

Amira 5.5

Reference Guide

Amira® 5.5

Amira Reference Guide

Intended Use

Amira® is intended for research use only. It is not a medical device.

Copyright Information

©1995-2013 Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany

©1999-2013 Visualization Sciences Group, SAS

All rights reserved.

Trademark Information:

Amira is being jointly developed by Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Visualization Sciences Group, SAS.

Amira® is a registered trademark of Visualization Sciences Group, SAS.

HardCopy, MeshViz, VolumeViz, TerrainViz, ScaleViz are trademarks of Visualization Sciences Group, SAS.

Visualization Sciences Group, SAS is a source licensee of OpenGL®, Open Inventor® from Silicon Graphics, Inc.

OpenGL® is registered trademarks of Silicon Graphics, Inc.

Open Inventor® is registered trademarks of Visualization Sciences Group, SAS.

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

This manual has been prepared for FEI Visualization Sciences Group licensees solely for use in connection with software supplied by FEI Visualization Sciences Group 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 FEI Visualization Sciences Group. Users are cautioned that FEI Visualization Sciences Group 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

I Alphabetic Index of Modules	1
1 Amira	3
1.1 2DMesh	3
1.2 Access LargeDiskData	4
1.3 AffineRegistration	6
1.4 AlignPrincipalAxes	10
1.5 AlignSlices	11
1.5.1 Tool Bar	13
1.5.2 Menu Bar	14
1.5.3 Alignment Methods	21
1.5.4 Image Viewer	23
1.5.5 Key Bindings	23
1.6 Animate	29
1.7 Annotation	30
1.8 AnonymizeImageStack	32
1.9 ApplyTransform	32
1.10 ArbitraryCut	35
1.11 Arithmetic	38
1.12 AverageVolumes	43
1.13 Axis	45
1.14 BlockFaceCorrection	46
1.15 BoundingBox	47
1.16 CalculusMatlab	48
1.17 CameraPath	56
1.18 CannyEdgeDetector	57
1.19 CastField	58
1.20 ChannelWorks	60
1.21 CityPlot	61

1.22 ClippingPlane	64
1.23 ClusterDiff	64
1.24 ClusterFilter	66
1.25 ClusterGrep	67
1.26 ClusterSample	68
1.27 ClusterStringLabels	69
1.28 ClusterView	70
1.29 CollectiveTCL	73
1.30 ColorCombine	74
1.31 Colorwash	76
1.32 CombineLandmarks	79
1.33 CompareLatticeData	80
1.34 CompassPosition	80
1.35 ComputeContours	82
1.36 ComputeDistanceToPoint	83
1.37 ConnectedComponents	84
1.38 ContrastControl	86
1.39 CornerCut	90
1.40 CorrelationPlot	91
1.41 CreateCluster	96
1.42 CreateSphere	96
1.43 Curl	97
1.44 CurvedSlice	98
1.45 Cutting Plane	101
1.46 CylinderSlice	102
1.47 DataProbe	104
1.48 Delaunay2D	110
1.49 DemoDirector	112
1.50 DemoMaker	126
1.51 DemoSequence	139
1.52 Digital Image Filters	142
1.53 DisplayColormap	142
1.54 DisplayDate	144
1.55 DisplayLogos	146
1.56 DisplayTime	147
1.57 ExtractSurface	149
1.58 FilteredObliqueSlice	150
1.59 GetCurvature	152

1.60	GridView	154
1.61	Grouping	155
1.62	HeightField	157
1.63	Histogram	159
1.64	IlluminatedLines	162
1.65	InterpolateLabels	163
1.66	Intersect	164
1.67	Isolines	165
1.68	Isolines (Surface)	168
1.69	Isosurface (Regular)	170
1.70	IvDisplay	172
1.71	IvToSurface	173
1.72	LabelSpatialGraph	174
1.73	LabelVoxel	175
1.74	LandmarkSurfaceWarp	177
1.75	LandmarkView	177
1.76	LandmarkWarp	179
1.77	LatticeAccess	181
1.78	LegoSurfaceGen	183
1.79	LineProbe	183
1.80	LineSetProbe	183
1.81	LineSetToSpatialGraph	185
1.82	LineSetView	185
1.83	MaterialStatistics	189
1.84	Measurement	192
1.85	Merge	195
1.86	MovieMaker	197
1.87	MoviePlayer	201
1.87.1	The Amira Movie Format	201
1.87.2	Optimized Amira Movies	201
1.87.3	Which Movie or Image Format Should I Use ?	202
1.88	NormalizeImage	205
1.89	ObliqueSlice	206
1.90	OrthoSlice	210
1.91	OrthoViewCursor	213
1.92	ParallelMovieMaker	215
1.93	PlotSpreadSheet	218
1.94	PointProbe	219

1.95 PointWrap	219
1.96 ProbeToLineSet	221
1.97 ProjectionView	221
1.98 ProjectionViewCursor	224
1.99 RefineTetras	225
1.100 Relabel	227
1.101 RelabelTetras	228
1.102 Resample	230
1.102.1 Resampling Non-labeled Data Fields	230
1.102.2 Resampling Labeled Data Fields	231
1.102.3 Coordinates of the Resampled Data Set	231
1.102.4 Resampling in Progress	231
1.103 SQLite	233
1.104 Scale	234
1.105 ScanConvertSurface	236
1.106 SelectLines	237
1.107 SelectRoi	239
1.108 SeriesControl	240
1.109 Shear	240
1.110 ShortestEdgeDistance	241
1.111 ShowViewerSpreadSheet	242
1.112 SmoothSurface	243
1.113 Sound	244
1.114 SpatialGraphStatistics	245
1.115 SpatialGraphToLineSet	246
1.116 SpatialGraphView	246
1.117 SplineProbe	250
1.118 SplitVolume	250
1.119 StandardView	251
1.120 SurfaceArea	253
1.121 SurfaceCut	254
1.122 SurfaceDistance	257
1.123 SurfaceField	259
1.124 SurfaceGen	259
1.125 SurfaceIntersector	262
1.126 SurfacePathView	262
1.127 SurfaceThickness	264
1.128 SurfaceView	264

1.129	TimeSeriesControl	268
1.130	Trajectory	270
1.131	TransformAnimation	272
1.132	TriangleDistortion	273
1.133	TridelityView	274
1.134	VRML-Export	277
1.135	Vertex Morph	280
1.136	VertexDiff	281
1.137	VertexShift	281
1.138	VertexView	282
1.139	ViewerPlot	286
1.140	Volren	288
1.141	Voltex	292
1.142	VolumeEdit	296
1.143	VoxelView	298
2	Molecular Option	301
2.1	AlignMolecules	301
2.2	AlignSequences	303
2.3	AtomicMolecularDensity	306
2.4	BondAngleView	310
2.5	BondCalculation	312
2.6	CompMolInterface	313
2.7	CompMolSurface	314
2.8	ComputeHBonds	318
2.9	ComputeSecondaryStructure	319
2.10	ConfigurationDensity	320
2.11	HBondView	323
2.12	MeanMolecule	325
2.13	Measurement	327
2.14	MolElectrostatics	329
2.15	MolOptimizer	331
2.16	MolSurfaceView	332
2.17	MolTrajBundleConverter	335
2.18	MoleculeLabel	336
2.19	MoleculeView	339
2.20	NoncovalentInteractionGrid	342
2.21	Observables	343

2.22	PrecomputeAlignment	345
2.23	PseudoElectronDensity	346
2.24	RankTimeStep	347
2.25	SecStructureView	349
2.26	TubeView	355
3	Virtual Reality Option	357
3.1	ShowConfig	357
3.2	VRSettings	359
4	Very Large Data Option	365
4.1	ArithmetricRendering	365
4.2	CastLattice	365
4.3	ConvertToLargeDiskData	367
4.4	IsosurfaceRendering (LDA)	368
4.5	LDAExpertSettings	369
4.6	ObliqueSlice (LDA)	373
4.7	OrthoSlice (LDA)	376
4.8	SelectRoi (LDA)	379
4.9	Voltex (LDA)	382
5	Mesh Option	387
5.1	AlignSurfaces	387
5.2	BoundaryConditions	389
5.3	ComponentField	390
5.4	ConePlot	391
5.5	ContourView	394
5.6	Displace	395
5.7	DisplayISL	396
5.8	Divergence	400
5.9	DoseVolume (Tetrahedra)	401
5.10	DuplicateNodes	402
5.11	ExtractEigenvalues	403
5.12	FieldCut	403
5.13	FieldCut (Polyhedra)	406
5.14	Gradient	408
5.15	GridBoundary	409
5.16	GridCut	411

5.17	GridCut (Polyhedra)	413
5.18	GridToSurface	414
5.19	GridView (Block Structured)	415
5.20	GridVolume (Hexahedra)	416
5.21	GridVolume (Polyhedra)	418
5.22	GridVolume (Tetrahedra)	420
5.23	HexToTet	422
5.24	HexaQuality	423
5.25	Interpolate	425
5.26	Isosurface (Hexahedra)	427
5.27	Isosurface (Polyhedra)	429
5.28	Isosurface (Tetrahedra)	431
5.29	Lambda2	433
5.30	LatToHex	433
5.31	LineStreaks	434
5.32	MagAndPhase	435
5.33	Magnitude	435
5.34	ParametricSurface	436
5.35	ParticlePlot	439
5.36	PlanarLIC	442
5.37	RateOfStrainTensor	447
5.38	RemeshSurface	447
5.39	SampleScalarField	452
5.40	SeedSurface	456
5.41	Splats	458
5.42	StreamRibbons	460
5.43	StreamSurface	462
5.44	SurfaceLIC	464
5.45	TensorDisplay	467
5.46	TetToHex	472
5.47	TetraCombine	472
5.48	TetraGen	473
5.49	TetraQuality	476
5.50	TetraVectors	478
5.51	TriangleQuality	480
5.52	VectorProbe	481
5.53	Vectors	483
5.54	Vectors/Normals (Surface)	485

6 Microscopy Option	489
6.1 BeadExtract	489
6.2 Convolution	491
6.3 CorrectZDrop	492
6.4 DataPreprocess	493
6.5 Deconvolution	494
6.6 DistanceMap	497
6.7 FourierTransform	498
6.8 PSFGen	500
7 Quantification+ Option	503
7.1 Quantification	503
7.2 Quantification-Analyse	508
7.3 Quantification-Threshold	509
8 Neuro Option	511
8.1 ComputePerfusion	511
8.2 ComputeTensor	515
8.3 ComputeTensorOutOfCore	517
8.4 ConvertTalairach	519
8.5 CreateGradientImage	520
8.6 EigenvectorToColor	522
8.7 FiberTracking	523
8.8 SegmentBrain	526
9 DICOM Reader	529
9.1 DicomSend	529
II Alphabetic Index of Ports	531
10 Amira	533
10.1 Port	533
10.2 Connection	536
10.3 MasterConnection	537
10.4 Port3DPointList	538
10.5 PortButtonList	542
10.6 PortButtonMenu	544

10.7	PortChannelConfig	545
10.8	PortColorList	546
10.9	PortColormap	547
10.10	PortDoIt	550
10.11	PortDrawStyle	551
10.12	PortFilename	552
10.13	PortFloatSlider	553
10.14	PortFloatTextN	556
10.15	PortFontSelection	558
10.16	PortGeneric	559
10.17	PortInfo	562
10.18	PortIntSlider	562
10.19	PortIntTextN	563
10.20	PortMultiChannel	563
10.21	PortMultiMenu	564
10.22	PortRadioBox	565
10.23	PortSeparator	566
10.24	PortSharedColormap	566
10.25	PortTabBar	567
10.26	PortText	568
10.27	PortTime	568
10.28	PortToggleList	571
III	Alphabetic Index of File Formats	573
11	Amira	575
11.1	Amira Script	575
11.2	Amira Script Object	575
11.3	AmiraMesh Format	575
11.3.1	Fields with Uniform Coordinates	579
11.3.2	Fields with Stacked Coordinates	580
11.3.3	Fields with Rectilinear Coordinates	581
11.3.4	Fields with Curvilinear Coordinates	582
11.3.5	Label Fields for Segmentation	583
11.3.6	Landmarks for Registration	585
11.3.7	Line Segments	586
11.3.8	Colormaps	587

11.3.9 Spatial Graph	588
11.4 AmiraMesh as LargeDiskData	590
11.5 Analyze 7.5	590
11.6 AnalyzeAVW	591
11.7 BMP Image Format	591
11.8 DXF	592
11.9 Encapsulated PostScript	592
11.10HTML	592
11.11Hoc	593
11.12HxSurface	595
11.13Icol	601
11.14Interfile	601
11.15JPEG Image Format	601
11.16MATLAB Binary Format (.mat)	602
11.17MATLAB M-files Format (.m)	602
11.18Nifti	602
11.19Open Inventor	603
11.20PNG Image Format	604
11.21PNM Image Format	604
11.22PSI format	604
11.23Ply Format	606
11.24Raw Data	606
11.25Raw Data as LargeDiskData	608
11.26SGI-RGB Image Format	609
11.27STL	609
11.28SWC	609
11.29Stacked-Slices	610
11.30TIFF Image Format	611
11.31Tecplot	613
11.32VRML	614
11.33Veo Mode Raw Images	614
11.34Wavefront Technologies 3D Geometry (.obj)	615
12 Molecular Option	617
12.1 AMBER	617
12.2 AMF	618
12.3 DX	621
12.4 GROMACS	622

12.5 MAP	623
12.6 MDL	623
12.7 PDB	623
12.8 PHI	626
12.9 PSF/DCD (CHARMM)	626
12.10Tripos	626
12.11UniChem	627
12.12ZIB Molecular File Format	627
13 Very Large Data Option	635
13.1 LDA	635
13.2 LargeDiskData	635
13.3 Stacked-Slices as LargeDiskData	635
14 Mesh Option	637
14.1 AVS Field	637
14.2 AVS UCD Format	637
14.3 Abaqus format	638
14.4 FIDAP NEUTRAL	638
14.5 Fluent / UNS	639
14.6 Hypermesh	639
14.7 IDEAS universal format	639
14.8 Plot 3D Single Structured	640
15 Microscopy Option	643
15.1 Bio-Rad Confocal Format	643
15.2 FEIStackedScalarField3	643
15.3 FEIUniformScalarField3	643
15.4 Leica 3D TIFF	643
15.5 Leica Binary Format (.lei)	644
15.6 Leica Image Format (.lif)	644
15.7 Leica Slice Series (.info)	644
15.8 MRC	644
15.9 Metamorph STK Format	645
15.10Olympus (.oib/.oif)	645
15.11Zeiss LSM	645

16 DICOM Reader	647
16.1 ACR-NEMA	647
16.2 DICOM export	647
16.3 DICOM import	648
IV Alphabetic Index of Data Types	655
17 Amira	657
17.1 AnnaScalarField3	657
17.2 AnnaVectorField3	658
17.3 B-Splines	661
17.4 BlockStructuredGrid3	661
17.5 BlockStructuredScalarField3	661
17.6 BlockStructuredVectorField3	662
17.7 CameraRotate	662
17.8 Cluster	663
17.9 ColorField3	665
17.10 Colormap	666
17.11 Data	669
17.12 Field3	670
17.13 HexaGrid	671
17.14 HexaScalarField3	672
17.15 IvData	672
17.16 LandmarkSet	672
17.17 LargeDiskData	675
17.18 Lattice3	675
17.19 Light	677
17.20 LineSet	680
17.21 Movie	682
17.22 MultiChannelField3	685
17.23 Object	686
17.24 RawAsExternalData	689
17.25 RegScalarField3	689
17.26 ScalarField3	689
17.27 ScriptObject	689
17.28 SpatialData	694
17.29 SpatialGraph	698

17.30SpreadSheet	698
17.31StackedLabelField3	698
17.32StackedScalarField3	700
17.33Surface	700
17.34Surface Path Set	703
17.35TetraGrid	703
17.36TetraScalarField3	704
17.37Time	704
17.38UniformLabelField3	705
17.39UniformScalarField3	707
17.40VertexSet	707
17.41ViewBase	708
18 Molecular Option	715
18.1 MolSurface	715
18.2 MolTrajectory	716
18.3 MolTrajectoryBundle	720
18.4 Molecule	723
19 Very Large Data Option	739
19.1 VolumeDataObject	739
20 Mesh Option	741
20.1 AnalyticTensorField	741
20.2 PbScalarField3	742
20.3 PolyhedralGrid	742
21 Skeleton Option	743
21.1 Mosaic	743
V Alphabetic Index of Editors	745
22 Amira	747
22.1 Advanced Colormap Editor	747
22.1.1 Description of the User-interface Elements	748
22.1.2 How to Modify a Colormap	752
22.2 CameraPath Editor	754

22.3	Color Dialog	757
22.4	Colormap Editor	759
22.4.1	Overview	760
22.4.2	Menu Bar	760
22.4.3	Global Settings	760
22.4.4	Histogram	761
22.4.5	Gradient Editor	762
22.4.6	Alpha Curve	763
22.4.7	Color Chooser	763
22.4.8	Opacity Chooser	763
22.4.9	Control Buttons	764
22.5	Curve Editor	764
22.6	Demo GUI	765
22.7	Digital Image Filters	769
22.7.1	Noise reduction minimum filter (Minimum)	771
22.7.2	Noise reduction maximum filter (Maximum)	771
22.7.3	Unsharp masking	771
22.7.4	Laplacian edge detection filter (Laplacian zero-crossing)	771
22.7.5	Noise reduction median filter (Median)	772
22.7.6	Gaussian smoothing filter (Gauss)	772
22.7.7	Sobel edge detection filter	772
22.7.8	Equalize filter (Histogram)	772
22.7.9	Edge-Preserving smoothing	773
22.7.10	Non-local means filter	773
22.7.11	Resampling/Low pass filter (Lanczos, Phase 0)	777
22.7.12	Intensity remapping filter (Sigmoid)	777
22.7.13	Brightness and contrast filter	777
22.7.14	Statistical feature detection filter (Moments)	778
22.7.15	Lighten/Darken filter (Gamma correction)	778
22.8	ImageCrop Editor	778
22.8.1	Cropping an Image by Dragging and Moving the Box	780
22.8.2	Cropping an Image by Setting Values Explicitly in the Text Fields	780
22.8.3	Adding New Slices	780
22.8.4	Changing the Size of the Bounding Box	781
22.8.5	Flipping Slices in One Dimension	781
22.8.6	Exchanging Two Dimensions	781
22.9	Landmark Editor	782

22.10 LineSet Editor	783
22.11 Multiplanar Viewer	784
22.11.1 Overview of the Multiplanar Viewer	784
22.11.2 The Viewers	788
22.12 Parameter Editor	789
22.13 Plot Tool	790
22.13.1 Plot Basics	791
22.13.2 Editing Parameters	792
22.13.3 Working with Plot Objects	796
22.13.4 Printing	797
22.13.5 Saving Data	797
22.13.6 Saving the Plot State	797
22.13.7 Specific Option	797
22.14 Segmentation Editor	800
22.14.1 Overview of the Segmentation Editor	801
22.14.2 Manipulating the Material List	804
22.14.3 Working in 4-viewer Mode	807
22.14.4 Selection Tools	809
22.14.5 Segmentation Tools	811
22.14.6 Selection Filters	815
22.14.7 Label Filters	817
22.14.8 Segmentation Editor - Key Bindings	819
22.15 Simplification Editor	820
22.16 Surface Editor	823
22.16.1 Menu Entries	825
22.16.2 Selectors	828
22.16.3 Tools	829
22.17 Surface Path Editor	832
22.18 Transform Editor	837
23 Molecular Option	841
23.1 MolTrajectoryBundle Editor	841
23.2 Molecule Attribute Editor	842
23.3 Molecule Data Editor	849
23.4 Molecule Editor	850
24 Mesh Option	861
24.1 Grid Editor	861

25 Microscopy Option	867
25.1 Filament Editor	867
25.1.1 Overview of the Filament Editor	867
25.1.2 Basic principles and terminology	869
25.1.3 The Tool Box	870
25.1.4 The Viewers	876
25.1.5 Working with the Label Editor	882

Part I

Alphabetic Index of Modules

1 Amira

1.1 2DMesh

This module creates a NxM surface mesh of squares. The module provides a basic Tcl interface to define the dimensions of the mesh and to move points. Each point can have an associated value which is mapped as a color.

In order to set the location of each grid point the Tcl command *setPoint* needs to be used.

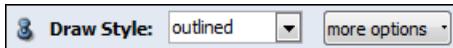
Connections

Colormap [optional]

Connection to the colormap which is used to map the values of each vertex.

Ports

Draw Style



Specify the draw style of the surface.

Colormap



Specify properties of the colormap used to map values of each vertex to a color.

Position



Shift the initial position of the mesh.

Dimension



The dimensions of the quad-mesh.

Commands

`getPoint <i> <j>`

Returns the spatial location of the grid point (*<i>*, *<j>*) and its value.

`setPoint <i> <j> <x> <y> <z> [value]`

Sets the location and value of the grid location (*<i>*, *<j>*) to *<x>*, *<y>*, *<z>*. Optionally the value can be specified as well.

`show`

Enables the display of the grid in the viewer.

`hide`

Disables the display of the grid in the viewer.

`getSize`

Returns the grid dimensions currently used.

`setColor <i> <j> <value>`

Set the value of grid point (*<i>*, *<j>*).

`getColor <i> <j>`

Returns the value associated to the current grid location.

1.2 Access LargeDiskData

This module is deprecated. Please use *LatticeAccess* instead.

This kind of module is attached to a *LargeDiskData* object. provides an interface to select a subblock of the large volume 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 volume and an immediate reload is started. This might be useful to easily explore a large volume.

Connections

Data [required]

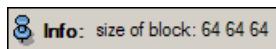
Connection to the *LargeDiskData* object.

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



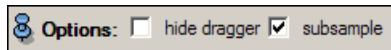
The position of the lower corner of the block. Printed as a lattice index.

BoxSize



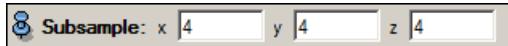
The size of the block to be loaded. The size is in lattice indices in the base resolution. The numbers will not change if you select subsampling but the real size of the block to be loaded will change. This is indicated in the *Info* port.

Options



Hides the dragger and/or enables subsampling.

Subsample



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

1.3 AffineRegistration

This module computes an affine transformation for the co-registration of two image data sets, using an iterative optimization algorithm. A hierarchical strategy is applied, starting at a coarse resampling of the data set, 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.

Connections

Model [required]

The model data set to be transformed.

Reference [required]

The reference data set to which the model is registered.

Reference2 [optional]

See *Reference3* below.

Reference3 [optional]

The connection ports *Reference2* and *Reference3* are used for a particular type of images routinely taken in MRI, so-called *Localizers*. A Localizer consists of few ($\tilde{3}$) slices each of an axial/xy, sagittal/yz, and coronal/yz scan. A CT or MR taken from the same patient or object can be registered to the Localizer images by using high resolution information for each direction in separate volumes. To

do so, connect the two additional Reference ports with the Localizer images. Note that the port *Options2: Localizer* automatically gets checked.

Ports

Optimizer

 Optimizer:	Extensive direction	<input type="button" value="▼"/>	Ramp	<input type="button" value="▼"/>
--	---------------------	----------------------------------	------	----------------------------------

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 *LineSearch* 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

 Optimizer step:	initial: 40.1569	final: 0.261438
---	------------------	-----------------

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 data sets 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.

Gradient optimizer

 Gradient optimizer:	Finest levels: 1	Tolerance: 0.0001
---	------------------	-------------------

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

 Coarsest resampling:	x 4	y 4	z 2
--	-----	-----	-----

At this port you can define the resampling rate for the coarsest resolution level where registration starts. The resampling rate refers to the *reference* data set. If

the voxels of the reference data set 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.

Metric

	Metric:	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 images from different modalities, e.g., CT and MRT, are to be registered.

