored according to the color scheme (see *Atom Colors* port).

## Atom Colors



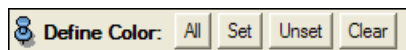
Determines how the field regions will be colored. For more information see the description of *Color* port in the section on displaying molecules.

## Colormaps



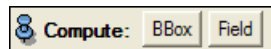
These two colormaps are used to color the atoms and sticks according to the selected color scheme in *Atom Colors*. See the description in the section on *display of molecules* for a detailed explanation.

## Define Color



The port is described in the section on *display of molecules*.

## Compute



Invoke actions.

- Since for the computation of the grid dimension all molecules need to be aligned, for a long trajectory this might take some time. So it may not be desirable to recompute the grid dimension all the time. If you are interested in the size of the grid before invoking the computation of the field, the *BBox* button must be pressed. The bounding box of the field, including all alignments, will be computed and from this, the grid dimension.
- Press the *Field* button to start the computation of the field. If the grid dimension is not up to date, it will be computed first.

## 2.11 CreateSphere

This module enables you to create a triangulated sphere as an object of class *Surface*. Center, radius, and resolution of the sphere can be specified. Furthermore, you can choose among two different ways of triangulation; they are shortly described below.

Press the *Apply* button to start the computation.



## Connections

Data [not used]

## Ports

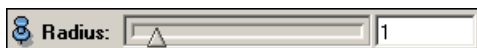
### Sphere Type

A control box for 'Sphere Type' with a blue icon on the left. It contains two radio buttons: 'latitude' (selected) and 'octahedron'.

If the first option, *latitude*, is specified, the triangulation is done by first placing a number of small circles on the sphere. Neighboring small circles are placed equidistantly to each other in terms of their geodesic distance. We then place a number of points on each small circle, such that neighboring points on each circle are again equidistant to each other. The distance between the rings and points on each ring is close to a specified edge length. We then connect neighboring points on each small circle by edges. In the next step, we connect points on neighboring small circles with each other to form a complete triangulation. Here, points are connected such that the edge lengths are as short as possible. Finally, we add points at the north and south pole and connect them to all points on the smallest circles, respectively.

The second option, *octahedron*, triangulates a sphere by starting with an octahedron. The octahedron is then retriangulated to match the given edge length, and the points of this triangulation are projected onto the surface of the sphere. The edge lengths using this triangulation differ much more than with the *latitude* triangulation.

### Radius

A control box for 'Radius' with a blue icon on the left. It contains a text input field with a value of '1'.

Radius of the sphere.

### Point coords

A control box for 'Point coords' with a blue icon on the left. It contains three text input fields labeled 'x', 'y', and 'z', each with a value of '0'.

Center coordinates of the sphere.

### Edge length

A control box for 'Edge length' with a blue icon on the left. It contains a text input field with a value of '0.339809'.

Desired edge length of the triangles.

### Number of points per Unit<sup>2</sup>

A control box for 'Number of points per Unit^2' with a blue icon on the left. It contains a text input field with a value of '10'.

Number of points per unit square. This port influences the edge length, and vice versa.

## 2.12 HBondView

The *HBondView* module allows you to display hydrogen bonds using two different representations: dashed lines or tubes. The display quality of the tubes can be selected by the user.

If the connected molecule does not contain information about hydrogen bonds, the bonds will be computed on the fly. These bonds will not be inserted into the original molecule.

A hydrogen bond is displayed using 5 differently colored parts called "sticks". The middle section represents the hydrogen atom that is being shared by the two atoms that this bond is connecting.

### Connections

#### Data [required]

The molecule whose hydrogen bonds should be displayed.

#### Time [optional]

Time port.

#### Continuous CM [optional]

Optional colormap used for mapping float data to colors.

#### Discrete CM [optional]

Optional colormap used for mapping discrete attributes to colors.

### Ports

#### Type



Selects the display mode: *lines* or *tubes*.

#### Line width



Specifies the width of the lines.

#### Complexity



Specifies the complexity of the tubes. Higher complexity will produce more attractive results, but will take longer to draw.

#### Tube radius



Specifies the radius of the tubes.

## Animation

A control panel for animation. It features a blue icon of a cylinder with a loop, followed by the text "Animation:". To the right are two radio buttons: "no" (which is selected) and "yes".

Animation: ☒ no ☐ yes

Starts an animated view of the hydrogen bond when using the tube display mode. To visualize the sharing of the hydrogen atom between the two atoms, the middle section representing the hydrogen atom will move slowly back and forward in the two directions of the atoms involved in this bond.

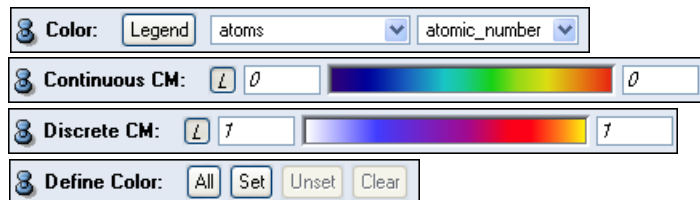
## End caps draw style

A control panel for end caps draw style. It features a blue icon of a cylinder with a loop, followed by the text "End caps draw style:". To the right are three radio buttons: "Flat", "Cone", and "Hemisphere" (which is selected).

End caps draw style: ☐ Flat ☐ Cone ☒ Hemisphere

Selects the way the end caps of the hydrogen bonds should be displayed.

## Coloring

A control panel for coloring. It contains four sections: 1. "Color:" with a "Legend" button, a dropdown menu set to "atoms", and another dropdown menu set to "atomic\_number". 2. "Continuous CM:" with a color bar ranging from blue to red, and input fields for "L" (0) and "T" (0). 3. "Discrete CM:" with a color bar ranging from blue to red, and input fields for "L" (1) and "T" (1). 4. "Define Color:" with four buttons: "All", "Set", "Unset", and "Clear".

Color: Legend atoms atomic\_number

Continuous CM: L 0 T 0

Discrete CM: L 1 T 1

Define Color: All Set Unset Clear

For an explanation of the four ports above, see the description in the general section on *display of molecules*.

## 2.13 MeanMolecule

This module computes the *mean* molecule of a molecular trajectory by averaging the atom positions. In order to do this, the molecules need to be transformed to a common coordinate system. This can be done either by aligning all time steps to some reference molecule, or by computing transformations with the *PrecomputeAlignment* module.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

Molecular trajectory for which the mean molecule should be computed.

### AlignMaster [optional]

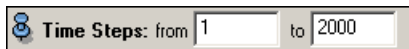
In order to compute the mean molecule, all molecules of the trajectory need to be aligned with respect to a certain molecule. This is done by right-clicking on the white square of the module icon in the *Pool*, selecting *AlignMolecule*, and connecting it to a molecule in the *Pool*.

### PrecomputedAlignment [optional]

Instead of aligning all time steps of a trajectory to the *AlignMaster*, you can use a precomputed alignment to align the time steps.

## Ports

### Time Steps

A port interface for Time Steps. It features a blue robot icon on the left. The text "Time Steps: from" is followed by a text box containing the number "1", then "to" followed by a text box containing the number "2000".

Time Steps: from 1 to 2000

Specify the range of molecules used for the computation.

### Alignment

An alignment port interface. It starts with a blue robot icon and the label "Transformation:". To the right is a dropdown menu showing "align to master". Below this is a "Select:" label followed by four buttons: "all", "in slave", "in master", and "in both". To the right of these buttons is a "Set" button and a small dropdown arrow. At the bottom is a "Selection" label followed by the text "empty".

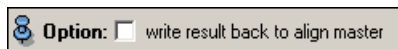
Transformation: align to master

Select: all in slave in master in both Set

Selection empty

The three ports above specify how the molecules are aligned to each other. See the description in the general section on *alignment of molecules* for a detailed explanation.

### Option

An option port interface. It features a blue robot icon on the left. The text "Option:" is followed by an unchecked checkbox and the text "write result back to align master".

Option: ☐ write result back to align master

Select this option if you want to iteratively improve the mean molecule.

## 2.14 Measurement

This module lets you determine distances, angles, and dihedral angles between atoms of the molecule.

To obtain a measurement, you need to select the atoms that are to be examined. In order to select objects in the viewer, you need to switch into the interaction mode of the viewer (for example by pressing the ESC key as a toggle) and **Ctr** L-clicking on the desired atoms. You can read about other ways to select structures in the documentation of the *Selection Browser*.

## Connections

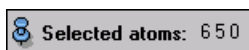
### Module [required]

A module of type *MoleculeView* which can be used for the selection of atoms.

## Ports

The following ports will only be visible if you have selected two atoms for distance measurement, three atoms for angle measurement, or four atoms for measurement of dihedral angles.

### Selected atoms

A "Selected atoms" port interface. It features a blue robot icon on the left. The text "Selected atoms:" is followed by a text box containing the sequence "650".

Selected atoms: 650

This port shows the indices and sequence of the atoms that are used for the measurement.

## Options

 **Options:** ☒ show mean + standard deviation

This port is visible if your base molecule is derived from a trajectory. If you activate the toggle, not only the measurement of the currently active time step, but also the mean value of the complete trajectory and its standard deviation will be evaluated. For large trajectories this may take some time as all atomic coordinates have to be loaded for each time step. So if you do not need this option, leave it unchecked.

## Angle

 **Angle:** 106.559°

This port shows the value of the measurement. For two selected atoms, this is the distance; for three, the angle; and for four atoms, their dihedral angle. (The dihedral angle is defined as the angle between the two planes determined by the first three selected atoms and the last three selected atoms). The distance is given in angstroms, the angles in degrees.

## Time-Mean

 **Time-Mean:** 113.055 ± 6.92911°

This port is visible if your base molecule is derived from a trajectory and the *Time-mean* toggle is activated. It will show the mean value of the measurement by determining the arithmetic mean over all molecules that are part of the trajectory. In addition, the standard deviation of the time-mean will be shown.

## Time Plot

 **Time Plot:** 

Like the time-mean port, this port is only visible for trajectories. By pressing the *Show* button you can see the plot of the measurement against the time steps of the trajectory. The vertical line of the plot indicates the position of the currently shown time step. The horizontal lines show the mean and the standard deviation.

## Selection

 **Selection:** 

With the *Clear* button you can deselect all atoms.

## 2.15 MolElectrostatics

This module calculates the electric or potential field around a molecule. Both potential and E-field can be evaluated both on a uniform grid or on a surface. The surface needs to be supplied as a data connection, while the grid will be computed in a spatial region which may be defined by a tab box.

To be able to compute the fields, the atom level of the molecule must contain a float attribute of name *charge* which must contain the atomic partial charges. Additionally a surface field may be supplied as a data connection, which contains surface polarization charges.

When choosing the voxel size, the resulting dimensions of the field will be shown. Keep in mind that the dimension does not only affect the memory usage but also the computation time, which might be excessively large if the molecule contains many charges. While the number of grid points of the field is  $\text{dim.x} * \text{dim.y} * \text{dim.z}$ , the number of function evaluations to compute the fields is the number of grid points multiplied by the number of charge centers different from 0.

The dielectric is assumed to be constant in space. To use inhomogeneous dielectrics you will need to use a program that solves the poisson equation and import the generated field in Amira.

Because of the inverse relationship with *r*, field values may become very large if grid points are near to charge centers. To avoid the implicated problems, the distance will be set to 0.01 Angstrom during the computation, whenever it is smaller than this minimum value.

Units of the calculated fields will be  $\text{kg} * \text{Angstroem}^2 * \text{s}^{-2} * \text{chargeunit}^{-1}$  for the potential field and  $\text{kg} * \text{Angstroem} * \text{s}^{-2} * \text{chargeunit}^{-1}$  for the electric field, with *chargeunit* being the unit of the charges supplied by the user in the charge attribute.

## 2.16 MolSurfaceView

This module visualizes data objects of type *MolSurface*. Objects of type *MolSurface* can be either molecular surfaces, such as *van der Waals*, *solvent accessible*, or *solvent excluded surfaces*, generated by the *CompMolSurface* module, or it can be molecular interfaces, generated with the *CompMolInterface* module. Apart from a connection to the molecular surface, this module also should be connected to the molecules the surface was generated from, in order to exploit the full functionality of the *MolSurfaceView*. If the molecular surface was generated in the current *amira* session and the molecules were not deleted in between, the *MolSurfaceView* will be automatically connected to these molecules.

Like the other molecular visualization modules, this module allows you to color the molecule, i.e., its surface, according to various different color schemes, using *Molecular Pack*'s *coloring component*. The *MolSurfaceView* has several pick modes implemented. Depending on the pick mode, clicking on the surface results in different actions. E.g., you can highlight parts of the surface by clicking on it with the left mouse button. How the click is interpreted depends on the current selection mode. If *default* colors are used, clicking is interpreted as in the *SurfaceView* module. Clicking on the surface with the middle mouse button displays information about the clicked on part of the molecule. See the description on the *pick action* port for more information.

### Connections

**Data** [required]

Object of class *MolSurface*.

**ColorField** [optional]

Scalar field to color the surface according to physical or chemical properties.

**Colormap** [optional]

Colormap for pseudocoloring the surface according to the values in a scalar field.

**Molecule** [optional]

Object of type molecule. Needed if information about the surface is required.

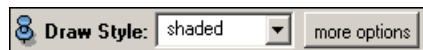
**Continuous CM** [optional]

Needed for coloring the molecular surface according to atomic information.

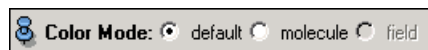
**Discrete CM** [optional]

Needed for coloring the molecular surface according to atomic information.

## Ports

**Draw Style****Colormap****BaseTrans**

See the description of MolSurfaceView's *base class* for an explanation of the first three ports.

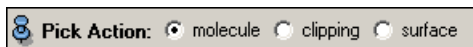
**Color Mode**

There are three color modes. The first mode uses default colors to color the outside triangles differently than the inside triangles. If one or two molecules are connected to the *MolSurfaceView*, you can choose the second mode to color the molecular surface according to the *coloring* component(s). The last mode, finally, allows you to color the surface by use of a scalar field (representing chemical or physical properties). This mode can only be chosen if the *ColorField* port is connected to a field.

**Default Colors**

You can change the default colors by clicking on them. A color editor will appear, enabling you to change the color.

### Pick Action



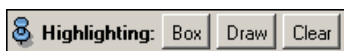
You can change the behavior of the pick action by selecting one of three modes. If the first mode, *molecule*, is selected, left-clicking on the surface highlights the selected parts of the surface, while picking with the middle mouse button leads to information about the picked atom(s) being displayed in the upper left corner of the viewer. If *clipping* is selected, left-clicking on the surface clips all those parts of the surface further away than the value specified by the *clipping distance*. Finally, if the mode *surface* is selected, the behavior will be like this in the *SurfaceView*.

### Clipping Distance



Distance threshold used for clipping.

### Highlighting



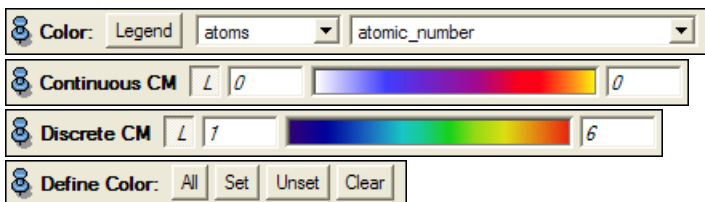
This port provides functionality for selecting (or highlighting) atom surface patches in the viewer window. For more information, see the general section on *highlighting*.

### Buffer



This port is the front end to a filter that determines the atom surface patches being displayed by the *MolSurfaceView*. The functionality of the port's buttons are described in the general section on *highlighting*.

### Coloring



The four ports above belong to the *coloring* component. There will be one component for each molecule connected to the *MolSurfaceView*.

## 2.17 MoleculeLabel

This module allows you to label certain parts of the molecule. If a *MoleculeLabel* module is connected to an object of type *Molecule*, all clicks in the viewing window normally handled by a viewing module connected to the same molecule will instead be handled by the *MoleculeLabel* module, by default.



Handling of a click by the *MoleculeLabel* module means that it will change the labeling, i.e., add or remove a label. The kind of label to be displayed depends on the viewing module and the current selection mode. For example, if the selection mode is *atoms*, a label for the selected atom will be displayed; if it is *residues*, a label for the residue the atom belongs to will be displayed. Up to two data items per label can be displayed. Which data is displayed, can be set in the *Attributes* port (see below).

The user interface is built as follows. The *levels* port allows you to choose a label level. The other ports, except the last one, *Options*, correspond to this level. They allow you to specify how the labels of the selected level will be displayed, i.e., what data is displayed for each label, and in which color and size labels are displayed. Do not assume that a label of the selected level will appear if you click parts of the molecule. This solely depends on the selection mode as mentioned above. You may also hide all labels of the current level.

The last port, *Options*, does not refer to any level, but determines the overall behavior of the module.

There are a few Tcl commands that allow you to set labels. The `setLabelString` command enables you to set arbitrary labels to an atom or a group of atoms. See the description of commands at the end of this page for more information.

## Connections

**Data** [required]

The molecule to be labeled.

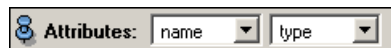
## Ports

### Levels



Choose a level to change the display properties of all labels of this level. The *Attributes*, *Level Option*, *Buttons*, *Font size*, and *Color* ports have individual values for each level.

### Attributes



Port to select two attributes that will make up the label string for the current level. By default, only one attribute is displayed.

### Level Option



If the level option *isVisible* is set, all currently selected labels of this level are displayed. In order to hide all labels of a certain level without removing them, deselect this option.

### Labels



The four buttons of this port affect only the labels of the current level. The first button displays all labels, the second adds a label for each completely highlighted group, the third button removes all labels. Finally, the fourth button updates the label strings according to the currently selected attributes, then updates the positions of the labels to display them as well as possible.

### Font size



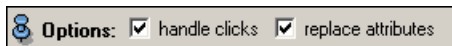
Change the font size of the level's labels.

### Color



Color of the level's labels. To change the color, click on the *Color* button. As result, a color editor will be displayed in which you can choose the color you wish.

### Options



Overall options. If the first option is selected, clicks that are normally handled by the viewing modules will instead be handled by the *MoleculeLabel* module, resulting in new labels being displayed and already visible labels being removed. The second option allows you to compose labels of one level differently. If the option is not set, the currently displayed labels will not be changed if the *Attribute* port changes, otherwise they will.

## Commands

There are a few commands to set labels via the command line interface.

`setLabel <levelName/{groupName|groupRange}> [<attr1>] [<attr2>]`

This command sets new labels for a group or a range of groups specified by the first argument. If one or two attributes are given (as strings), those attributes will be assigned as labels to the group(s). If no attribute is given, the currently selected attributes of the level are used. For more information about specifying a group or a range of groups, see the `list` command in the description of *Molecule*.

Example:

```
setLabel residues/NIL14-NIL98 name index
```

`unsetLabel <levelName/{groupName|groupRange}>`

Hides the labels of the specified group or range of groups.

`setLabelString <levelName/groupName> <string>`

Assign the label 'string' to the specified group.

## 2.18 MoleculeView

The *MoleculeView* allows you to display molecules in three different representations: atom spheres, wire frame, and ball-and-stick. For each of these representations there are two *Quality* modes, *fast* and *correct*.

Atoms can be colored according to the *groupings* defined on the molecule, e.g., atoms, residues, and chains. Moreover, the displayed part of the molecule can be restricted by using the *selection browser* as well as by interaction in the viewer.

In addition to just visualizing molecules, this module provides tools that facilitate interaction with the molecule. For example, you can select atoms to find out which index they have and which group they belong to. The ability to select atoms is also a precondition for the alignment procedure described in the section on *aligning molecules*.

In order to measure distances and angles in the molecule, right-click the *MoleculeView* in the *Pool*, which will open a pop-up menu with the entry *Measurement*.

### Connections

#### Data [required]

The molecule to be visualized.

#### ReferenceMolecule [optional]

A reference molecule must be connected if you want to study the alteration of the torsion angles from one time step of a molecular trajectory to another (see the *Options* port below for more information).

#### Continuous CM [optional]

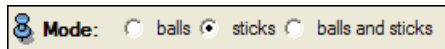
The colormap that is used to map values of float attributes to colors.

#### Discrete CM [optional]

The colormap that is used to map values of discrete attributes, e.g., integers and strings, to colors.

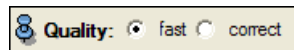
### Ports

#### Mode



Select one of three main display modes: balls, sticks, or balls and sticks. The default is sticks.

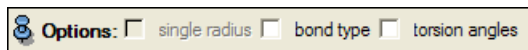
#### Quality



The appearance of the balls and sticks can be altered by selection of one of two different qualities,

a *fast* mode and a *correct* mode. The fast mode shows sticks as lines and spheres by rendering a square with the image of the sphere. This permits the display of large numbers of spheres at interactive speed. However, showing the image of a sphere means the 3D expansion of the sphere in only two directions is taken into account. As long as none of the spheres intersect, there will be no visible difference between spheres drawn in *fast* mode and spheres drawn in *correct* mode. In the correct mode, spheres are represented by textured triangular meshes and sticks by cylinders.

### Options



Options: ☐ single radius ☐ bond type ☐ torsion angles

*Single radius*: display all atoms with the same radius.

*Bond type (only in fast quality mode)*: display correct bond type, e.g., single, double, triple, or aromatic.

*Torsion angles*: display torsion angles by the use of a coil around the stick.

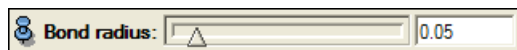
### Atom radius



Atom radius:  0.1

If the option *single radius* is selected, the value given here will be interpreted as the sphere radius for all atom spheres. Otherwise, the value scales the van der Waals radius of the atom.

### Bond radius



Bond radius:  0.05

Radius of cylinders representing bonds (only visible in *correct* display mode).

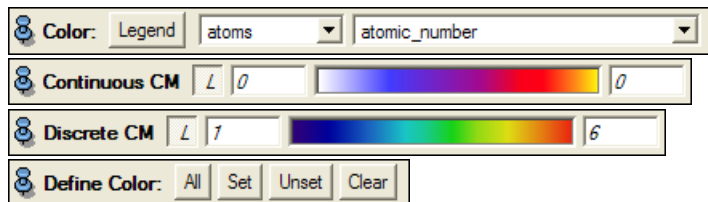
### Twist factor



Twist factor:  1

Allows adjustment of the twist of the coil.

### Coloring



Color: Legend atoms atomic\_number

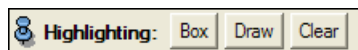
Continuous CM L 0  0

Discrete CM L 1  6

Define Color: All Set Unset Clear

The four ports above are described in the general section on *visualization of molecules*.

### Highlighting

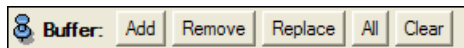


Highlighting: Box Draw Clear

This port provides functionality for selecting (or highlighting) atoms in the viewer window. For

more information, see the description of the highlighting port in the general section on *visualization of molecules*.

### Buffer



This port is the front end to a filter that determines the set of atoms being displayed by the *Molecule-View*. The buttons are described in the section on *visualization of molecules*.

## Commands

`setTextureAntiAliasing <value>`

This command enables (1) or disables (0) antialiasing for the textures used in fast sphere mode. Antialiasing has no impact on the rendering speed. However, if antialiasing is enabled and you have many different colors, the generation of the textures might be considerably slower.

## 2.19 Observables

This module is used to display information about a molecular trajectory. All properties stored in the trajectory can be displayed, e.g., *potential energy*, *kinetic energy*, *total energy* etc.

### Connections

**Data** [required]

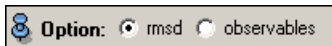
The molecular trajectory for which properties should be displayed.

**Time** [optional]

A time object that may be connected to the module in order to synchronize, e.g., a viewing module with the line in the plot window, indicating the position of the time step within the trajectory.

### Ports

#### Option



The two options allow you to specify which kind of value, *root mean square deviation (rmsd)* or some observable value, should be displayed. The observable type needs to be further specified by the *Observable* port.

#### Alignment



Select: all in slave in master in both Set

Selection: empty

The three ports above specify how the molecules are transformed before computing the *rmsd* value. See the description in the general section on *alignment of molecules* for a detailed explanation.

### Observable Type

Observable Type: potential\_energy

Choose the type of property that should be plotted.

### Action

Action: Show plot

Plot the *Observable Type* within a new window.

### Plot Option

Plot Option: ☒ display probe values ☐ single plot

Select the first toggle if the values of the plotted properties at the current time are shown in the plot window. If the second option is selected, only a single plot will be shown, and changing the *Observable Type* will update the current plot.

### Time

Time: [Slider] 1

This port is most useful in conjunction with a viewing module in order to see the property of the displayed time step. In order to create a time object, click the *Time* slider with the right mouse button and select *Create time* from the menu. As a result a time object will appear in the Pool, which should be connected with an object of type *Molecule* which can be visualized with, for example, the *MoleculeView* module.

## 2.20 PrecomputeAlignment

This module can be used to generate a data object that contains an alignment for every step of a whole *trajectory*.

Press the *Apply* button to start the computation.

### Connections

#### Data [required]

The molecule must be connected to the *molecular dynamics trajectory* that the alignment should be computed for.

**AlignMaster** [optional]

Reference molecule.

**PrecomputedAlignment** [optional]

You will find the description of the above two connections in the section on *alignment of molecules*.

## Ports

### Mode



Two modes are available. In the first mode (*simple alignment*), the module acts as a recorder for the time-step-based alignments described in the section *Aligning Molecules*. In the second mode (*multiple alignment*), the mean squared distance of every time step to all other time steps is minimized. A reference molecule can be connected to specify atoms to be taken into account. If a reference is connected, the mean of the aligned time steps will be aligned to the reference.

### Alignment



You will find the description of the three ports above in the section on *alignment of molecules*.

## 2.21 PseudoElectronDensity

This module computes a “pseudo electron density” for a given molecule by representing its atoms by three-dimensional Gaussian functions, which are summed up on a regular grid. The three-dimensional density can then be visualized with the *Vortex* or *Isosurface* modules.

## Connections

**Data** [required]

Molecule for which the density should be computed.

## Ports

### Voxel Size



Voxel size of the regular grid to be computed.

### Down Factor

A control panel for the 'Down Factor' parameter. It features a small icon of a person with a question mark, followed by the text 'Down Factor:'. To the right is a horizontal slider with a triangle marker, and a numeric input field containing the value '1'.

The *down factor* influences the width of the Gaussian functions. The smaller the value, the larger the width of the Gaussian function.

### Minimal Value

A control panel for the 'Minimal Value' parameter. It features a small icon of a person with a question mark, followed by the text 'Minimal Value:'. To the right is a horizontal slider with a triangle marker, and a numeric input field containing the value '0.001'.

The *minimal value* determines the cut off distance for each Gaussian function. The smaller the value the smoother the density will look.

### Scalarfield dimensions

A control panel for the 'Scalarfield dimensions' parameter. It features a small icon of a person with a question mark, followed by the text 'Scalarfield dimensions:'. To the right is a text input field containing the value '146 x 135 x 165'.

This info port gives you the size of the density field to be computed, so that you can adjust the parameters, in particular the voxel size, to meet a field size that is acceptable to you.

### Cutting distance

A control panel for the 'Cutting distance' parameter. It features a small icon of a person with a question mark, followed by the text 'Cutting distance:'. To the right is a text input field containing the value '2.628261'.

Maximal cutting distance of the Gaussian functions according to the specified *down factor* and *minimal value*.

## 2.22 RankTimeStep

This module takes an object of type *MolTrajectory* as input and either sorts all molecules in the trajectory or selects a single molecule from the trajectory. Selection and sorting can either be done using the *observable* values of the trajectory or the *rmsd* value in comparison to a specific molecule, which then needs to be connected to the module.

The module is, e.g., very helpful in conjunction with the module for the computation of the *mean molecule*. It allows you to find the time step in the trajectory that best approximates the mean molecule.

Press the *Apply* button to start the search (and sorting). When pressing the button for the first time, a new object of type *Molecule*, connected to the trajectory, will be added to the Pool. This molecule is used to set the index of the molecule with min value, max value or according to the index of the sorted list.

### Connections

#### Data [required]

Molecular trajectory for which the mean molecule should be computed.



### AlignMaster [optional]

This connection is only required if you want to search the trajectory according to the *rmsd* value, since then you will need a reference molecule.

### PrecomputedAlignment [optional]

Instead of aligning all time steps of a trajectory to the *AlignMaster* molecule, you can use a pre-computed alignment to align the time steps.

## Ports

### Search Option



The two options allow you to specify which kind of value, *root mean square deviation (rmsd)* or some observable value, should be used for ranking the time steps of the trajectory. The observable type needs to be further specified by the *Observable* port.

### Search Option 2



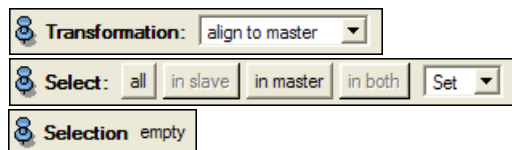
Three options are possible. The first two allow you to search for a single time step with either minimum or maximum value. The last option, *all sorted*, gives you the most flexibility and is therefore the default.

### Observable



This port allows you to select the observable you are interested in.

### Alignment



The three ports above specify how the molecules are transformed before computing the *rmsd* value. See the description in the general section on *alignment of molecules* for a detailed explanation.

### Sorted Timesteps



This slider allows you to step through the time steps after sorting. Time step 0 is that having the minimal value.

## 2.23 SecStructureView

This module provides the functionality for viewing the secondary structure of a molecule. It relies on the information given in the data entry of the connected molecule. Helices, sheets, and turns can only be displayed if information about these structures has been read in (for example from a *pdb* file).

The **backbone** will be interpolated by using the coordinates of those atoms whose type is set to “N”, “C” and “CA”. If the atom-type attribute does not exist (for example, molecules that have been imported from UniChem files) no backbone can be viewed.

The **user interface** of the module includes general ports to configure the overall appearance of the molecule, as well as specific ports for each secondary structure type. The latter let you adjust type-specific parameters, such as colors or widths. To switch between the specific ports, click on the structure type you want to configure in the *View Options* port.

**Selecting and Highlighting:** Parts of the molecule can be selected by clicking onto specific regions in the selection mode (you can switch to the selection mode of the viewer by pressing the ESC key). The default level of the structure that is selected is the “residue” level. A detailed description of the selection concept can be found in the section *visualization of molecules*. Selected structures will be highlighted in the color that you choose in the port *Highlighting Color*.

### Connections

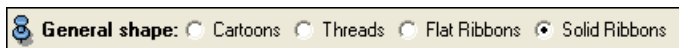
**Data** [required]

The molecule that is to be viewed.

### Ports

#### General Ports:

##### General Shape



This port determines the general shape for viewing the secondary structure. The following view modes are available:

- **Cartoons:** In the *cartoon* mode, you can select a view mode for each secondary structure type separately. The backbone and the turns will be shown as tubes that run through the central atoms of each residue. For helices and sheets, additional (more abstract) view modes like cylinders or arrows are available. You can configure these submodes in the specific port sections for the secondary structure type. This is the mode most frequently used for representing the secondary structure of a molecule. However, due to its complex triangulation, the rendering performance is distinctly worse than that of the other two modes.
- **Threads:** The backbone will be drawn as a set of lines running parallel to the peptide planes of the amino acids. This mode will show the surface orientation of the backbone, thus making

parts containing symmetries (like helices or sheets) easily recognizable. It is also the view mode which can be rendered fastest.

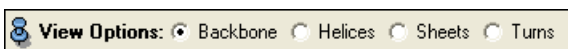
- **Flat Ribbons:** Same as the thread mode, but the areas between the lines will be a filled surface, thus giving a better surface impression with the trade-off of slightly lower rendering performance.
- **Solid Ribbons:** Unlike flat ribbons, solid ribbons will have an elliptic cross section.

### Threads Line Width



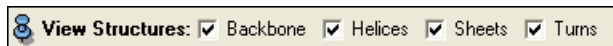
This slider lets you adjust the width of the threads-lines.

### View Options



To switch between the options of the different structure types, you can use these radio buttons. The option ports for the different structure will then be visible.

### View Structures



This port can be used to filter out the display of all sheets, helices, or turns. The backbone parts will be displayed as random coil tubes instead.

### Filter

The secondary structure view also employs a filter which can be used to hide certain parts of the molecule. You can hide sections by using the *selection browser*. This port is explained further in the section *visualization of molecules*.

### Complexity



The *Complexity* port allows the adjustment of the overall exactness of the structures displayed. It will influence the number of points with which the backbone coordinates will be interpolated as well as the detail of triangulation or the number of lines in the thread view mode. If you are viewing large molecules on a machine with slow graphics hardware, turn this value down to a more appropriate value.

### Highlighting Color



Lets you choose in which color selected residues will be highlighted.

### Backbone Ports:

### Backbone Width

A control panel for Backbone Width. It features a blue icon on the left, followed by the text "Backbone Width:". To the right is a horizontal slider with a triangle marker, and a numerical input field on the far right containing the value "0.5".

With this port you can adjust the width of the backbone shown as threads or ribbons. A similar port exist for the other secondary structure type.

### Backbone Eccentricit

A control panel for Backbone Eccentricity. It features a blue icon on the left, followed by the text "Backbone Eccentricity:". To the right is a horizontal slider with a triangle marker, and a numerical input field on the far right containing the value "0".

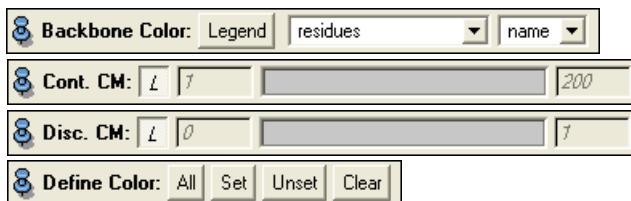
If the general shape is *solid ribbons* this ports lets you configure the elliptical eccentricity in the range between 0 (completely flat ellipse) to 1 (circle). A similar port exists for the other secondary structure types.

### Backbone Tube Radius

A control panel for Backbone Tube Radius. It features a blue icon on the left, followed by the text "Backbone Tube Radius:". To the right is a horizontal slider with a triangle marker, and a numerical input field on the far right containing the value "0.2".

This port lets you change the backbone radius.

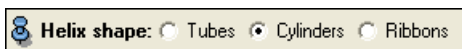
### Coloring

A set of four control panels for coloring. The first panel is labeled "Backbone Color:" and includes a "Legend" button, a dropdown menu showing "residues", and another dropdown menu showing "name". The second panel is labeled "Cont. CM:" and has two input fields with values "7" and "200". The third panel is labeled "Disc. CM:" and has two input fields with values "0" and "7". The fourth panel is labeled "Define Color:" and includes four buttons: "All", "Set", "Unset", and "Clear".

The four ports above are described in the section on *visualization of molecules*. Similar ports exist for helixces, sheets, and turns.

## Helix Ports:

### Helix shape

A control panel for Helix shape. It features a blue icon on the left, followed by the text "Helix shape:". To the right are three radio buttons labeled "Tubes", "Cylinders", and "Ribbons". The "Cylinders" radio button is currently selected.

This port lets you choose between different methods for viewing helices in *cartoon* mode.

- **Tube:** The tube view will show the helix by drawing a tube connecting the interpolated backbone coordinates.
- **Cylinder:** The cylinder view will approximate the axis using a least squares error minimization for the sum of the distances between the backbone coordinates of the helix and the axis. These axes will be used to center the cylinders for the helices.
- **Ribbon:** The ribbon view is simillar to the tube representation except that the backbone is shown using a surface whose normals are set perpendicular to the peptide planes.

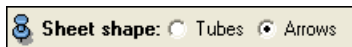
## Helix Radius

A control port for Helix Radius. It features a blue icon on the left, the text "Helix Radius:", a horizontal slider with a triangle marker, and a numeric input field on the right containing the value "0.5".

With this port you can adjust the radius of the shown tubes, or cylinders.

## Sheet Ports:

### SheetShape

A control port for Sheet shape. It features a blue icon on the left, the text "Sheet shape:", and two radio button options: "Tubes" and "Arrows". The "Arrows" option is selected.

This port lets you choose between different methods for viewing sheets in *cartoon* mode.

- **Tube:** The tube view will display the sheet using a tube connecting the interpolated backbone coordinates.
- **Arrow:** The arrow view will display arrows that indicate the orientation of the peptide planes, thus showing the surface determined by the hydrogen bonds between the strands.

### Sheet Arrow Width

A control port for Sheet Arrow Width. It features a blue icon on the left, the text "Sheet Arrow Width:", a horizontal slider with a triangle marker, and a numeric input field on the right containing the value "1.5".

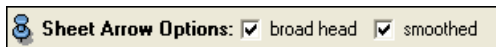
This port is visible if you view the sheets as arrows. It determines the width of the arrows.

### Sheet Arrow Height

A control port for Sheet Arrow Height. It features a blue icon on the left, the text "Sheet Arrow Height:", a horizontal slider with a triangle marker, and a numeric input field on the right containing the value "0.2".

This port is visible if you view the sheets as arrows. It determines the height of the arrows. If the height is 0 an optimized (and faster) triangulation will be used.

### Sheet Arrow Options

A control port for Sheet Arrow Options. It features a blue icon on the left, the text "Sheet Arrow Options:", and two checked checkbox options: "broad head" and "smoothed".

The *broad head* option will show the start of the arrow head wider than the arrow tail. The *smoothed* option will Bezier interpolate the backbone coordinates of the sheet to produce a smoother visualization of the surface.

### Sheet Radius

A control port for Sheet Radius. It features a blue icon on the left, the text "Sheet Radius:", a horizontal slider with a triangle marker, and a numeric input field on the right containing the value "0.3".

If you view the sheets as tubes, this port lets you configure the radius of the tubes.

## Turn Ports:

### Turn radius

A control port for Turn radius. It features a blue icon on the left, the text "Turn radius:", a horizontal slider with a triangle marker, and a numeric input field on the right containing the value "0.3".

This port lets you specify the radius of the *turn* tubes.

## Commands

The console window offers some additional functionality. The easiest way to use the console for a SecStructureView is to click on the viewer object in the Pool and then press the TAB key in the console window to display the name of the selected object. After the name you can use the following selection commands:

`setResidueTransparency <residueIndex> <transparency>`

Will set the transparency of the residue with index ;residueIndex<sub>i</sub> to the value of ;transparency<sub>i</sub> which must be in the interval [0,1].

`setSecStructTransparency <secIndex> <transparency>`

Same as above but will affect all residues of the secondary structure ;secIndex<sub>i</sub>.

`setResidueColor <residueIndex> <r> <g> <b>`

Will set the color of the residue with index ;residueIndex<sub>i</sub> to the color which is given in rgb values in the interval [0,1]. The effect of the change will be undone as soon as you change a coloring port of the module.

`setSecStructColor <secIndex> <r> <g> <b>`

Same as above but will affect all residues of the secondary structure ;secIndex<sub>i</sub>.

`setSpecularity <s>`

Set the specularity to ;s<sub>i</sub> where 0 is minimal and 1 is maximal specularity.

## 2.24 TubeView

With the *TubeView* module you can connect an arbitrary set of consecutive atoms with a tube through interpolated atom coordinates. To define the atom set, you can use *atom expressions*. The order with which the atoms are connected is determined by their sequence.

**Selecting and Highlighting:** Parts of the molecule can be selected by clicking onto specific regions in the selection mode (you can switch to the selection mode of the viewer by pressing the ESC key). A detailed description of the selection concept can be found in the section *visualization of molecules*. Selected structures will be highlighted in the color that you choose in the port *Highlighting Color*.

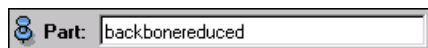
## Connections

**Data** [required]

The molecule you want to be shown.

## Ports

**Part**



This port lets you enter an *atom expression*. The atom expression will determine the set of atoms that will be connected. The default value is *backbonereduced*, which means that the tube will indicate the location of polypeptide and RNA/DNA backbone atoms.

To restrict the display to just a few atoms it is more convenient to use the filter that can be accessed with the *Selection Browser*. To include atoms in the sequence to be shown it must be active in the set defined by the atom-expression as well as the filter.

### Distance Cutoff

A control panel for the Distance Cutoff. It features a small icon of a tube on the left, followed by the text "Distance Cutoff:". To the right is a horizontal slider with a triangle marker, and a text box on the far right containing the value "8".

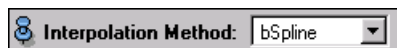
All atoms that are further away than the specified value will not be connected.

### Radius

A control panel for the Radius. It features a small icon of a tube on the left, followed by the text "Radius:". To the right is a horizontal slider with a triangle marker, and a text box on the far right containing the value "0.2".

This port specifies the radius of the tube.

### Interpolation Method

A control panel for the Interpolation Method. It features a small icon of a tube on the left, followed by the text "Interpolation Method:". To the right is a dropdown menu with "bSpline" selected.

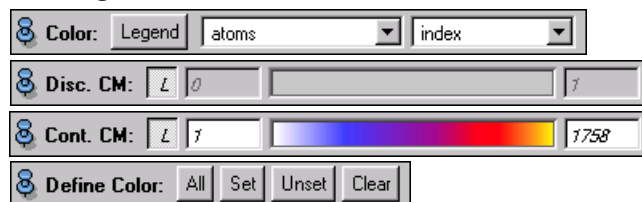
The tube coordinates will be interpolated by using the atom coordinates as base points. This port lets you choose the interpolation method. *BSpline* interpolation is generally smoother than *cubic* interpolation but the tube will not necessarily run through the atom coordinates.

### Complexity

A control panel for the Complexity. It features a small icon of a tube on the left, followed by the text "Complexity:". To the right is a horizontal slider with a triangle marker, and a text box on the far right containing the value "0.3".

This port specifies the complexity of the tube triangulation as well as the number of interpolation points to be added between each atom coordinate. The higher the value the better the display quality but the slower the rendering.

### Coloring

A set of controls for coloring the tube. The first row has a "Color:" label, a "Legend" button, a dropdown menu set to "atoms", and another dropdown menu set to "index". The second row has a "Disc. CM:" label, a small input field with "1", a slider, and a text box with "7". The third row has a "Cont. CM:" label, a small input field with "7", a color gradient bar, and a text box with "1758". The fourth row has a "Define Color:" label and four buttons: "All", "Set", "Unset", and "Clear".

You will find the description of the four ports above in the section on *visualization of molecules*.

### Highlighting Color

A control panel for the Highlighting Color. It features a small icon of a tube on the left, followed by the text "Highlighting color:". To the right is a small yellow square color swatch.

Tube sections that belong to atoms that have been highlighted with the *Selection Browser* will be shown in this color.

# Chapter 3

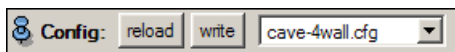
## VR Pack

### 3.1 VR Pack control module

The VR Pack module is the central control module of the *VR Pack Virtual Reality Extension*. It allows you to activate different screen configurations specified in config files located in \$AMIRA\_ROOT/share/config or \$AMIRA\_LOCAL/share/config. For details about supported features such as multi-pipe or multi-thread rendering, soft-edge blending, passive stereo support, or head-tracking, please refer to the main VR Pack documentation. In the following we assume that you are already familiar with the basic concepts of VR Pack.

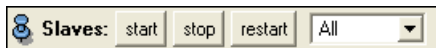
#### Ports

##### Config



This port provides a list of all config files found in \$AMIRA\_ROOT/share/config and \$AMIRA\_LOCAL/share/config. This is the same as in the *Config* menu of the *amira* main window. Note that valid config files are only searched once. It is not possible to add a config file when *amira* is running. However, existing config files may be modified and reloaded using the *reload* button. The *write* button allows you store a modified configuration into a file (for example after the tracking system has been calibrated). If this button is pressed, the file dialog pops up.

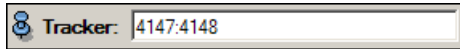
##### Slaves



This port provides a list of all the slaves included in the config file. Use the *start*, *stop*, and *restart* buttons to start, stop, and restart one or all slaves included in the config file.



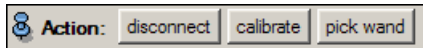
## Tracker



Specifies how the connection to the tracking system should be established. In order to connect to a running trackd server, the value of this port should be `<id of controller reader>:<id of tracker reader>`, where the two ids are the shared memory ids of the trackd controller reader and the trackd tracker reader as specified in the *trackd.conf* file.

In order to activate a simple *tracker emulator*, you can type in *test* into the text field.

## Action

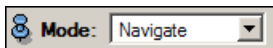


The *connect* button allows you to connect to the tracking system. Once you are connected to the tracking system, the button label changes to *disconnect*. Pressing the button then disconnects from the tracking system.

The *calibrate* button allows you to calibrate the tracking system, i.e., to find the transformation between raw tracker coordinates and the screen coordinates used in the config file. For more information about the calibration process, please refer to Section ??.

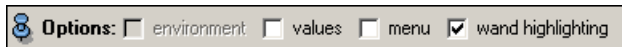
The *pick wand* button allows you to change the offset between the 3D input device and the visual representation of the virtual wand. If you press the button, the wand is displayed half-way between the current eye position and the default object position. You can then pick the virtual wand in the way which is most convenient for you.

## Mode



Option menu allowing you to change between ordinary interaction mode (translate and rotate the whole scene using the 3D wand), fly mode (navigate through the scene like an airplane), or 2D mouse mode (control the 2D mouse using the 3D wand). In order to exit 2D mouse mode, double-click the 3D wand while the 2D mouse is over an insensitive GUI element.

## Options



If the *values* toggle is activated, the current coordinates of the wand sensor and of the head sensor are displayed in the upper left corner of the screen. The coordinates are transformed into the screen coordinate system which was used in the config file. Only during calibration raw tracker coordinates are displayed.

The *menu* toggle displays or hides the 3D menu in the scene.

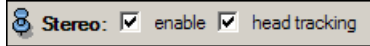
The *wand highlighting* toggle can be turned off to gain some performance. This is particularly useful when the scene contains a large number of interactive objects.

## Adjust size

A slider control for 'Scene size'. It features a horizontal bar with a triangular slider in the center. To the right of the bar is a numerical input field containing the value '0'.

This slider allows you to change the overall scaling of the scene.

## Stereo

A control panel for stereo settings. It includes a 'Stereo' label, a checked 'enable' checkbox, and a checked 'head tracking' checkbox.

The *enable* toggle allows you to enable or disable stereo mode. Depending on the camera mode defined in the config file, either active stereo or passive stereo will be used. This option is the only one which is provided for flat screen configurations (compare Section ??).

The *head tracking* toggle allows you to temporarily enable or disable head tracking. If head tracking is disabled, the default camera position specified in the config file will be used instead of the position reported by the head sensor. If no head sensor has been configured or if a flat screen configuration is being used, this button is disabled.

## Left eye offset

Three input fields for 'Left eye offset'. The first field contains '6', the second contains '0', and the third contains '2'.

Define here the center of the left eye relative to the head tracker position. This port is only visible if stereo mode is enabled for immersive configurations. Its value is initialized with the *config file* leftEyeOffset field.

## Right eye offset

Three input fields for 'Right eye offset'. The first field contains '13', the second contains '0', and the third contains '2'.

Define here the center of the right eye relative to the head tracker position. The same remarks as for *Left eye offset* apply.

## Zero parallax balance

A slider control for 'Zero parallax balance'. It features a horizontal bar with a triangular slider. To the right of the bar is a numerical input field containing the value '1'.

This slider allows you to adjust the stereo balance. This port is only visible if stereo mode is enabled for flat screen configurations.

## Camera offset

A slider control for 'Camera offset'. It features a horizontal bar with a triangular slider. To the right of the bar is a numerical input field containing the value '0.7'.

This slider allows you to adjust the offset between each eye. This port is only visible if stereo mode is enabled for flat screen configurations.

## Commands

`start <hostname>`

Starts the slave hostname. If hostname is null, starts all slaves.

**stop <hostname>**

Stops the slave hostname. If hostname is null, stops all slaves.

**restart <hostname>**

Restarts the slave hostname. If hostname is null, restarts all slaves.

**setHeadTrackerId <id>**

Changes the head tracker id on-the-fly. Usually the id is defined in the config file. A value of -1 disables the head tracker.

**setWandTrackerId <id>**

Changes the wand tracker id on-the-fly. Usually the id is defined in the config file. A value of -1 disables the wand tracker.

**setSoftEdge -g <gamma> <overlap> [<overlap>...]**

Initializes a dual-screen soft-edge configuration with the given overlap percentage <overlap>. This is a value between 0 (no overlap) and 1 (full overlap). <gamma> defines the gamma value for the soft-edge blending.

**setPanorama <overlap1> [<overlap2> ...]**

Initializes a multi-screen panorama configuration with the given overlap percentages. These are values between 0 (no overlap) and 100 (full overlap). The current configuration must be a flat screen configuration.

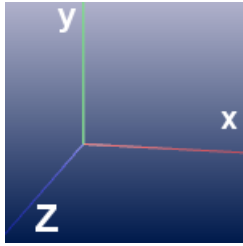
**setDefaultCameraPosition <x> <y> <z>**

Sets the default camera position for a 3D configuration. The position must be specified in the same coordinate system as the screens in the VR Pack config file. The default camera position is used if head tracking is turned off.

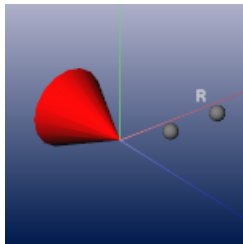
## 3.2 VR Pack show config module

This module displays a model of the currently loaded VR Pack configuration. It's intended to give the visual feedback needed during the configuration process and for easier error detection. It shows the projection planes, camera, sensor and scene positions as well as the button state. Therefore the fourth viewer is used.

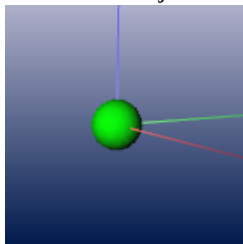
Objects associated with a local coordinate system are drawn with an axis consisting of three colored line segments. The x axis is red, y is green, and z is blue.



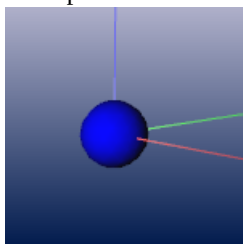
The camera position is marked with a red cone and two small gray spheres, one for each eye. The eye spheres are translated according to the configured eye offset value. One eye sphere is marked with the letter *R* which means that this is the sphere for the right eye.



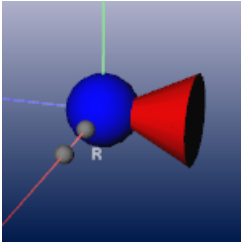
The default object- or scene position is marked with a green sphere.



Blue spheres represent sensors (3D-tracker), one for every sensor. If one of the buttons is pressed, the blue sphere of the first sensor turns white until all buttons are released.

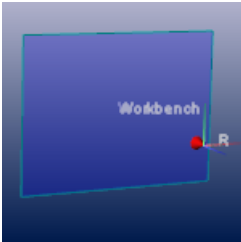


If the camera is bound to one of the 3D sensors, which means that head tracking is turned on, the red cone representing the camera is glued to the blue sensor sphere and follows its movement. At the moment the head-tracking feature is turned off, the camera model snaps back to the default camera position.

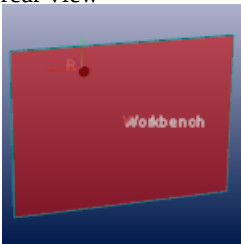


Finally the projection planes are drawn at the positions and dimensions specified in the configuration file. The following two images are showing an environment with a single projection plane. The front side looks blue and a projection plane observed from its back side appears red. By observing the color one can verify that the edge points were specified in the right order.

front view



rear view



The user-defined name of the plane is drawn in its center. Since projection planes are given as a set of four edge points it is possible to produce a non-planar polygon. This misconfiguration leads to a clearly visible edge crossing the plane.

## Connections

**Data** [required]

Connect the *VR Pack control module*.



# Chapter 4

## Large Data Pack

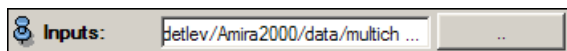
### 4.1 ConvertToLargeDiskData

This module allows the conversion of *Raw Data* or stacks of image data as described in the *TIFF format* to *LargeDiskData*.

You can create a *ConvertToLargeDiskData* module with the *Create* menu. Use the submenu *Data*. You must select the input files with the *Inputs* port, select the output with the *Output* port. The *Apply* button starts the conversion process. You are asked to supply parameters as described in *TIFF format* or *Raw Data* during the conversion process. After a successful conversion, a *LargeDiskData* data object appears in the Pool.

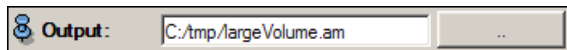
#### Ports

##### Inputs



The list of input files.

##### Output



One output file.

#### Commands

```
forceAmira311Format {0|1}
```

Lets you select the LDA data format. Provided for compatibility with amira 3.1 and earlier.



## 4.2 Isosurface (LDA)

This module generates isosurfaces that are computed and rendered on the graphics hardware. This module does not compute geometry on the CPU. Thus, unlike for the non-LDA Isosurface module, it is not possible to store the results as a surface object or as an Open Inventor format file.

This rendering technique needs more slices than typically used for volume rendering. A low number of slices will create holes in the surface.

This module provides a hardware-accelerated implementation, which uses programmable shader hardware to allow real-time rendering. On platforms that do not have programmable shader support, no isosurface will be drawn.

### Connections

#### Data [required]

The 3D scalarfield from which the isosurface is generated.

#### ROI [optional]

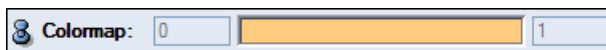
Connection to a module providing a region-of-interest, like *SelectRoi*. If such a module is connected, only the selected part of the volume will be displayed.

#### Colormap [optional]

Colormap used to visualize the data.

### Ports

#### Colormap



Port to select a colormap.

#### Number of slices



The larger this number, the better the image quality and the less the rendering performance. A low number of slices will create holes in the surface.

#### Threshold



Isovalue to be rendered.

## 4.3 LDAExpertSettings

The LDA technology is a multi-resolution data manager that allows the loading of extremely large data sets (hundreds of gigabytes) with interactive navigation. This makes it ideal for interactive exploration of very large volumes.

The *LDAExpertSettings* module allows you to control various parameters in order to optimize the reading and display of the data depending on the capabilities of the hardware being used – amount of system memory, the graphics card, the number of CPUs, etc.

In order to create the *LDAExpertSettings* module, choose *Others/LDAExpertSettings* from the main window's *Create* menu.

A number of the settings that can be set using this module can also be set using the *LDA* tab of the *Edit/Preferences* dialog. They are:

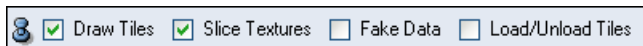
- **View culling**
- **Viewpoint refinement**
- **Move low resolution**
- **Loading policy**
- **Main memory amount**
- **Texture memory amount**

In the *Preferences* dialog the corresponding item is identified as *Video memory amount*.

A change specified in this module will be reflected in the *Preferences* dialog, and vice versa.

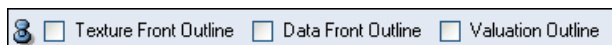
### Ports

#### VisualFeedback1



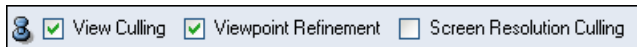
- **Draw Tiles:** Display tile texturing or not.
- **Slice Texture:** Display slice texturing or not.
- **Fake Data:** If selected, fake data is used instead of the real data during loading into main memory. The fake data is programmatically generated and is designed to illustrate features of LDA. It is necessary to reload the data after setting this flag in order to see the result of the selection.
- **Load Unload Tiles:** Show tiles loaded (in red) and unloaded (in blue) in texture memory.

#### VisualFeedback2



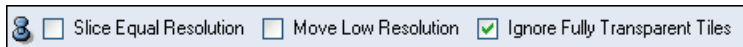
- **Texture Front Outline:** Draw tile outlines for all primitives. The tile outlines of tiles of full resolution are drawn with a brighter gray for the volume, and a brighter green for the slices.
- **Data Front Outline:** Show tiles loaded in main memory. This implies showing the multi-resolution topology. The tile outlines of tiles of full resolution are drawn with a brighter yellow.
- **Valuation Outline:** Displays an octree representation showing the status of the octree. Red dots are tiles that have been valued. Blue indicates the minimum and maximum resolution thresholds. Yellow shows the octree itself.

### Options1


 A control panel for Options1. It features a blue icon with a white '8' on the left. To its right are three checkboxes: 'View Culling' (checked), 'Viewpoint Refinement' (checked), and 'Screen Resolution Culling' (unchecked).

- **View Culling:** If checked, LDA will not load tiles located outside of the view frustum.
- **Viewpoint Refinement:** If checked, LDA will load tiles of higher resolution close to the viewpoint.
- **Screen Resolution Culling:** If checked, LDA will only load only load tiles for which the projection of a voxel is greater than or equal to 1 pixel on screen. This avoids unnecessary loading of high resolution data.

### Options2


 A control panel for Options2. It features a blue icon with a white '8' on the left. To its right are three checkboxes: 'Slice Equal Resolution' (unchecked), 'Move Low Resolution' (unchecked), and 'Ignore Fully Transparent Tiles' (checked).

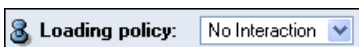
- **Slice Equal Resolution:** If checked, all tiles on a slice will be taken from the same resolution level.
- **Move Low Resolution:** If checked, LDA will use a lower texturing resolution when the user is interacting with the scene.
- **Ignore Fully Transparent Tiles:** If checked, LDA will not load fully transparent tiles (i.e., if all voxels within the tile have an alpha value of 0).

### Num loading threads


 A control panel for Num loading threads. It features a blue icon with a white '8' on the left. To its right is a label 'Num loading threads:' followed by a slider control. The slider has a triangle marker and a numeric input field on the right showing the value '4'.

Number of separate threads used for tile loading. Using more threads can increase performance on multiprocessor machines.

### Loading policy


 A control panel for Loading policy. It features a blue icon with a white '8' on the left. To its right is a label 'Loading policy:' followed by a dropdown menu currently showing 'No Interaction'.

LDA loads the data asynchronously using separate threads. The loading policy allows you to specify when the loading threads should load the data into main memory. *No interaction* means the threads will only load data when there is no user interaction. *Always* is at any time. *Never* means no more data loading will occur.

### Fixed resolution

A control element for 'Fixed resolution' consisting of a blue icon with a gear and a small square checkbox.

If set, LDA will only load tiles from the specified resolution level (specified using the *Fixed resolution level* port).

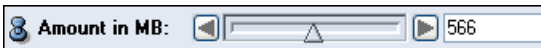
### Fixed resolution level

A control element for 'Fixed resolution level' consisting of a blue icon with a gear, the text 'Fixed resolution level:', and a dropdown menu showing '1/2'.

Resolution level for the fixed resolution mode.

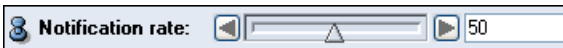
### Main memory:

#### Amount in MB

A control element for 'Amount in MB' consisting of a blue icon with a gear, the text 'Amount in MB:', a slider, and a text box showing '566'.

Specifies how much main memory (in MB) LDA can use to cache data.

### Notification rate

A control element for 'Notification rate' consisting of a blue icon with a gear, the text 'Notification rate:', a slider, and a text box showing '50'.

LDA will redraw after a certain number of tiles are loaded into memory. This slider allows you to specify this number of tiles.

### Texture memory (volumes):

#### Amount in MB

A control element for 'Amount in MB' consisting of a blue icon with a gear, the text 'Amount in MB:', a slider, and a text box showing '100'.

Specifies how much texture memory (in MB) LDA can use for the volume (3D textures).

### Texture load rate

A control element for 'Texture load rate' consisting of a blue icon with a gear, the text 'Texture load rate:', a slider, and a text box showing '1'.

Specifies the maximum number of textures to load into texture memory between two frames.

### Texture memory (slices):

#### Number of textures

A control element for 'Number of textures' consisting of a blue icon with a gear, the text 'Number of textures:', a slider, and a text box showing '256'.

Specifies how much texture memory is allowed in MB for slices (2D textures).

### Texture load rate



Specifies the maximum number of textures to load into texture memory between two frames.

## 4.4 ObliqueSlice (LDA)

The *ObliqueSlice (LDA)* module lets you display arbitrarily oriented slices through a 3D scalar field of any type, as well as through an RGBA color field or a 3D multi-channel field. In the medical context such slices are known as multi-planar reconstructions (MPR).

The module is derived from *ArbitraryCut*. See the documentation of the base class for details about how to adjust position and orientation of a slice. Like in the *OrthoSlice (LDA)* module, two different methods are supported to map scalar values to screen colors, namely linear mapping and pseudo-coloring.

### Connections

#### Data [required]

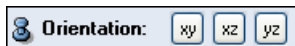
The 3D field to be visualized. 3D scalar fields, RGBA color fields, or 3D multi-channel fields are supported. The coordinate type of the input data doesn't matter. The data will be evaluated using the field's native interpolation method (usually trilinear interpolation).

#### Colormap [optional]

The colormap used to map data values to colors. This port is ignored when linear mapping is selected, or when the module is connected to an RGBA color field or to a 3D multi-channel field.

### Ports

#### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

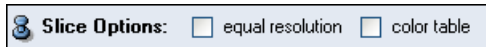
#### Options



If the *adjust view* toggle is set (set by default), the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set, the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved

and the update takes place when the mouse button is released. When *fit to points* is checked, you will be asked to pick 3 points on any geometry in the scene. The slice plane will be updated to intersect the specified points.

### Slice Options



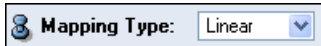
If *equal resolution* is set (unset by default), all ortho slices and oblique slices are drawn using the same resolution. You can toggle *lighting* on or off.

### Interpolation



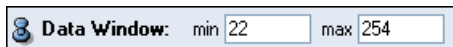
Specifies interpolation type (nearest value or linear interpolation).

### Mapping Type



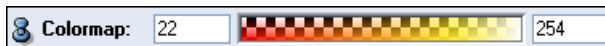
This option menu controls how scalar values are mapped to screen colors. In the case of a *linear* mapping a user-defined data window is mapped linearly to black and white. *colormap* can be used to activate pseudo-coloring. The port is hidden if the module is connected to a color field or a multi-channel field.

### Data Window



This port is displayed if linear mapping is selected. It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, while values above the upper bound are mapped to white.

### Colormap



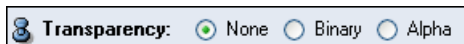
This port is displayed if *colormap* is selected. Choose a colormap to map data to colors.

### Translate



This slider allows you to select different slices. The slices may also be picked with the mouse and dragged directly in the 3D viewer.

### Transparency



This radio box port determines the transparency of the slice. The option is only available if the module is connected to an *RGBA color field*. *None* means that the slice is fully opaque. *Binary*

means that black parts are fully transparent while other parts are opaque. *Alpha* means that opacity is taken as is.

### Embossing



Enables/disables embossing. This rendering technique (also known as "bump mapping") emphasizes gradient changes in the data on the slice. Note that this feature requires programmable shader support in the graphics hardware.

### Embossing Factor



Specifies the intensity of the embossing effect. Default value is 0 (no embossing). Values can be positive or negative. Note that this feature requires programmable shader support in the graphics hardware.

### Channels



This port is shown if the module is attached to a 3D multi-channel object. It allows you to toggle individual channels on or off. Each channel is mapped using a linear intensity ramp. The data window for each channel can be adjusted in the multi-channel object itself.

## Commands

All commands described for *ArbitraryCut* can be applied to *ObliqueSlice (LDA)* as well.

`createImage <image base name>`

This command creates a 2D image from the oblique slice and adds it to the Pool. The image base name can be omitted.

## 4.5 OrthoSlice (LDA)

The *OrthoSlice (LDA)* module is an important tool for visualizing scalar data fields defined on uniform Cartesian grids, e.g., 3D image volumes. Such data are visualized by extracting an arbitrary axial, frontal, or sagittal slice out of the volume. The data values can be mapped to colors or gray levels by one of two mapping methods. The most simple mapping technique uses a linear gray ramp together with two threshold values. This range determines which data values are mapped to black and which are mapped to white. As a second mapping method, an external colormap can be used in *OrthoSlice (LDA)*.

The *OrthoSlice (LDA)* module is also capable of extracting slices of an RGBA color field or of a 3D multi-channel field. In these cases the slices are displayed as is and no mapping method need to be chosen.

## Connections

### Data [required]

The 3D field to be visualized. Currently, regular scalar fields, multi-channel fields, and RGBA color fields with uniform or stacked coordinates are supported.

### Colormap [optional]

Optional colormap used to map scalar data to colors. This port is hidden when *linear* or *histogram* mapping is selected. See also *Colormap*.

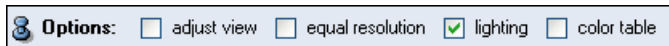
## Ports

### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

### Options



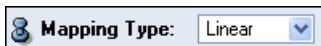
If the *adjust view* toggle is set (unset by default), the camera of the main viewer is reset each time a new slice orientation is selected. *Equal resolution* is false by default. When it is set, all ortho slices and oblique slices are drawn using the same resolution.

### Interpolation



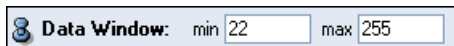
Controls whether interpolation is *nearest* or *bilinear*.

### Mapping Type



This option menu lets you select between two different mapping methods, namely a *linear* gray ramp and *colormap* mode. The port is not available if the *OrthoSlice (LDA)* module is connected to an RGBA color field or to a 3D multi-channel field.

### Data Window



This port is displayed if *linear* mapping is selected. It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, values above the upper bound are mapped to white. In order to quickly change the data window a *ContrastControl* module can be attached to the *OrthoSlice (LDA)*.



## Colormap



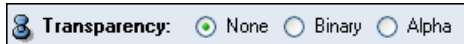
Choose colormap if *mapping type* is set to *colormap*.

## Slice Number



This slider allows you to select different slices. The slices may also be picked with the mouse and dragged directly in the 3D viewer.

## Transparency



This radio box port determines the transparency of the slice. *None* means that the slices are fully opaque. *Binary* means that black parts are fully transparent while other parts are opaque. *Alpha* means that opacity is proportional to luminance. If a colormap is used for visualization, opacity values are taken from there.

## Embossing



Enables/disables embossing. This rendering technique (also known as "bump mapping") emphasizes gradient changes in the data on the slice. Note that this feature requires programmable shader support in the graphics hardware.

## Embossing Factor



Specifies the intensity of the embossing effect. Default value is 0 (no embossing). Values can be positive or negative. Note that this feature requires programmable shader support in the graphics hardware.

## Channels



This port is shown if the module is attached to a 3D multi-channel object. It allows you to toggle individual channels on or off. Each channel is mapped using a linear intensity ramp. The data window for each channel can be adjusted in the multi-channel object itself.

## Commands

```
frame {0|1}
```

This command lets you turn on or off the orange frame indicating the intersection of the plane with the bounding box of the 3D field.

`setFrameColor <color>`

Lets you change the color of the plane's frame.

`setFrameWidth <width>`

Lets you change the width of the plane's frame in pixels.

`createImage <image base name>`

This command creates a 2D image and adds it to the Pool. The image base name can be omitted.

## 4.6 SelectRoi (LDA)

This module defines a region-of-interest with the shape of an axis-aligned 3D box. This box can be used to restrict the output of many visualization modules like *VoltexLDA*, *OrthoSliceLDA*, and *ObliqueSliceLDA*.

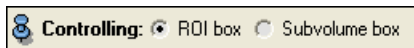
### Connections

#### Data [required]

Connection to a 3D data object which defines the maximum size of the region-of-interest. Only the bounding box of the data object but not the data itself is interpreted by this module.

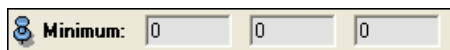
### Ports

#### Controlling



Voxels within the ROI are rendered; voxels outside the ROI are not rendered. The *ROI box* and the *subvolume box* are specified in slice coordinates. The limits are included in the ROI. The region defined by the ROI box always acts upon the region defined by the subvolume, not the entire volume. For example, in *exclusion* mode (see the *Cropping* port), the visible portion of the volume is the subvolume region minus the ROI box region. You are allowed to set the ROI box region larger than (or completely outside) the subvolume region, but only the intersection of the two regions is significant.

#### Minimum



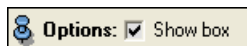
Minimum x-, y-, and z-coordinates of the region-of-interest.

#### Maximum



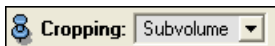
Maximum x-, y-, and z-coordinates of the region-of-interest.

## Options



If the option *show dragger* is enabled, a tab-box dragger is shown which allows you to interactively adjust the region-of-interest in the 3D viewer.

## Cropping

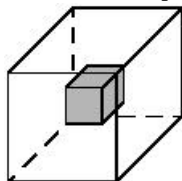


The crop box is defined by three sets of parallel planes that define three slabs:

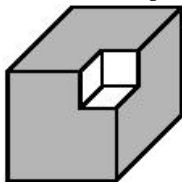
- The xmin and xmax planes define the X slab.
- The ymin and ymax planes define the Y slab.
- The zmin and zmax planes define the Z slab.

Four cropping options are available: *Subvolume*, *Exclusion*, *Cross*, and *Fence*.

*Subvolume* Displays voxels inside the ROI.



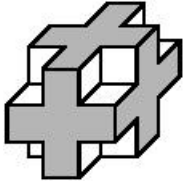
*Exclusion* Displays voxels outside the ROI.