(Histogram) range reference

	Histogram range reference:	min	22	max	254
--	-----------------------------------	-----	----	-----	-----

This port is only active if one of the *Mutual information* metrics, or the *Correlation* metric has been selected. Here you can define the range of gray values for the *reference* data set. For the *Mutual information* metrics, gray values outside the range are sorted into the first or last histogram bin, respectively. For the *Correlation* metric, voxels with gray values outside the range are ignored. It is a good idea to determine the range via a visualization of the data set, e.g., using an *OrthoSlice* module.

(Histogram) range model

	Histogram range model:	min	21	max	243
--	-------------------------------	-----	----	-----	-----

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

Histogram bins

	Histogram bins:	reference:	232	model:	222
--	------------------------	------------	-----	--------	-----

For the *Normalized Mutual Information* and *Mutual Information* metric it is possible to set the number of bins for model and reference histograms. The number of bins basically determines the resolution by which the range of gray values set in ports *Histogram range reference/model* is partitioned. The more bins the more accurate the gray values are resolved but the more computation time is needed. The default values set by the module are usually a good compromise.

Options

	Options:	<input checked="" type="checkbox"/> Ignore finest resolution
--	-----------------	--

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.

Options2

	Options2:	<input type="checkbox"/> Localizer
--	------------------	------------------------------------

This port is automatically checked upon connecting ports *Reference2* and/or *Reference3* with localizer images. If checked *AffineRegistration* will use those images during registration.

Transform

	Transform:	<input checked="" type="checkbox"/> Rigid	<input type="checkbox"/> Iso-	<input type="checkbox"/> Aniso-Scale	<input type="checkbox"/> Shear
--	-------------------	---	-------------------------------	--------------------------------------	--------------------------------

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 anisoscale transformation. 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.

Register

	Register:	<input type="radio"/> 2D	<input checked="" type="radio"/> 3D
--	------------------	--------------------------	-------------------------------------

Select here the registration mode. If *2D* is checked, the search is restricted to transformations within the same plane. In the case *3D* is checked a full 3D registration is performed.

Extended options

	Extended options:	<input checked="" type="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*, *Gradient optimizer*, *CoarsestResampling*, *Histogram range reference*, *Histogram range model*, *Options*, *Options2*, and *Register* become visible.

Action



If the *Align Centers* button is pressed, the centers of gravity of both data sets are computed, taking the image intensity as a mass density. The model data set 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 data sets 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.

1.4 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]

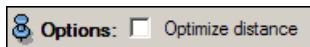
Model surface that will be transformed.

Reference [optional]

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

Ports

Options



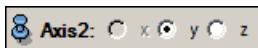
This port is only visible when a reference surface is connected. For each possible solution (see explanation below at *Port Action*), the root mean square distance is computed when the option 'Optimize distance' is chosen. The final transformation will be the one yielding the minimum value.

Axis1



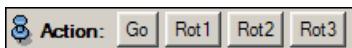
If no reference surface is connected, the principal axes with the largest moment of inertia will be aligned to the axis specified at this port.

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 to 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.

1.5 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.

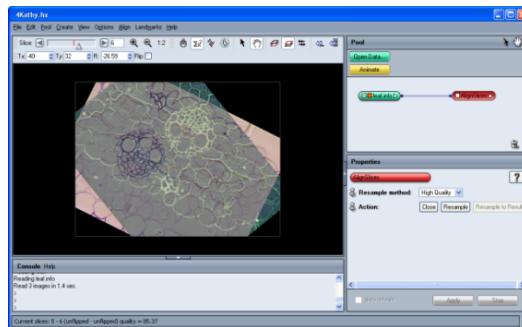


Figure 1.1: User interface of the align tool.

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 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 quality value is based on the sum of squared gray-value difference (SSD) between the two slices being aligned. The SSD is computed for all pixels in the area in which the two slices overlap (the size of this area depends on the transformation of the slices). The SSD is then normed, so that it lies between 0 and 1. The quality in *alignSlices* is 1 minus this value, and therefore also lies between 0 (0%/bad) and 1 (100%/best).

The *AlignTransform* parameters are written into the file header of the aligned stack (see *Port Action* below and *Save transformation* in the *Options* menu) and can be viewed with the Parameter Editor. For example,

```
slice044 6 4 -16.3025 -1
```

indicates that slice # 44 was translated 6 voxels in X, -4 voxels in Y, and rotated 16.3025 degrees about Z. The -1 means that the slice was not mirrored. (If 1, it would mean that the slice has been mirrored.)

The documentation of the align tool is organized into the following sections:

- *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.

1.5.1 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.

1.5.2 Menu Bar

1.5.2.1 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.

1.5.2.2 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 transforma-

tions 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.

1.5.2.3 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*.

1.5.2.4 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

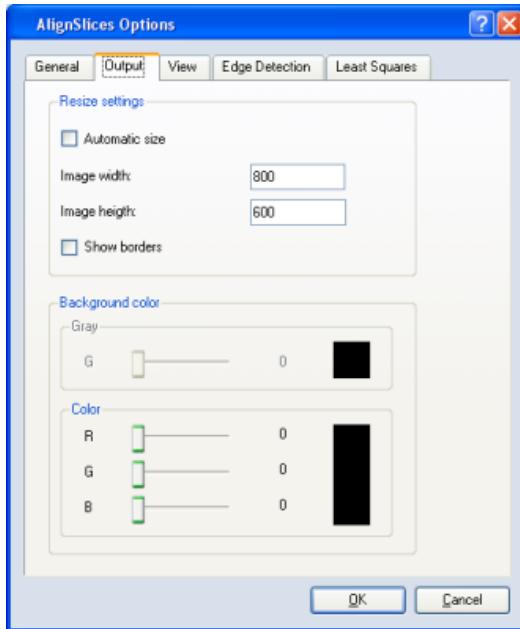


Figure 1.2: The *Output options* window.

checked. Translations are always considered.

1.5.2.5 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 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.

1.5.2.6 Align/Options/View

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

1.5.2.7 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.
- **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

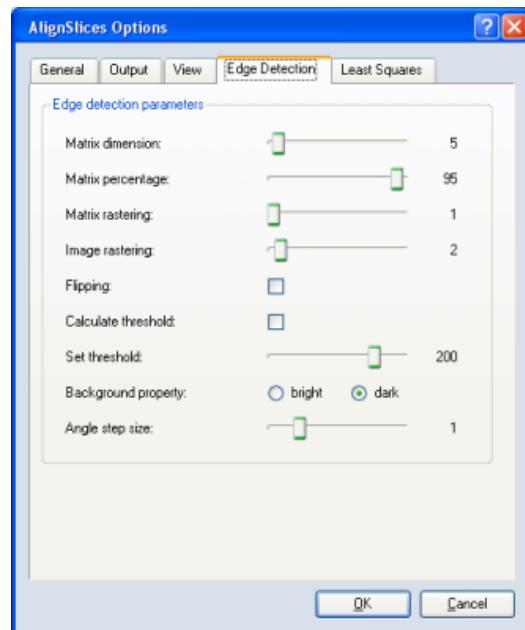


Figure 1.3: The *Edge detection options* window.

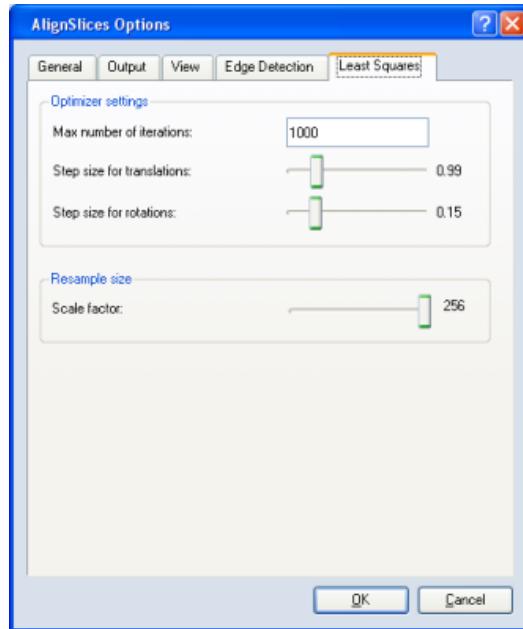


Figure 1.4: The *Least squares options* window.

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.

1.5.2.8 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*.

1.5.3 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 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.

1.5.3.1 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.

1.5.4 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**.

1.5.5 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.

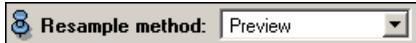
Ports

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* to the left, and by *ty* downwards.

```
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*.

```
snapshot [filename]
```

Take a snapshot of the slice alignment viewer.

```
edit
```

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

1.6 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 that 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

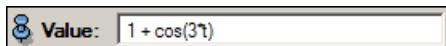
and the increment used for animation can be adjusted in the configure dialog which can be popped up using the right mouse button.

Port



Option menu listing all ports of the connected object that 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 $((t > 5) \&\& (t < 15)) \mid\mid (t > 100)$.

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.

1.7 Annotation

This module displays a 2D text string (with Unicode support) in the 3D viewer. Position and color of the text can be specified by the user. This is useful for anno-

tating snapshots. For better results, use a 1x1 tiling for the snapshot. To create the module, choose *Annotation* from the main window's *Create* menu. For color legends refer to the module *Display Colormap*.

Ports

Background

	Background:	<input type="checkbox"/> transparent	<input checked="" type="checkbox"/> color
--	--------------------	--------------------------------------	---

Text's background color. If the transparent toggle is off, the annotation text will be drawn inside a filled rectangle. The background color of this rectangle can be set via the color button of this port.

Position type

	Position type:	<input checked="" type="radio"/> in pixels	<input type="radio"/> in normalized screen coordinates (0..1)
--	-----------------------	--	---

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

A positive x-coordinate moves the annotation text to the right, a negative moves it to the left. Likewise, a positive y-coordinate moves the text upwards and a negative coordinate downward. The text is always moved relative to the defined origin described next.

Origin (0,0)

	Origin (0,0):	upper left	
--	----------------------	------------	--

Defines the origin used to calculate the position on the screen. The four corners of the screen can be chosen.

Position (pixels)

	Position (pixels):	x	20	y	-20
--	---------------------------	---	----	---	-----

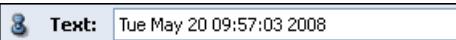
Text position must be given in screen pixels.

Position (normalized)

	Position (normalized):	x	0.04	y	-0.04
--	-------------------------------	---	------	---	-------

The position is given as a fraction of screen width/height, that is, a number between 0 and 1.

Text



The text to be displayed. This port supports Unicode character strings. This enables to use every alphabet for all languages in the world.

Font



This port lets you specify the font and font attributes used by the displayed text.

1.8 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.

1.9 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 voxel size will be retained although the size seems to be different on the screen. If you rotate a field, the directions of the sampling planes will still be 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. Afterward 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 *ApplyTransform* 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* interpolates linearly 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 voxel size or the dimensions have to be changed. If you select this mode, the *Preserve* port is made visible.

The *Preserve* port:

- *Voxel Size*: Adjust the result field's dimensions in a way that it has the same voxel size 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 have a transformation attached to it 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]

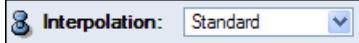
The source field. The module operates on any regular 3D field with uniform coordinates and an arbitrary number of data components per node.

Reference [optional]

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

Ports

Interpolation



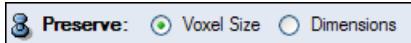
The interpolation method that will be used.

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.

PaddingValue



The value used to fill the new cells when the result is *extended*.

1.10 ArbitraryCut

This module defines a plane that 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* and *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.

PointsToFit [optional]

Connection to a vertex set with arbitrary number of points, on which the position of the plane is based. The option "fit to points" is only available, if you have not connected your own vertex set.

Ports

Orientation



This port provides three buttons for resetting the plane's orientation. *Axial*/xy slices are perpendicular to the z-axis, *coronal*/xz slices are perpendicular to the y-axis, and *sagittal*/yz slices are perpendicular to the x-axis. The plane will be translated into the center of the bounding box.

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.

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 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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 numbers in the same format as returned by the getPlane command.

```
setMinPlanePoint <point>
```

Define the minimum plane position.

```
unsetMinPlanePoint <point>
```

Reset the minimum plane position.

```
setMaxPlanePoint <point>
```

Define the maximum plane position.

```
unsetMaxPlanePoint <point>
```

Reset the maximum plane position.

```
setAutoClippingPlane <state>
```

Set the clipping plane auto mode to state. If auto mode is set, the clipping plane is adjusted to clip the front of the plane related to the camera position, otherwise the clipping plane uses the plane normal.

```
connectPlane <ArbitraryCut>
```

Interconnect origin and normal ports to given module so that both planes are synchronized.

```
disconnectPlane <ArbitraryCut>
```

Disconnect origin and normal ports from given module if previously connected.

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 any number of 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.
- $\text{A1} \dots \text{An}$: Numbered component access of input A.
- 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 the 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* and *or*. 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*, *or*, and *xor*. 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. For bitwise *xor*, the bits in the result are set to 1 if at exactly 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 $\hat{\text{o}}$ perator exists but means bitwise *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 fol-

low:

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 two 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

 **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), 6 channels (either a complex vector or a symmetric tensor field of second order), or a user defined number of channels.

NValues

 **NValues:**

Number of user-defined output channels.

Expr

 **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

 **Result type:** input A regular

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

 **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

 **Min box:**

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

	Max box:	1	1	1
--	-----------------	---	---	---

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

Result location

	Result location:	<input checked="" type="radio"/> on Nodes	<input type="radio"/> on Cell Center
--	-------------------------	---	--------------------------------------

Specify the location of the result. Produces either a value for every vertex of the grid (on Nodes), or for every grid cell (on Cell Center). This port is available if the first input connection is of type *HxSurface*, *HxTetraGrid* or *HxHexaGrid*.

Options

	Options:	<input type="checkbox"/> ignore errors
--	-----------------	--

Ignore errors: Tell the Arithmetic module to ignore errors. For example, this can be useful to force the computation of divisions if the data contains zeros.

1.12 AverageVolumes

This module computes the per-voxel average of an arbitrary number of 3D image data sets or label fields. The data sets are loaded from disk successively so that they do not have to fit into main memory simultaneously. A transformation per data set is taken into account. The transformation matrix needs to be stored along with each data set as a parameter.

If label fields are to be averaged, two modes are provided which are described below.

Connections

Data [required]

The data set connected to this port defines location and resolution of the resulting field. In the case of label averaging, this data must be a label field.

Ports

Mode



Four different modes are available:

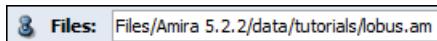
- **Average** computes the gray value average per voxel.
- **Avg+Std.Dev.** computes the gray value average and the standard deviation per voxel. This requires twice as much time as the *Average* option.
- **Label** computes the probability map for one specific label which must be specified. The probability map contains for each voxel the probability for that material voxel to occur at that voxel. The output ranges from 0 to 100 where 100 stands for a probability of 1.
- **BestLabel** computes for each voxel the label which is most likely to occur at that point. Additionally it creates a probability map which contains the probability for the specific label at each voxel.

Material



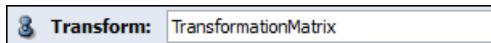
This port is only visible when port *Mode* is set to *Label* and it specifies the material to be averaged. The numbers correspond to the material IDs as defined in the *Segmentation Editor*. **0** corresponds to *Exterior*, **1** corresponds to the first material in the list, and so forth.

Files



A list of files to be processed separated with space. This list can be filled using the *Choose Files* button explained below.

Transform



The name of a parameter holding the 16 parameters of a transformation matrix. This parameter should be present in all data sets. If no parameter with that name is found the identity matrix is assumed.

In order to set such a parameter you can follow this recipe: After you have aligned a data set to your template (e.g. by using the transform editor or using module *AffineRegistration*, you need to save it to disk. Doing so writes a new parameter *TransformationMatrix* into the parameter section of the data object.

If you need to set the parameter to multiple objects you may also use the following TCL commands in a script:

```
eval <dataset> parameters setValue <parameter name>
{<t00 <t01> <t02> <t03> <t10> <t33>} where {<t00>
<t01> <t02> <t03> <t10> ... <t33>} refers to the 16 values of the transformation matrix, <dataset> to the name of the particular data set and <parameter name> to the name you have chosen for the parameter, like e.g. MyTransformation).
```

Choose



Clicking on this button will open a file browser which allows you to select multiple files to be averaged.

1.13 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.

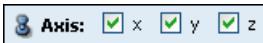
Connections

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



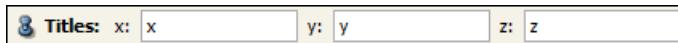
Lets you adjust the thickness of the axes.

Colors



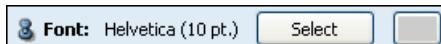
Provides buttons for changing the colors of different parts of the geometry. By default, the x-, y-, and z-axes are drawn in red, green, and blue, respectively.

Titles



This port lets you edit axis names. Default axis names are *x*, *y*, *z*.

Font



This port lets you specify the font and font attributes used when displaying text.

Commands

`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.

1.14 BlockFaceCorrection

This module corrects for slice-based intensity fluctuations, which often occur with, but are not restricted to, block face scanning data. To use the module, the user

must create a *LabelField* and connect it to the LabelField connection port. This *LabelField* specifies on each slice a homogeneous region (e.g., by using the *Segmentation Editor*). Clicking *Apply* starts the computation of the mean intensity for the segmented region in each slice, and the global mean for all slices. Each slice is then linearly scaled to match the global mean. The output is the corrected data and has the same size as the input data. Depending on the imaged object, the region segmented on each slice can belong to any homogeneous region such as identical tissue types. If the *LabelField* contains two materials, it is assumed that each material specifies voxels of two different homogeneous regions (e.g., one for background and the other for foreground). In this case, *BlockFaceCorrection* calculates a linear regression from the means and uses the mean slope for correcting the slices. In the case of only one material, only the means are used for the correction.

Connections

Data [required]

The image data to be corrected.

Label field [required]

A *LabelField* containing one or two non-Exterior materials specifying on each slice voxels of a homogeneous region. These homogeneous regions should contain identical tissue/material types with a minimum of 30 voxels in each slice.

1.15 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 extent in 3D. More precisely, it is obtained by determining the minimum and maximum coordinates in the x, y, and z directions.

Connections

Data [required]

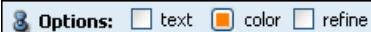
Any spatial data object.

Projection [optional]

Any *Projection* object.

Ports

Options



This port allows you to select several options:

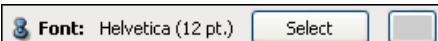
- *text*: Show/hide the coordinates of the lower left front and upper right back corner of the box.
- *color*: Select the color of the box.
- *refine*: Refine the bounding box by subdividing it. The bounding box is initially composed of 8 points and 12 edges. When setting refine mode to true, the bounding box is refined: 8 lines are added in the middle of each original line. When using projections and bounding box refinement, the lines are subdivided so that the projected bounding box is well defined.

Line width



This port allows you to modify the line width of the box.

Font



This port allows you to specify the font and font attributes used when displaying text. Note that this port is shown only when the "*text*" option is activated.

1.16 CalculusMatlab

This module is available for Windows (32-bit and 64-bit), Linux, and Mac OS X. 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.

HxLattice3 objects are stored as 3D/4D arrays A, A1, ... , An ($n > 1$) in order of connection or optional in structs named 'field'. If several HxLattice3 objects are transferred, the name includes a counter 'AID' or 'fieldID'.

The struct looks like:

field.data	- 3D/4D array of data values (scalar/vector);
field.coordsType	- string indicating the type of coordinates
field.bbox	- 1D array of bounding box (xmin xmax ymin ymax zmin zmax)
field.name	- string of the object name
field.type	- primeType of the object

HxSurface objects are stored in a struct named 'surf'. If several HxSurface objects are transferred, the name includes a counter 'surfID'.

The struct looks like:

surf.vertices	- 2D array of coordinates (numPoints x 3)
surf.faces	- 2D array of vertex-IDs (numTriangles x 3)
surf.FaceVertexCData	- 2D array of triangle vertex-colors (numTriangles x 3)
surf.FaceColor	- string 'flat'
surf.EdgeColor	- string 'none'
surf.bbox	- 1D array of bounding box (xmin xmax ymin ymax zmin zmax)
surf.type	- primeType of the object
surf.name	- string of the object name

HxSurfaceField objects are stored in a struct named 'surf'. If several HxSurface-Field objects are transferred, the name includes a counter 'surfID'. The underlying surface is transferred. The structure is the same as for HxSurface.

The struct looks like:

surf.data	- 2D-array of data values (nD x numPoints/numTriangles/numEdges)
-----------	---

HxLineSet objects are stored in a struct named 'lineset'. If several HxLineSet objects are transferred, the name includes a counter 'linesetID'.

The struct looks like:

```
lineset.points - 3D array of coordinates  
                  (3 x numPoints)  
lineset.lines - cell array of vectors describing the line  
lineset.data - (optional) data array  
                  (numDataPoints x numPoints)  
lineset.type - primeType of the object  
lineset.name - string of the object name
```

Note: In Amira the numbering of points starts with 0 while the numbering in MATLAB starts with 1, since C arrays start counting with 0 while MATLAB arrays start with 1.

For HxTime objects a single scalar value 't' is transferred to MATLAB. If several HxTime objects are transferred, the name includes a counter 'tID'.

The counter ID is added in order of connection to the module.

Every other object will be mapped to a struct array named the same as the object in the objectpool, where '.' and ':' are replaced by '_'. The struct entries contain information about the object's ports as name and value of the port.

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, x-dim, y-dim, z-dim

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 with data type uint8)
- X=6 is mapped to a uniform complex vector field

Structs which contain fields as above are copied back to suitable objects. If the field 'name' exists Amira tries to use this name for the resulting object. In the other case the variable name of the struct is used.

Time Sliders 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.

Important Notes:

On Linux 64-bit, the AMIRA_LD_LIBRARY_PATH environment variable should be set to <matlab_installation_path>/bin/glnxa64. The PATH environment variable should be set to <matlab_installation_path>/bin.

On Mac OS X, the AMIRA_LD_LIBRARY_PATH environment variable should be set to <matlab_installation_path>/bin/maci (for instance: /Applications/MATLAB_R2008b.app/bin/maci).

On Windows, the PATH environment variable should be set to <matlab_installation_path>/bin/<win32|win64> (for instance: C:/Program Files/MATLAB/bin/win32).

Connections

Data [required]

You can add any number of data objects to the module. Look at the description above to know how the objects are represented in MATLAB.

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

```
% A is the first connected HxLattice3 data object
% (in double precision).
% X is newly created and will be sent back to
% the application.
X = fftn(A);
magnitude = abs(X);
amplitude = unwrap(angle(X));
clear A % clear all variables that you do not wish
        % to export
```

After the script is executed the 'X', 'magnitude', and 'amplitude' variables are copied back into the Amira workspace.

Here is a list of the supported MATLAB data types and how to create each one in the MATLAB workspace.

```
Z1 = randn(10,10,10,1);           % HxUniformScalarField3
Z11 = randn(10,10,10);          % HxUniformScalarField3
Z12 = randn(1,10,10,10);         % HxUniformScalarField3
Z2 = rand(2,10,10,10);          % HxUniformComplexScalarField3
Z3 = rand(3,10,10,10);          % HxUniformVectorField3
Z4 = uint8(rand(4,10,10,10).*255); % HxUniformColorField3
Z5 = randn(5,10,10,10);          % definition of Z5 will
                                % produce an error
Z6 = randn(6,10,10,10);          % HxUniformComplexVectorField3
```

A minimal field as struct could be

```
F1.data = rand(10,10,10);
F1.bbox = [0 1 0 1 0 1];
```

A minimal lineset could be

```
LS1.points = [0 0 0; 1 1 1]';
LS1.lines{1} = [1 2];
```

describing one line from (0, 0, 0) to (1, 1, 1). Be aware of transposing the point array. The optional data array behaves the same, e.g.

```
LS1.data = [1 2 3; 4 5 6]';
```

would add 3 data values for the two points of the above lineset.

Here is a more complex example which implements anisotropic diffusion:

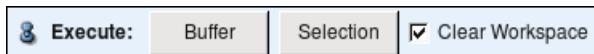
```
% anisotropic diffusion in 2D (Perona Malik)
% adapted from Peter Kovesi
erg = A;
for i=1:size(A,3),
    ima = double(reshape(A(:,:,i),size(A,1),size(A,2)));
    [r,c] = size(ima);
    diff = ima;
    niter = 4;
    kappa = 50;      % condition as function of gradient
    lambda = .25;   % speed 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.*deltaW);
    end;
    erg(:,:,:,i) = diff;
end;
clear delta* c* diff* i j niter ima lambda kappa A r;
```

This example also illustrates the time slider feature of the module here in order to control the number of iterations.

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

```
useimage=1; % which image from the stack
if exist('A') == 1, % input?
    if size(size(A),2) == 4, % input 4D?
        if size(A,1) == 4, % the first dimension
            % should be 4 (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 = (float(ima)-mi)./(ma-mi);
            imagesc(ima); % display the color image
        end;
    end;
end;
clear mi ma ima useimage A field % remove local variables
```

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*).

You can specify whether the MATLAB workspace should be cleared before running the script (*Clear Workspace*). If activated, all variables in the workspace will be deleted. Otherwise, all variables, including Amira-MATLAB transfer variables, will be kept. Note that variables with identical names will be overwritten.

You can specify a custom list of variables that will be transferred back to Amira instead of all variables. To specify variables, create the cell array 'MatlabEx-

portList'. Store names of variable to be copied to the Amira object pool as strings in the list. The transfer will be restricted to these variables. Amira will show a warning if more than 5 variables are to be transferred without explicitly specifying them in the transfer list.

Here is an example of working with a custom list:

```
a = 1; b = 2; c = 3;
MatlabExportList{1} = 'a';
MatlabExportList{2} = 'b';
```

or

```
MatlabExportList = {'a' 'b'};
```

The variables names 'a' and 'b' are stored in the cell array and will be copied to Amira. The variable 'c' will be skipped.

MatlabBuffer

```
%% Matlab script file
% Input is mapped to A, A1, A2, etc.
% Output has to be 1-, 2-, 3- or 4-D
if exist('A') == 1 && length(size(A))==3,
    imagesc(A(:,:,1));
    colormap(gray);
end;
```



Enter or change a loaded script in this buffer.

Options



Options: Display Matlab Console lattice to struct

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. This option is restricted to Windows only. Note: Even if the MATLAB interface is toggled to be visible, it may be minimized and therefore not displayed.

Warning: If you close the MATLAB interface, MATLAB is closed and the CalculusMatlab module will not work any more.

The second option 'lattice to struct' enables the storage of HxLattice3 objects as structs in MATLAB as described above.

1.17 CameraPath

This object can be created from the main window's Create menu. It lets you create a path for the cameras of all viewers activated in the object's viewer mask. This is useful in order to create complex animations. The animations can be recorded as movie files by attaching a MovieMaker module to this object.

If you are using the Camera Path Editor make sure that you interact with the camera in the main viewer to set up the key frames. The perspective camera should be used to allow changes of zoom to be recorded.

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.

1.18 CannyEdgeDetector

Classic edge detector. The algorithm consists of 4 steps:

1. Application of Gauss filter.
2. Application of Sobel filter (gradient filter).
3. Along the gradient direction (normal to edges), a non-maximal suppression (thinning) is performed.
4. A so-called "hysteresis thresholding": edges are connected with respect to lower and upper bounds provided by the user.

The result of the computation is a label field where all edge voxels are set to 1 and all others to 0. Press the *Apply* button to start the computation.

Connections

Data [required]

A uniform scalar field has to be connected.

Ports

Gauss sigma


Gauss sigma: x: y:

Sigma of the 2D Gaussian filter in units of the bounding box.

Gauss Kernel Size


Gauss Kernel Size: x: y:

Size of the 2D Gaussian filter kernel in voxels.

Suppression range


Suppression range:

Number of neighboring voxels in both gradient directions which shall be compared to each voxel.

Connect Edges Mask Size


Connect Edges Mask Size:

Parameter for the hysteresis step. A value of 3 means that edges that are separated by one voxel are considered connected.

Connect Edges Threshold



Connect Edges Threshold: Upper: 100 Lower: 1

All sobel edge values greater than *Upper* are regarded as true edges. If within their neighborhood as defined by *connectEdgesMask* there is a value greater than *Lower*, the voxel will be included in the true edges set.

1.19 CastField

This is a computational module that allows you to change the primitive data type of a regular 3D scalar field or 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]

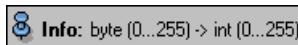
The scalar field or color field to be converted.

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

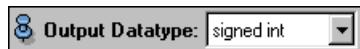


Info: byte (0...255) -> int (0...255)

Shows how the input data will be mapped during conversion. If no clipping

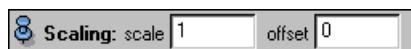
occurs, the input range covers all values between the minimum and maximum data value of the incoming scalar field. This range will be mapped as indicated. If clipping occurs, the minimum or maximum value of the output range or both will be equal respectively to the smallest or largest value that can 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*.

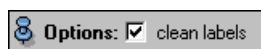
Scaling



Defines a linear transformation that 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: $I = .3 \cdot R + .59 \cdot G + .11 \cdot B$.

1.20 ChannelWorks

This module allows conversion between scalar and vector fields. You can, for example, 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.

1.21 CityPlot

The *CityPlot* module offers another method for the visualization of scalar data fields defined on regular grids, such as stacks of tomographic images. The data is visualized by extracting an arbitrary axial, frontal, or sagittal slice out of the

volume. This slice is represented as a set of bars, one for each value. The height of each bar is determined by the data value and a tunable scale parameter.

Connections

Data [required]

The scalar or color field to be visualized. If an RGBA color field is connected, the bars of the city plot are colored as in the color field. The height of the bars is determined by the grayvalue intensity which is computed using the formula $0.3*R+0.59*G+0.11*B$.

Color field [optional]

Optional scalar field used for pseudo-coloring. Only RGBA color fields or scalar fields can be connected. If an RGBA color field is connected, the colormap is not used and the bars of the city plot are colored according to the RGBA values. If a scalar field is connected, the bars are colored according to the scalar values and the colormap specified in the *Colormap* port.

Colormap [optional]

The colormap used to map data values to colors.

Texture [optional]

Not used.

ROI [optional]

Not used.

Projection [optional]

Specifies if the bars of the city plot are projected by a *Projection* module.

Filter mask [optional]

Optional scalar field used for masking bars of the city plot. Zero values mask bars of the city plot. The size of the scalar field should be the same as the scalar field connected to the port *Data*.

Ports

Draw Style



This port is inherited from ViewBase. Please refer to the ViewBase documentation.

Colormap



The colormap used to map data values to colors.

Texture wrap



Not used.

Culling mode



This port is inherited from ViewBase. Please refer to the ViewBase documentation.

Orientation



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

Filter Show



This port provides three buttons to filter the bars according to the filter mask:

- *All*: Bars are not filtered,
- *On Data*: All bars with a filter mask different from 0 are displayed,
- *Off Data*: All bars with a filter mask equal to 0 are displayed.

1.22 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.

1.23 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

Color



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

Options



The following two options can be set:

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 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



This port allows you specify 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.

1.24 ClusterFilter

This module filters a connected cluster by an expression build from the coordinates and data values of each cluster point. The resulting cluster will contain only points for which the expression entered results in a true (not equal to 0) value. As an additional input a label field can be connected to the module and is used as a mask. Only points that are part of a label will be copied to the output.

Connections

Data [required]

Connect a cluster data set to this port. A cluster can be created for example from a PSI file.

Mask [optional]

If a label field is connected to this input port only points which are inside the mask will be copied into the result cluster.

Ports**Info**

Info: variables are: *x*, *y*, *z*, and data columns identified as symbols

This port lists the available symbols for the expression.

Expression

Expression: *x<50*

This port needs to be filled with an expression that evaluates to true (1) or false (0). Available variables are the coordinates of each point as *x*, *y*, and *z*. Additionally each data value of the input cluster is available and can be referenced by its symbol. If ClusterDensity is used an expression that filters for a density larger 3 would be: *d > 3*.

1.25 ClusterGrep

This module identifies common points in two different data objects of type *Hx-Cluster* 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

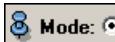
Points



Points:

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

Mode



Mode: inclusion exclusion

This port provides two radio 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



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.

1.26 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 a uniform scalar field.

Resolution



This port provides three text fields 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.

1.27 ClusterStringLabels

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.

1.28 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

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



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



The following 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 specify 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.

Scale plates enables scaling of plates by a data variable. The data variables' symbols are displayed in brackets following the names.

Filter



This port allows you specify 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 symbols 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.

Scale data



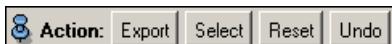
Selects the data variable used for scaling plates. This port is visible if option *plates* and option *scale plates* are enabled.

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. If option *scale plates* is enabled, the size is computed as the data variable selected in *scale data* multiplied by *sphere scale*.

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.

1.29 CollectiveTCL

With this module a Tcl command can be executed on a number of modules. These can either be all visible or selected objects in the object pool or all objects of a certain type. Select *Create/Animation/Demos/CollectiveTCL* from the menu to add a CollectiveTCL module to the pool.

Press the *Apply* button to execute the Tcl command.

Connections

Data [not used]

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

Ports

TCL Command



TCL Command:

`setLineColor 1 0 0`

Specify here the Tcl command to be executed, e.g., "setLineColor 1 0 0" to color all objects of type HxDisplayLineSet.

Note that you can execute several commands on the object(s) if you use the Tcl variable *obj*. Pressing the *Apply* button of Isosurface could be accomplished by "doIt hit; \$obj fire".

Apply to Modules



Apply to Modules:

visible

selected

by type

Specify here whether the Tcl command should be executed on all visible objects, on all selected objects, or on objects of a specific type.

Type

If you choose to execute the Tcl command on objects of a specific type, select the type here. All types available in the Pool are listed here. The list is updated when this option is chosen.

1.30 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.

1.31 Colorwash

The *Colorwash* module helps you to visualize two scalar fields in combination, e.g., 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 a weighted sum between the two scalar fields.

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.

By default, the *Colorwash* module uses the colormap *physics.icol*. This default can be overwritten by defining the global Tcl variable *ColorwashMap*. If the default colormap does not exist, the colormap found in the object pool will be used.

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.col*. 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.

Weighted Sum: 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.

Magic Lens: 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.

Alpha Blending: The background image and the color image are blended depending on the alpha values stored in the colormap.

Luminance Blending: The background image and the color image are blended like in standard alpha blending. However, instead of the alpha values stored in the colormap the luminance of the color values are used. Bright colors are more opaque than dark colors. This mode is useful for non-transparent colormaps.

Label Blending: The colors of the overlaid image are defined by the Materials parameter bundle of the input label field. The background image and the color image are blended using alpha blending. The transparency of the overlaid image can be controlled by the slider in the *Alpha* port. The display of the material *Exterior* can be turned on/off using the *Options* port. Data values, for which no material color has been defined, can be colored using a colormap (see *Options* port below), allowing for non-label data to be overlaid.

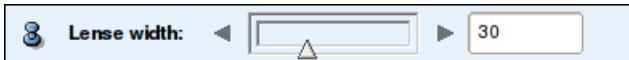
None: The background image is not modified at all. The module is essentially disabled.

WeightFactor



Interpolation factor used in *WeightedSum* mode.

LenseWidth



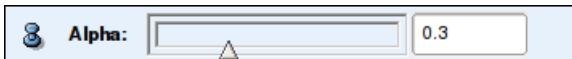
Width of the checkerboard tiles in *MagicLens* mode.

OverlayRange



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

Alpha



Transparency of the overlaid color data. Only available when the fusion method *Label Blending* is selected.

Options



The option *Color Exterior* toggles the colored display of voxels with label *Exterior*. When this material has not been defined, this setting applies to voxels with value 0.

The option *Use colormap when material color not defined* toggles the display of voxels for which no material color has been defined in the parameters. When checked, the current colormap is used for assigning a color to such voxels. This option allows for the colored display of non-label data.

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.

1.32 CombineLandmarks

This module merges an arbitrary number of vertex set objects into a single *landmark set*. The input sets may either be connected to the *Data* and *Source2* port or they can be loaded from disk by specifying file names in *Select LandmarkSet from file port*.

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 a practically unlimited number of data objects to be merged.

Source2 [optional]

See above.

Ports

Select LandmarkSet from file



The files to be combined can be selected from the file system.

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.

1.33 CompareLatticeData

This module takes two regular fields with the same dimensions and the same number of data variables per voxel (i.e. lattice node) as input and computes the average difference per voxel between both. In addition, a new field with the pointwise 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 (including rectilinear, curvilinear), primitive data type, or number of data values per voxel is accepted.

InputB [required]

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

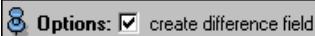
Ports

Average error per voxel



Displays the difference between the two input data sets.

Options



If this toggle is set, a new field with the pointwise difference between both input data sets is created.

1.34 CompassPosition

This module draws the compass widget within the scene. It is used when large snapshots, containing the compass, are needed (using tiling or offScreen). Before taking the snapshot, add this module, select the compass model and the compass position.

Note: When the camera is moved, the compass moves also. To reposition the compass press the Compute button.

Connections

Data [not used]

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

Ports

Compass



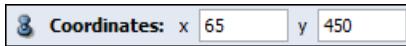
Select the compass model. There are 4 models available: Arrow, AxisCube, Axis and Medical.

Position



Represents the position of the compass on the screen. There are 5 available positions: Upper right, Lower right, Upper left, Lower left and Custom. When Custom position is chosen, the coordinates port is shown and any screen position can be set.

Coordinates



Set a custom compass position. This port is shown only when the Position port is set to Custom.

Scale



This port represents the model scaling.

Actions



By pressing this button, the script calculates the position of the compass in world coordinates and repositions the compass on the screen.

Further links

script object documentation for script programming.

1.35 ComputeContours

This module computes 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 information (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 specific material; this can be done by selecting a material from a menu. By default, contours are computed between all of 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



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.

1.36 ComputeDistanceToPoint

This module creates a copy of the input *SpatialGraph* and adds a new attribute of type float named *DistanceToPoint*. This attribute contains for each point the distance to the specified reference point. If a point cannot be reached from the reference point, its distance will be set to the maximum floating point value.

The module allows the user to pick or set a point, which then will be used to find the closest point on the *SpatialGraph* used as the reference point for the later computation.

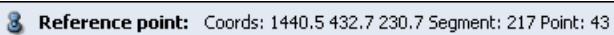
Connections

Data [required]

Connect a *SpatialGraph* to this module.

Ports

Reference point



Shows information about the current specified reference point, which will be used for the computation.

Set reference point



Enter 3D coordinates of an arbitrary point and click *Apply* to find the closest point on the *SpatialGraph* which then will be shown by the *Reference point* port.

Pick reference point



Press the button and pick an arbitrary point using the middle-mouse button in interactive mode while the module is selected. If successful, the coordinates of the picked point will be shown by the *Set reference point* port, while the closest *SpatialGraph* point will be shown by the *Reference point* port.

1.37 ConnectedComponents

This module searches for connected regions in a 3D image volume. In the case of a gray value image the regions are detected based on thresholding, i.e., a connected region is a set of adjacent voxels with intensity values lying inside a user-defined range. In the case of a label image, i.e., an image where the intensity values represent different materials (tissues or regions), the module looks for connected regions within each label and eventually assigns it new labels. The module is useful for quantifying large numbers of small items in a 3D image volume. If the items provide sufficient contrast to the background and do not touch each other, *ConnectedComponents* can be used directly with the grey image volume. Otherwise it is recommended to create a binary image with one of Amira's segmentation tools. See also the documentation of the *Segmentation Editor* and the *LabelVoxel* module for details. If in the binary image multiple items are so close that they form a single connected region, you may want to use module *WatershedSegmentation* and/or *MorphologicalSeparation* to separate the items. The output of *ConnectedComponents* can also be analyzed with respect to shape parameters. See the documentation for *ShapeAnalysis* for details. If the number of connected components is larger than the output data type's data range (e.g., byte fields can represent at most 256 unique labels), the module will issue a warning in the *Console* and assign the same label to multiple components. In this case you may want to change the output data type and repeat the computation. Press the *Apply* button to start the computation.

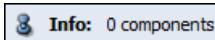
Connections

Data [required]

Image data set to be analyzed. Only byte, short, ushort, and integer fields are supported.

Ports

Info



Once the computation has been performed, this port displays the number of detected regions, the volume of the smallest and the largest region, and the average volume.

Input image

 Input image:	<input checked="" type="radio"/> Gray image	<input type="radio"/> Label image
---	---	-----------------------------------

If *Gray image* is checked, a binary segmentation is performed prior to analyzing connected regions. The segmentation is done according to the intensity range specified in port *Intensity*. If *Label image* is checked, voxel values are considered labels and connectivity is analyzed for each label.

Intensity

 Intensity:	Min <input type="text" value="20"/>	Max <input type="text" value="255"/>
---	-------------------------------------	--------------------------------------

If port *Input image* is set to *Gray image*, voxels with values outside the specified intensity range are considered to be part of the background. This port will be insensitive if in port *Input image* the *Label image* option is selected.

Connectivity

 Connectivity:	Face	
--	------	---

This port defines the connectivity type. In the case of *Face* voxels with a common face are considered connected. In the case of *Edge* voxels with at least one common edge are considered connected, and, in the case of *Corner*, voxels with at least one common vertex are considered connected.

Size

 Size:	Min <input type="text" value="10"/>	Max <input type="text" value="0"/>
--	-------------------------------------	------------------------------------

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

 Output:	<input type="checkbox"/> Label image	<input checked="" type="checkbox"/> Spreadsheet
--	--------------------------------------	---

If *Label image* is checked, a scalar field 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. Depending on the output type (see below) the region labels are not unique if more components are found than are representable with the selected output type. The regions can be visualized in 3D by using e.g. an *Isosurface* module with threshold 0.5.

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

Output type



The output type defines the maximum number of labels in the output field. This also corresponds to the maximum number of objects to be detected (excluding background). These are:

- 255 for *LabelField (8 bit)*
- 65,535 for *Unsigned short (16 bit)*
- 4,294,967,295 for *Unsigned int (32 bit)*

Preserve exterior



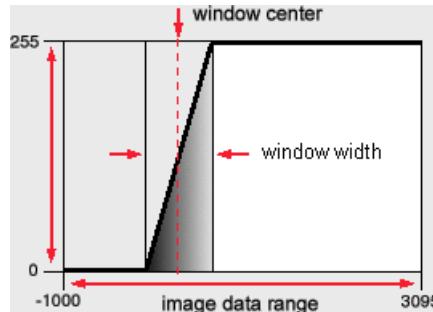
If checked, all voxels labelled as background (= 0) in the input label field will be assigned to background in the output image, independent of whether they are connected or not. Otherwise, if multiple regions are found in the background label, the first region encountered will be labelled as background and all others will be assigned non-background labels. This port will be insensitive if in port *Input image* the *Gray image* option is selected.

1.38 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 3D 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> `

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.

1.39 CornerCut

This module defines a cutting region with the shape of an axis-aligned 3D box originating from a corner of the bounding box. The cutting box can be used to clip the volume displayed by the *Volren* module.

Connections

Data [required]

Connection to a 3D data object, which defines the maximum size of the cutting region. Only the bounding box of the data object, but not the data itself is interpreted by this module.

Ports

Octant



Position of the cutting box according to the volume octant.

X



Relative position of the moving X-edge of the cutting box within the bounding box.

Y



Relative position of the moving Y-edge of the cutting box within the bounding box.

Z



Relative position of the moving Z-edge of the cutting box within the bounding box.

1.40 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. 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 images such as CT and MRI. 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

data set to the lattice of the reference data set. 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 data sets. 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 data sets 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

Input1: min: max: num bins:

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

Input2: min: max: num bins:

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

Gamma correction

Gamma correction:

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

Colormap:

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

Selection by: manual draw subrange

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



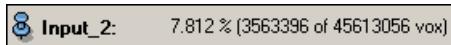
The *Compute Histogram* button recomputes the correlation histogram and pops up the plot window. The *Create LabelField* button computes a *LabelField* using the current selection.

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

 **Input:** 0.082 % (37490 vox-pairs), PSNR: 9.632 dB

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.

1.41 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.

1.42 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

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



Radius:

Radius of the sphere.

Point coords



Point coords: x y z

Center coordinates of the sphere.

Edge length



Edge length:

Desired edge length of the triangles.

Number of points per Unit $\hat{2}$



Number of points per Unit $\hat{2}$:

Number of points per unit square. This port influences the edge length, and vice versa.

1.43 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 (*HxUniformVectorField3*). The vector components must be floats.

1.44 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 equal to 1 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



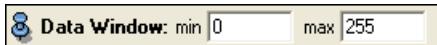
Extra viewer representing the texture.

Mapping type



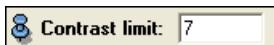
This option menu controls how scalar values are mapped to screen colors. For *linear* mapping, a user-defined data window is mapped linearly to black and white. If *histogram* is selected, an adaptive histogram equalization technique is applied. This method attempts to show all features of the data, 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.

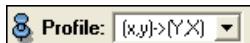
medium: medium resolution. For 3D fields, 256x256. For 2D fields, size-of-field divided by 4.

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

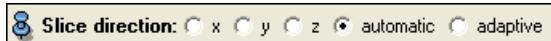


This port is only shown for 2D fields, and specifies the coordinate relation between the profile and the 2D field.

$(x,y) \rightarrow (X,Y)$: 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



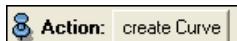
This radio box port specifies the slice orientation.

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.

1.45 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 *Surface View*. If no input is specified the entire scene will be cut with the plane.

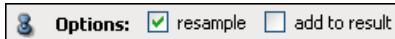
Ports

Orientation



This port provides three buttons for resetting the cutting plane's orientation. *Axial*/xy slices are perpendicular to the z-axis, *coronal*/xz slices are perpendicular to the y-axis, and *sagittal*/yz slices are perpendicular to the x-axis.

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

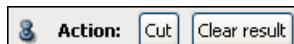
be adjusted using a separate slider (see below). If *add to result* is checked, the resulting lines will be added to an existing result object. Otherwise, for each cutting operation a new result will be created.

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.

1.46 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 the cylindrical slice to be saved 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

Viewer



Displays the viewer. *Data* and *Module* should be selected first.

Mapping Type

	Mapping type:	Linear	
--	----------------------	--------	--

Refer to *ObliqueSlice* for help on this port.

Data Window

	Data Window:	min <input type="text" value="-200"/>	max <input type="text" value="200"/>
--	---------------------	---------------------------------------	--------------------------------------

Refer to *ObliqueSlice* for help on this port.

Contrast Limit

	Contrast limit:	<input type="text" value="7"/>
--	------------------------	--------------------------------

Refer to *ObliqueSlice* for help on this port.

Colormap

	Colormap:	<input type="text" value="0"/>		<input type="text" value="255"/>	
--	------------------	--------------------------------	--	----------------------------------	--

Refer to *ObliqueSlice* for help on this port.

Sampling

	Sampling:	medium	
--	------------------	--------	--

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.

Transparency

	Transparency:	<input checked="" type="radio"/> None	<input type="radio"/> Binary	<input type="radio"/> Alpha
--	----------------------	---------------------------------------	------------------------------	-----------------------------

Refer to *ObliqueSlice* for help on this port.

Dragger

	Show Dragger:	<input checked="" type="checkbox"/>
--	----------------------	-------------------------------------

Switches the dragger on and off.

Cylinder

	Cylinder:	X <input type="text" value="0"/>	Y <input type="text" value="0"/>	Radius <input type="text" value="0"/>
--	------------------	----------------------------------	----------------------------------	---------------------------------------

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

Distance



Distance: 0

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

Zero Angle

Zero angle:

Sets the angle (in degrees) that determines the position of the vertical line along which the cylinder surface is "cut" to be then mapped on the planar 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. To perform this action, the viewer should be visible.