*Fence* Displays voxels between:

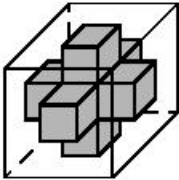
- (Xmin, Xmax), or
- (Ymin, Ymax), or

- (Zmin, Zmax).



Cross Displays voxels between:

- (Xmin, Xmax) and (Ymin, Ymax), or
- (Xmin, Xmax) and (Zmin, Zmax), or
- (Ymin, Ymax) and (Zmin, Zmax).



(Figures courtesy Real Time Visualization)

#### Action



## 4.7 Voltex (LDA)

Direct Volume Rendering is a very intuitive method for visualizing 3D scalar fields. Each point in a data volume is assumed to emit and absorb light. The amount and color of emitted light and the amount of absorption is determined from the scalar data by using a *colormap* which includes alpha values. Then the resulting projection from the points in the data volume is computed. Default colormaps for volume rendering are provided with the distribution and can be edited using the *colormap editor*.

This module provides you with a hardware-accelerated implementation, which uses 2D or 3D texture hardware, to allow for real-time rendering. Using this module on platforms without this acceleration can be extremely slow. Rendering consists of drawing slices from back to front and compositing them

according to the composition port. These slices are textured polygons, whose textures are computed by using the data and the current colormap.

Press the *Apply* button to start the computations necessary to display the volume. Most parameter changes require pressing this button again.

## Connections

### Data [required]

The 3D scalarfield to be visualized. Alternatively an RGBA data volume (Colorfield) can be connected. In this case no colormap is used, but the color and opacity values are taken directly from the data. As a third mode, the module can operate on *multi-channel fields*. Here the transfer function for each channel is computed automatically based on the channel's native color, the channel's data range, and the value of the *Gamma port* (see below).

### ROI [optional]

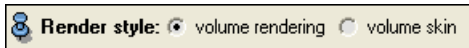
Connection to a module providing a region-of-interest, like *SelectRoi*. If such a module is connected, only the selected part of the volume will be displayed.

### Colormap [optional]

Colormap used to visualize the data.

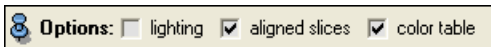
## Ports

### Render style



*Volume rendering* consists of drawing slices from back to front and compositing them according to the *Composition* port. *Volume skin* is a cubic shape made of textured polygons whose textures are computed using the data. Both use the colormap for rendering.

### Options



*Lighting* Indicates if lighting is required. Default is false. Note that activating or deactivating lighting when using 2D/3D texture rendering will usually force the textures to be recreated, which may be slow.

*Aligned slices* Indicates if slices must be drawn in a view-aligned manner. This value is used for 3D texturing only. Default is true.

*Color table* enables a transfer function lookup for monochrome data when *RGBA* lookup mode is on. When this option is activated, only a quarter of the texture memory is needed for *RGBA* rendering and the color table and its range can be modified in real-time (i.e., without pressing the *Apply* button). Note that due to incomplete OpenGL implementations, some graphics boards which

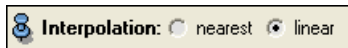
claim to support color tables do not. If you see artifacts or only plain white cubes, disable this option. Default is false.

### Composition



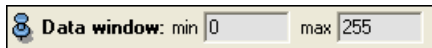
Specifies color composition type. *Alpha* composes the slices by blending the R, G, B components for each pixel based on their alpha values. *Sum* adds slice colors along the viewing axis. (Not available on VolumePro hardware.) *Max* draws the maximum intensity for each pixel drawn along the viewing axis. *Min* draws the minimum intensity for each pixel drawn along the viewing axis.

### Interpolation



Specifies interpolation type (*nearest* value or *linear* interpolation).

### Range



This port is only available if the module operates on a 3D scalar field and no colormap is connected. In this case data values are mapped according to this range. Values smaller than the minimum are mapped to completely transparent (no absorption and no emission). Values larger than the maximum appear completely opaque and emit the maximum amount of light. Values in between are mapped proportionally.

### Lookup



Only available if a colormap is connected. In *Alpha* mode, the colormaps alpha value is used for both absorption and emission. In *LumAlpha* mode, the colormaps alpha value is used for absorption, while the luminance is taken for (uncolored) emission. In *RGBA* mode, colored images are generated by using all four channels of the colormap.

### Colormap



Port to select a colormap.

### Gamma



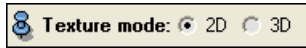
Controls the shape of the transfer function when multi-channel fields are visualized. The opacity value is taken to be  $\alpha = x^\gamma$ ,  $x = 0 \dots 1$  (proportional to data values). The smaller the gamma value is, the more prominent regions with small data values will be.

## Alpha scale



A global factor to change the overall transparency of the object independent of the data value.

## Texture mode



2D texture mode requires some precomputation time but also works on machines which do not support hardware-accelerated 3D texturing, e.g., SGI O2. 3D mode requires less setup time and sometimes provides superior quality on high-end machines.

## Number of slices



Only available in 3D texture mode. The larger this number, the better the image quality and the less the rendering performance.

## Use RGBA



Specifies how to build textures. If not checked, indexed textures will be used if the graphics board supports them. If checked, RGBA textures will be used instead of indexed textures. RGBA textures allow higher precision but are slower to load and use more texture memory. Default is false.

## Commands

`showSlices {0|1}`

If set to 1, the slices used to display the volume are drawn outlined instead of full textured. This mode is mainly useful for debugging.

`setInterpol {0|1}`

Enables or disables linear interpolation of texture values. If linear interpolation is disabled a nearest neighbor lookup is performed. By default, linear interpolation is enabled.

`getInterpol`

Checks if linear texture interpolation is enabled or not.

`setColorTableInterpol {0|1}`

Enables or disables linear interpolation within the color table. The setting will be ignored if color table mode is off or if color table rendering is done using paletted textures. The default value is on.

`getColorTableInterpol`

Checks if linear color table interpolation is enabled or not.

**setColorTableMode** {0|1|2|3|4}

Sets the type of OpenGL extension used for color table mode. The modes are encoded as follows:  
0 = Don't use any extension, turning off color table mode.

1 = GL\_SGI\_texture\_color\_table.

2 = GL\_EXT\_paletted\_texture.

3 = GL\_NV\_fragment\_program.

4 = GL\_ARB\_fragment\_program.

**getColorTableMode**

Returns the type of OpenGL extension used for color table mode.

**doBricking** {0|1}

Enables or disables bricking in 3D texture mode. If bricking is enabled, the volume is rendered in smaller blocks even if its size exceeds predefined size. The predefined size is by default 0.8 times the assumed texture memory size which depends on the hardware and operating system and is obtained by *amira* on program startup. The texture memory size assumed by *amira* can be modified by setting the environment variable AMIRA\_TEXMEM (in megabytes). To alter the predefined size factor of 0.8, one may set a different value in the Tcl variable *voltexTextureAmount*.

**verbose** {0|1}

If value is 1, additional messages for debugging are printed.





# Chapter 5

## Mesh Pack

### 5.1 AlignSurfaces

This module computes a rigid transformation (6 degrees of freedom) to align two triangulated surfaces. Several different alignment strategies are implemented:

- Minimization of the root mean square distance between the points of the model surface to corresponding points on the reference surface (often referred to as Procrustes method): the corresponding points will be the closest points on the reference surface in the Euclidean measure. The iteration of this process is called the iterative closest point algorithm (ICP). If fixed correspondences have been pre-computed by some other method, these can be used alternatively.
- Alignment of the centers of mass: all nodes of the triangulations are assigned the same mass.
- Alignment of the principal axes of the inertia tensor: since the principal axes are defined modulo orientation (see also *AlignPrincipalAxes*), the final solution is the one with the minimum root mean square distance.

### Connections

**Data** [required]

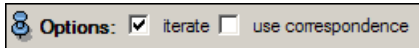
Surface to be transformed onto the reference.

**Reference surface** [required]

Reference surface to be aligned to.

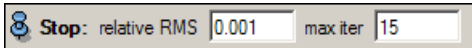
## Ports

### Options



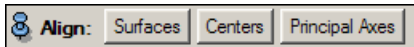
Offers the possibility to iterate the process of finding the closest points on the reference surface (ICP) and to use pre-determined correspondences, if existent.

### Stop



Stopping criteria for the ICP algorithm: either a given value for the root mean square distance relative to its value in the first iteration, or a maximum number of iterations.

### Align



Starts one of the three different alignment methods, as described above.

## 5.2 AnalyticTensorField

This module generates standard analytical symmetric second order tensor fields. You can use the Arithmetic module to sample the tensors on a uniform grid but note that the interpolation method used in Arithmetic does not take into account the metric tensor space. Because of this, the tensors sampled may have negative eigenvalues.

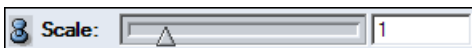
## Ports

### Metric



Selects a tensor metric. The Schwarzschild and the Kerr metrics are used in astronomy to describe the structure of black holes. The Moebius strip metric, the spherical tensor field, and the random tensor field are used primarily as analytical fields to test tensor analysis modules.

### Scale



This port scales the fields with the distance from the center of the data cube. This is useful because the Arithmetic module usually samples fields in the interval -1..1. In the random metric setting this port scales the entries of the tensor entries.

## Mass

A graphical user interface element for the 'Mass' parameter. It consists of a small icon on the left, followed by the text 'Mass:', then a horizontal slider bar with a triangle marker, and finally a text box containing the value '0.1'.

The mass is a parameter of the Schwarzschild and Kerr metrics.

## Angular moment

A graphical user interface element for the 'Angular moment' parameter. It consists of a small icon on the left, followed by the text 'Angular moment:', then a horizontal slider bar with a triangle marker, and finally a text box containing the value '0.6'.

The angular moment used in the Kerr metric.

# 5.3 BoundaryConditions

The *Boundary Conditions* module visualizes boundary conditions defined in a tetrahedral grid for a *finite elements* simulation.

Visible faces are stored in an internal buffer similar to the *GridVolume* module. Likewise, the selection domain can be restricted interactively by adjusting a selection box. Ctrl-clicking on a face makes it invisible.

In amira there are predefined boundary condition ids with special meanings. The meaning of an id depends on the type of simulation to be performed on the grid (E-field or temperature simulation).

- predefined boundary condition ids for E-field simulation
  - id 1: outer boundary ('open' boundary conditions)
  - id 11-59: antenna triangles (metallic boundary conditions)
  - id 61-99: triangles of antenna junctions
- predefined boundary condition ids for temperature simulation
  - id 1: predefined temperature 37°C (Dirichlet condition)
  - id 2: insulating surface (Neumann condition)
  - id 3: boundary to water bolus (Cauchy condition)
  - id 5: boundary to air (Cauchy condition)

## Connections

### Data [required]

The tetrahedral grid to be visualized.

## Ports

### Bound.cond.id

A graphical user interface element for the 'Bound.cond.id' parameter. It consists of a small icon on the left, followed by the text 'Bound.cond.id:', then a dropdown menu with 'All' selected and a downward arrow.

This port provides a menu where all boundary condition ids occurring in the input grid are listed. Faces belonging to the selected id are displayed using a red (highlighted) wireframe in the viewer. You may crop the faces by turning the viewer into interactive mode and move the green handles of the selection box. Only the faces inside the bounding box can be added to the buffer.

### Buffer



The Buffer buttons give you some control of the internal face buffer of the BoundaryConditions module. Only the faces present in the buffer are displayed according to the current drawing style. The *Add* button adds the highlighted faces to the buffer. The *Remove* button removes highlighted faces from the buffer. The *Clear* button removes all faces from the buffer. The *Hide* button deselects all faces but does not change the buffer.

### Draw Style



This option menu lets you select between different drawing styles:

- **Outlined:** The faces are shown together with their edges.
- **Shaded:** The faces are shown.
- **Lines:** The edges of the faces are rendered as a wireframe.
- **Flat:** The edges of the faces are rendered as a wireframe without lighting.
- **Points:** Only the vertices of the faces are rendered as points.

## 5.4 ComponentField

This class is a special data object which helps you to analyze uniform complex scalar fields. The class itself is a uniform real scalar field. However, it can be attached to a complex scalar field like an ordinary display module. The component field provides an option menu allowing you to select which real quantity should be computed from the complex input values. These real quantities may be visualized using standard modules like *OrthoSlice*, *Isosurface*, or *Voltex*.

### Connections

**ComplexField** [required]

Complex scalar field for which a real quantity should be computed.

### Ports

#### Component Field



Specifies the real quantity to be computed. Possible choices are *Real Part*, *Imaginary Part*, *Magnitude*, and *Phase*. The phase is defined in radians ranging from  $-\pi$  to  $\pi$ .

## 5.5 ComputeTensor

This module computes a symmetric second order (diffusion) tensor from a set of  $N$  ( $N \geq 7$ ) diffusion weighted images. The images must be loaded into memory first and are connected as inputs to this module.

The gradient information must be entered into the gradient ports for each attached gradient measurement. The first connected data object is assumed to be the data object without gradient information.

If the number of data objects that need to be loaded is a limiting factor because of limited memory, consider using the *ComputeTensorOutOfCore* module.

### Connections

**S0** [required]

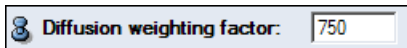
This port must be connected to the data set representing the measurement without Gradient.

**S1,...,S7** [required]

Further ports attached to data will enable a corresponding gradient information port.

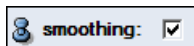
### Ports

#### Diffusion weighting factor

A control panel for the Diffusion weighting factor. It features a blue circular icon with a white 'g' on the left, followed by the text 'Diffusion weighting factor:' and a text input field containing the value '750'.

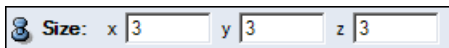
The diffusion weighting factor used in this study.

#### smoothing

A control panel for the smoothing option. It features a blue circular icon with a white 'g' on the left, followed by the text 'smoothing:' and a checked checkbox.

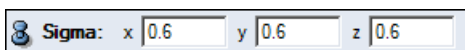
If enabled, two more ports will allow you to request a Gaussian filtering of the raw data prior to the computation of the tensor field.

#### Size

A control panel for the Gaussian kernel size. It features a blue circular icon with a white 'g' on the left, followed by the text 'Size:'. To the right are three text input fields labeled 'x', 'y', and 'z', each containing the value '3'.

Specify the size of the Gaussian kernel used for smoothing.

#### Sigma

A control panel for the Gaussian filter variance. It features a blue circular icon with a white 'g' on the left, followed by the text 'Sigma:'. To the right are three text input fields labeled 'x', 'y', and 'z', each containing the value '0.6'.

Specify the variance used by the Gaussian filter for smoothing.

G1,G2,...

|                                                                                       |                                |                                |                                |
|---------------------------------------------------------------------------------------|--------------------------------|--------------------------------|--------------------------------|
|  G1: | <input type="text" value="1"/> | <input type="text" value="0"/> | <input type="text" value="1"/> |
|---------------------------------------------------------------------------------------|--------------------------------|--------------------------------|--------------------------------|

Port allows you to supply a user defined Gradient which is used for the computation of the diffusion tensor. The order of these ports corresponds to the order of the data objects connected to the module.

## 5.6 ComputeTensorOutOfCore

This module computes a symmetric second order (diffusion) tensor from a set of  $N$  ( $N_i=7$ ) diffusion weighted images. The images must be available on disk as AmiraMesh files. The module will load them in using amira's LargeDiskData format, which can handle data sizes above the maximum size of available main memory.

The gradient measurements and the gradients for a data set are specified by the parameter section of an AmiraMesh file.

```
# AmiraMesh3D BINARY 2.0

# define a dummy lattice to make this AmiraMesh valid
define Lattice 1 1 1
Lattice { byte Data } @1
@1
a

Parameters {
  BoundingBox -1 1 -1 1 -1 1,
  CoordType "uniform",
  Data {
    directory "C:/AmiraData/",
    gradients "MyGradients",
    # file0 is always the b=0 measurement (no gradient)
    file0 {
      name "201/Anonymized.am",
    }
    # fileN are entries for each gradient measurement
    file1 {
      name "201/Anonymized2.am",
      # use the gradient in the gradient list with index 0
      gradient "0"
    }
    ...
  }
}
```

```

MyGradients
  1 0 0
  0 1 0
  ...
}

```

The module creates a tensor data set and two additional fields that contain statistics about the quality of the tensor fit. The correlation field contains the correlation of the tensor diffusion values with the diffusion values in the raw data (per gradient direction). A large value indicates that the tensor fits the raw data well. The Bingham statistics ensure that the diffusion tensor is statistically different from a uniform distribution (see Bengner et al. "Visualizing Neuronal Structures in the Human Brain via Diffusion Tensor MRI", Journal of Neuroscience, 2006).

## Connections

### Specification [required]

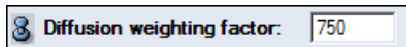
Connect an AmiraMesh file which contains the specification for the computation in its parameter section.

### Mask [optional]

Connect a scalar field like a label field to specify for which material the tensors should be computed. This will remove the undefined tensors in parts of the data.

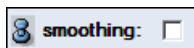
## Ports

### Diffusion weighting factor


 A port control for the Diffusion weighting factor. It features a blue circular icon with a white 'g' on the left, followed by the text 'Diffusion weighting factor:' and a text input field containing the value '750'.

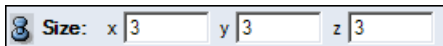
Diffusion weighted factor (b-value, after LeBihan et al., 1986) which is defined by the gradient sequence used to generate the gradient images. In this version only a single b-value can be assigned to all volumes of the measurement sequence.

### Smoothing


 A port control for the Smoothing option. It features a blue circular icon with a white 'g' on the left, followed by the text 'smoothing:' and a checkbox that is currently unchecked.

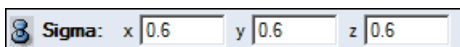
If checked, smoothing is enabled.

### Size


 A port control for the Size parameters. It features a blue circular icon with a white 'g' on the left, followed by the text 'Size:'. To the right are three text input fields labeled 'x', 'y', and 'z', each containing the value '3'.

This port is only visible if smoothing is enabled.

### Sigma


 A port control for the Sigma parameters. It features a blue circular icon with a white 'g' on the left, followed by the text 'Sigma:'. To the right are three text input fields labeled 'x', 'y', and 'z', each containing the value '0.6'.



This port is only visible if smoothing is enabled. It defines the variance of the Gaussian filter kernel. Large values mean large smoothing.

### MaskExpr



An expression to select a specific region in the data. This port is only visible if a scalar field is connected to the Mask port. If a label field is connected, you can select a material by using an expression like  $A==1$  which would select the first non-exterior material region for the tensor computation.

## 5.7 ConePlot

ConePlot is a display module that can be attached to a vector field. It visualizes scalar and complex vector fields by colored objects (cones) pointing in the direction of the local flow. The cones can be animated to generate a sequence of cones "walking" through the vector field.

The module features SelectRoi, Colorfields, and arbitrary shapes to be displayed instead of cones.

The module can also generate a density based on the cone positions over time.

```
ConePlot setGhostDims 30 30 30
```

```
ConePlot generateGhost
```

The module uses a Gaussian kernel with a variance of a tenth of the maximum bounding box size of the volume. This option is computationally very expensive.

The module also exports the path of the cones as a LineSet. To generate the LineSet, call:

```
ConePlot saveAsLineSet
```

in the *amira* console window. The LineSet is computed with a data entry per LineSet point which represent the magnitude of the vector at this position.

ConePlot uses Open Inventor render caching. All animation steps are computed beforehand and then played back by making the appropriate parts of the scene visible. This technique requires little CPU computation but larger amounts of memory and a fast graphics card. Because of the caching step, the first pass through the animation may be slow but subsequently render passes speed up.

Press the *Apply* button to recompute the animation or the single plot.

### Connections

#### Data [required]

The 3D vector field to be visualized. It can be either 3 or 6 floats per voxel. This module works therefore with arbitrary grid topologies by sampling them uniformly.

#### Colormap [optional]

Port to connect a colormap object. The color is computed from the magnitude of the vectors.

### Animate [optional]

Attach a Time object to control the *Animate* slider. Please make sure that the range of the time slider is set to integers and that 0 is within that range.

### ROI [optional]

Connect a region of interest to the module and the cones will only be generated inside this region. This helps in exploring complicated vector fields.

### Colorfield [optional]

Instead of the vector field applied, the module will use a second scalar field to color the cones.

### Shape data [optional]

Supply a custom shape to be replicated instead of cones. Basically any geometry displayed in amira can be used. Be careful not to use very complex objects because the number of triangles used in this case can be a limiting factor for the speed of the animation.

## Ports

### Resolution



Provides three text inputs defining the resolution of the regular array of cones in the local  $x$ -  $y$ - and  $z$ -direction. The larger these values are, the more cones are displayed.

### Colormap



Port to select a colormap. The color is computed from the magnitude of the vectors.

### Complex phase angle



This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controlling which part of the complex 3D vectors is visualized by the arrows.

$$y(t) = \cos(\text{phase})\text{Re}(x(t)) + \sin(\text{phase})\text{Imag}(x(t)) \quad (5.1)$$

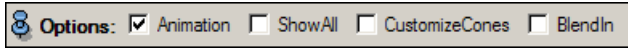
A value of 0 degree corresponds to the real part, while a value of 180 degrees corresponds to the imaginary part. The display can be animated with respect to the phase by the *Animate* port. This way polarization properties of the field can be revealed or wave phenomena become visible.

## Threshold



Any cone is removed from display which would be placed on a position with a vector length smaller than this threshold. Using this option you can "thin-out" parts of the volume, e.g., regions which would contain cones that never move.

## Options



This port provides the following toggle options.

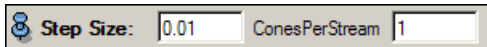
*Animation:* If this option is set, then two additional ports are displayed. The *Time* port allows you to set a time frame (from  $-m > 0$  to  $n$  frames). The *Step size* port allows you to set the time between successive time steps. Currently we use an Euler integration. The smaller the step size, the more accurately the flow field is sampled.

*Show All:* Shows all cones in the current animation. When switching on this mode, there is no need for animations because the animation is done by switching cones on and off.

*Customize Cones:* You can change the height and the bottom radius of all cones.

*BlendIn:* If this option is set, the size of the cones is increased or decreased at the start or end point of the animation. This behavior generates nicer graphics for continuous animations, especially in the case of  $\text{ConesPerStream} > 1$ .

## Step Size



The current implementation uses Euler integration and this defines the step size calculated by

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \cdot \mathbf{v}(\mathbf{x}(t), t),$$

where  $\mathbf{x}(t)$  is the position and  $\mathbf{v}(t)$  the velocity at time  $t$ . Please be careful to not work on stiff flow fields using this method. In general, small values result in higher accuracy and smaller path length of the cones.

If you set the value of *ConesPerStream* to  $> 1$ , more than one cone is displayed at a time. Together with the loop mode of the time slider and the *BlendIn* option set, this creates continuous animations.

## Animate



This time slider allows you to start and stop animations. If you right-click in the associated text window you can switch to loop mode which generates an infinitely long animation.

The range of the slider must be set to integer numbers. For example:  $0 \dots 50$  is interpreted as 50 time steps for the forward integration, or  $-30 \dots 30$  is interpreted as 30 steps in both forward and backward directions. The module makes sure that the increment is always 1.

### Height



For cones this changes their height. If another shape is connected to the module, the port controls the global scaling of the shape.

### Bottom radius



For cones this changes the radius of their base circle.

## 5.8 ContourView

This module displays contours of a surface. Contours are typically computed automatically and they are defined to be the boundaries of patches. If a surface contains no contours, this module will not display anything. You can automatically compute contours by using the `recompute` command of the *Surface*.

This module is mostly useful for developers.

### Connections

**Data** [required]

The underlying surface data.

### Ports

**Index)**



Choose index of contour to display. If -1 is chosen, all contours will be shown.

## 5.9 Displace

This module takes a displacement vector field defined on a tetrahedral or hexahedral grid or on a surface and creates a mesh with translated vertices. For example, if a surface vector field is used as input a new surface with modified vertex coordinates is computed. The displacement vector field must be defined on the vertices of the mesh. Other encodings are not supported. The new position of a vertex

is computed by adding the scaled displacement vector defined at that vertex to the original position. The scaling factor can be adjusted globally for all vertices using the *scale port*.

Press the *Apply* button to start the computation.

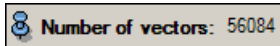
## Connections

**Data** [required]

The input vector field.

## Ports

**Number of vectors**



Shows the number of displacement vectors of the input field.

**Scale**



Scaling factor applied to the displacement vectors ranging from 0 (no displacement) to 1 (full displacement).

## 5.10 DisplayISL

This module visualizes a 3D vector field using so-called illuminated field lines. This technique was first presented on the *Visualization 96* conference. More details are described in *M. Zöckler, D. Stalling, H.-C. Hege, Interactive Visualization of 3D-Vector Fields using Illuminated Streamlines, Proc. IEEE Visualization '96, Oct./Nov. 1996, San Francisco, pp. 107-113, 1996.*

The module computes a large number of field lines by integrating the vector field starting from random seed points. The lines are displayed using a special illumination technique, which gives a much better spatial understanding of the fields structure than ordinary constant-colored lines. In order to execute a script demonstrating this module click [here](#).

The functionality of *DisplayISL* can be extended by means of the *SeedSurface* module.

To initiate distribution of seeds and recomputation of field lines, press the *Apply* button. Once the incoming vector field has changed or you have modified the number of field lines or the line's length, you must press *Apply* again in order to update the display.

## Connections

**Data** [required]

The vector field to be visualized. Since a procedural data interface is used, all types of vector fields are supported (e.g., fields on regular or curvilinear grids, as well as procedurally defined fields).

### ColorField [optional]

An optional scalar field which, if present, will be used for pseudo-coloring. Often it is useful to create a scalar field from the vector field using the *Magnitude* module.

### AlphaField [optional]

An optional scalar field which can be used to control the lines' opacity locally.

### Distribution [optional]

An optional scalar field which can be used to control the distribution of seed points.

### Colormap [optional]

The *colormap* used to encode the color field. Will only be visible if a color field is connected to this module.

## Ports

### Colormap



Optional colormap.

### Num Lines



Number of field lines to be displayed. The technique typically gives the best results with a rather large number of lines (e.g., 100-500). Note however that on systems with slow 3D graphics without texture hardware, having large number of lines can slow down rendering speed significantly.

### Length



The length of the field lines, or more precisely, the number of atomic line segments, in forward respectively backward direction. The lines may stop earlier if a singularity (i.e. zero magnitude) is encountered or if the field's domain is left. By default lines are traced the same distance in the forward and backward direction. You may use the command `setBalance` to change this behavior.

### Opacity



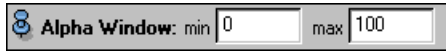
Base opacity factor of the lines. A value of 1 produces completely opaque lines, while 0 results in fully transparent lines.

### Fade Factor



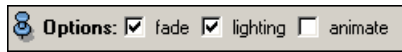
This port controls how fast opacity decreases along field lines if *fade mode* is enabled. The first atomic line segment will be assigned the base opacity set in port *Opacity*. The opacity of each successive segment will be multiplied by the value of this port. This is useful in order to encode the directional sign of a vector field. The vector field points in the direction of increasing transparency.

### Alpha Window



This port will only be visible if *fade mode* is disabled and if an alpha field is connected to the module. In this case the alpha field's values are mapped to opacity according to the range specified by this port. At locations where the alpha field is equal to or smaller than *min*, field lines will be completely transparent. Likewise, at locations where the alpha field is equal to or greater than *max*, field lines will be completely opaque.

### Options

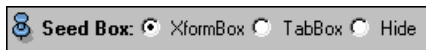


*Fade*: Enables fading mode as described above.

*Lighting*: Controls illumination of lines. If off, constant colored lines are drawn (flat shading).

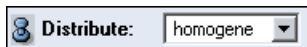
*Animate*: Activates particle-like animation. The animation speed may be controlled via the command `setAnimationSpeed`.

### Seed Box



Clicking on *XformBox* or *TabBox* brings up a 3D dragger in the 3D Viewer. This allows you to restrict the region of interest, i.e. the region in which seed points are placed, by interactively transforming the dragger (Remember to switch the viewer into interaction mode by pressing ESC). After changing the box, you must press *Apply* to trigger recalculation.

### Distribute



This port provides an option menu specifying how seed points are distributed inside the seed box. By default a *homogeneous* distribution will be used. Alternatively, seed points may be distributed according to the vector field's magnitude or according to the value of the distribution field, if such a field is connected to the module. The *equalize* option provides a mixture of homogeneous and proportional seed point distribution.

## 5.11 Divergence

The *Divergence* module computes the divergence of a vector field consisting of floats defined on a uniform grid. The output is a uniform scalar field.

$$\text{div}V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$$

Press the *Apply* button to start the computation.

## Connections

### Data [required]

Vector field defined on a uniform grid (*UniformVectorField3*). The vector components must be floats.

## 5.12 DoseVolume (Tetrahedra)

The *Dose Volume* module creates a value-volume-histogram for a scalar field defined on a *tetrahedral grid*, e.g. a temperature-volume-histogram for a scalar field that represents a temperature distribution. For this purpose it has to be attached to a *GridVolume* module and a selection of tetrahedra specifying parts of the grid or the complete grid must be made there. A histogram is constructed for the selected tetrahedra only. This is done in the following way:

The range of scalar values is divided into subranges given by the number of sample points. With respect to each subrange one or more sub-volumes of the tetrahedral grid are determined where the scalar field values fall into that range. Then the volumetric sizes of the sub-volumes are computed and these are depicted as differential histogram function values over the total range of scalar field values or used to calculate cumulative histogram values.

If the selected tetrahedra are members of different regions, histograms are calculated separately for each region, as well as for all selected tetrahedra (called 'total Volume').

The histograms can be shown in several ways:

- **differential:** shows a differential form of the histogram.
- **absolute** and **relative:** show an integral form of the histogram, i.e., for each value the partial volume is shown where *at least* that value is assumed. For the integral form either the **absolute** volume [ccm] or the **relative** volume, i.e., the percentage of the total volume of the corresponding region, can be shown. With the latter type of representation you can easily find out which value is at least assumed in 90 percent of a selected region.

## Connections

### Data [required]

A scalar field defined on a tetrahedral grid.

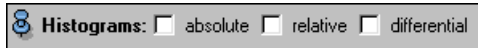


## PortGridVol [required]

A *GridVolume* module that selects the tetrahedra for which the histogram is calculated.

### Ports

#### Histograms



If you select one of these toggles, a plot window appears showing the corresponding histogram for all tetrahedra selected by the *GridVolume* module. If no tetrahedra have been selected, the plot window will not be shown. For the three different modes of representation see above.

#### Samples



This slider lets you select the number of samples for the histogram.

## 5.13 DuplicateNodes

This module takes a labeled tetrahedral grid as input and produces a new grid with all nodes on *interior boundaries* being duplicated. Interior boundaries can be visualized using module *GridBoundary*. They consist of all triangles incident on two tetrahedra with different labels. Duplicated nodes are useful to represent discontinuities within a vector field, e.g., an electric field on a tetrahedral patient model. On an interior boundary such a field may have multiple values - one for each incident material subvolume.

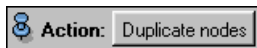
### Connections

#### Data [required]

A tetrahedral grid without duplicated nodes.

### Ports

#### Action



Button *duplicate nodes* causes a new output grid with duplicated nodes to be computed.

## 5.14 EigenvectorToColor

This module creates a color representation of the connected vector field. It also needs to be connected to a second vector field which first component will be used as a weighting of the color per voxel.

Connected to the first eigenvector of a tensor field and a vector field representing the eigenvalues (both produced by *ExtractEigenvalues*) this module will generate a color representation which describes the directions of the first principal direction of the tensor field by a color code.

The output of this field is a regular color field. It contains in its components:

(red)  $|e_1(x)|FA$

(green)  $|e_1(y)|FA$

(blue)  $|e_1(z)|FA$

(alpha) the fractional anisotropy  $FA = \frac{1}{\sqrt{2}} \frac{\sqrt{(\lambda_1 - \lambda_2)^2 + (\lambda_2 - \lambda_3)^2 + (\lambda_1 - \lambda_3)^2}}{\lambda_1 + \lambda_2 + \lambda_3}$

The alpha channel is scaled from 0 to 255 as well as the three element vector containing the red, green, and blue component.

## Connections

**Data** [required]

The largest eigenvector (pca1) created by *Eigenvector2Image*.

**Eigenvalues** [required]

The length sorted eigenvalues of the tensor (a 3 element field).

**Mask** [optional]

A label field which can be used in order to calculate the color data for one or more selected materials. For example, one can exclude the background from the computation. See [5.14](#).

## Ports

**MaskExpr**



This field is only visible if a mask label set is connected to the data. The expression selects a specific material or a group of materials from the connected label set. Example are: "A!=0" to exclude the exterior or "A==3" to select the forth material from the material list of the Segmentation Editor display of the mask.

**Exposure**



Controls the exposure  $e$  of the color representation. Larger values will make the images brighter. It is computed as:

$$C_{r=x,g=y,b=z} = (1 - \exp(-|e_1(x, y, z)|FA) e) 255 \quad (5.2)$$

where  $C$  is the value assigned to the color component,  $e_1$  is the connected first principal direction (pca1) and  $FA$  describes the fractional anisotropy computed from the connected eigenvalues.

## 5.15 ExtractEigenvalues

This module extracts the eigenvalues from a symmetric tensor of second order. Either the eigenvalues are returned (in the case of a indefinite tensor) or a singular value decomposition is used to ensure a positive definite basis.

The result are three vector fields which contain the first, second, and third principal direction and a three-component field which contains for each direction the corresponding eigenvalue.

### Connections

**Data** [required]

The symmetric tensor field of second order.

**Mask** [optional]

If you connect a scalar field here, you can specify for which region in the data the module computes the eigenvalue expansion.

### Ports

#### Options



If *positive definite* is checked, a singular value decomposition ensures a positive definite basis (for diffusion tensors). If the checkbox is disabled, an eigenvalue decomposition is performed which may result in negative eigenvalues.

#### Output



You can specify which output fields should be exported by the module. Note that only the first three fields are vector fields; the *evals* field contains the eigenvalues sorted according to size.

#### MaskExpr



If you connect a scalar field to the *Mask* input port, this port allows you to specify a region of interest based on the value of the scalar field. If you connect a label field, an expression like  $A==1$  will select the first non-exterior material and only for this region the eigenvalues and eigenvectors will be computed.

## 5.16 FiberTracking

This modules implements the tensorline algorithm and computes line sets from vector fields by numerical integration. The input vector fields may represent the first principal direction of a tensor field and

its Eigenvalues (see *ComputeTensor*, *ComputeTensorOutOfCore* and *ExtractEigenvalues*). The algorithms implemented for the fiber tracking are streamline tracking (STT) and tensor deflection (TEND). The user may choose between a combination of both methods. The module takes care of the ambiguity of the sign of the Eigenvector solution and uses a fourth order Runge-Kutta integration method.

The module requires a label field to be present. As many separate line set objects are produced as there are non-background materials in the label field.

Note that the number of line segments may be very large. The *SelectLines* module may help in this case to focus the analysis to a sub-region. Also selecting a higher threshold for the fractional anisotropy will reduce the number of lines produced.

## Connections

### Data [required]

A vector field usually representing the first principal direction of the vector field. If this module is connected to the input the module will search the workspace for a second data set with a similar name (icon name ending 'evals') and connect it to the Eigenvalue port.

### Mask [required]

A label field which is used to produce separate line sets for each material.

### Tensor [optional]

A symmetric second order tensor which is needed to perform the tensor diffusion algorithm.

### Eigenvalues [required]

The Eigenvalues which belong to the Eigenvector solution of the vector field input. These values are used to compute the fractional anisotropy (threshold value).

## Ports

### Info

 **Info:** STT weight f, (1-f)g weight TEND, (1-f)(1-g) weight v\_in

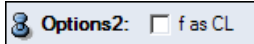
Port printing information about the expected number of integration steps using the currently selected values for the integration. The second info port explains the implementation of the two parameters f and g which weight solutions of the STT and the TEND algorithm.

### Options

 **Options:** ☒ local alignment ☒ FA threshold ☒ direction threshold

Port to specify if the module should guess the sign of the vector (local alignment), if the fractional anisotropy threshold should be used to stop the integration (FA threshold) and if the module should check for how straight the path of integration is (direction threshold).

## Options2

A control panel with a blue icon on the left, the text "Options2:", a checkbox, and the text "f as CL".

Options2: ☐ f as CL

Port which when enabled replaces the weighting parameter  $f$  by the linear tensor shape measure for each voxel.

### FA threshold

A control panel with a blue icon on the left, the text "FA threshold:", a slider, and a text box containing "0.1".

FA threshold:  0.1

Port to specify the threshold for the fractional anisotropy. If the FA value for the current voxel is below the threshold the integration scheme will stop.

### Tensor weight g

A control panel with a blue icon on the left, the text "Tensor weight g:", a slider, and a text box containing "1".

Tensor weight g:  1

Port to specify the weighting between STT and TEND fiber tracking.

The weighting factors  $f$  and  $g$  should vary between 0 and 1. The algorithm is using three directional terms. The first is the vectors found in the data set in the data port (weighted by  $f$ ), the second is the current direction of travel (weighted by  $(1-f)(1-g)$ ) and the third is the tensor deflection term (weighted by  $(1-f)g$ ).

### Tensor weight f

A control panel with a blue icon on the left, the text "Tensor weight f:", a slider, and a text box containing "1".

Tensor weight f:  1

Port to specify the weighting between STT and TEND fiber tracking.

### Direc threshold

A control panel with a blue icon on the left, the text "Direc threshold:", a slider, and a text box containing "90".

Direc threshold:  90

The angle at which the fiber tracking is stopped.

### Step size

A control panel with a blue icon on the left, the text "Step size:", a slider, and a text box containing "0.1".

Step size:  0.1

The step size of the fiber tracking in units of the voxel size.

### Length

A control panel with a blue icon on the left, the text "Length:", a slider, and a text box containing "10".

Length:  10

The length of the fiber tracked in units of the voxel size.

## 5.17 FieldCut

The *Field Cut* module is a tool for visualizing cross sections of a 3-dimensional scalar field defined on a tetrahedral grid. In general such grid volumes are made up of several materials. A (2-dimensional)

cutting plane has to be defined for this purpose and may be shifted through the (3-dimensional) tetrahedral grid in orthogonal direction, an *Orientation* port is provided for setting the orientation of the cutting plane perpendicular to one of the main axes and a *Translate* port for specifying an orthogonal translation. *FieldCut* renders a slice with the values of the scalar field mapped to colors of a connected colormap. This is called *pseudo coloring*. The cutting plane is usually clipped to a rectangle. With the 3D viewer turned into interactive mode you can pick one of its edges and shift it through the grid volume. To define arbitrary cutting planes you have to set the *rotate* toggle which makes a rotation handle appear in the center of the cutting plane. Having turned the viewer into interactive mode you can pick the handle with the left mouse button and rotate the plane as you please.

## Connections

### Data [required]

The field of type *TetrahedralGrid*.

### Colormap [optional]

An optional colormap used for pseudocoloring the values of the scalar field on the cutting plane.

## Ports

### Orientation



This port provides three buttons to reset the slice orientation. Axial slices are perpendicular to the z-axis, frontal slices are perpendicular to the y-axis, and sagittal slices are perpendicular to the x-axis. See also *PortButtonList*.

### Options



If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

### Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

### Colormap



See also *PortColormap*.

## Interpolation



In pseudo-color mode you can switch between two rendering methods by the radio buttons *Gouraud* and *Texture*. See also *PortRadioBox*.

- **Gouraud** shaded faces have their colors assigned at their vertices. The colors in between are linearly interpolated. Therefore misleading colors occasionally may occur or colors may appear to be washed out along the edges of the faces.
- **Texture** mapping means an exact mapping of values to colors in this special case, but with the drawback of increased computing time if you have no hardware supported for texture mapping on your machine.

## Selection



This port maintains a list of materials assigned for cutting. With the selection menu one can select a single material. The *Add* button adds a previously selected material to the list and the *Remove* button removes the current selected material.

## 5.18 FieldCut (Polyhedra)

The *Field Cut* module is a tool for visualizing cross sections of a 3-dimensional scalar field defined on a polyhedral grid. In general such grid volumes are made up of several materials. A (2-dimensional) cutting plane must be defined for this purpose and may be shifted through the (3-dimensional) polyhedral grid along the x, y, or z axis, an *Orientation* port is provided for setting the orientation of the cutting plane perpendicular to one of the axes, and a *Translate* port for specifying a slice along the selected orientation axis. *FieldCut* renders a slice with the values of the scalar field mapped to colors of a connected colormap. This is called *pseudo coloring*. The cutting plane is usually clipped to a rectangle. With the 3D viewer in interactive mode, you can pick one of its edges and shift it through the grid volume. To define arbitrary cutting planes, you must set the *rotate* toggle which makes a rotation handle appear in the center of the cutting plane. With the viewer in interactive mode, you can pick the handle with the left mouse button and rotate the plane as desired

## Connections

**Data** [required]

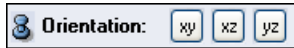
The field of type *PolyhedralGrid*.

**Colormap** [optional]

An optional colormap used for pseudocoloring the values of the scalar field on the cutting plane.

## Ports

### Orientation



This port provides three buttons to reset the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis. See also *PortButtonList*.

### Options



If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set, the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button. When *fit to points* is checked, you will be asked to pick 3 points in the scene. The slice plane will be updated to intersect the specified points.

### Translate



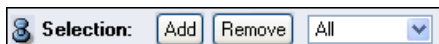
This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

### Colormap



See also *PortColormap*.

### Selection



This port maintains a list of materials assigned for cutting. With the selection menu one can select a single material. The *Add* button adds a previously selected material to the list and the *Remove* button removes the currently selected material.

## 5.19 Gradient

The *Gradient* module computes the gradient of a scalar field defined on a regular (i.e. uniform, rectilinear, or curvilinear) grid. The output is a vector field defined on an appropriate grid. The direction of the gradient vector depends on the setting of port *Output*. If *Force* is selected the negative gradient vector is computed.



$$\text{grad}F = \pm \left( \frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)$$

For non-uniform regular grids the gradient at vertex  $v$  is the mean of the left hand and right hand difference quotient at vertex  $v$ .

Press the *Apply* button to start the computation.

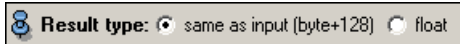
## Connections

**Data** [required]

Scalar field defined on a regular grid (*HxRegScalarField3*).

## Ports

**Result type**



Selection of the result type. Depending on the type the result may be clamped. For unsigned types an appropriate offset will be added. This port is only shown if the input type is not float.

**Output**



If *Force* is selected the negative gradient vector is computed.

## 5.20 GridBoundary

The *Grid Boundary* module is a tool for visualizing individual faces of a tetrahedral grid. The faces may be colored according to an arbitrary scalar field. As the name implies, the module extracts boundaries between tetrahedra of different material type. The particular materials to be shown can be selected manually. In some cases two materials may have no common faces and nothing will be seen. However, selecting *All* as the first material parameter and any other material as the second will show the surface of the second material. Visible faces are stored in an internal buffer similar to the *GridVolume* module. Likewise, the selection domain can be restricted interactively by adjusting a selection box. Ctrl-clicking on a face makes it invisible.

If the *ColorField* port is connected to a scalar field, an additional colormap port becomes visible and the faces are drawn in pseudo-color mode. By default the colormap port is not connected to any colormap. Therefore the grid appears in a constant color. You can click with the right mouse button over the colormap to get a popup menu of all available colormaps. When a colormap has been connected to the module, the data values at the vertices of the selected triangles are mapped to their associated colors.

## Connections

### Data [required]

The *tetrahedral grid* to be visualized.

### ColorField [optional]

Scalar field used for pseudo-coloring the selected triangles. Pseudo-coloring also requires that a *colormap* has been connected to the module.

### Colormap [optional]

A colormap is used to map the data values of the optional scalar field.

### ROI [optional]

Optional connection to an object providing a region-of-interest, e.g., *SelectRoi*. Only triangles inside this region will be visualized.

## Ports

### Draw Style



This port is inherited from the *ViewBase* class and therefore the description will be found there. In contrast to other modules derived from the *ViewBase* class this module does not provide the possibility to consider vertex and direct normals. The triangles will always be drawn flat.

### Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

### Buffer



The Buffer buttons give you some control of the internal face buffer of the *GridBoundary* module. Only the faces present in the buffer are displayed according to the current drawing style. The *Add* button adds the highlighted faces to the buffer. The *Remove* button removes highlighted faces from the buffer. The *Clear* button removes all faces from the buffer. The *Hide* button deselects all faces but does not change the buffer. See also *PortButtonList*.

### Materials



This port provides two menus where all different materials of the input grid are listed. Faces which

belong to the boundary between the selected materials are displayed using a red (highlighted) wire-frame in the viewer. You may crop the faces by turning the viewer into interactive mode and move the green handles of the selection box. Only the faces inside the bounding box can be added to the buffer, also see *PortButtonList*.

### Color Mode



Here you may choose among several color modes. For details see the module *SurfaceView*.

## 5.21 GridCut

The *GridCut* module is a tool for visualizing a cross section of a tetrahedral grid consisting of different materials. Within the cross section the different materials are indicated by their respective colors. The module is derived from *ArbitraryCut* and thus provides the same methods for manipulating the position and orientation of the cross section as this base class. An similar module *SurfaceCut* exists for displaying filled cross-sections through a surface separating different regions in space.

### Connections

**Data** [required]

The tetrahedral grid to be visualized.

### Ports

#### Orientation



This port provides three buttons to specify the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

#### Options



If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch on the rotate handle for the cutting plane and off again. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

#### Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

### Selection



This port maintains a list of materials to be displayed within the cross section. With the selection menu one can select a single material. The *Add* button adds the currently selected material to the list so that it becomes visible and the *Remove* button removes the material so that it becomes invisible.

## Commands

Inherits all commands of *ArbitraryCut*.

`selectMaterial <id1> [<id2> ...]`

Selects the materials with the specified ids so that intersections of these materials with the cutting plane will be shown. You need to call `fire` before changes take effect.

`unselectMaterial <id1> [<id2> ...]`

Unselects the materials with the specified ids so that intersections of these materials with the cutting plane will not be shown. You need to call `fire` before changes take effect.

## 5.22 GridCut (Polyhedra)

The *GridCut* module is a tool for visualizing a cross section of a polyhedral grid consisting of different materials. Within the cross section the different materials are indicated by their respective colors. The module is derived from *ArbitraryCut* and thus provides the same methods for manipulating the position and orientation of the cross section as this base class.

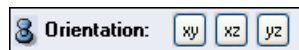
### Connections

**Data** [required]

The polyhedral grid to be visualized.

### Ports

#### Orientation



This port provides three buttons to specify the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

## Options



If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off again. If the *immediate* toggle is set, the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button. When *fit to points* is checked, you will be asked to pick 3 points in the scene. The slice plane will be updated to intersect the specified points.

## Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

## Selection



This port maintains a list of materials to be displayed within the cross section. With the selection menu one can select a single material. The *Add* button adds the currently selected material to the list so that it becomes visible and the *Remove* button removes the material so that it becomes invisible.

## Commands

Inherits all commands of *ArbitraryCut*.

## 5.23 GridVolume

The *GridVolume* module is a powerful tool for visualizing labeled tetrahedral volume grids, e.g., tetrahedral patient models. Views on various sub-grids can be specified, e.g., a view on a sub-grid corresponding to one of the tissue compartments of a patient model. The module maintains an internal buffer which contains all currently visible tetrahedra. Tetrahedra belonging to a particular compartment can be selected and added to the buffer. Selected tetrahedra are displayed using a red wireframe. Having been added to the buffer they are displayed in their associated colors. The selection can be restricted by means of an adjustable selection box. In addition, individual tetrahedra can be removed from the buffer by Ctrl-clicking on one of their faces, but notice that the viewer has to be turned into interactive mode for this. Similarly, a simple click on a face adds the tetrahedra in front of it to the buffer.

## Connections

### Data [required]

The labeled *tetrahedral grid* to be visualized.

### ColorField [optional]

Arbitrary scalar field which is mapped onto the grid volume using pseudo-coloring.

### Colormap [optional]

The colormap is used for pseudo-coloring the grid volume.

### ROI [optional]

Optional connection to an object providing a region-of-interest, e.g., *SelectRoi*. Only triangles inside this region will be visualized.

## Ports

### Draw Style



This port is inherited from the *ViewBase* class and therefore the description will be found there. In contrast to other modules derived from the *ViewBase* class this module does not provide the possibility to consider vertex and direct normals. The triangles will always be drawn flat.

### Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

### Buffer



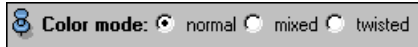
This port lets you *add* and *remove* highlighted triangles (being displayed in red wireframe) to an internal buffer. For a further description and for the functionality of each of the port buttons see *ViewBase*.

### Materials



This port provides a menu where all materials of the input grid are listed. If you choose a material, all tetrahedra of that material are selected and displayed using a red wireframe model. You may crop the selection domain by turning the viewer into interactive mode and moving the green handles of the bounding box to a position of your choice. Only the tetrahedra inside the bounding box can be added to the buffer.

## Color mode



Here you may choose among several color modes. For details see the module *SurfaceView*.

## Commands

### `getSelectedTetra`

Displays a run-length encoded list of all currently selected tetrahedra. The list consists of pairs of integer numbers. The first number is the index of a selected tetrahedron. The second value denotes the number of subsequent selected tetrahedra.

### `setSelectedTetra <list>`

Adds some tetrahedra to the internal buffer so that they become visible. The argument is a run-length encoded list like the one displayed by `getSelectedTetra`.

## 5.24 GridVolume (Polyhedra)

The *GridVolume* module is a powerful tool for visualizing labeled polyhedral volume grids, e.g., polyhedral patient models. Views on various sub-grids can be specified, e.g., a view on a sub-grid corresponding to one of the tissue compartments of a patient model. The module maintains an internal buffer which contains all currently visible polyhedra. Polyhedra belonging to a particular compartment can be selected and added to the buffer.

## Connections

### **Data** [required]

The labeled *polyhedral grid* to be visualized.

### **ColorField** [optional]

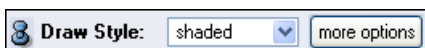
Arbitrary scalar field which is mapped onto the grid volume using pseudo-coloring.

### **Colormap** [optional]

The colormap is used for pseudo-coloring the grid volume.

## Ports

### Draw Style



This port is inherited from the *ViewBase* class and therefore its description will be found there.

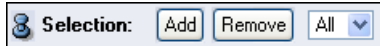
In contrast to other modules derived from the *ViewBase* class this module does not provide the possibility to consider vertex and direct normals. The triangles will always be drawn flat.

### Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

### Selection



This port provides a menu where all materials of the input grid are listed. It lets you *add* or *remove* the selected material skin to the scene.

### Color Mode



This port is available only on meshes with data located on cells. It allows you to change the color mapping of the dataset on the geometry.

- **Constant:** Constant color.
- **Average:** Each cell is colored by a single color defined by the average value of its nodes.
- **Mapping:** Cell coloring is obtained through interpolation of its node values.
- **Contouring:** Each cell is colored relating to its node values and to isovalues between these values.
- **Texturing:** Same as Contouring but a texture is used for creating contours.

## 5.25 HexToTet

The *HexToTet* module converts a hexahedral grid to an equivalent tetrahedral grid. The new tetrahedral grid has the same points as the hexahedral grid to be converted. Each hexahedron is converted to six tetrahedra which have the same material ID as the "parent" hexahedron.

The module also converts the data objects connected to the hexahedral grid.

Press the *Apply* button to start the computation.



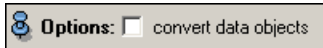
## Connections

**Data** [required]

Unstructured hexahedral grid to be converted.

## Ports

### Options



Toggles whether the data objects connected to the hexahedral grid are also converted or not.

## 5.26 HexaView

The HexaView module displays an unstructured hexahedral grid, or parts of the grid. Optionally an independent scalar field can be mapped onto the grid as pseudo-colors. The module behaves almost identical to the *GridVolume* module for tetrahedral grids.

## Connections

**Data** [required]

Unstructured hexahedral grid to be visualized.

**ColorField** [optional]

An arbitrary scalar field used for pseudo-coloring.

**Colormap** [optional]

The colormap used for pseudo-coloring, only used when a color field is connected.

## Ports

### Draw Style



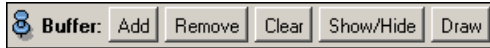
Please refer to the *GridVolume* documentation.

### Colormap



Please refer to the *GridVolume* documentation.

### Buffer



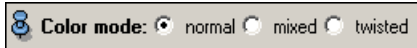
Please refer to the *GridVolume* documentation.

### Materials



Please refer to the *GridVolume* documentation.

### Color mode



Please refer to the *GridVolume* documentation.

## 5.27 IlluminatedLines

This module displays line segments of a line set object taking into account ambient, diffuse, and specular illumination. The illumination effect is implemented internally using the same texture mapping technique which is also applied in module *DisplayISL*.

### Connections

#### Data [required]

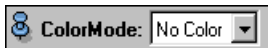
The line set to be visualized.

#### Colormap [optional]

The *colormap* used to encode the additional data of the line set. If no colormap is connected to this port it is used to set the default diffuse color of the illuminated lines.

### Ports

#### ColorMode



If the line set object contains additional data values per vertex this menu allows you to select one such variable which will be used to lookup vertex colors. If *No Color* has been selected the lines will be displayed in uniform default color. This color may be changed using the default (Constant) color setting of the colormap above.

#### Colormap



Optional colormap.

## Options



*Lighting:* Controls illumination of lines. If off, constant colored lines are drawn (flat shading).

*Animate:* Activates particle-like animation. The animation speed may be controlled via the command `setAnimationSpeed`.

## Fade Factor



Values smaller than 1 have the effect that line segments become more and more transparent in backward direction.

## Transparency



Base transparency of the line segments.

# 5.28 Interpolate

This module takes two or more data objects as input, e.g. two surfaces or two tetrahedral grids, and computes an output object by linearly interpolating the vertex positions. This can be used to create a smooth transition between the two objects. In addition, it is possible to interpolate data fields attached to the surfaces or grids as well.

Note that in any case the two input data sets must have the same number of points in order to produce meaningful results. For example, the second input could actually be a copy of the first one with an applied transformation.

Usually, the transition is specified by a parameter  $u$  varying between 0 and  $n-1$ , where  $n$  is the number of input objects. However, it is also possible to associate a physical time with each input. For this, each input must define an entry *Time* in its parameter list which specifies the actual physical time of this input. The time value of any input must be bigger than the time value of the preceeding input. If these conditions are met the toggle *physical time* described below becomes active. A *Time* parameter can be defined interactively using the parameter editor.

## Connections

### Data [required]

First input object to be interpolated. Must either be a surface, a tetrahedral grid, a hexahedral grid, a field defined on one of these objects, or a lattice object like a 3D image.

### Input2 [required]

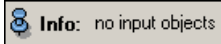
Second input object to be interpolated. This object must be of the same type as the first one.

### Input3 [optional]

Optional third input object. Once a third input object is connected automatically additional input ports will be created. In this way it is possible to connect an arbitrary number of inputs.

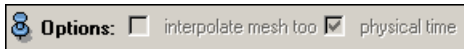
## Ports

### Info



This info port reports the number of input objects, and - if available - the current physical time or the current fractional time step depending on whether the physical time toggle is activated or not.

### Options



If the input is a surface, a tetrahedral grid or a hexahedral grid and if there are additional fields connected to these objects the first toggle allows you to interpolate the data fields as well. If the input itself is such a data field the first toggle allows you to interpolate the mesh too. In this case the toggle's label will be changed appropriately.

The second toggle allows you to activate physical time mode, provided the input objects have a *Time* entry in their parameter list. In this case the time slider (see below) denotes physical time, whereas otherwise it denotes a fractional time step.

### Time



Interpolation parameter. For  $t=0$  the output will be identical to the first input. For  $t=1$  output will be second input. If physical time mode is active, this port specifies the physical time for which an output object should be computed.

## 5.29 Isosurface (Hexahedra)

This module computes an isosurface within a three-dimensional scalar field defined on an unstructured hexahedral grid.

A second independent scalar field may be connected to the module. This field determines how the isosurface is colored. If no color field is connected to the module the isosurface has constant color.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

The scalar field defined on an unstructured hexahedral grid. The *encoding* must be PER-VERTEX.

### ColorField [optional]

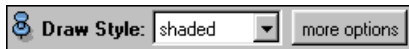
Arbitrary scalar field which is mapped onto the isosurface using pseudo-coloring.

### Colormap [optional]

The colormap is used for pseudo-coloring the isosurface.

## Ports

### Draw Style



The draw style port is inherited from class *ViewBase*. For a description of this port see *there*.

### Colormap



In case a colormap is connected to the isosurface module, this colormap will be shown here. If no colormap is connected to the module the port's default color is used. To change this color, click into the color bar with the left mouse button. This will bring up the color selection dialog. To connect the port to a colormap, use the popup menu under the right mouse button. See also *Colormap*.

### Buffer



This port must be enabled explicitly with the command `Isosurface showBuffer`. For a detailed description see the *ViewBase* documentation.

### Threshold



Determines the value used for isosurface computation. The slider is automatically adjusted to cover the whole range of data values.

## 5.30 Isosurface (Polyhedra)

This module computes an isosurface for a scalar field defined on a three-dimensional tetrahedral grid. A second independent scalar field may be connected to the module. This field determines how the isosurface is colored. If no color field is connected to the module, the isosurface has constant color.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

The scalar field defined on a polyhedral grid.

### Color field [optional]

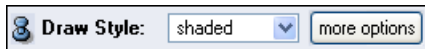
Arbitrary scalar field which is mapped onto the isosurface using pseudo-coloring.

### Colormap [optional]

The colormap is used for pseudo-coloring the isosurface.

## Ports

### Draw Style



The draw style port is inherited from class *ViewBase*. For a description of this port see *ViewBase*.

### Colormap



If a colormap is connected to the isosurface module, this colormap will be shown here. If no colormap is connected to the module, the port's default color is used. To change this color, click into the color bar with the left mouse button. This will bring up the color selection dialog. To connect the port to a colormap, use the popup menu under the right mouse button. See also *Colormap*.

### Threshold



Specifies the value used for isosurface computation. The slider is automatically adjusted to cover the whole range of data values. If the threshold value is changed, the computation of the new isosurface is immediately invoked.

## 5.31 Isosurface (Tetrahedra)

This module computes an isosurface for a scalar field defined on a three-dimensional tetrahedral grid. The triangulation inside a tetrahedron exactly matches a linearly interpolated field.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

The scalar field defined on a tetrahedral grid.

**ColorField** [optional]

See port *ColorField* in section *Isosurface (Hexahedra)*.

**Colormap** [optional]

See port *Colormap* in section *Isosurface (Hexahedra)*.

## Ports

### Draw Style



The draw style port is inherited from class *ViewBase*. For a description of this port see *there*.

### Colormap



See port *Colormap* in section *Isosurface (Hexahedra)*.

### Threshold



Determines the value used for isosurface computation. The slider is automatically adjusted to cover the whole range of data values. If the threshold value is changed the computation of the new isosurface is immediately invoked.

### Buffer



This port must be enabled explicitly with the command `Isosurface showBuffer`. For a detailed description see the *ViewBase* documentation.

## 5.32 Lambda2

The *Lambda2* module computes the second largest eigenvalue of the matrix  $L = S^2 + \Omega^2$  where  $S$  and  $\Omega$  are the symmetric and anti-symmetric parts of the Jacobian of a vector field.

The points where this is negative marks a vortex core.

Press the *Apply* button to start the computation.

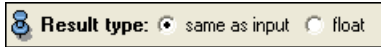
## Connections

**Data** [required]

Vector field defined on a uniform grid (*UniformVectorField3*). The vector components must be floats.

## Ports

### Result type



## 5.33 LatToHex

The *LatToHex* module converts a 3D lattice to an equivalent hexahedral grid. The new hexahedral grid has the same points as the lattice to be converted. The default material ID 0 will be assigned to each hexahedron.

The module can also convert the data contained in the lattice. Lattices having a dimension of 1 in one direction, e.g., images, are converted to grids containing only vertices. Such grids can be visualized with *VertexView*.

Press the *Apply* button to start the computation.

## Connections

### Data [required]

3D lattice (uniform to curvilinear) to be converted.

## Ports

### Options



Toggles whether the data contained in the lattice is also converted or not.

## 5.34 LineStreaks

This module takes a surface and randomly distributes a number of short line segments on it. The line segments are computed as field lines of a 3D vector field projected onto the surface or as field lines of a vector field directly defined on that surface. The latter can for example be produced by the *GetCurvature* module in *Max Direction* mode. Instead of being visualized directly the resulting line streaks will be stored in a *LineSet* data object.

Press the *Apply* button to start the computation.



## Connections

**Data** [required]

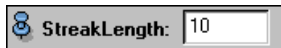
The surface where the streaks will be put on.

**VectorField** [required]

The vector field from which the streaks will be computed. May be either a surface vector field or a 3D vector field.

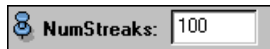
## Ports

**StreakLength**

A port control for StreakLength. It features a blue circular icon with a white 'S' on the left, followed by the text 'StreakLength:' and a text input field containing the value '10'.

Relative length of the field line segments. The length is scaled by the length of the diagonal of the surface's bounding box.

**NumStreaks**

A port control for NumStreaks. It features a blue circular icon with a white 'S' on the left, followed by the text 'NumStreaks:' and a text input field containing the value '100'.

Number of line streaks to be computed. The number of streaks per surface area will be approximately constant.

## Commands

`setResolution <res>`

Allows you to adjust the resolution of the line segments. The higher the value the smaller the point spacing. The default value is 256, resulting in a point distance of  $1/256$  of the length of the surface's bounding box.

## 5.35 MagAndPhase

The *MagAndPhase* module connects to a `UniformComplexScalarField`. For each point in space it computes the phase and the squared magnitude and creates a `UniformColorField` from this information. The squared magnitude is used to determine the alpha value (i.e. the opacity) of a voxel. The phase determines the color using an arbitrary colormap. This module can be used to visualize quantum mechanical wave functions.

Press the *Apply* button to start the computation.

## Connections

**Data** [required]

Complex scalar field defined on a regular grid.

### Colormap [optional]

A colormap that is used to visualize the phase. In order to avoid that phase values are being clipped the range of the colormap should extend from  $-\pi$  to  $+\pi$ .

## Ports

### Colormap



Phase colormap input port.

## 5.36 Magnitude

The *Magnitude* module computes the magnitude of the vectors of a vector field, i.e.,

$$|V| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

These vectors might be located on regular or on irregular grids. Regular or complex vector fields are typically defined by uniform, stacked, rectilinear, or curvilinear coordinates. Vectors located on vertices of triangular surface meshes or tetrahedral grids are defined by irregular coordinates.

The result of *Magnitude* is a field of scalars, located at positions according to the underlying grid of the input data.

Press the *Apply* button to start the computation.

## Connections

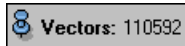
### Data [required]

Field or complex vector field defined on a regular grid (*Lattice3*).

Vector field on an irregular grid, either specified by nodes of a surface, a tetrahedral or a hexahedral grid (*SurfaceField*, *TetraData*, *HexaData*).

## Ports

### Number of Vectors



Shows number of input vectors and the vector dimension for computing the magnitude field.

### Mode



Choose whether to compute the magnitude of the vector, its normal or tangential component relative to the primary surface normals. This port is only shown if a surface vectorfield has been connected.

## 5.37 ParametricSurface

This module can be used to define and animate arbitrary parametric surfaces in two, three dimensions. There are some pre-defined surfaces like the Moebius strip, Klein bottle and Random mapping to illustrate the use of the text fields that define expressions for X, Y, and Z.

The expressions may depend on `u` and `v` which are Cartesian coordinates of the plane. They can also depend on `theta` and `r` which are their counterparts in polar coordinates. A separate spherical coordinate system is accessible by the variables `lambda` and `phi`. The variable `t` is linked to the time port and can be used to perform for example linear interpolations between two mappings like:

$$(t - 1) * (\text{expression1}) + t * (\text{expression2}). \quad (5.3)$$

Press the *Apply* button if you want to do an animation because you will need to re-compute the mesh every time `t` changes.

Surfaces can be exported by the `Draw Style` port ( $\Rightarrow$  more options  $\Rightarrow$  Create surface).

### Connections

#### **Data** [optional]

If a regular scalar field is supplied, the generated mesh will be scaled and shifted according to its bounding box. Using the transform editor of the data is a convenient way of scaling and shifting the generated mesh.

#### **Colormap** [optional]

The colormap is used to calculate the color of the mesh based on the expression in the Colorport (initially `r`, the radial distance of the u-v mesh point from its mean position).

#### **Time** [optional]

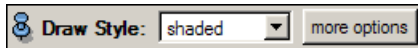
An additional variable which can be used in all the expression fields. It can be used to generate animations.

#### **Cluster** [optional]

If you connect to this module a Cluster object (points in space), the module will generate a second cluster object and apply the transformation in the expression fields to its `x` and `y` coordinates. This may be useful to visualize the u/v plane as a point cloud.

## Ports

### Draw Style



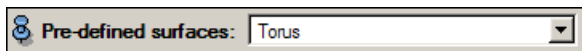
Select a specific draw style for the mesh currently on display. For higher resolution meshes usually shaded works best.

### Colormap



This colormap will be used to map colors per vertex using by the values in the *Color* port.

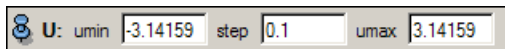
### Pre-defined surfaces



The pre-defined surfaces are some commonly used to illustrate analytically defined meshes. They are mainly here to illustrate the use of the different variables in the Colorports.

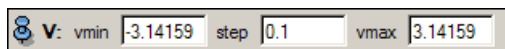
As a last entry you will find a Random generator which will generate random entries for the x, y and z ports, i.e. a random mapping. Currently there is a maximum depth of 10 for the randomly generated expressions.

### U



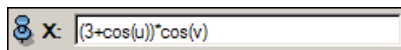
This port describes the u resolution of the initial mesh (before the transformation was applied). Its values code the minimum value of u (step size) and the maximum value of u. Lowering the step-size will directly increase the resolution of the mesh.

### V



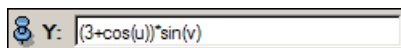
This port works the same way as the [5.37](#) port for the v parameter of the initial mesh.

### X



Enter an expression to be evaluated in order to describe the x-position of a mesh in 3D space. Use the [5.37](#) port to see some useful settings for this port.

### Y



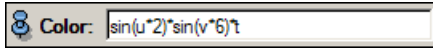
Enter an expression to be evaluated in order to describe the y-position of a mesh in 3D space. Use the [5.37](#) port to see some useful settings for this port.

## Z



Enter an expression to be evaluated in order to describe the z-position of a mesh in 3D space. Use the 5.37 port to see some useful settings for this port.

## Color



Enter an expression to be evaluated in order to describe the vertex color of the mesh. All variables of the other fields can be used in this field. By default it contains  $r$  which is the radial distance from the center position of the mesh.

## Time



This slider is directly linked to the expression  $t$  which can be used in all  $x$ ,  $y$ ,  $z$  and Color ports.

## 5.38 ParticlePlot

ParticlePlot is a replacement for the older *ConePlot* module. For compatibility reasons the old ConePlot module is still available.

This module visualizes scalar vector fields by particles (cones) pointing in the direction of the local flow. The placement of particles can be done similar to the distribution modes of the *DisplayISL* module. The particles are animated to generate a sequence of objects "walking" through the vector field. An example network can be found at `share/examples/HxConePlot.hx`. ParticlePlot uses Open Inventor render caching. All animation steps are pre-computed and played back by making the appropriate parts of the scene visible. This technique requires little CPU computation but larger amounts of memory and a fast graphics card. Notice that the first pass through the animation may be slow but subsequent render passes speed up. To control the overall speed of the animation use the `setRealTimeDuration` command of the time slider.

Note that you must press the *Apply* button for every change in the parameters. Only the objects' *Height* and *Radius* are updated without you needing to explicitly press the *Apply* button.

## Connections

### Data [required]

The 3D vector field to be visualized.

### ROI [optional]

A connection to a *SelectRoi* module which specifies a rectangular region of interest. The seed points will be restricted to its region of interest.

### Shape data [optional]

Supply a custom shape to be replicated instead of the cones. Any geometry displayed in *amira* can be used.

### Distribution [optional]

An optional scalar field which can be used to control the distribution of seed points.

### Color field [optional]

An optional scalar field which, if present, will be used for pseudo-coloring. Often it is useful to create a scalar field from a vector field using the *Magnitude* module.

### Colormap [optional]

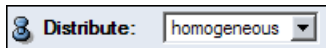
The *colormap* used to encode the color field. If no color field is connected, the magnitude of the vector field at each position will be used for pseudo-coloring.

### Animate [optional]

A time slider which controls the currently shown step of the animation. To control the overall speed of the animation use the `setRealTimeDuration` command of the time slider.

## Ports

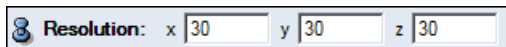
### Distribute



This port provides an option menu specifying how seed points are distributed inside the seed box. By default a *homogeneous* distribution will be used. Alternatively, seed points may be distributed according to the vector field's magnitude or according to the value of the distribution field, if such a field is connected to the module. The *equalize* option provides a mixture of homogeneous and proportional seed point distribution.

The *regular* option distributes the objects at regular intervals in the bounding box using the information in the *Resolution* port. For this last setting, the length of integration is set initially to 1 in order to restrict the number of cones produced.