```
setResolution <n> <r0> <r1>
```

Sets the image resolution to $r0 \times r1$, with n determining the sampling: 0 for *coarse*, 1 for *medium*, 2 for *fine* and 3 for *finest*.

```
setResolution <n> <r1>
```

Same as the previous command with $r0 = r1 \times \frac{cylinderPerimeter}{cylinderHeight}$.

```
setViewer <x0> <y0> <x1> <y1>
```

Determines the position and size of the viewer by setting its upper left and lower right corners positions. Positions are computed from the upper left corner of the screen.

```
setZeroAngle <alpha>
```

Refer to *Zero Angle* for help on this command.

1.47 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 control-points 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 (ESC key) and picking and moving the crosshair dragger of the current point (see *Points* port options to show/hide the dragger). You can also pick a location with the middle mouse button on a pickable object in the scene, e.g. an OrthoSlice or a surface, to set the current point to that location.

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 and of the spline respectively.

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 and the maxima respectively of the corresponding axis and to the center of the two axes perpendicular to the probe line.

Options

	Options:	<input checked="" type="checkbox"/> immediate	<input type="checkbox"/> orthogonal	<input checked="" type="checkbox"/> customize sampling	<input type="checkbox"/> follow slice
--	-----------------	---	-------------------------------------	--	---------------------------------------

The *immediate* toggle determines whether the samples are taken while the controlpoints are being moved or when the motion is finished. If the *interpolate* toggle is on interpolation is used to evaluate value of Point Probe. This port is shown only for module *PointProbe*. 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*.

The *customize sampling* option is only available for *SplineProbe*. If it is unchecked, sample points will be equals to those of the tessellated displayed spline. Otherwise, *Control port* will be used to compute the sample points.

The *follow slice* toggle is only available for regular scalar fields. If on, the probe coordinates will be placed on a virtual slice whose position is given by the sliceNumber port value. The slice orientation must have been previously defined with the orientation buttons. The sliceNumber defines a unique x, y or z (depending of the chosen orientation) for all the points of the probe.

Evaluate

	Evaluate:	<input checked="" type="radio"/> magnitude	<input type="radio"/> all	<input type="radio"/>
--	------------------	--	---------------------------	-----------------------

If the probe is connected to a vector field, these radio buttons 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

	Points:	1		0.28	0.394	00028125	options
--	----------------	---	--	------	-------	----------	----------------

Here you can see and/or 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. Furthermore in case of a *SplineProbe* or a *LineProbe*, you have the possibility to see where the line is being drawn by using the "Interactive Feedback" option.

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



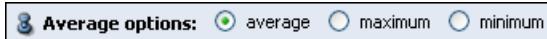
This slider allows you to choose the distance between two consecutive sample points along the probeline.

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* or *SplineProbe* module.

Average options



If averaging using the port *Options* is enabled this port selects the type of sampling. By default the average value is reported but the user might also select the maximum or minimum value.

Radius

A horizontal slider control with a blue trackbar, a white slider handle, and a text input field showing the value "0.5".

Radius:

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

Longitudinal Width

A horizontal slider control with a blue trackbar, a white slider handle, and a text input field showing the value "0".

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

A button labeled "Plot" with a small icon to its left, and a "Show" button to its right.

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.

Offset

A horizontal slider control with a blue trackbar, a white slider handle, and a text input field showing the value "0".

Offset:

This port is used in *LineProbe* and *SplineProbe*. It allows you to add an offset (i.e. translation) to the *plot window* X-Axis coordinates.

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.

```
currentValue
```

Returns all sampled values.

```
getSampledValues
```

Returns all sampled values.

```
setImmediate {0|1}
```

Switches the immediate mode on or off, i.e. data is shown while the probe line or 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 `indexth` point.

```
removePoint <index>
```

Removes the given controlpoint.

1.48 Delaunay2D

This module takes a set of 3D vertices and produces a triangulated surface with 2D topology, using a Delaunay algorithm. In order to do so the vertices are internally 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 *Vertex Set* in chapter *Program Description* of the Amira user's guide. 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 helpful 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 with it, then 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, 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 former version. MeshVizDelaunay (0:default) is a faster implementation.

1.49 DemoDirector

Using the *DemoMaker* module together with the *DemoDirector* GUI, 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. The *DemoDirector*'s user interface shares the space with the console window of Amira. You can switch between *DemoDirector* and the console window by clicking on the respective text in the console area's frame.

Defining and storing a demo sequence

The demo sequence is a set of *events* over a time range represented by the *DemoDirector*'s timeline. A wide range of such events 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.

The start and end time of a demo are depicted by blue lines which restrict the timeline left and right. The "current time" of the demo is visualized with a red line. All three elements can be moved with the mouse.

You can create new events using the *New event* dialog, which is accessible via the *New event...* button in the *DemoDirector* area. After clicking the button a dialog window appears which contains a tree view. The tree view shows all currently loaded modules (except the *DemoMaker* itself) at the first level and the available events for each module at the second and third level. You choose a new event either by selecting it and clicking on the *Ok* button or by double-clicking the event. If necessary the list of the currently available events will be refreshed at invocation of the dialog. This may take a few seconds if the pool contains many modules. You can also trigger refreshing the list manually by clicking the *Update event list* button. Such a manual update is necessary, e.g., after changing the viewer layout in order to get correct viewer mask entries in the event list.

Once you have selected the right element (or a different event type like a pause, break, go-to, or an arbitrary Tcl command) from the *New event* dialog, a new event is created at the current time's position and additional ports will show up in the Properties area, depending on the type of the selected element (e.g., numeric, toggle, button, ...). Using these ports, you can define the value to which the selected GUI element will be set at that time (see *example* below).

Once a part of the desired event sequence is defined, simply store it by choosing *File/Save Network...* from the menu. The demo in its current state will be saved

along with the rest of the Pool. When saving the network, make sure the current time slider is in the desired starting position.

Working with events

You can select an event by clicking on it with the mouse. Selected events are marked with a blue border. All values of the currently selected event are displayed in the Properties area and can be changed there.

If you hover over an event for a short time you will see a tooltip with all the event's relevant information.

You can change the point in time at which an event should be triggered by moving the event with the mouse horizontally within the timeline. Whenever the event is close to another event it jumps to the other one's trigger time and snaps. Snapped events are marked with red lines. Snapping can be disabled by pressing the Shift key while moving events.

The starting time and end time of events depicting a duration can be changed in the same way. The mouse cursor alters its shape in order to indicate whether the whole event would be moved or the starting time or the end time.

Multiple events can be selected with a selection rectangle. The starting point of a selection rectangle is determined by pressing the left mouse button within the timeline (but not in the scale area, see below). The rectangle is then created by moving the mouse while keeping the mouse button pressed. After releasing the mouse button all events inside the rectangle are selected. With the Ctrl key pressed the newly selected events are added to the existing selection. Otherwise previously selected events are deselected. Individual events can be added to or removed from the selection by clicking on them while the Ctrl key is pressed.

The left and right border of interval events can be selected separately. Event borders can be added to a selection or removed from a selection just as described above.

The timeline can be moved by pressing the left mouse button at the scale area and then moving the mouse. Furthermore, the timeline is moved automatically whenever an event reaches the timeline window's left or right border while moving it.

The vertical positioning of events is calculated automatically. Recalculation is done whenever two events overlap each other. In all other cases the arrangement will not change automatically in order to not confuse the user. Rearranging can be triggered by clicking with the right mouse button at the timeline's background.

This can be useful to condense the events in the timeline after major changes or after zooming in. Zooming can be done with the buttons labeled + and -

Deleting events

Selected events can be deleted by either pressing the Del key or using the context menu, which is accessible by pressing the right mouse button on an event.

Playing a demo sequence

Playing a demo sequence can be controlled with the buttons below the timeline:



For playing the demo, simply press the play button (triangle pointing to the right) or press the corresponding function key F 4. 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 F 3 function key immediately stops playing. If the demo stops at a user-defined break, continue it by again clicking the play button or pressing F 4.

If the demo is stopped at the beginning or at some user-defined break, you can jump to the point of the next break, or to the end of the demo by clicking on the play button while the Ctrl key is pressed (or use the function key F 10).

Similarly, clicking on the play backward button while the Ctrl key is pressed (or the function key F 9) jumps to the previous break or to the very beginning of the event sequence. Once you have jumped to the right step of the demo, simply press play or F 4 again to play it from there.

If you press the jump right button (triangle pointing to the right together with a vertical line at its tip) the current time slider jumps to the end of the demo. If you hold down the Ctrl key while clicking the button, the slider jumps to the rightmost event, and with the Shift key pressed the timeline jumps to the right boundary without changing the current time slider's position. The jump left button works correspondingly.

Press the step forward button (triangle pointing to the right together with a vertical line at its base) to step one time step forward.

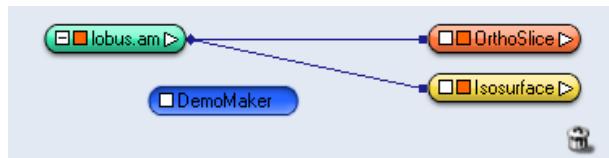
If the event sequence is running too fast or too slow, you can adjust its speed using the *Configure* dialog, accessible through the rightmost button (with the small rectangle). Note that the playback speed can only be adjusted for the complete time range at once. If you want to change the speed of only a part of the sequence, you must do so by adjusting the time range of the corresponding event entries.

Usage Example

Here is a small example to demonstrate the way *DemoMaker* works.

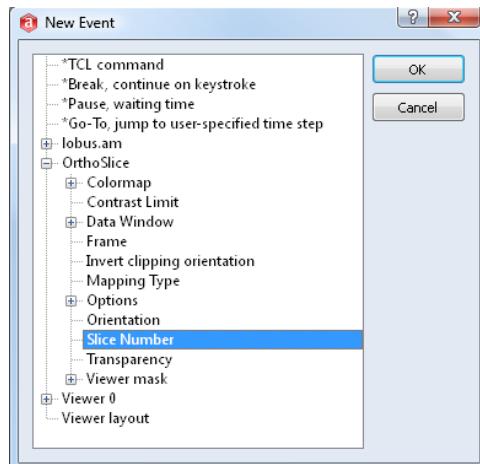
Load `lobus.am` from the `tutorials` subdirectory of the Amira installation directory. Connect an *OrthoSlice* module and an *Isosurface* module. Create a *DemoMaker* by selecting it from the *Create/Animation/Demo* menu.

Now your network should look similar to this:

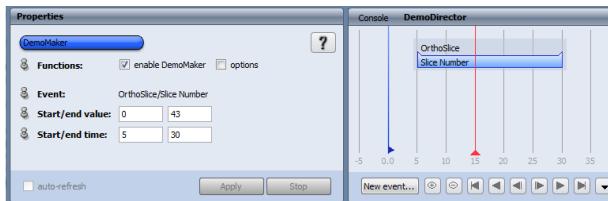


All available user interface ports of the modules in the Pool should now appear in the *New event* dialog (accessible via DemoDirector's *New event...* button). If not, press the *Update event list* button.

Now select *OrthoSlice/Slice Number* in the dialog:



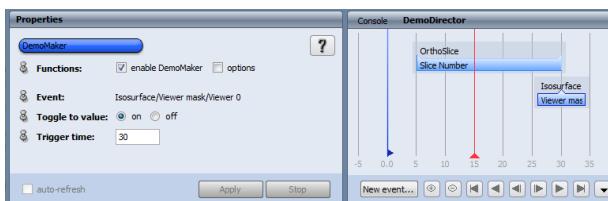
Fields for specifying start/end values and start/end time appear. Enter 0 and 43 as start and end values and move the event with the mouse to position 5 and adjust its end time to 30:



Press the play button to see the demo sequence just specified. You should see the *OrthoSlice* plane moving during the time range from 5 to 30, but nothing happening in the time range from 30 to 100.

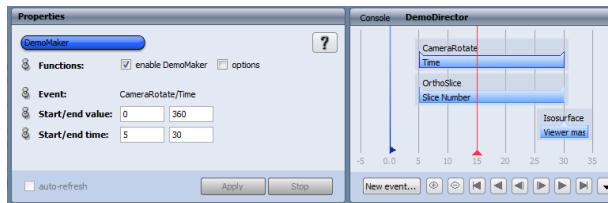
Now select a threshold of 70 in the *Isosurface* module and press *Apply* to make the surface appear in the viewer.

Now select *Isosurface/Viewer mask/Viewer 0* from the *New event* dialog, select *on* as the toggle value and move the event to 30 to determine its trigger time:



Jump back to 0 by clicking on the jump left button. You should see that when jumping back (from later than 30 to earlier than 30) 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 select *CameraPath/Time* or *CameraRotate/Time* in the *New event* dialog. Now the start/end value refers to the value of the camera time slider. Enter the minimum and maximum values of that time slider. Move the event's start and endpoint to 5 and 30:



Now you see that in parallel to the events 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.

Loading existing networks

The way Amira stores DemoMaker objects in network files changed slightly in order to incorporate the DemoDirector. Old network files are converted automatically to the new format, but saved networks cannot be used with older versions of Amira.

The DemoDirector is always connected to a single DemoMaker object. When a network is loaded which contains more than one DemoMaker, the other DemoMakers act as in previous versions of Amira (see *DemoMakerClassic* documentation). But it is reasonable to only use one DemoMaker and merge all events from other DemoMakers.

For this purpose the events have first to be exported from the other DemoMakers. This is done by enabling the *options* toggle in the *Functions* row and then using the *SaveEventList* button in the *Options (advanced)* row (in the properties area). A proper file name can be set before in the row *Event list file*. The created event list can then be imported into the DemoDirector's DemoMaker in the same way. Using the *LoadEventList* replaces all existing events, using *AppendEventList* keeps them.

Ports

Functions

	Functions:	<input checked="" type="checkbox"/> enable DemoMaker	<input checked="" type="checkbox"/> options
--	-------------------	--	---

Activate/deactivate *DemoMaker*, 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.
- *options*: display option ports (see *Options (1) - Options (3)* 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.

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 event 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.

Options (advanced)

 Options (advanced):	<input type="button" value="SaveEventList"/>	<input type="button" value="LoadEventList"/>	<input type="button" value="AppendEventList"/>
---	--	--	--

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

- *SaveEventList*: Saves a list of all events of the DemoDirector in a file given in the *Event list file* port.
- *LoadEventList*: Loads a list of events which are given in a file you can choose in the *Event list file* port. All existing events are replaced.
- *AppendEventList*: As *LoadEventList* this loads a list of events from a given file. But it keeps existing events.

Event list file

 Event list file:	<input type="text" value="3.0/share/script-objects/event_list.txt"/>	<input type="button" value="Browse"/>
--	--	---------------------------------------

This port is only shown when the *options* toggle of the *functions* port is enabled. Here you can choose the file to load or save event lists.

Waiting image

 Waiting image:	<input type="text" value="3/script-objects/img/amira_150_red.png"/>	<input type="button" value="Browse"/>
--	---	---------------------------------------

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.

Additional Ports

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

Defining a break

Selecting **Break, continue on keypress* from the *New event* dialog 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 *New event* dialog 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 events accordingly.

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 left and right boundary slider in the DemoDirector.

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

 Trigger time:	15.0491
---	---------

This defines the point in time at which the go-to is inserted.

Time to jump to

 Time to jump to:	25.8449
--	---------

This defines the point in time which the go-to will jump to.

Defining a toggle

Toggle events can toggle certain GUI elements between the two states *on* and *off*. Examples for toggle values are the different options in a *ToggleList port*, 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

 Trigger time:	15.0491
--	---------

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

Toggle to value

 Toggle to value:	<input checked="" type="radio"/> on	<input type="radio"/> off
--	-------------------------------------	---------------------------

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 events 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



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

Modifier keys



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 event

Select events 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



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



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

Defining a numeric event

Numeric events 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

 Start/end value:	86	0
--	----	---

The two fields of this port define from which start value to which end value the value of the event will be varied.

Start/end time

 Start/end time:	5	30
---	---	----

This defines the time range (on the timeline) 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 *New event* dialog, 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

 Command:	echo "%0% %1% %2%"
--	--------------------

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)

 Start value(s):	000
---	-----

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)

 End value(s):	1 1 1
---	-------

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

 Start/end time:	5	30
---	---	----

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

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 called 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 timeline. 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 our hotline as a bug or feature request.

- The **Load network* event type for loading a different network file is currently not implemented.
- For *button event*, "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 *New event* dialog of the DemoDirector.
- 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*.
- You cannot control *DemoMaker/DemoDirector* with a *GlobalTime* object (in order to synchronize several objects). As a workaround you can control the time ports of these objects by adding them to the *DemoDirector*.

1.50 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.

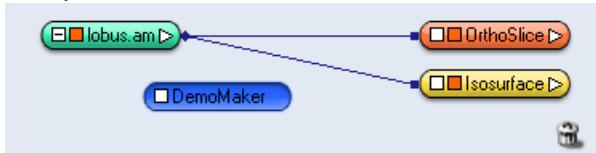
If the demo is stopped at the beginning or at some user-defined break, you can press the function key F10 to jump to the point of the next break, or to the end of the demo. Similarly, pressing F9 jumps to the previous break or to the very beginning of the action sequence. Once you have jumped to the 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:



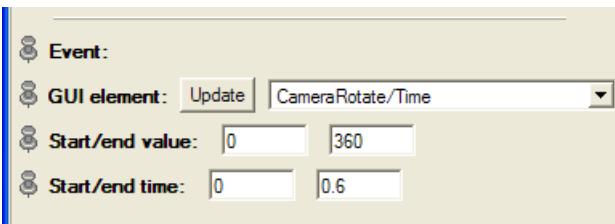
Now press the *Add* button to add this action event to the event list. Press the time slider's play button to see the demo sequence just specified. You should see the *OrthoSlice* plane moving during the time range from 0 to 0.4, but nothing happening in the time range from 0.4 to 1.0.

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.

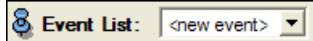
Ports

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)



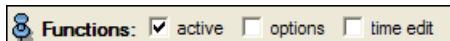
This menu contains the list of currently defined demo actions that make up the demo sequence. The buttons below this menu always refer to the currently selected menu entry.

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.

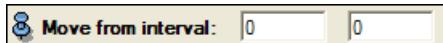
- *explicit redraw*: when toggled on, for each time step the active viewers are explicitly called to perform a redraw. If toggled off, Amira's *auto redraw* feature is used. If in some cases the desired action sequence is not properly displayed in the viewer, try toggling *explicit redraw* on.
- *debug*: 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



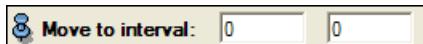
If the *wait screen* option is enabled, this port is used to specify the file name of the waiting screen image. The image will be displayed during jumps on the time slider.

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



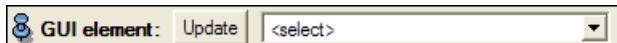
This port is only shown when the *time edit* toggle of the *Functions* port is enabled. Specify the target time interval for relocation on the time line (see *Moving events on the time line* below).

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 data set or connecting a new module), press *Update* to update the contents of the selection menu.

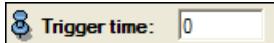
Additional Ports

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

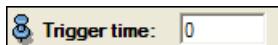


This defines the point in time which the go-to will jump to. For example, to create a loop between time 0.2 and 0.4, insert a go-to with *Trigger time* 0.4 and *Time to jump to* 0.2.

Defining a toggle

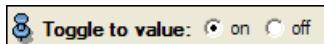
Toggle actions can toggle certain GUI elements between the two states *on* and *off*. Examples for toggle values are the different options in a *ToggleList port*, or the orange *viewer mask* toggle(s) present in every data object or display module (to switch the display of the module in the viewer on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

Trigger time



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

Toggle to value

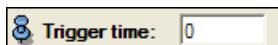


This defines the value (on or off) to which the selected GUI element is set at the defined trigger time, if the demo sequence is played forwards. When playing or jumping backwards, the inverse value is used.

Defining a button press

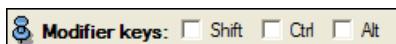
Button actions emulate the pressing of a button or executing certain events with no parameters. Examples for buttons elements are the *ButtonList port*, or inverting the orientation of a clipping plane.

Trigger time



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

Modifier keys



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

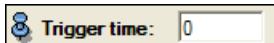
Select actions are used to set a port's value to one of a set of choices as offered by a *selection menu* or a *radio box*.

Change from/to value



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

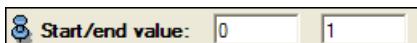


This defines the point in time (on the time slider) at which the port is set to a new selection value.

Defining a numeric action

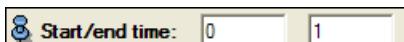
Numeric actions are used to vary a port's numerical value from a start value to an end value over time. Examples for numeric ports are *numerical sliders* or *numerical text fields*.

Start/end value



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



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

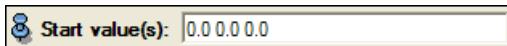
When selecting **Tcl command* from the GUI element menu, you can insert arbitrary Tcl commands into the demo sequence. Furthermore, like for *numeric events*, the command can be parameterized by one or more numeric values that will be varied between user-defined start and end values.

Tcl command



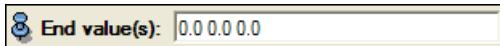
Text field for typing the desired Tcl command. Within the command, special placeholders $\%0\%$, $\%1\%$, $\%2\%$, ... can be used that will be replaced by numeric values. The start and end values for these placeholders are defined in the ports described below.

Start value(s)



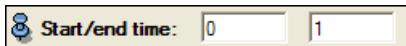
If the numeric placeholder $\%0\%$ is used in the *Command* field, specify the start value for this placeholder. If multiple placeholders are used (e.g. $\%0\%$ and $\%1\%$), specify space-separated start values for all of the employed placeholders.

End value(s)



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



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 called 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 our 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.

1.51 DemoSequence

DemoSequence is a script object for steering Amira demos. Demos in Amira should be prepared using the *Demo GUI* or the *command-line tool*.

Connections

Data [not used]

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

Demo status display [optional]

Here you may add a module of type *Annotation* which displays the status of the current demo using special characters:

| < First demo or demo step reached.

> | Last demo or demo step reached.

X Demo is executing.

. Idle, i.e., it is possible to jump to another step or demo.

L Loading a new demo.

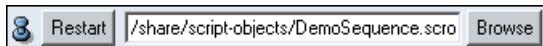
E An error occurred during execution the current step.

The *Annotation* module should display its status in the lower right corner of the viewer (x,y: -20, 20).

If a *DemoSequence* module is created, an annotation module named *DemoStatusDisplay* is automatically created and connected to the *DemoSequence* module.

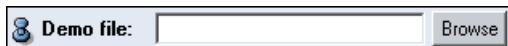
Ports

Script [required]



Mandatory port of a script object. Please do not press the *Restart* button!

Demo file



Filename of a file describing a sequence of predefined demos. This port is provided for compatibility with older demos that do not use the Demo GUI. It is recommended to use the Demo GUI instead of using demo files.

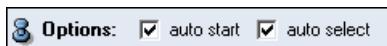
Note: If *DemoSequence* is used with the *Demo GUI*, this port is empty and not used.

Demos



Menu containing the list of all demos. You may select one of them and then press the *Start* button.

Options



auto start: If *DemoSequence* is used with a *demo file*, the first demo will start automatically if this option is on.

auto select: The *DemoSequence* will be selected after executing a new demo or step if this toggle is on.

Demo



Displays the name of the current demo and its number out of the total number of selected demos.

Demo Navi



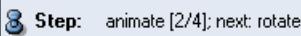
Steps one demo backward (*<< Demo*) or forward (*Demo >>*) or reloads the current demo. It is also possible to switch the demos by using the following function keys:

Ctrl-F2: Go back to the previous demo.

Ctrl-F3: Reload the current demo again.

Ctrl-F4: Go to the next demo.

Step



Displays the name of the current step and its number out of the total number of possible steps.

Step Navi



Steps one step of the current demo backward (*<< Step*) or forward (*Step >>*) or executes the current step again (*<Repeat>*).

The *Jump* buttons allow stepping through single steps without executing them. They are activated if the current demo defines jump commands.

It is also possible to switch the demo steps by using the following function keys:

Ctrl-F6: Go back to the previous step.

Ctrl-F7: Execute the current step again.

Ctrl-F8: Go to the next step.

Status:



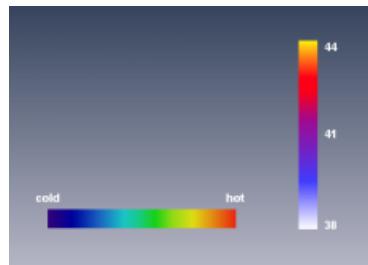
Shows the status of the current demo.

1.52 Digital Image Filters

Encloses the *ImageEditor* in a compute module.

1.53 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. For better results, use a 1x1 tiling for the snapshot.



Connections

Data [required]

The colormap to be displayed.

Histogram [optional]

The histogram to be displayed.

Ports

Options

	Options:	<input type="checkbox"/> custom text	<input checked="" type="checkbox"/> vertical	<input checked="" type="checkbox"/> rel. size	<input checked="" type="checkbox"/> 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 [-80] y [40]

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 [200] thickness [0.1]

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: [-8/cold 50/hot]

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".

Title

Title: Temperature

This port holds the title of the colormap. By default, this port contains an empty string.

Histogram Options

Histogram options: Overlay ▾

Defines the position of the histogram.

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.

1.54 DisplayDate

This module displays the value of a time object which can provide time as a calendar date. This is useful for animations which shall contain an illustration of date.

Connections

Data [required]

Object which provides current date.

Ports

Position



Position of the date 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.

Colors



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

Options



This port adjusts additional settings influencing the drawing.

- **frame** Draw a frame around the date icon.
- **solid** Fill the icon's background.

Point size



This port adjusts the size of the text being drawn.

Display



Select the parts of the date value (date, time) to be displayed.

Date format



Choose the format for the date part. When a custom format is specified, these expressions may be used:

- **d** the day as number without a leading zero (1-31)
- **hdd** the day as number with a leading zero (01-31)
- **hddd** the abbreviated localized day name (e.g. 'Mon'..'Sun'). Uses QDate::shortDayName().
- **hddd** the long localized day name (e.g. 'Monday'..'Sunday'). Uses QDate::longDayName().
- **hM** the month as number without a leading zero (1-12)
- **hMM** the month as number with a leading zero (01-12)
- **hMMM** the abbreviated localized month name (e.g. 'Jan'..'Dec'). Uses QDate::shortMonthName().
- **hMMMM** the long localized month name (e.g. 'January'..'December'). Uses QDate::longMonthName().
- **hy** the year as two digit number (00-99)
- **hyyy** the year as four digit number (1752-8000)

Time format



Choose the format for the time part. When a custom format is specified, these expressions may be used:

- **h** the hour without a leading zero (0..23 or 1..12 if AM/PM display)
- **hh** the hour with a leading zero (00..23 or 01..12 if AM/PM display)
- **m** the minute without a leading zero (0..59)
- **mm** the minute with a leading zero (00..59)
- **s** the second without a leading zero (0..59)
- **ss** the second with a leading zero (00..59)
- **z** the milliseconds without leading zeroes (0..999)
- **zzz** the milliseconds with leading zeroes (000..999)
- **AP** use AM/PM display. AP will be replaced by either "AM" or "PM".
- **ap** use am/pm display. ap will be replaced by either "am" or "pm".

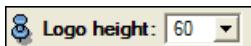
1.55 DisplayLogos

This module displays the product logo (Amira) as well as the FEI Visualization Sciences Group 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.

1.56 DisplayTime

This module displays the value of a time object in the 3D viewer using a textual or iconic representation. This is useful for 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. If no *Time* object is connected, an arbitrary value which can be set using the commands of the port *value* can be shown.

Connections

Time [optional]

The time object to be displayed.

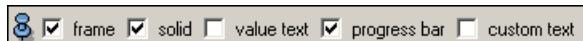
Ports

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:** x y

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

 **Size:** length thickness

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

 **Colors:** text foreground background frame

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

Custom Text

 **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

 **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.

1.57 ExtractSurface

This module allows you to extract surfaces displayed by one or more modules like *SurfaceView*, *GridVolume (Tetrahedra)*, or *GridVolume (Hexahedra)* (in fact, any module derived from *ViewBase* will work). In order to extract a surface, 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. All visible triangles will be extracted when you press the *Apply* button.

The resulting surface can be used for further computations, for example, as a *Seed-Surface*.

Connections

Module1 [required]

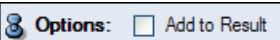
Connection to a display module like *SurfaceView*, *GridVolume (Tetrahedra)*, or *GridVolume (Hexahedra)*.

Module2 ... ModuleN [optional]

Connections to additional display modules.

Ports

Options



Allows you to specify whether the extracted triangles will be added to an already existing result. If not checked, the result will always be cleared before

new triangles are added. If this port is checked, the result cannot be restored on-the-fly when loading the network. Instead, the result needs to be saved.

Result Actions



Allows you to clear the result.

Commands

`getNumInputs`

Returns the total number of used and unused input connections.

`setNumInputs num`

Sets the total number of input connections.

1.58 FilteredObliqueSlice

The *FilteredObliqueSlice* extends the *ObliqueSlice* module by adding filtering capabilities on visualized data.

See the documentation of the base class for details about how to adjust position and orientation of a slice and also choose the best data mapping method. User can defined the number of filter to apply. Default filters are the ones proposed by the *ImageEditor* editor or *ImageFilters* compute module. The ports of chosen filters are shown dynamically according the filters order. The number of filters is currently limited for performance reasons.

Ports

Brightness



Controls the brightness of the slice. By default the brightness is set to zero; the slice is displayed as is. The brightness value is in the range [0,1].

Contrast



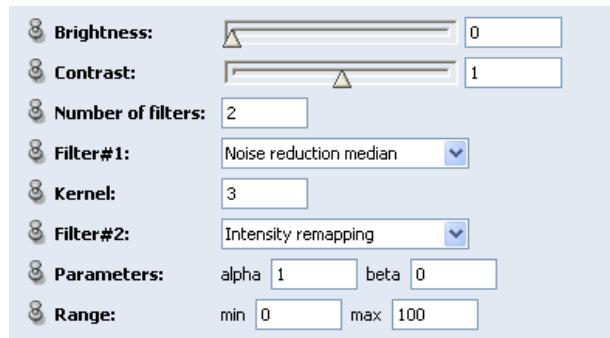


Figure 1.5: Example of filter combination on an oblique slice.

Controls the contrast of the slice. By default the contrast is set to one; the slice is displayed as is. The contrast value is in the range [0,2].

Number of filters

Number of filters:	5
---------------------------	---

This port sets the number of filters applied to the slice. The first filter to be applied is filter number 1. Filters are applied in the order they appear in the GUI. You can apply up to 5 filters.

Filter#1

Filter#1:	None	
------------------	------	--

This port allows to choose the type of filter of rank 1.

Filter#2

Filter#2:	None	
------------------	------	--

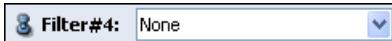
This port allows to choose the type of filter of rank 2.

Filter#3

Filter#3:	None	
------------------	------	--

This port allows to choose the type of filter of rank 3.

Filter#4



This port allows to choose the type of filter of rank 4.

Filter#5



This port allows to choose the type of filter of rank 5.

1.59 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.

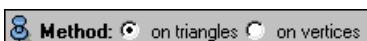
Connections

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

 Parameters: nLayers nAverage

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

 Output: 

This menu controls the output of the curvature module.

- *Max curvature*: a surface scalar field is generated containing the maximal principal curvature of a triangle or of a point.
- *1/Max curvature*: 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.
- *Mean curvature* is similar to *Max curvature*. 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.
- *1/Mean curvature*: the mean curvature values are inverted.
- *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).
- *1/Gauss curvature*: Gauss curvature values are inverted.

- *Both curvature values* returns the two principal curvature values as a surface complex scalar field that should be interpreted as a field of pairs of scalar values (use *Arithmetic* to retrieve each value).
- *Both 1/curvature values* returns the inverted two principal curvature values.
- *Max curvature vector*: 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.
- *Both curvature vector*: a surface complex vector field (that should be interpreted as a field of pairs of vectors) is computed indicating the direction of the two principal curvatures. The length of a directional vector is equal to the corresponding curvature value.
- *Shape index* computes the surface scalar field which values are equal to

$$\frac{2}{\pi} \operatorname{atan} \frac{C_1 + C_2}{C_1 - C_2}$$

where C_1 and C_2 are the two principal curvatures.

- *Curvedness*: computes the surface scalar field which values are equal to

$$\frac{1}{2} \sqrt{C_1^2 + C_2^2}$$

where C_1 and C_2 are the two principal curvatures.

1.60 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.

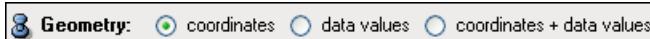
Ports

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* or using both nodes and data values. 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.

1.61 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 (Tetrahedra)*, or *GridVolume (Hexahedra)* (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 (Tetrahedra)*, or *HexaView (Hexahedra)*.

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.

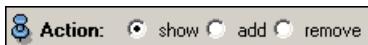
Ports

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



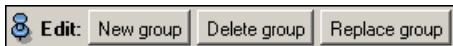
This radio box defines what happens when a new group is chosen in the *Groups* menu. If *show* is selected, the elements of the new group become visible and all other elements become invisible. If *add* is selected the elements of the new group are added to the view. If *remove* is selected the elements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

1.62 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 $0.3*R+0.59*G+0.11*B$.

ColorField [optional]

Optional scalar field used for pseudo-coloring. Only RGBA color fields or scalar fields can be connected. If an RGBA color field is connected, the colormap is not used and the vertices of the height field are colored according to the RGBA values. If a scalar field is connected, the vertices are colored according the scalar values in the colormap.

Colormap [optional]

The colormap used to map data values to colors.

ROI [optional]

Connection to a module defining a region of interest (not supported).

Ports

DrawStyle



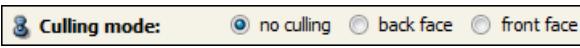
This port is inherited from *ViewBase*. For a description see there.

Colormap



The colormap used to map data values to colors.

Culling mode



Please refer to the *ViewBase* documentation.

Orientation



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

1.63 Histogram

This module computes the histogram of the data values of a scalar field or lineset 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. Furthermore, you may connect linesets and spread sheets to this port.

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

 Overall :	mean=60.5897 deviation=42.326 rms=73.9094
 In range:	mean=60.5897 deviation=42.326 rms=73.9094

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

 Range:	min <input type="text" value="21"/>	max <input type="text" value="243"/>	<input type="button" value="Reset"/>
---	-------------------------------------	--------------------------------------	--------------------------------------

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



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.

Histogram options



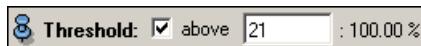
- *absolute* the absolute frequencies are displayed.
- *relative* the relative frequencies i.e. the absolute frequencies divided by the number of events (voxels, tetrahedra,...) are shown.
- *in %* displays relative frequencies as a percentage of events.
- *+ cumulative* displays additionally the cumulative frequencies as a line graph.

Plot options



If the *line drawing* toggle is set, the histogram is plotted as a line instead of a histogram. With the *logarithmic* toggle set, the plot is shown with logarithmic scaling [Default].

Threshold



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



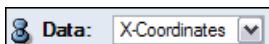
The tindex value defines a reversed threshold value, i.e., a tindex of 90 returns the value where 90% of the data values lie above that value. It is denoted by the reddish markerline in the plot window. The computation and display of the index value can be toggled on or off. This port is only shown when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

Material



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.

Data



If a lineset is used as input, then this port provides the opportunity to select the data channel for which the histogram is to be computed. You may also compute the histogram for the coordinates of the lineset.

Commands

`showCumulative [{0|1}]`

Switches the display of a cumulative plot on (1 (Default)) or off (0).

`showRightCumulative [{0|1}]`

Switches the display of a right cumulative plot on (1 (Default)) or off (0).

`showDifferential [{0|1}]`

Switches the display of a differential plot on (1 (Default)) or off (0). If there is a differential plot to be shown the axis scaling is switched to linear, since there might be negative values.

1.64 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 look up 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. The mode *DEC* is used to color the lines based on the direction of each segment. Directionally encoded colors are used to illustrate the orientation in dense fiber bundles by mapping the spatial x-direction to red, the spatial y-direction to green, and the z-direction to blue.

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.

1.65 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



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

Intermediate slices



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



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



Cubic interpolation will in general generate smoother shapes.

1.66 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.

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 reset. Afterwards triangles adjacent to a particular material can be added again by choosing that material from the port's option menu. With the *Create LineSet* button the line segments drawn by this module can be exported into a *line set* 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.

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> `

Sets the color of the intersection lines.

1.67 Isolines

This module computes isolines and their annotations 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

Values: 80 196

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

Parameters: resolution 128 line width 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

Options: update min-max resample annotate

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

Colormap: 0 1

Port to select a colormap.

Annotations path

Annotations path: Tangential ▾

Indicates how annotations are drawn on isolines. If choice is *tangential path* then annotations are drawn according the tangent line of isolines. *Camera aligned* means that annotations are drawn always facing the camera.

Font

Font: Helvetica (12 pt.) Select

Select the font and color of annotations of contour lines. The font size is a multiplicative factor of the domain size. The font size impacts annotations position

and number of annotations. As much as possible, the annotations are placed to avoid crossing mesh lines or mesh contours.

Text scale



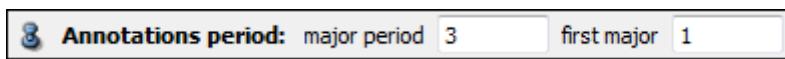
Font size factor of annotations of contour lines. It's a multiplicative factor of font size. Useful when using projection, a heuristic is applied to keep a ratio font size / mesh size. Sometimes to obtain a better effect the default factor must be customized.

Text spacing



Curvilinear distance between two annotations of a same contour line. It is a multiplicative factor of the domain size.

Annotate every



The *major period* defines the major and the minor contour lines. *majorPeriod* is the period of major contour lines. Only major contour lines can be annotated. The *first major* defines the first major contour lines.

1.68 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**

	Spacing:	<input checked="" type="radio"/> uniform	<input type="radio"/> explicit	<input type="radio"/> offsets
--	-----------------	--	--------------------------------	-------------------------------

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. In *offsets* mode an iso value can be selected using a slider. For each given offset that is added to that iso value, a isoline is depicted.

Values

	Values:	50	100	num 3
--	----------------	----	-----	-------

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

	Values:	50 75 100
--	----------------	-----------

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.

Isovalue

	Isovalue:	<input type="text"/> ▲	75
--	------------------	------------------------	----

In *offsets* mode this value is used as the base for each isoline. An isoline is displayed for each offset in the Offsets port, adding each offset to the iso value chosen here. This mode is particularly suited for animation using *Animate*.

Offsets

	Offsets:	-25 0 25
--	-----------------	----------

In *offsets* mode an arbitrary number of blank-separated threshold values can be defined in this text field. Each value results in an isoline. The isovalue is the sum of this value and that chosen in the slider Isovalue.

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.

1.69 Isosurface (Regular)

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

Very high-resolution data sets 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.

Texture [optional]

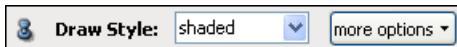
The texture field must be an HxUniformColorField3 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

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*.

ROI [optional]

Connection to a module defining a region of interest.

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

To use this port, you must first create a dragger. To do so, type the command `Isosurface showBox`. Then the buffer port must be enabled explicitly with the command `Isosurface buffer show`. For a detailed description see the *ViewBase* documentation.

Threshold



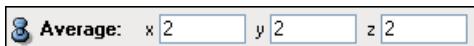
Determines the value used for isosurface computation. The slider is automatically adjusted to cover the whole range of data values. For CT data, use a value of -200 to extract the skin surface or a value of about 150 to extract the bone structures.

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.

TextureWrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

1.70 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.

1.71 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 *Tolerance distance* factor.

Triangles of similar color are grouped into patches if you select the *create patches by color* option. Colors are considered similar if they differ by less than the *Tolerance color* value in RGB space.

Connections

Data [required]

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

Ports

Tolerance

	Tolerance:	distance	1e-05	color	0.1
--	-------------------	----------	-------	-------	-----

Tolerance distance controls duplicated points removal.

Tolerance color controls which colors are considered similar.

Options

	Options:	<input type="checkbox"/> create connectivity	<input type="checkbox"/> create color field	<input type="checkbox"/> create patches by color
--	-----------------	--	---	--

If the *create 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.

If the *create patches by color* option is set, triangles of similar color are grouped into patches.

1.72 LabelSpatialGraph

This module evaluates a *Label Field* at *Spatial Graph Node* and *Point* locations. The result will be a copy of the input *Spatial Graph*, where for each chosen location an additional *label group* will be created, containing a the *Label Field*'s materials. The default material (Exterior) will be assigned for *Nodes* and *Points* respectively, which do not intersect the *Label Field*.

Connections

Data [required]

The SpatialGraph input data object.

LabelData [required]

The Label Data input object.

Ports

Location

Available locations to be evaluated.

1.73 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

A dialog box with a title bar containing a small icon and the text "Regions: Exterior Material1 Material2 Inside". Below the title bar is a scrollable list area.

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

A dialog box with a title bar containing a small icon and the text "Exterior-Material1: 50". Below the title bar is a scrollable list area.

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

Fat-Muscle

A dialog box with a title bar containing a small icon and the text "Material1-Material2: 200". Below the title bar is a scrollable list area.

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

Muscle-Bone

A dialog box with a title bar containing a small icon and the text "Material2-Inside: 255". Below the title bar is a scrollable list area.

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

Options

A dialog box with a title bar containing a small icon and the text "Options:". Below the title bar are three checkboxes labeled "subvoxel accuracy", "extend exteriors", and "remove bubbles".

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, see 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 *extend exteriors* 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.

1.74 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.

1.75 LandmarkView

This module displays a *landmark set* as small spheres, or in case of specialized 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



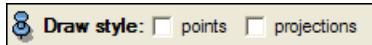
You may select whether only the first, only the second, or all point sets are shown.

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.

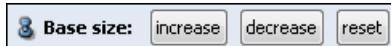
In the second box, the projected positions are displayed as circles on the bounding box boundary. The box is either taken from the connection *Box* or computed from the landmarks.

Size



Size of displayed spheres or points.

BaseSize



Base size of the spheres or points which the size slider relates to. The base size can be increased, decreased, and reset (which tries to create a meaningful size based on the currently loaded data).

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).

```
getColor <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.

1.76 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

$$f(p1, p2) = \exp(-d(p1, p2) * \text{beta}).$$

Norm



The norm to measure the distance in the *flow* algorithm. Either the L1 norm $d(p1, p2) = |x1 - x2| + |y1 - y2| + |z1 - z2|$ is used or the L2 norm $d(p1, p2) = \sqrt{|x1 - x2|^2 + |y1 - y2|^2 + |z1 - z2|^2}$.

1.77 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]

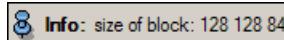
A data object providing a *Lattice* interface.

Time [optional]

Optional connection to a time object, only useful if the input object provides time-dependent data. 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

Info



Info: size of block: 128 128 84

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



The lattice index of the lower corner of the subblock.

Box size



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.

Box max



Convenience buttons to use the maximum box size for the width, height and depth, respectively. Also enables the subsampling open to quickly load a coarse version of the whole volume.

Options



Hides the dragger and/or enables subsampling.

Subsample



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

Timestep



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

1.78 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.

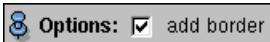
Connections

Data [required]

Connection to input label field.

Ports

Options



Same behavior as the corresponding option in the SurfaceGen module.

1.79 LineProbe

See Section *Data Probing* for details.

1.80 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 by at least one curve in the plot window. For higher dimensional fields, e.g., vector fields, the value shown depends on the setting 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

Evaluate: magnitude all normal+tangent

A group of three radio buttons labeled "Evaluate". The first button, "magnitude", is checked. The other two buttons are "all" and "normal+tangent".

If the probe is connected to a vector field, these radio buttons 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

Options: take average

A group of checkboxes labeled "Options". The "take average" checkbox is checked.

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

Radius: 3.36926

A slider control with a numerical value of 3.36926.

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

Longitudinal Width

Longitudinal Width: 2.62411

A slider control with a numerical value of 2.62411.

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

Samples

Samples:

A group of two buttons labeled "Samples". The "Plot" button is highlighted.

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.

1.81 LineSetToSpatialGraph

This module converts a *LineSet* object into an object of type *SpatialGraph*. The resulting *SpatialGraph* object will have the same geometry as the *LineSet* with all *lines* mapped to *segments (edges)*. An existing data column *Data 0* will be mapped to point attribute *thickness* of the *SpatialGraph* object.

Connections

Data [required]

The *LineSet* data object to be converted.

1.82 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 wire-frame 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 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 all lines to be rotated 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 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 all parts of the line to be twisted the same amount. Following, there will be one entry for each available data variable. Selecting such an entry influences the strength of the twist.

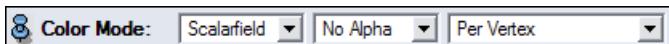
Selecting *Accumulate Twist Values* in the second menu causes the selected twist variable to be accumulated along the line before usage.

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 look up 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



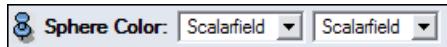
By default *No Spheres* is selected. Changing this selection causes a sphere to be displayed at each line vertex. *Constant* will display the spheres with a constant radius. Following, there will be one entry for each available data variable. Selecting such an entry causes the sphere radii to be scaled 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 an 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 an 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.

1.83 MaterialStatistics

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 such as the original image data is connected to the *Field* input port of the module, then five more columns will be generated, presenting results from a statistical analysis of the image data that corresponds to the segmented materials.

- **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, e.g., the image data associated with the label field. The data is used to compute additional statistics. If the scalar field has the same dimensions as the label field, the voxels are accessed directly. Otherwise, data values are interpolated using the native method of the field.

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.

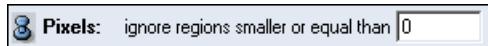
For example, you may have two separate objects both assigned to the same material. In the *Material* mode, the two objects will be interpreted as one object, in the *Region* mode as two objects.

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



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



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



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.

1.84 Measurement

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 length, 2D or 3D angle, and 2D or 3D annotation). To change the position of a selected tool, click on one of its red handles and drag it to a new location. Some tools also provide text entries that allow changing their position.

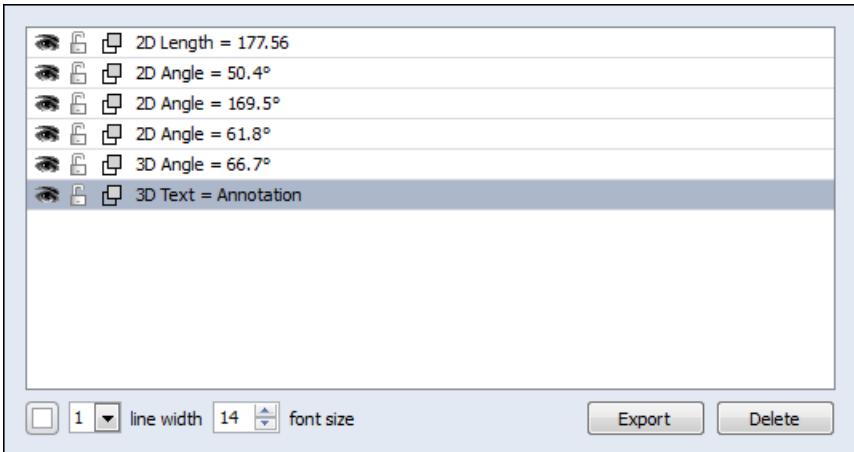
Once the *Measurement* tool is activated, the mouse pointer changes its shape. While the tool is activated, every measurement you perform will be automatically added to the list. For example, you can activate the 2D Length tool to measure a distance, measure a second one, then activate the 2D Angle tool and add an angle measurement. The three measurement objects will be added to the list one by one automatically.