### Resolution



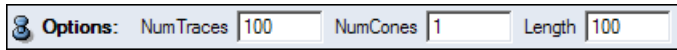
For proportional and equalized distribute settings, this port specifies the resolution at which the vector field will be sampled. If the regular mode for the distribution is selected, this port will specify the number of objects distributed in the bounding box.

### Colormap



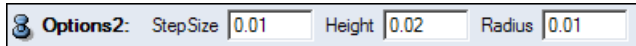
A colormap which specifies the pseudo-coloring used. Please note that the maximum magnitude detected is printed out by the module after each *Apply* button action.

### Options

A widget with a blue icon and the label "Options:". It contains three text input fields: "NumTraces" with value "100", "NumCones" with value "1", and "Length" with value "100".

This port specifies the number of traces computed, the number of cones per trace which are displayed at each point in time, and the number of steps of the vector field integration.

### Options2

A widget with a blue icon and the label "Options2:". It contains three text input fields: "StepSize" with value "0.01", "Height" with value "0.02", and "Radius" with value "0.01".

This port specifies the initial step size of the Runge-Kutta solver and the height and radius of the cones. If a shape object is attached to the module, only the height setting will be used to specify its isotropic scaling.

If you want to change the default data range of each of the *Options* ports, you can use the following Tcl command:

```
ParticlePlot Options setMinMax 2 0 500
```

which will for example set the minimum and maximum values of the *Length* text field to 0 and 500.

### Threshold

A widget with a blue icon and the label "Threshold:". It contains a text input field with the value "x>0".

This port specifies an expression field similar to the one used in *Arithmetic*. The value *x* is defined as the magnitude of the vector field evaluated at each position of the field.

### Animate

A widget with a blue icon and the label "Animate:". It contains a horizontal slider bar with play, stop, and first/last buttons, and a text input field with the value "0".

A time slider which controls the currently shown step of the animation. To control the overall speed of the animation use the `setRealTimeDuration` command of the time slider.

## Commands

#### `createLineSet`

Generates a line set object from the current particle trace.

#### `getStepsToBlend`

Returns the current number of steps that will be used at the begin and end of each particle trace to blend out the objects.

#### `setStepsToBlend x`

Sets the number of steps that will be used at the begin and end of each particle trace to blend out the objects.

## 5.39 PlanarLIC

This module intersects an arbitrary 3D vector field and visualizes its directional structure in the cutting plane using a technique called *line integral convolution* (LIC). The LIC algorithm works by convolving a random noise image along the projected field lines of the incoming vector field using a piecewise-linear hat filter. The synthesized texture clearly reveals the directional structure of the vector field inside the cutting plane. As long as no valid LIC texture has been computed, a default checkerboard pattern is displayed instead.

*PlanarLIC* is derived from *ArbitraryCut*. See the documentation of this module for details on how to adjust the position and orientation of the cutting plane. [Click here](#) in order to execute a script demonstrating the use of the *PlanarLIC* module.

Press the *Apply* button to start the computation. If no LIC texture has been computed yet, a default checkerboard pattern is displayed. This pattern is also displayed as soon as filter length or resolution are changed. Press the *Apply* button again in order to update the texture.

### Connections

#### Data [required]

Vector field to be visualized.

#### ColorField [optional]

A scalar field which may be used for pseudo-coloring.

#### Colormap [optional]

Colormap used for pseudo-coloring. If no colormap is connected the default color of the colormap port will be used. The port is hidden if pseudo-color mode is set to *none*.

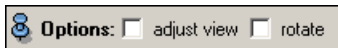
### Ports

#### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

#### Options



If toggle *adjust view* is active, then the camera of the 3D viewer will be reset whenever one of the orientation buttons is clicked.

If the *rotate* toggle is active, then a virtual trackball is displayed. By picking and dragging the trackball you may change the orientation of the plane. Remember that the viewer must be in interaction



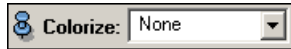
mode in order to do so. The ESC-key inside the viewer window toggles between navigation mode and interaction mode. The trackball of the last active ArbitraryCut can also be turned on and off by pressing the TAB-key inside the viewer window.

### Translate



This port lets you translate the plane along its normal direction.

### Colorize



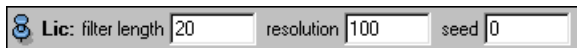
An option menu allowing to select different pseudo-color modes. If item *none* is selected, the LIC texture will be displayed in grayscale only. If *magnitude* is selected, each pixel of the LIC texture will be colored according to vector magnitude at this point. If *normal component* is selected, then color denotes the signed length of the vector component perpendicular to the cutting plane. This length will be positive if the vector points upwards or negative if the vector points downwards. If *parallel component* is selected, then color denotes the length of the vector component tangential to the plane. This length will always be greater or equal than zero. Finally, if *color field* is selected, and if a scalar field is connected to port *ColorField*, the external scalar field will be used for pseudo-coloring.

### Colormap



Port to select a colormap.

### Lic



The input denoted *filter length* controls the one-sided length of the filter kernel used for line integral convolution. The larger this value is, the more coherent the grayscale distribution along the field lines. Often larger values are visually more attractive than smaller ones. A value of 0 lets you see an isotropic noise pattern without any directional information.

The second input of this port determines the *resolution* of an intermediate sampling raster used to compute field lines. Fine details of the vector field might be missed if the sampling resolution is too low. The resolution value also has an effect on the granularity of the resulting LIC image. The size of the LIC texture being computed is chosen to be the next power of two larger than or equal to the sampling resolution. For example, if the resolution is set to 128, the size of the LIC texture will be 128x128. If the resolution is set to 129, then a LIC texture of size 256x256 will be computed, resulting in much finer structures.

The third input of this port denoted as *seed* allows control of the generation of the noise pattern. A value of 0 means that a different random noise pattern is used for each successive calculation. Any

other value allows use of the same noise pattern each time a computation is started. The latter yields better results for videos where the LIC plane slices through a volume.

#### Phase



This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controlling which part of the complex 3D vectors is visualized. A value of 0 degree corresponds to the real part, while a value of 90 degrees corresponds to the imaginary part. Other values yield intermediate vectors

## Commands

`setNumSubPixels {1|2|3}`

Allows you to change an internal parameter of the LIC algorithm. Since LIC images contain very high spatial frequency components, they are susceptible for aliasing. Aliasing can be almost eliminated by choosing the number of sub-pixels to be 2 or even 3. However, this is achieved at the expense of increased computing time.

`writeTexture <filename>`

This command allows you to write the current LIC texture into a file in raw PPM format.

## 5.40 RateOfStrainTensor

This module computes the rate of strain tensor for a given displacement vector field. If the displacement vector field describes the displacement of matter in space, the rate of strain tensor field will specify how the space is distorted by this field.

Analyze the resulting tensor field by extracting its eigenvalues (ExtractEigenvalues module) where the tensor field should be indefinite and thus may contain negative and positive eigenvalues for directions in which the matter is compressed or expanded during the distortion.

## Connections

**Data** [required]

A vector field which describes the space distortion.

## 5.41 SeedSurface

This module extends the vector field visualization module *DisplayISL*. It allows you to compute illuminated field lines of a vector field with seed points distributed across an arbitrary *surface*.

First, load both the vector field and the surface into *amira*. Then attach *DisplayISL* to the vector field. From the popup menu of *DisplayISL* choose *SeedSurface*. The *SeedSurface* module automatically connects itself to the first surface found in the Pool. Of course, you may change the surface connection at any time later on. Properties of the illuminated field lines such as base opacity, fade factor, or color will be determined by *DisplayISL*.

The *Apply* button is used to initiate distribution of seeds and recomputation of field lines. Once the incoming vector field has changed or you have modified the number of field lines or the line's length, you must press *Apply* again in order to update the display.

## Connections

### Data [required]

Module of type *DisplayISL* which is used to display the illuminated field lines.

### DisplayISL [required]

The surface on which the field lines will be distributed.

## Ports

### NumLines



Number of field lines to be displayed. This port will only be visible if distribution mode *on surface* has been selected. The distribution algorithm tries to achieve a constant seed density per surface area.

### Length



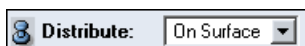
The length of the field lines, or more precisely, the number of atomic line segments, in forward respectively backward direction. The lines may stop earlier if a singularity (i.e. zero magnitude) is encountered or if the field's domain is left.

### Balance



By default field lines are equally long in forward and backward direction, corresponding to a balance value of 0. This port allows you to change this behavior. A value of -1 indicates that field lines should extend in backward direction only, while a value of 1 indicates that field lines should extend in forward direction only.

### Distribute



This port provides an option menu specifying the seed distribution mode. If *at vertices* is chosen, a field line is started at each vertex of the surface. If *on surface* is chosen, a user-defined number of field lines will be uniformly distributed across the surface.

## 5.42 Splats

This module visualizes scalar fields defined on tetrahedral grids using a direct volume rendering technique called cell projection splatting. The principle of this method is to display a kind of semi-transparent clouds. The higher the data values the brighter and more opaque these clouds are. Often, meaningful results are obtained if this technique is used in conjunction with a standard *isosurface module*.

In order to get correct results for linearly varying functions a special texture mapping technique is applied. On machines where texture mapping is not supported in hardware this might be quite slow. As an alternative untextured splats may be used. However, in this case the scalar field is assumed to be constant in each tetrahedron. To obtain such a piecewise constant function the vertex values of each tetrahedron are averaged. As a consequence artificial discontinuities might be observed.

Press the *Apply* button to start the computation.

### Connections

**Data** [required]

The tetrahedral scalar field to be visualized.

### Ports

#### Alpha Scale



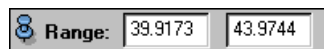
A global factor used to change the overall transparency of the individual splats. Higher values produce denser clouds.

#### Gamma



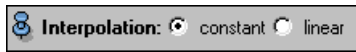
This value determines how the function value between the min and max values of port *Range* will be mapped to opacity and color. If the gamma value is 1 a linear mapping will be used. If the gamma value is smaller than 1 the overall appearance of the cloud gets more opaque. If the gamma value is bigger than 1 the cloud gets more transparent.

#### Range



Data range. Regions where the function value is below the min value will be completely transparent. Likewise, regions where the function value is above the max value will be opaque.

### Interpolation



Lets you select the type of splats being used. *Constant* enables untextured splats. *Linear* enables textured splats. The latter setting will be able to correctly display linearly varying scalar fields.

## Commands

`setColor <color>`

Lets you define the base color of the volume rendered clouds. On default an orange color will be used (0.8 0.6 0.1). The color may be specified by either an RGB triple in range 0...1 or by a common X11 color name, e.g., *green*.

## 5.43 StreamRibbons

This module displays stream lines or stream ribbons in a flow field. Stream ribbons are computed by tracing two individual stream lines and connecting them by triangles. The initial orientation of a stream ribbon is orthogonal to the normal direction of the flow field at the seed point. The seed points themselves are defined interactively by moving a seed shape in space (a line, a circle, or a filled square). The seed shape can be transformed using an Open Inventor transformer dragger. In VR Pack it is also possible to pick and transform the dragger using the 3D mouse.

### Connections

**Data** [required]

The vector field to be visualized.

**Colormap** [optional]

Colormap used to depict vector magnitude.

### Ports

**Colormap**



Colormap used to depict vector magnitude.

**Resolution**



Logarithmic slider allowing to adjust the resolution of the stream line tracing algorithm. A value of 1 means a 10 times higher resolution compared to the default, while a value of -1 means a 10 times smaller resolution. The higher the resolution the more line segments and triangles are generated.

### Density



Logarithmic slider allowing to adjust the density of stream lines or stream ribbons. The higher the value the more lines or ribbons are traced.

### Width



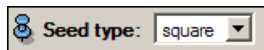
This slider can be used to adjust the width or thickness of the stream ribbons. The default thickness is set proportional to the length of the diagonal of the bounding box of the input data set.

### Length



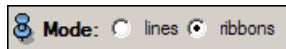
Adjusts the length of the stream lines or stream ribbons.

### Seed type



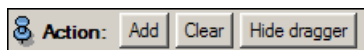
Defines the seed type: along a straight line segment, along a circular line, or inside a square regions.

### Mode



Defines if simple stream lines or stream ribbons should be traced.

### Action



The *Add* and *Clear* buttons have no meaning yet. The third button allows you to show or hide the Open Inventor dragger for manipulating the seed shape position.

## Commands

Inherits all ports of *Object*.

`setBox -b xmin xmax ymin ymax zmin zmax`

Sets the position of the Open Inventor dragger so that it matches the specified box. The display is updated automatically.

`setBox [-t x y z] [-r x y z phi] [-s x y z]`

Alternate way of setting the Open Inventor dragger. The three optional arguments indicate the

translational, rotational, and scaling part of the dragger's transformation matrix. With this command it is possible to set the dragger in an arbitrary way (it must not be axis-aligned). If all three parts are specified the option strings `-t`, `-r`, and `-s` can be omitted.

### getBox

Returns the current transformation of the dragger in terms of translation (first three numbers), rotations (next four numbers), and scaling (last three numbers). The result can be used as the arguments of the `setBox` command.

## 5.44 StreamSurface

This module computes stream surfaces of an arbitrary 3D vector field. A stream surface consists of multiple interconnected stream lines started along a predefined line source. The algorithm automatically inserts new stream lines in divergent regions of the field. Likewise, in convergent regions stream lines are automatically removed. The resulting stream surfaces can be accumulated in a separate *Surface* object. In this way further post-processing is facilitated, for example, computation of a LIC texture using *SurfaceLIC*.

### Connections

#### Data [required]

The 3D vector field to be visualized.

#### Colormap [optional]

Optional colormap. By default, the current stream surface will be display in wireframe mode and will be colored according to the arc-length of the stream lines forming the stream surface.

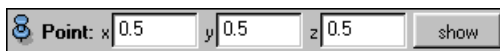
### Ports

#### Colormap



Port to select a colormap.

#### Origin



This port defines the seed point of the stream surface. By pressing the toggle *show* a crosshair dragger can be activated which allows you to change the seed point interactively in 3D.

Actually, in order to compute a stream surface not only a seed point but a seed line is required. Starting from the seed point, such a line will be defined automatically taking into account the seed selection mode chosen in port *Action*.

## Resolution



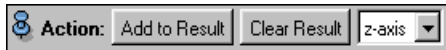
Controls the resolution of the discretized stream surface or, more precisely, the preferred edge length of the triangulation. Smaller values results in more details.

## Length



Value *n* determines how far the surface should be traced in backward direction. Value *m* determines how far the surface should be traced in forward direction. Value *width* determines the extent of the seed line. The actual lengths in physical space depend on the value of port *resolution*. All values may be changed using the right mouse buttons by means of a virtual slider.

## Action



This port provides push buttons allowing you to store the current stream surface in a surface object (*add to result*), respectively to remove all triangles from this surface again (*clear result*). Moreover, the port provides an option menu allowing you to select the mode used for automatic seed line definition. The following modes are supported:

*Binormal*: The seed line will be traced in a direction perpendicular to both the stream line's tangent and normal (curvature) direction. Often this gives quite meaningful results. However, note that in general the stream surface's normal vectors do not coincide with the stream line's curvature vectors except at the seed line.

*Normal*: The seed line will be traced along the field line's normal (curvature) direction.

*X-axis*: The seed line is chosen in x-direction

*Y-axis*: The seed line is chosen in y-direction.

*Z-axis*: The seed line is chosen in z-direction.

## Commands

`setLineWidth <width>`

Sets line width of wireframe model.

`setTolerance <eps>`

Sets relative tolerance used for field line integration. The default is 0.001 times the sampling width set in port *resolution*.

`doLineSet {0|1}`

If argument is 1 only stream lines will be displayed instead of the stream surface's triangulation.

`setDrawStyle {1|2}`

Allows you to set the draw style used in surface mode, i.e., when `doLineSet` is off. A value of 1



denotes wireframe mode, while a value of 2 denotes shaded surface mode. In order to display the surface in a more fancy way convert it into a surface object and use *SurfaceView*.

## 5.45 SurfaceLIC

This module visualizes a vector field defined on an arbitrary triangular surface using line integral convolution (LIC). Alternatively, a 3D vector field projected onto such a surface can be visualized. The LIC algorithm works by convolving a random noise function along field lines tangential to the surface using a piecewise-linear hat filter. In this way for each triangle a small piece of texture is computed and mapped back onto the surface. The final surface texture clearly reveals the directional structure of the surface vector field. A similar 2D algorithm is implemented by the *PlanarLIC* module.

Press the *Apply* button to start the computation of the surface LIC texture. Computation may take a minute or more depending on texture resolution and on the number of triangles of the surface.

Click here to execute a script demonstrating the *SurfaceLIC* module. Computation of the surface LIC texture may take about half a minute.

### Connections

#### Data [required]

Surface for which a LIC texture is to be computed.

#### VectorField [required]

Surface vector field or 3D vector field to be visualized.

#### ScalarField [optional]

An optional scalar field which can be used for pseudo-coloring.

#### Colormap [optional]

Colormap used for pseudo-coloring. If no colormap is connected the default color of the colormap port will be used.

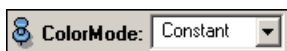
### Ports

#### Colormap



Port to select a colormap.

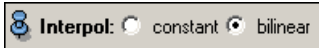
#### ColorMode



Three different color modes are provided. If *Constant* is selected, then a uniform overall base color

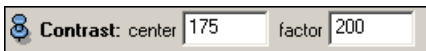
is used for the LIC texture. This will be the default color of the colormap port or the left-most color of the colormap connected to this port, if any. If *Magnitude* is selected, then the LIC texture will be colored according to the magnitude of the vector field. Finally, if *Color field* is selected, and a scalar field is connected to the module, then the LIC texture will be colored according to the values of this scalar field.

### Texture Interpolation



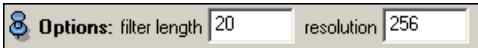
A radio box determining how the triangle textures are being filtered by the underlying OpenGL driver. Possible choices are *constant* or *bilinear* interpolation. Usually, you will not see a big difference unless you zoom up the image very much.

### Contrast



This port provides two parameters controlling the amount of contrast of the final LIC texture. Input field *center* specifies the average gray value of the texture. Higher values result in brighter images. Input field *factor* determines the width of the gray value histogram, which is of Gaussian type. Higher numbers produce more contrast.

### Options



Parameter *filter length* specifies the one-sided length of the triangular filter kernel used for line integral convolution. The larger this value, the more coherent the grayscale distribution along the field lines. Often larger values are visually more attractive than smaller ones.

The second input determines the *resolution* of the LIC texture. More precisely, the width of a single texture cell is chosen to be equal to the length of the diagonal of the incoming vector field's bounding box divided by the value of the *resolution* field.

## Commands

`setAmbientColor <color>`

Allows you to change the ambient color of the surface.

`setDiffuseColor <color>`

Allows you to change the diffuse color of the surface.

`setSpecularColor <color>`

Allows you to change the specular color of the surface.

`setShininess <value>`

Allows you to change the shininess of the surface.

**setCreaseAngle** <value>

Neighboring triangles will share a common vertex normal if the angle between their face normals is smaller than the crease angle. The default value is 60 degrees. This command lets you overwrite this value. Note that discontinuities will appear if the triangles of the input surface are not oriented in the same way. In order to fix this, use the command **fixOrientation** of the surface.

## 5.46 TensorDisplay

This module displays symmetric second order tensors using tensor glyphs. You can select either a sphere, a cylinder, a cone, or three lines to indicate the tensor shape in a slice through the volume.

The module can display positive definite tensors or tensors with negative eigenvalues. Positive definite tensor fields occur, for example, in the form of diffusion tensors. Negative eigenvalues occur, for example, in the analysis of rate of strain (stress) tensors.

### Connections

**Data** [required]

A symmetric second order tensor field with 6 degrees of freedom.

**Module** [optional]

A connected EmptyPlane module which is used to orient the tensor display plane in space.

**Colormap** [optional]

Tensor glyphs can be colored by this colormap.

**Color field** [optional]

A scalar field which can be used to color the tensor display.

**Mask** [optional]

In order to restrict the display of tensors to a specific area, a mask volume can be attached. If such a module is connected, an additional port is displayed with an expression to select a sub-area in the mask. If you connect, for example, a label field as a mask, you can use an expression like  $A \neq 0$  to disable the display of tensors in the exterior part of the label field.

**Shape data** [optional]

You can connect any currently displayed geometry to this port and it will be used instead of the settings in the Display port.

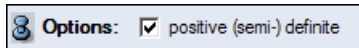
### Ports

**Display**



Select either spheres, cylinders, cones, or lines to be displayed at regular positions in the tensor field.

### Options



If you select this checkbox, only positive definite tensors will be computed by using a singular value decomposition. This is the default setting for diffusion tensors. If you de-select this checkbox, a standard algorithm for principal component analysis will be used, and also negative eigenvalues may occur.

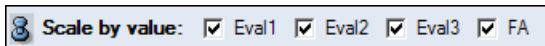
### Mappings and colorings



The first drop down menu is only active if we have also negative eigenvalues (see port Options). The negative eigenvalues can be mapped to positive eigenvalues by either using absolute values for the eigenvalues, by doing a sigmoid mapping of the eigenvalues to 0..1 (eigenvalues close to zero will be around 0.5), or by using the Cauchy-Green tensor or the Green strain tensor.

The color drop down menu allows you to color the tensors by either their fractional anisotropy (FA) with large values for tensors with a preferred direction, by their trace, or by either their first, second, or third eigenvalue. You can also connect a ColorField to the module and the data values from this volume will be used instead.

### Scale by value



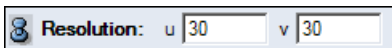
The tensor glyphs are scaled according to the value of their eigenvalues. If Eval2 and Eval3 are disabled, the tensor is drawn with a default scaling of 1 to 1/2 for the first principal direction to the second and third principal direction.

### Display complexity



Values larger than 0.2 will increase the number of triangles used to display the glyphs.

### Resolution



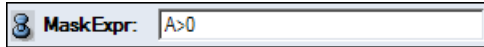
The number of glyphs sampled over the size of the EmptyPlane. Larger numbers result in more dense tensor displays.

### Colormap



The colormap used to color the tensors.

### MaskExpr



This port is only available if a scalar field is connected to the *Mask* input port. Enter an expression like  $A_{128}$  and the tensor display will be restricted to regions where the mask contains values greater than 128.

### Scale



Scales the tensor display. For larger resolution use smaller scalings.

## 5.47 TetToHex

The *TetToHex* module converts a tetrahedral grid to an equivalent hexahedral grid. Each tetrahedron is converted to four hexahedra which have the same material ID as the “parent” tetrahedron. The new hexahedral grid does not have the same point set as the original tetrahedral grid. The new point set has more points, the new points being the center of each tetrahedron, the center of each face, and the middle point of each edge.

The module also converts the data objects connected to the hexahedral grid.

Press the *Apply* button to start the computation.

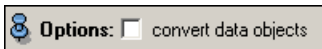
### Connections

**Data** [required]

Tetrahedral grid to be converted.

### Ports

#### Options



Toggles whether the data objects connected to the tetrahedral grid are also converted or not.

## 5.48 TetraCombine

This module takes two tetrahedral grids as input, puts them together and creates a new combined grid. The module can be used to create an *extended grid* from a patient grid and a grid representing the exterior space including parts of a treatment device or support to which the patient’s body is attached. Triangles representing the boundary of the patient’s body may be present in both input grids. *Combine*

provides an option to remove any duplicated triangles and vertices from its output. The order in which the input grids are combined does not matter.

To make sure that both grids being combined fit exactly together, you can attach a *GridVolume* module to the combined grid. When invoking that module, all *exterior* triangles of the grid are highlighted, i.e., all triangles which are incident to only one tetrahedron. The displayed triangles should all be located at the outer boundary of the combined grid, not in its interior.

Press the *Apply* button to start the computation.

## Connections

**GridA** [required]

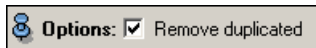
The first input grid.

**GridB** [required]

The second input grid.

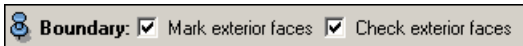
## Ports

### Options



*Remove duplicated* causes all duplicated points and triangles to be removed from the output.

### Boundary



*Mark exterior faces* sets `boundaryConditionId = 1` for all exterior faces. `BoundaryConditionIds` may be useful if a numerical simulation shall be performed on the resulting grid.

*Check exterior faces* checks whether all exterior triangles lie on a sphere around the center of the bounding box of the grid.

## 5.49 TetraGen

The *TetraGen* module creates a *volumetric tetrahedral grid*. Its input is a description of the 3D geometry by triangulated surfaces. The *advancing front method* is applied to fill each region defined by the surface data with tetrahedra. Tetrahedron generation can be performed *on-line* or as a *batch job*.

There are some 'reserved region names' for the exterior region that should not be filled with tetrahedra (otherwise the grid would extend to infinity): no name at all, 'Outside', and all names starting with 'Exterior'. If you choose a different name for the exterior region, tetrahedron generation will fail. Currently the *SurfaceGen* module creates correct names ('Exterior' and 'Exterior2'), but the module *IvToSurface* does not. You must edit the region names before applying TetraGen to a surface created by the latter module.

## Connections

### Data [required]

The input surface file (suffix `surf`).

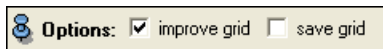
## Ports

### Region



The menu of this port allows you to specify whether tetrahedra should be generated for all regions or just for one selected region. The latter choice is needed for testing purposes only.

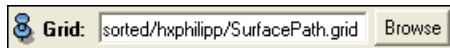
### Options



If toggle 'improve grid' is set, the quality of the resulting tetrahedral grid will be improved in a post-processing step. A combined smoothing will be applied which includes moving inner vertices and flipping inner edges or faces of the grid.

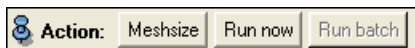
If toggle 'save grid' is set, an additional port *Grid* will occur where you can enter a filename. The resulting tetrahedral grid will be automatically saved under that filename. This toggle must be set if tetrahedron generation shall be performed as a batch job.

### Grid



This port is only visible if toggle 'save grid' at port *Options* is set. Here you can define the filename for the resulting tetrahedral grid. We recommend to include a suffix `grid` into the filename.

### Action



If you press the *Meshsize* button, an editor window appears. It allows you to define a desired mesh size for each region. Note that there are predefined values for some materials in *amira*'s material database. Check whether these values are appropriate for your application.

After you have set up the simulation, you can commit it by pressing the *Run batch* or the *Run now* button. If the file specified at port *Grid* already exists, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename.

If you press the *Run batch* button, the *job dialog* window appears, showing the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

If you press the *Run now* button, tetrahedron generation will be performed *on-line*. This may take some time for large surfaces. The progress bar indicate the current region and which part of its volume is already filled with tetrahedra.

## 5.50 TetraQuality

The *TetraQuality* module creates a histogram of qualities for tetrahedral volume grids, e.g., tetrahedral patient models. For this purpose it has to be attached to a *Grid Volume* module. Quality is calculated for all tetrahedra selected by that module. By default, the histogram is shown with a logarithmic scale to direct the focus on the tetrahedra with worst quality.

### Connections

**Data** [required]

The tetrahedral volume.

**GridVolume** [required]

A *Grid Volume* module that selects the tetrahedra for which the quality is calculated.

### Ports

**Quality Measure**



This option menu lets you select between different quality measures:

- **Diameter Ratio:** Ratio of diameters of circumscribed and inscribed sphere. The optimal (minimal) value is 3.
- **Aspect Ratio:** Aspect ratio = 3 / diameter ratio. The optimal (maximal) value is 1.
- **Dihedral Angle:** For each tetrahedron edge the dihedral angle is defined as the angle between its adjacent faces. For an equilateral tetrahedron all dihedral angles are about 70 degrees.
- **Solid Angle:** For each tetrahedron vertex the solid angle is defined as the part of the unit sphere which is occupied by the tetrahedron. For an equilateral tetrahedron all solid angles are about 30 degrees.
- **Edge Length:** If you choose this measure, a histogram of edge lengths is created for all selected tetrahedra.

**Select Angle**



If the quality measure is dihedral or solid angle, you can select whether a histogram is created for

- all angles,
- the minimal angle,
- the maximal angle, or
- minimal and maximal angle



of each tetrahedron.

### Samples



This slider lets you select the number of samples for the histogram. Normally, the default values should be adequate.

### Histogram



If you select this toggle, a plot window appears showing the histogram of qualities for all tetrahedra selected by the *Grid Volume* module. If no tetrahedra have been selected, the plot window will not be shown.

## 5.51 TriangleQuality

The *TriangleQuality* module computes the triangle qualities for a *triangular surface*. You can attach to the result a *SurfaceView* module to visualize the triangle qualities or a *Histogram* module to plot a histogram of triangle qualities. You can also use the *TriangleQuality* module in conjunction with the *Surface Editor* to detect the worst triangles and manually repair them.

Press the *Apply* button to start the computation.

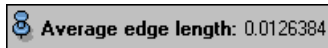
### Connections

**Data** [required]

A *triangular surface*.

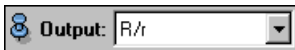
### Ports

**Average edge length**



Displays the average length of triangle edges in the input.

**Output**



This option menu lets you select between different quality measures:

- **R / r**: Ratio of diameters of circumscribed and inscribed circle. The optimal (minimal) value is 2.

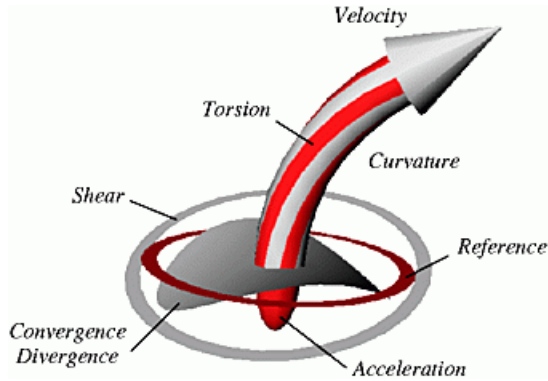


Figure 5.1: Components of the vector field probe.

- **Largest Angle:** Computes the largest angle [degree] of each triangle. For numerical applications obtuse angles above 90 degree should be avoided.
- **Dihed Angle:** Computes the dihedral angle [degree] for each pair of adjacent triangles. Small dihedral angles below 5 - 10 degree should be avoided.
- **Triangle Number:** For testing purposes only: assigns the triangle number to each triangle.

## 5.52 VectorProbe

The *VectorProbe* module allows you to interactively investigate a 3D vector field by moving around a dynamic vector field probe. The probe displays certain quantities associated to the first order derivative of the field in an intuitive way. This kind of probe has been originally proposed by W.C. de Leeuw and J.J. van Wijk in *A Probe for Local Flow Field Visualization, Proceedings of Visualization'93*, pp. 39-45. It looks as follows:

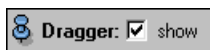
### Connections

**Data** [required]

The 3D vector field to be visualized.

### Ports

**Dragger**



Shows or hides the dragger and the vector field probe attached to it. The dragger provides a cylinder

handle and a square plate handle. The cylinder handle allows you to translate the icon along the center axis. The orientation of the cylinder can be changed using the [Ctrl] key while the mouse is located somewhere over the dragger.

### Buffer



Adds the current vector field probe to an internal buffer. In this way multiple probes can be displayed at once.

### Scale



Scales the overall size of the vector field probe.

### Length



Adjusts the length of the probe's arrow part.

## 5.53 Vectors

This is a so-called *overlay module* which can be attached to any module defining a cutting plane, e.g. *OrthoSlice* or *ArbitraryCut*. Inside this plane a 3D vector field can be visualized using a regular array of vector arrows.

### Connections

#### Data [required]

The 3D vector field to be visualized.

#### Module [required]

The module which defines the cutting plane where the arrows are placed.

#### Colormap [optional]

An optional colormap used for pseudo-coloring.

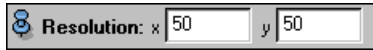
### Ports

#### Colormap



Port to select a colormap.

## Resolution



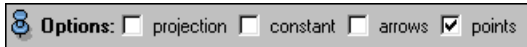
Provides two text inputs defining the resolution of the regular array of vector arrows in the plane's local x- and y-direction. The larger these values are the more arrows are displayed.

## Scale



Scaling factor used to control the length of the vector arrows.

## Options



This port provides the following toggle buttons.

*Projection:* If this option is set then 3D vectors are projected into the current plane. Otherwise, the arrows will indicate the true direction of the vector field.

*Constant:* If this option is set then all arrows will be of equal length. Otherwise, the length of the arrows is chosen to be proportional to vector magnitude.

*Arrows:* If this option is set then true arrows will be displayed. Otherwise, only simple line segments will be drawn.

*Points:* If this option is set then a little dot will be drawn at the bottom of an arrow. This is useful to highlight a sampling point at locations where the vector magnitude is so low that the arrow vanishes completely.

## Colorize



An option menu allowing to control how arrows are colored. In order to see an effect a colormap must be connected to the module. If *Magnitude* is chosen then the vector magnitude is used to lookup a color from the colormap. If *Normal component* is chosen then color denotes the signed length of the vector component perpendicular to the cutting plane. This length is positive if the vector points upwards or negative if the vector points downwards. Finally, if *Parallel component* is chosen then *color* denotes the length of the vector component tangential to the plane. This length will always be greater or equal than zero.

## Phase



This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controlling which part of the complex 3D vectors is visualized by the arrows. A value of 0 degree corresponds to the real part, while a value of 90 degrees corresponds to the imaginary part. The display can be animated with respect to the phase by the *cycle* button, this way polarization properties of the field can be revealed or wave phenomena become visible.

## Commands

`setLineWidth <value>`

Allows to change the line width of the arrows. By default arrows are drawn two pixels wide.

## 5.54 Vectors (Tetrahedra)

This display module can be attached to a *GridVolume* or a *GridBoundary* module as well as to a 3D vector field defined on a tetrahedral grid. In the latter case all vectors are displayed otherwise the vector field is displayed on the surface nodes or on all nodes of the selected tetrahedra (*GridVolume*) resp. triangles (*GridBoundary*).

The vector representation can be done using simple *lines* or *arrows*. The vector lines/arrows can be colored using a colormap taking the magnitude of the vectorfield in to account.

## Connections

**Data** [required]

The 3D tetrahedral vector field to be visualized.

**PortModule** [required]

The *GridVolume* resp. *GridBoundary* master module.

**Colormap** [optional]

An optional colormap used for coloring the magnitude of the vectors.

## Ports

**Colormap**



Choose a colormap to control how lines/arrows are colored. In order to see an effect the colormap must be connected to the module. The vector magnitude is used to lookup a color from the colormap.

**Scale**



Scaling factor used to control the length of the vector lines/arrows.

**Options**

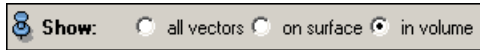


*Constant:* If this option is set then all lines/arrows will be of equal length. Otherwise, the length is chosen to be proportional to vector magnitude.

**Arrows:** Choose between simple lines and arrows

**Points:** If this option is set then a little dot will be drawn at the bottom of an arrow. This is useful to highlight a point at locations where the vector magnitude is so low that the arrow vanishes completely.

#### Show



**All Vectors:** If this radio option is on the vectors from all nodes are displayed.

**On Surface:** Only those vectors are displayed which lie on the surface of the selected boundaries.

**In Volume:** All vectors of all selected tetrahedrons are shown.

#### Phase



This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controlling which part of the complex 3D vectors is visualized by the arrows. A value of 0 degree corresponds to the real part, while a value of 90 degrees corresponds to the imaginary part. The display can be animated with respect to the phase by the *cycle* button, this way polarization properties of the field can be revealed or wave phenomena become visible.

## Commands

`setLineWidth <value>`

Allows to change the line width of the arrows. By default arrows are drawn two pixels wide.

`setPointSize <value>`

Allows to change the size of the points at the bottom of the arrows. By default points are drawn two pixels wide.

`setLogScale 0|1`

Switches logarithmic scaling on or off.

## 5.55 Vectors/Normals (Surface)

This is a multifunctional display module having the following applications:

- attached to a *SurfaceView* module, it shows the normals on the *Surface* displayed by the module. Only the normals corresponding to selected patches/triangles are displayed; the normal binding in the *SurfaceView* module (triangle normals, vertex normals or direct normals) is respected.
- attached to a 3D vector field defined on a *Surface*, it visualizes the whole vector field

- connected to both *SurfaceView* module showing a *Surface* and a 3D vector field defined *on the same Surface*, it displays the vector field only in the selected regions of the surface.

The representation can be done using simple *lines* or *arrows*. In the case of vector fields it can be chosen between a constant length or a magnitude - dependent length of the lines/arrows. The lines are colored using the colormap.

## Connections

### Data [optional]

The 3D vector field to be visualized. If there is no vector field, the normals on surface are displayed.

### Surface [optional]

The *SurfaceView* master module.

### Colormap [optional]

An optional colormap used for coloring the magnitude of the vectors.

## Ports

### Colormap



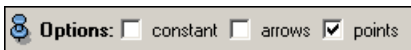
Choose a colormap to control how lines/arrows are colored. In order to see an effect the colormap must be connected to the module. The vector magnitude is used to lookup a color from the colormap.

### Scale



Scaling factor used to control the length of the vector lines/arrows.

### Options



*Constant:* (Only for vector fields) If this option is set then all lines/arrows will be of equal length. Otherwise, the length is chosen to be proportional to vector magnitude.

*Arrows:* Choose between simple lines and arrows

*Points:* (Only for vector fields) If this option is set then a little dot will be drawn at the bottom of an arrow. This is useful to highlight a point at locations where the vector magnitude is so low that the arrow vanishes completely.

## Commands

`setLineWidth <value>`

Allows to change the line width of the arrows. By default arrows are drawn two pixels wide.

`setPointSize <value>`

Allows to change the size of the points at the bottom of the arrows. By default points are drawn two pixels wide.

`setArrowSize <value>`

Allows to change the size of the arrows.





# Chapter 6

## Quantification+ Pack

### 6.1 Visilog

This module provides access to the *Visilog* software. The full range of Visilog filters can be accessed in a way that Visilog users are used to. Input and output images are connected to Visilog as in any other amira network.

#### Connections

**Data** [required]

Initial input image.

#### Ports

##### Execution



In the *on Go* mode, a Visilog command is executed after pressing go.

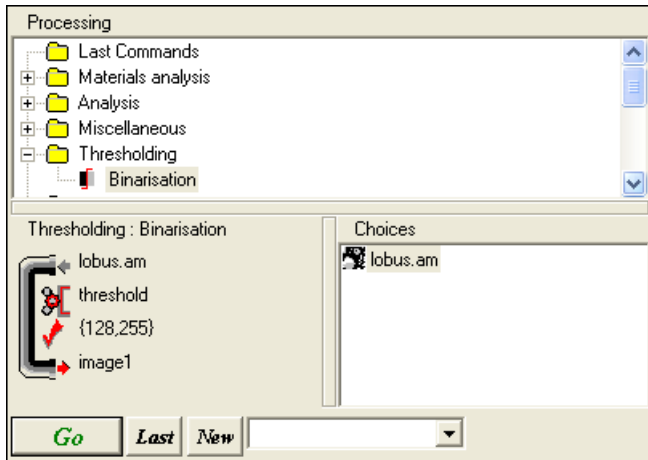
In the *immediate* mode, the current Visilog command, if valid, is executed immediately.

##### Action



Pressing the first button launches the Visilog result viewer. The second button brings up the the Visilog documentation within a PDF viewer installed on the system.

## Visilog



This port is identical to the Visilog user interface, in which the command is chosen from the command tree. Click F1 over this port to get context sensitive help on Visilog.

## Commands

DataType list

**Values for DataType :**

- DATATYPE\_TMP
- DATATYPE\_LUT
- DATATYPE\_ANALYSIS or DATATYPE\_ANL
- DATATYPE\_SEGMENT
- DATATYPE\_KNL
- DATATYPE\_GPH
- DATATYPE\_CHAIN
- DATATYPE\_ARRAY
- DATATYPE\_TAMI

getFieldCount

For a Visilog data object, get size of the field FieldName.

getFieldValue <DataName> <Datatype> <FieldName> <Val>

For a Visilog data object, get the first value of the field FieldName.

## Example

```
Visilog:algo exeCommand {cmd=extrema input=image1 param=none}  
set result "  
Visilog:algo getFieldValue Extrema DATATYPE_TMP Maximum $result
```

```
setFieldValue <DataName> < Datatype> <FieldName> <fVal>
```

For a Visilog data object, set the first value of the field FieldName.

```
getFieldArrayValue <DataName> <Datatype> <FieldName> <Pos>  
<fVal>
```

For a Visilog data object, get the value from the index pos of the field FieldName.

```
setFieldArrayValue <DataName> <Datatype> <FieldName> <Pos>  
<fVal>
```

For a Visilog data object, set value to the index pos of the field FieldName.

```
getFieldArrayValue2 <DataName> <Datatype> <FieldName> <Pos1>  
<Pos2> <fVal>
```

For a two-dimensional array, get (pos1, pos2) of the field FieldName of the data DataName.

```
setDspOutput <DataType> <bDisplay>
```

Lock or unlock display object after command (bDisplay = 0 or 1).

```
getFieldStat <DataName> <DataType> <FieldName> <StatName>  
<fval>
```

Get statistics from a data field

| Field  | Description                           |
|--------|---------------------------------------|
| min    | minimum                               |
| max    | maximum                               |
| mean   | mean                                  |
| stddev | standard deviation                    |
| sum    | sum                                   |
| Q5     | value at 5 % (percentage in number)   |
| Q10    | value at 10 % (percentage in number)  |
| Q25    | value at 25 % (percentage in number)  |
| Q50    | value at 50 % (percentage in number)  |
| Q75    | value at 75 % (percentage in number)  |
| Q90    | value at 90 % (percentage in number)  |
| Q95    | value at 95 % (percentage in number)  |
| T5     | value at 5 % (percentage in measure)  |
| T10    | value at 10 % (percentage in measure) |
| T25    | value at 25 % (percentage in measure) |
| T50    | value at 50 % (percentage in measure) |
| T75    | value at 75 % (percentage in measure) |
| T90    | value at 90 % (percentage in measure) |
| T95    | value at 95 % (percentage in measure) |

`createObject <DataName> <datatype> <param>`

Create a Visilog data object (array, polynome, analyze, ...).

`removeObject <DataName> <Datatype>`

Remove a Visilog data object (array, polynome, analyze, ...).

`addField <DataName>, <DataType> <FieldName> <Size>`

Add a field to a Visilog object.

`getInfoImage <ImageName> <Field> <Value>`

Returns a field of an image.

| Field              | Description                              |
|--------------------|------------------------------------------|
| ox                 | X origin                                 |
| oy                 | Y origin                                 |
| oz                 | Z origin                                 |
| ot                 | Time origin ( for sequence )             |
| oc                 | Color origin ( color channel )           |
| gx                 | X image size, Pixels per row             |
| gy                 | Y image size, Rows per plane             |
| gz                 | Z image size, Number of planes Z         |
| gt                 | Time image size                          |
| gc                 | Color image size                         |
| ocode              | Number of bytes per pixel when in buffer |
| gtype              | The image type                           |
| gcode              | Arithmetic type                          |
| gbits              | Number of bits per pixels                |
| other header fiels | Calibration , .....                      |

`printText <text>`

Print text in a notepad of ResultViewer

`setCommand <command>`

Initialize the Visilog Mmodule with a command (you must use *fire* to execute it)

`exeCommand <command>`

Execute a Visilog command.

### Example:

```
Visilog exeCommand "cmd=threshold input=grey_ima param={0,100} output=bin_
Visilog exeCommand "cmd=I_analyze input=grey_ima param= outanl=label_ima"
```

`setPlaneMode <z loop> <color loop> <sequence loop>`

Lock or unlock a loop process (0 for lock and 1 for unlock)

`setCurPlane <z plane> <color plane> <sequence plane>`

Set a specific plane for a locked loop process.

`closeDiskImage <disk image name>`

Close a disk image (but don't delete the file).

**Example:**

```
Visilog closeDiskImage input=c:\\tmp\\image1_bin.im6
```

```
exeIp <Ip function and parameters>
```

Execute a Visilog Ip function.

**Example:**

```
Visilog exeIp "ip=IpCreateImage input=grey_ima output=ima2 param=4"    # UCH  
Visilog exeIp "ip=IpCopy input=grey_ima output=ima2"
```

## 6.2 VisilogAnalyse

This module represents the analysis function of Visilog.

The dragger can be moved either by selecting one of its handles and moving it to the target position or by clicking on the target position with the middle mouse button. If the user selects a field in the result viewer, the dragger is set to the corresponding position.

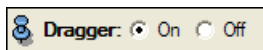
### Connections

**Data** [required]

Field to be analyzed.

### Ports

**Dragger**



Show or hide the dragger.

## 6.3 VisilogThreshold

This tool allows thresholds to be selected interactively. The current selection is displayed in an orthoslice representation. This tool is used by the *Visilog I\_threshold* command. When the tool is not called from Visilog, it can create a label field itself.

Press the *Apply* button to create the binary image. A new field is created that is 1 for each value within the threshold interval and 0 for all other field values. The *Data* input port not visible when the tool is called from Visilog.

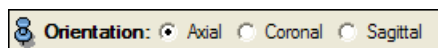
### Connections

**Data** [required]

Image data the thresholds are applied to.

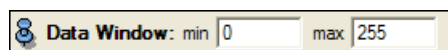
### Ports

#### Orientation



This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

#### Data Window



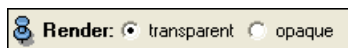
It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, values above the upper bound are mapped to white.

#### Slice Number



It allows you to restrict the range of visible data values. Values below the lower bound are mapped to black, values above the upper bound are mapped to white.

#### Render



Labeled voxels can be rendered either in a transparent or opaque style.

#### Minimum-threshold



Specifies the lower value of the threshold interval.



### Maximum-threshold



Specifies the upper value of the threshold interval.

## **Part II**

# **Alphabetic Index of Ports**



# Chapter 7

## Amira

### 7.1 Port

This is the base class of all ports in amira.

Ports are used to interact with an amira object. There are several different types of ports, e.g., option menus, sliders, or toggle buttons. If an object is selected its ports are displayed in the lower part of the amira main window, the so-called Properties Area. An object may choose not to display all ports at all times, depending on the internal state of the object. Most ports have a pin, allowing to keep the port visible even if the the object the port belongs to is deselected.

*Connections* are special ports, allowing to specify dependencies between objects. In contrast to other ports connections usually don't have their own user interface, but they are represented by corresponding lines in the Pool.

Every port provides its own Tcl command interface. This allows script programmers to interact with the port in an automated way. In addition to the commands defined by a port itself also all commands of the port's base class are available (this is the same as for amira modules and data classes). Port commands are called using the following syntax:

```
<modulename> <portname> <commandname> [optional parameters]
```

In order to get a list of all ports of an object you can use the command `<modulename> allPorts`. In most cases, but not in all cases, the name of a port used in Tcl commands is equal to the port's label as displayed in the graphical user interface, with the only exception that the name usually starts with a lower case letter and that white spaces are omitted.

## Commands

### `help`

Displays all commands available for the port. The same result can be obtained by just calling the port without any command.

### `isNew`

Returns 1 if the port was changed after the object was fired the last time, and 0 otherwise.

### `getState`

Returns a string which later on can be used to restore the state of a port using the command `setState`. The format of the state string is not specified and it may be different for each port. Usually the state string contains the same information which is also stored in *amira* network files, but not for example label strings which might have been modified using the script interface as well.

### `setState <state-string>`

Restores the state of a port from a string previously obtained using the `getState` command.

### `getLabel`

Returns the label of the port displayed in graphical user interface, for example *Options*:

### `setLabel <label>`

Sets the label of the port.

### `getLabelWidth`

Returns the width of the port's label in pixels.

### `setLabelWidth <width>`

Sets the width of the port's label in pixels, see also `align`.

### `align <port2> <port3> ...`

Align this and all other ports specified as arguments so that all labels have the same width.

### `getPin`

Check if the port has been pinned or not.

### `setPin <value>`

Set or unset the port's pin button. If the port is pinned it will be displayed in the Properties Area even if the owner of the port is deselected.

### `touch`

Touch the port to simulate a change of value.

### `untouch`

Untouch the port so that `isNew` will return 0 the next time even if the port was changed.

### `object`

Returns the name of the object the port belongs to.

**send**

Fires the object the port belongs to as well as all downstream objects.

**show**

Shows the port, provided the owning object is selected.

**hide**

Hides the object, provided the owning object is selected.

**isVisible**

Returns 1 if the port is shown, or 0 if it is hidden.

**reposition <index>**

Changes the position of the port within the object's port list and in the graphical user interface. Note that connection ports are counted too even if they may not have any controls.

**isOfType <typeid>**

Checks if this port is of the specified type or derived from it.

**getTypeId**

Returns the port's type name.

## 7.2 Connection

dfsdfsdfsdfsdf This port represents a connection from one object to another. It is commonly used to specify on which input objects some other object should depend on. A connection does not provide its own user interface like other ports, but is visually represented in the Pool by a line between the source object and the object the connection port belongs to. In order to change the connection this line can be picked with the mouse and dragged to some other source object. All connection ports of an object are listed in a special popup menu which is activated by clicking with the right mouse button in the connection area, the small white square on the left side of the object's icon.

### Commands

Inherits all commands of *Port*.

**source**

Returns the name of the source object connected to this port. If there is no source object connected an empty string is returned.

**connect <source>**

Tries to connect the port with the specified source object. If the connection cannot be established a Tcl error is generated.

## **disconnect**

Disconnects the port from any source object.

## **setTightness {0|1}**

Sets the tightness flag of the port. If the connection is tight no connection line is drawn in the Pool. Instead, the icon of the owning object is glued together with the icon of the source object, so that both icons comprise an icon group. Each object can only have one connection port with tightness set to 1.

## **isTight**

Checks if the connection is tight or not (see above).

## **setVisibility {0|1}**

Sets the visibility flag of the port. If visibility is switched off the connection will not be represented by a line in the Pool.

## **isVisible**

Checks if the connection is visible or not (see above).

## **allowEditing {0|1}**

Allow or disallow interactive editing of the connection. If editing is disallowed the connection cannot be changed interactively by moving the connection line to some other source object.

## **isEditable**

Checks if the connection can be edited interactively or not.

## **validSource <source>**

Checks if the specified object can be connected to this port. If this is the case 1 is returned, otherwise 0 is returned.

# **7.3 MasterConnection**

This port represents a special connection port which is provided by every data object. It allows to connect the data object to a particular slot of a compute module or to an editor. A master connection is not available for script objects.

## **Commands**

Inherits all commands of *Connection*.

## **editor**

Returns the name the editor which currently has control over the data object, i.e., which possibly modifies the data object.

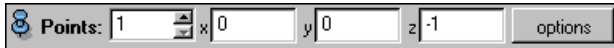
`connect <source> [<slot>]`

If this command is called with only one argument, it behaves like the `connect` command of an ordinary connection. If this command is called with two arguments and if the source object is a compute module, it connects the data object the master connection belongs to to the specified result slot of the compute module. Result slots are used to allow compute modules to have multiple result objects.

## 7.4 Port3DPointList

This port lets you enter an arbitrary number of 3D points. The points can be specified either by typing in their coordinates in appropriate text fields or by activating a dragger in the 3D viewer. In order to move the dragger you have to put the viewer into interaction mode (press the **ESC** key). Then move the mouse over one of the dragger's crosshairs and push the left mouse button. The color of the picked crosshair changes and a gray transparent plane is shown. The moving area of the dragger is restricted to this plane. The new coordinates are delivered to the application when you release the mouse button, or, if *immediate mode* is on, while you are moving. If the *orthogonal mode* is on, all other points are moved in sync. You can only move one point, the current point, at a time. Type **TAB** in the viewer window to cycle through all points if you want to change the current point. You can also hit the up or down arrows of the port's point index spin box. These arrows are shown only if the port manages a list of at least 3 3D points. The current point is shown as a red sphere, all other points as yellowish spheres.

Note: You can pick a location with the middle mouse button on a pickable object in the scene, e.g., an *OrthoSlice*, to set the current point to that location.



This port is used for example by the *data probing* modules.

### Commands

Inherits all commands of *Port*.

`getNumPoints`

Returns the current number of points defined by this port.

`getValue [<index>]`

Returns the coordinates of point `<index>` or of the current point, if no argument was specified.

`setValue [<index>] <x> <y> <z>`

Sets the coordinates of point `<index>` or of the current point, if no argument was specified. The coordinates are automatically clipped against the bounding box of the dragger.



### **getBoundingBox**

Returns the bounding box of the dragger. The command returns six number, namely the xmin, xmax, ymin, ymax, zmin, and zmax coordinates of the bounding box in that order.

### **setBoundingBox <xmin> <xmax> <ymin> <ymax> <zmin> <zmax>**

Sets the bounding box of the dragger. Most modules adjust the bounding box if the module is connected to a new input data object.

### **getFormat**

Returns the format used for the coordinate text fields.

### **setFormat <format>**

Sets the format used for the coordinate text fields in printf syntax. The default is %g.

### **getImmediate**

Checks whether immediate mode is enabled or not.

### **setImmediate {0|1}**

Enables or disables immediate mode. If immediate mode is enabled the module the port belongs to is fired permanently while the dragger is moved. Otherwise the module is fired only once after the dragger was moved.

### **getOrtho**

Checks whether orthogonal mode enabled or not.

### **setOrtho {0|1}**

Enables or disables orthogonal mode. In orthogonal mode all points of the port are moved at the same time when the dragger is moved. Otherwise only the current point is moved.

### **showDragger**

Shows the dragger at the current point.

### **hideDragger**

Hides the dragger.

### **showPoints**

Shows the points. The points are represented by little red spheres.

### **hidePoints**

Hides the points.

### **getPointSize**

Returns the current point size (radius) in absolute coordinates.

### **setPointSize <radius>**

Sets the point size (radius). The point size is automatically adjusted if the dragger's bounding box changes.

### getPointScale

Returns the current point size scale factor.

### setPointScale <factor>

Sets the point size scale factor. By default the scale factor is 1. If you find that the default size of the points is too small or too big you can adjust this factor.

### appendPoint [<x> <y> <z>]

Appends a new point to the port's point list. If no coordinates are specified a point with some default coordinates will be appended. If the number of points exceeds some limit (which cannot be modified via the script interface) nothing happens and -1 is returned. Otherwise the index of the appended point is returned.

### insertPoint <index> [<x> <y> <z>]

Inserts a new point at position *index* in the port's point list. If no coordinates are specified a point with some default coordinates will be inserted. If the number of points exceeds some limit (which cannot be modified via the script interface) nothing happens and -1 is returned. Otherwise *<index>* is returned.

### removePoint <index>

Removes point *<index>* unless a minimal number of points (which cannot be modified via the script interface) is reached. If the point couldn't be removed 0 is returned. Otherwise 1 is returned.

## 7.5 PortButtonList

This port provides an arbitrary number of push buttons. The buttons are implicitly numbered 0, 1, 2, ... from left to right. When a button is pushed the port's new flag is set. The index of the pushed button can be obtained using the command `getValue`. After the module was fired the port's new flag is unset again and `getValue` returns -1. This means that the port does not store a permanent state. A push button is frequently used to trigger some action.



Modules using this port are for example *GridVolume* or *ObliqueSlice*.

### Commands

Inherits all commands of *Port*.

#### getValue

Returns the index of the pushed button. If no button was pressed since the module was fired the last time -1 is returned. The method does not interpret the snap toggle. In order to handle snapped buttons better use the command `wasHit` (see below).

**setValue** <index>

Marks button `index` as being pushed.

**wasShiftDown**

Checks if the shift key was down the last time a button was pressed.

**wasCtrlDown**

Checks if the control key was down the last time a button was pressed.

**wasAltDown**

Checks if the alt key was down the last time a button was pressed.

**setShiftDown** {0|1}

Sets the shift key modifier flag.

**setCtrlDown** {0|1}

Sets the control key modifier flag.

**setAltDown** {0|1}

Sets the alt key modifier flag.

**getSensitivity** [<index>]

Checks whether the first button or button `index` is enabled or disabled.

**setSensitivity** [<index>] {0|1}

Enables or disables the first button or button `index`. If a button is disabled it is grayed out and cannot be pushed anymore.

**getLabel** [<index>]

Returns the label of button `index` or the port's label.

**setLabel** [<index>] <label>

Sets the label of button `index` of the port's label.

**setCmd** [<index>] <command>

Sets a Tcl command which is executed when the specified button is pressed. This feature is especially useful in script objects since it avoids writing long Tcl code with many if statements. If a command has been set for a button the owning module will not be fired and the port will not be touched when the button is pressed. The command will be executed in the global Tcl namespace. The variable `this` refers to the owning module.

**getCmd** [<index>]

Returns the Tcl command associated with the specified button or an empty string if no Tcl command has been set.

**getNumButtons**

Returns the number of buttons of the port.

**setNumButtons** <number>

Sets the number of buttons of the port. New buttons are created with an empty label.

**enableSnapToggle** [<index>] {0|1}

Enable or disable the snap toggle for the first button or for button *index*.

**snap** [<index>] {0|1}

Snap or unsnap a button. The button's snap toggle must be enabled.

**isSnapped** [<index>]

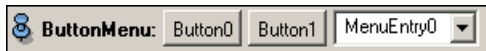
Check whether the specified button is snapped or not.

**wasHit** [<index>]

Returns 1 if the specified button was hit or if its snap toggle is down.

## 7.6 PortButtonMenu

This port combines an arbitrary number of push buttons with one or more option menus containing an arbitrary number of options. The port is derived from *PortButtonList* and hence inherits its behavior.



This port is used for example by the modules *FieldCut* or *DisplayISL*.

### Commands

Inherits all commands of *PortButtonList*.

**setNumOptEntries** [<menu>] <number>

Sets the number of entries in the option menu.

**getNumOptEntries** [<menu>]

Returns the number of entries in the option menu.

**setOptValue** [<menu>] <index>

Selects item <index> in the option menu.

**setOptValueString** [<menu>] <label>

Selects the item with the specified label string in menu <menu> or in the first option menu, if only one argument is given. If no item with such a label exists, nothing happens and 0 is returned. Otherwise 1 is returned.

**getOptValue** [<menu>]

Returns the index of the selected item in the option menu.

`setOptLabel [<menu>] <index> <label>`

Sets the label of the option menu item specified by <index>.

`getOptLabel [<menu>] <index>`

Returns the label of the option menu item specified by <index>.

`setOptSensitivity [<menu>] <index> {0|1}`

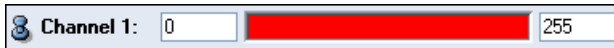
Enables or disables a particular item in the option menu.

`getOptSensitivity [<menu>] <index>`

Check whether a particular item in the option menu is enabled or not.

## 7.7 PortChannelConfig

This is a special port used exclusively by *multi-channel fields*. There is one such port for each channel of a multi-channel field. Derived from *connection*, this port manages the link to the actual channel object. In addition, it provides two text fields allowing the user to define a data range used for mapping the channel data to gray values. The channel's colormap can be set using the colormap port. This port is not available for script objects.



### Commands

Inherits all commands of *Connection*.

`getMinValue`

Returns the lower value of the channel's data window.

`setMinValue <value>`

Sets the lower value of the channel's data window.

`getMaxValue`

Returns the upper value of the channel's data window.

`setMaxValue <value>`

Sets the upper value of the channel's data window.

`getColor`

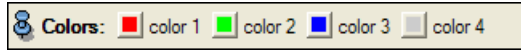
Returns the channel's color as an RGB tuple of floating point values. (obsolete)

`setColor <color>`

Sets the channel's color. The color can be specified either as a tuple of three RGB integer values in the range 0...255, or as a tuple of three RGB floating point values in the range 0...1, or as a text string (e.g., blue or red). (obsolete)

## 7.8 PortColorList

This port provides one or more color buttons which can be used to define a constant color. Colors can be easily edited using the *amira* color dialog which is popped up when one of the color buttons is pushed.



This port is used for example by the *Axis* module.

### Commands

Inherits all commands of *Port*.

**setColor** <index> <color>

Sets the color of button <index>. The color can be specified either as a tuple of three RGB integer values in the range 0...255, or as a tuple of three RGB floating point values in the range 0...1, or as a text string (e.g., blue or red).

**getColor** <index>

Returns the color of button index as an RGB tuple of three floating point numbers in the range 0...1.

**setAlpha** <index> <alpha>

Sets the alpha value of button <index>.

**getAlpha** <index>

Returns the alpha value of button <index>.

**setLabel** [<index>] <label>

Sets the label of button index of the port's label, if only one argument is specified.

**getLabel** [<index>]

Returns the label of button index or the port's label, if no argument is specified.

**setNumButtons** <number>

Sets the number of color buttons of the port.

**getNumButtons**

Returns the number of color buttons of the port.

**setContinuousMode** {0|1}

Enables or disables continuous mode. If continuous mode is activated the owning module is fired permanently when changing a color using the color dialog.

**getContinuousMode**

Checks whether continuous mode is enabled or not.

## 7.9 PortColormap

This port provides a connection to a *colormap* object. In contrast to a standard *connection* port, this port has a user interface, i.e., it is represented in the Properties Area like any other port if the owning object is selected. In particular, the contents of the connected colormap as well as its range are shown. New colormaps can be quickly connected to the port via a context menu which pops up when pressing the right mouse button over the colormap area of the port.

The context menu item *Adjust range* allows you to automatically adjust the colormap range to the connected data range.

The context menu also allows you to switch between global and local range mode. In global range mode the coordinates used to map data values to colors are taken from the colormap itself. If the same colormap is used by two different modules and if the range then is modified, both modules are updated. In contrast, in local range mode the coordinates are defined by the port itself. Thus, although the same colormap might be used by two different modules, the ranges still can be different.

If no colormap is connected to the port, still a default color and a default alpha value can be defined. The default colors can be modified via the *amira* color dialog which is popped up by double-clicking on the colormap area with the left mouse button. If a colormap is connected, a left mouse button double-click makes the icon of the colormap visible in the Pool.



**See also:** *Colormap*, *ColormapEditor*

### Commands

Inherits all commands of *Connection*.

`setDefaultColor <red> <green> <blue>`

Sets the default color of the port. An RGB tuple of three floating point values between 0...1 must be specified.

`getDefaultColor`

Returns the default color of the port.

`setDefaultAlpha <alpha>`

Sets the default alpha (opacity) value of the port.

`getDefaultAlpha`

Returns the default alpha (opacity) value of the port.

`enableAlpha {0|1}`

Enables or disables the display of the alpha channel in the port's colormap area. Even if the alpha channel is not displayed by the port it is up to the owning module to interpret alpha values or not.

`isAlphaEnabled`

Checks if alpha values are displayed by the port or not.

`setLocalRange {0|1}`

Enables or disables the local range feature (see description above).

`getLocalRange`

Checks if the local range feature is enabled or not.

`setLocalMinMax <min> <max>`

Sets the coordinate range used for mapping data values to colors when the local range feature is enabled. The port is touched but the network is not fired.

`getLocalMinValue`

Returns the lower bound of the local range.

`setLocalMinValue <min>`

Sets the lower bound of the local range.

`getLocalMaxValue`

Returns the upper bound of the local range.

`setLocalMaxValue <max>`

Sets the upper bound of the local range.

`setMinMax <min> <max>`

Sets the coordinate range used for mapping data values to colors. If local range is enabled or if no colormap is connected the local range is updated. Otherwise the range of the connected colormap is updated. In both cases the network is fired, i.e., the module owning this port or modules connected to the modified colormap are updated.

## 7.10 PortDoIt

This port is essentially identical to *PortButtonList*. The only difference is that this port always comes up with a single button initially and that this button has a snap toggle by default.



or



This port is mainly used by compute modules, for example *Arithmetic* or *CastField*.



**Note:** Since amira 4.0, the `HxPortDoIt` port is not by default visible in the control panel of its associated module. Rather, the fact that a module has an `HxPortDoIt` activates (makes green) the *Apply* button at the bottom of the Properties Area. Checking the *auto-refresh* toggle is equivalent to snapping down the red *DoIt* port snap toggle. To request display of the *DoIt* port in the module control panel, check the *Show "DoIt" buttons* box in the *Layout* tab of the *Edit/Preferences* dialog.

## Commands

Inherits all commands of *PortButtonList*.

### 7.11 PortDrawStyle

This is a special port for adjusting the drawstyle of display modules derived from *ViewBase*. The port provides a combo box (option menu) offering the following drawstyles:

- outlined
- shaded
- lines
- points
- transparent

In addition, the port allows you to set additional options via a popup menu which is activated by pressing the *more options* button. For a detailed description of these options please refer to the documentation of *ViewBase*. This port is not available for script objects.



## Commands

Inherits all commands of *Port*.

`getValue`

Returns the index of the current draw style.

`setValue <index>`

Sets the current draw style. The draw styles are indexed in the same order as they appear in the combo box, i.e., 0=outlined, 1=shaded, 2=lines, 3=points, 4=transparent.

`isTexture`

Checks if 1D textures for pseudo-coloring are enabled (as opposed to Gouraud shading).

`setTexture {0|1}`

Enables or disables 1D textures for pseudo-coloring.

**getNormalBinding**

Returns an index describing the current normal binding mode.

**setNormalBinding** {0|1|2}

Sets the normal binding. The indices have the following meaning: 0=triangle normals, 1=vertex normals, 2=direct normals (computed using a crease angle criterion, compare *ViewBase*).

## 7.12 PortFilename

This port is used to define a file name. The file name may be either manually typed in the text field, or it may be selected using the *amira* file browser. The file browser can operate in three modes, one allowing to select any file, one allowing to select one existing file only, and one allowing to select multiple existing files. However, if the file name is typed in manually no check is performed to ensure that the file really exists. Typically the first mode is used for saving while the second and third modes are used for loading. Associated with every filename is a filetype.



This port is used for example by the *TetraGen* module.

### Commands

Inherits all commands of *Port*.

**getFilename**

Returns the filename or a list of filenames.

**getFileType**

Returns the filetype or a list of filetypes.

**setFilename** <filename> <filetype>

Sets the filename.

**setFilenames** [list <filename1> <filename2> ...] [list <filetype1> <filetype2> ...]

Sets a list of filenames in *multi file* mode.

**getValue**

Returns the filename (same as **getFilename**) or a list of filenames.

**setValue** <filename> <filetype>

Sets the filename (same as **setFilename** or **setFilenames** in *multi file* mode).

**getMode**

Returns 0 if the file browser operates in *any file* mode, 1 if it operates in *existing file* mode, or 2 if it operates in *multi file* mode (see description above).

**setMode** {0|1|2}

Sets the mode of the file browser (0 for *any file*, 1 for *existing file*, 2 for *multi file* (see description above).

**registerFileType** [<formatname>] [<extension>]

If the file browser is opened in *any file* mode, it shows a combo box allowing the user to specify an optional format to be used to save a data set. This command allows you to register format names and optional file extensions which are added to the file type combo box. Calling this command without any arguments removes all previously registered file types.

**exec**

Pops up the file browser. The file browser is opened in modal mode, i.e., the command blocks until the user closed the dialog again. There is no way to close an open file browser using a Tcl command. The method returns 0 if the browser was closed using the cancel button and the port's filename field was not updated. Otherwise it returns 1.

## 7.13 PortFloatSlider

This port provides a slider allowing the user to specify a floating point number. The floating point number is limited to the current range of the slider. The range can be changed either using the Tcl command **setMinMax** or interactively using a configure dialog, which can be popped up by pressing the right mouse button over the slider.

Optionally the slider may have arrow buttons allowing the user to decrease or increase the value by some predefined increment. The slider may also have so-called subrange buttons allowing the user to restrict the actual range of the slider to a smaller subrange. In this way it is possible to interactively adjust the slider's value with high precision. In order to numerically adjust the lower (left) subrange value, the command **L <value>** can be typed in the slider's text field. In order to numerically adjust the upper (right) subrange value, the command **R <value>** can be typed in the slider's text field.



This port is used for example to specify the threshold of the *Isosurface* module.

### Commands

Inherits all commands of *Port*.

**getValue**

Returns the float value.

**setValue <float>**

Sets the float value.

**getMinValue**

Returns the lower bound.

**getMaxValue**

Returns the upper bound.

**setMinMax** <min> <max>

Sets the lower and upper bounds of the slider.

**setFormat** <format>

Sets the format used for the slider's text field in printf syntax. Typically, the default format is %g.

**getFormat**

Returns the printf format used for the slider's text field.

**setIncrement** <increment>

Sets the increment. The increment is used for adjusting the slider value using the optional arrows buttons. It is also used to constrain the slider value in discrete mode (see below).

**getIncrement**

Returns the slider's increment.

**setButtons** {0|1}

Enables or disables the display of the optional arrow buttons.

**hasButtons**

Checks whether the optional arrow buttons are enabled or not.

**setSliderWidth** <width>

Sets the width of the slider in pixels.

**getSliderWidth**

Returns the width of the slider in pixels.

**setNumColumns** <columns>

Sets the width of the slider's text field in characters.

**getNumColumns**

Returns the width of the slider's text field in characters.

**setTracking** {0|1}

Enables or disables tracking mode. If tracking mode is enabled, the network is fired permanently while the user is moving the slider. Otherwise it is fired only once when the slider is released.

**getTracking**

Checks whether tracking mode is enabled or not.

`setDiscrete {0|1}`

Enables or disables discrete mode. If discrete mode is enabled, the slider value will always be equal to the lower bound plus an integer multiple of the current increment.

`getDiscrete`

Checks whether discrete mode is enabled or not.

`setSubRangeButtons {0|1}`

Enables or disables the subrange buttons (see description above),

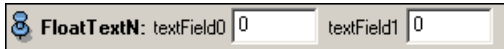
`hasSubRangeButtons`

Checks whether the subrange buttons are enabled or not.

## 7.14 PortFloatTextN

This port represents a variable number of bounded float values which can be edited interactively. The actual number of float fields can be changed at run-time. Each float field has a label, a printf-style format string, and a lower and upper bound used to constrain the value of the text field. minimum and maximum values, as well as the sensitivity of each field.

In order to quickly change the value of a float field, a *virtual slider* is provided. The virtual slider is activated by clicking into a text field with the shift key held down, and then moving the mouse up or down.



This port is used for example by the *Resample* module.

### Commands

Inherits all commands of *Port*.

`getValue [<index>]`

Returns the value of the text field <index> or of the first text field, if no argument is specified.

`getMinValue [<index>]`

Returns the lower bound of the text field <index> or of the first text field, if no argument is specified.

`getMaxValue [<index>]`

Returns the upper bound of the text field <index> or of the first text field, if no argument is specified.

`setValue [<index>] <value>`

Sets the value of the text field <index> or of the first text field.

**setMinMax** [*<index>*] *<min>* *<max>*

Sets the lower and upper bound of the text field *<index>* or of the first text field.

**setValues** *<value1>* *<value2>* ... *<valueN>*

Sets the values of all text fields. The number of arguments must match the number of text fields.

**setMinMaxAll** *<min1>* *<max1>* ... *<minN>* *<maxN>*

Sets the lower and upper bounds of all text fields. The number of arguments must be equal to twice the number of text fields.

**getFormat** [*<index>*]

Returns the printf format string of the specified text field.

**setFormat** [*<index>*] *<format>*

Sets the printf format string of the specified text field.

**getSensitivity** [*<index>*]

Checks if the specified text field is enabled or not. Insensitive text fields are grayed out and they accept do not accept user input.

**setSensitivity** [*<index>*] {0|1}

Enables or disables the specified text field.

**getLabel** [*<index>*]

Returns the label of text field *index* or the port's label.

**setLabel** [*<index>*] *<label>*

Sets the label of text field *index* of the port's label.

**setPart** *<index>* {0|1}

Shows or hides the specified text field. A hidden text field is not displayed at all (in contrast to an insensitive one), but still exists and stores a value.

**getNum**

Returns the current number of text fields.

**setNum** *<value>*

Sets the current number of text fields.

## 7.15 PortGeneric

This is a generic port which can be dynamically configured in various way. Several different predefined user-interface components can be used:

- text fields for integer values

- text fields for floating point values
- check boxes
- groups of radio buttons
- combo boxes (option menus)
- push buttons
- labels

Interface components inserted into this port are identified using a unique ID, which must be specified when creating the component. This ID is used in all commands to set or get the value of the particular component.

The design goal of this class was primarily ease-of-use. Thus flexibility is somewhat limited. If you want to create a port with custom GUI components, you should use *Developer Pack* and derive a new class from *Port* using Qt widgets.

This port is used for example by the *TransformEditor*.

## Commands

Inherits all commands of *Port*.

**getValue <id>**

Returns the value of the specified component. For combo boxes and radio groups the index of the selected item is returned.

**setValue <id> <value>**

Sets the value of the specified component. If the specified component is a combo box or a radio group and if value is not a number, then the value is interpreted as a label and the combo box item or the radio button with that label is selected.

**getColor <id>**

Returns the color of a color button inserted via the command *insertColorButton*. If the component <id> is not a color button, the result is undefined.

**setColor <id> <color>**

Sets the color of a color button with the specified id. If the component <id> is not a color button, the command returns immediately.

**isItemNew <id>**

Checks if the specified component was modified since the owning module was fired the last time.

**deleteItem <id>**

Deletes the specified item.

**insertIntText <id> [<columns> [<index>]]**

Inserts an integer text field. In order to set the text field to e.g., 27, use *setValue <id> 27*. In

order to query the value of the text field, use `getValue <id>`. `<columns>` specifies the width of the text field. `<index>` specifies the position where the text field should be inserted. If this argument is omitted, the text field is appended after all other elements.

`insertFloatText <id> [<format> [<columns> [<index>]]]`

Inserts a floating point text field. In order to set the text field to e.g., 5.5, use `setValue <id> 5.5`. In order to query the value of the text field, use `getValue <id>`. `<format>` specifies the printf format of the text field. Usually `%g` is a good choice. `<columns>` specifies the width of the text field. `<index>` specifies the position where the text field should be inserted. If this argument is omitted, the text field is appended after all other elements.

`insertCheckBox <id> [<label> [<index>]]`

Inserts a check box with a label. In order to set the check box use `setValue <id> {0|1}`. In order to query the value of the check box use `getValue <id>`. `<index>` specifies the position where the check box should be inserted. If this argument is omitted, the text field is appended after all other elements.

`insertRadioGroup <id> <num> <label1> ... <labelN> [<index>]`

Inserts a group of `n` radio buttons. The button labels are specified by `<label1>` to `<labelN>`. The number of labels must match the number of radio buttons. In order to set, for example, the second radio button (and unset all others), use `setValue <id> 1`. In order to query which radio button is checked, use `getValue <id>`. `<index>` specifies the position where the radio group should be inserted. If this argument is omitted, the radio group is appended after all other elements.

`insertComboBox <id> <num> <label1> ... <labelN> [<index>]`

Inserts a combo box with `n` items. The item's labels are specified by `<label1>` to `<labelN>`. The number of labels must match the number of radio buttons. In order to select, for example, the second item (and unset all others), use `setValue <id> 1`. In order to query which item is selected, use `getValue <id>`. `<index>` specifies the position where the combo box should be inserted. If this argument is omitted, the combo box is appended after all other elements.

`insertPushButton <id> <label> [<index>]`

Inserts a simple push button. `<label>` specifies the label of the button. In order to simulate a button press event, use `setValue <id> 1`. In order to check if the button was pressed, use the command `isItemNew <id>`. `<index>` specifies the position where the button should be inserted. If this argument is omitted, the button is appended after all other elements.

`insertColorButton <id> [<index>]`

Inserts a color button. The button can be used to specify an RGB color. Pushing the button opens the color dialog, In order to set the color, use the command `setColor <id> <color>`. In order to query the color, use the command `getColor <id>`. `<index>` specifies the position where the button should be inserted. If this argument is omitted, the button is appended after all other elements.

`insertLabel <id> <label> [<index>]`



Inserts a label. `<index>` specifies the position where the label should be inserted. If this argument is omitted, the label is appended after all other elements.

**setSensitivity** `<id> {0|1}`

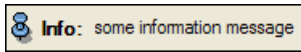
Sets the sensitivity of an element. If an element is insensitive, it is grayed out and accepts no user input. By default, all new elements are sensitive.

**getSensitivity** `<id>`

Returns the sensitivity of an element.

## 7.16 PortInfo

This port displays a simple single line informational message. It does not allow any user interaction, and thus never causes the owning object and the network to be fired.



This port is used for example by the *Interpolate* module.

### Commands

Inherits all commands of *Port*.

**getValue**

Returns the text displayed by the port.

**setValue** `<text>`

Sets the text displayed by the port.

**printf** `<fmt> <arg1> <arg2> ...`

Sets the text displayed by the port using a C-style printf command. `<fmt>` is a message string which may contain %S format specification fields. These fields are substituted by the arguments. Note that only %S string type fields must be used, but no numerical format fields like %d or %g.

## 7.17 PortIntSlider

Slider port representing an integer value. This class is derived from *PortFloatSlider* and it provides exactly the same features and commands. The only exception is that the values returned by **getValue**, **getMinValue**, and **getMaxValue** will be rounded to the nearest integer value. In addition, the default format string used by the slider's text field is `%.0f` which is suitable for integer values.



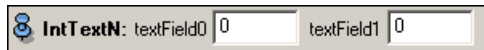
Modules using this port are for example *OrthoSlice* or *Vectors*.

## Commands

Inherits all commands of *PortFloatSlider*.

### 7.18 PortIntTextN

This port provides an arbitrary number of text fields for entering integer values. The port is derived from *PortIntTextN* and it provides exactly the same features and commands. The only exception is that the values returned by `getValue`, `getMinValue`, and `getMaxValue` will be rounded to the nearest integer value. In addition, the default format string used for the text fields is `%.0f` which is suitable for integer values.



This port is used for example by the *Resample* module.

## Commands

Inherits all commands of *PortFloatTextN*.

### 7.19 PortMultiChannel

This port is commonly used by modules operating on *multi-channel fields*. The port provides a set of toggle buttons, one for each channel of the multi-channel field. Using these toggle buttons individual channels of the multi-channel field can be quickly switch on or off.



This port is used for example by the *ProjectionView* module.

## Commands

Inherits all commands of *Port*.

`getValue <index>`

Checks whether channel `<index>` is switched on or off.

`setValue <index> {0|1}`

Switches channel `<index>` on or off.

`getNum`

Returns the number of channels controlled by this port.

## 7.20 PortMultiMenu

This port provides one or more combo boxes with a variable number of items each. At any time exactly one item is selected in each combo box.



Modules using this port are for example *SurfaceView* or *GridBoundary*.

### Commands

Inherits all commands of *Port*.

`getValue [<menuindex>]`

Returns the index of the item currently being selected in the specified menu or in the first menu, if no argument is given.

`setValue [<menuindex>] <itemindex>`

Selects item `<itemindex>` in the specified menu or in the first menu, if only one argument is given.

`setValueString [<menuindex>] <label>`

Selects the item with the specified label string in menu `<menuindex>` or in the first menu, if only one argument is given. If no item with such a label exists, nothing happens and 0 is returned. Otherwise 1 is returned.

`setNum [<menuindex>] <numitems>`

Sets the number of items in menu `<menuindex>` or in the first menu, if only one argument is given.

`getNum [<menuindex>]`

Returns the number of items in menu `<menuindex>` or in the first menu, if only one argument is given.

`setLabel [<menuindex>] [<itemindex>] <newlabel>`

If only one argument is given, this command sets the port's label. If two arguments are given, this command sets the label of item `<itemindex>` in the first menu. If three arguments are given, this command set the label of item `<itemindex>` in menu `<menuindex>`.

`getLabel [<menuindex>] [<itemindex>]`

If no argument is given, this command returns the port's label. If one argument is given, this command returns the label of item `<itemindex>` in the first menu. If two arguments are given, this command returns the label of item `<itemindex>` in menu `<menuindex>`.

`setMenuSensitivity [<menuindex>] {0|1}`

Enables or disables menu `<menuindex>` or the first menu, if `<menuindex>` is missing. If a menu is disabled it is grayed out and it does not accept any user input.

`getMenuSensitivity [<menuindex>]`

Checks whether the specified menu is enabled or not.

`setSensitivity [<menuindex>] <itemindex> {0|1}`

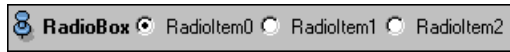
Enables or disables item <itemindex> in menu <menuindex> or in the first menu, if <menuindex> is missing. If an item is disabled it is no longer listed in the menu.

`getSensitivity [<menuindex>] <itemindex>`

Checks whether the specified item in menu <menuindex> is enabled or not.

## 7.21 PortRadioBox

This port provides multiple toggle buttons with a radio behavior, i.e., only one button can be switched on at a time. Thus the port facilitates a one-of-many choice. For a larger number of possible choices usually a combo box as provided by *PortMultiMenu* is more suitable.



Modules using this port are for example *OrthoSlice* or *SplineProbe*.

### Commands

Inherits all commands of *Port*.

`getValue`

Returns the index of the button currently being selected.

`setValue <index>`

Selects button <index> and deselects the previously selected button.

`setLabel [<index>] <label>`

If only one argument is given, this command sets the port's label. If two arguments are given, this command sets the label of radio button <index>.

`getLabel [<index>]`

If no arguments are given, this command returns the port's label. If one argument is given, this command returns the label of radio button <index>.

`setSensitivity <index> {0|1}`

Enables or disables radio button <index>. A disabled button is grayed out and cannot be selected interactively.

`getSensitivity <index>`

Checks whether radio button <index> is enabled or not.

`getNum`

Returns the total number of radio buttons of the port.

## 7.22 PortSeparator

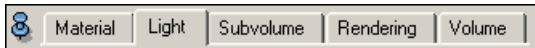
This port displays a horizontal line. It is used to visually separate different groups of ports from each other. In contrast to most other ports this port does not provide a pin button and thus cannot be pinned. It does not accept any user input.

### Commands

Inherits all commands of *Port*.

## 7.23 PortTabBar

The QTabBar class provides a bar consisting of a list of labeled, selectable tab elements, e.g., for use in tabbed dialogs. Only one tab can be switched at a time. Thus the port facilitates a one-of-many choice. This port has all methods of a *PortRadioBox* with the additional ability to assign a list of port objects to every single tab element. If a tab element is chosen, all ports included in the tabs port list and which are also marked as visible, are scheduled for drawing. If a tab gets deselected, the ports included in its port list become hidden. It's possible to add the same port to multiple tab elements and even to multiple QTabBar objects but it gets only visible if all its parent tab elements are selected, which is never the case for two tabs at the same QTabBar.



Modules using this port are for example *VolPro*.

### Commands

Inherits all commands of *PortRadioBox*.

`portAdd <index> <objectLabel> <portLabel>`

Adds the given port <object><port> to the tab objects <index> port list.

`portRemove <index> <objectLabel> <portLabel>`

Removes the given port <object><port> from the tab objects <index> port list.

## 7.24 PortText

This port provides a simple text input field.



The port is used for example in the *Arithmetic* module.

## Commands

Inherits all commands of *Port*.

**getValue**

Returns the contents of the text field.

**setValue** <string>

Sets the text in the text field to <string>.

**getNumColumns**

Returns the width of the text field in characters.

**setNumColumns** <numcolumns>

Sets the width of the text field so that <numcolumns> characters fit in.

**getSensitivity**

Checks if the text field is enabled or not.

**setSensitivity** {0|1}

Enables or disables the text field. If the text field is disabled, it is grayed out and it does not accept any user input.

## 7.25 PortTime

This port defines a floating point value like *PortFloatSlider*. However, the value is typically interpreted as a time value. It can be automatically animated in forward or backward direction, or it can be synchronized with any other object providing a time port, for example a *global time object*. In order to facilitate this kind of synchronization, *PortTime* is derived from *Connection*. If the port is connected to some other object providing a time port, the value of that port is used, i.e., the time port becomes an alias of the connected time port. The range of the time port as well as its increment (which is relevant for animation) can be edited in a little dialog which pops up when pressing the right mouse button over the slider.

In addition to the main button the time slider also provides so-called subrange buttons. These buttons allow you to temporarily restrict the range of the slider. For animations only this restricted range is considered. The subrange buttons can be set to an exact value via the port's popup dialog or by typing L <value> or R <value> in the sliders text field. The letters L or R indicate that the left or right subrange button should be set, and not the main button.



## Commands

Inherits all commands of *Connection*.

### `getValue`

Returns the current time value.

### `setValue <value>`

Sets the current time value.

### `getMinMax`

Returns two numbers indicating the full range of the time port.

### `setMinMax <min> <max>`

Sets the range of the time port.

### `getSubMinMax`

Returns two numbers indicating the subrange of the time port. The subrange is relevant for animations.

### `setSubMinMax <min> <max>`

Sets the subrange of the time port.

### `getIncrement`

Returns the increment of the time port. During animations or when pressing the forward or backward buttons the time value is increased or decreased by the increment.

### `setIncrement <value>`

Sets the increment of the time port.

### `getDiscrete`

Checks if the time port is in discrete mode or not. In discrete mode the time can only take on values  $min + n * increment$ , where *min* is the minimum value of the range, *increment* is the port's increment value, and *n* is an integer value.

### `setDiscrete {0|1}`

Sets discrete mode on or off.

### `setFormat <format>`

Sets the format used for the slider's text field in printf syntax. Typically, the default format is %g.

### `setTracking {0|1}`

Turns tracking on or off. If tracking is enabled the object the port belongs to is fired permanently while the moving the slider with the mouse. If tracking is off the object is only fired once when the mouse button is released.

### `play [-forward|-backward] [-once|-loop|-swing]`

Starts to animate the time value. Additional arguments can be specified for the animation direction (forward or backward) and for the animation mode (play once, loop forever, play forth and back forever). If no arguments are specified forward play and the last animation mode are assumed.

**stop**

Stops a running animation.

**setRealTimeDuration** <duration\_in\_seconds>

This command enables real time mode. In real time mode a full animation cycle takes as long as specified by this command. The increment added to the current time value after each animation step depends on the time required for that step. When real time mode is enabled discrete mode will be automatically disabled.

**getRealTimeDuration**

Returns the time duration of a full animation cycle in real time mode. If real time mode is disabled, 0 is returned. When the min or max time value is changed while real time mode is enabled the time duration is automatically adjusted.

## 7.26 PortToggleList

This port provides multiple toggle buttons. The buttons are implicitly numbered 0, 1, 2, ... from left to right and each button can be switched independently. Usually each toggle represents a particular option affecting the operation of a module.



The port is used for example in the *LabelVoxel* module.

### Commands

Inherits all commands of *Port*.

**getValue** <index>

Returns 1 if toggle <index> is checked or 0 if it is not.

**setValue** <index> {0|1}

Sets the state of the button <index>.

**getNum**

Returns the number of toggle buttons of the port.

**setNum** <numbuttons>

Sets the number of toggle buttons of the port.

**getLabel** [<index>]

If no arguments are given, this command returns the port's label. If one argument is given, this command returns the label of toggle button <index>.



`setLabel [<index>] <label>`

If only one argument is given, this command sets the port's label. If two arguments are given, this command sets the label of toggle button <index>.

`getLabel [<index>]`

If no arguments are given, this command returns the port's label. If one argument is given, this command returns the label of toggle button <index>.

`setLabel [<index>] <label>`

If only one argument is given, this command sets the port's label. If two arguments are given, this command sets the label of toggle button <index>.

`getSensitivity <index>`

Checks whether toggle button <index> is enabled or not.

`setSensitivity <index> {0|1}`

Enables or disables toggle button <index>. A disabled button is grayed out and cannot be modified interactively.

## **Part III**

# **Alphabetic Index of File Formats**



# Chapter 8

## Amira

### 8.1 Amira Script

amira scripts are written in the Tcl command language. Scripts allow you to start demos, to perform routine tasks, or to create animations. Networks are saved as script files too. When such a file is executed the network is restored. A detailed description of the amira scripting facilities is contained in the user's guide in Chapter ?? (Scripting). A script file is recognized either by the special comment `# Amira Script` in the first line of the file, or by the file extension `.hx`.

### 8.2 Amira Script Object

amira script objects are custom modules with user-defined GUI elements (ports) written in the Tcl command language. Files describing script objects must obey certain requirements as stated in the data types section about *ScriptObjects*. When a file describing a script object is loaded, an instance of a *ScriptObject* module is created and initialized as requested. Files describing script objects are recognized by the special comment `# Amira Script Object V3.0` in the first line of the file, or by the file extension `.scro`. Note that in previous versions of amira a different syntax was used for script objects. In order to make clear that a new-style script object is defined, the comment mentioned above should be used.

### 8.3 AmiraMesh Format

*AmiraMesh* is amira's native general-purpose file format. It is used to store many different data objects like fields defined on regular or tetrahedral grids, segmentation results, colormaps, or vertex sets such as landmarks. The format itself is very flexible. In fact, it can be used to save arbitrary multi-dimensional arrays into a file. In order to create an amira data object from an AmiraMesh file the

contents of the file are analyzed and interpreted. For example, a tetrahedral grid is expected to have a one-dimensional array *Nodes* containing entries of type `float[3]` called *Coordinates*, as well as a one-dimensional array *Tetrahedra* containing entries of type `int[4]` called *Nodes*. If the AmiraMesh file contains an entry *ContentType* in its parameter section, the value of this parameter directly determines what kind of amira data object is to be created.

**A first example.** In order to describe the syntax of an AmiraMesh file, we first give a short example. This example describes a scalar field defined on a tetrahedral grid. *Concrete examples of how to encode other data objects are given below.*

```
# AmiraMesh ASCII 1.0

define Nodes 4
define Tetrahedra 1

Parameters {
    Info "This is an AmiraMesh example",
    Pi 3.1459
}

Materials { {
    Name "Stone",
    Color 0.8 0.3 0.1
} {
    Name "Water",
    Color 0 0.3 0.8
} }

Nodes { float[3] Coordinates } = @1
Tetrahedra { int[4] Nodes } = @4

Nodes { float Values } = @8
Tetrahedra { byte Materials } = @12

Field { float Example } = Linear(@8)

@1
0 0 0
1 0 0
0 1 0
0 0 1

@4
```

```

1 2 3 4

@8
0 0 0 1

@12
1

```

The first line of an AmiraMesh file should be a special comment including the identifier **AmiraMesh**. Moreover, if the tag **ASCII** is given in this line, all data arrays are stored in plain ASCII text. If the tag **BINARY** is given, the data arrays are stored in IEEE big-endian binary format. Note that the header section of an AmiraMesh file is always given as ASCII text.

The statement **define Nodes 4** defines a one-dimensional array of size 4. Later on, this array can be referenced using the name **Nodes**. Similarly, a statement **define Array 100 100** defines a two-dimensional array of size 100 x 100. The actual kind of data stored per array element will be specified later on.

The optional section **Parameters** allows the user to define arbitrary additional parameters. Each parameter consists of a name (like **Pi**) and a value (like **3.1459**). Values may be one or multiple integer or floating point numbers or a string. Strings have to be quoted using a pair of **"**-characters.

The optional section **Materials** allows the user to define additional material information. This is useful for finite element applications. The material section consists of a comma-separated list of parameters just as in the **Parameters** section.

The statement **Nodes { float[3] Coordinates } = @1** specifies that for each element of the array **Nodes** defined earlier, three floating point numbers (floats) should be stored. These data are given the name **Coordinates** and a tag in this line and will appear below as a tagged block with the marker **@1**. Such data markers must always begin with the **@** character.

Similar, the following lines define additional data to be stored in the arrays called **Nodes** and **Tetrahedra**. The primitive data types must be one of **byte**, **short**, **int**, **float**, **double**, or **complex**. Vectors of primitive data types are allowed, aggregate structs are not, however.

The statement **Field { float Example } = Linear(@8)** defines a continuous scalar field with the name **Example**. This field will be generated by linear interpolation from the data values **Values** defined on the nodes of the tetrahedral grid. Other interpolation methods include **Constant(@X)** and **EdgeElem(@X)**.

After the marker **@1**, the coordinate values of the grid are stored. Likewise, the other data arrays are given after their corresponding markers. In case of a **BINARY** file the line containing the marker is read up to the next new line character. Then the specified number of bytes is read in binary format. It is assumed that **sizeof(short)** is 2, **sizeof(int)** is 4, **sizeof(float)** is 4, **sizeof(double)** is 8, and **sizeof(complex)** is 8. Multidimensional arrays indexed via **[k][j][i]** are read with **i** running fastest.

**Backward compatibility.** For backward compatibility the following statements are considered to be equal:

```
nNodes 99 is equal to define Nodes 99
nTriangles 99 is equal to define Triangles 99
nTetrahedra 99 is equal to define Tetrahedra 99
nEdges 99 is equal to define Edges 99
```

```
NodeData is equal to Nodes
TriangleData is equal to Triangles
TetrahedraData is equal to Tetrahedra
EdgeData is equal to Edges
```

**Other data objects.** Of course, not only scalar fields defined on tetrahedral grids can be encoded using the AmiraMesh format. Many other data objects are supported as well. In each case there are certain rules about what data arrays have to be written and how these arrays have to be named. Below, we describe how to encode the following data objects:

- *Fields with uniform coordinates*
- *Fields with stacked coordinates*
- *Fields with rectilinear coordinates*
- *Fields with curvilinear coordinates*
- *Label fields for segmentation*
- *Landmarks for registration*
- *Line segments*
- *Colormaps*

### 8.3.1 Fields with uniform coordinates

In order to encode 3D scalar or vector fields defined on a uniform grid you first have to define a 3D AmiraMesh array called *Lattice*. The field's data values are stored on this array. The coordinate type of the field as well as the bounding box are specified in the parameter section of the AmiraMesh file. This is illustrated in the following example:

```
# AmiraMesh ASCII 1.0

# Dimensions in x-, y-, and z-direction
define Lattice 2 2 2

Parameters {
    CoordType "uniform",
```

```

        # BoundingBox is xmin xmax ymin ymax zmin zmax
        BoundingBox 0 1 0 1 0 1
    }

    Lattice { float ScalarField } = @1

    @1
    0 0 1 1 0 0 2 2

```

Use `float[3]` in order to encode a vector field instead of a scalar field. Likewise, you may modify the field's primitive data type. For example, 3D images are commonly encoded using `byte` or `short`. Instead of `ScalarField` you may use any other name in the data definition statement.

The field's bounding box is given by the minimum and maximum x-, y-, and z-coordinates of the *grid nodes* or voxel centers, not of the voxel boundaries. *amira* will always assume the width of a single voxel to be  $(x_{max}-x_{min})/(dims[0]-1)$ . For degenerated 3D data sets with one dimension being 1 choose equal minimum and maximum coordinates in that direction.

### 8.3.2 Fields with stacked coordinates

A field with stacked coordinates has uniform pixel spacing in the x and y direction, but slices may be arranged arbitrarily in the z direction. This type of coordinates is commonly used to encode 3D medical images with non-uniform spacing.

In order to encode a 3D scalar or vector field with stacked coordinates, you must define a 3D array called *Lattice* and 1D array called *Coordinates*. The field's data values are stored at *Lattice* while the slices' z positions are stored at *Coordinates*. The coordinate type of the field as well as the bounding box in xy are specified in the parameter section of the *AmiraMesh* file. Here is an example:

```

# AmiraMesh ASCII 1.0

define Lattice 2 2 3
define Coordinates 3

Parameters {
    CoordType "stacked",
    # BoundingBoxXY is xmin xmax ymin ymax
    BoundingBoxXY 0 1 0 1
}

Lattice { byte Intensity } = @1
Coordinates { float z } = @2

```



```

@1
0 0 0 0
1 1 1 1
2 2 2 2

@2
0 0.75 2

```

Use `float[3]` in order to encode a vector field instead of a scalar field. Likewise, you may modify the field's primitive data type. For example, 3D images are commonly encoded using `byte` or `short`. Instead of `Intensity` you may use any other name in the data definition statement.

The field's xy bounding box is given by the minimum and maximum x and y coordinates of the *grid nodes* or pixel centers, not of the pixel boundaries. `amira` will always assume the width of a single pixel to be  $(x_{max}-x_{min})/(dims[0]-1)$ . For degenerate 3D data sets with one dimension being 1, choose equal minimum and maximum coordinates in that direction.

### 8.3.3 Fields with rectilinear coordinates

A field with rectilinear coordinates still has axis-aligned grid cells. However, the x-, y-, and z-coordinates of the grid nodes are specified explicitly for each direction.

In order to encode a 3D scalar or vector field with rectilinear coordinates, you have to define a 3D array called *Lattice* and 1D array called *Coordinates*. The field's data values are stored at *Lattice* while the x, y, and z positions for each direction are stored at *Coordinates* in subsequent order. The size of the *Coordinates* array must be equal to the sum of the sizes of *Lattice* in the x, y, and z directions. The coordinate type of the field is specified in the parameter section of the AmiraMesh file. Here is an example:

```

# AmiraMesh ASCII 1.0

define Lattice 2 2 3
define Coordinates 7 # This is 2+2+3

Parameters {
    CoordType "rectilinear"
}

Lattice { float ScalarField } = @1
Coordinates { float xyz } = @2

```

```

@1
0 0 0 0
1 1 1 1
2 2 2 2

@2
0 1      # x coordinates
0 1.5    # y coordinates
-1 1 2   # z coordinates

```

Use `float[3]` in order to encode a vector field instead of a scalar field. Likewise, you may modify the field's primitive data type. Instead of `ScalarField` you may use any other name in the data definition statement.

### 8.3.4 Fields with curvilinear coordinates

A field with curvilinear coordinates consists of a regular array of grid cells. Each grid node can be addressed by an index triple (i,j,k). The coordinates of the grid nodes are specified explicitly.

In order to encode a 3D scalar or vector field with curvilinear coordinates, you have to define a 3D array called *Lattice*. This array is used to store the field's data values as well as the grid nodes' coordinates. The coordinates must be given as a `float[3]` vector containing x-, y-, and z-values. The coordinate type of the field is specified in the parameter section of the AmiraMesh file. Here is an example:

```

# AmiraMesh ASCII 1.0

define Lattice 2 2 3

Parameters {
    CoordType "curvilinear"
}

Lattice { float ScalarField } = @1
Lattice { float[3] Coordinates } = @2

@1 # 2x2x3 scalar values
0 0 0 0
1 1 1 1
2 2 2 2

@2 # 2x2x3 xyz coordinates
0 0 0

```

```

1 0 0
0 1 0
1 0 0
0 0 1
1 0 1
0 1 1
1 0 1
0 0 2
1 0 2
0 1 2
1 0 2

```

Use `float[3]` in order to encode a vector field instead of a scalar field. Likewise, you may modify the field's primitive data type. Instead of `ScalarField` you may use any other name in the data definition statement.

### 8.3.5 Label fields for segmentation

Label fields are closely related to ordinary scalar fields with uniform coordinates. However, the data values at each voxel are interpreted as labels denoting the different materials or regions the voxels have been assigned to during a segmentation process. Therefore, the most important difference of label fields compared to *uniform scalar fields* is the occurrence of a *Materials* section in the AmiraMesh file. Whenever such a section occurs and elements of type *byte* denoted *Labels* are found, the AmiraMesh file is interpreted as a *label field*. Here is a simple example of a label field containing two different materials:

```

# AmiraMesh ASCII 1.0

define Lattice 2 2 2

Parameters {
  CoordType "uniform",
  # BoundingBox is xmin xmax ymin ymax zmin zmax
  BoundingBox 0 1 0 1 0 1
}

Materials {
  { Id 0, Name "Exterior" }
  { Id 4, Name "Something" }
}

```

```

Lattice { byte Labels } = @1
Lattice { byte Probability } = @2

@1
0 0 0 4
0 0 4 4

@2
255 255 130 180
255 200 190 230

```

Each material should have a parameter *Id* specifying the correspondence between labels and materials. In the example above all voxels labeled with 0 belong to material *Exterior*, while all voxels labeled with 4 belong to material *Something*.

Optionally, label fields may contain probability information or weights as shown in the example above. These weights denote the degree of confidence of the labeling. This information is used by the *GMC* module when extracting boundary surfaces.

### 8.3.6 Landmarks for registration

The data type *Landmark Set* is useful for registration and alignment of multiple 3D image data sets. It allows you to store multiple sets of corresponding marker positions. The data type can also be used to represent a simple list of 3D points in *amira*. In this case you would only specify a single set of markers. Consider the following example:

```

# AmiraMesh ASCII 1.0

define Markers 3

Parameters {
    ContentType "LandmarkSet",
    NumSets 2
}

Markers { float[3] Coordinates } = @1
Markers { float[3] Coordinates2 } = @2

@1
38.5363 15.2135 20.3196
35.1264 14.0106 37.155
31.6494 14.2791 31.0932

```

```

@2
40.2112 15.907 20.3119
35.9551 13.8241 40.4785
30.1375 13.7279 28.9235

```

In this example, first the number of markers or points is defined to be 3. In the parameter section of the AmiraMesh file the content type is specified, as well as the number of marker sets. The marker coordinates of the first set are denoted `Coordinates` (xyz-values stored as `float[3]`). Likewise, the marker coordinates of the second set are denoted `Coordinates2`. If more sets are defined, the coordinate values must be called `Coordinates3`, `Coordinates4`, and so on.

It is also possible to define additional data values for each marker such as `MarkerTypes` or `Orientations`. How these values are interpreted in detail will be specified in a future release of amira.

### 8.3.7 Line segments

The data type *Line Set* is used to represent a generic set of indexed line segments, i.e., line segments defined by an index into a vertex list. Optionally, an arbitrary number of scalar data values may be associated with each vertex.

In order to store line sets in an AmiraMesh file, two 1D arrays have to be defined, namely *Lines*, used to store the indices, and *Vertices*, used to store the vertex coordinates as well as additional vertex data. Here is an example:

```

# AmiraMesh ASCII 1.0

define Lines 15
define Vertices 12

Parameters {
    ContentType "HxLineSet"
}

Vertices { float[3] Coordinates } = @1
Vertices { float Data } = @2
Lines { int LineIdx } = @3

@1 # 12 xyz coordinates
0.9 0 0
1 0 0.1

```

```

1 0 1.9
0.9 0 2
0 0.9 0
0 1 0.1
0 1 1.9
0 0.9 2
-0.9 0 0
-1 0 0.1
-1 0 1.9
-0.9 0 2

@2 # 12 data values
1 1 1 1 2 2 2 2 1 1 1 1

@3 # 15 indices, defining 3 line segments
0 1 2 3 -1
4 5 6 7 -1
8 9 10 11 -1

```

Lines are defined using vertex indices as shown above. The index of the first vertex is 0. An index value of -1 indicates that a line segment should be terminated. An arbitrary number of additional vertex data values can be defined. Multiple values should be distinguished by denoting them `Data2`, `Data3`, and so on.

### 8.3.8 Colormaps

An *amira colormap* consists of a one-dimensional array of RGBA components accompanied by two numbers *min max* specifying the data window that should be linearly mapped to the RGBA values. The RGBA array should have 256 elements in order to allow the colormap to be edited using the *colormap editor*.

Colormaps are encoded in an AmiraMesh file as follows:

```

# AmiraMesh ASCII 1.0

define Lattice 256

Parameters {
    ContentType "Colormap",
    MinMax 10 180
}

```

```

Lattice { float[4] Data } = @1

@1
1 0 0 0
1 0.00392157 0 0
1 0.00392157 0 0.00392157
1 0.0117647 0 0.00392157
1 0.0196078 0 0.0196078
1 0.027451 0.00392157 0.0196078
...

```

The RGBA values are stored in floating point format. A component value of 0 means no intensity (black), while a component value of 1 means maximum intensity (white). The fourth component denotes opacity (*alpha*). Here a value of 0 indicates that the color is completely transparent while a value of 1 indicates that the color is completely opaque.

## 8.4 AmiraMesh as LargeDiskData

Image data stored in an uncompressed *AmiraMesh* file can be loaded as *LargeDiskData*. Use the *File Dialog's* Popup Menu to force the file format to "AmiraMesh as LargeDiskData".

The file is opened readonly. You cannot change the image data, nor add or modify parameters.

## 8.5 Analyze 7.5

This format was used by older versions of the Analyze medical imaging software system. Today it has been widely replaced by the *AnalyzeAVW* format. The format is used to store 3D medical images. The actual image data and the header information are stored in two different files. In order to import the data in *amira*, you must select just the header file in the *amira* file browser. The header file is recognized by the extension *.hdr*. If it has some other extension, you must manually select the file format via the file browser's popup menu.

The header file contains the dimensions of the 3D image, the voxel size, and the primitive data type (8/16/32-bit grayscale, 24-bit color). In addition, some other information such as a short data description or a patient id are contained in the header. This information will be stored in the parameter section of the generated *amira* data object.

Note that the input data type may be interpreted as unsigned short if the key for the maximum value (*glmax*) in the header of the Analyze file contains a value of larger than 32,768.

## 8.6 AnalyzeAVW

This format is used by newer versions of the Analyze medical imaging software (version 2.5 or higher). Previous versions used the *Analyze 7.5* format, where image data and header information were stored in different files. The *Analyze AVW* format can be used to store 2D, 3D, and 4D medical images, but the 4D time series option is currently not supported in *amira*. Additional attributes stored in the files (like patient name or examination time) are stored in the parameter section of the generated *amira* data objects. *Analyze AVW* files are safely recognized by inspecting the file header. The common file name extension is *.avw*.

Besides *Analyze AVW* image files, sometimes also so-called *Analyze AVW* volume files are used, which contain a list of 2D file names forming a 3D image stack. Such volume files currently cannot be interpreted by *amira*. However, note that *amira* provides a very similar concept with the *Stacked-Slices* format.

*amira* exports scalar and color fields with uniform coordinates to the *Analyze AVW* format. Other coordinate types can be written as well, but in these cases the coordinate information will be lost and no accurate recovery will be possible.

## 8.7 BMP Image Format

BMP is a standard image format mainly used on the Microsoft Windows platform. Image data is stored with 8 or 24 bits per pixel without applying any compression. When writing RGBA color fields the alpha channel will be discarded. When reading BMP images an alpha value of 255 (full opacity) will be assumed. BMP files are automatically identified by the file name extensions *.bmp*.

Regarding the import and export of multiple slices the same remarks apply as for the *TIFF image format*. When reading BMP images the *channel conversion dialog* is popped up. This dialog is also described in the TIFF section.

## 8.8 DXF

The *DXF* file format is a general-purpose format used by the AutoCAD (TM) software. The format is able to store a large variety of 2D and 3D geometries. Currently, *amira* only exports files in plain ASCII format containing 3D triangular surfaces and 3D line sets. The information defined in the parameter section of an *amira* data object will only be saved in case of 3D triangular surfaces.

3D line sets can be exported in two different ways: the whole line set in a single file or on a per-slice basis (*DXF slice-by-slice*). The latter format can only be selected if the line set contains a special parameter describing the structure of the individual slices. Groups of lines belonging to the same slice, i.e., having the same x, y, or z coordinate, are saved in the same *DXF* file. Line sets of this special type are produced by the module *ComputeContours*.



When importing *DXF* files *amira* will create an Open Inventor scene graph object. You may use the module *IvToSurface* in order to convert the scene graph into an *amira* surface. Currently, only *DXF* files in ASCII format can be read.

*DXF* files are identified by the file name extension `.dxf`.

## 8.9 Encapsulated PostScript

*amira* is able to save snapshots as well as individual slices of a 3D image data set in *Encapsulated PostScript* format. You may directly send EPS files to a PostScript printer, or you may include these files in many standard desktop publishing programs. The EPS files produced by *amira* contain bitmaps rather than vector information.

The import of EPS files is not supported.

## 8.10 HTML

HTML files loaded via the *amira* file dialog are displayed in the online help viewer. The help viewer supports certain but not all HTML formatting tags. It also is not able to resolve web links. Only links to local files are resolved. Nevertheless, it is very convenient to describe projects and to invoke demo scripts from an HTML page displayed in the *amira* help viewer. Whenever a file with the extension `.hx` is linked, this file is interpreted as an *amira* script. The script is executed when the link is clicked. As an example you may look at the demo pages provided with the online user's guide, for example `share/usersguide/recon.html` or other files listed in `share/usersguide/AmiraIndexDemo.html`.

## 8.11 HxSurface

**Simplified version.** The surface format has been designed to represent *triangular non-manifold surfaces*. The triangles of such a surface are grouped in patches. In addition, information about so-called boundary contours and branching points can be stored in a surface file. However, since this information can be recomputed automatically if required, we discuss a simplified version of the format first. Here is an example:

```
# HyperSurface ASCII

Parameters {
    Info "GMC: 3 colors, case 13"
}
```

```

Materials { {
    color 0.83562 0.78 0.06,
    Name "Yellow" }
{
    color 0.21622 0.8 0.16,
    name "Green" }
{
    color 0.8 0.16 0.596115,
    name "Magenta" }
}

Vertices 11
1.000000 0.666667 0.500000
0.666667 0.500000 1.000000
1.000000 0.500000 0.000000
0.500000 1.000000 0.000000
0.000000 0.000000 0.500000
0.000000 1.000000 0.500000
1.000000 1.000000 0.500000
0.500000 0.000000 1.000000
1.000000 0.500000 1.000000
0.500000 1.000000 1.000000
0.523810 0.523809 0.500000

Patches 3
{ InnerRegion Green
  OuterRegion Yellow
  Triangles 7
    3 1 11
    4 3 11
    6 4 11
    5 6 11
    8 5 11
    2 8 11
    1 2 11
}
{ InnerRegion Magenta
  OuterRegion Yellow
  Triangles 1
    9 2 1
}
{ InnerRegion Magenta

```

```

OuterRegion Green
Triangles 2
  2 10 7
  7 1 2
}

```

The first line is required and identifies the surface format. Additional comments starting with a hash-mark (#) may appear at any point in the file. Next, an optional parameter section and a material section follow. These sections have the same format as in an *AmiraMesh* file. The parameter section may contain an arbitrary number of name-value items. The material section contains additional information about imaginary regions the surface patches are supposed to separate. In contrast to an *AmiraMesh* file, individual materials need not to have an *Id* since they are referenced via their names.

The statement `Vertices 11` indicates that the x, y, and z coordinates of 11 vertices follow. Likewise, the statement `Patches 3` indicates that 3 patches follow. The definition of each patch is enclosed by a pair of curly braces { and }. Inside these braces `InnerRegion` and `OuterRegion` indicate the two regions the patch is supposed to separate. If you don't want to generate a tetrahedral grid from your surface, you may omit these statements, or you may choose both regions to be the same. Finally, the triangles of a patch are specified by indexing vertices defined in the vertex section. Like in an *AmiraMesh* file, indices start at 1, not at 0.

**Extended version.** In its extended version the surface format is able to store additional topological information of a surface. Before we discuss this in detail, let us first provide some definitions and introduce the underlying concepts.

- **Region:** In finite element applications regions are usually called materials. A region is defined by its surrounding surface, which may consist of multiple patches. Each region must have a unique name.
- **Surface:** A surface is defined by the boundary of a 3D region and therefore must be closed. This means that, for example, each edge must be connected to an even number of triangles. In one half of the triangles the edge is referenced in forward orientation, in the other half in backward orientation. A surface may consist of multiple pieces, so-called patches. In the file format the patches of a surface are given by a list of signed indices. A negative index means that the patch has negative orientation in the current surface.
- **Patch:** A part of a surface which separates exactly two different regions. Patches are built from triangles. The triangles are all oriented in a unique way so that we can define an inner region and an outer region. The triangle normals point into the outer region. For each patch, an `InnerRegion` and an `OuterRegion` may be specified in the file format. If one of these specifiers is missing, it is assumed that the patch delimits the exterior space. Otherwise the exterior region should be called `OUTSIDE`. Patches may be closed or may be delimited by so-called boundary curves. Boundary curves are specified by a list of signed integers. The sign denotes the orientation of the curve segment for a particular patch. In addition, for each patch

inner branching points may be specified if necessary.

- **BoundaryCurve:** At a boundary curve multiple patches join. The curves are defined by a set of vertex indices. For closed curves the first and the last index must be equal.
- **BranchingPoint:** A point where multiple regions join. The start and end points of a boundary curve are branching points. In addition, there may be also branching points inside a patch which are not part of a boundary curve.
- **Vertices:** These are the points from which the surface triangles are built. In the file format the vertex coordinates are specified after the identifier `V e r t i c e s` followed by the number of vertices. First, branching points should be specified, then points on boundary curves, and then inner points of the patches. In the definition of boundary curves and patches, the vertices are referenced by indices starting from 1, not from 0.

The following example describes the surfaces of three connected tetrahedra which are assigned to three different regions. Some of the definitions are optional. Really necessary are only the list of vertex coordinates as well as the definition of the patches. Boundary curves and surfaces may be reconstructed from this.

```
# HyperSurface ASCII
```

```
Vertices 6
    0.0  0.0  0.0
    0.0  0.0  1.0
    0.0  1.0  0.0
   -1.0  0.0  0.0
    1.0  0.0  0.0
    0.0 -1.0  0.0
```

```
NBranchingPoints 2
NVerticesOnCurves 2
```

```
BoundaryCurves 3
{
    Vertices 3
        1  3  2
} {
    Vertices 2
        1  2
} {
    Vertices 3
        1  4  2
}
```

Patches 5

```
{
    InnerRegion Material1
    OuterRegion OUTSIDE
    BranchingPoints 0
    BoundaryCurves 2
        1 -2
    Triangles 3
        5 1 3
        1 5 2
        5 3 2
} {
    InnerRegion Material2
    OuterRegion OUTSIDE
    BranchingPoints 0
    BoundaryCurves 2
        3 -1
    Triangles 2
        3 1 4
        2 3 4
} {
    InnerRegion Material3
    OuterRegion OUTSIDE
    BranchingPoints 0
    BoundaryCurves 2
        -3 2
    Triangles 3
        4 1 6
        2 4 6
        1 2 6
} {
    InnerRegion Material1
    OuterRegion Material2
    BranchingPoints 0
    BoundaryCurves 2
        2 -1
    Triangles 1
        3 1 2
} {
    InnerRegion Material2
    OuterRegion Material3
    BranchingPoints 0
```

```

        BoundaryCurves 2
            -3 2
        Triangles 1
            1 2 4
    }

    Surfaces 3
    {
        Region Material1
        Patches 2
            1 4
    } {
        Region Material2
        Patches 3
            2 -4 5
    } {
        Region Material3
        Patches 2
            3 -5
    }

```

## 8.12 Icol

This is a simple ASCII file format coming in two variants, indexed and non-indexed, to store colormaps. The *Icol* file format and the *Icol Colormap Editor* originate from the Graphics and Visualization Lab (GVL) of the Army High Performance Computing Research Center (AHPCRC), Minnesota. The *Colormap Editor* in *amira* shares many ideas with the original AHPCRC tool. The structure of an *Icol* format can be immediately understood by looking at the examples in *amira*'s demo data directory.

## 8.13 Interfile

Interfile is a file format for the exchange of nuclear medicine image data.

With this reader you can import files in the Interfile exchange format into *amira*. Interfile data objects consist of two files, a .hdr file containing the header information and an .img file which contains the raw image data. Both files should be in the same directory.

### Limitations:

- The .hdr file must start with the sequence '!INTERFILE' to be recognized by the reader.

## 8.14 JPEG Image Format

JPEG is a standard format which can be used to store RGB and grayscale images in a highly compressed form. However, note that the compression algorithm is lossy. The quality of the compression for file export can be specified by typing the command

```
set AmiraJPEGQuality <quality>
```

into the *amira* console prior to the export, where <quality> ranges from 0 to 100. The default is 90. When writing RGBA color fields the alpha channel will be discarded. When reading RGB images an alpha value of 255 (full opacity) will be assumed. JPEG files are automatically identified by the file name extensions `.jpg` or `.jpeg`.

Regarding the import and export of multiple slices the same remarks apply as for the *TIFF image format*. When reading JPEG images the *channel conversion dialog* is popped up. This dialog is also described in the TIFF section.

## 8.15 MATLAB Binary Format (.mat)

Saves an *amira* data object as a MATLAB *MAT* file. Only the raw data will be saved but information like the size of the bounding box will be lost. MATLAB arrays are generated that may be 4-dimensional. For example an RGBA colorfield will be saved as a 4-dimensional data set where the first three dimensions represent spatial coordinates and the fourth dimension contains the RGBA values. The data variables will be named like the *amira* variable name with an `AMIRA_` in front of them and all `'.'` characters replaced by `'_'` characters (to avoid conflicts with structs in MATLAB).

*MAT* files are identified by the file name extension `.mat`.

## 8.16 MATLAB M-files Format (.m)

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. You create M-files using a text editor. You may use the module *CalculusMatlab* in order to load, save, and execute M-files.

## 8.17 Open Inventor

The *Open Inventor* file format is used for storing 3D geometries. The format is very powerful. The VRML format known from the World Wide Web is very similar to Open Inventor. Since *amira* is built on top of Open Inventor, it naturally supports this format. However, *amira* has added a lot of special purpose nodes to Open Inventor. Therefore currently some geometries which can be displayed in *amira*'s 3D viewer cannot be saved in Open Inventor file format.

When reading an Open Inventor file a data object of type *IvData* will be created. This data object stores the Open Inventor scene graph and can be visualized using a *IvDisplay* module. *IvData* objects can be saved again in ASCII or binary Open Inventor format. In addition, *amira surfaces* can be exported in Open Inventor format.

## 8.18 PNG Image Format

PNG stands for *Portable Network Graphics*. The format is mainly used for internet applications. Usually, image data is stored in compressed form using a lossless compression algorithm. The format is able to store an alpha channel besides the ordinary color channels. PNG files are automatically identified by the file name extensions `.png`.

Regarding the import and export of multiple slices the same remarks apply as for the *TIFF image format*. When reading PNG images the *channel conversion dialog* is popped up. This dialog is also described in the TIFF section.

## 8.19 PNM Image Format

This format includes the PPM, PGM, and PBM image formats. These formats are used to store RGB color images, grayscale images, as well as black and white images, respectively. For each of the three formats there is a binary and an ASCII version. *amira* is able to read all six of them, but will only write binary PPM and PGM files. Black and white PBM images can only be read if the image width is a multiple of eight. PNM files will be automatically identified by their file headers.

Regarding the import and export of multiple slices, the same remarks apply as for the *TIFF image format*. When reading PNM images the *channel conversion dialog* is popped up. This dialog is described in the TIFF section.

## 8.20 PSI format

The PSI format stores a set of 3D points with additional data variables associated to them in a column-oriented way. In *amira* PSI files are represented by data objects of type *HxCluster*.

A PSI file starts with an (optional) header section. This section describes the contents of the file, i.e., the number and the meaning of the individual data columns. The file syntax is illustrated in the following example:

```
# PSI Format 1.0
#
# column[0] = "x"
# column[1] = "y"
```



```

# column[2] = "z"
# column[3] = "Energy"
# column[4] = "Grain"
# column[5] = "Id"
# column[6] = "Coordination Number"
# column[7] = "Crystallographic Class"
#
# symbol[3] = "E"
# symbol[4] = "g"
# symbol[6] = "n"
# symbol[7] = "c"
#
# type[3] = float
# type[4] = byte
# type[6] = byte
# type[7] = { PFCC, GFCC, PHCP, GHCP, OT12, OTHR }

5 2694 115001
1.00 0.00 0.00
0.00 1.00 0.00
0.00 0.00 1.00

-0.748 -0.748 -0.777 -4.3840 15 327909 12 PFCC
-0.735 -0.739 -0.757 -4.3840 15 327910 12 PFCC
-0.742 -0.784 -0.754 -4.3400 15 328800 12 GFCC
-0.757 -0.769 -0.766 -4.3823 15 328812 12 PFCC
-0.747 -0.762 -0.745 -4.3638 15 328813 12 PFCC

```

The very first line indicates that this is a PSI file. The statement `column[0] = "x"` indicates that the first data column represents the x-coordinates of the points. In this example in total eight data columns are defined. Note that the names "x", "y", "z", and "Id" have a special meaning. These columns are required.

Next, the statement `symbol[3] = "E"` defines a symbol for the data column labeled "Energy". This symbol may be used in an arithmetic filter expression as provided for example by the modules *ClusterView* and *ClusterDiff*. Note that symbol definitions for the four special columns labeled "x", "y", "z", and "Id" have no effect.

Finally, the statement `type[3] = float` indicates that internally energy values should be stored as 32-bit floating point numbers. Alternatively, data values may be stored as 32-bit integers (`int`) or as 8-bit characters (`byte`). As a special feature, data values may also be represented by textual tokens. In this case, the set of allowed tokens should be specified in a comma-separated list as shown in the above example.

The header section may also contain any other user-defined comment, provided the first character of a comment line is a #. The first non-blank line after the header section specifies the number of points, as well as two other parameters, which are ignored by *amira*. Next, the bounding box of the data set is specified. However, *amira* will ignore this definition. Instead, the bounding box will be calculated from the point coordinates itself.

If a *PSI* file contains no header section at all, *amira* assumes that the file contains exactly eight data columns and that these data columns are arranged like in the example above.

## 8.21 Ply Format

The *Ply* format was developed at Stanford University Computer Graphics Lab. It is used for storing points and geometries. In *amira* it is possible to read and save surfaces in this format. When writing *amira* surfaces in this format, additional information such as the material section of the surface will be written into *Ply* files as well. However, other application will not be able to interpret this additional data.

*amira* identifies a *Ply* file by its header.

## 8.22 Raw Data

Sometimes you may want to read data defined on uniform lattices in raw format, i.e., plain three-dimensional arrays of data. Tomographic images might be given in this way, and raw data is often the easiest format to produce with, e.g., custom simulation programs.

To read in raw data, use the *Load/Load Data* menu, select the file in the file browser and click *OK*. Since *amira* cannot recognize the file format automatically, a dialog will popup. Within this dialog choose *Raw Data* as file format and click *OK*. Since the raw data file does not contain any information on how to format the data, some user specifications are required. *amira* will bring up a dialog:

Now adjust the parameters:

- **Data Types.** Primitive type of data: byte for 8-bit data, short for 16-bit data, int32 for 32-bit integer data, float for 32-bit float data, double for 64-bit floating point data.  
In addition, the number of data values per data point must be specified: 1 for scalar data, 3 for vector data, 6 for complex vector fields.
- **Dimensions.** Size of the three-dimensional array. If incorrect dimensions are specified, the data will be scrambled.
- **Min/Max Coords.** The bounding box of the data. These parameters are not as critical as the other ones. In particular the bounding box can be adjusted afterwards using the crop editor.
- **Header.** Many file formats consist of a raw data block with a prepended header. If such files are read with this method, the size of the header can be specified here. The header will then simply be skipped when reading the file

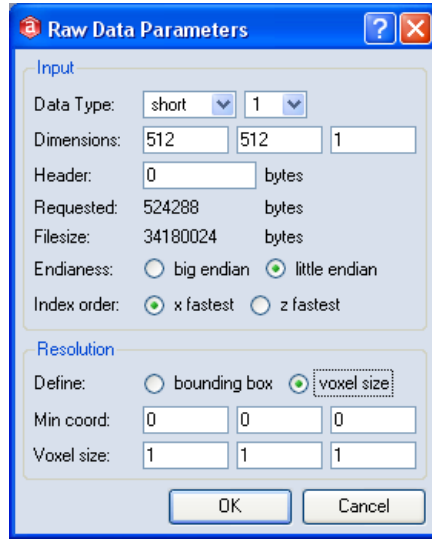


**Figure 8.1:** amira's raw data read dialog.

- **Endianness.** The byte order of data types other than byte is system dependent. If you read your files on the same type of processor as on which they have been produced, the default setting will be okay. But if you read data produced, e.g., under Linux (little endian) on an SGI (big endian), you must specify the correct byte order of the data to be read (little endian in this case).
- **Index Order.** Order in which the data points are read.

Instead of filling in this data dialogue, you might prefer to load the data using the Tcl interface. The syntax is

```
load -raw FileName Endianness IndexOrder DataType nDataVar dimX dimY dimZ xMin
```

where

- **FileName** is the name of the file to load,
- **Endianness** is either *little* or *big*,
- **IndexOrder** is either *xfastest* or *zfastest*,
- **DataType** is one of *byte*, *short*, *ushort*, *int32*, *float*, *double*,
- **nDataVar** is the number of data values per voxel, e.g., 1 for scalar fields, 3 for vector fields,
- **dimX dimY dimZ** denotes the dimensionality of the data set, and
- **xMin xMax yMin yMax zMin zMax** denotes the bounding box of the data set.

The raw data format is a very powerful tool, especially for quick-access/prototyping use. However it may sometimes be tricky to figure out the parameters. Some tools which may help are `vi` or `od -c` (to examine the header of files).

After loading, click on the icon. If the data range is obviously completely wrong, then you have either specified a wrong data type, or a wrong endianness, or a too short header. If most of the volume looks okay, but is shifted in the x direction, then you probably have specified a wrong header. If the data range looks okay, but the data seems scrambled, then you have specified incorrect dimensions or a wrong index order.

## 8.23 Raw Data as LargeDiskData

This file format allows to load subvolumes out of one large raw data block on disk. Use the *File Dialog's* Popup Menu to force the file format to "Raw Data as LargeDiskData". The format of the file and the parameters you have to provide are described in *Raw Data*. The file will be loaded as a *LargeDiskData* objekt.

The file is opened readonly. You cannot change the image data, nor add or modify parameters.

## 8.24 SGI-RGB Image Format

This is the SGI file format for RGB and grayscale images. When writing an RGBA color field the alpha value will be discarded. When reading RGB images full opacity is assumed. 16-bit data values, even though allowed by the SGI file format specification, are not supported. SGI-RGB files are automatically identified by the filename extensions `.sgi`, `.rgb`, or `.bw`.

Regarding the import and export of multiple slices the same remarks apply as for the *TIFF image format*. When reading SGI-RGB images the *channel conversion dialog* is popped up. This dialog is also described in the TIFF section.

## 8.25 STL

STL is a CAD format for Rapid Prototyping. It is a faceted surface representation, i.e., a list of the triangular surfaces with no adjacency information. Currently the ASCII version of the format is supported for writing *amira* objects of type Surface.

## 8.26 Stacked-Slices

This file format allows a stack of individual image files to be read, with optional z-values for each slice. The slice distance need not to be constant. The images must be one-channel images in an image

format supported by amira (e.g., TIFF). The reader operates on an ASCII description file, which can be written with any editor. Here is an example of a description file:

```
# Amira Stacked Slices

# Directory where image files reside
pathname C:/data/pictures

# Pixel size in x- and y-direction
pixelsize 0.1 0.1

# Image list with z-positions
picture1.tif 10.0
picture7.tif 30.0
picture13.tif 60.0
colstars.jpg 330.0
end
```

Some remarks about the syntax:

- `# Amira Stacked Slices` is an optional header, which allows `amira` to automatically determine the file format.
- `pathname` is optional and can be included in case the pictures are not in the same directory as the description file. A space separates the tag "pathname" from the actual pathname.
- `pixelsize` is optional too. The statement specifies the pixel size in the x and y direction. The bounding box of the resulting 3D image is set from 0 to `pixelsize*(number_of_pixels-1)`.
- `picture1.tif 10.0` is the name of the slice and its z-value, separated by a blank.
- `end` indicates the end of the description file.
- Comments are indicated by a hashmark character (#).

## 8.27 TIFF Image Format

This is the 2D TIFF file format. It can be used to read and write one or more 2D images. If multiple images of equal size have been selected in the *file dialog*, they will be combined into a single 3D image volume, i.e., a uniform scalar field of bytes or an RGBA color field. TIFF files will be automatically identified by looking at the file header, irrespective of the actual file name extension.

Likewise, if a 3D image data set is to be saved in TIFF format, in fact, for each slice a separate file will be created. If you choose the 2D TIFF format in the file dialog's format menu, a sequence of hashmark characters [#] will be automatically inserted into the filename. When saving the images, the hashmark sequence will be replaced by the current slice number (formatted with leading zeros).

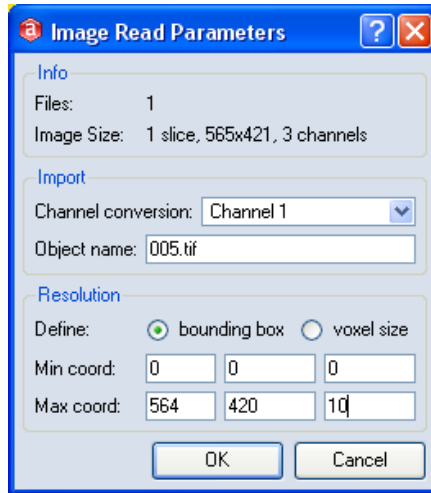


**Figure 8.2:** amira's channel conversion dialog.

For example, if the base filename is `image.####.tif`, the files actually being written will be named `image.0000.tif`, `image.0001.tif`, and so on.

Note that not all variants of the TIFF format are supported. In particular, the following limitations apply:

- The number of channels must be 1, 3 or 4.
- The number of bits per pixel must be 8 or 16 (1-channel images only).
- Images defined in YbCbR colorspace can not be read.
- Tiled images can not be read.
- Only scalar fields consisting of bytes can be saved.

## The Channel Conversion Dialog

When reading 2D image files a special dialog window will be popped up. This dialog asks the user to specify how the 2D images should be converted into amira data objects. In addition, the world coordinates of the resulting 3D data object can be adjusted. The channel conversion dialog looks as depicted in Figure 8.2.

First of all, the dialog displays the number of files to be read, the number of 2D slices (in most cases equal to the number files), the size of a 2D slice in pixels, as well as the number of channels stored in the files. An option menu lets you select whether a 1-component uniform scalar field should be created or a 4-component RGBA color field. Depending on the type of input not all options may be active.

The meaning of the individual items is described below.

- **Maximum.** The maximum value of the red, green, and blue channel is stored in a uniform scalar field (3- and 4-channel input only).
- **Weighted Average.** A grayscale uniform scalar field is created according to the NTSC formula,  $I = .3 * R + .59 * G + .11 * B$  (3- and 4-channel input only).
- **Channel 1.** The first channel of the input is converted into a uniform scalar field (will always be active).
- **Channel 2.** The second channel of the input is converted into a uniform scalar field (3- and 4-channel input only).
- **Channel 3.** The third channel of the input is converted into a uniform scalar field (3- and 4-channel input only).
- **Channel 4.** The fourth channel of the input is converted into a uniform scalar field (4-channel input only).
- **Color Field.** An RGBA color field is created (4-channel input or 1-channel input with additional colormap).

If the first image file contains a colormap, the map will be loaded as a separate object if *Channel 1* is selected, or it will be used to compute an RGBA color field if *Color Field* is selected.

Moreover, the dialog allows you to adjust the bounding box of the resulting image data set. The bounding box specifies the world coordinates of the center of the lower left front voxel (min values) and the center of the upper right back voxel (max values). For example, if your input is 256 pixels wide and the size of each voxel is 1mm, then you may set *xMin* to 0 and *xMax* to 255. The bounding box of a data object may also be changed later using the *ImageCrop Editor*.

## 8.28 Tecplot

Tecplot, Inc. delivers visualization software for engineers and scientists to analyze, discover, and communicate results. With this reader you're able to import your Tecplot data into *amira* and take advantage of *amira* visualization and computation modules.

The .tec extension is registered in *amira* for ASCII Tecplot files. The .plt extension is registered in *amira* for Binary Tecplot files.

If the `AMIRA_TECPLOT_USE_TIME_SERIES` environment variable is defined, then if more than one variable is defined in the `VARIABLES` section (after X, Y, and Z), the generated fields will be controlled through an `HxDynamicSeriesCtrl` *amira* object. This can be used, for instance, if each variable represents a different time step for a given parameter, so you can play back animation in *amira*.

### Limitations:

- The following ZONE types are supported:
  - IJK POINT and BLOCK generate HxHexaGrid data. ( $I_z1$  &&  $J_z1$  &&  $K_z1$ ). Each additional variable creates an HxHexaScalarField3.
  - FEPOINT and FEBLOCK TRIANGLE, QUADRILATERAL generate HxSurface data. Each additional variable creates an HxSurfaceScalarField.
  - FEPOINT and FEBLOCK TETRAHEDRON generate HxTetraGrid data. Each additional variable creates an HxTetraScalarField3.
  - FEPOINT and FEBLOCK HEXAHEDRON, BRICK generate HxHexaGrid data. Each additional variable creates an HxHexaScalarField3.

All other ZONE types are ignored.

- x, y, and z variables are always defined as the first three of the VARIABLES zone. So at least three variables must be defined. All variables must be defined between double quotes.
- C=, D=, DT= and NV= parameters and binary equivalents in the ZONE are ignored.
- All variables are read back as float values.
- Value replication using \* is not implemented. So 120\*0.000, for instance, is not supported.
- GEOMETRY, TEXT, CUSTOMLABELS sections are ignored.
- Maximum line length is 4096 in ASCII format.
- Maximum number of variables is 1024.

The binary reader follows the binary format generated by preplot.

## 8.29 VRML

VRML is a file format for storing 3D geometries. The format is especially popular for web or internet applications.

If you are using an amira version that is linked with the Open Inventor 2.5 or higher, you may import VRML files as scene graph objects. Use *IvToSurface* in order to convert the surface parts of the scene graph into a *surface* object.

On the other hand, surface objects may be exported into a VRML file using the *VRML-Export* module. Unlike most other formats, VRML-Export is implemented as a module rather than an export filter. The reason is that in addition to the surface itself, other data objects, e.g., 3D image data, can be included in the VRML file.





## Chapter 9

# Molecular Pack

### 9.1 AMBER

AMBER is a format consisting of three different file formats for storing structures of biomolecules and molecular simulations such as trajectories and its velocities. The three supported file formats are:

- \*.top files: saving the structural information of molecules such as atoms position, their residues. Additional information may be stored here (dihedral angles, meta information on the molecule). This file can be loaded on its own, since enough information to display the molecule are guaranteed.
- \*.crd files: storing for each time step the position of every atom of the molecule. Additional information is included for each time step, such as velocities, energy status. In order to load this file, a name.top file has to be loaded, and the name.md.crd file has to exist (where name is variable). It will be loaded automatically (no need to load the two files simultaneously).
- \*.vel files: storing for each time step the velocities of every atom of the molecule. Additional information is included for each time step, such as velocities, energy status. In order to load this file, a name.top file has to be loaded, and the name.md.vel file has to exist (where name is variable). It will be loaded automatically (no need to load the two files simultaneously).

#### Supported records

amira uses all information supplied in a AMBER Structure or Topology file. Following record types can be parsed while reading those files:

ATOMS RESIDUES DIHEDRALS ANGLES BONDS USER-DEFINED

Though not all information supplied by the trajectory File is parsed. Only the position of each atom is saved for every time step of the trajectory.

You can get further information about AMBER at <http://amber.scripps.edu/>

## 9.2 AMF

AMF is an XML based data description format for molecular data in Molecular Pack.

By default, AMF stores the xml file in gzip compression (files ending on amf.gz). If you want to edit the contents of the file, you may decompress it with the gzip program:

```
gzip -d Molecule.amf.gz
```

Both uncompressed and compressed files can be loaded.

AMF can be used to store 4 different datatypes which will be described in the following sections:

- Molecule
- MoleculeBundle
- MolTrajectory
- MolTrajectoryBundle

Each of these entries, which is encountered in a file, will be loaded into a separate data module in Amira.

**A first Example:** As a first simple example the following text shows how to define a water molecule. The levels "atoms", "bonds" and "residues" exist. The first two levels are of fixed group size. As root level, the atom level has no reference level and has therefore groups of size 0. Thus, the GROUP element can be omitted. Bonds reference two atoms and are therefore of fixed group size 2. Residues reference an arbitrary number of atoms and the first number for each residue reference in the groups element therefore needs to contain the number of atoms referenced by the residue (in this case 3), followed by these reference indices (0 1 2).

```
<?xml version="1.0" encoding="ASCII"?>
<!DOCTYPE AMF SYSTEM "amf.dtd">
<AMF version="1.00">
<MOLECULE name="water">
<TOPOLOGY>
<LEVEL name="atoms" size="3" fixedgroupsize="0">
<ATTRIBUTE name="atomic_number" type="int32">
1 8 1
</ATTRIBUTE>
```

```

<ATTRIBUTE name="MMFF94_type" type="string">
<![CDATA[HOH OH2 HOH ]]>
</ATTRIBUTE>
</LEVEL>
<LEVEL name="bonds" size="2" fixedgroupsize="2">
<GROUPS>
1 0 1 2
</GROUPS>
<ATTRIBUTE name="type" type="int32">
1 1
</ATTRIBUTE>
</LEVEL>
<LEVEL name="residues" size="1">
<GROUPS>
3 0 1 2
</GROUPS>
</LEVEL>
<COORDINATES>
0.251 -0.36 -0.046 0.249 0.684 0.231 0.586 -0.954 0.791
</COORDINATES>
</MOLECULE>
</AMF>

```

Now suppose you want to have a trajectory of a water molecule with 3 time steps. How would the file differ? First of all the `MOLECULE` element would be replaced by a

```
<MOLTRAJECTORY name="water" size="3">
```

Next the `COORDINATES` element needs to contain 3 times as many coordinates. Finally, a trajectory may contain observables, which for example might be the intramolecular energy per time step as computed during a simulation in an NVT ensemble.

```

<OBSERVABLE name="intramolecular_energy">
-1.31 -1.29 -1.30
</OBSERVABLE>

```

### **Molecule:**

A molecule consists of a topology, optional coordinates and an arbitrary number of fields.

A topology consists of an arbitrary number of levels which contain groups and an arbitrary number of attributes.

There is a set of predefined levels and attribute which is used by some of the data, viewer or computational modules in amira. A molecule must have the root level "atoms" which needs to be of fixed

group size 0. The atom level must contain the attribute "atomic\_number". Bonds must be defined in the level "bonds" of fixed group size 2 and usually have the attribute "order" which has the following meaning: 0=undefined, 1=single, 2=double, 3=triple, 4=aromatic bond.

Modules for sequence alignment additionally require a level "residues" with an attribute "type".

Modules requiring secondary structure information require a level "secondary structure" with the reference level "residues" and the attributes "type" which can be "helix", "sheet" or "turn".

### **MolTrajectory:**

MolTrajectories are saved like Molecules except that they can contain several different timesteps (i.e. sets of coordinates) and an arbitrary set of observables. Observables are fields which contain one value per timestep.

Unlike with Molecules, the coordinate section of MolTrajectories is not completely read. This is because very large trajectories may not fit entirely into system memory. Instead, filepositions are stored in a table to allow fast access to a certain timestep. However, random access inside of a compressed file is inefficient. To increase speed when showing trajectories, decompress the file first.

### **Data types**

Data can be stored as int32 (c-type: integer), float, or string(c-type: char[]).

### **ASCII Data**

The encoding of the file must be ASCII.