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 *Measurement* object which was created in the Pool.

Note: 2D distance measurements require orthographic view and the measurement is in pixels.

Ports

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 *Export* button you can export all containing Measurement tools information to a *SpreadSheet*.

With the *Delete* button you can delete the currently selected tool(s).

Commands

`tool init`

Erase all points and settings of a tool.

`tool getValue <id>`

Return the value of the tool with the specified id. If the tool is an Angle or a Distance, a float will be returned. If the tool is an Annotation, the annotation text will be returned. The same command can be called as: `tool<id> getValue`.

`tools getPoints <id>`

Return the points of the tool with the specified id. If the tool is a 2D Angle or Distance, 2 points are returned as a TCL list. If the tool is a 3D Angle or Distance, 3 points are returned as a TCL list. If the tool is an Annotation 1 point is returned. The coordinates are normalized between 0 and 1 for the 2D points. The same command can be called as: `tool<id> getPoints`.

`tools getName <id>`

Return the name of the tool with the specified id. The same command can be called as: `tool<id> getName`.

`tools getNumberTools`

Returns the number of elements in the tool form.

`tools exportSpreadSheet`

Export all containing Measurement tools information to a *SpreadSheet*.

`tools dump`

Returns a TCL list containing all the Measurement tools information.

`tool<id> point <p1>[<p2>...]`

Set the points for the specified tool.

`tool<id> format "<frm>"`

Set the format for the specified tool.

```
tool<id> color <red> <green> <blue>
```

Set the color for the specified tool.

```
tool<id> pointSize <size>
```

Set the pointSize for the specified tool.

```
tool<id> name <newName>
```

Set the name for the specified tool.

```
addLine 2D
```

Create a 2D line as a tool object. By using the previous TCL commands, the line properties can be set.

```
addLine 3D
```

Create a 3D line as a tool object. By using the previous TCL commands, the line properties can be set.

```
addAngle 2D
```

Create a 2D angle as a tool object. By using the previous TCL commands, the angle properties can be set.

```
addAngle 3D
```

Create a 3D angle as a tool object. By using the previous TCL commands, the angle properties can be set.

```
addAnnotation 2D
```

Create a 2D annotation as a tool object. By using the previous TCL commands, the annotation properties can be set.

```
addAnnotation 3D
```

Create a 3D annotation as a tool object. By using the previous TCL commands, the annotation properties can be set.

1.85 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]

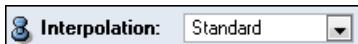
The first data field to be merged.

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.

Padding Value



Padding value is used for monochrome images to define the value used to fill in undefined pixels.

Padding Color



Padding value is used for RGBA images to define the value used to fill in undefined pixels.

1.86 MovieMaker

This module can be used to create an MPEG 1 movie or an animation consisting of a series of 2D images in JPEG, TIFF, PNG or RGB 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. Note: The movie is always generated using the contents of viewer 0. No other viewer can be used for generating a movie.

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

Viewer



Select the viewer to record the movie from and the antialiasing option to enhance the resulting image quality.

File format



The format of the resulting movie or image file(s). In order to write a series of JPEG, TIFF, PNG or RGB files, select the corresponding image types. To create a MPEG movie, select *MPEG movie*. On Microsoft Windows it is also possible to select *AVI movie* as file format. The selection also has an influence on which ports are shown/not shown in the Properties area.

Filename



The name of the resulting movie or image file(s). A file dialog pops up when the *Browse* button is pressed. For image sequences the filename should contain a series of hash marks #####. The hash marks will be replaced by the actual frame number. The output format is bound to the previously selected movie format. If no filename is specified, the file name will be asked on apply. If no file name was given or on pressing cancel, the movie/images will not 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.

AVI encoder



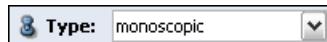
This port is only shown when selecting *AVI movie* as desired file format. You can select the codec in which the AVI movie should be compressed with. The *Options* button shows the codec's property page and is only active if the codec provides such.

Quality



This port is displayed 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



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



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



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.

Commands

```
filename <filename>
```

Set the file name for the movie that will be generated by the MovieMaker

```
action setValue 0
```

This command instructs the MovieMaker module that everything is ready for movie to be generated. Then a subsequent "fire" command is needed to start the movie making process

```
fire
```

Issues an event notice to MovieMaker module. This command will force

MovieMaker to update all its ports. When action setValue 0 is executed before hand then the module will start the movie making process.

1.87 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 packed together in one or more movie data files (*moviename.amovstream*). To create movies using Amira, use the *MovieMaker*.

1.87.1 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.

1.87.2 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.

1.87.3 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-save. The BMP-reader is known to be not thread-save.

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.

1.88 NormalizeImage

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.

Press the `Apply` button to start the computation.

Connections

Data [required]

The data set the correction is performed on.

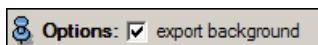
Ports

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.

1.89 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 scanner 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 two different methods are supported to map scalar values to screen colors, namely monochrome mapping based on adaptive histogram equalization, and pseudo-coloring based on a colormap.

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 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/xy* slices are perpendicular to the z-axis, *coronal/xz* slices are perpendicular to the y-axis, and *sagittal/yz* 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.

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the

shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

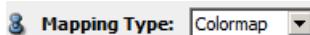
Checking the *lighting* option simulates the slice being illuminated by a head light. Thereby, the intensity of the slice is modulated according to the direction of the slice's normal. Thus, the more obliquely the slice is viewed the darker it becomes.

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. If *histogram* 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. Alternatively, *colormap* can be used to activate pseudo-coloring. The default colormap is a simple linear gray ramp. The port is hidden if the module is connected to a color field or a multi-channel field.

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,

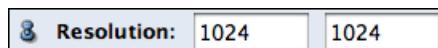
coarse, *medium*, *fine*, and *finest*, correspond to an internal resolution of the underlying texture map of 128, 256, 512, and 1024 square pixels, respectively. The fifth choice, *user-defined*, allows explicit specification of the resolution.

The toggle labeled *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.

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.

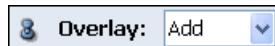
Finally, the toggle labeled *square texels* control how data values are extracted when the oblique slice is axis-aligned. If checked, the slice will display square texels. It means that an isotropic sampling is done and so, data values are displayed like squares. Otherwise, original data values are reproduced.

Resolution



This port is visible when the resolution type *user-defined* is selected. Specify width and height of the underlying texture map.

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 when using *linear mapping* or that only fully transparent parts are transparent while others are fully opaque when using *colormap mapping*. *Alpha* means that opacity is taken as is.

Channels



Channels:

This port is displayed 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.

1.90 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 two mapping methods. The less complex mapping technique uses an external colorfield with two threshold values. The default colormap is a linear gray ramp, so the 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.

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 *histogram* mapping is selected. See also *Colormap*.

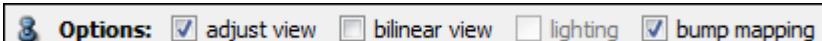
Ports

Orientation



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

Options



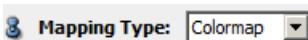
If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. The *bilinear* toggle determines how color is interpolated within a slice. By default, the toggle is off and constant interpolation is used. With constant interpolation individual pixels become visible. Bilinear interpolation often produces somewhat blurry results. Checking the *lighting* option simulates the slice being illuminated by a head light. Thereby, the intensity of the slice is modulated according to the direction of the slice's normal. Thus, the more obliquely the slice is viewed the darker it becomes. The *bump mapping* option, enables a bump mapping visualization of the slice.

Depth



Factor controlling the bump mapping visual depth effect

Mapping Type



This option menu lets you select between the two different mapping methods, namely *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.

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*. The default colormap is a linear gray ramp.

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 when using *linear mapping* or that only fully transparent parts are transparent while others are fully opaque when using *colormap mapping*. *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 displayed 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

```
createImage <image base name>
```

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

1.91 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:	<input checked="" type="checkbox"/> Dip	<input checked="" type="checkbox"/> Strike	<input checked="" type="checkbox"/> Dragger
--	-------	---	--	---

Show or hide the slices or the dragger.

Mapping type

	Mapping type:	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



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



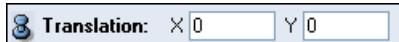
Choose colormap if mapping type is set to colormap.

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.

Translation



The translation of the dragger center with respect to the center of the base plane.

Rotation



The rotation of the dragger within the base plane.

Additionally, the color-mapping ports of *ObliqueSlice* are provided.

1.92 ParallelMovieMaker

This script module can be used to create an animation consisting of a series of 2D images in TIFF, JPEG, or PNG format, generated in parallel by instances of Amira running on several 'slave' machines (typically a cluster of PCs with graphics board). This module - running on the 'master' machine - can start or kill automatically the slave Amira instances. This requires that the user can connect to slave systems via ssh without needing a password (you may also customize a copy of this script module with your preferred remote command). This module also communicates with Amira instances by using commands *app listen*, *app send*.

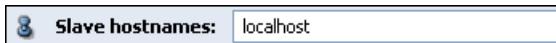
WARNING: Using this module can create security holes, as it makes Amira listen commands sent to socket ports. Do not use this unless behind a firewall and if you know what you are doing.

This module requires to provide the filename of an animation script, which can be obtained for instance by saving a network containing a *time module*. The first module with a time port that has no upstream connection with another time module will be used to control the animation. The animation script can be then loaded on every slave. When starting the animation, the image sequence is split across the slaves: the time value will be set iteratively on each slave (depending on interleave policy), the network is fired so that other objects can adjust themselves, and a snapshot is saved.

As the input files and output directory can be accessed in parallel, it is recommended for best performance to have these files located either on a fast shared storage or on a local storage of each slave system.

Ports

Slave hostnames



List of slave hostnames.

Slaves

	Slaves:	(re)start	kill	(re)load script	start script
--	----------------	-----------	------	-----------------	--------------

- *restart*: Starts or restarts Amira instances (kill and restart).
- *kill*: Kills Amira instances started by this module.
- *Load or reload script*: makes Amira instance load the job script *start script*: start animation.

Job script

	Job script:	/tmp/test.hx	Browse
--	--------------------	--------------	--------

Animation script filename.

Output images

	Output images:	/tmp/output-####.jpg	Browse
--	-----------------------	----------------------	--------

The name of the resulting 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. 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.

Time steps

	Time steps:	start 1	total 10
--	--------------------	---------	----------

The start step and the number of steps (number of saved images).

Resolution

	Resolution:	X 706	Y 458
--	--------------------	-------	-------

This is the requested resolution for the generated images (see below offscreen option). It initialized to the size of the viewer when this script module is created or reloaded.

Slaves display



This is passed as DISPLAY environment variable to the slave instances of Amira. It can be used for instance for checking what is actually displayed on the slaves, including the messages output to the slave's Amira console.

Options



- *offscreen*: If selected, offscreen snapshots are generated using the specified resolution (offscreen rendering). Otherwise, direct snapshots of the viewer are saved. See viewer command snapshot. The viewer window will be tentatively resized to required resolution, however this may fail if the viewer is defined in layout preferences as a top level window. Then the default resolution of the viewer will be used. Notice also that off-screen rendering may be different from viewer rendering as different graphics capabilities or limitations or driver issues can be involved.
- *bg image*: Display image sequence as background image (see viewer command setBackgroundImage). This may be used for control.
- *interleave*: When selected, the image sequence is interleaved across slaves so that each slave generate an image closest to each other, for instance 2 slaves will generate in turn images 1,3,5... and 2,4,6... Otherwise, each slave generates a contiguous sequence of images (1,2,3... - n,n+1,n+2) In some case this option can impact performance. Interleave mode can also limit latency for control display with the background image option.

1.93 PlotSpreadSheet

This module uses a connected *spreadsheet* or *lineset* and displays its numeric columns in a separate plot window. For linesets, the point coordinates x,y,z and per vertex data values are each regarded as comlumns. Each line of a lineset leads to a separate graph in the plot.

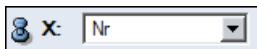
Connections

Data [required]

The *spreadsheet* of *lineset* object which contains the information to be plotted.

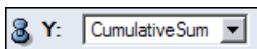
Ports

X



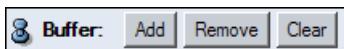
The column in the connected spreadsheet that is used for the X axis. Only columns that contain numeric values can be selected. When another column is selected, the plot will automatically switch to the newly selected data.

Y



The column in the connected spreadsheet that is used for the Y axis. Only columns that contain numeric values can be selected. When another column is selected, the user must add or remove this column using the *Buffer* port. As a special entry, *All* refers to all columns of the table.

Buffer



This port allows the user to specify which data columns should be plotted. Only the data in the Y port will be considered. Single curves can be added, removed, or all currently displayed curves can be removed from the plot window.

Plot



Pressing this button will show the plot window.

Commands

`getActiveYCurves`

Returns a list of numbers which identify the column indexes in the spreadsheet of the currently displayed curves.

`getActiveXCurve`

Returns the index of the currently used X axis.

`setActiveYCurves [list]`

Sets a list of numbers which identify the column indexes in the spreadsheet of the curves to be displayed.

`setActiveXCurve [number]`

Sets the index of the X axis to be used for plotting.

`getXLabel`

Returns the currently used X label.

`getYLabel`

Returns the currently used Y label.

`setXLabel [name]`

Sets the string used for the X label.

`setYLabel [name]`

Sets the string used for the Y label.

`createSnapShot [file]`

Creates a snapshot of the plot window and saves it in the file with the name 'file'. The type of image created is determined by the extension of 'file'.

1.94 PointProbe

See Section *Data Probing* for details.

1.95 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.

1.96 ProbeToLineSet

This module can be connected (tight connection) to a *LineProbe* or *SplineProbe* module. It saves the probe line or probe 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 or LineProbe module.

Data [optional]

Can be optionally connected to a *LineSet*. A copy of that *LineSet* is used to save the probe lines.

1.97 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 data set 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 *CylinderSlice*, which defines a cylindrical ROI.

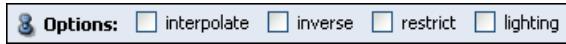
Colormap [optional]



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.

restrict: Enable to restrict the visualization to a the region-of-interest. Minimum/Maximum x-, y-, and z-coordinates of the region-of-interest could be controlled with a dragger (see *dragger* check box) or with the ports *Min. slice* and *Max. slice*.

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 first toggles can be used to turn off the *yz*-, *xz*-, or *xy*-slice. The last toggle *dragger* enables to show/hide the dragger visibility which could be used to restrict the visualization of this display module. This toggle is enabled only if the *restrict* option is checked.

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 displayed 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.

Min. slice

 Min. slice:	x: 0	y: 44	z: 0
---	------	-------	------

This port is only visible if *restrict* option is checked. It controls the Minimum x-, y-, and z-coordinates of the region-of-interest.

Max. slice

 Max. slice:	x: 224	y: 255	z: 127
---	--------	--------	--------

This port is only visible if *restrict* option is checked. It controls the Maximum x-, y-, and z-coordinates of the region-of-interest.

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.

1.98 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



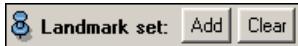
If the 3D sphere option is selected, a sphere is shown at the 3D position of the selected location. Otherwise only the three projections of the sphere are shown.

Size



Radius of sphere and circles in percent of the bounding box diagonal length.

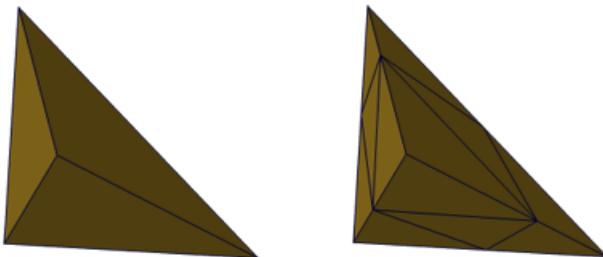
LandMarkSet



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 *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.

1.99 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.

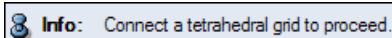
Connections

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.

1.100 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 *Merge-Conflict*.

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*.

1.101 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.

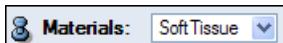
Ports

Info



This port becomes only visible in case no input grid is connected, or a label field is connected.

Materials



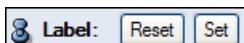
A list of materials that is defined within the parameters of the input grid. This port is only visible in case no label field is connected.

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.

1.102 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 coordinates, the voxel size is also displayed. The other ports depend on whether the input field contains ordinary numbers or labels as used in image segmentation (see *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.

1.102.1 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, *Lanczos*, 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 *Resolution* port will be enabled to take your input. If you select ‘voxel size’, the *Voxel size* port 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.

1.102.2 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 each direction. Note that no enlargement is possible.

As above, the *Apply* button initiates the resampling.

1.102.3 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.

1.102.4 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.

Connections

Data [required]

The underlying data field to be resampled.

Reference [optional]

A reference lattice. It can provide the new resolution, the new voxel size, or the new lattice.

Ports

Input Resolution

 **Input resolution:** 128 x 128 x 87,

Displays the resolution of the input data set.

Input Voxel Size

 **Input voxel size:** 1.568629 x 1.568629 x 1.583864

Displays the voxel size of the input data set (if available).

Filter

 **Filter:** ▾ Triangle

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

 **Mode:** dimensions voxel size reference

This lets you select whether you want to specify the new resolution or the new voxel size. 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

 **Resolution:** x y z

Specifies the resolution of the output data set. This port will only be visible if the input data set does not contain labels.

Voxelsize

	Voxel size:	x 1.56863	y 1.56863	z 1.58386
--	--------------------	-------------	-------------	-------------

Specifies the voxel size 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

	Average:	x 2	y 2	z 1
--	-----------------	-------	-------	-------

Specifies how many labels should be averaged during down-sampling. This port will only be visible if the input data set contains labels.

1.103 SQLite

This module lets you connect to a SQLite database. You can execute SQL statements and retrieve the results using Tcl.

Connections

Ports

Filename

	Filename:	c:/example.sql	<input type="button" value="Browse"/>
--	------------------	----------------	---------------------------------------

The path of the database.

Commands

```
exec <SQL statement>
```

Returns the result of the SQL statement as a nested Tcl list. Inner list elements have the following form:

```
{ (<rowname> <value>) * }
```

You can iterate over the result with

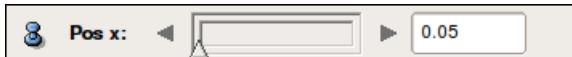
```
foreach row [SQLite exec "<statement"] {  
    foreach {column value} $row {  
        ...  
    }  
}
```

1.104 Scale

This module displays a 2D coordinate system on top of the rendering area. The coordinates are 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 projections only for the mentioned plane. The location and size of the coordinate system can be controlled by the ports. If fixed size option is unchecked, the module scales things to get "nice" numbers (no fractions). Create an instance with the *Create* menu. This may be useful, for example, to produce snapshots that contain the scale. For better results, use a 1x1 tiling for the snapshot.

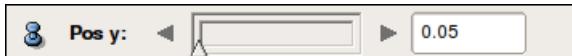
Ports

PosX



The position of the origin (fraction of horizontal size of the viewing area).

PosY

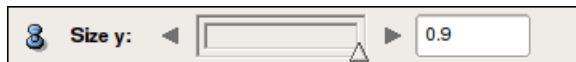


The position of the origin (fraction of vertical size of the viewing area).

SizeX



Maximum size of the coordinate system (fraction of viewport size).

SizeY

Size y: A slider control with a triangular arrow pointing right.

Maximum size of the coordinate system (fraction of viewport size).

Frame

Frame: x-Axis y-Axis Border

Select the parts of the frame which should be drawn.

Ticks

Ticks: Show Grid Text

Select whether Ticks, Grid, and/or Text should be drawn.

SubTicks

Sub ticks: Show Grid

Select whether SubTicks and SubGrid should be drawn.

Unit

Unit: units

Units label (e.g., meters, mm, etc.) that is displayed at the axes.

Color

Color:

Color of the coordinate system.

LineWidth

Line width: 1

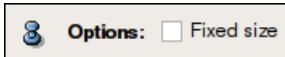
Line width of axes and tick marks.

Font

Font: Sans Serif (14 pt.)

Font type, size (in points) and color of displayed text (units and numbers).

Options



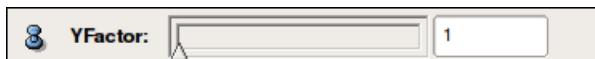
Fixed size : if checked, scale bars screen size is fixed in pixel size. Otherwise, the module scales things to get "nice" numbers (no fractions), and scale bars screen size is changing accordingly.

XFactor



Multiplicative factor of horizontal displayed text value. Default value is 1. This is used to add a virtual scale, /eg convert pixel size to model size.

YFactor



Multiplicative factor of vertical displayed text value. Default value is 1. This is used to add a virtual scale, /eg convert pixel size to model size.

Commands

```
setTextFormat <format>
```

Sets the format used for the dispaled text in printf syntax. For example, the default format is %.4g.

```
getTextFormat
```

Returns the printf format used for the displayed text.

1.105 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 horizontal input field for a bounding box. It contains six numerical values separated by thin vertical lines: 0.134766, 46.045, 0.134766, 46.045, 128.2, and 168.7. To the left of the first value is a small icon of a cube.

Bounding box of the resulting field.

Dimensions



A horizontal input field for dimensions. It contains three numerical values: 258, 258, and 136. To the left of the first value is a small icon of a cube.

Dimensions of the resulting field.

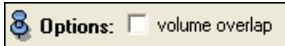
Materials



A horizontal input field for materials. It contains the word "All" followed by a dropdown arrow icon. To the left of "All" is a small icon of a cube.

Material(s) of the surface to be converted.

Options



A horizontal input field for options. It contains a checkbox icon followed by the text "volume overlap". To the left of the checkbox is a small icon of a cube.

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.

1.106 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]

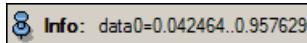
A region of interest. Lines need to go through this region to be copied to the output.

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

 Expression:	data0>0 data0<=0
---	---------------------

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> where N is the number of dimension starting with 0. An expression like data0>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.

1.107 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 *SurfaceView*.

When a *SelectRoi* is created, a 3D box with green handles is displayed in the 3D viewer that can be adjusted interactively. In the *Pool*, the ROI input port of the visualization modules should be connected to the *SelectRoi* object in order to restrict the visualization.

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:	0.392157	0.392157	0.393678
--	----------	----------	----------

Minimum x-, y-, and z-coordinates of the region-of-interest.

Maximum

	Maximum:	199.608	199.608	136.606
--	-----------------	---------	---------	---------

Maximum x-, y-, and z-coordinates of the region-of-interest.

Options

	Options:	<input checked="" type="checkbox"/> show box
--	-----------------	--

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

	Draw:	restrict	reset
--	--------------	----------	-------

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.

Size

	Size:	1.35 MB
--	--------------	---------

Size of data in the region of interest in MBytes

1.108 SeriesControl

Manages dynamic data stored as a series of HxObject.

Initially this class simply adds all objects of the first time step to the object pool. If a new time step is chosen all modules connected to data objects of the previous step will be disconnected.

Then the previous objects will be removed from the object pool and the new time step is loaded. Finally, the connections to the downstream modules will be reestablished as far as possible and the data objects will be fired.