Data tokens in PCDATA records must be separated by one or more whitespaces.

### **DTD:**

```
<!ELEMENT AMF (MOLECULEBUNDLE*,MOLTRAJECTORYBUNDLE*,MOLTRAJECTORY*,MOLECULE*)>
<!ATTLIST AMF
    version CDATA #REQUIRED>

<!ELEMENT MOLTRAJECTORYBUNDLE (MOLTRAJECTORY*)>
<!ATTLIST MOLTRAJECTORYBUNDLE
    name CDATA #REQUIRED>

<!ELEMENT MOLECULEBUNDLE (MOLECULE*)>
<!ATTLIST MOLECULEBUNDLE
    name CDATA #REQUIRED>

<!ELEMENT MOLTRAJECTORY (TOPOLOGY,COORDINATES?,OBSERVABLE*)>
<!ATTLIST MOLTRAJECTORY
    name CDATA #REQUIRED
    size CDATA #REQUIRED>

<!ELEMENT MOLECULE (TOPOLOGY,COORDINATES*)>
<!ATTLIST MOLECULE
```

```

    name CDATA #REQUIRED>

<!ELEMENT TOPOLOGY (LEVEL+,DATA*)>

<!ELEMENT LEVEL (GROUPS?,ATTRIBUTE*)>
<!ATTLIST LEVEL
    name CDATA #REQUIRED
    size CDATA #REQUIRED
    refllevel CDATA #IMPLIED
    fixedgroupsize CDATA #IMPLIED>

<!ELEMENT GROUPS (#PCDATA)>

<!ELEMENT ATTRIBUTE (#PCDATA)>
<!ATTLIST ATTRIBUTE
    name CDATA #REQUIRED
    type (int32|float|string) #REQUIRED>

<!ELEMENT DATA (#PCDATA)>
<!ATTLIST DATA
    name CDATA #REQUIRED
    type (int32|float|string) #REQUIRED
    size CDATA #REQUIRED>

<!ELEMENT COORDINATES (#PCDATA)>
<!ATTLIST COORDINATES
    size CDATA #REQUIRED>

<!ELEMENT OBSERVABLE (#PCDATA)>
<!ATTLIST OBSERVABLE
    name CDATA #REQUIRED
    type (int32|float) #REQUIRED>

```

#### Version History:

## 9.3 GROMACS

Gromacs is a format consisting of three different file formats for storing structures of biomolecules and molecular simulations such a trajectories. The three supported file formats are:

- \*.gro files: saving the structural information of molecules such as atoms position, their residues. Additional information may be stored here (dihedral angles, meta information on the molecule).

This file can be loaded on its own, since enough information to display the molecule are guaranteed.

- \*.top files: saving topological information of molecules such as bonds, secondary structures. This file must be loaded in combination with the structural file (\*.gro) since it is not providing enough information to display the molecule.
- \*.trr files: storing for each time step the position of every atom of the molecule. Additional information is included for each time step, such as velocities, energy status

### Supported records

amira uses all information supplied in a Gromacs Structure or Topology file. Following record types can be parsed while reading those files:

ATOMS RESIDUES DIHEDRALS ANGLES BONDS USER-DEFINED

Though not all information supplied by the trajectory File is parsed. Only the position of each atom is saved for every time step of the trajectory.

When saving a Molecule, the following record types will be written:

ATOMS RESIDUES BONDS

When saving a Trajectory, only the positions of each atom in the molecule will be saved.

You can get further information about GROMACS at <http://www.gromacs.org>

## 9.4 MDL

The *sdf* file format is part of the set of MDL file formats used for saving chemical structures.

amira uses and writes the information of the atom and bond blocks.

## 9.5 PDB

PDB is a file format for storing structures of biomolecules. The format is used for archiving molecules in the online protein data base. [<http://www.rcsb.org/pdb/>]

### Supported records

amira uses most, but not all, information supplied in a PDB file. The following record types will be parsed:

HELIX SHEET TURN SITE MODEL ENDMDL ATOM HETATM TER CONECT END

The rest will be skipped.

When saving a structure, the following record types will be written:

```
HEADER TITLE AUTHOR  
SEQRES HELIX SHEET TURN SITE SSBOND  
ATOM HETATM TER CONECT MASTER END
```

amira supports PDB files that follow the conventions of PDB format version 2.0 or higher.

### How to identify PDB groups in amira

In amira all levels (atoms, residues, bonds , ...) contain a set of information fields called attributes. Every level contains the field 'index' which is an internal identifier that does not necessarily correspond to the succession of the groups in the file.

If the molecule has been read from a PDB file then the **atom** 'name' field is composed of the chain name and the sequence number (columns 7 - 11) of the respective ATOM records. The field 'index' only corresponds to the sequence number if the enumeration of pdb atoms is continuous.

The 'type' field contains the information of the field named 'atom name' (columns 13 - 16) in the PDB file.

The 'name' field of **chains** contains the chainID (column 22) of the respective ATOM record. HETATM records whose chainID field is empty will not belong to any chain in amira. The chain name of atoms belonging to ATOM records whose chainID field is empty will be set to 'NIL'.

The 'name' field of **residues** is composed of the chain name and the residue sequence number (columns 23 - 26). If the residue is a het-group, the name will start with 'HET'.

The field 'type' contains the residue names (columns 18 - 20). If none is given, the field will be set to 'UNK' (for unknown).

### Example:

```
ATOM    441  CB  VAL L   58    21.025   37.362  -2.515   1.00   8.15   1IGM  562
```

The atom will be known with name 'L441' and comment 'CB'. It will belong to the residue with name 'L58' and of type 'VAL' which is sited on the chain with name 'L'.

It is possible to save trajectories in the PDB format. Note, however, that this is inefficient due to the redundancy of information of the atom records in each model entry.

### Possible problems

Some molecular modelling packages use slightly altered conventions for saving PDB files. If you encounter any problems in reading such files please check the following section and adjust your files as necessary.

amira can read **several models of one molecule as a trajectory** if the different versions are separately declared between MODEL and ENDMDL records. If there are several possible conformations in one model (by using alternate location identifiers), amira will read only the first version (that one with the altLocId 'A').



Molecules that are declared as models need to have the same number of atoms as **amira** will only read the topology once. Only the coordinates are read in for each timestep. If you have **several different molecules** in one PDB file, you can separate them by END records. **amira** will load every molecule ended by an END record separately, creating its own data object.

The **automatic creation of bonds** of peptide and RNA/DNA residues is done by comparing the atoms of the residues with a residue database. The method requires the atom names and the residue names to follow the PDB standard. Sometimes third party programs write atoms like O3\* as O3'. This will prevent **amira** from recognizing the atoms in the database. Connections between consecutive residues is done by checking the sequence of residues of each chain. **Bonds of non-standard residues** must be declared in the connect records.

Columns 77-78 should contain the element name of the atom. In some PDB files, this field is empty or contains other information. In this case, **amira** uses the first two letters of the atom name to determine the atom type. However, in special cases this is not unequivocal. The field is not allowed to contain non-numerical information.

All information in columns 79-80 is interpreted as the charge of the atom. Thus, line numbers in this field will lead to strange charge values.

You can get complete documentation of the file format at <http://www.rcsb.org/pdb/>.

## 9.6 PSF/DCD (CHARMM)

This is a format used by CHARMM. It consists of two files, one containing the structural information (suffix .psf), and one containing the trajectory, i.e., the atom coordinates (suffix .dcd).

**amira** can read this format and translates the sections of the psf file into grouping levels.

## 9.7 Tripos

The *mol2* file format is used to save Tripos Sybyl molecule information. See more at <http://www.tripos.com/services.old/mol2/index.html>.

**amira** uses and writes the information given in the molecule atom and bond records.

## 9.8 UniChem

The UniChem file format (extension *uni*) is used by the UniChem molecular modelling software to save the structure of molecules and computational results. See more at <http://www.oxmol.com/software/unichem/>.

**amira** uses all structural information supplied in the COORDINATES and BONDS blocks. Files that are exported by **amira** follow file format version 4.0 and can be used with the UniChem software.

## 9.9 ZIB Molecular File Format

The *zmf* file format is a structured file format developed for exchanging molecular data.

This description is divided into two parts. First we describe the general and context-independent file structure. Second, we explain our structuring conventions for molecular data.

### Structured files

A structured file consists of a header and two sections: a plain text section describing the file's structure, and a binary section containing the actual data:

```
STRUCTURED FILE V0.1 BINARY_BE MOLECULE_TRAJECTORY
TYPE
struct {
    atoms : array of struct {
        id      : integer*4;
        name     : string;
        mass     : real*4;
        properties : integer*4[8];
    };
};
DATA;
..... binary data .....
```

The file structure is similar to a structured data type in C or PASCAL. There are two composite data types (**array** and **struct**), and three elementary data types (**integer**, **real**, and **string**). Finally there exists a special type (**extern**) for including other structured files.

**struct** contains named fields of arbitrary heterogeneous type.

**array** contains numbered fields of the same type. Arrays are one-dimensional only. Their size can either be specified in the type declaration (Example: **array[3] of integer**) or in the data section.

**real** and **integer** contain numerical, possibly multi-dimensional, data. The number of bytes in an element may be specified using **"\*n"**. At the moment only 4-byte types are allowed. The rank (number of dimensions) must be specified in the type declaration. The number of elements in each dimension may be specified in the type declaration or in the data section (indicated by **":"**). Example: **integer\*4[4, ., 3]**.

**string** contains ASCII data of arbitrary length. Again the length can be specified in the type declaration (Example: **string\*22**) or in the data section.

**extern** is a special kind of a **string**. The length must be specified in the data section. The string is interpreted as a structured file whose contents should appear as part of the current file structure.

The data section is written as follows:

**struct:** The individual members are written in the given order.

**array:** The elements are simply written one after another. If the array size is unspecified, it is written in front of the array as a 4-byte integer.

**string** and **extern:** The data is written byte by byte. If the size is unspecified in the data type, it is prepended as a 4-byte integer.

**real** and **integer:** Individual elements are written in the format specified in the file header (big endian or little endian). Floating point numbers are in IEEE format. Matrices are written in FORTRAN order. If any dimension is unspecified in the data type, it is written as a 4-byte integer before the data. If more than one dimension is unspecified, they (and only they) are written before the data in their natural order. If a dimension is specified in the data type, it is never written in the data. Example:

```
integer[2,.,4] = [ [ [1,2,3,4], [5,6,7,8], [9,10,11,12] ],  
                  [ [13,14,15,16], [17,18,19,20], [21,22,23,24] ] ]
```

is written as a sequence of 4-byte integers.

```
3 1 13 5 17 9 21 2 14 6 18 10 22 3 15 7 19 11 23 4 16 8 20 12 24
```

The leading 3 specifies the dimension unspecified in the data type. The 24 numbers that follow it are the entries of the array.

## Molecular data - a minimal file

When reading a structured file containing molecular data, **amira** expects the file structure to have certain elements. Moreover the file may have arbitrary additional elements, which will be ignored. The minimal structure a molecular data file must supply is:

```
struct {  
    molecules : array of struct{  
        name : string;  
        groupings : struct {  
            atoms : array of struct{  
                id : integer*4;  
                type : integer*4;  
                name : string;  
            };  
        };  
        trajectory : array of struct{  
            coordinates : real*4[3,.];  
        };  
    };  
};
```

```

};
typbase : struct {
    atoms : array of struct{
        id : integer*4;
        name : string;
        number : integer*4;
        radius : real*4;
    };
};
};

```

In the following we will use a notation inspired by C to denote elements of the structure. A `struct`'s field will be specified as the `struct` followed by a “.” and the name of the field. For array elements we will write the array followed by brackets (“[]”). For example, “`molecules[].groupings.atoms[].id`” means the field “`id`” in an element of the array “`atoms`” which is a field of the `struct` “`groupings`” which is a field of an element of the array “`molecules`”.

The overall type of the file is a `struct` containing the fields **`molecules`** and **`typbase`**. **`molecules`** is an array of elements, each of which describes a molecule. **`molecules[].name`** contains the name of the molecule.

**`molecules[].groupings`** specifies the topology of the molecule, i.e., which atoms it contains. Every element of **`molecules[].groupings.atoms`** describes one atom of the molecule. **`atoms[].id`** is a unique numerical identifier. **`atoms[].type`** is a numerical reference to a field in the **`typbase`** described below. **`atoms[].name`** is a textual identifier unique with respect to all atoms. It is used to specify this atom on the command line.

**`molecules[].trajectory`** is an array with every element describing one time step of a molecule trajectory. **`trajectory[].coordinates`** has the same number of elements as **`molecules[].groupings.atoms`** and every element specifies a position for the corresponding atom.

**`typbase`** is structured analogously to **`molecules[].groupings`**. Corresponding to **`molecules[].groupings.atoms`**, the array **`typbase.atoms`** holds information common to atoms of the same type. Specifically, the type information of an atom is stored in the **`molecules[].groupings.atoms[].type`**'th element of **`typbase.atoms`**.

**`typbase.atoms[].id`** holds a numerical identifier that should be identical to the element index. **`typbase.atoms[].name`** is a textual label for making the types user transparent. **`typbase.atoms[].number`** and **`typbase.atoms[].radius`** are the atomic number and radius of an atom having this type.

## Bonds

Bonds can be specified by adding an array named **`bonds`** to **`molecules[].groupings`**:

```

bonds : array of struct{

```

```

        id : integer*4;
        type : integer*4;
        components : integer*4[2];
        index : integer*4;
    };

```

**bonds[].id** is a unique numerical identifier which should be greater than any value specified in **atoms[].id**. Analogous to **atoms[].type** for atoms, **bonds[].type** specifies a numerical type for every bond, although this information is not yet interpreted by *amira*. The array **bonds[].components** holds the numerical identifiers of the two atoms connected by the bond. **bonds[].index** finally indicates if it is a single (1), double (2), triple (3), or an aromatic bond (4). The value 8 indicates a bond of unknown type.

## Molecular data - a big example

The information stored in the described file format can be divided into different degrees of generality. Information stored in **typbase** is specific to the molecular force field used in the calculations leading to a molecular trajectory. It would be redundant to store this information in every file for which the same molecular force field was used. Thus it can be stored in an extra file, which is then referenced by an **extern** declaration.

A similar mechanism applies if more than one trajectory is calculated for the same molecule. The topological information about atoms making up the molecule and bonds connecting them will be the same for all the trajectories. On the other hand, we do not want a file for every single molecule topology. Thus our trajectory can reference another file containing a multitude of molecular topologies and just supply the time steps itself.

The following is an example realizing these concepts:

### trajectory file:

```

struct {
    typbase    : extern;
    molecules : array of struct{
        id            : integer*4;
        observations : struct {
            global_obs : string[.];
        };
        text          : string[.];
        name           : string;
        molbase        : extern;
        trajectory     : array of struct{
            global_obs : real*4[.];
            coordinates : real*4[3,.];
        };
    };
}

```

```

    };
};
};

```

**molbase file:**

```

struct {
    typbase    : extern;
    molecules  : array of struct{
        id      : integer*4;
        observations : struct {
            global_obs    : string[.];
        };
        text : string[.];
        name : string;
        groupings : struct {
            atoms : array of struct{
                id      : integer*4;
                type     : integer*4;
                name     : string;
            };
            bonds : array of struct{
                id      : integer*4;
                components : integer*4[2];
                type     : integer*4;
                index    : integer*4;
            };
            dihedrals : array of struct{
                id      : integer*4;
                components : integer*4[4];
                type     : integer*4;
            };
            residues : array of struct{
                id      : integer*4;
                name     : string;
                remark   : string;
                type     : integer*4;
                from_id  : integer*4;
                to_id    : integer*4;
            };
        };
        trajectory : array of struct{
            coordinates : real*4[3,.];
        };
    };
};

```

```

};
};
};

```

**typbase file:**

```

struct {
    atoms    : array of struct{
        id      : integer*4;
        type    : integer*4;
        name    : string;
        number  : integer*4;
        periodic_row : integer*4;
        mass    : real*4;
        charge  : real*4;
        partial_bond_charge_increment : real*4[2];
        radius  : real*4;
        N_i     : real*4;
        A_i     : real*4;
        g_i     : real*4;
        alpha   : real*4;
        datyp   : string;
        properties : integer*4[8];
        equivalence : integer*4[4];
    };
};

```

Some yet undescribed features appear in this example. The first is the possibility to supply scalar observables corresponding to the single time steps. Possible values are different energies like kinetic and potential energy. The names of the observables are read from **molecules[.observations.global\_obs[]** in the trajectory file. Their values are stored in **molecules[.trajectory[.global\_obs[]**.

Besides **bonds**, the file can contain arbitrary groups of atoms or groups of groups. These groups are organized in levels which appear here as the elements of **molecules[.groupings**.

Every level is an array whose elements describe the particular groups. Every group is a struct with the fields **id** and **type**, and also fields specifying the contents of the group. **id** is numerical identifier unique over all groups specified for the molecule. Specifically, the identifiers in one level should be consecutively numbered and bigger than all the identifiers used in the preceding levels.

There are two ways of specifying a group. The general way is to specify all elements explicitly. In that case, the group contains a field **components[]** containing the ids of all elements of the group. If the elements of a group cover a consecutive sequence of ids, it is sufficient to specify only the first and the last id. In that case the group contains the fields **from\_id** and **to\_id**.

**dihedrals** are groups of four atoms connected by three bonds like a chain. Every such group defines two intersecting planes. The angle of intersection is called a *dihedral angle* or *torsion angle*. For performance reasons amira will read *dihedrals* only if the molecule has less than 500 atoms. Torsion angles can be displayed with the *MoleculeView*.

**residues** are groups representing the building blocks of a complex molecule, for example the amino acids of a protein. All groupings can be used to color the molecule in the *MoleculeView* or in the *BondAngleView* using the general *coloring facilities*.





# Chapter 10

## Large Data Pack

### 10.1 LDA

This is the native VolumeViz file format for storing hierarchical multi-resolution volume data, defined by Mercury. It can be a single file (.lda) or coupled with a .dat file. In the latter case, the .lda file references the .dat file. This innovative technology allows rendering of extremely large data sets with reliable interactive navigation even on relatively low-end machines.

### 10.2 LargeDiskData

This is the native amira file format for blockwise access of image data stored on disk as described in *LargeDiskData*. It supports read/write operations, multi-resolution access and saving of changed parameters. The data is stored in a couple of files, e.g.:

```
visMaleCT  
visMaleCT-SUPR.dat  
visMaleCT-0001.dat  
visMaleCT-0000.dat  
visMaleCT-0002.dat
```

The first one is recognized by **amira** as an AmiraMesh file. It contains the parameters and a link to the other files. It can be loaded into **amira** with the *File Dialog*. The other files contain the actual image data.

You can create such a fileset with the *ConvertToDiskData* module.

## 10.3 Stacked-Slices as LargeDiskData

Stack of image files can be loaded as `LargeDiskData`. This file format allows to read a stack of individual image files with optional z-values for each slice. The slice distance need not to be constant. The images must be one-channel images in an image format supported by `amira` (e.g., TIFF). The reader operates on an ASCII description file, which can be written with any editor.

Here is an example of a description file:

```
# Amira Stacked Slices as ExternalData

# Directory where image files reside
pathname C:/data/pictures

# Pixel size in x- and y-direction
pixelsize 0.1 0.1

# Image list with z-positions
picture1.tif 10.0
picture7.tif 30.0
picture13.tif 60.0
colstars.jpg 330.0
end
```

The format of the file and the parameters you have to provide are described in *StackedSlices*.

The image files are opened readonly. You cannot change the image data but you can add or modify parameters in the description file. The file will be loaded as a *LargeDiskData* object.

# Chapter 11

## Mesh Pack

### 11.1 AVS Field

This format provides a bridge between `amira` and AVS, the *Advanced Visual System* software. It enables you to read and write data in the native AVS field format.

Note that AVS supports fields of arbitrary dimensions and with an arbitrary number of components per node. For some combinations there are no corresponding `amira` data objects. Thus, the following restrictions apply:

- Only three-dimensional fields will be handled.
- AVS fields with a one-component data vector at each voxel will be loaded as a *regular scalar field*.
- Two-component fields will become *regular complex fields*.
- Three-component fields will become *regular vector fields*.
- Four-component fields will be loaded as *RGBA color fields*, provided they are defined on a uniform lattice. If not, they are rejected.
- Six-component fields will be loaded as *regular complex vector fields*.
- Since AVS does not support *stacked coordinates* these will be written as rectilinear fields and therefore appear as such when reloaded into `amira`.

`amira` identifies AVS Field files by the file name suffix `.fld`.

## 11.2 AVS UCD Format

The AVS UCD (*Unstructured Cell Data*) format can be used to represent finite-element grids and associated data fields in 2 and 3 dimensions. Currently, in *amira* only 2D triangular cells, 3D tetrahedral cells, and 3D hexahedral cells are supported. Grids with these cells types will be converted into objects of type *Surface*, *TetraGrid*, or *HexaGrid*, respectively. Corresponding data fields will be converted into appropriate *amira* objects as well, provided the data is defined on a per-node basis. Cell data are currently not supported.

Two variants of the AVS UCD format exist, an ASCII version and a binary version. *amira* is able to read and write both of them. *amira* identifies AVS UCD data files by the file name suffix `.inp`.

## 11.3 FIDAP NEUTRAL

The *FIDAP NEUTRAL* describes 3D simulations on geometries consisting of 3D vertices and a number of geometry elements based on them. *amira* can only read *FIDAP NEUTRAL* files in plain ASCII format containing 3D triangular/quad surfaces, and tetrahedral grids.

After describing the geometry, the *FIDAP NEUTRAL* file contains a number of time steps, each time step specifying the same subset set of data sets defined on the 3D nodes (velocity, pressure, temperature etc.). *amira* loads the geometry and then displays a dedicated module called *FIDAPControl* which allows the user to select the desired timestep. In this way, the evolution of the data in time can be followed.

## 11.4 Fluent / UNS

The *Fluent* file format is used for storing 2D and 3D geometries, such as unstructured finite-element grids. The format is quite powerful. Currently, *amira* only supports *Fluent* files in plain ASCII format containing 3D triangular surfaces, tetrahedral or hexahedral grids.

When exporting *Fluent* files, additional information defined in *amira*'s surface or grid structures such as the material section is saved as well. However, other applications won't be able to read or interpret these additional data.

*Fluent* files are identified automatically by analyzing the file header.

## 11.5 Hypermesh

The *Hypermesh (TM)* file format is used by the *Altair HyperWorks* product family. The format describes 3D geometries consisting of 3D vertices and a number of cells based on them. *amira* can only read *Hypermesh* files in plain ASCII format containing 3D triangular surfaces, tetrahedral grids, or

so-called *tria6* components (a special kind of triangles). If tetrahedral components are present, the result of the import is a TetraGrid, otherwise it is a 3D surface.

Tetrahedral grids and 3D surfaces can also be exported in *Hypermesh* format. Every 3D surface component (patch) is saved with its interior-exterior material names, separated by a "-". However, it is not guaranteed that other applications understand this coding. A tetrahedral grid is saved together with its boundary surface. The patch structure of the boundary surface is retained. In order to separate the boundary into different patches, the boundary condition ids are interpreted. The material ids of every tetrahedron are also saved.

amira identifies *Hypermesh* files by the extensions `.hm` or `.hmascii`. The decision whether the file contains a 3D surface or a tetragrid is made after reading and analyzing the content.

## 11.6 IDEAS universal format

The *I-DEAS universal* file format is a general format originally used by SRDC's I-DEAS Master Series software to encode CAD/CAM models and FEM simulation results. The format is able to store lots of different data entities and FEM cell types. In amira the following subset of so-called *universal dataset numbers* is supported when importing I-DEAS files:

- **15:** Nodes with single precision coordinates.
- **781, 2411:** Nodes with double precision coordinates.
- **71, 780, 2412:** Cell definition. Lines, triangles, quads, tetrahedra, hexahedra, and prisms with linear shape functions are supported. Quads are decomposed into triangles. Prisms are decomposed into triangles as well. Hexahedra are decomposed into tetrahedra if tetrahedra or prisms appear too (amira currently is not able to handle meshes with mixed element types or elements with higher-order shape functions).
- **55, 2414:** Data at nodes.

The common file name extension of this format is `.unv`. For *I-DEAS* files with some other extension the file format has to be specified manually via the popup menu of the file dialog. Surfaces, tetrahedral grids, and hexahedral grids can also be exported into an *I-DEAS universal* file.

## 11.7 Plot 3D Single Structured

Plot3D is a simple binary file format, to represent structured curvilinear grids and scalar or vector fields defined on these grids. This format originates from the Plot3D program developed by Pieter Buning at NASA Ames. This reader will also accept complex valued solution files.

To read data in this format, you have to select a *Grid* file, and optionally additional *scalar*, *vector*, or *solution* files. The preferred file name suffix for Plot3D files is `.p3d`. In order to load Plot3D files from the command line use `load -plot3d`.

To write a data object in Plot3D format first create the grid file by choosing *Plot3D Grid File* in amira's file dialog. Then write the data itself by choosing *Plot3d Data File*. From the command line you may use `save -plot3dgrid` and `save -plot3ddata`, respectively.

The Plot3D may contain binary 32-bit values in either big or little endian format. The reader will attempt to detect the correct type and convert the data if necessary. In detail Plot3D files must have the following structure:

#### **Grid File:**

3 integers:

[IDIM, JDIM, KDIM]

IDIM x JDIM x KDIM (call this product NPOINTS) floating-point X coordinates (bracket indicate one record):

[x1,x2,...,xNPOINTS,

NPOINTS floating-point Y coordinates:

y1,y2,...,yNPOINTS,

NPOINTS floating-point Z coordinates:

z1, z2,...,zNPOINTS]

#### **Solution File:**

3 integers:

[IDIM, JDIM, KDIM]

4 floating-point conditions:

(free-stream mach number, angle-of-attack, Reynold's number, and integration time)

[FSMACH, ALPHA, RE, TIME]

IDIM x JDIM x KDIM (call this product NPOINTS) floating-point Q1 values (bracket indicate one record):

(Q1 is density (RH0), may be complex)

[q11, q12, ..., q1NPOINTS,

NPOINTS floating-point Q2 values:

(Q2 is x momentum ( $\text{RHO} \cdot U$ ), may be complex)

q21, q22, ..., q2NPOINTS,

NPOINTS floating-point Q3 values:

(Q3 is y momentum ( $\text{RHO} \cdot V$ ), may be complex)

q31, q32, ..., q3NPOINTS,

NPOINTS floating-point Q4 values:

(Q4 is z momentum ( $\text{RHO} \cdot W$ ), may be complex)

q41, q42, ..., q4NPOINTS,

NPOINTS floating-point Q5 values:

(Q5 is total energy per unit volume ( $E$ ), may be complex)

q51, q52, ..., q5NPOINTS]

**Scalar File:**

4 integers:

[IDIM, JDIM, KDIM, 1]

IDIM x JDIM x KDIM (call this product NPOINTS) floating-point F values:

[f1, f2, ..., f1NPOINTS]

**Vector File:**

4 integers:

[IDIM, JDIM, KDIM, 3]

IDIM x JDIM x KDIM (call this product NPOINTS) floating-point F values:



```
[fx1,fx2,...,fxNPOINTS,  
NPOINTS floating-point FY values:  
fy1,fy2,...,fyNPOINTS,  
NPOINTS floating-point FZ values:  
fz1,fz2,...,fzNPOINTS]
```

# Chapter 12

## DICOM reader

### 12.1 ACR-NEMA

The ACR-NEMA file format is the predecessor of the DICOM standard file format for medical images. *amira* can import both old ACR\_NEMA files and new DICOM files. Actually, both formats are interpreted by the same reader. Therefore, for more information please refer to the documentation of the *DICOM reader*. In contrast to DICOM files ACR\_NEMA files can not be exported by *amira*.

### 12.2 DICOM

The DICOM data format and its predecessor the ACR-NEMA format are widely used to exchange medical image data, provided by various modalities. DICOM stands for **D**igital **I**maging in **C**ommunications and **M**edicine, and it was originally designed as pure transfer format between imaging modalities and image retrieval systems (client/server). The data stream has a so called *tagged* format, with a variable amount of tags (DICOM data elements). Each element is defined by a unique group-element identifier. These group-element pairs are always sorted in ascending order within the DICOM data stream.

In *amira* the import of sequences of axis-aligned CT or MRI images *stored* in DICOM or ACR-NEMA format is supported. It will be checked whether the spacing between subsequent slices is constant or not. In the first case an image stack with uniform z-coordinates is created, in the latter one so called stacked coordinates are used. Nonuniform stacks can be easily converted into uniform stacks using the *arithmetic module*. Both image stacks, either uniform or stacked, can be segmented using the *image segmentation editor*. Such labeled data can be converted into polygonal models using the *SurfaceGen* module, and furthermore into tetrahedral models using the *TetraGen* module.

When reading DICOM or ACR-NEMA image stacks, all files of the data volume must be selected simultaneously within the file browser. This is done by selecting each file with a mouse

click holding the control key down or by clicking the first file and then shift-clicking the last one. amira automatically identifies files in DICOM or ACR-NEMA format if the file name suffix is .dcm, dc3, .ima or .ani, if the file name matches a DICOM unique instance ID, e.g., 1.3.12.2.1107.5.1.2.20395.19980429 . . . , or if the DICM sequence can be found at byte position 128 within the data stream. Individual files are automatically uncompressed if they were compressed using gzip or compress, and if the file name ends with the suffix .gz or .Z.

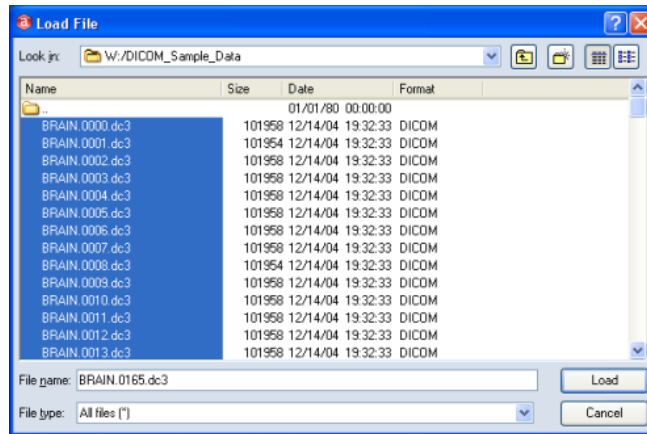


**Figure 12.1:** Selecting multiple DICOM files in the file dialog.

After choosing files from the file selection dialog, a list of images is displayed within the DICOM loader dialog view (see below), which allows you to adjust several parameters for image stack generation. This intermediate step is necessary because the conversion of sequences of DICOM images into image stacks can be ambiguous, although in many cases the standard settings will produce the desired results. The evolution of the ACR-NEMA/DICOM file format is fast and striking. However, the primary goal of reading image stacks into amira is the consistency of the entire data volume. The Loader has to handle *retired* data elements, different *transfer syntaxes*, explicit or implicit *value representation*, *image compression*, and *multi-frame data*, to name a few specialities. Furthermore, the availability of certain tags is not always guaranteed, and so-called *private* data elements can be added at will by any creating instance. Further information on the DICOM 3.0 data format can be found in NEMA: Standards Publications PS3.x

## The DICOM Load Dialog

After selection of DICOM/ACR-NEMA files from the file selection dialog, all images stored within these files are listed within the DICOM Loader dialog. The images are initially sorted by location in ascending order. Images of different studies are grouped into separate image stacks. Each stack is represented by a stack symbol and the patient's name if available. Stacks with equally distributed images are depicted with a uniform stack symbol, and stacks with variable spacing have a non-uniform

stack symbol. Single slices are represented by image icons within the list view. Clicking on a stack or on one of its images will display additional information in the top area of the dialog's view. To preview an individual image, right click on it and select *Image Preview* from the popup menu.

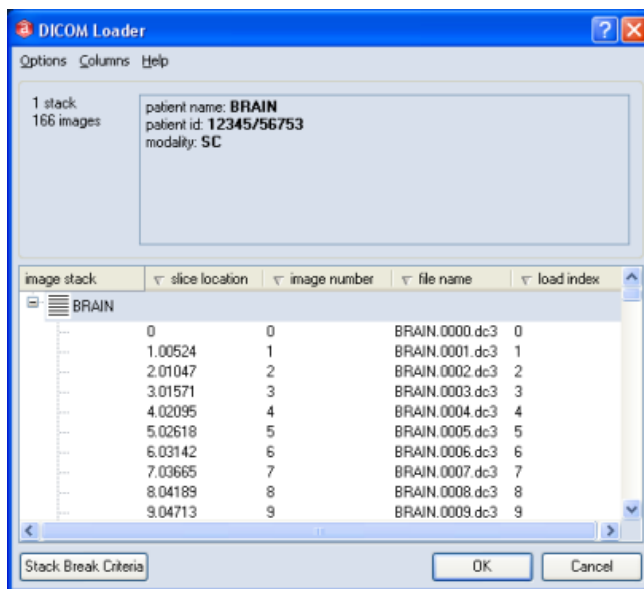


**Figure 12.2:** The DICOM load dialog.

Sorting order and policy can be modified by clicking on the column headers or by moving columns from one position to another. The order of the columns' precedence defines the significance of the sort keys. Clicking on a column's header toggles the sorting order between ascending and descending, depicted by an arrow within the column's header. The leftmost column has the highest priority. Rows with equal contents, e.g., equal slice locations, can be subsorted depending on the order and contents of all remaining columns.

Sorted sequences of images are automatically broken into substacks when certain image parameters do not coincide, i.e., the image/pixel size or the bits/samples per pixel vary. Furthermore there are stack criteria like patient name, patient id, series instance uid etc. that are supposed to enforce stack consistency but can be overridden or additionally be set by the user.

Building image stacks for 3D reconstruction requires unique per-slice locations, thus image stacks of a single study are automatically broken into substacks if duplicate slice locations occur. This is especially the case when large areas were scanned in several steps. Such duplicate slices can be removed by right clicking on the image row, choosing *remove image* from the popup menu. Remaining images are automatically combined to one stack if they are not affected by any other stack break criterion. Stack break criteria can be modified or disabled within the dialog that is displayed after you

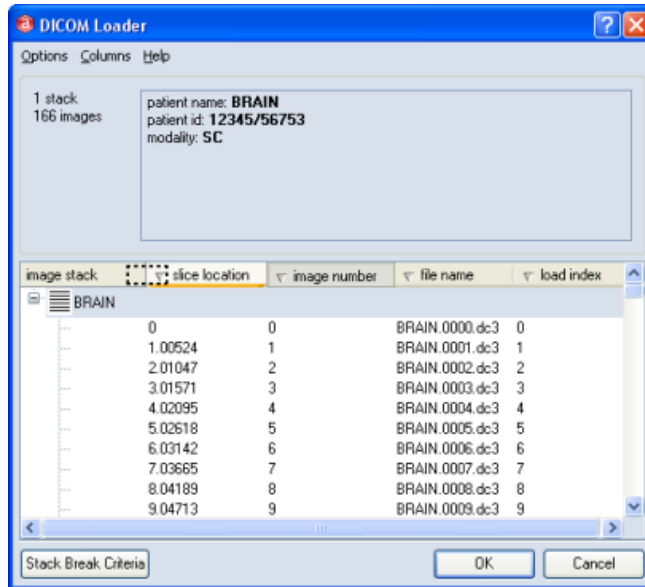


Figure 12.3: Reordering columns in the load dialog.

press the *Stack Break Criteria* button on the DICOM Loader.

The slice location options are defined as follows:

- **ignore:** Do not use the slice location as the criterion to break the stack.
- **constant:** Within a stack, slice location must be constant. In this case, the stack would be, for example, a time stack instead of a volume stack.
- **monotonic:** For two consecutive slices, the slice location must be equal to or greater/smaller than the previous throughout the entire stack. Stacks may have two slices at the same location and slice distance may vary.
- **strictly monotonic:** For two consecutive slices, the slice location must be greater/smaller than the previous throughout the entire stack. No two slices may have the same location, but slice distance may vary.
- **linear:** Slice distance must be equal.

Similar options are available for other numerical stack break criteria.

Load options can also be removed completely by choosing the appropriate column, right clicking on the column's header and selecting *remove column* from the popup menu showing up at this point. Options are disabled when *ignore* is selected from the *Stack Break Criteria* dialog. After any modification of load options, the new settings can be applied to the list of images by either pressing the *Apply* button

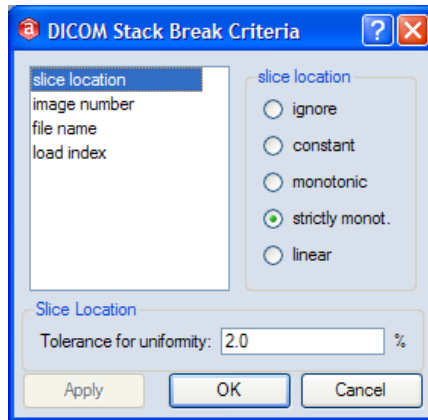


**Figure 12.4:** Dialog for specifying stack break criteria.

or accepting all options by leaving the dialog with *OK*. Any changes are immediately visible through rearrangement of the images and stacks within the list view.

Initially the major stack criteria, like slice location, image number, patient name, patient id, series id and date, as well as the file name and the load index, are shown in different columns. If any of the parameters remain constant for all images (except the slice location), the respective column will not show up. Columns can be manually removed by clicking on the column's header with the right mouse button and choosing *remove column* from the popup menu. New columns can also be added to the list view.

Imagine that you were loading a time series of images showing cardiac motion or the distribution of nuclear tracers. The location could be the same for all images but there might be another parameter of interest describing the order of acquisition. Usually the image number or even the file name will suffice to represent such an order, but there are other parameters that might be of interest, too. Clicking on any image row with the right mouse button shows a popup menu where the menu option *DICOM parameters* will open up a list of all DICOM data elements for the belonging image series.

The parameter list is sorted by the group-element pairs as described in the *DICOM format*. It can be sorted by element description or parameter values as well, for easily finding a certain data element. Any parameter can be appended as stack criterion by pressing the *Add* button. Moving this parameter column to the first position will break sequences of images according to the load option in effect, which can be either *constant* for alphanumeric values or incrementing for numerical values.

If you would like to force all images into one stack of a given sorting order, remove all insignificant columns or set their load options to *ignore*. This will only fail if any of the stringent stack criteria (image size, etc.) varies. If the sorting order conflicts with the order of slice locations, or no slice locations are given at all, you will be prompted for the position of the first image within the list view and the spacing between two adjacent images. Note that this will always result in stacks with uniform

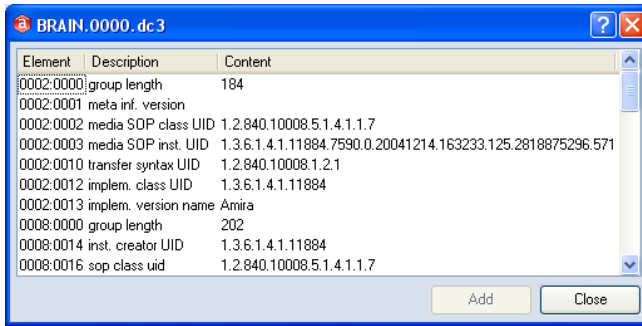


**Figure 12.5:** Dialog listing all DICOM parameters.

image distribution.

## The DICOM Save Dialog

amira is able to export 3D images with uniform or stacked coordinates consisting of either 8-bit or 16-bit values in the DICOM 3.0 format. If the data set to be exported originally has been read from DICOM files, the DICOM attributes (which are stored in the parameter section of a data object) will be exported too. Otherwise, default values for the required attributes will be used. The image dimensions, the voxel size, and the slice location are written correctly in any case.

When writing a 3D image in the DICOM 3.0 format, the DICOM save dialog pops up, allowing you to define certain attributes of the exported files, such as the image modality.

## Chapter 13

# Madymo reader (TNO)

### 13.1 Madymo

MADYMO from TNO Automotive is a software package used to simulate the dynamic behavior of mechanical systems, such as automobile crash situations (multibody and finite elements techniques).

The *MADYMO* kinematics format (.KN3) describes 3D animation of geometric entities such as ellipsoids, planes, etc. It contains static as well as time-dependent information.

After describing the static geometry, the MADYMO file contains a number of time steps, each time step specifying further positions, orientations, etc. *amira* loads the geometry and then displays a dedicated module called *MadymoControl*, which allows the user to select the desired time step. In this way, the evolution of the data over time can be followed. The MADYMO reader creates a data module containing Open Inventor geometry data that can be displayed using an *IvDisplay* module.

This reader currently supports MADYMO version 6.0.

*Note that this reader is available as a separate option for amira.*

### 13.2 Madymo Time-history

Madymo, from TNO Automotive, is a software package used to simulate crash situations (multibody and finite elements techniques).

The *Madymo* Time-history files contain time-history output from Madymo. The Madymo Time-history reader creates a SpreadSheet data module containing a first column "time" and one column for each parameter:channel in the time-history. This data can be displayed using a *TimeHistoryPlot* module.

This reader currently supports Madymo version 6.0.



Note that this reader is available as a separate option for amira.

# Chapter 14

## Mecalog Radioss reader

### 14.1 Radioss

Radioss from Mecalog is finite element analysis software used for, for instance, crash simulation.

The *Radioss* format (.RAD) describes 3D animation of geometric entities and associated scalar or vector data. Simulation results comes as a set of files. The user can use either *Load..* or *Load time series...* in order to load several time steps. This will create a module *TimeSeriesControl* which allows the user to select the desired time step. See *TimeSeriesControl* for more information.

*Usage:*

- [xxxY000 .RAD] - initial output file (optional)
- xxxYnnn .RAD - result output for time step nnn

The reader also requires xxx .RAD to be present in the same directory as the loaded files. This file stores the definitions for parts (/PART/) and possibly subsets (/SUBSET/).

If xxxY000 .RAD is explicitly loaded, a set of initial surface/grid/lines is created without attached fields.

With *Load...* with a single result file, a set of surface/grid/lines is created with attached fields. For lines, node data is directly stored in the HxLine module: dataset 0 stores the node id, datasets 1, ..., *n* correspond to node data in order (see console messages).

With *Load time series...* a unique set of surface/grid/lines is created. Do not select the initial output file xxxY000 .RAD when using the time series option. Otherwise, it is not possible to interpolate between subsequent time steps.

Selecting multiple files with the *Load...* command does the same, except that the initial state read from xxx .RAD is inserted as an initial time step (time 0). In this case xxxY000 .RAD should not be

selected.

The Radioss reader can "group" the data objects read by using "setTightness". This is enabled by Tcl variable `radiossForceTightness` (set by default in `radioss.hx`).

This reader currently supports the Radioss version 4.1k ASCII file format. The following element types are interpreted: SOLID, QUAD, BEAM, SPRING, TRUSS, SHELL, SH3N, ADMAS, CLOAD, RWALL, RWALLCYL, CYLJOINT, RBODY, ACCEL, SECT. Only GRNOD/NODE groups are interpreted.

*Note that this reader is available as a separate option for amira.*

## **Part IV**

# **Alphabetic Index of Data Types**



# Chapter 15

## Amira

### 15.1 AnnaScalarField3

This data class represents a user-defined 3D scalar field based on a regular grid. It provides a port *Expression* by which an arithmetic expression depending on uniform Cartesian coordinates  $x$ ,  $y$ ,  $z$  can be entered that defines the value for each point of the unit cube. The range of values with respect to this (default) domain is indicated as *Component Range*. A data object of type *AnnaScalarField3* has two additional input ports that can be connected to other data objects representing scalar fields on a regular grid, e.g., to image data objects. The predefined variables  $a$  and  $b$  are available for referencing such connected data objects in the arithmetic expression, this way a scalar field depending on other regular scalar fields can be defined. Whenever an expression evaluation is triggered to compute the value for a point  $(x,y,z)$ , the variables  $a$  and/or  $b$  will be substituted by the corresponding input values at the same point  $(x,y,z)$ . For instance the value at a single point may be obtained by attaching a *PointProbe* module to the *AnnaScalarField3* data object and entering the point's coordinates by the *Coord* port of that module.

An expression consists of variables and mathematical and logical operators, the syntax is basically the same as for C expressions. The following variables are always defined:

- **x**: the x-coordinate of the current point
- **y**: the y-coordinate of the current point
- **z**: the z-coordinate of the current point

If data objects *ScalarField A* or *ScalarField B* are connected to port *InputA* resp. *InputB* the following variables are also defined:

- **a**: the values of input object A
- **b**: the values of input object B

The same C style mathematical and logical operators as well as built-in functions that are available for arithmetic expressions can be used here, see *Arithmetic* module description.

## Connections

**InputA** [optional]

Optional scalar field.

**InputB** [optional]

Optional second scalar field.

## Ports

**Expr**



Input field for arithmetic expression defining the scalar field values.

## 15.2 AnnaVectorField3

This data class represents a user-defined 3D vector field based on a regular grid. It provides ports *X*, *Y*, *Z* by which arithmetic expressions can be entered that define the component mappings with respect to uniform Cartesian coordinates *x*, *y*, and *z*. The (default) domain is the unit cube, thus for each point (*x*,*y*,*z*) in it, the associated vector (*X*,*Y*,*Z*) is given by scalar functions *X*(*x*,*y*,*z*), *Y*(*x*,*y*,*z*), and *Z*(*x*,*y*,*z*) which can be specified by the user in the same way as a function for a *HxAnnaScalarField3* data object.

The range of vector magnitudes, defined as Euclidean vectors lengths, is indicated as *Magnitude*.

A data object of type *HxAnnaVectorField2* has three additional input ports named *InputA*, *InputB*, *InputC* that can be connected to other data objects representing scalar or vector fields on a regular grid. There are some predefined variables for referencing such connected data objects in the arithmetic expressions, namely *a*, *b*, *c* for scalar fields and *ax*, *ay*, *az*, *bx*, *by*, *bz*, *cx*, *cy*, *cz* for *x*-, *y*-, resp. *z*-components of vector fields. This way a vector field depending on other regular scalar and/or vector fields can be defined. Whenever an evaluation of the three expressions is triggered to compute the vector associated to a point (*x*,*y*,*z*), each of the variables mentioned that occurs in them will be substituted by the corresponding input values at the same point (*x*,*y*,*z*). For instance the component values of a vector associated to a single point may be obtained by attaching a *PointProbe* module to the *HxVectorField3* data object, entering the point's coordinates by the *Coord* port of that module and setting the *Vector* toggle to *all*.

An expression consists of variables and mathematical and logical operators, the syntax is basically the same as for C expressions. The following variables are always defined:

- **x** - the *x*-coordinate of the current point

- **y** - the y-coordinate of the current point
- **z** - the z-coordinate of the current point

If a scalar data object is connected to any of the ports *InputA*, *InputB*, *InputC* a corresponding variable for access to the data values is also defined, namely:

- **a** - the values of input object A
- **b** - the values of input object B
- **c** - the values of input object C

If a vector data object is connected to any of the ports *InputA*, *InputB*, *InputC* corresponding component variables for access to the data values are also defined, namely:

- **ax**, **ay**, **az** - the xyz vector components of input object A
- **bx**, **by**, **bz** - the xyz vector components of input object B
- **cx**, **cy**, **cz** - the xyz vector components of input object C

The same C style mathematical and logical operators as well as built-in functions that are available for arithmetic expressions can be used here, see *Arithmetic* module description.

## Connections

**InputA** [optional]

Optional scalar or vector field.

**InputB** [optional]

Optional second scalar or vector field.

**InputC** [optional]

Optional third scalar or vector field.

## Ports

**X**

Input field for arithmetic expression defining the x-component field values.

**Y**

Input field for arithmetic expression defining the y-component field values.



## Z



Input field for arithmetic expression defining the z-component field values.

## 15.3 B-Splines

Free-form curves that are defined by their polynomial degree and a sequence of control points. They can be created or modified by the *CurveEditor*.

A curve is displayed by the module *LineSetView*.

## 15.4 CameraRotate

This object can be created from the main window's *Create* menu. It lets you rotate the cameras of all viewers activated in the object's viewer mask. This is useful in order to create simple animations. The animations can be stored in MPEG movie files by attaching a *MovieMaker* module to this object. More complex camera animations can be created using the *CameraPath* object and its *associated editor*

### Connections

#### Time [optional]

Optional connection to some other object providing a time source. This allows you to synchronize multiple time-dependent objects.

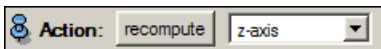
### Ports

#### Time



The current time value. Changing the time modifies the cameras in all viewers activated in the object's viewer mask, i.e., with the orange viewer mask button being set.

#### Action



This port lets you specify the orientation of the camera rotation. Whenever the *recompute* button is pressed or a new menu option is selected the camera path is recomputed, i.e., the center of rotation and the radius are determined by analyzing the camera in the main viewer.

## 15.5 Cluster

Data objects of type *Cluster* are used to represent sets of 3D points with additional data variables associated to them. For each point, at least the x-, y-, and z-coordinates as well as an additional *id* are stored. The *id* is an arbitrary number which can be used to identify corresponding points in different data sets. Note that the *id* is to be distinguished from the *index* of each vertex which refers to the internal consecutive numbering of the vertices.

For each point an arbitrary number of additional data columns may be defined. The elements of a data column may be stored as 32-bit floats, as 32-bit integers, or as 8-bit characters. Each data column also has a string specifying its name. Data columns are consecutively numbered starting with 0. Optionally, a symbol may be defined, which is used to denote the column in an arithmetic filter expression as provided by the modules *ClusterView* and *ClusterDiff*.

Finally, for each data point one or more text labels can be specified. As with data columns, label columns have a name and are consecutively numbered starting with 0. Labels can be displayed using *ClusterLabelView*.

### Commands

`computeBounds`

Computes the bounding box of the cluster.

`computeConnectivity`

Computes connectivity information required to display bonds between neighboring points. Bond detection does not take into account any chemical information. Instead, merely nearest neighbors are computed. Each point may have at most 12 such neighbors. In addition the length of the longest bond of a point may not be larger than 1.4 times the length of the shortest one.

`addPoint <x> <y> <z>`

Adds a new point to the cluster. The arguments are the three coordinates. The new point will be assigned the id 0. The command returns the index of the new point.

`removePoint <index>`

Removes a point from the cluster. The *index* is passed as argument.

`resetIds`

Resets the ids of all points so that they are equal to the respective point indices.

`setId <index> <id>`

Sets the id of the point specified by *index*.

`getId <index>`

Returns the id of the point specified by *index*.

`getIndex <id>`

Returns a list of the indices of all points with the specified id. This method performs a linear search over all points and thus is slow.

## `clear`

Deletes all points and sets the number of data columns to zero.

## `setNumDataColumns <number_of_columns>`

Set the number of data values that are stored for each point. The number of data columns is passed as argument. If the number is less than the number of existing data columns, trailing columns are removed. If the number is greater than the number of existing data columns, empty columns are added.

## `getNumDataColumns`

Returns the number of data columns, i.e., the number of data values per point.

## `setDataValue <column> <index> <value>`

Sets the value of a point in a data column.

## `getDataValue <column> <index>`

Returns the value of a point from a data column.

## `setDataColumnName <column> <name>`

Sets the name of a data column.

## `getDataColumnName <column>`

Returns the name of a data column.

## `setNumLabelColumns <number_of_columns>`

Create and add label columns. The number of label columns is passed as argument. If the number is less than the number of existing label columns, trailing columns are removed. If the number is greater than the number of existing label columns, empty columns are added.

## `getNumLabelColumns`

Get the number of labels that are stored for each point.

## `setLabelValue <column> <index> <string>`

Sets the label of a point in a label column.

## `getLabelValue <column> <index>`

Returns the label of a point from a label column.

## `setLabelColumnName <column> <name>`

Sets the name of a label column.

## `getLabelColumnName <column>`

Returns the name of a label column.

## 15.6 ColorField3

An RGBA color field is a regular 3D field with 4 data components per voxel. Each data component is an 8-byte value. The first 3 components are interpreted as red, green, and blue color values. The fourth component represents an opacity value (alpha). Color fields usually have uniform coordinates, i.e., equal slice distances in all directions, but other coordinate types are possible as well.

Color fields can be visualized using the slicing modules *OrthoSlice* and *ObliqueSlice*. They are especially useful in combination with a *Voltex* module for direct volume rendering. Since the color field stores an RGBA tuple for each voxel, no additional transfer function is required. This allows you for example to visualize different data sets in a single volume rendered image at once. The conversion of one or multiple scalar fields into a color field is accomplished using the *ColorCombine* module.

### Commands

`alphaAverage [min] [max]`

Sets the alpha value of all voxels equal to the luminance  $0.3*R + 0.59*G + 0.1*B$ , i.e., brighter objects become more opaque. If min and max are specified the alpha values are scaled so that they fill this range. By default min and max are 0 and 255, respectively.

`alphaSet <alpha>`

Sets the alpha value of all voxels to the specified value.

`alphaThreshold <luminance>`

Sets the alpha value of all voxels with a luminance smaller than the specified value to 0. The alpha of all other voxels is set to 255. Luminance is computed as  $0.3*R + 0.59*G + 0.1*B$ .

`swapRGBA`

Swaps ABGR tuples into RGBA tuples or vice versa.

## 15.7 Colormap

A *Colormap* is a sequence of RGBA-tuples, where every tuple specifies a color by a red, green and blue value in the *RGB color model*. Each value ranges from 0.0 to 1.0 and is represented by a floating point value. A fourth value, the so-called alpha value, defines *opacity*. It also ranges from 0.0 to 1.0, where 0.0 means that the color is fully transparent, and 1.0 that the color is fully opaque. A colormap usually stores 256 different RGBA-tuples, but other sizes and even procedurally defined colormaps are possible too.

Beside the raw RGBA values, the colormap also stores two *coordinates* defining a range used for color interpolation. Color lookup requests for an argument smaller than the minimum coordinate evaluate to the first colormap entry. Requests for an argument greater than the maximum coordinate evaluate to the last entry.

## Connections

### Datafield

Connection to a data field from which the min-max values of the colormap are taken.

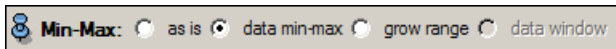
## Ports

### Colormap



This port displays the contents of the colormap. Transparent values are drawn over a checkerboard background. The coordinate range can be edited via the two text fields left and right from the graphics area.

### Min-Max



This port is only visible if an input is connected to port *Datafield*. If this is the case and *data min-max* is selected, the coordinate range of the colormap is automatically adjusted so that it matches the min max values of the connected data field. If *grow range* is selected, the coordinate range is enlarged if the min max values of the connected data field fall outside the current range. The option *data window* becomes active only if the connected data field contains a parameter *DataWindow*. If the option is selected the coordinate range is set to the range specified by the *DataWindow* parameter (see also Section ?? (Parameters)).

### Shift range



This slider allows you to shift the range assumed for a connected data field. Instead of the true min max values, shifted values are used if the value is different from 0.

### Scale range



This slider allows you to scale the range assumed for a connected data field. Instead of the true min max values, scaled values are used if the value is different from 1. The scaling is applied relative to the shifted center of the range. The shifted and scaled range will always be clamped to the original range. The shift and scale ports are useful for investigating a subrange of a data set in detail.

## Commands

Inherits all commands of *Data*.

`setMinMax <min> <max>`

Sets the coordinate range of the colormap.

**minCoord**

Returns the lower bound of the coordinate range.

**maxCoord**

Returns the upper bound of the coordinate range.

**isTransparent**

Checks whether the colormap contains transparent values or not.

**getRGBA <u>**

Returns the interpolated RGBA values for the parameter <u>, which is a value between 0.0 and 1.0. The value 0.0 corresponds to the first colormap entry, while 1.0 corresponds to the last colormap entry.

**getRGB <u>**

Similar to the above command, but returns only three values (RGB, not alpha).

**makeSteps <numsteps>**

Discretizes the colormap so that only *numsteps* different colors remain.

**setInterpolate {0|1}**

Turns interpolation on or off. When interpolation is on, the colors of two neighboring colormap entries are interpolated when a color is looked up. When interpolation is off, the color of the nearest colormap entry is returned. This is useful if a colormap modified with the **makeSteps** command is used.

**getInterpolate**

Checks whether interpolation is on or off.

**makeRandom**

Replaces all colors of the colormap by random values.

**makeVolren <r> <g> <b> <power> <huewidth>**

Replaces the colormap by something useful for volume rendering. The first three arguments specify the base color of the colormap. *power* denotes an exponent used to compute an alpha curve. If this is 1, the colormap goes linearly from fully opaque to fully transparent. *huewidth* indicates the width of the color spectrum around the base color between 0 and 1.

**interpol <map1> <map2> <u>**

Replaces the colormap by the weighted average of two other colormaps <map1> and <map2>. The interpolation parameter <u> should be chosen between 0 and 1.

## 15.8 Data

This is the base class of all **amira** data objects. Data objects are usually represented by green icons in the Pool. In contrast to modules, data objects can be duplicated and saved to a file. They also provide

a hierarchical list of parameters or attributes. This list can be edited using the *parameter editor*.

## Commands

Inherits all commands of *Object*.

### **touch**

Touches the data object, marking it as modified. When the network is fired modules connected to a data object will only be invoked if the data object was modified.

### **duplicate**

Duplicates the data object and returns the name of the duplicated object.

### **save [format filename]**

Saves the data object. If no arguments are specified the command does the same as choosing *Save* from the *File* menu, i.e., it saves the object under the same name as it was saved before. Otherwise, a format and a file name must be specified. The format should be the name of a format as it is displayed in the file dialog's file type menu, e.g., "Amiramesh ascii" or "HxSurface binary". When an object is saved its name is replaced by a possibly modified version of the filename. The command returns the actual name of the object after it has been saved.

### **parameters [options ...]**

Provides access to the data object's parameter list. This command takes several different options allowing you query and set parameters. A list of all folders containing parameters too can be obtained using **list**. The name of each folder is used to access that folder. For example, to get a list of all materials of a surface or of a label field, use **parameters Materials list**.

A parameter value can be set using **setValue <name> <value>**, and it can be returned using **getValue <name>**. For example, to set the color of the material *Exterior* of a surface or of a label field, use **parameters Materials Exterior setValue Color <color>**.

## 15.9 Field3

This class is the base class for all 3D fields in *amira*, e.g., scalar fields, vector fields, or color fields with uniform, stacked, or some other coordinates, or for fields defined on unstructured finite-element grids. This class provides a transparent interface to evaluate the field at any position without needing to know how the field is actually represented. This interface can be accessed via the Tcl command **eval** described below. A field may have an arbitrary number of data variables which can be queried using the Tcl command **nDataVar**. For example, a scalar field has one data variable, while a vector field has three.

## Commands

Inherits all commands of *SpatialData*.

**nDataVar**

Returns the number of data variables of the field.

**eval <x> <y> <z>**

Evaluates the field at the position <x> <y> <z>. On success the command returns as many numbers as there are data variables. The command may fail because the specified position lies outside of the grid the field is defined on. In this case the string **domain error** is returned.

**getRange**

Returns the minimum and maximum data component of the field. For field with more than one data variable this is not the magnitude range of the field.

**primType**

Returns the primitive data type of the field, i.e., the way how the values are represented internally. A number with the following meaning is returned: 0 = bytes, 1 = 16-bit signed integers, 2 = 32-bit signed integers, 3 = 32-bit floating point values, 4 = 64-bit floating point values, 7 = 16-bit unsigned integers.

## 15.10 IvData

This is a simple data object which encapsulates an Open Inventor scene graph. The scene graph can be displayed in *amira* using the module *IvDisplay*. However, it cannot be edited or processed further. Instead, *amira* provides a separate data type *Surface* for representing triangular surfaces with connectivity information and with an optional patch structure. An Open Inventor scene graph can be converted into an *amira* surface object using the module *IvToSurface*.

## 15.11 LabelField3

Data objects of type *LabelField* are used to represent the result of a segmentation applied to a 3D image volume.

A *LabelField* is a regular cubic grid with the same dimensions as the underlying image volume. For each voxel it contains a label indicating the region that the voxel belongs to. Use module *LabelVoxel* to create a *LabelField* from an image data stack. You can manually modify a *LabelField* using *amira*'s image editor *GI*.

In addition to the labels themselves a *LabelField* may also contain *weights* indicating the degree of confidence of the label assignment made for each voxel. Such weights are calculated automatically when you choose option *sub-voxel accuracy* in *LabelVoxel*, when you apply the *smoothing filter* of *GI*, or when you resample a *LabelField* to a smaller resolution using the *Resample* module.

You can visualize a *LabelField* by attaching an *OrthoSlice* module to it. If a *LabelField* contains weights, the port *Primary Array* allows you to choose whether the labels or the probabilities are to be displayed.



## Connections

**Master** [unused]

**ImageData** [required]

Connection to the image data that the segmentation results refer to. You cannot connect this port to an image object with dimensions different from that of the *LabelField*, except the *LabelField* has been newly created via the *Create* menu. In this case, the *LabelField* will be resized so that it matches the dimensions of the image object.

## Ports

**Primary Array**



An option menu which only appears if the *LabelField* contains weights. In this case the menu lets you select whether the labels or the weights are the primary data array. The primary data array is the default array visible to modules expecting an ordinary uniform scalar field like *OrthoSlice* or *Arithmetic*.

## Commands

`hasMaterial <name>`

Returns true if the specified material is defined in the material section of the *LabelField*.

`makeColormap`

Creates a new *colormap* object in *amira*'s Pool, containing the default colors of all materials of the *LabelField*.

`relabel`

Computes new labels so that the materials are numbered in consecutive order starting from 0.

`deleteAltData`

Deletes the weight information if it is present.

## 15.12 LandmarkSet

This data type represents specific points or markers in 3D space. It can be used to flag markers in medical MRI images, or to specify pairs or n-tuples of corresponding points in multiple data sets.

An empty set of landmarks can be created by typing

`create HxLandmarkSet`

into the *amira* console window, cf. Section ???. Individual landmarks can be interactively added, repositioned, or removed from a landmark set by means of the *landmark editor*.

## Commands

**setNumSets** <n>

Sets the number of point sets contained in this data object. Upon creation a landmark set contains one set of points. In order to represent pairs of corresponding points two sets are required.

**getNumSets**

Returns the number of point sets in this data object.

**setPoint** <index> <x> <y> <z> [<set>]

Sets the coordinates of the specified marker <index> in a particular set. If <set> is omitted the first set is used.

**getPoint** <index> [<set>]

Returns the coordinates of the specified marker <index> in a particular set. If <set> is omitted the first set is used.

**setOrientation** <index> <x> <y> <z> <rad> [<set>]

Sets the orientation of the specified marker <index> in a particular set. If <set> is omitted the first set is used. The orientation is specified by an axis plus an angle of rotation around this axis in radians.

**getOrientation** <index> [<set>]

Returns the orientation of the specified marker <index> in a particular set. If <set> is omitted the first set is used. The orientation is returned as an axis plus an angle of rotation around this axis in radians.

**appendLandmark** <x> <y> <z>

Appends a new marker to the data object. The new marker will have the same coordinates in all sets.

**removeLandmark** <index>

Removes the specified marker from all sets, reducing the number of points by one.

**swapSets** [<set1> <set2>]

Exchanges the coordinates of the specified sets. If no arguments are given the first and the second set are swapped, provided both sets exist.

**computeRigidTransform** [<src-set> <dst-set>]

Computes a rigid transformation which move the points of the first set as close as possible onto the points of the second set (the sum of the squared distances between corresponding points is minimized). The result is returned as a 4x4 transformation matrix, which for example can be used to transform some other data object using the **setTransform** command.

**translateCoords** <x> <y> <z> [<set>]

Translate the landmarks of the specified set by the given displacement vector. If <set> is omitted the landmarks in all sets are translated.

```
scaleCoords [-center <x> <y> <z>] <xscale> [<yscale> <zscale>]  
[<set>]
```

Scales the coordinates of the landmarks in the specified set. If `<set>` is omitted the landmarks in all sets are scaled. The optional argument `-center` defines the center of the scaling operation. If no center is specified the origin (0,0,0) will be used.

## 15.13 LargeDiskData

A `LargeDiskData` object is useful for large image data. It allows to extract subvolumes loaded as a usual field object. In this way `amira` can manage data bigger than main memory. Note that only uniform coordinates are supported.

A couple of fileformats might be loaded as `LargeDiskData` (*AmiraMesh*, *Raw Data*, *Stacked-Slices*, *LargeDiskData*).

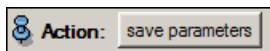
To access the data you have to attach an `Access` module. It provides an interface to load a subblock into `amira`. You can use all the `amira` visualization techniques on this subblock.

Like every other `amira` data object, a `LargeDiskData` object has parameters associated with it. In contrast to most other data types, the `LargeDiskData` cannot be saved directly. But some of the mentioned file formats allow you to save the actual parameters to the file defining the `LargeDiskData`. This is done by pressing the *save parameters* button. It is only visible if you have write access to the file.

The *Save As* menu entry allows export of the data into a common image file format or as raw data. It does not save parameters. This might seem strange at a first glance, but in contrast to all other data formats, the `LargeDiskData` values are not in main memory. The actual data resides only on disk and cannot be saved to another place by `amira`.

### Ports

#### Action



Save parameters to the file defining the `LargeDiskData`.

## 15.14 Lattice3

This class represents regular 3D data arrays. Every node of a regular data array can be addressed by an index tuple (i,j,k). The data array is characterized by its dimensions (the number of nodes in each direction), the primitive data type (e.g., bytes or shorts), the number of data variables per node, and by its coordinates. In `amira` uniform, stacked, rectilinear, and curvilinear coordinates are supported, compare Section ?? (Coordinates and Grids). *Lattice3* is a simple but powerful data type. In particular, all 2D and 3D images in `amira` are represented by this type.

As a technical detail it should be mentioned that in contrast to other data types *Lattice3* is not a data class by itself, i.e., it is not derived from *Data* or *Object*. Instead it is a so-called interface class which is used by other classes such as *RegScalarField3*, *RegVectorField3*, or *RegColorField3*. Usually this fact will not be important for end-users, but only for Developer Pack users.

## Commands

Data objects using this class inherit all commands of *Field3*.

**getDims**

Returns three numbers indicating the number of nodes in each direction of the 3D array.

**coordType**

Returns a number indicating the coordinate type of the lattice, 1 = uniform, 2 = stacked, 3 = rectangular, 7 = curvilinear.

**getValue** <i> <j> <k>

Evaluates the field at the index position <i> <j> <k>. As many numbers are returned as there are data variables in the lattice.

**setValue** <i> <j> <k> <value1> [<value2> ...]

Sets the field values at the index position <i> <j> <k>. The number of values specified by this command must match the number of data variables of the field.

**swapByteOrder**

Swaps the byte order of the lattice's data values from little endian to big endian or vice versa.

**clearSlice** <k>

Sets all values of slice <k> to zero.

**exchangeSlices** <k1> <k2>

Swaps the contents of the slices <k1> and <k2>.

**crop** <imin> <imax> <jmin> <jmax> <kmin> <kmax> [<value>]

Crops the lattice. The first six arguments specify the index bounds of the subvolume to be cropped. It is possible to enlarge the data set by specifying negative lower bounds or upper bounds exceeding the current size of the lattice. In this case the last slice is replicated unless <value> is specified. If this is the case the new slices are initialized with <value>.

**flip** {0|1|2}

Flips the lattice in i-, j-, or k-direction, depending on whether the argument was 0, 1, or 2.

**swapDims** <iIdx> <jIdx> <kIdx>

Performs a kind of rotation about 90 degrees. The arguments tell at which position an index was before, i.e., they must be a permutation of 0, 1, 2. For example, to convert ijk into jki you have to use the arguments 1 2 0.

`setBoundingBox <xmin> <xmax> <ymin> <ymax> <zmin> <zmax>`

Sets the bounding box of the lattice. The bounding box encloses the centers of all voxels of the lattice (not the complete voxels). In case of stacked, rectilinear, and curvilinear coordinates the coordinates of inner points are scaled appropriately.

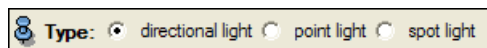
## 15.15 Light

Light objects are used to define additional lights in the *amira* viewer windows. Actually light objects are neither modules nor standard data objects. Nevertheless, they are displayed in the Pool. Like modules they provide some ports allowing the user to adjust the object's properties. In particular, three different light types are supported, namely directional lights, point lights, and spot lights. Lights can be defined in scene coordinates or in camera coordinates (camera slave mode). In order to interactively change the light parameters appropriate Open Inventor draggers can be activated.

Note that the *amira* viewers define separate headlights by default. Light objects represent additional light sources and are not related to the viewer's head light. New lights can be interactively created using the *View Lights* menu of the *amira* main window. This menu also allows to activate new light settings consisting of multiple lights. A light setting is stored as an *amira* script in the subdirectory *share/lights* in the *amira* installation directory. New settings can be added dynamically by copying new scripts into this directory.

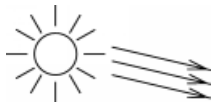
### Ports

#### Type

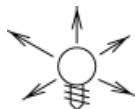


This radio box determines the light type. Three different types are supported:

A *directional light* emits parallel light. It is faster to compute than the other lights. Another advantage is that it has no particular location (although the dragger used to edit the light is located somewhere). Therefore light settings consisting of directional lights only can be easily applied to new scenes, regardless of the actual size or position of the objects in that scene.



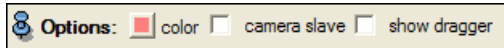
The second type is a *point light*. A point light is specified by its location only. It emits light symmetrically in all directions.



The third type is a *spot light*, which has a position like a point light, but which also defines cone restricting the shape of the light being emitted.



### Options

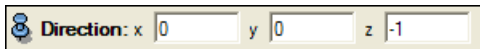


First, this port provides a color button indicating the color of the light. Pressing the button pops up the color dialog and lets you change the light's color.

The next toggle called *camera slave* specifies, whether the light remains fixed relative to the camera. If not, the light's position and direction are fixed with respect to other objects in the scene.

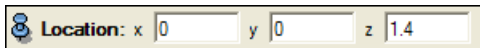
Finally, the last toggle called *show dragger* allows you to activate an Open Inventor dragger which can be used to move the light or to modify its direction.

### Direction



Shows the direction of the emitted light. The values represent the direction in world or camera coordinates depending on the *camera slave* setting. The port is not available for point lights.

### Location



Shows the location of the light source. The values represent the location in world or camera coordinates depending on the *camera slave* setting. The port is not available for directional lights.

### Spot



Here the two additional parameters for spot lights are specified:

The *cut off angle* determines the spread of the cone of the emitted light, measured from one edge of the cone to another.

The *drop off rate* controls how concentrated the light is. The light's intensity is highest in the center of the cone. It's attenuated toward the edges of the cone. A value of 0 produces very sharp edges, A value of 1 produces very soft edges.

## Commands

Inherits all commands of *Object*.

`getColor`

Returns the color of the light as an RGB tuple of floating point number.

**setColor** <color>

Sets the color of the light. The color can be specified either as a tuple of three RGB integer values in the range 0...255, or as a tuple of three RGB floating point values in the range 0...1, or as a text string.

**getIntensity**

Returns the intensity of the light.

**setIntensity** <value>

Sets the intensity of the light. The intensity modulates the light's color. Instead of modifying the light's intensity the brightness of the light's color could be changed as well.

**getDirection**

Returns the direction of the light (undefined for a point light).

**setDirection** <x> <y> <z>

Sets the direction of the light. Has no effect for a point light.

**getLocation**

Returns the location of the light. For a directional light the location of the associated light dragger is returned.

**setLocation** <x> <y> <z>

Sets the location of the light. For a directional light the location of the associated light dragger is set.

## 15.16 LineSet

A *LineSet* data object is able to store independent line segments of variable length. Optionally, for each vertex one or more scalar data items can be stored. *LineSet* objects inherit the vertex set interface, cf. Section ???. In order to visualize the line segments of a *LineSet* object the module *LineSetView* can be used. *LineSets* sets can be stored using the *AmiraMesh file format*.

### Commands

Inherits all commands of *VertexSet*. In particular, the methods `getNumPoints`, `getPoint`, and `setPoint` are inherited.

**setNumPoints** <num>

Sets the number of points of the line set. The coordinates of new points need to be initialized afterwards using `setPoint`. Care must be taken that only existing points are referenced, i.e., that no point index is bigger than  $n-1$ .

**addPoint** <x> <y> <z>

Adds a new point to the line set's vertex array. The new point will not yet be referenced by any line segment. The method returns the index of the new point.

**getNumDataValues**

Returns the number of data values per point.

**setNumDataValues** <num>

Sets the number of data values per vertex. New data values need to be initialized afterwards using **setData**.

**getNumLines**

Returns the number of lines.

**getLineLength** <line>

Returns the number of points of the specified line.

**getLineVertex** <line> <point>

Returns the index of point <point> of line <line>.

**getData** <line> <point> [<set>]

Returns the data value in set <set> of point <point> of line <line>. If <set> is omitted the first data value at that point is returned.

**setData** <line> <point> <value> [<set>]

Sets the data value in set <set> of point <point> of line <line>. If <set> is omitted the first data set is used. Before using this method the number of data sets has to be set using **setNumDataValues**.

**addLine** <p1> [<p2> [<p3> ...]]

Adds a new line consisting of the points <p1>, <p2>, <p3> ... to the line set. The index of the new line is returned.

**deleteLine** <line>

Deletes the specified line without changing the number of points of the line set.

**deleteAllLines**

Deletes all lines of the line set.

**removeLineVertex** <line> <point>

Delete point <point> of line <line>. The method does not remove the point from the global vertex array even if the point is not referenced by any other line.

**addLineVertex** <line> <point> [<pos>]

Adds an additional vertex to line <line>. The second argument <point> is the index of the referenced point. <pos> specifies the position of the new vertex within the line. If this argument is omitted the new vertex will be appended after all other vertices of the line.



`smooth <factor>`

Smooths the lines by replacing the coordinates of each vertex by the weighted average of its neighboring vertices. The bigger `<scale>` the more are the vertices smoothed.

`getRange`

Returns the min and the max of all data values of the line set.

## 15.17 Movie

This data module stores a movie description. The general concept of *amira* movies is described in the manual section of the *MoviePlayer* module.

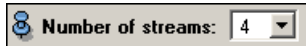
### Connections

#### Master

If this port is connected to a *MoviePlayer* modules result port, this movie can be used as destination movie during a movie conversion process.

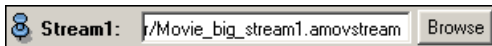
### Ports

#### Number of streams



Select the requested number of streams. After this, the appropriate number of the following file name fields will appear.

#### Stream1



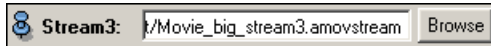
Source specification of the image sequence for stream one. This may be an *amira* movie data file ( `.amovstream` ), a single image file or a sequence of images specified by a wildcard expression. For example specify `c:/mymovie/left*.jpg` to get all JPEG-images from directory `c:/mymovie` starting with "left" in the filename. Possible wildcards are `*` and `?`. `*` matches a sequence of arbitrary characters. `?` matches a single character. If specifying an *amira* movie data file (which is an *amira*-specific file containing a series of images) wildcards are not permitted. For experts: to specify a whole list of wildcard patterns or single image files edit the *amira* movie info file ( `.amov` ).

#### Stream2



Identical to above for stream number 2.

### Stream3

A configuration field for Stream3. It consists of a small icon on the left, followed by the text "Stream3:", then a text box containing the path "/Movie\_big\_stream3.amovstream", and a "Browse" button on the right.

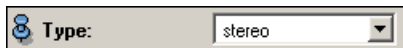
Identical to above for stream number 3.

### Stream4

A configuration field for Stream4. It consists of a small icon on the left, followed by the text "Stream4:", then a text box containing the path "/Movie\_big\_stream4.amovstream", and a "Browse" button on the right.

Identical to above for stream number 4.

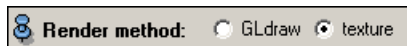
### Type

A configuration field for Type. It consists of a small icon on the left, followed by the text "Type:", then a dropdown menu currently showing the value "stereo".

Select here how the *MoviePlayer* module has to interpret and render the final image sequence.

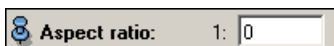
- *mono* - every image forms a single frame.
- *stereo* - two images form a stereo frame.
- *stereo(interlaced)* - every image forms a stereo frame. The left eye channel is stored in the even lines and the right in the odd lines. Internally the module resorts the lines to get an image of the type *stereo(up/down)* .
- *stereo(up/down)* - every image forms a stereo frame. The left eye channel is taken from the upper half of the image and the right from the lower one.
- *stereo(left/right)* - every image forms a stereo frame. The left eye channel is taken from the left half of the image and the right from the right one.

### Render method

A configuration field for Render method. It consists of a small icon on the left, followed by the text "Render method:", then two radio buttons. The first radio button is labeled "GLdraw" and is currently selected. The second radio button is labeled "texture".

This option affects the playback behavior. If set to *GLdraw*, the images are copied directly to the *OpenGL* frame buffer by using the function *glDrawPixels()*. If set to *texture*, the images are first transferred into an *OpenGL* texture object and then rendered as polygons textured with this texture. If an image was compressed using the *OpenGL* texture compression feature by a preceding movie conversion process, this image gets rendered as textured polygon independently of how this port is set. Be warned, that *OpenGL* texture compression is not available for all systems. *SGI* for example seems to implement it less often in it's *OpenGL*. On *PC* systems *OpenGL* texture compression seems to be a standard, due to the limited bandwidth and memory storage. Which render method performs better depends on the individual hardware and software conditions.

### Aspect ratio

A configuration field for Aspect ratio. It consists of a small icon on the left, followed by the text "Aspect ratio:", then a text box containing the value "1: 0".

Force the the aspect ratio of the rendered images to a fixed value. If set to *0* the aspect ratio is taken from the individual image resolutions.

### Swap stereo



Swap left and right images for stereo movies.

### Flip



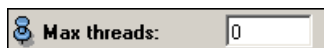
Flip the movie in X- and/or Y-direction.

### Max fps



Limit the playback speed to a maximal value of frames per second. Set this value also if the movie playback looks jerky, that gives the content retrieval more time between the single frames. A value of 0 disables this feature.

### Max threads



On multiprocessor systems, the image retrieval and decompression is performed by default with as many threads as processors are available. That gives much speed but can hamper other users or tasks on this machine. Set this option to a value greater than 0 (which is to disable limitations) to change the number of retrieval threads. Set this to 1 if the movie includes images read by non thread-safe readers.

## 15.18 MultiChannelField3

Multi-channel objects are used to group multiple gray level images of the same size. Display modules such as *OrthoSlice*, *ProjectionView*, or *Voltex* then can be directly connected to the multi-channel object, thus allowing all channels to be operated on simultaneously.

Multi-channel objects are created automatically when reading microscopic image files containing multi-channel information, e.g., Zeiss TIF files or Leica image files. Alternatively, channels can be manually attached to a multi-channel object. Details of how to work with multi-channel objects are described in a separate *tutorial*.

For each scalar field attached to a multi-channel object a special-purpose port is shown, allowing you to define the channel's data window as well as its preferred color. The data window is usually interpreted in such a way that the lower data value is mapped to black while the upper is mapped to the channel's preferred color.

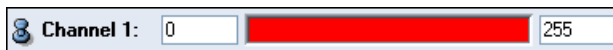
## Connections

**Channel 1** [required] Used to attach the first scalar field to the multi-channel object. This input determines the dimensions and the data type of the multi-channel object.

**Channel 2** [optional] Used to attach a second scalar field to the multi-channel object. The second input must have the same dimensions and the same data type as the first one. After a second input has been connected, a new input called *Channel 3* will be created, and so on. In this way an arbitrary number of channels can be connected.

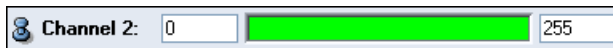
## Ports

### Channel 1



Determines the preferred data window of a channel as well as its colormap. For constant colormap, if the lower bound of the data window is set to 100, voxels with values less than or equal to 100 will be drawn in black by the slicing modules *OrthoSlice* and *ObliqueSlice*.

### Channel 2



Specifies the settings of the second channel.

## 15.19 Object

All amira objects represented by icons in the Pool are derived from this base class. The class provides some basic Tcl commands allowing to select or deselect an object, or to show or hide its icons. The class is not of interest for end users, but only for script programmers and developers.

## Commands

**hasInterface** <typename>

Checks if the object provides an interface matching the specified type. For more information about interfaces please refer to the *amira Programmer's Guide*.

**showIcon**

Makes the object's icon visible.

**hideIcon**

Hides the object's icon. Although the object is no longer displayed the object itself is still contained in the Pool.

### **iconVisible**

Checks if the object's icon is visible or not.

### **select**

Selects the object, so that the ports are shown in the Properties Area.

### **deselect**

Deselects the object, hiding the ports in the Properties Area.

### **setLabel <name>**

Renames the object. If another object with the same name already exists, the specified name is modified so that it becomes unique. In any case, the new name of the object is returned.

### **fire**

Updates the object and all downstream objects.

### **compute**

Updates the object by calling its update and compute methods. In contrast to **fire** downstream objects are not updated.

### **allPorts**

Returns a list of all *ports* of an object.

### **connectionPorts**

Returns a list of all *connection ports* of an object.

### **downStreamConnections**

Returns a list of all objects connected to this object. For each object also the name of the corresponding connection port is reported. That is, each element of the returned list in turn is a list containing the the name of the connected object and the name of the connection port.

### **setIconPosition <x> <y>**

Sets the position of the object's icon in the Pool.

### **getIconPosition**

Returns the position of the object's icon in the Pool.

### **clipGeom <PlaneModule>**

Causes all geometry display of the object to be clipped by the plane defined by **<PlaneModule>**. For example, a plane module is any module derived from *Arbitrary Cut*. The geometry of an object might be clipped by up to six clipping planes. Also see **unclipGeom** below.

### **unclipGeom <PlaneModule>**

Undos the effect of the **clipGeom** command described above.

### **destroy**

The object is removed, as well as certain dependent objects.

**getTypeId**

Returns the type name of the object.

**help**

Displays all commands specific to that object.

**setLabel <name>**

Changes the name of the object to <name>. If already some other object with the same name exists, <name> will be automatically modified.

**setViewerMask <mask>**

This command is used to show a possible 3D output of the object in certain viewer windows and to hide it in other viewers. The bits in <mask> controls the viewers, e.g., a mask value of 2 shows the output in viewer 1 and hides it in viewer 0.

## 15.20 ScriptObject

*Note: If you have worked with script objects prior to amira version 3.0, please read the compatibility notes at the end of this file.*

amira is fully scriptable via its built-in Tcl interface (see Section ?? (Scripting)). The *ScriptObject* module allows the user or a custom solution provider to create scripts that fit seamlessly into the amira user interface, and define their own user interface components.

A script object is an object showing up in the Pool similar to an *OrthoSlice* or an *Axis* module. It can have ports, like sliders or buttons. The ports are defined by Tcl code and the reaction to a change of the ports is also implemented in Tcl. The Tcl code consists of a portion for initialization and a Tcl procedure that is called whenever the script has to react to changes of input parameters, the *compute* procedure.

Any amira script can be turned into a script object by putting a special header line into the script. Write an amira script (using your favorite text editor) and put a special header for script objects in the first line:

```
# Amira-Script-Object V3.0
echo "Hello world, a script is called."
```

Load this file into amira. A blue icon appears. Each time you click on the *Restart* button, the script will be read and executed and the above message appears in the console window.

During the execution of a script object, the global variable *\$this* always contains the name of the currently active script object. This allows to easily access object-specific commands, the so-called *methods*. Declaring a method is very similar to declaring an ordinary Tcl procedure:

```
$this proc name args body
```

Just like the Tcl command `proc`, you can declare a method by using `$this proc`. Methods can be executed by calling `$this name args`. The syntax is completely analogous to global Tcl procedures (see Section ?? (Introduction to Tcl)). The special point about a method is that inside the method the `$this` variable is set appropriately. Example:

```
# Amira-Script-Object V3.0
$this proc sayHello {} {
    echo "module $this is greeting you"
}
```

If you load this script object, nothing will happen visibly. However, if your script object is called `MyScript.scro`, you can type

```
MyScript.scro sayHello
```

in the `amira` console window, and you will get a personalized greeting line as the result.

There are several methods with a special meaning:

- `$this proc constructor {} {...}` defines a method that is called when the script object is created. This is used for creating user interface elements and initializing the object.
- `$this proc destructor {} {...}` defines a method that is executed when the object is deleted or restarted. Used for cleaning up or terminating communications.
- `$this proc compute {} {...}` defines a method that is called whenever a user interface component of the script object is changed by the user. See examples below.

Here is an example that uses the `constructor` and `compute` methods in order to define a simple user interface and to query the current state of that user interface:

```
# Amira-Script-Object V3.0

$this proc constructor {} {
    $this newPortIntSlider myValue
    $this myValue setLabel "Value:"
}

$this proc compute {} {
    set val [$this myValue getValue]
    echo "The value is $val"
}
```

In the example, the constructor creates a new port, an integer slider which will appear in the user interface. The port has the internal name `myValue` and its visible label is set to `Value`. Whenever the user modifies the value of the slider, the `compute` method is called, and outputs the current port value.

In addition to defining methods, a script object also allows to define member variables. Analogous to Tcl variables, a member variable is a placeholder for a certain value, but a member is local to each script object. If you have two script objects A and B, both can have a member variable `x`, and the values of these two variables is kept separately. In order to define and query member variables, use the commands `$this setVar` and `$this getVar` (see below).

You can save amira networks containing script objects. When loading the saved network into amira, the following things will happen:

- the script object is created
- the saved value of all member variables is restored
- the `constructor` method is called
- the `compute` method is called
- the value of all ports is restored
- the `compute` method is called again

## Connections

**Data** [optional]

Can be connected to any data object. Can be used by the script.

## Ports

**Script**



This port is available for any script object. The *Restart* button deletes all dynamically created ports, sets the `isFirstCall` flag to 1 and calls the script. The text field indicates the location of the script file.

## Commands

`newPortButtonList <name> <number-of-buttons>`

Creates a new *button list* port.

`newPortButtonMenu <name> <number-of-buttons> <number-of-options>`

Creates a new *button menu* port.

`newPortColormap <name>`

Creates a new *colormap* port.

`newPortDoIt <name>`

Creates a new *DoIt* port.



**Note:** Starting with amira 4.0, a green *Apply* button is displayed by default, rather than the usual *DoIt* port. See the *DoIt* port documentation for further details.

`newPortFilename <name>`

Creates a new *fi lename* port.

`newPortFloatSlider <name>`

Creates a new *fbat slider* port.

`newPortFloatTextN <name> <number-of-fields>`

Creates a new *fbat text* port.

`newPortMultiMenu <name> <num-options-1> [<num-options-2> ...]`

Creates a new *multi menu* port.

`newPortInfo <name>`

Creates a new *info* port.

`newPortIntSlider <name>`

Creates a new *integer slider* port.

`newPortIntTextN <name> <number-of-fields>`

Creates a new *integer text* port.

`newPortRadioBox <name> <number-of-toggles>`

Creates a new *radio box* port.

`newPortSeparator <name>`

Creates a new *separator* port.

`newPortText <name>`

Creates a new *text* port.

`newPortTime <name>`

Creates a new *time* port.

`newPortToggleList <name> <number-of-toggles>`

Creates a new *toggle list* port.

`newPortConnection <name> <type-name>`

Creates a new connection port. The type name specifies what type of objects can be connected to the port. The type name of an existing object can be obtained using the Tcl command `getTypeId`.

`deletePort <name-of-port>`

Deletes a port which has been created using one of the *newPort* commands.

`proc name args body`

Define a Tcl member procedure (see above). The syntax is analogous to the global Tcl `proc` command. This command is not specific to the *ScriptObject*, but it is available in all *amira* objects.

**setVar** <variable> <value>

Variables stored in this way keep their values between successive calls of the **compute** procedure. Ordinary Tcl variables get lost. This command is not specific to the ScriptObject, but it is available in all **amira** objects.

**getVar** <variable>

Returns the value of a variable set using **setVar**. This command is not specific to the ScriptObject, but it is available in all **amira** objects.

**testBreak**

This commands checks if the stop button has been pressed. If so the execution of the script is automatically terminated. Use this command inside long animation loops or similar constructs.

In addition to the script object extensions, each script objects inherits a number of methods from the *general amira object* type.

*Compatibility Note: The semantics of script objects has slightly changed with amira 3.0 compared to older amira versions. If the first line of the script contains the line "# Amira-Script-Object V0.1", the old behavior is enforced.*

## 15.21 SpatialData

In **amira** all data objects embedded in 3D space are derived from this class. Every spatial data object provides a 3D bounding box as well as an optional transformation matrix. The transformation matrix allows the user to translate, rotate, or scale the object and the geometry of any display modules attached to it. Transformations can be defined interactively using the *Transform Editor*, or or from a script using the Tcl commands described below.

### Commands

Inherits all commands of *Data*.

**getBoundingBox**

Returns the bounding box of the data object. The bounding box consists of 6 values denoting the *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, and *zmax* coordinates in that order. For data objects defined by a set of discrete points like *point clusters*, *surfaces*, tetrahedral or hexahedral grids, the bounding box is the smallest box containing all points. For 3D images it is the smallest box containing all voxel centers, but not all voxels as is.

**getTransform** [-d]

Returns the transformation matrix of the data object. The transformation matrix is a 4x4 matrix which can be applied to a 3D vector in homogeneous coordinates. It encodes a translation, rotation, and scaling operation. If the *-d* option is specified, the transformation matrix is returned in *decomposed* form, i.e., the translation, rotation, and scaling operations are separated.

**setTransform** [**<a11>** **<a12>** ... **<a44>**]

Sets the transformation matrix of the data object. If no arguments are given the transformation is reset to the identity matrix.

**getInverseTransform**

Returns the inverse transformation of the data object as a 4x4 matrix.

**getTranslation**

Returns the translation part of the decomposed transformation matrix of the object.

**setTranslation** [**<x>** **<y>** **<z>**]

Sets the translation part of the decomposed transformation matrix of the object. If no arguments are given the translation part is reset to zero.

**getRotation**

Returns the rotation part of the decomposed transformation matrix of the object. Four number are returned. The first three number denote the axis of rotation. The fourth number denotes the angle of rotation in degrees (0...360).

**setRotation** [**-center** **<x>** **<y>** **<z>**] **<x>** **<y>** **<z>** **<degrees>**

Sets the rotation part of the decomposed transformation matrix of the object. The rotation is specified by a rotation axis and an angle of rotation. The optional argument **center** can be used to specify the center of rotation.

**getScaleFactor**

Returns the scaling part of the decomposed transformation matrix of the object. Three number are returned, denoting the scaling in x-, y-, and z-direction.

**setScaleFactor** [**<x>** **<y>** **<z>**]

Sets the scaling part of the decomposed transformation matrix of the object. If no arguments are given the scaling part is reset to unity.

**translate** [**-l|-w**] **<x>** **<y>** **<z>**

Translates the object by modifying its transformation matrix. The optional argument **-l** indicates that the translation is applied in local coordinates (after the existing transformation). This is the default. The optional argument **-w** indicates that the translation is applied in world coordinates (before the existing transformation).

**rotate** [**-lx|ly|-lz|-wx|-wy|-wz**] [**-l|-w**] **<x>** **<y>** **<z>**] **<degrees>**

Rotates the object by modifying its transformation matrix. The object can be rotated around the local x-, y-, or z-axis or around the world x-, y-, or z-axis (as indicated by the arguments **-lx** to **-wza**). Alternatively, the object can be rotated around a user-specified axis in either local or world coordinates. **degrees** specifies the angle of rotation in degrees (0...360).

**scale** [**-l|-w**] **<x>** **<y>** **<z>**

Scales the object by modifying its transformation matrix. The optional argument **-l** indicates that

the scaling is applied in local coordinates (after the existing transformation). This is the default. The optional argument `-w` indicates that the scaling is applied in world coordinates (before the existing transformation).

`multTransform [-l|-r] <a11> <a12> ... <a44>`

Multiplies the current transformation matrix with the specified matrix. The arguments `-l` and `-r` indicate whether the matrix should be multiplied from left or from right. Multiplication from left means that the matrix is applied to the objects local coordinates.

## 15.22 SpreadSheet

This data type represents a spreadsheet. A spreadsheet will be created e.g., by the module *TissueStatistics*.

## 15.23 Surface

In amira data objects of type *Surface* are used to represent non-manifold triangulated surfaces. Such surfaces are required as an intermediate step in generating a tetrahedral patient model from the results of segmentation of a 3D image stack.

Surfaces mainly consist of a list of triangles as well as a list of 3D coordinates. Each triangle is defined by three indices pointing into the list of coordinates. Moreover, triangles are grouped into so-called *patches*. Conceptually, a patch describes the boundary between two adjacent regions (tissue types). These two regions, called *inner region* and *outer region*, are represented by indices into the surface's *material list*. Although required for grid generation, the patch structure of a surface does not necessarily define a valid space partitioning. However, any surface must have at least one patch.

Surfaces may also contain additional data, such as edges, boundary contours, or connectivity information. Because these data can be computed online they are usually not written into a file. If the data are not already present but a certain module requests them, they are recomputed automatically. You may notice a small time delay in this case. Recomputation of the connectivity information can be enforced by the Tcl command `recompute`.

Surfaces may additionally contain *level-of-detail* information, which can be generated using the *Simplification Editor*. If such data is present, the surface receives a port *Level of Detail*, where the desired level can be set.

You may use the *SurfaceGen module* to extract boundary surfaces from a *LabelField* describing the results of segmentation. You can visualize surfaces by attaching a *SurfaceView* module to it.

There are two editors that can be applied to modify surfaces: the *Surface Simplification Editor* and the *Surface Editor*. Use the former once to reduce the number of triangles contained in the surfaces. The latter allows you to perform an intersection test and to modify the surfaces manually.

## Commands

### `recompute`

Recomputes any additional data such as edges, boundary contours, or connectivity information from scratch. If the surface contained patches consisting of unconnected groups of triangles these patches are automatically subdivided into new patches consisting of connected triangles only.

### `fixOrientation [patch]`

Checks if all triangles of a given patch are oriented in the same way. If this is not the case some triangles will be inverted in order to fix the orientation. If no patch number is specified all patches of the surface will be processed in this way.

### `invertOrientation`

Inverts all triangles of the surface.

### `makeOnePatch`

Puts all triangles of the surface into a single patch.

### `cleanup`

Removes any additional data such as edges, boundary contours, or connectivity information from the surface.

### `getArea <i>`

Compute area of all surface patches incident on material *i*.

### `getVolume <i>`

Compute volume of material *i*.

### `setColor <material> <color>`

Defines the color of a material used in module *SurfaceView*. The material may be specified by either a material name or by a material index. The color may be specified by either an RGB triple in range 0...1 or by a common X11 color name, e.g., *red* or *blue*.

### `setTransparency <material> <t>`

Defines the transparency of a material used in module *SurfaceView* when draw style is set to transparent. The material may be specified by either a material name or by a material index. The transparency value *t* must be a floating point number in range 0...1.

### `add -point <x> <y> <z>`

Adds a new point to the surface. The method returns the index of the new point.

### `add -triangle <p1> <p2> <p3>`

Adds a new triangle to the surface and returns its index. The triangle will be inserted into the first patch of the surface. If no patch yet exists one will be created.

### `refine`

Refines the surface by subdividing all edges. After this operation the surface will contain four times the number of triangles.

## 15.24 TetraGrid

A data object of type *TetraGrid* represents an unstructured finite-element grid composed of tetrahedra. The geometric information is stored in terms of vertices, edges, faces, and tetrahedra. For instance such data objects are useful as patient models. Like a *LabelField* with its uniform hexahedral grid structure a tetrahedral grid also contains a 'dictionary' of different material types or regions. In addition to the material names the dictionary may contain colors and other parameters related to material properties.

*amira* is able to reconstruct tetrahedral grids from 3D image data. This procedure involves several steps, including image segmentation, extraction of boundary faces, surface simplification, and finally grid generation. The tutorial in Section ?? of the user's guide illustrated this process in more detail. The actual grid generation step is performed by the computational module *TetraGen*. The quality of a tetrahedral grid may be improved by applying certain operations provided by the *Grid Editor*.

### Commands

`hasMaterial <name>`

Returns true if the specified material is defined in the material section of the *TetraGrid*.

`hasDuplicatedNodes`

Returns the number of duplicated nodes, i.e., nodes with exact identical coordinates. Such nodes may be used in order to represent discontinuous piecewise linear fields.

`removeDuplicatedPoints`

Removes all duplicated points from the grid. No field object must be connected to the grid.

`add <othergrid>`

Copies all vertices and tetrahedra from an other tetrahedral grid into this one.

`removeTetra <n>`

Marks the tetrahedral cell specified by <n> as obsolete.

`cleanUp`

Removes all obsolete tetrahedra from the grid.

`fixOrientation`

Fixes the orientation of all tetrahedra so that the enclosed volume is positive.

## 15.25 Time

Modules such as *Time Series Control* or other data objects dealing with time-dependent data provide a time port, i.e., a special slider which also can be animated. Multiple such modules can be synchronized by connecting them to one global *Time* object. The time object provides the same functionality as the time port in the modules themselves. In fact, the time value can be changed in both the time port or

in an upstream time object. A time object can be created by choosing *Time* from the main window's *Create* menu. Alternatively, it can be created by choosing *Create time* from the popup menu of a time port (press the right mouse button over a time slider in order to activate this menu).

## Connections

**Time** [optional]

Connection to an upstream time object. Usually there will only be one instance of a time object and this port will not be connected.

## Ports

**Time**



This slider specifies the current time value. Additional settings can be modified via a popup menu which is activated by pressing the right mouse button over the slider. The inner buttons proceed one step in backward or forward direction, respectively. The outer buttons activate animation mode. The popup menu lets you choose between simple animation (play once) and two endless modes (loop and swing). Animation is always restricted to a subrange of the whole time interval. The subrange can be controlled graphically via the two upper arrow buttons. All settings can also be adjusted in a configure dialog which can be activated via the popup menu, too.

## 15.26 VertexSet

The *HxVertexSet* class is an abstract base class. It is derived by many other amira data objects containing a list of 3D vertices, for example *landmark sets*, *surfaces*, or *tetrahedral grids*. *HxVertexSet* provides an interface allowing other modules to access the vertices in a transparent way.

In order to visualize the vertices of a vertex set you may use the *VertexView* module.

## Commands

**applyTransform**

This command changes the coordinates of the vertices of the data object according to the object's current transformation matrix. This matrix can be defined using the *Transform Editor*. After the vertices have been transformed the transformation matrix is reset to the identity.

This command is useful in order to make transformations permanent. In particular, it should be issued before a transformed data object is written to a file. Otherwise, the transformation will be ignored by most file formats.

**translate <dx> <dy> <dz>**

Translates all vertices by a constant amount.

`scale {<f> | <fx> <fy> <fz>}`

Scales all vertices by a common factor. If three arguments are specified the x-, y-, and z-coordinates are scaled by different factors.

`jitter {<d> | <dx> <dy> <dz>}`

The coordinates of the vertices are jittered randomly. The arguments indicate the maximal amount of jitter. Each coordinate of a vertex will be changed by at most  $\pm d/2$ . The method is useful in order to resolve problems due to degenerate configurations in certain geometric algorithms.

`getNumPoints`

Returns the number of vertices of the data object.

`getPoint <n>`

Returns the coordinates of the specified vertex.

`setPoint <n> <x> <y> <z>`

Sets the coordinates of the specified vertex.





# Chapter 16

## Molecular Pack

### 16.1 MolSurface

The data class *MolSurface* represents molecular surfaces in *amira*, generated by the computational modules *CompMolSurface* and *CompMolInterface*. The class *MolSurface* is derived from the class *Surface* and hence, everything that can be done with an object of type *Surface* can also be done with an object of type *MolSurface*. Additional information stored in class *MolSurface* includes:

- the molecular surface type, i.e., van der Waals, solvent accessible, or solvent excluded surface,
- the solvent probe radius,
- for each molecule that contributed to the surface, its number of atoms,
- and for each point or triangle, the index of the atom it belongs to.

The entire functionality of the editors provided for objects of type *Surface* is available, including simplification. However, some care needs to be taken. If you modify the number of points or triangles, the module *MolSurfaceView* will no longer be able to color the surface according to atom attributes. In contrast, other operations, such as edge flipping to improve the surface quality, will not affect the representation.

### Connections

**Master** [optional]

Connection to the module that generated the molecular surface.

### Commands

For a description of Tcl commands, see the documentation of the *Surface* data class in the *amira* User's Guide.

## 16.2 MolTrajectory

An object of type *MolTrajectory* represents a series of molecular configurations. A trajectory can be created by loading a file containing a trajectory, or it can be attached to an object of type *MolTrajectoryBundle*, extracting one of the trajectories contained in that bundle.

### Connections

#### Bundle [optional]

The trajectory may be connected to a *trajectory bundle*. If so, the *Trajectory* port will appear in the user interface of the data object which enables you to control which member of the bundle this trajectory represents.

### Ports

#### Trajectory



The *Trajectory* port is visible only if the connection port *Bundle* is connected to a trajectory bundle. The menu lets you select one of the trajectories contained in the bundle.

### Commands

#### getTrajectoryName

Returns the name of the trajectory.

#### getNumTimeSteps

Returns the number of time steps in the trajectory.

#### getGlobalWeight

Returns the global weight of the trajectory, which is the probability of the molecule to be in the metastable conformation (conformational ensemble) represented by the trajectory.

#### setGlobalWeight

Sets the global weight.

#### setIntObservable <name> <value> <timeStep>

Overwrites the 'value' of observable 'name' for time step 'timeStep'. If 'timeStep' is omitted or -1, a constant observable over all time steps is assumed and the values for all time steps are overwritten, so some care is needed. This command is for integer observables.

#### setFloatObservable <name> <value> <timeStep>

Same as above but for float observables.

**getObservableNames**

Returns the names of all known observables.

**getObservable** <name> <timeStep>

Returns the value of observable 'name' for time step 'timeStep'. If 'timeStep' is omitted, the value of the first time step is returned.

## 16.3 Molecule

An object of data type *Molecule* contains information about the structure of the molecule. Typical information that is stored are the types and positions of the molecule's atoms and the bonds between the atoms, plus the type of each bond. Furthermore, the data object stores information about groups of atoms in a hierarchical way. For example a functional group consists of a number of atoms, several functional groups may form a residue, and a couple of residues may form a secondary structure. This allows quick traversal of the molecule's structure.

Molecules can be visualized with the following viewing modules: *MoleculeView*, *BondAngleView*, *SecStructureView*, or *TubeView*. In addition, there are two modules for generating molecular surfaces. The *CompMolSurface* module enables you to generate the *solvent accessible*, *solvent excluded*, and *van der Waals surfaces* of a molecule. The *CompMolInterface* module can be used to generate intra- and intermolecular interfaces, such as between single atoms or residues, or between two molecules, respectively.

When comparing the structures of several molecules with each other, one is faced with the problem of aligning molecules to each other. This can be easily done by connecting the *AlignMaster* connection port to a second molecule which will serve as reference. There are several alignment modes. If the molecules have the same number of atoms you can align the two molecules to each other by using all atoms, whereby the *i*'th atom of the first molecule corresponds to the *i*'th atom of the second molecule. You can also select atoms in either the *slave* or the *master* molecule. The *slave* is the molecule to be aligned. If the molecules have different numbers of atoms, the only way to align molecules is to select atoms in both molecules.

### Connections

**Data** [optional]

The molecule may be connected to a *molecular dynamics trajectory*. If so, a *Time* port will appear in the *Properties Area* which enables you to load new time steps into the molecule.

**Time** [optional]

Sometimes it is necessary to synchronize two or more time-dependent data objects, such as a molecule from a trajectory and the corresponding electrostatic field. In this case, the *Time* port can be connected to a *Time* object.

**AlignMaster** [optional]

## PrecomputedAlignment [optional]

You find the description of the two ports above are described in the section on *alignment of molecules*.

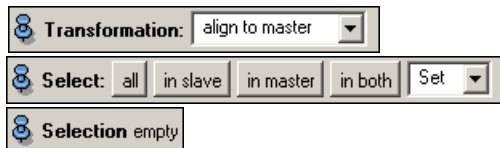
## Ports

### Time



The *Time* port is visible only if the molecule object is time dependent, i.e., it is connected to a molecular dynamics trajectory. In the slider you can select a specific time step. If you wish to step through the trajectory, use the buttons next to the slider. For continuous animation, use the outer buttons.

### Alignment



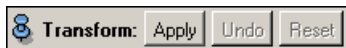
You will find the description of the above three ports in the section on *alignment of molecules*.

### Selection Browser



Press this button if you wish to open the selection browser for this molecule.

### Transform



If you want to transform all atom coordinates of the molecule by the current transformation in order to save the molecule with the transformed coordinates to a file, press the *Apply* button. As long as the *Transform Editor* is active you can also undo your actions.

## Commands

You can use the *console window* for entering additional commands for molecules. The easiest way to use the console for a molecule is to click on the molecule object in the Pool and then press the TAB key in the console window to display the name of the selected object. After the name you can use the following selection commands:

`list`

Lists all levels defined for the molecule.

**list** <levelName>

Lists all groups of level *levelName*.

**list** <levelName/[groupName|groupRange]>

Prints information about a single group or multiple groups specified by a group range.

**define** <levelName/groupName> <groups>

This command defines a new group *groupName* in the level *levelName*. If *levelName* does not yet exist, it will be added as a new level. *groups* is a list of groups of possibly different levels. Groups can be specified using ranges.

Example: The following command defines group *one* in level *test* consisting of atoms 1 and 3 and residues 4 to 6.

- **define** test/one atoms/1 atoms/3 residues/4-6

**defregex** <levelName/groupName> <atomExpr>

Define a new group by using *atom expressions*. The new group will only contain atoms, i.e., no higher level groups.

**sel** <atomExpr>

Lets you add atoms specified by *atomExpr* to the current selection.

Example: The following command will select all carbon atoms which are located in helices.

- **sel** a=C AND s/type=helix

**desel** <atomExpr>

Lets you remove atoms from the selection.

**applyTransform**

The current transform is applied to the molecule, i.e., the original atomic positions are replaced by the transformed positions.

**Editing Commands:** The following commands allow to access and edit all data entries inside of the molecular data structure. The commands work index bases. All indices must be valid. Note that removing groups or attributes may invalidate locally stored indices.

**getLevelIx** <levelName>

Returns index of level with name ;levelName;, -1 if no such level.

**getLevelName** <levelIx>

Returns name of level with index ;levelIx;.

**getNumLevels**

Returns number of levels.

**addLevel** <levelName> <referenceLevelIx>

Adds level of name ;levelName; which references the level with index ;referenceLevelIx;. Returns index of new level

**removeLevel** <levelIx>

Removes level with index levelIx. All dependant levels will also be removed. Level indices might change after using this command. No return value.

**getReferenceLevelIx** <levelIx>

Returns the index of the referenced level of level ;levelIx;.

**getNumGroups** <levelIx>

Returns number of groups of level with index ;levelIx;.

**getNumGroupElements** <levelIx> <groupIx>

Returns number of elements of group ;groupIx; of level ;levelIx;.

**getGroupElement** <levelIx> <groupIx> <elementIx>

Returns index of ;elementIx;-th group which is referenced by group ;groupIx; of level ;levelIx;.

**getGroupElements** <levelIx> <groupIx>

Returns list which contains indices of groups which are referenced by group ;groupIx; of level ;levelIx;.

**removeGroupElement** <levelIx> <groupIx> <elementIx>

Removes ;elementIx;-th reference of group ;groupIx; of level ;levelIx;. Element indices of the group will change after using this command. No return value.

**setGroupElement** <levelIx> <groupIx> <elementIx> <refGroupIx>

Sets the reference of the ;elementIx;-th element of the group ;groupIx; of level ;levelIx; to the group ;refGroupIx;. No return value.

**addGroupElement** <levelIx> <groupIx> <refGroupIx>

Add the reference ;refGroupIx; at the end of the of the elements of the group ;groupIx; of level ;levelIx;. No return value.

**removeGroup** <levelIx> <groupIx>

Remove the group ;groupIx; from index ;levelIx;. Will also remove all dependant groups and attribute entries. No return value.

**addGroup** <levelIx>

Adds group to level ;levelIx;. Returns index of new group.

**addGroupCS** <levelIx>

Adds group to level ;levelIx; and sets its group elements to the currently selected atoms. Returns index of new group.

**getNumAttributes** <levelIx>

Return number of attributes of level ;levelIx<sub>i</sub>.

**getAttributeIx** <levelIx> <attrName>

Return index of attribute ;attrName<sub>i</sub> of level ;levelIx<sub>i</sub>.

**getAttributeName** <levelIx> <attrIx>

Return name of attribute ;attrIx<sub>i</sub> of level ;levelIx<sub>i</sub>.

**addStringAttribute** <levelIx> <attrName>

Adds a string attribute of name ;attrName<sub>i</sub> to level ;levelIx<sub>i</sub>. Returns the index of new attribute. Returns -1 if not successful.

**addIntegerAttribute** <levelIx> <attrName>

Adds an integer attribute of name ;attrName<sub>i</sub> to level ;levelIx<sub>i</sub>. Returns the index of new attribute. Returns -1 if not successful.

**addFloatAttribute** <levelIx> <attrName>

Adds a float attribute of name ;attrName<sub>i</sub> to level ;levelIx<sub>i</sub>. Returns the index of new attribute. Returns -1 if not successful.

**removeAttribute** <levelIx> <attrIx>

Remove attribute ;attrIx<sub>i</sub> from level ;levelIx<sub>i</sub>. No return value.

**setAttributeName** <levelIx> <attrIx> <attrName>

Sets name of attribute ;attrIx<sub>i</sub> of level ;levelIx<sub>i</sub> to ;attrName<sub>i</sub>. No return value.

**getAttributeValue** <levelIx> <attrIx> <groupIx>

Returns value of attribute ;attrIx<sub>i</sub> of level ;levelIx<sub>i</sub> for the group ;groupIx<sub>i</sub>.

**setAttributeValue** <levelIx> <attrIx> <groupIx> <value>

Sets value of attribute ;attrIx<sub>i</sub> of level ;levelIx<sub>i</sub> for the group ;groupIx<sub>i</sub> to ;value<sub>i</sub>.

**addSet** <name>

Adds current selection as a new group to the 'sets' level. The name attribute of the new group will have the value ;name<sub>i</sub>.

**removeSet** <name>

Remove set of name ;name<sub>i</sub> from 'sets' level. No return value.

**useSet** <name>

Resets selection to atoms contained in the group in 'sets' level which has the name attribute value ;name<sub>i</sub>. No return value.

**getCoordinate** <atomIx>

Returns the coordinate of atom ;atomIx<sub>i</sub>. No return value.

**setCoordinate** <atomIx> <x> <y> <z>

Sets the coordinate of atom ;atomIx<sub>i</sub>. No return value.





## **Part V**

# **Alphabetic Index of Editors**



# Chapter 17

## Amira

### 17.1 Advanced Colormap Editor

To access this advanced colormap editor, press the *Advanced* button of the standard *colormap editor*. This colormap editor pops up. As you can see, there is a menu bar near the top border of the window with the submenus *Edit*, *Mode*, and *Brush*, which help you to control the behavior of the colormap editor and to change components of the chosen colormap. Below the menu bar a so-called "color chart" is displayed. The red, green, and blue lines are the graphs of the *color channel* components of the colormap. The underlying color model is RGB (at startup). The axes are not shown directly; the x-axis ranges from the lowest to the highest colormap index and the y-axis from 0.0 to 1.0 according to the RGB model. Thus, a point on a color line indicates the amount of color for the corresponding color channel and color index. You can manipulate the course of a line by setting a brush onto any of its points and moving it up or down. A brush is set by pressing a mouse button, the left one for the red line, the middle one for the green line, and the right one for the blue line.

Below the color chart a "color bar" is displayed which is a larger version of the one used for display in the colormap port. Its only function is to show you the actual appearance of the modified colormap in a smoother way by linearly interpolating the colors between the indices.

The largest area of the editor window is occupied by the "color buttons" which represent the color in every position (index) of the chosen colormap. Each position is called a "color cell". You can set the *focus* which is the target index of modifications applicable by the color sliders (see below), and it is also possible to modify the colormap by dragging a color cell. The index of the focus color cell is shown below the color buttons.

In the lower area of the window there are some "color sliders", one for each color channel by which you can modify color channel values with respect to the focus cell. This means that the values of neighboring cells are affected as well, as you can see in the color chart. The sliders are manipulated by dragging the small triangles; alternatively values in fixed-point format may be entered directly into the text fields to the right of the sliders.

The last three buttons named *OK*, *Apply*, and *Cancel* are for quitting the editor, applying your changes to the underlying *Colormap* data object, and quitting the editor without applying your changes.

Interactive editing of a colormap is facilitated by two user interface elements, the *focus* and *colormap knots*.

**Focus:** The focus marks the active color cell which can be edited using the color sliders. It is represented by a black-and-white box drawn around the active color cell and by a black vertical line in the color chart.

You set the focus by clicking with the left mouse button on a colormap entry in the color button panel. A focus cell also gets a *knot* marker (see below); clicking again on a focus cell removes the knot and places the focus on to the leftmost color cell, just setting the focus to a different cell does not remove a knot.

**Knots:** Knots are fixed points in the colormap, i.e., they retain their values while the colormap is being manipulated by snapping (see *Menu bar/Edit/Snap*) or by dragging the focus. Knots are represented by a small black box with a white surrounding in a color cell and a white vertical line in the color chart.

You set a knot by setting the focus and remove a knot (with the focus) by clicking another time on a color cell with the focus. The knots on the first and last cells of the colormap cannot be removed.

### 17.1.1 Description of the user interface elements

#### A. Menu Bar

The menu bar consists of the four submenus: *Edit*, *Mode*, *Brush*, and *Extras*.

##### a) Edit Menu

This submenu offers two opportunities for editing the colormap. The first group deals with the undoing and the opposite - redoing - changes you made in the colormap. The second group "snaps" one or more channels, like tightening a rope between the left and right neighboring knots of the focus.

- **Undo:** If you think that your last changes to the colormap have been a mistake, this entry lets you take back your last changes. You can easily go back to a point of editing when you were content with the colormap and start editing again. The colormap editor remembers up to 50 of your last changes.
- **Redo:** If you undid too much, this entry undoes the last undo, i.e. redoes the undone changes. This is the reverse function of *Undo*. You may undo/redo your changes as often you wish, but if you modify the colormap by another function (not *Undo/Redo*) you lose the opportunity to redo the last undos. *Redo* is only activated if your last action was invoking *Undo*.

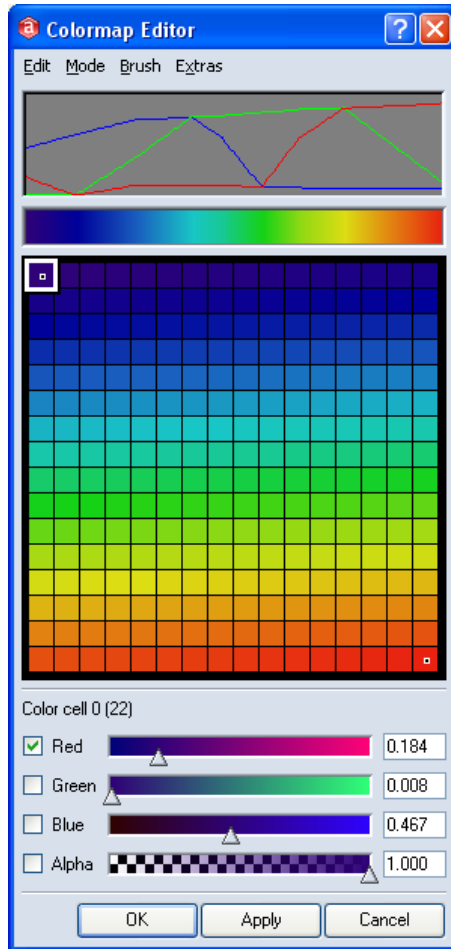


Figure 17.1: amira colormap editor

- **Snap All:** By selecting this entry, all colormap entries will be tightened like a rope (“snapped”) between their left and right neighboring knots leaving the knots itself untouched. This means that lines are drawn between all pairs of successive knots for each color channel. This function manipulates all color channels simultaneously.
- **Snap:** This operation behaves much like a local *Snap All*, i.e., snapping occurs only between the left and the right neighboring knots of the focus by tightening all color channels as if they were ropes between the knots.
- **Snap Red or Hue:** Depending on the chosen color model (see *Mode*), the first color channel “Red” or “Hue” is snapped between the left and right neighboring knots of the focus leaving the other color channels untouched. See *Snap* for an explanation of snapping.
- **Snap Green or Saturation:** Like in *Snap Red*, this entry manipulates the second color channel between the left and the right knot leaving the other color channels untouched.
- **Snap Blue or Value:** Like in *Snap Red*, this entry manipulates the third color channel between the left and the right knot leaving the other color channels untouched.
- **Snap Alpha:** This entry is only shown if *Show Alpha* (see *Mode*) has been activated. By selecting it, the alpha channel will be locally snapped as described in *Snap* between the left and the right neighboring knots of the focus leaving the other color channels untouched.

#### b) Mode Menu

This menu allows you to select the color model used to modify the colormap. *RGB Sliders* chooses the RGB model where colors are represented by a red, green, and blue component. *HSV Sliders* chooses the HSV model where colors are represented by a hue, saturation, and value (intensity) component. In addition, an *Immediate Mode* toggle is provided. If this toggle is active, all changes are immediately applied to the colormap and downstream modules are immediately fired so that they can update their display.

#### c) Brush Menu

This has only relevance for the color chart. Here you can set the brush type used for editing a color channel curve. Four different brushes are supported. Their shapes are visually represented by corresponding icons.

#### d) Extras Menu

This menu allows you to replace the whole colormap by one of a set of predefined maps. Currently four such predefined maps are available: a *Gray ramp* ranging from black to white, a *Hue ramp* ranging from blue over green and yellow to red, a *Hot iron* map ranging from red over yellow to white, and a *Glow* map ranging from black over red and yellow to white. This last map is frequently used in epi-fluorescence microscopy. Two additional options are provided to subdivide the current colormap into a discrete set of colors (*Make steps*) and to define an alpha curve with a predefined gamma value (*Alpha curve*). If one of these options is chosen, an additional dialog window pops up, allowing you to perform the appropriate operations.

### B. Color Chart

The color chart shows the graphs of the colormap’s color channel components. The red curve shows

the first color channel (red or hue), the green curve shows the second channel (green or saturation), the blue curve shows the third channel (blue or value), and the black curve shows the fourth channel (alpha). The knots and the focus are represented by white and black vertical lines, respectively. The left edge of the color chart shows the values of the leftmost index of the colormap and the right edge those of the rightmost index.

### C. Color Bar

Like in the colormap port, this color bar displays a continuous form of the chosen colormap by linearly interpolating the colors between the colormap entries. If *Show Alpha* is enabled, alpha values are represented by a certain amount of transparency in the colors, i.e., you see the colors translucent over a white and black checkerboard pattern. For an alpha value of 0.0, you see no color information but only the white and black checkerboard; for a value of 1.0 you do not see a checkerboard because the colormap entry is not translucent.

### D. Color Buttons

An entry of the color button panel is also called a color cell. Each color cell shows the color associated to a color index without the alpha value. The indices are counted from left to right and from top to bottom. Thus the color cell in the upper left corner shows the color of the leftmost colormap entry and the color cell in the lower right corner shows the color of the rightmost colormap entry. The index of the focus cell is displayed below the color button panel. Just after the index the data value which corresponds to the focus cell is shown in brackets. The data value depends on the current range of the colormap.

### E. Color Sliders

In the lower area of the *Colormap Editor* window there are four color sliders allowing you to modify the values of the focus cell. Depending on the current color model, the first three sliders are associated to red, green, and blue or to hue, saturation, and value, respectively. The fourth slider always modifies the alpha value. In front of each color slider there is a toggle button. If the toggle is activated, the corresponding color can be edited using the left mouse button in the *color chart* (see below).

### F. Control Buttons

These three buttons let you choose whether the changes to the colormap should be kept or not. By pressing the *OK* button the changes made with the editor are written back to the colormap object that you are working on. This action also causes the editor to exit. If you just want to write back the changes without exiting, e.g., if you want to see how your changes take effect, just press the *Apply* button. If *Apply* is done in the *Immediate* mode however, the previous state of the colormap cannot be restored by *Cancel*.

If you think that your manipulations have gone totally wrong, you can always decide to keep the old colormap and throw away your changes. This is done by pressing the *Cancel* button which also closes the editor window. *Cancel* restores the colormap to the last state when the *Apply* button was pressed, or to the initial, if *Apply* was left untouched.



## 17.1.2 How to modify a colormap

This section describes how to modify a colormap in various ways. For the effects of invoking the various menu items, see *Menu bar*.

### A. Using the color chart

A color channel can be edited by modifying the corresponding curve with a brush. Values are increased by approaching the curve from the bottom side with a brush while holding down the left mouse button, and vice versa. Only one curve can be edited at a time. The curve to be edited is determined by the toggle button in front of the four color sliders. For example, if you want to modify the alpha curve, you first have to select the toggle button in front of the alpha slider.

Four different brushes can be chosen in the *Brush* menu, namely a small square, a bigger square, a circle, and a diamond. The shape of the brush determines how a curve will be modified when approaching it with the mouse. When the brush touches the curve it moves the color channel up or down (depending from which side it comes) to the first pixel outside the brush's area. Colors are clamped to the channel's minimum and maximum allowed values.

The new colors are displayed instantaneously in the color bar and in the color buttons. If you modify the focus cell, the color slider corresponding to the curve being edited will also be updated.

### B. Using the color buttons

This item offers you the most opportunities to modify the colormap. You can set the focus, set/unset a knot, or modify a certain region of the colormap by dragging the focus cell. Due to the fact that a focus always exists, a special operation for unsetting the focus is not necessary because you unset a focus simply by setting a new one.

- **Setting the focus:** You simply select a focus by clicking once in a non-focus cell. The new focus cell will be surrounded by a black and white border while the border of the old focus cell disappears. The other displays will be refreshed, i.e. the color index displayed below the color buttons will be set to the focus cell's index, the color chart will display a black vertical line in a position corresponding to the new focus cell, and the color sliders will show the color channels' values in the focus cell. As mentioned above you simply unset a focus by setting a new one.
- **Setting a knot:** Because every focus cell must have a knot, you set a knot simply by setting the focus, i.e. by clicking once in a non-focus cell without a knot. The knot appears as a small black box with a white surrounding in the middle of the new focus cell. The color chart displays it with the focus as a black vertical line. If you change the focus, the knot still remains in the color cell but without the focus it is displayed in the color chart as a white vertical line.
- **Unsetting a knot:** You unset a knot by clicking another time into a focus cell. This does two things: First, the knot is removed from the color buttons and the color chart. Second, the focus is set to the first color cell, i.e. to the upper left corner of the color button panel and to the left edge in the color chart. The displays are refreshed, i.e. no small black box in the color cell and no vertical line at the corresponding position in the color chart will be shown.
- **Dragging the focus cell:** When you click the first time in a non-focus cell, you can hold the mouse button (be careful: a click in a focus cell unsets a knot) and drag the focus cell over the

color buttons. The corresponding color values will be temporarily copied to its new position and the color channels between the next knots to the left and to the right from the actual position will be snapped i.e. the color channels will be tightened like a rope between the focus and the left or right knot. When you move in a new region between two knots, the old region will be restored and the new region will be affected by snapping. When you release the button, the focus cell's color values (the one which you dragged) will be copied to its last position and the colormap remains in this state, i.e., with the copied color values in the focus cell and the snapped color values between the focus and the left or right knot. While moving the mouse, the color chart is updated appropriately. The color sliders stay the same because the focus cell values remain the same (remember: you drag the focus and copy its cell's values).

### C. Using the color sliders

The color sliders offer you three ways to modify the values associated to the focus cell, either by setting the value explicitly in the text field, by dragging the small triangle, or by clicking somewhere in the slider area.

All modifications have an effect on the surrounding colormap entries between the next knots to the left and to the right of the focus cell. A modification of a value changes the surrounding colormap entries relative to their distances to the focus. This is similar to raising / lowering a rubber band between the left and right knot in the focus position. You can clearly see the effect by looking at the curves displayed in the color chart. The values and displays are refreshed in the same way as described in the previous sections.

## 17.2 CameraPath Editor



The *CameraPath Editor* allows you to edit camera paths defined by a number of keyframes. To get started, choose *CameraPath* from the *Create* menu and click on the Camera Path Editor button of the new camera path object. The editor will appear and automatically open an additional viewer.

The original viewer (viewer 0) is the camera view, i.e., it shows the same as if you were looking through the camera, while the additional viewer will show you all of the keyframe cameras from a bird's eye view.

In order to start your camera path, choose an appropriate view in the original viewer (see Section ?? (Viewer Window) to learn how to do that) and click on the *add* button. Your first keyframe has been saved. The time slider should contain one red line and it should have advanced to 10. Now change the view in the original viewer and click on *add* again. A second keyframe appears and so on.

Once you have played your camera path using the play button, you may continue adding keyframes by simply typing the time for the new keyframe into the text field or by moving the slider and then clicking *add*. Note: The original view will not be changed when the slider is moved or the text in the text field is changed.

To change the time of a keyframe, jump to this frame, click on *remove*, type in the new time, and click

on *add* again.

If you want to have a constant camera velocity on its path, press the *const velocity* button. Depending on the distances between the camera positions at the keyframes, the keyframe times are changed to be equidistant between camera positions.

**Caution:** Pressing the *circular path* button will create a new camera path and destroy the old one. The new camera path lies on a circle around one of the major axes with the center at the point the camera in the original window is looking at. The radius is determined by the distance of the camera position and the point the camera looks at.

## 17.3 Color Dialog

The *Color Dialog* provides an interface for selecting colors. It pops up whenever a single color is to be changed in *amira*. The dialog provides several different interface components: a menu bar, color sliders, default color cells, a color picker, and five buttons.

The menu bar allows you to set some options for working with the color dialog. The sliders (and the corresponding text fields and arrow buttons) let you choose a color by hue, saturation, and value (HSV), or by the levels of red, green, and blue (RGB). The color picker allows you to pick a color by sight. Moreover, you can choose a predefined color from a palette of custom colors cells. The buttons are used to apply or reset changes and to quit the dialog. A detailed description of the interface components is given below.

**Menu Bar:** The menu bar holds two popup menus: the *Edit* and the *Options* menu. There are two items in the *Edit* menu allowing you to toggle between different editing modes: the *Immediate Mode* and *WYSIWYG Mode*.

If *Immediate Mode* is selected, each modification of the color to be changed is effective immediately,

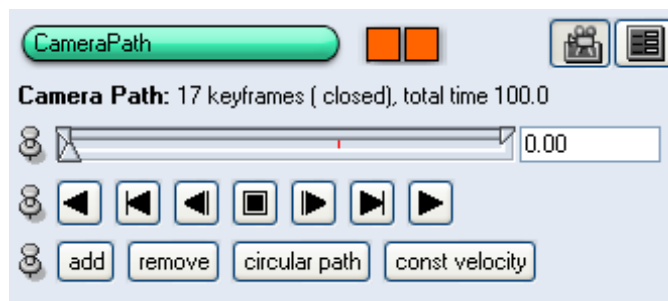


**Figure 17.2:** Control panel of the camera path editor. The arrow shaped control buttons can be used to play forward and backward, to jump from keyframe to keyframe, or to step frame by frame. When the editor is off, the camera path provides a time port.

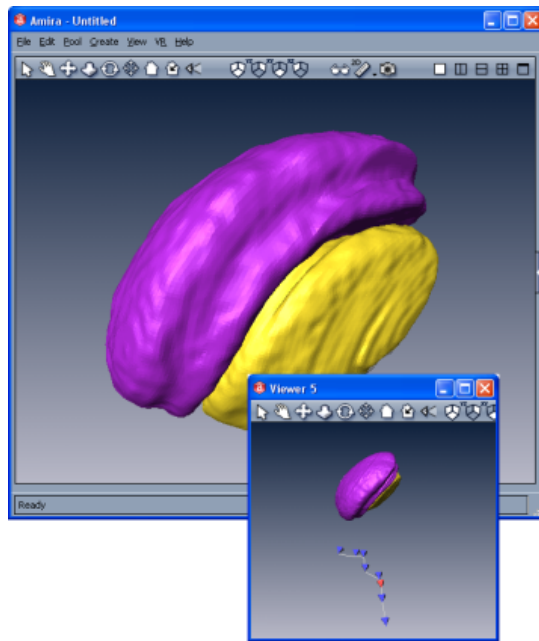


**Figure 17.3:** The camera path is shown in an extra viewer. You can click on a keyframe and modify the camera. The changes will be displayed in the original viewer simultaneously. The reddish icon represents the current camera. In general this is not a keyframe but an interpolated camera.

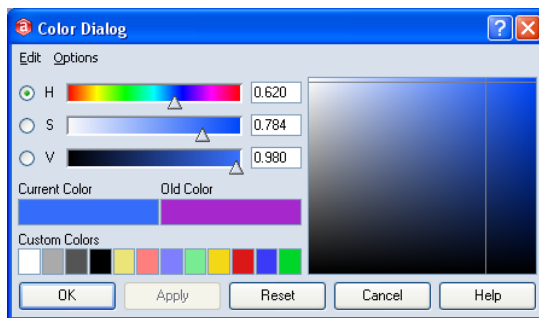


**Figure 17.4:** amira color dialog

e.g., if you are changing the background color of the 3D viewer and *Immediate Mode* is active, the effect of every slider or picker operation can be observed directly in the viewer.

The *WYSIWYG Mode* (meaning *what you see is what you get*) determines the background of the color sliders. If *WYSIWYG Mode* is active, the background of a slider shows a color range representing how the current color would change by moving this slider.

You have the option of displaying and manipulating two different combinations of sliders: RGB and HSV. The *Options* menu holds two items representing these two combinations. Selecting *RGB Sliders* provides controls over RGB color space. Selecting *HSV sliders* provides sliders to manipulate the HSV color space.

**Color Sliders:** The color sliders present the color range in hue, saturation, value (HSV) or red, green, blue (RGB) color spaces. If the color to be changed has an alpha component, an additional slider is automatically displayed. Each slider controls one color component. The color sliders are visible depending on the user selection in the *Options* menu. Each of the color sliders is accompanied by a text edit field to view the exact value of the current color component and to set its numerical value. The component values are always in the range from 0 to 1. Furthermore, two arrow buttons can be used to move each slider respectively.

On the left side of each slider (except the alpha slider) an additional check button is displayed allowing you to select one color component to control the appearance and functionality of the color picker.

**Color Cells:** The color cells are subdivided into three different groups: One cell named *Current Color* displays the color depending on the current settings. Each color manipulation is shown by this cell immediately. The cell named *Old Color* displays the original color the dialog has been invoked with. The old color does not change until current setting are applied by activating the *Apply* button. The other cells labeled *Custom Colors* provide a user-defined palette for storing colors. The custom colors stay resident between successive color dialog popups.

The *Drag and Drop mechanism* is applied to store and restore colors. The colors can be copied arbitrarily between all cells (except that a color cannot be stored to the old color cell). To drag and drop a color

1. move your mouse cursor on top of a color cell (source),
2. press down on the left mouse button and keep it pressed,
3. move your mouse cursor on top of another cell (destination),
4. release the mouse button.

**Buttons:** The four buttons named *OK*, *Apply*, *Reset* and *Cancel* are for quitting the dialog and applying the changes to the underlying color (*OK*), applying the changes without quitting (*Apply*), resetting the current color to the old color, i.e. discarding last changes (*Reset*), and quitting the dialog without applying the changes (*Cancel*). The fifth button named *Help* is for displaying this documentation in the amira help window.

**Color Picker:** The Color Picker provides visual selection of a color. Depending on the selected color

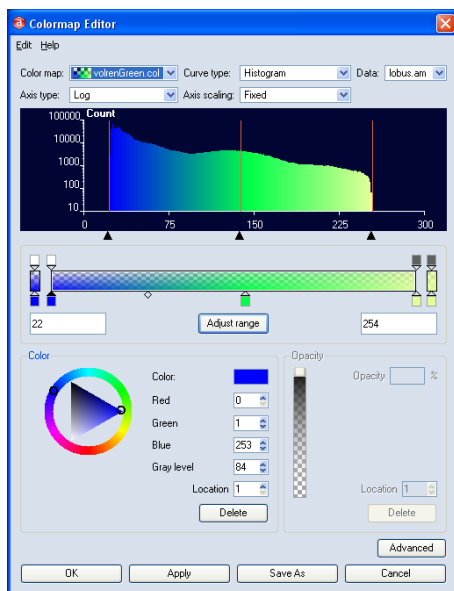


**Figure 17.5:** amira colormap editor.

component (selecting using the radio button on the left side of each slider), the two other components of the color can be set with the picker. One component corresponds to the vertical extent of the color picker and the other to the horizontal. The selected component cannot be changed with the picker. To select a color using the color picker

1. Move your mouse cursor on top of the color picker.
2. Press down on the left mouse button and move your mouse. While moving the mouse around the current color is set to the color at the mouse position.
3. Release the mouse when you are done.

## 17.4 Colormap Editor



To modify an existing *colormap*, press the Colormap Editor button which is visible in the Properties Area when the colormap is selected. The colormap editor pops up.

## 17.4.1 Overview

This editor allows you to interactively edit a colormap and to graphically visualize the result on a selected data set by means of the histogram. The histogram is automatically colored using the current color settings.

The interface is divided into three main parts: the histogram, the gradient editor, and the color/opacity chooser. The histogram is only for visualizing the relation between a selected data set and a selected colormap. The gradient editor and the color/opacity chooser work together to modify the selected colormap.

## 17.4.2 Menu Bar

The menu bar contains two items: a *Help* button which brings up this help topic in the help browser, and the *Edit* menu containing two menu items:

- **Background Color:** A color chooser dialog will be popped up allowing you to change the *Histogram* background color.
- **Immediate Mode:** If this toggle is active, all changes are immediately applied to the colormap and downstream modules are immediately fired so that they can update their display.

## 17.4.3 Global Settings

### 17.4.3.1 Colormap

This combo box is filled with all colormaps present in the Pool. The last item, *Load...*, allows you to load a colormap from disk. **Note:** Changing the current colormap will discard any changes that have not been applied or saved.

### 17.4.3.2 Curve Type

This field controls the *Histogram* curve representation.

- **Histogram:** Display as a bar graph.
- **Histogram with opacity:** Display as a bar graph with opacity.
- **Curve:** Display as line segments.
- **Impulse:** Display as single vertical lines.

### 17.4.3.3 Data

The *Data* field represents the input data used to calculate the *Histogram*. If no data is available, nothing is shown in the *Histogram* and the field indicates No Data. This field is filled with the data in the Pool

from which a histogram can be computed (see *Histogram*). If the selected colormap is referenced by a data set, this data set is selected. Otherwise, the first compatible data set found in the Pool is selected.

#### 17.4.3.4 Axis Type

This field controls how the *Histogram* vertical axis is drawn.

- **Linear:** Use vertical linear projection.
- **Log:** Use vertical logarithmic projection. (Allows you to better visualize small variations.)

#### 17.4.3.5 Axis Scaling

This field allow you to scale the *Histogram* depending on the min/max range to better visualize your mapping.

- **Fixed:** This is the default mode. All data values are shown on the *Histogram*, including the values outside the min and max range.
- **Auto:** In this mode, all data values outside the min and max range are ignored. The data values in the range are scaled to fit within the vertical space, allowing you to better visualize a range of data values.

### 17.4.4 Histogram

The purpose of the *Histogram* is to graphically summarize the distribution of values within the selected data set, colored by the current colormap. The horizontal axis displays the data set values, the vertical axis displays the frequency. The three vertical lines indicate the coordinate range used for mapping data values to colors. To adjust this range, you can drag the black triangle markers at the bottom. The left and right markers move the min and max range, while the middle marker moves the min and max at the same time.

#### 17.4.5 Gradient Editor

The gradient editor allows you to modify the colormap using color and opacity markers. The color is linearly interpolated between two consecutive markers. Color and alpha channels are treated separately.

When selecting a colormap, the markers are automatically extracted. To modify the gradient, you can add or remove markers. To add a new marker, click on the area containing the color or opacity marker at the desired position. A new marker is added with the interpolated value. You can move this marker horizontally by dragging or by setting a value in the *Location* field. The small triangle over or under the marker indicates its selection state: solid black when selected , transparent otherwise . The color chooser and opacity chooser values are updated each time you select a marker.



Depending on which kind of marker you select, the color chooser or opacity chooser is activated. The upper markers are for the alpha channel, the lower markers are for the RGB channel. The corresponding *Location* field and *Delete* button are enabled.

To remove a marker, select it and click on the *Delete* button in the active chooser.

To the extreme right and left are the min and max range color/opacity values, which cannot be moved. Instead, two fields are provided, one on the far left, one on the far right, to allow you to specify the range of the data that the colormap will be mapped to. All input data values less than the left (minimum) value or greater than the right (maximum) are colorized using the min range color/opacity or the maximum color/opacity respectively. All other input data values are colorized with the interpolated gradient. To reposition the median point ♦ (the gradient point containing equal start and end proportion), drag it horizontally to the desired position.

Click the *Adjust range* button to automatically adjust the colormap range to the connected data range.

### 17.4.6 Color Chooser

When a color gradient marker is selected, the color chooser is activated. To the left, the control is an HSV color triangle. The circle represents the hue and the triangle the saturation and value. By moving the two cursors, it is possible to generate all possible colors of the HSV color space.

The color is also displayed as an RGB triplet in the *Red*, *Green*, and *Blue* fields. You can also enter the desired values explicitly [0..255].

The *Gray level* input field allows you to quickly specify a gray level. The *Location* field shows or sets the position of the color marker. Press the *Delete* button to delete the selected marker.

### 17.4.7 Opacity Chooser

When an opacity marker is selected, the opacity chooser is activated. The vertical slider controls the opacity value (must be between 0 and 100, 100 meaning totally opaque). This value can be also modified by the *Opacity* input field.

The *Location* field and the *Delete* button work as described above.

### 17.4.8 Control Buttons

- **Advanced:** Launch the amira 3.1 advanced *colormap* editor.
- **Save As:** Pop up a *Save File* dialog to save your colormap to disk.
- **Cancel:** Cancel all unsaved modifications and close the editor.
- **Apply:** Apply your changes to the selected colormap.
- **OK:** Apply your changes to the selected colormap and close the editor.

## 17.5 Curve Editor



This editor allows you to create and modify *B-Spline* curves.