1.109 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.

1.110 ShortestEdgeDistance

This module computes Dijkstra's shortest edge-path distance from a set of seed vertices to all vertices on a surface. The result is given as a SurfaceScalarField on the input surface. The seed vertices do not have to lie on the surface, but can be a small tolerance apart. The closest vertex of the surface itself is used as seed in this case. The algorithm is equally efficient for one seed as for arbitrarily many seeds. Press the *Apply* button to start the computation.

Connections

Data [required]

Any triangulated surface.

Vertexset [required]

Any vertex set. The vertices of the vertex set should be close to the *input surface*. You can, for example, connect a HxSurfacePathSet of the respective surface.

1.111 ShowViewerSpreadSheet

This module uses a connected *spreadsheet* and displays the content of a selected row in the 3D viewer. Position and color of the text can be specified by the user. This may be useful, for example, to produce snapshots that contain the SpreadSheet inside the viewer. For better results, use a 1x1 tiling for the snapshot.

Connections

Data [required]

The *spreadsheet* object which contains the information to be displayed.

Ports

Row number



Enables to specify the row number of the spread sheet to display.

Background



Text's background color. If the transparent toggle is off, the annotation text will be drawn inside a filled rectangle. The background color of this rectangle can be set via the color button of this port.

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) to (+1,+1).

A positive x-coordinate moves the annotation text to the right, a negative moves it to the left. Likewise, a positive y-coordinate moves the text upwards and a negative coordinate downward. The text is always moved relative to the defined origin described next.

Origin (0,0)

Origin (0,0): upper left ▾

Defines the origin used to calculate the position on the screen. The four corners of the screen can be chosen.

Position (pixels)

Position (pixels): x y

Text position must be given in screen pixels.

Position (normalized)

Position (normalized): x y

The position is given as a fraction of screen width/height, that is, a number between 0 and 1.

Font

Font: Helvetica (14 pt.)

This port enables to select and tune the font used by the displayed text.

1.112 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

 Parameters: iterations lambda

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)

Operation

 Operation:

Pressing the *Reset* button restores the original surface, i.e., the input is copied to the output.

1.113 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

 Sound file:

Browse the file to be played.

Sound

 Sound:

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.

1.114 SpatialGraphStatistics

This module calculates a tabbed *spreadsheet* providing some basic statistic information on the *SpatialGraph* data object it is attached to. The content of the spreadsheet can be viewed by pressing the *Show* button in the Properties Area of the *SpreadSheet* object.

If the spreadsheet is connected to a *SpatialGraphView* module, selecting rows of the spreadsheet will highlight the corresponding segments in the viewer. Within the *Filament Editor*, the *Graph Info* spreadsheet also supports the opposite direction: Selecting segments in one of the viewers highlights the corresponding rows in the spreadsheet.

The meaning of the columns in the *Segment Statistics* tab are:

- **Segment ID:** Unique ID of the segment in the *SpatialGraph* data structure.
- **Length:** Length of the segment in units of the bounding box.
- **Mean Radius:** Amira's skeletonization tools (*AutoSkeleton*, *Centerline-Tree*) store at each point an estimate of the local thickness of the filamentous object as point attribute *thickness*. Thereby, it is assumed that the filament has a circular cross-section with a certain *radius*. This column presents the mean radius of all points of the segment in units of the bounding box.
- **Volume:** The sum of the partial volumes for all points of a segment assuming truncated cones between paired points.
- **Label Groups...:** One column for each label group of the *SpatialGraph*. Shows the corresponding label name or *<unlabeled>* if no label has been defined for a segment.

For each label group of the *SpatialGraph* object a table is generated providing segment statistics for every label. The tables are inserted as separate tabs in the spreadsheet where each tab has the name of the label group.

- **Label Name:** Name of the label or *<unlabeled>*.
- **Mean Length:** Mean length of all segments with *Label Name*.
- **Mean Radius:** Mean radius of all segments with *Label Name*.
- **Total Length:** Sum of the lengths of all segments with *Label Name*.
- **Total Volume:** Sum of the volume of all segments with *Label Name*.
- Number of Segemnts. Total number of segments with *Label Name*.

Connections

Data [required]

The *SpatialGraph* input data object.

1.115 SpatialGraphToLineSet

This module converts a *SpatialGraph* object into an object of type *LineSet*. All *edges* of the *SpatialGraph* object will be converted to equivalent *lines* in the resulting *LineSet* object. Point attributes on the *SpatialGraph* object, if any, will be mapped onto *Data 0*, *Data 1* ... variables of the *LineSet* object. Label attributes will have no equivalent mapping in the *LineSet* object and will therefore be ignored.

Connections

Data [required]

The *SpatialGraph* data object to be converted.

1.116 SpatialGraphView

This module visualizes data objects of type *SpatialGraph*. Individual *Segments* may be displayed in wireframe mode, in *Point*-mode or alternatively as true three-dimensional tubes. In the case of *Point*-mode, the *Points* framing the *Segment* are shown; *Nodes* are displayed as spheres. Additional features of *SpatialGraphView* are pseudo-coloring and scaling of *Points*, *Nodes* and *Segments*. In order to colorize and scale the displayed items, it is possible to use one of the scalar data or labels stored in the data structure of the connected *SpatialGraph* object.

Connections

Data [required]

The *SpatialGraph* object to be displayed.

Statistics [optional]

SpatialGraphStats created *SpreadSheet* object may be used for highlighting.

Node Colormap [optional]

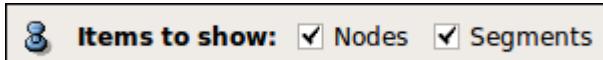
Colormap used for pseudo-coloring the *Nodes*.

Segment Colormap [optional]

Colormap used for pseudo-coloring the *Segments*.

Ports

Items to show



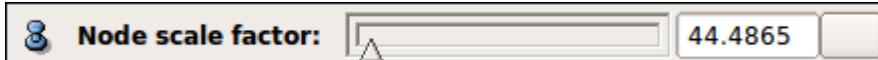
Selects the items to be shown.

Node scale



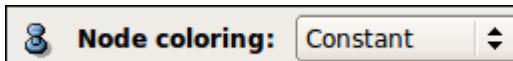
If data variables are available, this menu allows you to select the variable that will be used to scale the diameter of the spheres. If no additional data values are present, only Constant scaling will be available.

Node scale factor



Scaling factor used to adjust the diameter of the spheres.

Node coloring



If data variables or label sets are available, this menu allows you to select the

variable or label that will be used to look up *Node* colors. If Constant has been selected, the lines or tubes will be displayed in a uniform color.

Node Colormap



Colormap used for pseudo-coloring the spheres.

Node color



Color used for pseudo-coloring the spheres.

Segment style



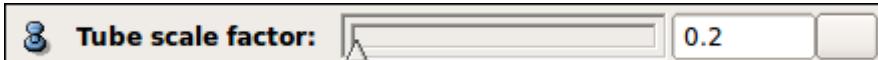
Controls how the *Segments* are displayed. Select *Lines* for the wireframe mode, *Points* for the *Points*-frame mode, *Tubes* for true three-dimensional tubes. When *Points* is selected, the *Points* defining the *Segments* are shown.

Tube scale



Optional connection to a scalar field, which can be used to influence scaling of the *Tubes*.

Tube scale factor



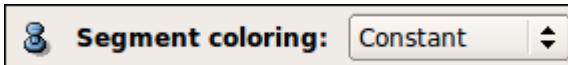
If data variables are available, this menu allows you to select the variable that will be used to scale the diameter of the tubes. If no additional data values are present, only Constant scaling will be available.

Segment width



Line width of segments when they are rendered in wireframe mode (not as *Tubes*).

Segment coloring



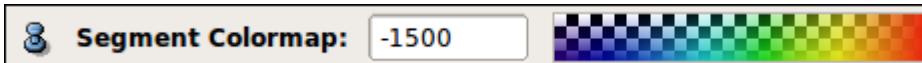
If data variables or label sets are available, this menu allows you to select the variable or label that will be used to look up *Segment* colors. If Constant has been selected, the lines or tubes will be displayed in a uniform color.

Label Coloring



If label sets are selected, this menu allows you to select how the segments will be colored. If Labels is selected, the lines or tubes will be displayed using the label-coded colors. The colors are selected automatically according to the color of the labels specified in the SpatialGraph object. If Colormap is selected, a colormap is used.

Segment Colormap



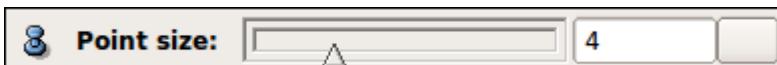
Colormap used for pseudo-coloring the *Segments*.

Segment color



Color used for pseudo-coloring the *Segments*.

Point size



Size of the markers representing the points that define the segments.

Commands

```
setLineWidth <width>
```

Specifies the width of width of lines in wireframe mode for *Segments*. This value can also be set using the *Segment width* port.

`getVisibility`

Returns the *Segment* visibility string followed by the *Node* visibility string. Where both strings contain only letters [A-Z], reflecting the current visibility mask of *Segments* and *Nodes*.

`setVisibility <segmentVisibility> <nodeVisibility>`

Sets the visibility mask of *Segments* and *Nodes*. Where *segmentVisibility* and *nodeVisibility* are strings containing only letters [A-Z].

1.117 SplineProbe

See Section *Data Probing* for details.

1.118 SplitVolume

The *SplitVolume* module creates volume stacks from mosaic or multi-stacked volumes.

Mosaic images contain the slices of a volume in an MxN pattern. The module will switch to this mode if the input consists of a single slice only (number of slices in z-direction equals 1). The output is an image stack that contains the mosaic slices reformed for 3d.

Multi-stack images are data objects that contain more than one volume. The module can only separate volumes that are not interleaved. All slices of the first volume have to be followed by all slices of the second volumes etc.. The output is a time series of volumes.

Connections

Data [required]

Connect the input data to this port. Either a single slice in mosaic format or a multi-stack of images where each slice of the first volume is followed by each slice of the second volume etc.

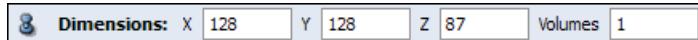
Ports

Info



This port lists the number of bytes of the input and output. Only if the two numbers are equal all the input voxels will be transferred into the output.

Dimensions



Specify the dimension of the output.

1.119 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 xz, 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]

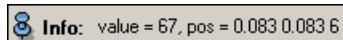
The 3D scalar field to be visualized.

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 data set 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 data sets alternately.

Overlay range

This port is only visible if a second image data set has been connected to input port *OverlayData*. It controls the mapping of the second image data set to gray values, similar to port *Range* for the first image data set.

Blend factor

This port is only visible if overlay mode *blend* has been selected. It controls the blending of both image data sets. Blend factors 0 and 1 mean that only the first or only the second data set 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.

1.120 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.

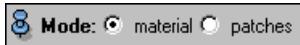
Connections

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.

1.121 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.

Points to fit [optional]

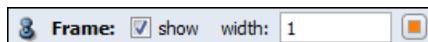
Perform a surface cut according to a plane. The plane should be defined as an *HxVertexSet*. This module type is created by a *LandmarkSet* or a *Cluster*.

ROI [optional]

Connection to a *SelectRoi* module defining a region of interest.

Ports

Frame



This port is used to display or hide the frame, set the frame width and color.

Orientation



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

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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.

Export



Export the current surface cut as a *Surface* module.

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 ma-

terials with the cutting plane will not be shown. You need to call `fire` before changes take effect.

1.122 SurfaceDistance

This module computes several different distance measures between two triangulated surfaces. 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 oct-tree 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]

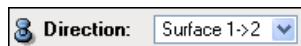
Any other triangulated surface

ROI [optional]

Connection to a module defining a region of interest.

Ports

Direction



The distance measures are asymmetric. Thus one can compute them from surface 1->2 or vice versa 2->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 its one-sided counterparts.

Consider

 **Consider:** patches contours isolated points

When computing the closest points, one can choose 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. Isolated points can be considered or not in the surface computation.

Maximal distance

 **Maximal distance:**

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

 **Above threshold:**

Threshold value for the computation of the area deviation.

Output

 **Output:** Vectors Distance

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

 **Info:**

Resulting distance measures will be displayed here: mean, standard deviation, root mean square, and maximum.

Info2

 **Info2:**

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.

1.123 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]

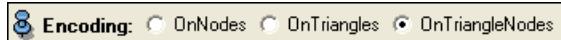
A surface that is the domain of the surface field to be generated.

Data [required]

A 3D scalar field.

Ports

Encoding



The encoding defines how the data is stored:

- On the nodes of the surface.
- On the triangles of the surface.
- Three data values per triangle, one for each node.

1.124 SurfaceGen

This module computes a triangular approximation of the interfaces between different material 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.

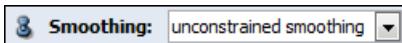
Connections

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

	Options:	<input checked="" type="checkbox"/> add border	<input type="checkbox"/> compactify
--	-----------------	--	-------------------------------------

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 enabled, an algorithm similar to the *compactify* option of the *Isosurface* module will be applied, and up to 50% less points and triangles will be produced. Alternatively, you can apply an edge-contraction technique to reduce the number of triangles. See the description of the port *Minimal Edge Length* below for a more detailed explanation.

Border

	Border:	<input checked="" type="checkbox"/> adjust coords	<input checked="" type="checkbox"/> extra material
--	----------------	---	--

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

	Minimal edge length:	0
--	-----------------------------	---

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 to ensure topological consistency.

Smooth material



On default, the value of this port is *none*. If you change it to the name of a material, a smooth surface will be generated for the selected material. Otherwise, the surfaces of all materials will show some 'ridges' at contours incident on 3 or more materials.

Material



On default, this port is invisible. It can be made visible via the Tcl command

```
SurfaceGen showMaterialList
```

On default, the value of the port is *All*. If you change it to the name of a material, only the surface of the selected material will be extracted.

1.125 SurfaceIntersector

This module intersects two *surfaces*, computes a *path* along the intersection and attaches it to each of the surfaces. Press the **Apply** button to start the operation.

Connections

Surface1 [required]

The first surface to be intersected.

Surface2 [required]

The second surface to be intersected.

1.126 SurfacePathView

This module displays a *HxSurfacePathSet*.

Connections

Data [required]

The module must be connected to a *surface path set*.

Ports

Point size

	Point size:	<input type="button" value="◀"/>	<input type="button" value="▶"/>	<input type="text" value="5"/>
--	--------------------	----------------------------------	----------------------------------	--------------------------------

This port allows the user to change the size of the displayed control-points in the paths.

Line width

	Line width:	<input type="button" value="◀"/>	<input type="button" value="▶"/>	<input type="text" value="4"/>
--	--------------------	----------------------------------	----------------------------------	--------------------------------

This port allows the user to change the width of the line segments connecting the nodes in the paths.

Control Point Shape

	Control Point Shape:	Cube	<input type="button" value="▼"/>
--	-----------------------------	------	----------------------------------

This port allows you to choose a display mode for the control points. Two modes are supported: cubes and spheres.

Control Point Color

	Control Point Color:	<input type="radio"/> active path	<input type="radio"/> inactive path	<input type="radio"/> end point	<input type="radio"/> active CP
--	-----------------------------	-----------------------------------	-------------------------------------	---------------------------------	---------------------------------

This port allows changing the color of the displayed control points.

Path Color

	Path Color:	<input type="radio"/> active path	<input type="radio"/> inactive path
--	--------------------	-----------------------------------	-------------------------------------

This port allows changing the color of the displayed surface paths.

Rendering

	Rendering:	Line Offset <input type="text" value="0.01"/>	CP factor <input type="text" value="0.0005"/>
--	-------------------	---	---

This port allows the user to influence the rendering of the lines and the control

points of the surface paths. The *Lineset offset* offsets the lines in viewing direction such that the lines are visible even though they are exactly on the surface. The second value, the *CP scale*, scales the cubes or spheres representing the control points by the given value. This value is multiplied by the value given by the *Point size* port.

1.127 SurfaceThickness

This module is useful for computing the thickness of flat regions from surface objects. To do so, at each vertex, the module computes the distance along the vertex normal to the normal's intersection with the closest triangle of the same material. To get a reasonably reliable thickness estimate the surface should be smooth and opposing triangles should be approximately parallel. The result is exported as a surface scalar field with a distance measure per vertex. The module calculates the thickness for a single material only. For surfaces composed of multiple materials the field values are set only for vertices belonging to the selected material. All other vertices are set to 0.

Connections

Data [required]

Connect this port to a 17.33 surface object.

Ports

Material



This port should be used to select the material for which the thickness is computed. For all other materials thickness is set to 0.

1.128 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 R1 and R2, you may first highlight all triangles separating these two regions by choosing R1 and R2 in port *Materials*. Highlighted triangles are displayed in red wireframe. 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.

Note: This module makes use of recent OpenGL features (OpenGL 2.0 and higher) in order to achieve maximum performance. If the graphics card does not support those features or if artifacts occur, a *Legacy rendering mode* can be enabled in the *Rendering Tab* of the *Preferences Dialog*.

Connections

Data [required]

The surface to be visualized.

ColorField [optional]

Scalar field to be visualized via pseudo-coloring. The field will be evaluated at the vertex positions of the surface and a color will be mapped onto the surface using the colormap connected to connection port *Colormap*. 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]

Colormap used for pseudo-coloring (see *Color Field*). To change the colormap right-click the colormap window or reconnect (drag) the blue connection line with another colormap data object. To change the port's default color double-(left-)click the colormap window and select a color from the *ColorDialog*. See also *Colormap*.

Texture [optional]

This port allows to connect a 2D image that will be mapped onto the surface as a texture. The image must be a single slice of type *ColorField*. A transformation

can be applied to the texture image in order to change the texture projection axis and texture coordinates scale. Note that texture projection overrides pseudo-color mode.

ROI [optional]

Connection to a *SelectRoi* module defining a region of interest.

Ports

Draw Style



This port is inherited from the *ViewBase* class and therefore the description will be found there.

Colormap



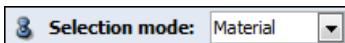
This port becomes visible only if a scalar field has been connected to the *ColorField* port.

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*.

Selection mode



This port lets you specify whether you want to highlight triangles on the basis of their associated material, their patch ID, or their boundary ID.

Patch



This port is visible if *Selection mode* is *Patch*. It allows you to highlight triangles having the specified patch ID.

Boundary id



This port is visible if *Selection mode* is *BoundaryID*. It allows you to highlight triangles having the specified boundary ID.

Materials



This port is visible if *Selection mode* is *Material*. It offers two pull-down menus each listing all *Materials* defined in the surface. Since each triangle exactly interfaces two *Materials*, triangles can be selected by choosing the appropriate pair of *Materials* in those menus. When triangles are selected they are highlighted in red wireframe. In addition, you may restrict the set of highlighted triangles by adjusting size and position of the dragger box. Use port *Buffer* to add or remove highlighted triangles to/from the internal buffer.

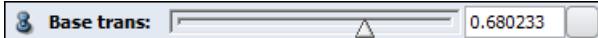
Colors



There are several different color modes:

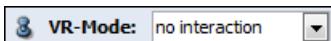
- *normal*: The surface is colored according to the color of the enclosed material.
- *mixed*: Both sides of a triangle are colored in the same color, which is a mixture of the colors of the two sides in *normal* mode.
- *twisted*: For each triangle the front and back colors are exchanged relative to *normal* mode.
- *boundary ids*: 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*.
- *same*: The material with the largest index determines the color of both sides of the triangle. The material index is defined by the order of the material in the material list in the *Segmentation Editor*.
- *constant*: Both sides of a triangle have the same color which, can be specified in the *Colormap* port.
- *patch*: All triangles of a patch have the same random color on both sides.

BaseTrans



This port is visible if *transparent* has been chosen as the draw style. It allows you to control the overall transparency of the surface object. Move the slider from fully opaque (0) to completely transparent (1) to achieve the desired appearance. It is also possible to control the transparency of materials individually. To do so, you need to create a parameter *transparency* in the *Material* bundle of the surface object's parameter section. If such a parameter exists, the transparency of the material is set exactly to this value when the slider is set to 0.5. Moving the slider thus modulates the *base transparency*, with greater values making it more transparent and smaller values making it more opaque.

VR-Mode



This port is only available in VR mode (see Virtual Reality Option 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*.

TextureWrap



Wrap mode used for the texture projection on the surface when a texture is applied to the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

1.129 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

 **Info:** 6 steps

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

 **Cached steps:** 6

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.

The third toggle is used to apply to the data of the next (or previous) timestep the geometric transformation(s) applied to the data open at the current time step.

The fourth toggle is used to copy all the parameters of the current time step data to the parameters of the data of the next (or previous) timestep.

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.

1.130 Trajectory

This module can control position and orientation of an HxArbitraryCut sliding it along a given curve or lineset. The Trajectory can be activated from an

HxObliqueSlice, for instance. Typical usage scenarios are 1) show a slice at a fixed position while sliding, 2) use a stencil mask to visualize only a circular region around the trajectory.

Connections

Data [required]

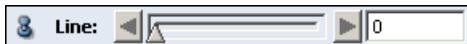
The lineset where the trajectory is part of.

Module [required]

The HxArbitraryCut modules whose slice position and orientation will be adjusted.

Ports

Line



The number of the line within the lineset that should be used as trajectory.

Position



Position on the trajectory. Can be the vertex number or line-segment number depending on the toggle "Use Line Segments"

View



If *Up front* is selected, the camera of the actual viewer is adjusted to the position and tangent of the trajectory position.

If *Frame* is selected, the border frame of the connected HxArbitraryCut module is switched on, otherwise it is switched off.

If *Orthographic* is selected, the camera is switched into orthographic projection mode.

If *Use Line Segments* is not selected, the slice position is set to the vertex positions of the trajectory. The normal direction for the plane is defined by the predecessor and the successor vertex of the line. If the option is set, the plane is positioned at the center of a line segment, and the line segment is used as plane normal.

1.131 TransformAnimation

This module allows you to control and animate the transformation of a *spatial data object*. This means translating, rotating, scaling and shearing an object without changing the camera. It is possible to define complex transformations consisting of an arbitrary number of basic transformations (translations, rotations, etc.) in an arbitrary order. Note, that the order of transformations is important: Translating an object after a rotation yields a different result than the other way around.

The module has two modes:

- **Edit Transformation:** In this mode you can create a complex transformation by editing a set of basic transformations (translations, rotations, etc.). New basic transformations can be added, existing transformations can be reordered and removed.
- **Edit Animation:** This mode assumes that a complex set of transformations exists and only its parameters can be changed. This means the degrees of a rotation or the size of a scale operation. Basic transformations can **not** be added, reordered or removed. However, a set of parameters can be saved as a so-called *animation key*. This allows you to create an animation.

Connections

SpatialData1 [required]

Connection to the *spatial data object* to be transformed.

SpatialData2 ... SpatialDataN [optional]

Additional *spatial data objects* to be transformed in the same way as the first one. This becomes handy if you want to control a group of objects belonging to each other. You can connect an arbitrary number of objects.

Time [optional]

Connection to other time modules.

Ports

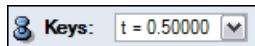
Mode



Switch between editing modes.

Time

Only available in *animation* mode. Controls the current time of the animation sequence.

Keys

Only available in *animation* mode. Allows you to select a previously defined animation key.

Action

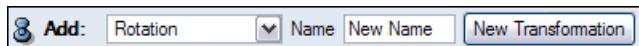
Only available in *animation* mode. Allows you to add, remove, or replace animation keys.

Info

Displayed at the beginning of a every basic transformation in order to identify it easily.

Edit

Only available in *transformation* mode. Allows you to add, remove, or reorder basic transformations.

Add

Only available in *transformation* mode. Allows you to add a new basic transformation and give it a name.

1.132 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



Type of distortion measure: choose from angle, area or stretch (= length distortion)

Encoding



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.