The curve is defined by a set of control points. The control points can be appended or modified by moving a dragger, by clicking on an object in the viewer with the middle mouse button, or by entering numerical values into the port *Control Points*;

### Ports

#### Degree

A port control for the Degree parameter. It features a blue icon with a gear and a plus sign, followed by the text "Degree:" and a text input field containing the number "3".

Enter the degree of the B-Spline. The higher the degree, the more control points are required and the smoother the curve. Also, the oscillations increase when the degree increases.

#### Mode

A port control for the Mode parameter. It features a blue icon with a gear and a plus sign, followed by the text "Mode:" and a dropdown menu currently set to "Interpolation".

In *interpolation* mode, the curve goes through the specified points. In *approximation* mode, the curve approximates the polygon spanned by the points.

#### Editing Mode

A port control for the Editing Mode parameter. It features a blue icon with a gear and a plus sign, followed by the text "Editing Mode:" and two radio buttons labeled "Append" and "Modify". The "Append" button is currently selected.

The dragger either appends new points or modifies the currently selected point. Both modes apply as well when clicking on an object in the viewer with the middle mouse button.

#### Points

A port control for the Points parameter. It features a blue icon with a gear and a plus sign, followed by the text "Point:" and a list of three input fields containing the values "3", "53.159", and "142.01". To the right of these fields is an "options" button.

Specifies the control points that define the curve. The points can be modified by entering numerical values here, by moving the dragger and clicking on the middle mouse button, or by doing a click on an object in the viewer with the middle mouse button. In the *Options* menu, the dragger and the control points can be switched on and off in the display.

## 17.6 Digital Image Filters



This editor provides digital image filters for 3D image data sets, such as smoothing, unsharp masking, and morphological operations. Some filters operate in 3D while others are applied to two-dimensional slices. In the latter case, the orientation of the 2D slices can be selected via an option menu. Press the *Apply* button to start the computation.

Currently, the following filters are supported:

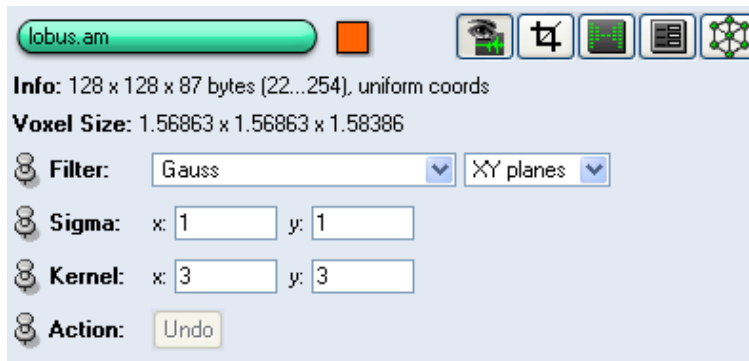


**Figure 17.6:** Editor for applying digital image filters.

- *Minimum Filter*
- *Maximum Filter*
- *Unsharp Masking*
- *Laplacian Zero-Crossing Filter*
- *Median Filter*
- *Gauss Filter*
- *Sobel Filter*
- *Histogram Filter*
- *Edge-Preserving Filter*
- *Lanczos Filter*
- *Sigmoid Filter*
- *Brightness/Contrast Filter*
- *Moments Filter*
- *Gamma Correction*

## Ports

### Filter

Specifies the filter and its domain. It allows you select whether the filter should be applied to the XY, XZ, or YZ slices or to the entire three-dimensional image. Depending on the selected filter, additional ports will be visible.

### Action

The *Undo* button allows you to undo the last filter operation.

### 17.6.1 Minimum Filter

The minimum filter replaces the value of a pixel by the smallest value of neighboring pixels covered by an  $N \times N$  mask. The size of the mask can be adjusted via the input field *kernel size*. A value of 3 denotes a 3x3 mask (3x3x3 in 3D). If applied to a binary *label field*, the minimum filter implements a so-called erosion operation. It reduces the size of a segmented region by removing pixels from its boundary.

### 17.6.2 Maximum Filter

The maximum filter replaces the value of a pixel by the largest value of neighboring pixels covered by an  $N \times N$  mask. The size of the mask can be adjusted via the input field *kernel size*. A value of 3 denotes a 3x3 mask (3x3x3 in 3D). If applied to a binary *label field*, the maximum filter implements a so-called dilation operation. It enlarges the size of a segmented region by adding pixels to its boundary.

### 17.6.3 Unsharp Masking

This filter sharpens an image using an unsharp mask. The unsharp mask is computed by a Gaussian filter of size *kernel size*.

Then, the smoothed image is subtracted from the original image such that only high contrast remains. The weighted difference of the original image (weight:  $\frac{c}{2c-1}$ ) and the blurred image (weight:  $\frac{1-c}{2c-1}$ ) is calculated afterwards using the *sharpness parameter*  $c$ . It determines the relation between the original and blurred image, effectively controlling the amount of sharpness.  $c$  can be adjusted via a text input field and should be in the range of 0.6 to 0.8. A value of 1 leaves the image unchanged.

### 17.6.4 Laplacian Zero-Crossing Filter

The Laplacian filter is a rotation invariant edge detection filter. The algorithm finds zero crossings of the second derivative, i.e., changes of the sign of the first derivative of the "image function" which may indicate an edge.

### 17.6.5 Median Filter

The median filter is a simple edge-preserving smoothing filter. It may be applied prior to segmentation in order to reduce the amount of noise in an image. The filter works by sorting pixels covered by an  $N \times N$  mask according to their gray value. The center pixel is then replaced by the median of these pixels, i.e., the middle entry of the sorted list. The size of the pixel mask may be adjusted via the text field labeled *kernel size*. A value of 3 denotes a 3x3 or mask (3x3x3 in 3D). An odd value is required.

### 17.6.6 Gauss Filter

The Gauss filter smooths or blurs an image by performing a convolution operation with a Gaussian filter kernel. The text fields labeled *kernel size* allow you to change the size of the convolution kernel in each dimension. A value of 3 denotes a 3x3 kernel (3x3x3 in 3D). Odd values are required. The text fields labeled *sigma* allow you to adjust the width of the Gauss function relative to the kernel size.

### 17.6.7 Sobel Filter

The Sobel-Filter is a rotation variant edge detection filter. It convolutes the image with 4 different filter kernels representing horizontal, vertical, and two diagonal orientations. Each kernel is constituted of a combination of Gaussian smoothing and the differentiation in the proper orientation.

### 17.6.8 Histogram Filter

This filter performs a so-called *contrast limited adaptive histogram equalization (CLAHE)* on the data set. The CLAHE algorithm partitions the images into *contextual regions* and applies the histogram equalization to each one. This evens out the distribution of used gray values and thus makes hidden features of the image more visible. Parameter *Contrast Limit* determines the contrast limit for the CLAHE algorithm.

### 17.6.9 Edge-Preserving Smoothing

This is a smoothing filter that models the physical process of diffusion. Similar to the Gaussian filter, it smooths out the difference between gray levels of neighboring voxels. This can be interpreted as a diffusion process in which energy between voxels of high and low energy (gray value) is leveled. In contrast with the Gaussian filter, it does not smear out the edges because the diffusion is reduced or stopped in the vicinity of edges. Thus, edges are preserved.

**Note** that in the 3D mode the computation can take a rather long time if the data is large. A faster preview is always possible by switching to the 2D mode.

The stop *time* determines how long the diffusion runs. The longer it runs, the smoother the image becomes. The *time step* determines how accurately this process is sampled.

The *contrast* parameter determines how much the diffusion process depends on the image gradient, i.e., how much the smoothing is stopped near edges. A value of 0 makes the diffusion independent of the image gradient and smooths out the edges, a large value prevents smoothing in all edge-like regions.

In order to make the diffusion process more stable, the image is prefiltered by a Gaussian filter with parameter *sigma*. All features of size *sigma* are removed. This allows noise to be removed from the image. But a too large value may also remove relevant features.

### 17.6.10 Lanczos Filter

The Lanczos filter can be used to sharpen images. It performs a convolution with a Lanczos kernel:

$$f(x) = \frac{\sin(\pi x) \sin(2\pi x)}{2\pi^2 x^2}, |x| < 2$$

The kernel size in each dimension can be adjusted using the parameter inputs *kernel size*. A value of 3 denotes a 3x3x3 kernel. Odd values are required.

Parameter *Sigma* determines the effective size of the Lanczos function. For large values of sigma the effect of the filter will be a smoothing rather than a sharpening.

### 17.6.11 Sigmoid Filter

This filter operates on single voxels (kernel size 1) and is used to raise a specific intensity range. This is useful as a preprocessing step in image segmentation. The intensity range is described by its center  $\beta$  and its width  $\alpha$ . The target image range is given by the interval  $[min, max]$ .

$$f(z) = (max - min) \cdot (1 + \exp(-\frac{z - \beta}{\alpha}))^{-1} + min$$

### 17.6.12 Brightness and Contrast Filter

This filter modifies the image brightness by adding an offset to the image values. The contrast is modified by multiplying the difference from the voxel values to the average image intensity.

$$f(x) = \text{Brightness} + x_{av} + \text{Contrast} \cdot (x - x_{av}),$$

where  $x_{av}$  is the average (brightness) value in the image.

### 17.6.13 Moments Filter

This filter calculates the  $n$ -th centralized moment of the data in a gliding window. The centralized moments of order  $n$  are defined by:

$$f(x) = \frac{1}{N-1} \sum_{i=0}^N (x_i - \bar{x})^n.$$

The second moment ( $n = 2$ ) is therefore the local variance in the data. For some data sets this can be used to mask out noisy regions or to detect edges.

Because of the  $n$ -th power involved in the computation, you may want to use *CastField* to do a conversion of your data set to floats or doubles first.

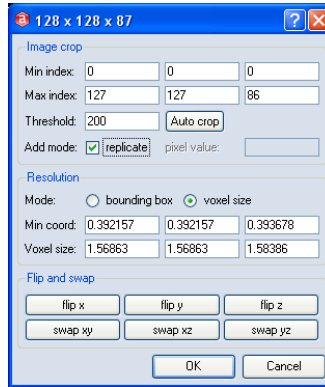


**Figure 17.7:** The ImageCrop editor allows you to crop a 3D image and to modify its bounding box or voxel size.

### 17.6.14 Gamma Correction

A gamma characteristic is a power-law relationship that approximates the relationship between the encoded luminance in a display system and the actual desired image brightness. With this nonlinear relationship, equal steps in encoded luminance correspond to subjectively approximately equal steps in brightness. Computer graphics systems that require a linear relationship between these quantities use gamma correction.

Specifying a large value *Gamma* leads to darker images. The *Range* specifies the intensity values that correspond to black and white. Intensity values outside this range are clipped off.

## 17.7 ImageCrop Editor



This editor works on 3D fields with an arbitrary number of components defined on a regular grid, e.g., 3D images or 3D vector fields with uniform, stacked, rectilinear, or curvilinear coordinates. It allows you to crop such a field, i.e., to discard all voxels lying outside of a specified subvolume in index space. Alternatively, you may adjust the physical size of the field by changing its bounding box. Note that this does not mean *resampling*, i.e., the number of voxels will not be changed.

Each node of a regular grid can be addressed by an index triple  $(i,j,k)$ . Each index ranges from zero to the number of slices minus one in that direction. This editor also allows you to add new slices on every side in every dimension. Last but not least, you can flip and rotate the slices with respect to the global x-, y-, or z-direction.

When you activate the crop editor, the dialog shown in Figure 17.7 pops up. If the editor is activated for a field with uniform, stacked, or rectilinear coordinates, in addition a tab box dragger is displayed

in the *viewer window*. The dragger allows you to specify the subvolume to be cropped interactively in 3D. When you resize the dragger, the index bounds of the current subvolume are permanently updated in the corresponding fields in the dialog window.

The *Auto crop* button automatically adjusts the subvolume to be cropped by taking into account the value of the *threshold* field. Slices at the boundaries of the original data volume are cropped if they contain only data values smaller than the specified threshold. In this way it is easy to isolate a bright object in a 3D image with a large dark background.

In order to add slices, you must set the indices in the text fields beyond the limits of the total data volume. When slices are added, by default the values of the last slice in that direction are replicated. If you switch off the *replicate* toggle, the text field *pixel value* becomes active. You can then specify a data value which is used to initialize the new slices.

In order to manipulate the bounding box of the data object, new values must be entered in the corresponding text fields. For data objects with uniform coordinates two modes are available, *bounding box* and *voxel size*. See section below for details.

All changes take effect if you press the *OK* or the *Apply* button. They are discarded by activating the *Cancel* button. As usual, the editor is closed by pressing *OK* or *Cancel*. *Apply* will leave it open, and *Cancel* restores the previous state.

### 17.7.1 Cropping an image by dragging and moving the box

Some experience with manipulating *dragers* is assumed. Using the draggers you can reduce or enlarge the data volume you want to preserve; enlarging is, of course, possible only up to the size of the bounding box. The size of the dragging box can be manipulated intuitively in every direction of the three axes, i.e., along the x-, the y-, or the z-axis. The values in the text fields of the editor change according to your manipulations. A certain minimum thickness is preserved while reducing a dimension of the box; for further reductions the text fields must be used.

Using the draggers you control the size of the subset of the data volume which must be preserved, but the location of it in the total data volume is set by moving the box around to the desired location.

### 17.7.2 Cropping an image by setting values explicitly in the text fields

The text fields in the cropping area of the editor show the indices of the slices that must be preserved for every dimension. You can easily define a subset by setting a range of indices explicitly in the text fields of an axis - this defines the size as well as the location of the data subset. Notice that a slice index refers to an axis that is perpendicular to the slice.

### 17.7.3 Adding new slices

Adding new slices is only possible by setting an index that does not fall into the range given by the minimum and maximum presently shown in the *Image Crop* panel; thus a negative index must be



specified if the minimum is zero. Use the *Min* or *Max* text field for the required axis text field to enter a new slice index.

A warning dialog pops up asking whether you really want to add new slices; upon your confirmation, the slices are added to the volume at the specified end and according to the axis selected. The new slices contain the data of the last slice in this direction, e.g., if you add two slices before the slice with index 0 along the x-axis, they contain exactly the same data as the slice with index 0.

### 17.7.4 Changing the size of the bounding box

You change the size of the bounding box by setting new values explicitly in the bounding box fields. This does *not* affect the stored data, but its representation as displayed by a viewing module, e.g., *OrthoSlice*, by way of a different scaling for the specified dimensions.

The bounding box of a data set with uniform coordinates specifies the world coordinates of the center of the lower left front voxel (min values) and the center of the upper right back voxel (max values). For example, if your input is 256 pixels wide and the size of each voxel is 1mm, then you may set *xMin* to 0 and *xMax* to 255.

If a data set with uniform coordinates is being edited, instead of the bounding box, also the *voxel size* can be modified. You can switch between the two modes and see how bounding box and voxel size influence each other.

### 17.7.5 Flipping slices in one dimension

Press one of the three flip buttons to reverse the order of slices in the corresponding dimension, e.g., click on *FlipX* to flip the slices in the x-dimension.

### 17.7.6 Exchanging two dimensions

Press one of the three exchange buttons to interchange the corresponding dimensions, e.g., click on *X-Y* to interchange the x- and y-dimension. CAUTION: The exchange operations are not rotations; they change the spatial orientation of the dataset.

## 17.8 LandmarkEditor



This component allows you to edit *landmark sets*. It operates by directly interacting with the 3D geometry in the viewer window, cf. Section ???. In order for this to work, the viewer must be switched to interaction mode, e.g., by pressing ESC as described in its documentation.

The editor has different **Edit modes**, which are described in the following:

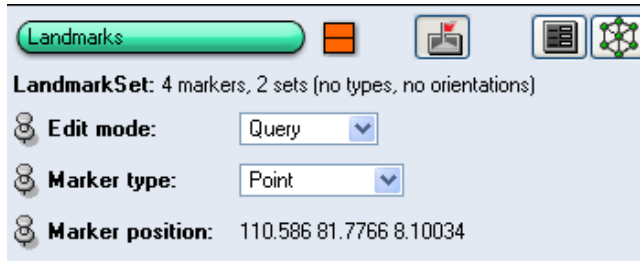


Figure 17.8: User interface of the landmark editor.

- **Add.** In this mode new landmarks are added by clicking onto a piece of 3D geometry (e.g., an Isosurface or an OrthoSlice). Multiple clicks are required if the landmark set contains more than one point set.
- **Move.** Markers can be moved in this mode by first clicking on the marker (selecting it) and then clicking to a new position.
- **Transform.** In this mode, markers can be moved using a dragger. Click on one of the markers, then use the dragger to manipulate it.
- **Flip.** Flip geometry of marker. This only makes sense if the marker type is not *Point*.
- **Remove.** The marker you click on will be removed. If more than one point set is present, the marker will be removed from all sets.
- **Query.** After you click on a marker, its type and its xyz-position are shown. The marker type can be changed via an option menu. The default marker type is a *Point*, represented by a little sphere. In addition, a number of predefined medical marker types can be selected. In contrast to point-type markers, medical markers have fixed predefined sizes (world coordinates assumed to be given in centimeters).

See also documentation for *Landmark Set*.

## 17.9 LineSet Editor



This component allows you to edit *linesets*. It operates by directly interacting with the 3D geometry in the viewer window, cf. Section ???. In order for this to work, the viewer must be switched to interaction mode, e.g., by pressing ESC as described in its documentation. Points and lines can be selected by clicking on them in the viewer. Selected points or lines will be highlighted in red. Multiple actions can be performed by pressing one of the buttons in the *Action* port. They are described below.

The *Selected* port tells you how many points and lines are selected at the moment.

In the *Display* port you may choose how the lineset is displayed. By clicking on the toggles, optionally *all points*, *endpoints*, and/or *lines* will be displayed in the viewer. Only displayed geometry is

selectable. The default is *endpoints* and *lines*.

The *Select* port provides the following selection modes:

- **All.** All lines will be selected.
- **By Length.** An additional dialog pops up where you can enter the minimal and maximal numbers of points in lines that are to be selected. Pressing the *OK* button actually selects the requested lines.
- **Clear.** The current selection will be cleared.
- **By Line Index.** Brings up a dialog that lets you enter a line index. Pressing the *OK* button selects this line.

The *Action* port provides the following actions:

- **Connect.** This button allows two lines to be connected. In order to do this, exactly two points must be selected, otherwise this button will not be active.
- **Delete.** Selected points and lines will be deleted from the lineset.
- **Stretch.** All selected lines will be smoothed.
- **Split.** All lines will be split at selected points.

## 17.10 Parameter Editor



The *Parameter Editor* lets you view and modify the parameter list associated with a data object. A parameter is a name/value pair which contains extra information related to the data. Some parameters have a special meaning within *amira*, see Section ?? of the user's guide.

To change the value or name of an existing parameter, click on the respective line and change the name/value in the text fields. To add a new parameter, right click on the root of the parameter tree, labeled *Parameters* and select *New Parameter*. Then select the new parameter to change its name and value.

Note that there are many standard file formats which do not support parameter information to be written. If in doubt, use *amira*'s native *AmiraMesh* format. This format does write parameters.

## 17.11 Plot Tool

The *Plot Tool* is a special-purpose viewer designed to display 1-dimensional data, e.g., function curves. Instead of being a separate *amira* module, the *Plot Tool* is invoked implicitly by certain modules such as the *data probing modules* or the *LabelVoxel* module. Each of these modules generates 1-dimensional

data such as a function plot along a line or a histogram. While the data-generating modules control the initial settings of the plot window, the user can freely adjust these settings afterwards.

### 17.11.1 Plot basics

The layout of the plot is determined by objects and groups of objects. These entities are processed by a plot engine. In particular, there are objects for

- Curves
- Cartesian axes
- Polar axes
- Plot areas
- Legends
- Annotations
- Marker lines
- Lattices
- Colormaps

The plot engine processes the objects top to bottom (see Figure below). The sequence of objects determines their behavior. Objects of type *PlotArea* define the area within the plot window where objects are placed. This area is given in normalized coordinates with the origin at the lower left corner. Besides this, a *PlotArea* object also acts as a grouping object. An axis is placed in such an area and establishes, together with the preceding *PlotArea*, a window-viewport (world coordinates to normalized coordinates) transformation. Note that the objects must be kept in the following sequence:

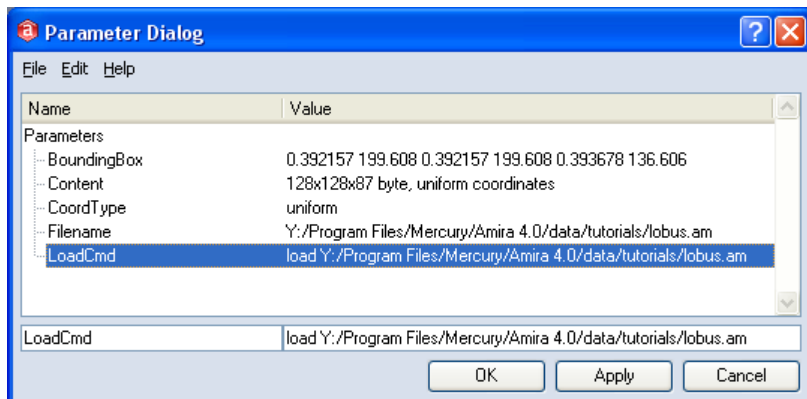


**Figure 17.9:** User interface of amira's parameter editor.

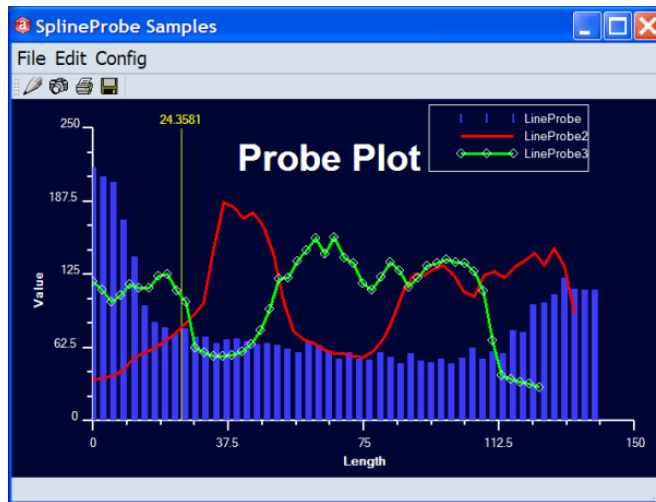


Figure 17.10: amira's plot window.

1. *PlotArea*, 2. *Axis*, 3. *Curves* and *Marker lines*. Legends display all succeeding curves in the sequence and can be placed wherever needed. It is possible to have more than one plot area or axis in a plot setup.

Every object is identified by a unique name. You will need to know these names if you want to change certain object attributes via the command interface of the *Plot Tool*.

## 17.11.2 Editing parameters

In order to change the parameters of a plot object, select the *Edit Objects...* item from the *Edit* pulldown menu of the *Plot Tool*. A window appears with the list of names of all objects currently in use. When you select one of these objects, the parameters of that object are displayed and can be changed. The *is active* toggle near the lower left corner of the window can be used to switch the processing of the selected object on or off. If the processing of an object which has group functionality (Plot Areas, Plot Groups) is switched off, all objects within that group are also not processed.

The following sections document all parameters that are not self-explanatory.

### 17.11.2.1 Editing axis parameters

To choose the x- or y-component of an axis, just click on the appropriate tab. If you want to change the range, you must switch off the *Auto* toggle. This causes the range fields to become sensitive. The *Nice Num* toggle sets the range to the next "good looking" boundaries. A *tick delta* can be typed in

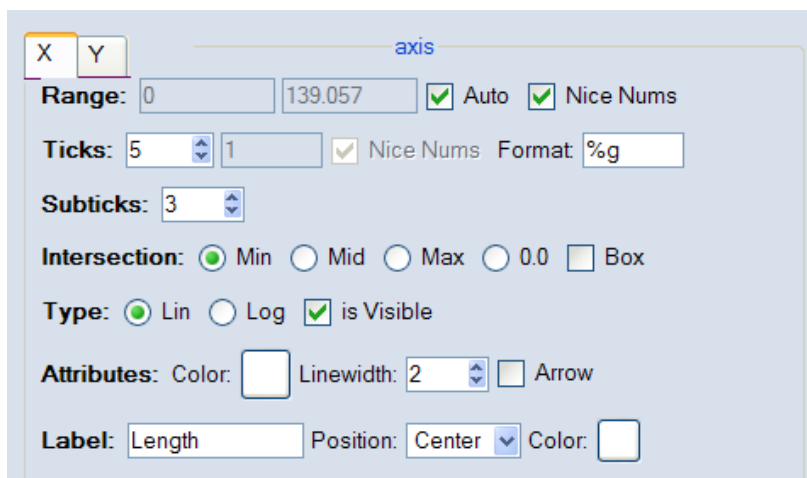


**Figure 17.11:** Dialog for editing plot objects. On the right hand side the controls for editing axis parameters are shown.

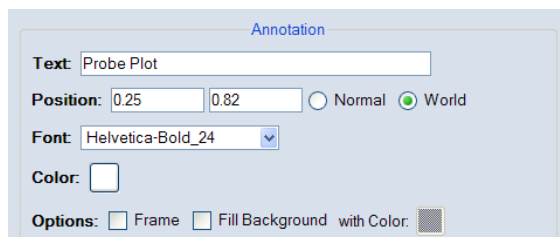


**Figure 17.12:** Editable annotation parameters.

after the *number of ticks* is set to -1.

It is possible to zoom interactively into your data using the mouse. To do this, drag a rectangle within the plot area by pressing the shift key *and* the left mouse button while moving the mouse. To go back to the automatic ranges, click into the plot area with the left mouse button while the Alt key is pressed at the same time.

Every axis object has a hidden grid child object which is not active by default. To show the grid, just open the grid child and select it. Then set the *is active* toggle to on.

### 17.11.2.2 Editing annotation parameters

To position an annotation on your plot, you can switch between world coordinates or normalized coordinates. Using world coordinates is convenient if you want to annotate a certain feature, e.g., an extrema

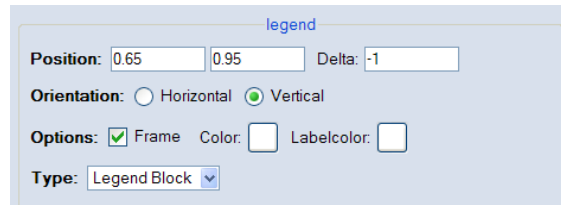


**Figure 17.13:** Editable legend parameters.

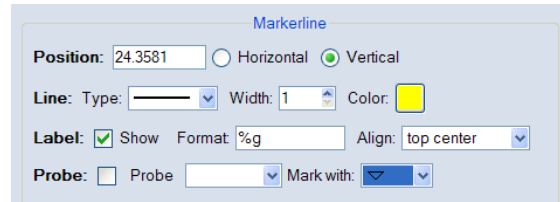


**Figure 17.14:** Editable marker line parameters.

of a curve. In this case you must insert the annotation after that curve object. You can also position an annotation in the plot window interactively by pressing the left mouse button on the annotation, moving the mouse, and then releasing it at the new position.

### 17.11.2.3 Editing legend parameters

There are three types of legends possible: A *Legend Block* displays an entry for each curve together with a short line depicting the appearance of the curves. A *Name Block* is just a list of the curve names. The position and orientation for both types can be manipulated easily in the editor window. The position denotes the coordinates of the first legend item. Positions of legends are always normalized coordinates. The delta parameter applies only to the vertical (y) coordinate. You can also position a legend in the plot window interactively by pressing the left mouse button on one of the names in the legend, moving the mouse, and releasing it at the desired position. The third legend type (*Name List*) displays a list of curve names, too. But in contrast to the latter type, you can move every name around separately, like you can with annotations (see above).

### 17.11.2.4 Editing marker line parameters

The position of a marker line must be given in world coordinates. Also be sure that the marker line is inserted after the appropriate axis. You can shift a marker line horizontally resp. vertically in the plot window by pressing the left mouse button on the marker line, move the mouse and release it at the new

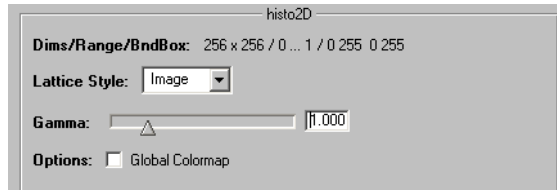


**Figure 17.15:** Editable lattice parameters.

position. Marker lines can be used to probe those curves which belong to the same group (PlotArea) as the marker line. You can display the value where the marker line intersects the selected curve the first time. This intersection can also be indicated with a marker symbol.

#### 17.11.2.5 Editing lattice parameters

A lattice is a two-dimensional field which is rendered as an image by default. If both dimensions are less than or equal to 32, a lattice can also be rendered as colored dots in a grid (*Gridded*) or as dots of different sizes (*Spot*) to represent the values. For lattices rendered as images, a gamma value can be set to enhance the image. Lattices have a colormap (Default: black to white) which can be set through the appropriate amira modules. Global colormaps are not supported within amira.

#### 17.11.2.6 Editing colormap parameters

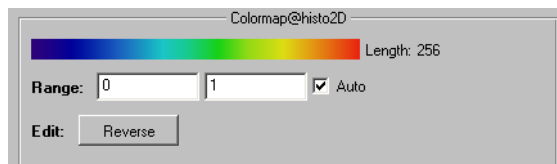


**Figure 17.16:** Editable colormap parameters.

Colormaps have only a few editable parameters: The minimum and maximum values can be given or taken from the axis if there is one. Furthermore the colormap can be reversed. The colors itself can only be changed with the amira colormap editor.

#### 17.11.2.7 Editing (analytical) curve parameters

Besides the more or less self-explanatory parameters of both the curve and analytical curve (AnCurve) parameters, there are a few things to mention regarding AnCurves only. The *Function* text widget contains the formula which is computed on every plot update. The default formula is  $x$ , which produces a transversal line where  $x$  runs from -1 to 1. It is possible to use the name of another curve as a variable in the function. In this case the  $x$ -values of the first curve in the formula are taken to evaluate the  $y$ -values of all variables in use. These  $y$ -values are then used for the computation. You can restrict the



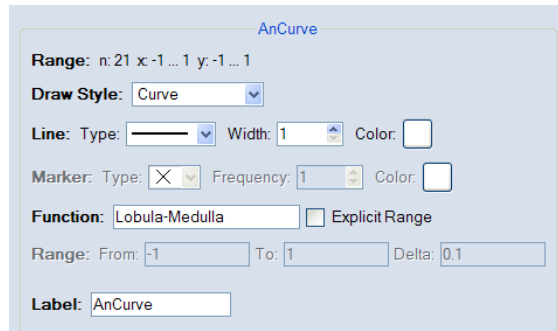


**Figure 17.17:** Editable (analytical) curve parameters.

range of the computation by activating the *Explicit Range* toggle and entering the appropriate range values. There are many built-in functions such as *sin*, *cos*, *tan*, *sqrt*, *exp*, ... which can be used in formulas.

### 17.11.3 Working with plot objects

You can copy or delete plot objects. For example, it may be useful for comparison purposes to copy a curve before the data of the original curve will be changed. To do so, you must select the object in the list and then choose the *Edit/Copy* pulldown menu. After that, choose *Paste* or *Append* in the menu and the object will be inserted at the current position (= position of the selected object) resp. the object will be placed behind the selected object.

With the *New* pulldown menu you can create new objects which will be inserted at the appropriate positions in the list of objects.

### 17.11.4 Printing

The *File* pulldown menu of the main plot window provides a *Snapshot* item where you can send the plot directly to the default printer or save it as an image file. It is also possible to generate a vector-based PostScript file of the plot by using the *print* command (see below).

### 17.11.5 Saving data

The *Save Data...* item under the *File* menu lets you save the data of all curves in a file. The file dialog presents a list of formats suitable for saving the data.

### 17.11.5.1 Data formats

The *Amira-Plot-Format* (.ampl) is a proprietary format which should be used if you want to plot the data later on within the amira plot facilities. Use the *Gnuplot-Format* (.dat) if you plan to use Gnuplot to plot the data outside of amira. If you need to import the plot data into a spreadsheet program like Microsoft's Excel, use the *CSV-Format* (CSV = Comma Separated Values) (.csv). The *Spreadsheet-Format* (.am) can be loaded and processed by amira.

### 17.11.6 Saving the plot state

If you changed your plot setup a lot, you can save this setup in a similar fashion like the *Save Network* functionality. That is, first save the amira network, and then save the plot state into a second file. To resume an amira session, invoke amira and load the two script files (1. amira network, 2. Plot state). Note that the size and the position of a plot window is saved automatically if it is opened by an amira module and the *Save Network* menu item is invoked.

## Commands

In amira, Plot Tool commands have the following structure:

`$thePlot command [ parameters ]`

or if the command applies to a plot object:

`$thePlot objectname command [ parameters ]`

The following general commands are available:

`getSize`

Returns the size of the plot window.

`setSize <width> <height>`

Sets the size of the plot window.

`getPosition`

Returns the size of the plot window.

`setPosition <x> <y>`

Sets the position of the plot window relative to the upper left corner of the screen.

`setBackgroundColor <r> <g> <b>`

This command sets the color of the background to a specific value. The color may be specified either as a triple of integer RGB values in the range 0...255, as a triple of rational RGB values in the range 0.0...1.0, or simply as plain text, e.g., white, where the list of allowed color names is defined in /usr/lib/X11/rgb.txt.

`hide`

Hides the plot window.

## **show**

Shows the plot window if it is hidden.

## **getObjects**

Displays a list of all plot objects currently in use.

## **update**

Processes the plot object and updates the display.

## **snapshot <filename>**

Takes a snapshot and saves it under the given name. The suffix of the filename determines the raster format used. Available formats are: TIFF (.tif, .tiff), SGI-RGB (.rgb, .sgi, .bw), JPEG (.jpg, .jpeg), PNM (.pgm, .ppm), BMP (.bmp), PNG (.png), and Encapsulated PostScript (.eps)

## **print [options] <filename>**

Prints the plot window into a PostScript file (vector based) with the following options:

- a4: generates a4 landscape output (Default).
- a5: generates a5 portrait output.
- auto: switches autoscaling on (Default).
- bw: generates black and white output.
- color: generates color output (Default).
- eps: generates encapsulated PostScript.
- fillbg: fills the background according to the plot window.
- frame: draws a frame around the plot.
- landscape: prints in landscape format.
- noauto: switches autoscaling off.
- portrait: prints in portrait format.
- window size: generates output of window size.

## **saveData <filename>**

Saves the data of all data based plot objects in a proprietary format.

## **saveState <filename>**

Saves the current plot state into a script file that can be used to restore the plot setup in a future amira session.

## **load <filename>**

Loads data from the filename and stores it in a curve plot object.

## **The following commands apply to plot objects:**

## **getMinMax**

Returns the minimum and maximum values of objects of type: *Curve*, *Cartesian Axis*, *Polaraxis*.

`setMinMax <minX> <maxX> <minY> <maxY>`

Sets the range of *Cartesian Axis*, *Polaraxis*.

`getArea`

Returns the plotting area of *PlotArea* objects.

`setArea <lowerleftX> <lowerleftY> <upperrightX> <upperrightY>`

Sets the plotting area of a *PlotArea* object.

`getXValues`

Returns the x-values of a *Curve*.

`getYValues`

Returns the y-values of a *Curve*.

`set[X|Y|Phi|R]TickValues <1. tick> <2. tick> ... <n. tick>`

Sets the position of the ticks along an axis. It is available for objects of type: *Cartesian Axis*, *Polaraxis*.

`set[X|Y|Phi|R]TickLabels <1. label> <2. label> ... <n. label>`

Sets the labels of the ticks along an axis. It is available for objects of type: *Cartesian Axis*, *Polaraxis*. If there are fewer labels than ticks the remaining ticks will be unlabeled.

## 17.12 Segmentation Editor



The *Segmentation Editor* is a tool for interactively segmenting 3D image data. Image segmentation is the process of dividing an image into different subregions (also called segments). In the bio-medical context these segments can be for example different organs or tissue types, e.g., materials. In *amira*, segmentation is done by first selecting voxels and then assigning these voxels to a particular material. The labels are stored in a *Label Field*. The *Segmentation Editor* lets you edit such a label field. From the final label field polygonal surfaces can be reconstructed using the *SurfaceGen* module. The documentation of the *Segmentation Editor* is separated into the following parts:

- *Overview of the segmentation editor*
- *Manipulating the material list*
- *Working in 4-viewer mode*
- *Selection tools*
- *Segmentation tools*
- *Selection filters*

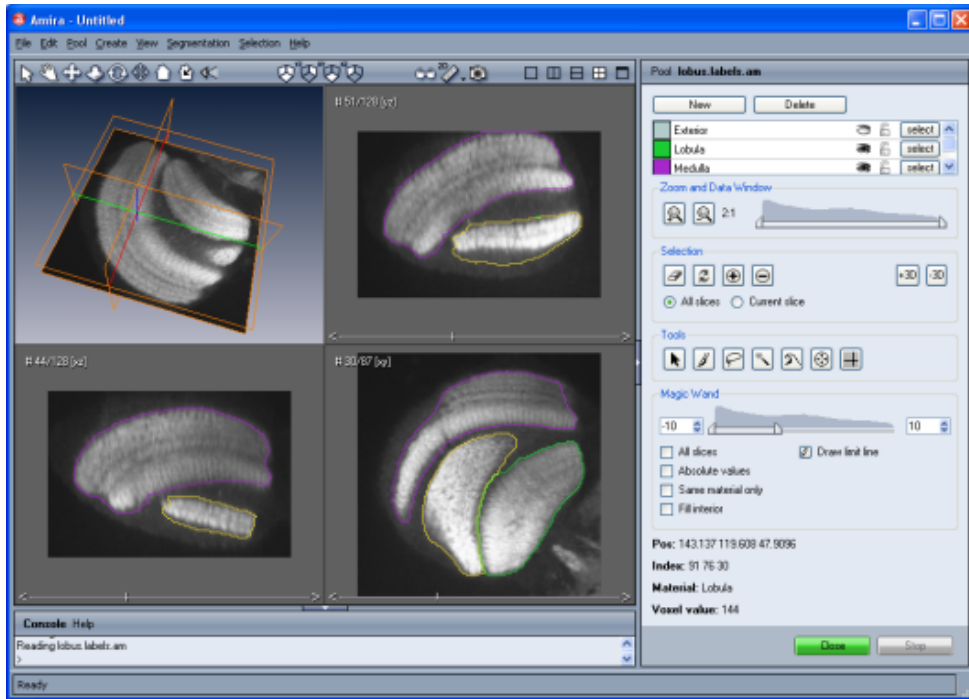


Figure 17.18: The Image Segmentation Editor.

- *Label filters*
- *Key bindings*

### 17.12.1 Overview of the segmentation editor

In order to activate the segmentation editor, press the Image Segmentation edit button of a selected *Label Field*. Then an instance of the editor will be created and a new window as shown in Figure 17.18 pops up. The major parts of this new window are:

- **Material List:** In the upper right corner of the editor window a list of materials is presented. Here you can add, remove, and rename materials, and you can select the current material.
- **Tool Box:** Below the material list several tools for interactive manipulation of the segmentation can be selected. Depending on the currently selected tool additional widgets show up in the options frame.
- **Info Area:** Below the tool box some basic information like the current cursor position or the material under the cursor is displayed.

- **Menu Bar:** From the menu bar additional tools and filters can be accessed. Menu entries with dots (...) after the name open an additional dialog window.
- **Image Viewer(s):** The biggest part of the window is covered by one or four image viewers, displaying the labels and the current selection in differently oriented slices.

By default, the viewer is displayed in *4-viewer* mode. Three 2D viewers and a single 3D viewer are displayed. If you prefer to work with one larger view rather than four smaller views, click on the *Layout1* button in the viewer toolbar. To cycle through each of the four views, press the *Layout1* button repeatedly. To return to 4-viewer mode, press the *Layout4* button.

### Basic principle of interactive segmentation

The basic idea of interaction in the segmentation editor is to first *select* some voxels and then to *assign* them to the *active material*. The simplest way of selecting pixels is to simply draw with the mouse when the *Brush* or the *Lasso* tool is active (see below for details). Selected pixels are displayed in red.

To add selected pixels to the active material, click the "+" button. The active material is the material, which is currently selected in the material list. New materials can be added by pressing the *New* button above the material list. Every pixel can only be assigned to one material.

Note that even with the advanced tools provided in *amira*, image segmentation can be a time-consuming process! Due to limited main-memory and for performance reasons, there is only a limited undo space for 2D interaction and most 3D operations cannot be undone. Therefore it is highly recommended to *frequently save the label field* during the process of segmentation.

To get started it might be a good idea, to choose the *image segmentation demo* from the Users Guide's demo section and start by modifying an existing label field.

### The menu bar

Two segmentation-specific menus are added to *amira*'s main menu bar: *Segmentation* and *Selection*. The entries of the *Selection* menu are described in a separate section below. Menu entries with dots (...) after their name open an additional dialog.

The items of the *Segmentation* menu are described here:

- **Undo:** This entry undoes the last operation. Successive invocation of *undo* is possible, allowing you to undo several operations. Note that for memory and performance reasons, 3D operations can not be undone. Therefore it is highly recommended to frequently save the label field.
- **Data window...:** This entry brings up an additional dialog, in which you can control the data window and the opacity of the selection.
- **Orientation:** You can use this to select the orientation of the active viewer. You can also flip the x, y, or z coordinates of all of the viewers.
- **TissueStatistics:** This entry brings up a dialog which lists the results computed by the TissueStatistics module on the current data set and segmentation. For an explanation of the modes, see the documentation of the *TissueStatistics* module.

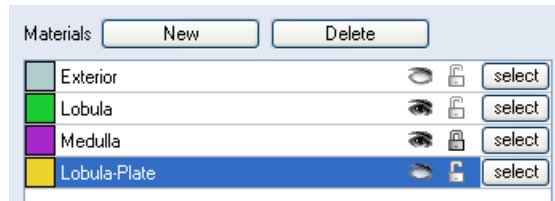


**Figure 17.19:** The material list.

- **Fill holes:** Invokes a label filter. See *Label filters*.
- **Remove islands...:** Invokes a label filter. See *Label filters*.
- **Smooth labels...:** Invokes a label filter. See *Label filters*.
- **Gray image:** Selects the data set to be displayed in the viewers.
- **Gradient image:** Selects the gradient that will be used by some of the tools of the segmentation editor, for example, the Blowtool.

## Preferences

It is possible to specify various preferences in the *Segmentation* tab of the *Edit/Preferences* menu of the main menu bar.

- **Slice Position:** Allows you to switch on and off display of the borders of the slices in the 3D viewer window.
- **Show 3D:** When checked on, the currently selected voxels are displayed in the 3D viewer window.
- **Points1, Points2, Solid:** These radio buttons allow you to choose the draw style used for drawing the currently selected voxels in the 3D viewer window.

### 17.12.2 Manipulating the material list

To select a material in the material list, click on the material name with the left mouse button. The selected material will be highlighted in dark blue.

Press the *New* button to add a new material to the list. To delete a material, select a material, then press the *Delete* button.

Click on the color square to the left of the material name to bring up a *color dialog* to edit the color.

Click on the eye icon to the right of the material name to switch the material visible or invisible in the segmentation editor. An empty eye means invisible, a normal eye means visible.

Click on the lock icon to lock or unlock the material. If a material is locked, no voxels can be subtracted from this region, neither explicitly, nor implicitly by adding them to another material. The lock icon

indicates the current state (locked, unlocked).

Press the *select* button to select all pixels of the corresponding material.

The current material is the material which is assigned to selected voxels if the "+" button is pressed. This material is displayed with a gray highlight. In the material list the current material can be selected by clicking on the material name with the left mouse button. A special behavior is obtained if one of the *Shift*, *Control*, or *Alt* keys on the keyboard is held down while selecting the material:

- If *Shift* is held down, all voxels which are already assigned to the selected material are added to the current selection (either in the current slice or in the whole volume, depending on the value of the 3D toggle described below).
- If *Control* is held down, all voxels assigned to the selected material will be deselected (either in the current slice or in the whole volume, depending on the value of the 3D toggle described below).
- Holding down the *Alt* key does the same as *Shift* except that the selection is cleared before new voxels are selected.

Additional options of the material list can be accessed via a popup menu, which shows up when the right mouse button is pressed on a particular material entry. The following menu entries are provided:

- **Draw Style:** A submenu offers different styles how pixels belonging to a particular material are rendered in the image viewer. The possible appearances are:
  - *Invisible:* The material is not visible. You can also control the visibility by clicking on the material's eye icon of the material list.
  - *Contour:* The segments are enclosed by lines.
  - *Hatched:* The segments are enclosed by lines and shown hatched.
  - *Dotted:* The segments are enclosed by lines and filled with dots.
  - *Light Dots:* The segments are enclosed by lines and filled with less dots.
- **Locate:** This command sets all viewers to the slice with the largest region of this material. This is especially useful to navigate in large data sets with many materials.
- **Delete Material:** If you choose *Delete*, the material you have clicked on will be deleted. Voxels belonging to that material will be assigned to the first material in the list (typically the background). You can also delete a selected material by pressing the *Delete* button above the material list.
- **Rename Material:** With this option you can change the name of a material. A text cursor appears and lets you edit the name as easily as any other text field. If the new name is equal to the name of an existing material you are asked if these two materials should be merged. It is not possible to have different materials with the same name.
- **Edit Color:** This command brings up a *color dialog* in which you can easily change the color of this material. You can also access the color editor by clicking on the material's color square



in the material list.

- **Lock/Unlock Material:** You can lock a material with this option. A lock icon to the right of the material indicates its current state (locked/unlocked). If a material is locked, no voxels can be subtracted from this region, neither explicitly, nor implicitly by adding them to another material. You can also lock/unlock the material by clicking on the material's lock icon in the material list.
- **New Material:** By selecting this option from the popup menu a new entry is added to the material list with a randomly chosen color and a default name, e.g., "NewMaterial". These properties can be changed as described above. You can also create a new material by pressing the *New* button above the material list.
- **Lock All:** This locks all materials.

### 17.12.3 Working in 4-viewer mode

Initially, in each 2D image of the image segmentation editor the current slice with colored contours surrounding the different segments is shown. Near the bottom of the image you find a slider to control which slice of the 3D volume is currently displayed. The slice number and the total number of slices is displayed in textual form at the top of each 2D slice window. The orientation of the slice can be changed via the menu entry *View Orientation* in the menu bar. In medical notation XZ denotes an axial slice, XZ a frontal slice, and YZ a sagittal slice.

A viewer can be made the active viewer by clicking in the viewer itself. The last window you worked in will be active. Be careful when clicking into the image in 3D mode. If the *Pick* tool or the *Magic Wand* tool is active the resulting selection operation may take some time to compute. Use *Segmentation/Undo* to remove any unwanted selections. All 2D operations like the "+" or "-" are always performed in the active viewer, i.e., the slice and orientation of the active viewer will be considered.

#### The 3D viewer

The upper-left viewer is a standard amira 3D viewer. All modules will display in this viewer too (depending on the modules' viewer masks, as usual). In addition there is some special 3D support for segmentation, which is controlled by the +3D and -3D buttons in Selection tools area.

By default, the current selection will be displayed in the 3D viewer as a point cloud. After each change of the current selection (i.e., after each brush stroke), the display will be updated. On slower computers or for very large data sets, this may become slow. Switch off the 3D check box of the *Segmentation* tab of the *Edit/Preferences* menu to disable display of the highlighted region in the 3D viewer.

The *Draw ROI* buttons allow you to modify the selection in 3D. You can add or crop 3D regions by drawing in the 3D viewer. See the *Selection tools* section for details of the operation of the +3D and -3D buttons. Note that the shape of the added or cropped region will also depend on the camera mode you are in, orthogonal or perspective. For a related tool for editing volumes, see *VolumeEdit*.

#### Using the zoom buttons

The zoom buttons control the size of the image. An info line on the right hand side of the buttons shows the current zoom factor. For example a zoom of 2:1 means that 2 pixels on the screen correspond to

one pixel of the original data set (magnification), while 1:4 would mean that four pixels of the data set correspond to one pixel on the screen. If the pixel size is not the same in all dimensions, the zoom factor belongs to the horizontal direction only, to maintain the correct aspect ratio of the voxels.

### Data range and histogram

The histogram displays the distribution of gray values in the gray value image. Use the sliders below it to adjust the data range for the display. You can adjust the range by clicking and dragging the right and left edges of the slider bar. You can adjust the entire range at once by clicking and dragging the center of the slider bar, which will adjust the brightness.

## 17.12.4 Selection tools

The basic principle of the segmentation editor is to first select voxels and then to assign the selected voxels to the current material. This assignment is done using the "+" button. Besides the "+" button, there are some other buttons which can be used to perform selection operations. The complete set of buttons follows:

**3D toggle:** Radio buttons let you choose to operate on *all slices* or just the *current slice*. If the 3D toggle is activated, i.e., you have selected *all slices*, the selection tools in this group operate in 3D mode. For example, the *Clear* button clears the whole 3D selection, not only the current slice. Currently, in 3D mode no undo is available. Therefore, be careful when activating the 3D toggle. Usually, it is advisable to disable the toggle again after completing a particular 3D operation.



**Clear:** This button lets you clear the current selection. If the 3D toggle is activated, the selection is cleared in all slices.



**Replace:** This button tries to replace a selected region. Assume you want to modify the contours of some region. As long as the new contours fully enclose the previous ones this can be easily achieved using the "+" button. However, if the new region is smaller, things are more complicated. The replace button looks for connected regions under the current selection belonging to the current material. Pixels which belong to such a region but which are not selected are automatically assigned to some other neighboring material. Selected pixels are assigned to the current material. In this way a replacement is performed.



**Add:** This button adds all selected voxels to the material currently selected in the material list. Voxels assigned to a locked material are not affected. If the *Shift* key is held down while clicking the button, the voxels remain selected. Otherwise the selection is cleared afterwards.



**Subtract:** This button lets you subtract all selected voxels belonging to the current material from that material, provided the current material is not locked. The pixels are automatically assigned to some neighboring material which is interpreted as a local background.



**Draw ROI Add:** This button selects voxels to be added to the current highlighted region. Click on this button, holding the left mouse button pressed down, and drag the mouse in the 3D viewer to

draw. When you release the mouse button, the voxels are added to the current selection. If the *Control* key is pressed when starting to draw, the inverse of the surrounding region will be selected. This draw tool can also be activated by pressing the *d* key in the viewer window in interaction mode.



**Draw ROI Subtract:** This button selects voxels to be removed from the current highlighted region. Click on this button, holding the left mouse button pressed down, and drag the mouse in the 3D viewer to draw. When you release the mouse button, the voxels are removed from the current selection. If the *Control* key is pressed when starting to draw, all 3D points which lie outside the surrounded region will be cropped.

### 17.12.5 Segmentation tools

Several segmentation tools are provided allowing you to select areas in the current slice for subsequent edit operations. A tool is activated by clicking on its icon with the left mouse button. By default, selected areas are drawn in transparent red color. The opacity of the red color can be modified using a dialog under the *Segmentation Data Window...* menu. Alternatively, selected areas can be visualized using the same draw styles as ordinary labels. The different draw styles are listed under the *Selection/Draw style* submenu. However, in most cases the default transparent display is most appropriate because then selected areas cannot be mistaken for labeled regions.

Once a segmentation tool is activated, certain actions can be performed in one of the viewer windows using the mouse. The operation of most of the tools can be modified by pressing the *Ctrl* or the *Shift* key. *Ctrl* typically deselects voxels instead of selecting them, while *Shift* adds voxels to the selection instead of replacing it. The descriptions of the individual tools provide more details.



#### **Pick & Move**

This tool does two things. First, it lets you select a connected region assigned to one particular material by clicking on an image voxel with the left mouse button. If the *select all* toggle is active, all voxels belonging to the same material as the clicked voxel will be selected, either in 3D or in the current slice only, depending on the value of the 3D toggle. If the *Ctrl* key is held down, voxels will be deselected.

If the mouse is over an already selected pixel, the mouse cursor takes on the shape of a hand. Then the tool lets you translate the current selection. By pressing down the *Shift* key you can rotate the current selection. However, note that this might give strange results for pixels with an aspect ratio different from one (for example in frontal or sagittal slices in a label field with stacked coordinates).

When *All slices* is checked, your selection will apply to all slices, not just the current slice. Currently, in 3D mode no undo is available. Therefore, be careful when activating *All slices*. Usually, it is advisable to deactivate the toggle again after completing a particular 3D operation.



#### **Brush**

If this tool is activated, you can select regions by painting voxels with the left mouse held down. The size of the brush can be modified via a slider in the tool's control panel. For common smaller sizes

dedicated push buttons are provided. Note that the size is specified in screen pixels, not in image pixels. If the center of an image pixel is covered by the brush, that pixel is selected. If the *Ctrl* key is held down, pixels are deselected instead of being selected.

The right mouse button is a flood fill tool. For example, you may surround a region in the image by drawing along its border and then click into the middle of the currently unselected interior part to fill it. Again the *Ctrl* key inverts the operation, i.e., it deselects connected parts of the selection.



### Lasso

The *Lasso* lets you define an area by generating a closed contour curve. Usually, you draw such a curve freehand with the left mouse button pressed, but it is also possible to fence an area with line segments by pressing *Alt*, and clicking successively to points with the left mouse button (holding down the *Alt* key all the time). Successive points will be connected with straight line segments. To finish interaction, release the *Alt* key and click a last time. The contour is then closed and filled automatically.

When the *auto trace* option is enabled, the line segments are automatically fitted to image edges. Click with the left mouse button to define a starting point. Dragging the mouse after releasing the button a contour curve automatically fitted to the closest image edge is drawn. Further mouse clicks mark fixed points on the contour. Contour parts drawn before the latest mouse click remain in place. Close the contour by clicking with the middle mouse button.

The underlying algorithm called *intelligent scissors* works by finding the shortest path in a cost matrix which is computed from the image's gradient image. If the option *trace edges* is switched off, then the image itself is interpreted as a cost matrix. Disabling this option is useful for images where object boundaries already appear as isolated bright lines.



### Magic Wand

This tool performs a so-called region growing either in 2D or in 3D, depending on whether the 3D toggle is activated or not. Clicking with the left mouse button on a voxel selects the largest connected area that contains the voxel itself and all voxels with gray values lying inside a user-defined range. The range can be specified via two spin boxes shown in the tool's control panel. The values of the spin boxes either define absolute gray values or values relative to the gray value of the seed pixel, depending on whether the *absolute values* toggle is activated or not. If necessary, the range will be automatically modified so that it contains the seed voxel. It is possible to modify the range even after the seed voxel has been selected. For convenience, the lower threshold can be quickly modified by clicking the middle mouse button (or shift-clicking the right mouse button) and moving the mouse horizontally. Likewise, the upper threshold can be quickly modified by clicking the right mouse button and moving the mouse horizontally (*virtual slider mode*). However, note that in 3D mode the selection is not updated on-the-fly while modifying the thresholds. Instead, it is updated once after releasing the mouse.

The *same material only* toggle restricts the search to voxels which are assigned to the same material as the seed voxel. For example, if the seed voxel belongs to *Exterior*, voxels which are already assigned to different materials will not be selected. This is useful to restrict the selection to certain parts of the

image. Paths out of a region of interest can be obstructed using a blocker region.

If the *fill interior* toggle is set, holes inside the selected region will be filled automatically. This has the same effect as pressing the *f* key afterwards, which also fills the current selection. Automatic filling is only done in 2D mode, but not in 3D mode.

Finally, the *draw limit line* button lets you specify additional barriers for 2D region growing. After pressing this button you can draw arbitrary polylines in the viewer window. Voxels covered by such a limit line will not be considered for region growing. Instead of pressing the limit line button you can also press the *Ctrl* key to temporarily enter the limit line mode. Existing limit lines can be deleted by clicking on them with the *Ctrl* key being pressed.

Note that all seed points together with the corresponding range values and limit lines are stored in the parameter section of the label field. This makes it possible to easily correct a selection at a later time.

When *All slices* is checked, your selection will apply to all slices, not just the current slice. Currently, in 3D mode no undo is available. Therefore, be careful when activating *all slices*. Usually, it is advisable to deactivate the toggle again after completing a particular 3D operation.



### Propagating Contour

This tool is a fast *Active Contour*. From a number of specified seed points a boundary front evolves. It computes the position of the front at all times up to the stop time. A slider and a text box allow the user select the appropriate front after pressing the initialization button. There are three buttons: *Menu* pops up the parameter menu, *Clear* removes the current seed points (hot-key *T*), and *Init* starts the computation (hot-key *B*).



### Blowtool

When you click with the left mouse button on a voxel and drag the mouse without releasing the button, an initially circle-shaped contour blows up. The greater the distance from the initial position of the mouse click, the more the contour grows. The contour is designed to grow in areas with homogeneous gray values and to stop where gray values change abruptly, i.e. at image edges.

The tolerance slider in the *option panel* controls how sharp the image edge has to be in order to stop the contour from growing at each contour point. The smaller the tolerance, the sharper the image edge has to be. If the tolerance is large, the contour will even stop at weak edges. After you release the mouse button, the area within the contour will be marked.

Before the computation, the image is smoothed using a Gaussian smoothing filter. You can change the width of the filter within ranges of 1 (no smoothing) to 8 (very broad smoothing). The default value is 4.



### Crosshair

It simply displays a crosshair in all orthogonal viewers. The colors denote the different directions (red=x-axis, green=y-axis, blue=z-axis). Dragging axis moves the crosshair and simultaneously updates the other viewers so that the crosshair's center is visible in all viewers.

### 17.12.6 Selection Filters

Under the *Selection* button of the menu bar several filters for modifying the current selection are provided. These include simple morphological operations, smoothing filters, and interpolation tools.

- **Grow:** This filter performs a morphological dilatation of the current selection, i.e., the selection is made bigger by one pixel in every direction. The filter can be applied to the current slice only (shortcut is *Ctrl +*), to all slices (orientation is defined by the active viewer), or to the whole volume.
- **Shrink:** This filter performs a morphological erosion of the current selection, i.e., the selection is made smaller by one pixel in every direction. The filter can be applied to the current slice only (shortcut is *Ctrl -*), to all slices (orientation is defined by the active viewer), or to the whole volume.
- **Fill:** This filter fills the current selection, i.e., it closes all holes in it. The filter can be applied to the current slice only (shortcut is key *F*), or to all slices (orientation is defined by the active viewer).
- **Smooth:** This filter smooths the current selection. Smoothing works by applying a Gauss filter to the binary selection image and then reselecting all pixels with an intensity bigger than 0.5. Usually the filter is applied to a selection computed by the threshold filter or by the magic wand tool. The shortcut for this filter is *Ctrl M*.
- **Invert:** This filter inverts the current selection, i.e., selected voxels are deselected, while unselected voxels are selected. The filter can be applied in the current slice only (shortcut is key *I*), or in the whole volume.
- **Active Contour...:** This filter belongs to a family of powerful tools for semi-automatic segmentation called *Active Contour*. From a user-defined initial selection the contour propagates automatically according to "forces" within the image. Use the *Stop* button of the *amira* main window to stop the process when the desired selection has been computed.
- **Snakes...:** This filter automatically fits the contour of a selected region to the edges of the image. Edges are regions with a high contrast.

An additional dialog is opened, which allows you to copy the selection from the current slice into the next or previous slice (up and down arrows). After copying, the snakes algorithm tries to adjust the contour to the gray values in the new slice.

Clicking on *Adjust* runs the same algorithm on the current selection in the current slice. This is especially useful to readjust after a manual correction.

If *Extrapolate* is switched on, the selection is not only copied to the next slice but extrapolated by taking into account selections in the two previous slices.

If *3DInfo* is switched on, a modified snakes algorithm is used which gives better results in the case of relatively equal voxel sizes in all three directions.

*Relax* smooths the contour of the current selection.

- **Threshold...:** Select voxels by threshold segmentation. In the additional dialog, a minimal and a maximal image intensity can be specified. When you click on *OK*, all voxels falling into the interval are selected. If the *same material only* option is activated, only voxels assigned to the current material are selected. If the *3D* option is active, the operation is performed in the whole volume, otherwise it is done in the current slice of the active viewer.
- **Make Ellipse:** Pressing this button causes the selected area to be replaced by an ellipse that fits the original selection as well as possible. The filter operates on the current slice of the active viewer.
- **Interpolate:** This button interpolates the selection between all slices where areas have been selected manually. Pressing the button again after performing corrections in certain slices will interpolate the selections between all slices where changes have been made. For example, suppose you have selected an object in slice 1 and in slice 10. Pressing the *Interpolate* button automatically computes a selection in slices 2...9. You may then modify the selection in slice 5, e.g., using the *Brush* tool. Pressing *Interpolate* again now interpolates the selection in slices 2...4 and 6...9. The shortcut for this filter is *Ctrl I*.
- **Wrap:** This filter also interpolates the selection. However, a different algorithm based on scattered data interpolation with radial basis functions is applied. The filter always operates in 3D. In contrast to the previous filter, it is possible to start from slices with different orientations, e.g., after selecting slices in each spatial direction (xy, xz, yz) this tool can be used to compute an interpolating selection which wraps up the object whose skeleton is given by the slices.  
**Note:** Initially no two successive slices with the same orientation must be labeled. Otherwise, an error message is displayed. The tool may take a few seconds to complete its operation. The shortcut for this filter is *Ctrl W*.

### 17.12.6.1 Active Contours

Active contour models are powerful tools for semi-automatic segmentation. An initial contour moves automatically according to image forces such that it separates two different materials.

The movement is governed by parameters that the user must adjust to suit his image data.

There exist two different Active Contour tools. The *Propagating Contour* tool computes the shape of the contour at all times up to the *Stop Time* value. The contour starts evolving from one or more seed points and stops at edges found in the image. After the computation the user can choose any time parameter in the range the separates the materials best.

The Active Contour filter takes any initial shape (which may only be a single seed point) and expands this shape until a material boundary is found, where the contour stops. Additionally, this tool takes into account the curvature to generate smoother shapes and avoid leaking through small holes.

The velocity of the active contour is computed from the original image data as well as from the gradient image. While the image data provides the image intensity values, the gradient image contains the

edge information. In the *Segmentation* menu, the user can choose a gradient image. By default, the Segmentation Editor computes a gradient image from the currently chosen gray image. Alternatively, the user can provide the gradient image and select it in the *Segmentation* menu.

**If the active contour does not advance, you have to change the parameters.** In particular, lower the values for edge and image sensitivity.

Note that often the results can be improved by preprocessing the image data, for instance by pre-smoothing. Furthermore, better results are achieved for isotropic data, i.e., large slice distances can be disadvantageous.

## Ports

### Stop Time

The *Stop Time* port limits the movement of the active contour to a certain range. If the region has not been fully segmented, this parameter must be increased.

### Edge Sensitivity

The *Edge Sensitivity* value governs the dependence on the the gradient image, i.e., when the contour stops. If this value is too small, the algorithm might run over edges. If it is too large, there might be no movement at all. If the contour does not move at all, this value should be decreased.

### Image Intensity Weight

The advancement of the contour can also depend on the image intensity (gray value). This parameter controls how strong the dependence is. The speed function is multiplied by a Gaussian function. The width of this Gaussian is the reciprocal of the value of this port (such that a value of 0 means no influence at all) and the mean intensity value if computed as the average over all seed point or over the average of the selection, respectively. Hence, the more different the image intensity of a voxel is from the starting points or region, the slower the progress is.

### Curvature

The *Curvature* parameter controls the regularity of the contour. A value close to 1 leads to a smooth contour. Set to 1, the algorithm works as a filter that refines an existing selection in order to snap to edges in the image. If this parameter is set to 0, the algorithm is merely expanding until stopped by an edge without enforcing any smoothness. Setting the curvature to a value different from 1 can be used to make a small seed selection expand until stopped by an edge. The *curvature* regularization is not available for the propagating contour tool.

### Attractor



The *Attractor* force parameter specifies the strength of the edge attraction. It allows the tool to be used as a filter that refines an initial approximation because it attracts the contour to the nearest edges in the image. Mostly, a value of 1 works best. To switch off the edge attraction, set the value to 0.

## Mode

The *Mode* port lets you choose whether the contour grows in 2D (*slice*), or 3D (*3D volume*).

### Practical hints:

- In all cases, apply these tools to smoothed data only. No smoothing is built in to allow the user to choose the smoothing filter. For instance, Gaussian smoothing is an appropriate preprocessing.
- It is recommendable to use the fast *PropagatingContour Tool* first and then refine the selection with the *ActiveContour Filter*.
- In cases where not all the targeted region is selected by the active contour or it leaks out before, the user can proceed step by step. First select a partial region and then segment the remaining region starting from another seed point or selection.
- At the beginning, certainly a bit of probing is necessary until you find the best set of parameters for your kind of data.

## 17.12.7 Label Filters

Under the *Segmentation* button of the menu bar several filters for modifying the current labeling are provided. These filters operate directly on the labels. They do not take the current selection into account.

- **Fill holes:** This filter removes islands of arbitrary size in the *current material*, i.e., all pixels completely surrounded by the current material are also assigned to this material (short cut is *Shift-H* for the current slice). Suppose you have segmented a CT image using thresholding as provided by the *LabelVoxel* module. You then want to assign the dark marrow of the bones to the material *Bone* as well. In order to do this, select *Bone* in the material list so that it becomes the current material and then call the *fill holes* filter. The filter can be applied to the current slice in the active viewer, or to all slices.
- **Remove islands...:** This filter removes islands in or between regions, depending on the *size* and *percentage* values that are set in the filters dialog. An island is a connected area of voxels containing a number of voxels less than or equal to the *size* value specified in the dialog. If an island is encountered, the percentages of the surrounding regions with respect to the total number of voxels adjacent to the island are calculated and compared with the *percentage* threshold. The island is assigned to the region with the highest percentage value greater than or equal

to *percentage*. If no region exceeds the *percentage* threshold, the island remains untouched. Note that the *percentage* value must be between 0.0 and 1.0.

With the *mode* buttons you can control whether the filter works on the current *slice*, on *all slices*, or on the *3D volume*. The difference between the two latter cases is that in 3D mode the filter searches for true 3D connected regions. Note that a long thin structure like a blood vessel can be a very small region in each slice, but has a rather large volume in 3D.

This filter is invoked by pressing the *Remove* button. Pressing the *Select* button performs the same computations as before. However, the islands are not yet removed but are only selected. This is useful in order to preview the effect of the filter and to find optimal size and percentage values.

- **Smooth labels...:** This is a modified Gauss filter that smoothes the region boundaries, and therefore slightly changes the labeling. In 3D mode, additionally probabilities of voxels belonging to regions are computed and assigned to voxels. The probability is one for voxels in the interior of a region and decreases towards the region boundary. Voxels near region boundaries are also assigned probabilities with respect to neighboring regions. The probability information is used in other modules - in particular in the *SurfaceGen* module - in order to make region boundaries appear smooth, otherwise these may be displayed as zigzag lines.

The *size* option allows you to enter the size of the filter mask which is a square with *size* x *size* pixels. The smoothing effect increases with the mask size, but computation time is higher for bigger masks.

The *2D* button lets you start the smoothing operation on the present slice, the *3D* button lets you start a 3D smoothing on the whole data volume.

- **Remove Skin...:** This filter is only active if the Tcl variable `giRemoveSkinFilter` is set to 1. Its purpose is to remove thin prolate regions of muscle or fat (skin) on the boundary to exterior.

The filter removes the currently chosen material on the boundary to exterior. No other material than muscle or fat can be removed.

The *Layers* option is for entering the number of layers (in units of pixels) outside of exterior (i.e. inside the body) where the filter looks for possible appearances of the chosen material. The larger the number of layers, the more of the skin region will be removed.

The filter can be applied to the current slice in the active viewer, or to all slices.

### 17.12.8 Key bindings

Important tools of the image segmentation editor can be accessed via hot keys. These hot keys are summarized below:

- **Changing the current slice**  
**Space** or **CursorDown** - Go to next slice  
**Backspace** or **CursorUp** - Go to previous slice

**PageDown** - Go forward five slices

**PageUp** - Go backward five slices

**Home** - Go to the first slice

**End** - Go to the last slice

By holding down the *Shift* key while pressing one of these keys the current selection is copied to the target slice. By holding down the *Control* key, the user can move only between slices that contain a selection.

- **Selection**

**A** or **+** - Add the selection to the current material

**S** or **-** - Subtract the selection from the current material

**R** - Replace current material under the selection

**C** - Clear the selection

**E** - Extrapolate the selection

**F** - Fill the selection

**I** - Invert the selection

**Ctrl+** - Grow selection in current slice

**Ctrl** - Shrink selection in current slice

**Ctrl-M** - Smooth selection in current slice

**Ctrl-I** - Interpolate selection between multiple slices

**Ctrl-W** - Wrap selection using radial basis functions

**Shift-S** - Save label field to disk

**RETURN** or **Enter** - Redraw

.

- **Tools**

**1** - Brush tool

**2** - Lasso tool

**3** - Magic Wand tool (region growing)

**4** - Blow tool

**5** - Crosshair tool

**6** - Pick, move, and rotate tool

- **Others**

**U** - Undo

**Q** - Toggle 3D button

**V** - Toggle 1- and 4-viewer mode

**.** - Select next material in the list

**,** - Select previous material in the list

**Z** - Decrease zoom factor

**Shift-Z** - Increase zoom factor

**D** - Toggle draw styles for all materials

**Shift-D** - Toggle draw style for region under cursor

**Alt+D** - Toggle 3D display of region under cursor (4-viewer mode only)

## 17.13 Simplification Editor



This editor can be used to reduce the number of triangles of a surface. In particular, the output of the *SurfaceGen* module has to be processed in this way before a tetrahedral patient model can be generated. Surface simplification is done by means of an edge collapsing algorithm. Edges of the original surface are successively reduced to points. The shape of the original surface is preserved by minimizing a certain error criterion. Special care is taken to prevent the triangles of the simplified surface from intersecting each other. However, in some cases intersections can still occur. Therefore, the resulting surface should be checked for intersections using the *surface editor*.



Figure 17.20: User interface of surface simplification editor.

### Ports

**Simplify** This port provides three text fields for controlling some parameters of the simplification process. Field *faces* determines the desired number of triangles of the simplified surface. You may simplify the surface in multiple steps. However, somewhat more accurate results are obtained if the simplification is performed in a single step. When the simplification process is finished, a check is made whether the surface contains duplicated triangles. Such triangles are removed automatically. Therefore the total number of triangles of the reduced surface will usually be somewhat less than the value specified in *faces*.

The second field, *max dist*, defines a maximum edge length for the triangles of the simplified surface. Usually, the default value of 3 cm should be suitable, but you can try to modify this value if the reduced surface still contains intersections.

The third field, *min dist*, defines a minimum edge length. Shorter edges are contracted if button *Contract Edges* is pressed (see below).

**Options** If the toggle *preserve slice structure* is set, edges of *exterior* triangles of the surface are treated in a special way, so that the slice structure of the original voxel grid is preserved.

If toggle *fast* is set, a less extensive intersection test strategy is selected. Surface simplification will run faster, but there is a higher probability that intersecting triangles will occur (see explanation of command *setIntersectionTestStrategy* below).

If toggle *create level-of-detail* is set, all intermediate steps (edge-collapse) of the simplification process are stored in the surface in such a way, that they can rapidly be restored. To this end, the surface receives a port *Level of detail*, where the desired level can be set.

**Action** Button *Simplify* starts surface simplification. The simplification process will take several minutes. You may interrupt it by clicking the stop button of the progress bar.

Button *Flip Edges* automatically flips some edges of a surface if this improves the triangle quality. The Quality is defined as the ratio of the circumscribed circle and the inscribed circle of a triangle. The smaller this ratio the higher the quality. Again, duplicated triangles are removed automatically after this operation.

Button *Contract Edges* contracts all edges shorter than the value defined at field *min dist* (see above).

## Commands

`Simplifier setRadiusRatio <value>`

This command defines a quality threshold for the edge flipping algorithm. Edges are flipped only if the radius ratio of a triangle exceeds the given value. By default, a radius ratio of 20 is used.

`Simplifier smooth <nSteps> <lambda>`

This commands shifts the vertices of the surface so that it gets smoother. The value for *lambda* should be in the range of 0...1. This option is experimental and should not be used for routine work.

`Simplifier setIntersectionTestStrategy <mode>`

This command lets you choose between different intersection test strategies. *mode* is a three-digit number, for example 100. The digits specify the strategy used for testing modified triangles against existing edges, for testing modified edges against existing triangles, and for testing planar intersections. The allowed values are 0-3 for the first, 0-2 for the second, and 0-1 for the last digit. Larger values correspond to more extensive tests which of course also require more computing time. By default, a value of 211 is used. Modes 111 or 101 are faster but more likely to produce intersecting triangles.

`Simplifier setFactError <value>`

In surface simplification the maximal edge length and the maximal error (estimated distance between original and simplified surface) can be controlled separately. The maximal error is computed by multiplying the maximal edge length by a certain factor *factError*. The value of *factError* can be set using command *setFactError*. The default value is 1.

`Simplifier getFactError`

This command returns the current value of *factError*.

## 17.14 Surface Editor



The surface editor allows you to modify a triangular surface in several ways, e.g., to remove or refine triangles, to flip edges or move points, or to set boundary ids for individual triangles. It also allows you to check a surface for intersecting triangles or for falsely oriented patches.

To open a surface editor, in the Pool select the surface object to edit, then click on the Surface Editor button in the Properties Area. The surface editor places its GUI controls directly into the main 3D viewer window. This makes interactive surface editing very comfortable. In particular, the editor adds a *Surface* menu to the main menu bar, as well as a toolbar. The user interface of the editor is depicted in Figure 17.21. To close a surface editor, click again on the surface editor button in the Properties Area. Note that only one surface editor can be active at a time. Activating a second editor causes the first one to be closed.

To save your work, you can use the following entries of the *File* menu:

- *Save*: Saves the surface under the same file name as it has been saved under before. This option allows you to quickly save intermediate results during complex edit tasks.
- *Save As...*: Pops up the file dialog and saves the surface under a user-chosen file name.

The surface editor works in conjunction with a *SurfaceView* module. If such a module is not already connected to the surface, an instance of it will be created automatically when the editor is activated. However, most settings of the *SurfaceView* module can be controlled directly via menu entries of the editor itself. Therefore there is no need to have the *SurfaceView* module permanently selected.

A basic concept of the surface editor are the notions of *visible* and *highlighted triangles* respectively. Highlighted triangles are drawn in red wireframe. Many tools operate not on the surface as a whole, but only on highlighted triangles. For example, if you want to refine parts of the surface, you'll first have to highlight the involved triangles before choosing *Refine faces* from the *Surface/Edit* menu. Besides the highlight buffer, there is another buffer determining which triangles will actually be drawn. This allows you to cut away parts of a complex surface. Highlighted triangles can be added to the visible buffer or removed from it, just as in an ordinary *SurfaceView* module. Note that hiding a triangle doesn't mean to delete it.

Triangles can be highlighted in a number of ways. On the one hand, there are interactive *tools* allowing you to select triangles using some kind of mouse interaction. For example, triangles can be picked individually or a contour can be drawn in the 3D viewer. Buttons representing the different mouse tools are listed in a tool bar. Clicking on a button causes the corresponding tool to be activated. On the other hand, triangles can be highlighted using automatic *selectors*. Selectors are listed in the leftmost combo box of the surface editor's selector tool bar. There are selectors for highlighting triangles depending on their inner and outer region, for highlighting triangles with a certain boundary id or belonging to certain patch, and many more.

In the following different elements of the surface editor will be described in detail, namely

- *menu entries*
- *selectors including surface tests*

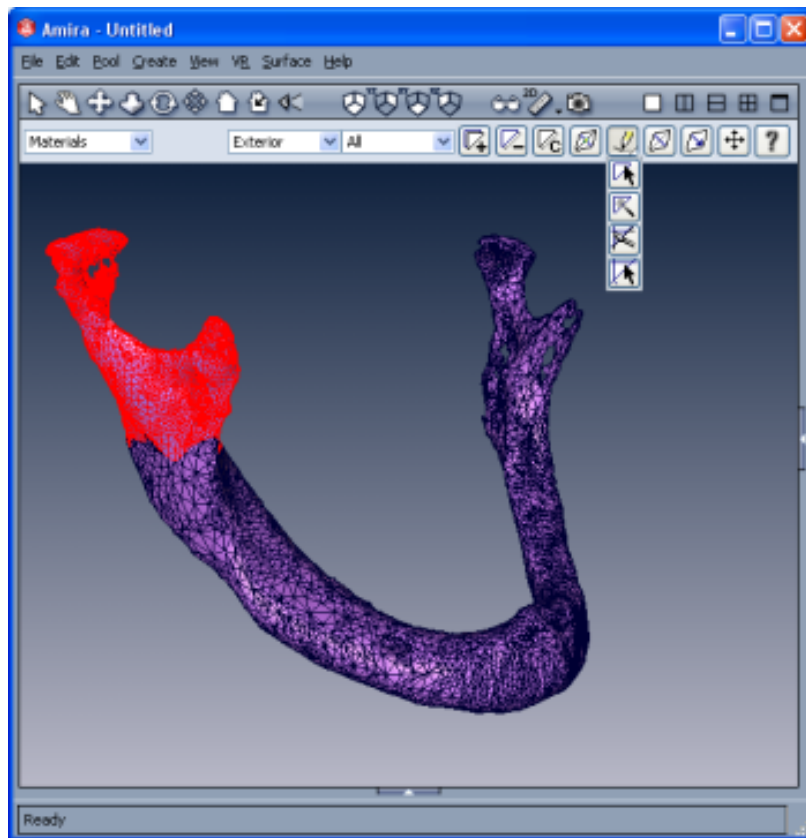


**Figure 17.21:** User interface of the surface editor.

- *mouse tools for selecting triangles and editing the surface*

### 17.14.1 Menu Entries

The surface editor's *Surface* menu is displayed inside the 3D viewer window while the editor is active. Four submenus are accessible from this menu: *Edit*, *View*, *Buffer*, and *Tests*. Each of these is described separately below.

#### 17.14.1.1 Edit Menu

- *Undo*: Undoes the last edit operation (edge flip, edge collapse, edge bisection, vertex movement). Selection and highlight changes can be undone as well. The size of the undo buffer is limited to 100 entries.
- *Delete highlighted faces*: Permanently removes the highlighted triangles from the surface.
- *Remove coplanar faces*: Permanently removes coplanar triangles, i.e., triangles with the same three vertices, from the surface. The inner region and outer region id of the remaining triangle is updated appropriately.
- *Recompute connectivity*: Recomputes the surface's connectivity information, i.e., contours and branching points. Connectivity information is optional. If it is present, contours can be displayed using the *ContourView* module. In addition, *recompute connectivity* causes unused points of the surface to be deleted.
- *Refine faces*: Subdivides all highlighted triangles by bisecting their edges. Each refined triangle is replaced by four new ones. In addition, adjacent non-highlighted triangles are split in two in order to avoid dangling nodes.
- *Smooth faces*: Smooths all highlighted triangles by shifting their vertices. Each vertex is shifted towards the average position of its neighbors. Special care is taken in the case of boundary vertices, for which not all the neighbors are considered, but only those that are also on the boundary. In this way sharp boundaries are preserved.
- *Flip edges...*: Pops up a dialog allowing you to flip certain edges inside the highlighted part of the surface. The edge flips are performed in order to improve the aspect ratio of the involved triangles. An edge flip is only attempted if the aspect ratio of one of the two involved triangles is worse than a user-specified value, and the crease angle between them is below a user-defined threshold.
- *Reorder patches...*: Pops up a dialog allowing you to rearrange the internal order of the surface patches. Two different patches can be selected and then swapped. This operation is useful for certain applications where two different surfaces are required to have the same patch structure.
- *Set boundary ids...*: Pops up a dialog allowing you to edit the surface's boundary ids. The dialog permits you to define new boundary ids or to change existing ones (including the preferred colors). Highlighted triangles can be assigned a new boundary id using the dialog's *set* button.
- *Fix intersections...*: Pops up a dialog for an automatic repair of intersecting triangles. The total number and the number of fixed intersections will be shown in the console window.



- *Fix small dihedral angles...*: Pops up a dialog for an automatic repair of small dihedral angles. The dialog offers two different repair modes: vertex smoothing, and a combination of edge flips and vertex shifts. The dialog lets you define a lower bound for the dihedral angle.
- *Fix tetra quality...*: This tool provides an automatic repair of configurations which might induce large tetrahedron aspect ratios. It pops up a dialog where you can specify an upper bound and an attempted value for the tetrahedron aspect ratio.
- *Prepare Tetragen...*: This tool may be useful if you want to create a tetrahedral volume mesh from the surface. It combines the *Flip edges*, *Fix small dihedral angles*, and *Fix tetra quality* tools. You can specify a lower bound and an attempted value for the tetrahedron aspect ratio.
- *Magic wand settings...*: This menu entry pops up a dialog allowing you to change parameters of the *magic wand tool*. Details for that are described below.

#### 17.14.1.2 View Menu

This menu allows you to change certain settings of the *SurfaceView* module used by the surface editor. The settings affect the draw style and the color mode used for rendering the surface. Possible draw styles are *outlined*, *shaded*, *lines*, *points*, and *transparent*. Possible color modes are *normal*, *mixed*, *twisted*, and *boundary ids*. In the first three modes colors are chosen according to the material ids assigned to the surface's patches. In the last mode colors are chosen according to the triangles' boundary ids.

#### 17.14.1.3 Buffer Menu

This menu contains several entries for modifying the editor's highlight and view buffers. The buffers determine whether a triangle is highlighted or whether it is visible in a normal fashion. Both buffers are independent from each other. They can be stored in an internal backup buffer using the menu entries *copy highlights* and *copy buffer* respectively. Once one of the buffers has been copied it can be restored using *paste highlights* or *paste buffer*.

#### 17.14.1.4 Tests Menu

This menu lists certain test operations which can be performed in order to check the quality and consistency of the surface. Internally, the tests are considered as *selectors* because they cause certain triangles of the surface to be highlighted. Therefore, the test are described in detail in the *selectors section*.

### 17.14.2 Selectors

This section describes tools for automatically highlighting certain parts of the surface. The selectors are listed in a combo box on the very left just beneath the main toolbar. Most selectors provide some additional controls which are displayed right beside this combo box.

- *Materials*: Just as in a *SurfaceView* module, this selector allows you to highlight parts of the surface separating two particular regions from each other. The two regions are specified in two additional combo boxes which are shown once the selector is activated.
- *Boundary ids*: Highlights all triangles with a particular boundary id. The boundary ids defined in the surface are listed in a separate combo box. Boundary ids can be modified using the *Set boundary ids...* option of the *Surface/Edit* menu.
- *Patches*: Highlights all triangles belonging to a particular patch of the surface. The different patches can be selected using a slider displayed right beside the selector combo box. A patch is a group of triangles separating the same two regions and having the same boundary id.
- *Intersection test*: Performs an intersection test and highlights intersecting triangles. The *compute* button actually initiates the test, while the *back* and *forward* buttons allow you to cycle through the list of intersecting faces. The total number of intersections is printed in the console window. Intersections can be repaired automatically using the *Fix intersections* tool from the *Surface/Edit* menu, or manually using the edit tools described *below* (edge flip, edge collapse, edge bisection, vertex movement).
- *Closedness test*: Performs a closedness test and highlights those triangles where the test fails. This test should only be applied to manifold surfaces.
- *Orientation test*: Performs an orientation test and highlights falsely oriented triangles. You should have removed all intersections before applying this check. For simple configurations the wrong triangles are removed automatically. The number of inconsistently oriented triangles and the number of removed triangles are printed in the console window. If the automatic method could not remove all incorrect orientations, you can proceed as in the case of intersections and try to repair them manually. **Important**: A prerequisite for the orientation test is that the outer triangles of the surface be assigned to material *Exterior*. If the surface does not contain such a material or if the assignment to *Exterior* is not correct, the test will falsely report orientation errors.
- *Aspect Ratio*, *Dihedral Angle*, and *Tetra Quality* sort all surface triangles according to different quality measures. The *aspect ratio* is the ratio of the radii of the circumcircle and the incircle of a triangle. The largest aspect ratio should be below 20 (better 10). The *dihedral angle* is the angle between two adjacent triangles at their common edge. The smallest dihedral angle should be above 5 degrees (better 10). *Tetra quality* may be useful if you want to create a tetrahedral volume mesh from the surface. For each surface triangle the aspect ratios of the tetrahedra which will probably be created for that triangle are calculated. The largest tetrahedral aspect ratio should be below 50 (better 25).

The worst triangle according to the selected quality measure is displayed, and the worst quality is printed in the console window. Using the *back* and *forward* buttons right beside the selector combo box you can cycle through all other triangles and edit the surface if necessary.

In the *Surface/Edit* menu, there are automatic tools for improving all of the quality measures:

*Flip edges* for the triangle aspect ratio, *Fix small dihedral angles* for the dihedral angle, and *Fix tetra quality* for the tetrahedron quality.

### 17.14.3 Tools

This section describes interactive tools for selecting triangles and for performing simple edit operations. Only one such tool can be active at a time. The active tool determines what effect left mouse button clicks have. For each tool there is a separate button contained in the editor's tool bar. Independent from the active mouse tool, edges and triangles can be picked using the middle mouse button. This causes the ids of the picked triangles, edges, and points to be printed on the screen.

- *Add tool [A]:*



Adds highlighted faces to the buffer.

- *Remove tool [R]:*



Removes highlighted faces from the buffer.

- *Clear tool:*



Clears highlighted faces or the buffer.

- *Bisect tool [B]:*



Subdivides an edge of the surface. A new vertex is inserted at the edge midpoint. Each triangle adjacent to the edge is bisected into two triangles.

- *Selection tools:*

The following selection tools are available from a pulldown menu in the surface editor toolbar. Press and hold the button down for at least 1 second to display the menu. The most recently selected tool button will be displayed in the surface editor toolbar.

- *Pick tool [P]:*



A single triangle can be highlighted using a simple mouse click and unhighlighted using a Control-click. If the Shift key is held down, a group of neighboring triangles will be highlighted. If the Shift and Control keys are held down, a group of neighboring triangles will be unhighlighted.

- *Magic wand [M]:*



Allows you to select a connected group of triangles. Unless the Shift key is held down, the highlight buffer will be cleared before new triangles are highlighted. Control-click causes all selected triangles to be unhighlighted. The behavior of the magic wand tool can be modified using the dialog *Magic wand settings...* which can be activated from the *Surface/Edit* menu. This dialog lets you define what triangles are considered to be a connected group. In particular, a crease angle smaller than 180 degrees can be chosen. In this case only triangles with roughly the same direction as the clicked triangle will be highlighted. Regardless of these settings, a connected group of highlighted triangles will always be unhighlighted if a highlighted triangle is Control-clicked.

- *Draw tool [D]:*



This tool allows you to highlight triangles by drawing a contour in the viewer window. Usually the left mouse button must be held down while the contour is drawn. However, you may hold down the Alt key and release the left mouse button. In this case straight line segments can be defined. Currently the tool selects all triangles inside the contour, i.e., hidden triangles are highlighted too. However, with the magic wand tool backward facing parts of the surface can be easily deselected again using a Control-click.

- *Brush tool [B]:*



This tool allows you to highlight triangles using your mouse like a paintbrush. Press the left mouse button and select the desired triangles. Hold down the Control key to deselect triangles. Right-click to bring up a brush configuration dialog which will allow you to specify the brush size and whether you want to select visible triangles only.

- *Pick patches or triangle groups tool:*



This tool allows you to select patches or triangle groups. Clicking on a triangle will select all triangles in the same patch. Control-click to deselect the triangles.

- *Flip tool [F]:*



Flips an edge of the surface. Only edges with two adjacent triangles can be flipped, but no boundary edges. Flipping the edge once again restores the original state.

- *Collapse tool [C]:*



Contracts an edge, i.e., moves one vertex of an edge onto the other. For non-boundary edges the operation will reduce the number of triangles by two. The vertex of the edge located more closely to the mouse position will be retained.

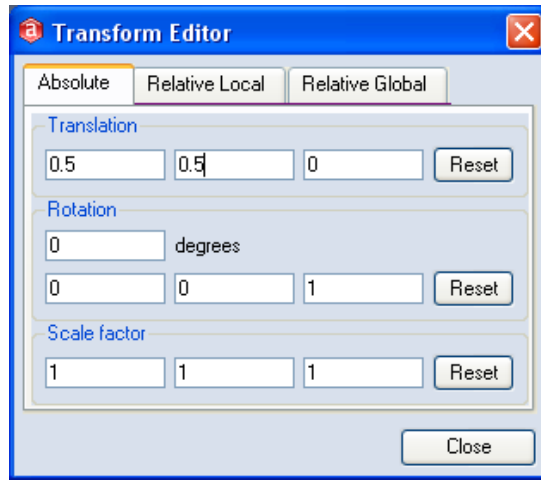


**Figure 17.22:** Dialog opened by the transform editor.

- *Translate tool [T]:*



Allows you to pick a vertex of the surface and to translate it. At the picked vertex a point dragger will be shown. The dragger can then be picked and translated. Alternatively another point on the surface can be shift-clicked while the point dragger is shown. This moves the dragger to the new position.

## 17.15 Transform Editor



This editor allows you to add a transformation to a data set or modify an existing one. A transformation may be a translation, rotation, and scaling, or a combination thereof. The transformation can be edited interactively in the 3D viewer using different Open Inventor draggers.

The *Transform Editor* also lets you enter transformations numerically. This can be done by pressing the *Dialog...* button which pops up the dialog shown in Figure 17.22.

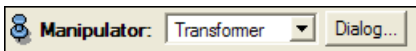
The dialog provides text fields to specify the translation, rotation, and scaling of the object individually. You can specify absolute values, or the object can also be translated, rotated, or scaled incrementally. This is done by activating the tab panels *Relative Local* or *Relative Global* instead of the default *Absolute*. The local and global tabs differ in the way that the translation, rotation, or scaling is applied, namely after an existing transformation (local) or before it (global). An alternative way to modify the transformation of a data object is to use the Tcl commands provided by *SpatialData* objects.

Display modules connected to a transformed data set will take the transformation into account automatically. Also many, but not all (!), compute modules interpret transformations. If you find a module that does not operate as expected for a transformed data object, try to apply the transformation first using the *Apply transform* button.

Most data file formats do not support transformations. Therefore, transformations are stored in amira network scripts only. Keep this in mind when working with transformed data objects. It is always possible to query and reapply a transformation using the Tcl commands `getTransform` and `setTransform`.

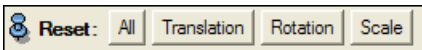
## Ports

### Manipulator



Lets you choose the Open Inventor dragger used to define the transformation in 3D. Make sure to switch the viewer to interaction mode when using the draggers (press the arrow button in the upper right corner of the viewer, or toggle between viewing mode and interaction mode using the ESC key). Details of how to interact with the draggers can be found in the Open Inventor documentation. The *Dialog...* button pops up the transform dialog described above.

### Reset



This port allows you to reset the transformation matrix, or its translational, rotational, or scaling components.

### Action



This port allows you to undo the last change of the transformation matrix, or to redo the last undone operation. In addition, transformations can be copied into an internal buffer, and afterwards pasted into the editor again. This provides a convenient way to copy a transformation from one object to another.

If the editor is invoked for an object derived from *VertexSet*, a button *Apply Transform* is shown. This button allows you to apply the transformation to the vertices of the data object and to reset the object's transformation matrix. This operation does not change the visual appearance of the object, but it is required, for example, to export the transformed object into a file.



# Chapter 18

## Molecular Pack

### 18.1 Molecule Attribute Editor



This tool allows you to edit attributes of groups contained in a *Molecule* data object. It has additional tabs containing interfaces for exporting attributes to a file or importing them from a file into the data object. We will start with a general description of the attribute concept of *amira* followed by a detailed description of the three different tabs.

#### The level and attribute concept of *amira*

In *amira*, each molecule consists of several grouping levels, which are chemical subdivisions of the molecule of different degrees of complexity. The levels are ordered in a hierarchy in the way they depend on each other. The most basic subdivision, and therefore the root of the hierarchical tree, is the atom level. All bonds or residues are subdivisions which contain sets of atoms, while secondary structures consist of sets of groups of the *residues* level, and so on. In the language of database systems, levels are entities and groups are instances of these entities. Figure 18.1 shows the entity relationship model for the most common grouping levels.

Each level has a set of attributes. Attributes are properties of groups of type string, integer, or float. The number of levels and attributes depends on the file format from which the molecule is read.

#### The Edit Tab

The edit tab of the attribute editor (Figure 18.2) offers two different tools:

- addition, deletion, or renaming of attributes from a level



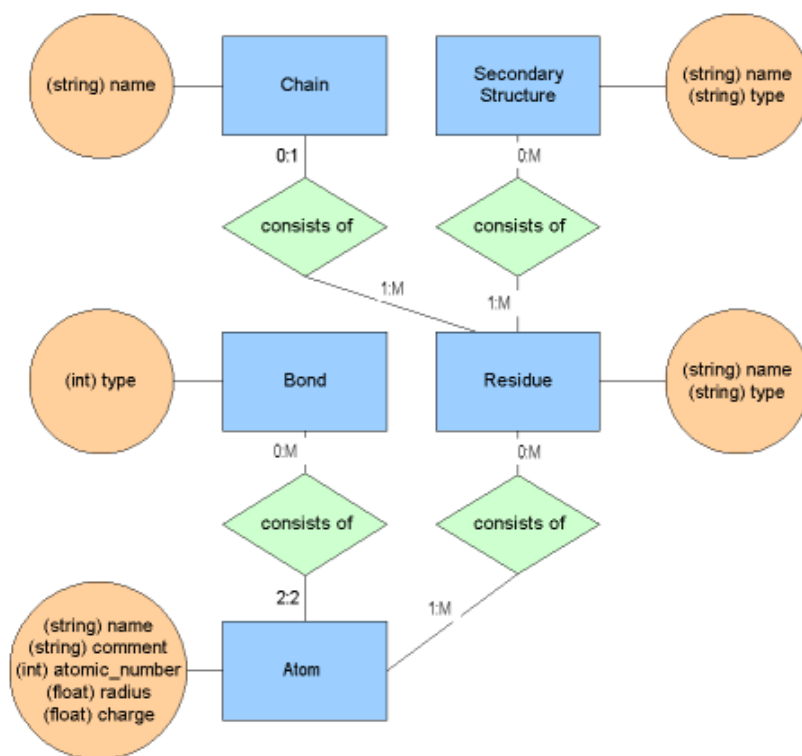


**Figure 18.1:** Entity relationship model of the most common levels

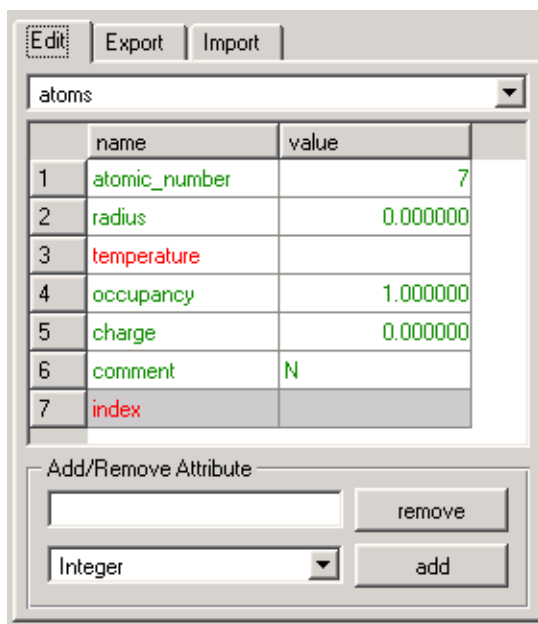


**Figure 18.2:** Attribute Editor

- changing of attribute values of certain groups of a level

The pull-down widget at the top of the window lets you choose the level to which the action will be applied. The table below will show all attribute names of the level in the left column. The second column will contain the corresponding attribute values of all completely selected groups of the given level.

The coloration of these table cells can change depending on the attribute and the selected groups:

- gray background: attribute cannot be changed or deleted (attribute is used as an index by amira)
- white background: attribute can be changed and deleted
- black text: only one group of the level is selected
- green text: several groups of the level are selected but all of their attribute values are equal
- red text: several groups of the level are selected and at least two attribute values of these groups differ (in this case no attribute value will be displayed in the right column)

**Changing attributes or attribute values:** To change the name of an attribute, simply left-click in the respective cell of the left column and enter the desired name. To change attribute values of the currently selected groups, click in the cell in the right column. Except for index attributes (gray background), all values can be changed. If several groups are selected and no value is displayed (because some values

differ), the adjustment of the value in the empty table cell will reset the value of all groups to the given choice. Therefore the color of this attribute will change from red to green.

To select groups you can use the *selection browser*.

**Adding and deleting attributes:** The lower part of the window lets you add or delete attributes by typing the name into the text box and using the appropriate button. When adding attributes, you also must choose the internal format type (string, integer, or float) with the pull-down menu.

## The Export Tab

If you have done calculations in *amira* which created attributes as a result, you might want to save these attributes to a file which can be used by other software (for example, statistics packages). This tab gives you a powerful ability to write and format your output.

**Using a predefined format specification:** To export attributes to a file, you must specify the format in the format widget. You can load a predefined format specification by clicking on the *Predefined Specification* button. We have included some simple specifications for common tasks. You can add your own specification by editing the file `share/molecules/exportPredefinitions.cfg` in your local *amira* directory.

**Creating a new format specification:** A complete explanation of the concept of the forming string can be found in the following section. We will start with a simple example which shall give you a first understanding of the idea behind the concept:

```
%(atoms)%(atoms, charge)
```

will write a file which will contain the charge of each atom. Note that after the last parenthesis you have to enter a whitespace, otherwise all charges will be written without delimiters between them.

**Iteration-context:** The first thing to think about when writing attributes to a file is to decide which level should be the base level of information. This will be the level over whose groups will be iterated, the 'iteration-context'. Usually this is just the same level as that of the attributes you want to write. However, imagine you want to write the residue names of the residues each atom belongs to. In this case the iteration context is the level 'atoms' while the attribute is of the level 'residues'. To set the iteration-level, just type `%(levelname)`. In the example given above the iteration-context was the level 'atom'.

**Text and attribute output:** The text after this iteration-context definition specifies the output for each member of the iteration-level. It can contain two things: specification of attributes and additional text that might contain delimiter characters or keywords needed by the importing function of another program. The attribute specification has the form `%(levelname,attributename)`. The additional text can contain everything (including carriage returns) except the character '%'. In the example above the attribute specification was `%(atoms,charge)` and the additional text was the trailing whitespace.

**Iteration context and level dependency:** Another example:

```
%(residues)%(residues,index) %(chains,index)
```

Results in a table which contains the chain index and the residue index for each residue (after the last parenthesis a carriage return must be entered, otherwise there wouldn't be linefeeds between the individual entries).

The slightly modified example

```
%(residues)%(residues,index) %(atoms,index)
```

might look okay at first glance, but what is the atom index for each residue? In fact it is undefined as each residue can contain several atoms. This is a direct result of the general concept of level dependency. If groups of a level lev1 can contain groups of level lev2, lev1 is said to be dependent on lev2. Inside of an iteration-context only such levels which depend on the iteration-level or the iteration-level itself may be used.

**Ending an iteration context:** The iteration-context can be ended by the definition of a new iteration-context or the empty definition %(). The latter enables you to write text between different sections. Example:

```
ATOM SECTION:
```

```
%(atoms)ATOM ix=%(atoms,index) z=%(atoms,atomic_number) isInRes=%(residu
```

```
%)RESIDUE SECTION:
```

```
%(residues)RESIDUE ix=%(residues,index) name=%(residues,name)
```

of course again ended by a carriage return after the last parenthesis. The %() was needed so that the text 'RESIDUE SECTION' wouldn't be repeated for each member of the 'atoms' iteration-context.

**Special attributes:** As atom coordinates are not stored as attributes but in an internal data array of the object, they are not directly available. However, you can write them by using %(atoms,coordinates) which will be internally translated from an attribute access to an access on the data array. It will write the x, y, and z coordinates delimited by whitespaces.

**Options:** The option *write selected only* will limit each iteration context to those groups which are currently selected. For unselected groups no output will be produced.

## The Import Tab

If you want to use results of other programs in amira, you can import them as attributes of groups. An example is the common task of generating partial charges with a molecular force field tool and then reading them in to examine the results.

If you haven't done so yet you should first read the previous section about exporting a file, as the syntax of the export format string can be considered to be a simple form of the syntax of the import format string.

**Using a predefined format specification:** Just as for exporting attributes you can also load predefined format specification with the *Predefined Specifications* button. You can edit the specification in the file `share/molecules/importPredefinitions.cfg` in your local `amira` directory.

**Range of application:** When you want to import an attribute your first task will be to take a look at the file to analyze its structure and to find the information that you need. If the information in the file is in nested structures, it is recommended to transform the file into a simpler format by writing, for example, a perl script or to write your own `amira` internal reader method using *Developer Pack*.

Most of the files you will encounter however contain information simply delimited by certain characters or keywords or located in certain columns of the file. If the format is quite complicated it is sometimes helpful to preprocess the file via the Unix commands 'egrep' and 'cut' to cut out the information needed.

**Iteration-context and attribute specification:** To get the idea behind the import format syntax, here is an example for a very simple file format:

```
%(atoms)%(atoms, charge, float);
```

This code expects that there are as many charge values in the file as the number of atoms in the molecule, each separated by a semicolon.

Thus the format string has the same concept of an iteration-context as the export method. The only new part in the example is the type specification 'float' which is needed if the attribute does not already exist inside `amira`. If it does, you can omit the type. Type can be 'integer', 'string', or 'float'. Another difference to the export format is that you can specify only attributes of the same level as the iteration-level.

**Skipping characters:** If you want to specify that there may be an arbitrary number of characters of some type, you can do so by `%"char"*`. In the example

```
%(atoms)%(atoms, charge)%" ";%" "*
```

would be needed if there may be an arbitrary number of whitespaces between the charge values and the semicolon. You can enter more than one character between the quotes. It will read and skip any character contained in the quotes until the token behind the asterisk is found. If the token is not a literal, but an attribute specification, it will skip the characters until the first occurrence of a token that might be of the same type as attribute. The quoted characters can of course also contain a linefeed. A `%"*` will skip any character.

An often occurring problem is that your information is located in a certain section of the file which is initiated by a certain keyword. To jump to this section simply type `%*keyword`. The following example could be used for the TRIPOS file format which uses the `'@;TRIPOS;ATOM'` keyword to initiate its atom block.

```
%*@<TRIPOS>ATOM
%(atoms)%(atoms,index) %(atoms,atomic_symbol,string)%*
```

Another common problem is to skip a token. Consider for example that at the start of each line is some alphanumerical token followed by one or more spaces followed by the attribute you want to read. Typing all characters which can be skipped would be tedious, but you can define ranges by using `[char1-char2]`. Thus the example looks like

```
%(atoms)%"[a-z][A-Z][0-9].,_-""%(atoms,charge,float)%*
```

Note that the range is interpreted as a range of the ASCII positions of the given characters.

**Using a file-internal iteration:** Take a look at the following example:

```
%(atoms)ATOM:%" ""%(atoms,index)%" ""%(atoms,charge)%*
```

The defined iteration-context won't be needed because for each charge value we know the atom index it is associated with. Thus, the index has the function of an own file-internal iteration-context. This allows you to import attributes of only a part of the groups contained in the molecule. You will need to specify the iteration context 'atoms', however, to make it clear which parts of the format string constitutes the pattern repeatedly searched for in the file. This however only applies if the index attribute has been read first. If there are other attributes before it in the same iteration context, it will not overrule the context.

**Reading information in predefined columns** Some file formats (like pdb) do not use delimiter characters, but have predefined columns in which information is located. To access these columns you can add width numbers to the attribute specifications and skip specifications. The format is: `%width(levelName,attributeName,attributeType)` and `%width*`.

If the charge attribute you want to read is located between the 50th and 59th column of each line, you could specify the format like this:

```
%(atoms)%49*%10(atoms,charge)%*
```

followed again by a linefeed. The first 49 columns of each line will be skipped. Then the charge attribute will be read from the next 10 columns and the rest of the line will be skipped.

## 18.2 Molecule Editor



This tool can be used to change the geometry and topology of a *Molecule* data object. To change group attributes, you can use the *Molecule Attribute Editor*.

Most of the editor's functions are applied to sets of atoms, which can be selected by using the *selection browser* or by directly clicking on atoms in the viewer.

The molecule editor, Figure 18.3, has three tabs:

- The *Transform* tab contains all tools to change the geometry of selected atoms by adjusting positions, bond lengths, torsional angles, and bond angles.
- The *Tools* tab offers methods to split or copy parts of the molecule, as well as an interface for adding or removing bonds between atoms.
- The *Building* tab offers methods to add or remove atoms, bonds and groups, or to change their chemical properties.

Additionally several menus exist to manipulate the display, assign or change chemical properties and to create new levels or attributes.

### Preparing the molecule

Many of the editing methods of the Molecule Editor need to compute chemical properties of the molecule. These properties are derived from some fixed properties representing the atomic configuration, namely the valency of the atoms, its atomic number and its formal charge. The atomic number has its own attribute and the valency is represented as explicit bonds in the molecular data structure and as an additional number of implicit hydrogens. Therefore the editor itself needs to store only two additional properties: the implicit hydrogen number and the formal charge and it will adapt these properties during the editing process.

When the editor starts, this information will be read from the `formal_charge` and `implicit_hnum` attributes, if they exist. If they don't exist, the properties will be set to 0 for each atom. Often this is incorrect. PDB files for example usually don't contain hydrogens and the number of implicit hydrogens therefore needs to be adjusted. It is not possible to generate such missing information without further guidance from the user, as the way in which this information needs to be generated depends on how the molecule was generated. The Editor, however, offers a set of tools which allow to generate this information easily for most typical cases. The tools menu allows to assign a standard implicit hydrogen number a implicit hydrogen number and formal charges. For molecules which contain neither a `formal_charge` attribute nor explicit hydrogens or an `implicit_hnum` attribute, you need to use the first method. It will assign sufficient hydrogens to each atom to reach its most common valency. After this you may need to adjust valencies and formal charges. If the `formal_charge` attribute is known, you should use the second method instead. It will consider changes in the standard valencies caused by the

electron transfer. If all hydrogens are known, use the third method which will adjust formal charges to reach standard valencies. You can always check these internal properties by enabling them as labels in the 'view' menu.

As mentioned earlier, this information may change during the editing process. If you want to store the properties in the amira data structure, (for example to be able to use it again when restarting the editor later), you need to convert the two internal properties into attributes. You may do so by using the 'Editor Attribute' entry in the 'Create' menu.

## The view menu

The view menu allows to adjust some visualization properties of the MolView that is used by the editor. 'Hide H' will hide or show the hydrogens atoms. Additionally labels can be activated which show important atomic properties:

- None: Labels deactivated.
- Standard: Modes formal charge and radicals combined.
- Formal charge
- Implicit H-Num: Number of implicit hydrogens.
- Lone Pairs: Number of lone electron pairs.
- Oxidation number
- Radicals: Shows an asterisk for atoms that contain unpaired electrons.

## The tools menu

- Assign standard implicit hydrogens: Assign the number of implicit hydrogens while not considering any additional data like the formal charge. Use this if the molecule contains neither explicit hydrogens nor formal charges. After using this assignment you will have to adjust formal charges (if there are any) manually.
- Assign implicit H-Num: Assign number of implicit hydrogens while taking into account formal charges. Use this if the molecule contains formal charges, but no explicit hydrogens.
- Assign formal charges: Assigns formal charges while taking into account the number of implicit hydrogens. Use this, if the molecule contains explicit hydrogens but no formal charges.
- Add H: Makes all implicit hydrogens explicit.
- Add Polar H: Makes all implicit hydrogens of polar atoms explicit.
- Strip H: Makes all hydrogens implicit.
- Strip Apolar H: Makes all apolar hydrogens implicit.
- Strip H2O: Removes all water molecules (including unbound O or H atoms and OH molecules).
- Kekulize: Assign a kekulé structure to the molecule, in effect replacing all aromatic bond systems with alternating single and double bonds.



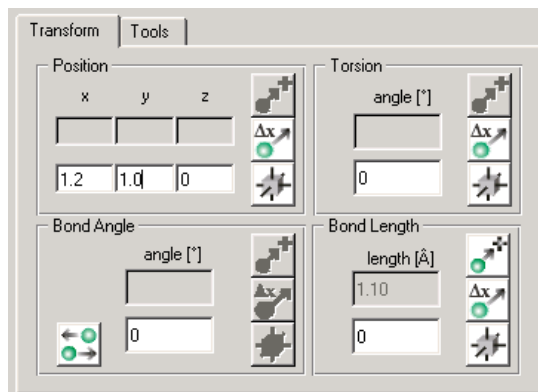


Figure 18.3: Transform Tab

- **Kekulize Non-Rings:** Assigns a kekulé structure to all non cyclic parts of the molecule. This is necessary for the editor to work properly as aromaticity is only allowed for rings.
- **Dekekulize:** This will check all rings systems whether they are aromatic, using the extended Hueckel rule and assign aromaticity where appropriate.

## Transform Tab

The transform tab is divided into four different sections. Each section can be used to adjust certain coordinates of the currently selected atoms. The different coordinate types are:

- *Position* is the Cartesian coordinate of an atom. If several atoms are selected, only relative changes are allowed.
- *Bond Length* is the distance between two selected atoms.
- *Bond Angle* is the planar angle between three selected atoms.
- *Bond Torsion* is the dihedral angle between four atoms.

Coordinates can be set absolute or relative to their current value. In each section, the upper row (with gray background) displays the current absolute value. In the lower row (with white background) you can enter a new value. In the right part of each section are three buttons. The first button transforms the coordinate while assuming that the entered value is an absolute value. The second button simply applies the transformation with the entered value text as a relative difference. The third button will activate a dragger in the viewer for adjusting the coordinate interactively. Figure 18.4 shows the bond torsion dragger as an example. The *Measurement* module can be used to display currently edited bonds or angles.

Which buttons can be used and which are disabled depends on the number of currently selected atoms.

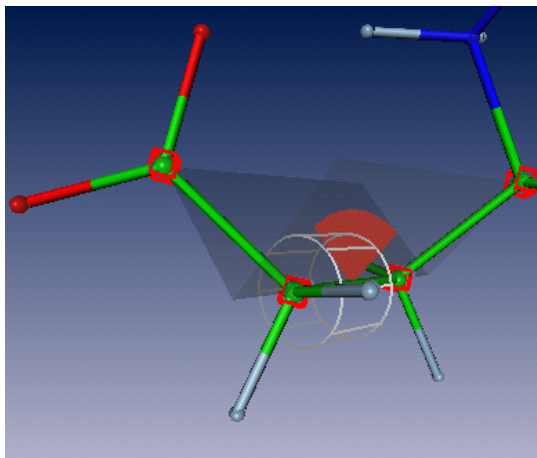


**Figure 18.4:** Editing a dihedral while viewing the angle with the measurement module

An additional toggle button can be found in the *Bond Angle* section. If this toggle is activated, the interactive adjustment with the dragger will cause both bonds to be bent symmetrically, otherwise only one bond will be bent.

The position dragger can be moved on the plane determined by the face of the cubic dragger you click on. You can additionally rotate the selected groups by dragging the green knobs.

## Tools Tab

The tools tab is subdivided into two sections:

### Change Topology

The *cut* button will cut the currently selected groups out of the molecule. The *split* button will do the same but will copy the groups into a new molecule which will be added in the Pool. The *copy* button will leave the current molecule unchanged while copying the selected groups into a new molecule which will be added to the Pool. When pressing the *add* button, a window will open which will let you choose another molecule in the Pool whose groups you want to add to the current molecule.

### Connection

The *Connection* section offers different options for influencing the bonding of the currently selected atoms. On the right side of the interface are buttons for adding or removing bonds between the currently selected atoms. The set of bonds which will be added or removed will depend on the connection mode that you can choose on the left side of the interface.

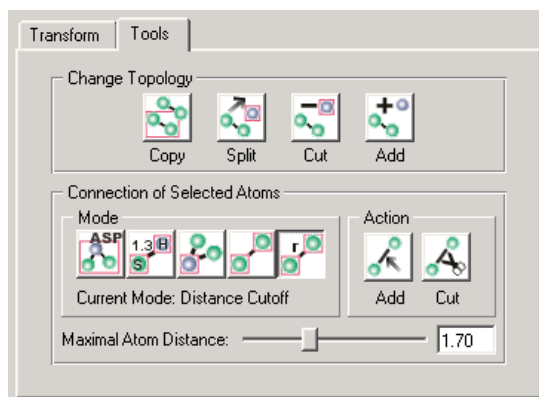


**Figure 18.5:** Tools Tab

- *Standard*: This is the most reliable method for adding bonds to a protein or DNA/RNA. It will look up all residues in a residue database and add bonds accordingly. This method will only work for molecules that contain the residue type attribute. For connections between different residues it will check all residues on a chain sequentially. When used together with the cut action all bonds will be removed which can not be found in the database.
- *Bond length table*: This option lets you add bonds between selected atoms by looking up their bond lengths in the file *bondLengths.cfg* which can be found and edited in your local *amira/share/molecules* directory. If the distance of two atoms does not deviate further than a certain threshold from the bond length between the respective elements in the table, the bond will be added. This method is able to distinguish between single, double, triple, and aromatic bonds. It should be the first choice for non-standard residues or molecules not containing residue information. Just like the *Standard* option this mode can also be used together with the *cut* action. In this case all bonds which deviate too strongly from the bond length table will be removed.
- *External*: This mode can only be used together with the *cut* action. It will remove all bonds between selected and unselected atoms thus enabling you to disconnect certain parts of the molecule from the rest.
- *All*: This mode will add or remove all possible bonds between the selected atoms.
- *Distance Cutoff*: This last mode uses a distance cutoff for deciding which bonds to add or remove. The *Maximal Atom Distance* slider determines the maximal distance of two bonded atoms in Å. When using this mode together with the *add* action all bonds between atoms which are nearer than the cutoff distance will be added. Equally, the *cut* action will remove all bonds whose length is greater than the threshold.

## Building

Adding atoms or groups is accomplished by selecting a single atom and replacing it with a new one. Usually, the selected atom would be a hydrogen, but you can replace heavy atoms as well. Hydrogens for the new parts are automatically added.

The *atoms* section contains buttons to add atoms of commonly used elements in organic chemistry. With the '⌘' button you can activate a periodic table to select less common elements. Other buttons in this section are:

- + and -: Increases or decreases the formal charge of the atom by 1. The number of hydrogens will be automatically adjusted.
- +V and -V: Increases or decreases the valency of the atom by one. They will appear as new or removed explicit or implicit hydrogens. Adding valencies is only possible if the atom has lone electron pairs or unpaired electron. Removing valencies is only possible, if the atom has explicit or implicit hydrogens.
- Delete: Deletes all selected atoms.

The *carbon chains / rings* section allows to add carbon chains or cyclocarbons. The number of the carbon atoms can be adjusted and the hybridization can be selected, with 'sp<sup>2</sup>' creating chains of the form C=CC=C... or aromatic rings.

The *bonds* section contains buttons to add bonds, replace bond order or, delete bonds. You need to select two atoms. Clicking on '-' will create a single, '=' a double, '#' a triple, and '~' an aromatic bond. If the atoms were already connected the bond order will be simply changed. We advise against assigning aromatic bonds by hands. There is no clear definition of what constitutes 'aromaticity' and creating aromatic bonds outside of ring structures can result in some chemo informatics algorithms of the editor returning incorrect result. Instead, always assign kekulé structures. You can then assign aromatic bond orders by using the 'dekekulize' tool which guarantees that the internal data-structures stay chemically consistent.

## Button Group

The *OK* button will close the editor and accept all changes made to the molecule. The *Cancel* button lets you return from the editor restoring the original state. Resetting the molecule without canceling the editor can be done with the *Reset* button. The *Apply* button will apply the changes you have made to the molecule object. This means that any further reset will return the molecule to the current state. With the *Undo* button any transformation can be undone up to 10 levels back from the current state.

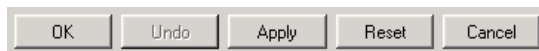


Figure 18.6: Button Group of the Molecule Editor



# Chapter 19

## Mesh Pack

### 19.1 Grid Editor



The Grid Editor allows you to analyze the quality of a *tetrahedral grid* according to different quality measures and to semi-automatically improve the grid quality.

To activate the Grid Editor, press the Grid Editor button of a selected tetrahedral grid. A user interface composed of different button controls (Figure 19.1) will appear in *amira's Properties Area*. If you select some other *Selector* or *Modifier*, other ports might be shown. The editor works in conjunction with the *GridVolume* module. If such a module is not already active, an instance of it will be automatically created when the editor is invoked.

Tetrahedra can be distorted in different ways (see Figure 19.2):

- *Slivers* contain four nearly coplanar vertices forming a quadrangle.
- *Caps* consist of a triangle and a fourth vertex 'just above' it.
- *Needles* consist of a triangle and a fourth vertex 'far away'.
- *Wedges* are characterized by one sharp edge.

#### Using the tools of the *Select* menu

At the menu of port *Selector* you can choose one of several selection criteria. At the next port you can enter an expression which will be evaluated for each tetrahedron or edge. When you press the *Select* button, the number of selected tetrahedra or edges is shown in the *Console Window*, and the selected tetrahedra are shown by the *GridVolume* module.

- **Index Selector:** This tool selects tetrahedra according to their index (i). This is mainly for debugging purposes.

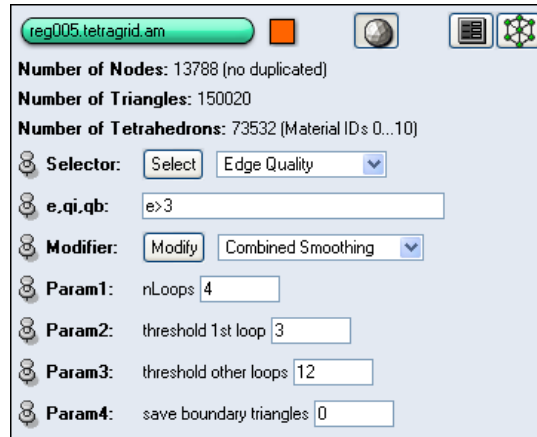


Figure 19.1: Grid editor for improving the quality of tetrahedral grids.

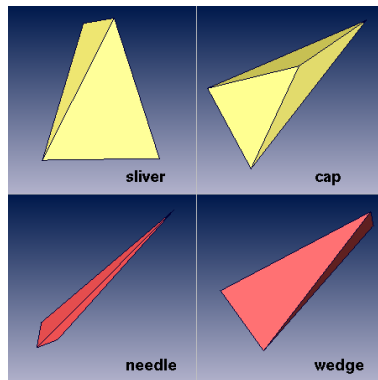


Figure 19.2: Four examples of distorted tetrahedra.

- Tetra Quality Selector:** This tool selects tetrahedra according to different quality measures.
  - 'd' is the determinant of the matrix composed of the vectors pointing from vertex 0 to the vertices 1, 2, and 3. The tetrahedron volume is 1/6 of the absolute value of d. d approaches zero if the four vertices are nearly coplanar.  $d \neq 0$  detects tetrahedra with an inverted orientation, which should normally not occur.
  - 'R' is the ratio of the radii of an inscribed and a circumscribed sphere. R reaches its optimal (maximal) value 1/3 for an equilateral tetrahedron. All types of distorted tetrahedra are detected by a small value of R. Therefore R is taken as the quality measure for all modifiers of the *Grid Editor* (see below). R shouldn't be smaller than approximately 1/10 of the optimal value.
  - 'r1' is the ratio of the smallest to the largest edge length. r1 reaches its optimal (maximal) value 1 for an equilateral tetrahedron. Needles and wedges are detected by a small value of r1.
  - 'r2' is the tetrahedron volume divided by the third power of the largest edge length, normalized to an optimal (maximal) value 1 for an equilateral tetrahedron. All types of distorted tetrahedra are detected by a small value of r2.
- Dihed/Solid Angle Selector:** This tool selects tetrahedra according to their minimal and maximal dihedral angles (d,D) or solid angles (s,S). For each edge of a tetrahedron the dihedral angle is defined as the angle between its adjacent faces. The solid angles are related to the tetrahedron vertices; they measure the part of the unit sphere which is occupied by the tetrahedron. The solid angle at a tetrahedron vertex can be maximally  $2\pi$ , or 360 deg. For an equilateral tetrahedron all dihedral angles are approx. 70 deg, all solid angles are approx. 30 deg.
 

You can combine different selections, e.g., the selection  $(d < 10) \ \&\& \ (D > 140)$  detects *slivers*, and the selection  $S > 180$  detects *caps*.
- Edge Quality Selector:** This tool primarily selects edges, and then shows all tetrahedra adjacent to that edges. Edges are selected according to their length (e) or to the 'edge quality' q, which is computed as follows: for each tetrahedron adjacent to the edge the length l of the opposite edge and the dihedral angle d at the opposite edge are determined. The contribution of the tetrahedron is  $l \cot(d)$ . q is the sum of those contributions over all adjacent tetrahedra. qi refers to the inner edges, qb to the boundary edges of the tetrahedral grid.
 

For an 'ideal' grid, all qi and qb should be positive. Such a grid would be well suited for a numerical simulation, because the Finite Element stiffness matrix (probably for the Laplacian operator) is an M-matrix if q is positive for all edges.

### Using the tools of the *Modify* menu

At the menu of port *Modifier* you can choose one of several modifiers. When you press the *Modify* button, the grid modification will start. Some information will be displayed in the *Console Window*.

There is a certain inconsistency between the tetra quality selector and the modifiers of the *Grid Editor*, because the tetrahedron quality criterion for all modifiers is the *inverse* of the radius ratio R as defined above. In applying the modifiers, keep in mind that the optimal (minimal) value of 1/R is 3 and distorted tetrahedra are detected by large values of 1/R.

The modifiers *Laplace Smoothing*, *Optimization Smoothing* and *Flip Edges and Faces* are mainly



for debugging purposes, because the *Combined Smoothing* modifier is a combination of them which should give the best results in most cases. The modifiers *Repair Bad Tetras* and *Bisect Inner Edges* are still in an experimental state. We recommend to apply the *Remove Inner Vertices* and the *Combined Smoothing* modifiers.

- **Laplace Smoothing:** This tool improves mesh quality by moving inner vertices. For each inner vertex the center of mass of the adjacent vertices is calculated. The vertex is moved to that location if this improves the quality 1/R of the adjacent tetrahedra. Otherwise the midpoint between the old location and the center of mass is examined. Parameter *nLoops* defines the number of smoothing loops (maximally 10). Laplace smoothing will in general improve the mesh quality 1/R, but in most cases an *Optimization Smoothing* will be superior.
- **Optimization Smoothing:** This tool improves mesh quality by moving inner vertices. For each inner vertex a new location is determined that optimizes the quality 1/R of the adjacent tetrahedra. Parameter *nLoops* defines the number of smoothing loops (maximally 10), parameter *threshold* defines a threshold for tetrahedron quality which is applied starting with the second loop. If all adjacent tetrahedra have a quality better than *threshold*, the vertex position is not changed. The default value 12 (quadruple of the optimal value) leaves tetrahedra unchanged which should be acceptable in most cases. Selecting a smaller value will induce optimization of more vertex locations.
- **Flip Edges and Faces:** This tool improves mesh quality by flipping edges and faces. For each inner (triangular) face the adjacent tetrahedra are determined. It is examined whether the face is a boundary face and whether the adjacent tetrahedra form a convex polyhedron. Depending on this classification a suitable type of edge or face flipping is selected. The flip operation is only performed if it improves the quality 1/R of the tetrahedra involved. Parameter *threshold* defines a threshold for tetrahedron quality. If all adjacent tetrahedra have a quality better than *threshold*, a face is not examined for flipping. If parameter *save boundary triangles* is set to 1, the edges and faces of the exterior grid boundary and the interior boundaries between different materials will not be flipped.
- **Repair Bad Tetras:** This tool tries to repair *slivers* and *caps*. For a sliver, two opposite edges with obtuse dihedral angles are bisected. If the distance between the new vertices is small compared to the sliver's mean edge length, the edge connecting them is collapsed. For a cap, the triangle opposite to the vertex with largest solid angle is determined. If that triangle is part of the outer boundary, the tetrahedron is removed. Otherwise, it is examined if the cap can be removed by a face flip. Parameter *threshold* defines a threshold for tetrahedron quality 1/R. Only tetrahedra with a quality worse than the given threshold will be examined for repair.
- **Remove Inner Vertices:** This tool improves mesh quality by removing inner vertices. In a tetrahedral grid, the mean number of tetrahedra incident on an inner vertex is 24. If a vertex is incident on less than 10 tetrahedra, it is very likely that the mesh quality can be improved by removing that vertex and reconnecting the hole.

All inner vertices incident on a number of tetrahedra less or equal the value of parameter *max num neighbors* are inspected for removal. They will be removed, if this improves tetrahedral quality 1/R. If a value  $\geq 0$  is set for parameter *max edge length*, this defines an upper bound for

the edge length. An inner vertex will not be removed, if this would imply creation of a longer edge.

- **Bisect Edges:** This tool improves mesh quality by bisection of inner edges. An inner edge is bisected if for its vertices different boundary conditions are defined. After bisection, apply *Optimization smoothing* to improve the position of the new vertices.
- **Combined Smoothing:** This tool combines edge and face flipping and optimization smoothing. Parameter *nLoops* defines the number of 'large loops' (maximally 10), the other parameters define thresholds for tetrahedron quality  $1/R$  which are applied in the first loop and the other loops, respectively. Setting the first threshold to 3 means that all edges and faces will be inspected for flipping and all vertices for optimization in the first loop. If parameter *save boundary triangles* is set to 1, the edges and faces of the exterior grid boundary and the interior boundaries between different materials will not be flipped.
- **Flip/Bisect Long Edges:** This tool tries to remove inner edges depending on their length or edge quality  $q$  as defined above. Parameter *max edge length* defines the maximal allowed edge length and parameter *min edge quality* the minimal allowed edge quality, which must be less or equal to zero. Setting a minimal quality of zero means that edge quality is ignored in edge selection. Parameter *threshold* sets an upper limit for tetrahedron quality  $1/R$ . The selected edges are removed, preferably by an edge flip, or by an edge bisection, if all newly generated tetrahedra will have a quality below that threshold.

This tool should be applied repeatedly, until the number of flips as reported in the *Console Window* goes down to 0, and applying an *Optimization smoothing* in between may improve the results.

# Index

Access LargeDiskData, [3](#)  
ACR-NEMA, [417](#)  
Advanced Colormap Editor, [475](#)  
AffineRegistration, [4](#)  
AlignMolecules, [185](#)  
AlignPrincipalAxes, [7](#)  
AlignSequences, [187](#)  
AlignSlices, [8](#)  
AlignSurfaces, [249](#)  
AMBER, [393](#)  
AMF, [394](#)  
Amira Script, [363](#)  
Amira Script Object, [363](#)  
AmiraMesh as LargeDiskData, [374](#)  
AmiraMesh Format, [363](#)  
AnalyticTensorField, [250](#)  
Analyze 7.5, [374](#)  
AnalyzeAVW, [375](#)  
Animate, [22](#)  
AnnaScalarField3, [429](#)  
AnnaVectorField3, [430](#)  
Annotation, [24](#)  
AnonymizeImageStack, [25](#)  
ApplyBSplineTransform, [25](#)  
ApplyTransform, [26](#)  
ArbitraryCut, [28](#)  
Arithmetic, [30](#)  
AVS Field, [411](#)  
AVS UCD Format, [412](#)  
Axis, [34](#)  
  
B-Splines, [432](#)  
BMP Image Format, [375](#)  
BondAngleView, [189](#)

BondCalculation, [190](#)  
BoundaryConditions, [251](#)  
bounding box, [115](#), [390](#), [496](#)  
BoundingBox, [35](#)  
  
CalculusMatlab, [36](#)  
CameraPath Editor, [481](#)  
CameraRotate, [432](#)  
CastField, [39](#)  
ChannelWorks, [41](#)  
ClippingPlane, [42](#)  
Cluster, [433](#)  
ClusterDiff, [42](#)  
ClusterGrep, [44](#)  
ClusterLabel, [45](#)  
ClusterSample, [46](#)  
ClusterView, [47](#)  
Color Dialog, [482](#)  
ColorCombine, [49](#)  
ColorField3, [435](#)  
Colormap, [435](#)  
Colormap Editor, [485](#)  
Colorwash, [51](#)  
CombineLandmarks, [52](#)  
CompareLatticeData, [53](#)  
CompMolInterface, [191](#)  
CompMolSurface, [192](#)  
ComponentField, [252](#)  
ComputeContours, [54](#)  
ComputeHAtoms, [195](#)  
ComputeHBonds, [195](#)  
ComputeSecondaryStructure, [196](#)  
ComputeTensor, [253](#)  
ComputeTensorOutOfCore, [254](#)

ConePlot, [256](#)  
 ConfigurationDensity, [197](#)  
 ConnectedComponents, [54](#)  
 Connection, [333](#)  
 ContourView, [259](#)  
 ContrastControl, [56](#)  
 ConvertToLargeDiskData, [231](#)  
 CorrelationPlot, [59](#)  
 CreateCluster, [62](#)  
 CreateSphere, [199](#)  
 Curl, [63](#)  
 Curve Editor, [489](#)  
 CurvedSlice, [63](#)  
 Cutting Plane, [66](#)  
 CylinderSlice, [67](#)  
  
 Data, [437](#)  
 DataProbe, [68](#)  
 Delaunay2D, [71](#)  
 DemoMaker, [73](#)  
 DICOM, [417](#)  
 Digital Image Filters, [83](#), [489](#)  
 Displace, [259](#)  
 DisplayColormap, [83](#)  
 DisplayISL, [260](#)  
 DisplayLogos, [85](#)  
 DisplayTime, [86](#)  
 Divergence, [262](#)  
 DoseVolume (Tetrahedra), [263](#)  
 DuplicateNodes, [264](#)  
 DXF, [375](#)  
  
 EigenvectorToColor, [264](#)  
 Encapsulated PostScript, [376](#)  
 ExtractEigenvalues, [266](#)  
  
 FiberTracking, [266](#)  
 FIDAP NEUTRAL, [412](#)  
 Field3, [438](#)  
 FieldCut, [268](#)  
 FieldCut (Polyhedra), [270](#)  
 Fluent / UNS, [412](#)  
  
 GetCurvature, [87](#)  
  
 Gradient, [271](#)  
 Grid Editor, [549](#)  
 GridBoundary, [272](#)  
 GridCut, [274](#)  
 GridCut (Polyhedra), [275](#)  
 GridView, [89](#)  
 GridVolume, [276](#)  
 GridVolume (Polyhedra), [278](#)  
 GROMACS, [397](#)  
 Grouping, [90](#)  
  
 HBondView, [201](#)  
 HeightField, [91](#)  
 HexaView, [280](#)  
 HexToTet, [279](#)  
 Histogram, [92](#)  
 HTML, [376](#)  
 HxSurface, [376](#)  
 Hypermesh, [412](#)  
  
 Icol, [381](#)  
 IDEAS universal format, [413](#)  
 IlluminatedLines, [281](#)  
 ImageCrop Editor, [494](#)  
 Interfile, [381](#)  
 Interpolate, [282](#)  
 InterpolateLabels, [95](#)  
 InterpolLabels, [94](#)  
 Intersect, [95](#)  
 Isolines, [97](#)  
 Isolines (Surface), [98](#)  
 Isosurface (Hexahedra), [283](#)  
 Isosurface (LDA), [232](#)  
 Isosurface (Polyhedra), [284](#)  
 Isosurface (Regular), [99](#)  
 Isosurface (Tetrahedra), [285](#)  
 IvData, [439](#)  
 IvDisplay, [101](#)  
 IvToSurface, [101](#)  
  
 JPEG Image Format, [382](#)  
  
 LabelField3, [439](#)

LabelVoxel, [102](#)  
 Lambda2, [286](#)  
 LandmarkEditor, [496](#)  
 LandmarkSet, [440](#)  
 LandmarkSurfaceWarp, [104](#)  
 LandmarkView, [104](#)  
 LandmarkWarp, [105](#)  
 LargeDiskData, [409](#), [442](#)  
 Lattice3, [442](#)  
 LatticeAccess, [107](#)  
 LatToHex, [287](#)  
 LDA, [409](#)  
 LDAExpertSettings, [233](#)  
 LegoSurfaceGen, [108](#)  
 Light, [444](#)  
 LineProbe, [109](#)  
 LineSet, [446](#)  
 LineSet Editor, [497](#)  
 LineSetProbe, [109](#)  
 LineSetView, [110](#)  
 LineStreaks, [287](#)  
  
 Madymo, [423](#)  
 Madymo Time-history, [423](#)  
 MagAndPhase, [288](#)  
 Magnitude, [289](#)  
 masking, [33](#)  
 MasterConnection, [334](#)  
 MATLAB Binary Format (.mat), [382](#)  
 MATLAB M-files Format (.m), [382](#)  
 MDL, [398](#)  
 MeanMolecule, [202](#)  
 Measurement, [203](#)  
 Measuring, [114](#)  
 Merge, [115](#)  
 Molecule, [467](#)  
 Molecule Attribute Editor, [535](#)  
 Molecule Editor, [542](#)  
 MoleculeLabel, [207](#)  
 MoleculeView, [210](#)  
 MolElectrostatics, [204](#)  
 MolSurface, [465](#)  
 MolSurfaceView, [205](#)  
  
 MolTrajectory, [466](#)  
 Movie, [448](#)  
 MovieMaker, [116](#)  
 MoviePlayer, [118](#)  
 MultiChannelField3, [450](#)  
  
 Normalize Image, [122](#)  
  
 Object, [451](#)  
 ObliqueSlice, [123](#)  
 ObliqueSlice (LDA), [236](#)  
 Observables, [212](#)  
 Open Inventor, [382](#)  
 OrthoSlice, [126](#)  
 OrthoSlice (LDA), [238](#)  
 OrthoViewCursor, [128](#)  
  
 Parameter Editor, [498](#)  
 ParametricSurface, [290](#)  
 ParticlePlot, [292](#)  
 PDB, [398](#)  
 PlanarLIC, [295](#)  
 Plot 3D Single Structured, [413](#)  
 Plot Tool, [498](#)  
 Ply Format, [385](#)  
 PNG Image Format, [383](#)  
 PNM Image Format, [383](#)  
 PointProbe, [130](#)  
 PointWrap, [130](#)  
 Port, [331](#)  
 Port3DPointList, [335](#)  
 PortButtonList, [337](#)  
 PortButtonMenu, [339](#)  
 PortChannelConfig, [340](#)  
 PortColorList, [341](#)  
 PortColormap, [342](#)  
 PortDoIt, [343](#)  
 PortDrawStyle, [344](#)  
 PortFilename, [345](#)  
 PortFloatSlider, [346](#)  
 PortFloatTextN, [348](#)  
 PortGeneric, [349](#)  
 PortInfo, [352](#)

- PortIntSlider, [352](#)
- PortIntTextN, [353](#)
- PortMultiChannel, [353](#)
- PortMultiMenu, [354](#)
- PortRadioBox, [355](#)
- PortSeparator, [356](#)
- PortTabBar, [356](#)
- PortText, [356](#)
- PortTime, [357](#)
- PortToggleList, [359](#)
- PrecomputeAlignment, [213](#)
- ProbeToLineSet, [131](#)
- ProjectionView, [131](#)
- ProjectionViewCursor, [133](#)
- PseudoElectronDensity, [214](#)
- PSF/DCD (CHARMM), [400](#)
- PSI format, [383](#)
  
- Radioss, [425](#)
- RankTimeStep, [215](#)
- RateOfStrainTensor, [297](#)
- Raw Data, [385](#)
- Raw Data as LargeDiskData, [387](#)
- RefineTetras, [134](#)
- Relabel, [135](#)
- RelabelTetras, [136](#)
- Resample, [138](#)
- resampling, [32](#)
  
- Scale, [140](#)
- ScanConvertSurface, [142](#)
- ScriptObject, [453](#)
- SecStructureView, [217](#)
- SeedSurface, [297](#)
- Segmentation Editor, [507](#)
- SelectLines, [143](#)
- SelectRoi, [144](#)
- SelectRoi (LDA), [241](#)
- SGL-RGB Image Format, [387](#)
- Shear, [145](#)
- Simplification Editor, [523](#)
- SmoothSurface, [145](#)
- Sound, [146](#)
  
- SpatialData, [457](#)
- Splats, [299](#)
- SplineProbe, [147](#)
- SpreadSheet, [459](#)
- stacked coordinates, [32](#)
- Stacked-Slices, [387](#)
- Stacked-Slices as LargeDiskData, [410](#)
- StandardView, [147](#)
- STL, [387](#)
- StreamRibbons, [300](#)
- StreamSurface, [302](#)
- Surface, [459](#)
- Surface Editor, [524](#)
- SurfaceArea, [149](#)
- SurfaceCut, [150](#)
- SurfaceDistance, [151](#)
- SurfaceField, [153](#)
- SurfaceGen, [153](#)
- SurfaceLIC, [304](#)
- SurfaceView, [155](#)
  
- Tecplot, [390](#)
- TensorDisplay, [306](#)
- TetraCombine, [308](#)
- TetraGen, [309](#)
- TetraGrid, [461](#)
- TetraQuality, [311](#)
- TetToHex, [308](#)
- TIFF Image Format, [388](#)
- Time, [461](#)
- TimeHistoryPlot, [157](#)
- TimeSeriesControl, [157](#)
- TissueStatistics, [159](#)
- Transform Editor, [532](#)
- TriangleDistortion, [161](#)
- TriangleQuality, [312](#)
- Tripes, [400](#)
- TubeView, [221](#)
  
- UniChem, [400](#)
  
- VectorProbe, [313](#)
- Vectors, [314](#)

Vectors (Tetrahedra), [316](#)  
Vectors/Normals (Surface), [317](#)  
Vertex Morph, [164](#)  
VertexDiff, [165](#)  
VertexSet, [462](#)  
VertexShift, [165](#)  
VertexView, [166](#)  
ViewBase, [170](#)  
Visilog, [321](#)  
VisilogAnalyse, [326](#)  
VisilogThreshold, [327](#)  
VolPro-1000, [173](#)  
Voltex, [178](#)  
Voltex (LDA), [243](#)  
VolumeEdit, [181](#)  
voxel size, [390](#), [496](#)  
VoxelView, [183](#)  
VR Pack control module, [223](#)  
VR Pack show config module, [226](#)  
VRML, [391](#)  
VRML-Export, [162](#)  
  
ZIB Molecular File Format, [401](#)