1.133 TrideliyView

The autostereoscopic displays are special display screens that allow many different types of content to be seen spatially in 3D. The major innovation with autostereoscopic displays compared to other stereo display technologies is that the 3D viewing becomes possible without special glasses. This is achieved by the application

of optical technologies which ensure that each eye of the viewer sees a slightly different perspective.

Once the correct display is detected Amira uses the *Tridelity* display as a full screen rendering area.

The following displays are currently supported:

MV1900

Size: 19"

Panel resolution: 1280x1024

Number of views: 5

Ideal viewing distance: 1.5 m

MV2700

Size: 27"

Panel resolution: 1920x1200

Number of views: 5

Ideal viewing distance: 2.5 m

MV4200

Size: 42"

Panel resolution: 1920x1080

Number of views: 5

Ideal viewing distance: 3.5 m

MV5700

Size: 57"

Panel resolution: 1920x1080

Number of views: 5

Ideal viewing distance: 3.5 m

SL2400

Size: 24"

Panel resolution: 1920x1200

Number of views: 2

Ideal viewing distance: 0.75 m

For more information, please refer to the *Tridelity* homepage (www.tridelity.com).

Setup

- Windows: The *Tridelity* display must be configured as a secondary display in extended desktop mode.
- Linux: The *Tridelity* display must be configured as a separate X screen with Xinerama enabled.

Important notes

- If the *Tridelity* viewer is not updated, you need to reposition the Amira application window such that the viewer has an overlap of at least one pixel with the *Tridelity* screen.
- The rendering performance in the autostereoscopic screen depends on the performance of your graphics card. Rendering may be slow on older graphics cards.
- If an area on the *Tridelity* display is not updated correctly, reducing the quality should help.
- Currently the *Tridelity* viewer offers no direct interaction. Thus camera navigation has to be performed in *viewer 0*.
- Only the scene visualized in *viewer 0* will be visualized in the *Tridelity* viewer.

Connections

Ports

Eye dist



Sets the stereo offset (the distance between the eyes).

Focal plane



Sets the stereo balance (the position of the zero parallax plane). This determines whether objects will be seen in front of the screen or behind it.

Quality



Sets the rendering quality (the size of the offscreen render targets). Low rendering quality will result in a higher frame rate.

Display type



See supported displays above.

View



Enable/disable the *Tridelity* viewer.

1.134 VRML-Export

This module enables you to export a *triangular surface* and optionally the 3D data set from which the surface is derived into a VRML scene. The scene consists of a coordinate system and some animations: the 3D data set is displayed by three orthoslices. To get an idea how the basic 3D data set 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]

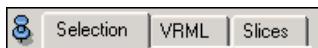
The surface to be inserted into the VRML scene. It can either be an AmiraSurface data set or the SurfaceView module.

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



In simple export mode, the coloring mode and the number of primitives is shown here. (only available in simple mode).

Selected



This port shows the selected materials (not available in simple mode).

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



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



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



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



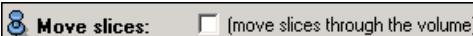
Adding a specular color to the scene which makes the surface more look like plastic.

Render smooth



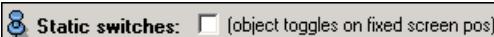
Gives the scene a smoother appearance by using vertex normals.

Move slices



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



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



Toggle this for displaying the material names near the material toggles in the VRML scene.

Export slices

 Export slices:

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

 Data window: min max

Needed to map data values of input field to gray values. See module *OrthoSlice* for details.

Slice numbers

 Slice numbers: x y z

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

 Filename:

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.

1.135 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.

1.136 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

1.137 VertexShift

The module displaces the vertices of a surface. The displacements are given by one or more vector fields. The vector field(s) might be assigned a multiplicative coefficient. 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 vector field. If a vector field is connected, then the *Lambda 1* multiplicative factor port appears as well as a second vector field port in order to connect another optional vector field and so on.

Ports

Info



Informs that a vector field should be connected to the module.

Lambda 1



Multiplicative factor to modify the contribution of the first vector field to the surface displacement.

1.138 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

Color



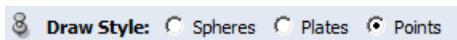
If the module is connected to a *line set*, this port allows selecting the line set's data field for coloring.

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 quadrants 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.

- *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.
- *transparent*: 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.
- *show all*: The default is to display only vertices which have been added to the buffer (initially the buffer contains all vertices). By clicking this toggle you can choose to ignore the buffer state, thus displaying every existing vertex.

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.
- *Box*: toggles boxes to select vertices.

- *Invert*: exchanges buffer content.
- *Draw*: activates a lasso style selection mechanism: Using the mouse you can draw a contour in the viewer. All triangles within this contour will be highlighted. If CTRL is pressed while drawing, the triangles within the curve are deselected.

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.

1.139 ViewerPlot

You may connect this module tightly to a module that provides a *plot window* in order to display the plot within Amira's viewer. If that module creates more than one plot window, it may be useful to connect one *ViewerPlot* module for each plot window. This may be useful, for example, to produce snapshots that contain the plot display inside the viewer. For better results, use a 1x1 tiling for the snapshot.

Connections

PlotModule [required]

This port must be connected to a module providing a plot window and an interface to access this window. *Histogram* and *LineProbe* are examples for such modules.

Ports

Which Plotwindow



This port is only shown if there are more than one plot window created by the module it is connected to. Use it to select which plot to display in the viewer.

Options



This port provides the following two toggles:

- **frame:** Draw a frame around the plot in the viewer.
- **transparent:** If off, the plot has an opaque background.

Position



This port provides the following two toggles:

- **relative position:** Toggle between absolute or relative plot positions in viewer.
- **cycle position:** If there is more than one plot created by the plotting module, the plot position in the viewer is cycled through the four corners clockwise.

Transparency



A value of 1 produces a completely transparent plot background, while 0 results in a fully opaque background.

Absolute Position

 **Absolute Position:** x y

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

Relative Position

 **Relative Position :** x y

Position of plot in viewer in normalized coordinates (0..1) relative to the lower left corner. If one of the numbers is negative, it is interpreted relative to the upper right corner.

Size

 **Size:** width height

Size of the plot in pixels.

Actions

 **Actions:**

Pressing the *Edit* button opens the object editor window of the plot. If you press the *Show plot window* button, the extra plot window is shown on the screen. Pressing the same button again, the plot window vanishes.

1.140 Volren

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 (see also *Vortex*).

The Volren module provides several visualization techniques: shaded and classical texture-based volume rendering (VRT), maximum intensity projection (MIP), and digitally reconstructed radiograph (DRR). The standard VRT and its shaded version enable a direct 3D visualization with flexible color and transparency colormaps with virtual lighting effects for better rendering of complex spatial structures and enhancing fine detail. The MIP rendering allows the visualization of the

highest or lowest intensity in a data volume along the current line of sight. The DRR simulates a radiograph display from arbitrary views using the loaded volume data. Furthermore, the Volren module enables you to render segmented regions at the same time with different colormaps. A LabelField can be attached to the Volren and each material of the label list can be rendered separately. In order to optimize the performance, the image volumes are represented at lower resolution in case of interactive rendering. For relatively large image volumes, you can also switch to a constant "Low Resolution" mode (see below). Volren is built for use with popular graphics accelerator technology such as the NVIDIA Quadro FX(TM) graphics boards.

Notes and Limitations

- On some systems a significant slowdown can occur if the data set is larger than the available texture memory (which is typically 4 - 16 MB). In this case select the option *LowRes only* (see below).
- When two or more *Volren* modules are used to render intersecting volumes (multivolume rendering), some rendering inaccuracies may occur. In case of multivolume rendering the supported combinations of modes are one MIP together with one MIP or one VRT with one VRT (see below *Mode*). Other modes and combinations may lead to incorrect visual results.
- The *Apply* button is enabled only when the rendering must be recomputed.

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.

Labels [optional]

Input *LabelField* to be rendered.

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.

RoiCornerCut [optional]

Connection to a *CornerCut* module providing a cutting box positioned at a spec-

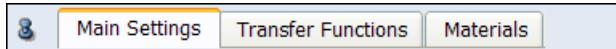
ified corner of the bounding box. If such a module is connected, the selected corner region of the volume will be not displayed.

Color1, Color2, Color3, ... [optional]

Colormaps used to visualize the data.

Ports

Tabs



Main Settings. The parameters used for the shading and transparency effects, and the rendering modes are displayed in this tab.

Transfer Functions. In this tab you can add or manipulate the Volren colormaps. This is particularly useful when a *label field* is rendered together with the volume data. In this case, several different colormaps could be used at the same time (see Materials tab below). Pressing the *New* button creates a new colormap which can be controlled by its *Option*, *Color*, and *AlphaScale* ports. To delete a colormap just press the *Delete* button.

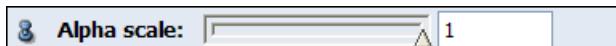
Materials. When a *LabelField* object is attached to the Volren module, the material list is shown in this tab. For each material, a new port is opened. Clicking on the colormap icon allows you to select a colormap from the list in the *Transfer Functions* tab (see above). To create and use a new colormap, it is necessary to generate it directly in the *Transfer Functions* tab and to select it in the Materials tab. By activating the option *Constant*, the material volume will be rendered using a constant color instead of a colormap. To modify the color, click on the color icon. Note that if a volume data set is connected to the Volren, only *LabelField* objects of the same volume size can be connected.

Colormap



Port to select a colormap.

Alpha scale



A global factor to change the overall opacity of the object independent of the data value.

Mode

	Mode:	<input checked="" type="radio"/> VRT	<input type="radio"/> DRR	<input type="radio"/> MIP	<input type="radio"/> LowRes only
--	--------------	--------------------------------------	---------------------------	---------------------------	-----------------------------------

VRT. Texture-based volume rendering, with optional shading effects. If *VRT* is selected the *Shading* port is activated.

DRR. Digitally reconstructed radiograph. This technique simulates a radiograph from arbitrary views using the loaded data. If this mode is selected, the *Gamma* port is shown.

MIP. 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. This mode is especially useful for very sparse data sets, for example, angiographic data or images of neurons.

LowRes only. The image volumes are rendered always at lower resolution. For relatively large image volumes select this mode to optimize the computational efficiency.

Gamma

	Gamma:	<input type="range" value="0"/>	0
--	---------------	---------------------------------	---

This port controls the shape of the transfer function when *DDR* mode is selected. The opacity value is taken to be proportional to the data values. The smaller the gamma value is, the more prominent regions with small data values will be.

Shading

	Shading:	<input type="radio"/> None	<input type="radio"/> Diffuse	<input checked="" type="radio"/> Specular	<input type="radio"/> User-defined
--	-----------------	----------------------------	-------------------------------	---	------------------------------------

This port is shown if *VRT* is selected in port *Mode*. If *None* is selected, no shading effects are visualized. With *Diffuse* and *Specular*, two pre-set shading effects are activated. Option *Diffuse* simulates a diffuse light source, while *Specular* simulates a specular one. The option *User-defined* opens the *Coefficients* port to specify the lighting parameters.

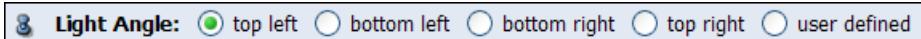
Coefficients

	Coefficients:	ka <input type="text" value="0.3"/>	kd <input type="text" value="0.5"/>	ks <input type="text" value="0.7"/>	light <input type="checkbox"/>
--	----------------------	-------------------------------------	-------------------------------------	-------------------------------------	--------------------------------

This port is enabled when the *User-defined* option in port *Shading* is selected. In this port you can specify the parameters of the light source. *Ka*, *kd* and *ks* are the

coefficients for ambient, diffuse, and specular components of the illuminating light. To set the color of the light source, click on the *light* color icon.

Light Angle



This port is enabled when Diffuse, Specular, or User-defined options are selected in the *Shading* port. This port controls the position of the simulated light source.

Light Angle Yaw



This port is enabled when option *User-defined* in the port *Light Angle* is selected. It allows you to displace the light source on a horizontal semi circle around the volume data.

Light Angle Pitch



This port is enabled when the User-defined option in the Light Angle port is selected. It allows you to displace the light source on a vertical semi circle around the volume data.

1.141 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.

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 visualized. Alternatively an RGBA data volume (Color-field) 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 colormap's alpha value is used for both absorption and emission. In *LumAlpha* mode, the colormap's 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



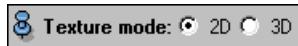
A global factor to change the overall transparency of the object independent of the data value.

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 palettes. 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.

1.142 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 or color 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 Volren or an Isosurface module.

Connections

Data [required]

The input data set to be edited (uniform scalar or color 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.

Padding value



This port specifies by which data value(s) voxels in selected regions are replaced when either *Cut Inside* or *Cut Outside* is pressed. For color fields an own value for each channel can be set.

Cut



If the button *Inside* is pressed, voxels inside the region selected by a dragger shape are replaced with the value(s) given by the *Padding Value* port. If the button *Outside* 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 *Inside* is pressed, voxels inside the region selected by a dragger

shape are replaced by the original data values. If the button *Outside* 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.

ColorChannels



If a color field is attached to the module this port allows to select if changes should affect only the *Alpha* channel or *All* channels.

1.143 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.

2 Molecular Option

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 possibility 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]

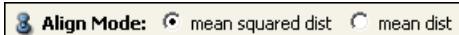
First molecule to which the second molecule will be aligned.

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**Align Mode**

Use this port to select alignment mode.

'Mean squared dist' tries to align molecules in the least squared sense, i.e. the module tries to optimize the transformations such that the sum of squared distances of the atoms in both molecules is minimal. Note that large distances are punished much higher when this alignment mode is selected.

'Mean dist' tries to find transformations such that the sum of distances becomes minimal.

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*.

silon. 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.

Custom align level



Apart from using computed alignment levels you can also create a custom level interactively. *New level* will create a new level in both molecules with the name *custom_align_<index>*. You can remove the currently active custom alignment level with *remove level*. You add groups which will be aligned by selecting one or multiple atoms in each of the molecules and clicking on *add group*. The number of atoms does not necessarily need to be the same, their center of gravity will be used for alignment. 'Clear Groups' removes all groups of the currently selected custom alignment level.

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

	Options:	<input type="checkbox"/> show alignment	<input type="checkbox"/> save first
--	-----------------	---	-------------------------------------

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



Input: molecules A+B molA/motif molB/motif

A menu with three radio buttons. The first one, 'molecules A+B', is selected and highlighted with a black border. The other two options, 'molA/motif' and 'molB/motif', are not selected.

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



Motif:

A text input field with a placeholder 'Motif:' preceded by a small icon. The field is empty.

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



Align Type: protein local

A menu with two dropdown lists. The first list contains 'protein'. The second list contains 'local' and has a downward arrow indicating it's a dropdown menu.

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.

Indel



Indel: -2

A horizontal slider with a central triangle button for adjustment. The number '-2' is displayed to the right of the slider.

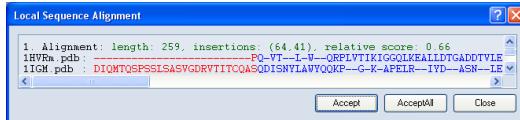
The sequence alignment algorithm requires Indel parameter. When the parameter is set to 0, insertion and deletions will not be punished. The smaller the value gets the less likely it will be that many deletions will appear in the sequence alignment. The name insertion/deletion refers to one of the sequences. Deletion in sequence B can be regarded as insertion in sequence A and vice versa.

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 AtomicMolecularDensity

This module computes the spatial distribution density of atoms on a regular grid. Inputs can be either molecules, trajectories or trajectory bundle objects. Atoms

can be filtered according to mmff94 atom types. Using the module makes most sense for TrajectoryBundles containing a set of molecules from which you want to derive some spatial distribution property like pharmacophores. TrajectoryBundles can be created from single molecules with the MolTrajBundleConverter module.

Preparing data

The following example demonstrates how to compute pharmacophore densities for hiv protease. We compute the densities from two known ligands but the example can be easily extended to a large number:

```
# fetch molecules from pdb
newMolFromPDB lpro
newMolFromPDB lhvr
# use lpro as reference and align others to it
lhvr alignToProtein lpro
lhvr applyTransform
# restrict to the ligands
lpro sel r/type=A88
lpro restrictToSelection
lhvr sel r/type=XK2
lhvr restrictToSelection
# compute bonds orders and add hydrogens
lpro computeBonds
lpro addHydrogns
lhvr computeBonds
lhvr addHydrogns
mergeMolecules
```

The last command will create a MolTrajBundleConverter module. In this object you select TrajectoryBundle and press apply. The molecules are now combined in one bundle which can be used as input for the AtomicMolecularDensity module. Parameterizing the MMFF atom types and filtering for certain atom types (like hydrogen acceptors) allows the creation of pharmacophore densities for this type.

Options

First you will need to determine on which grid the density is generated. The 'Bounding Box: Compute' button will generate the smallest possible bounding

box surrounding all atoms of the input object. You can show and manually modify the bounding box. By default the module will generate a density for all atoms. A filter based on mmff94 atom types can be applied by selecting 'Atom Types: MMFF' and using the atom type selector. The input data must, however, first have mmff94 atom types assigned. You can do this with the 'Parameterization: MMFF' button. If the input is a trajectory or a bundle of trajectories which have an observable describing the energy of each time step, you can weigh each time step according to this energy. To do so, select an appropriate 'Energy weighting option' and enter the name of the observable.

Connections

Data [required]

Input data which can either be a single molecule, a trajectory or a trajectory-bundle.

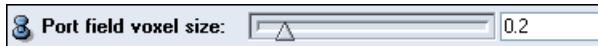
Ports

Lattice Box



Allows to show the bounding box for manual adjustment.

Port field voxel size



Determines the size of a voxel. Adjust the voxel size according to the bounding box size, so that the grid dimensions do not become too large.

Bounding Box



Allows to compute the smallest bounding box enclosing all atoms in the data object.

Radius Options



During the computation, each grid point within the radius of an atom is added to

the density. The larger the atom radius the smoother the density will be but also the more blurred. The 'fixed' option will use the radius which is given in the float slider below for every atom. 'VDW' means that the standard vdW radius is used for each atomic number. 'MMFF VDW' means that the vdW radius will be derived from half of the equilibrium vdW distance between atoms of this type. In the latter cases, the float slider acts as a factor which scales the radius.

Atom Radius



Determines the fixed radius or the scale factor depending on the radius option.

Atom Types



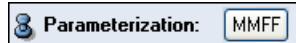
By default, all atoms will be counted in the computation. By selecting 'MMFF' you can select a subset of atoms by filtering for specific MMFF atom types with the atom type selector.

Selector



Shows the atom type selector which allows to select one or several atom types or classes of atoms types. If the MMFF option is chosen in the atom type options, only atoms matching these types will be considered in the computation.

Parameterization



This allows to assign MMFF parameters to the input data so that MMFF atom type filtering is possible.

EnergyWeighting Method



If the input data trajectories contain an observable, this can be used to assign a weight to each timestep. Weight means that the observable value will be used as the weight directly, i.e. the density contribution of this timestep will be multiplied with the value. 'Sum' will divide the contribution by the sum of all observable values. 'Boltzmann-Sum' will multiply the contribution with the boltzmann factor of the value. This makes sense if the observable describes the

energy. 'Linear-minmax' scales the contribution linearly so that the timesteps with the minimum observable value are multiplied with one while the timestep with the highest value is multiplied with 0.

Energy Weighting Observable

 Energy Weighting Observable:	<input type="text" value="energy_total"/>
--	---

Name of the observable that is used for energy weighting.

BBox Restrict

 BBox Restrict:	<input checked="" type="checkbox"/> Do
--	--

Allows to restrict the input data to the current bounding box. The computation will be faster after applying this restriction but this changes the input data object.

Field Normalization

 Field Normalization:	<input checked="" type="radio"/> none	<input type="radio"/> Sum=1
--	---------------------------------------	-----------------------------

Allows to normalize the generated density so that the sum of all grid points is 1.

2.4 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.

Culling mode

Please refer to the *ViewBase documentation*.

Colormap

This port becomes visible only if a scalar field has been connected to the color field port.

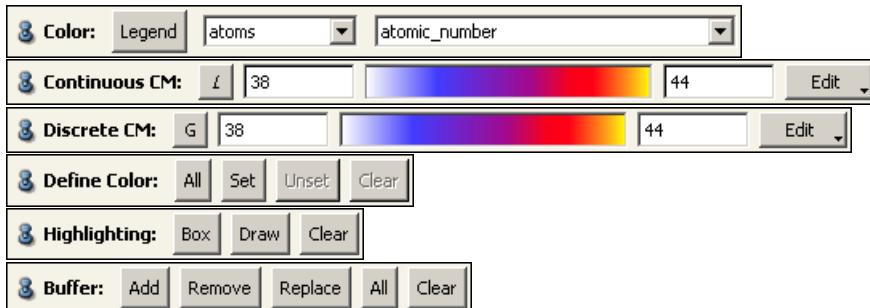
Transparency

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*.

2.5 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

Mode



Select bond calculation method. For more information on bond calculation methods please refer to *Molecule Editor*'s documentation.

Maximal atom distance

	Maximal atom distance:	<input type="text" value="1.7"/>	<input type="button" value="..."/>
--	-------------------------------	----------------------------------	------------------------------------

Available only when 'Mode' is set to 'distance'. The value determines the maximal distance between two atoms (in Å) below which they are considered to be connected by a bond.

Action

	Action:	<input type="button" value="Add"/>	<input type="button" value="Remove"/>	<input type="button" value="Replace"/>
--	----------------	------------------------------------	---------------------------------------	--

Starts the computation of the new molecule. "Add" will compute new bonds according to the chosen parameters and add the newly computed bonds to the molecule, "Remove" will remove bonds that match the respective parameters. Some of these bonds might not be present in the molecule, but this is not a problem. "Replace" will first remove all bonds and then add those that match the given parameter configuration.

2.6 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

Data [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.

Data2 [optional]

Second molecule for the computation of the intermolecular interface.

Ports

Voxel size

A horizontal input field with a small icon of a cube at the left end. The text "Voxel size:" is followed by a numeric input box 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 horizontal input field with a small icon of a cube at the left end. The text "Cutoff distance:" is followed by a numeric input box 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 horizontal menu with a small icon of a cube at the left end. The text "Distance to:" is followed by two radio buttons: one labeled "atom center" and another labeled "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 horizontal menu with a small icon of a cube at the left end. The text "Levels" is followed by a dropdown menu showing the word "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.

2.7 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

Surface Type: vdW sas ses

Choose between *van der Waals*, *solvent accessible*, and *solvent excluded* surface.

Quality

Quality: correct 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.

Radius

Radius: const attribute standard vdw mmff vdw

The radius option determines the basic radius of the van-der-Waals spheres for which the surface is computed. When 'const' is selected, a value of 1 will be used. For 'attribute' the radius is determined by the attribute given in 'radius attribute'. Then, for different force fields there exist different radii. 'standard vdw' refers to the original Amira standard vdW radii. 'mmff vdw' are the van der Waals radii according to the Merck Molecule Force Field (mmff).

Radius Attribute

Radius attribute:

Radius value needs to be specified when attribute option is selected with radius

port. This option is only available when 'attribute' option is selected with the radius port.

Atom Radius Scale and Offset

	Atom radius scale:	<input type="text" value="1"/>	<input type="button" value="..."/>
	Atom radius offset:	<input type="text" value="0"/>	<input type="button" value="..."/>

The radius of each atom as it is used in the molecular surface computation is the default atom radius as determined by the 'radius' port, scaled by the 'atom radius scale' with an 'additional atom radius' offset that is added at the end.

Probe Radius

	Probe Radius:	<input type="text" value="1.4"/>	<input type="button" value="..."/>
--	----------------------	----------------------------------	------------------------------------

The radius of the solvent probe sphere. This port is only visible for the surface types *sas* and *ses*.

Edge length

	Edge length:	<input type="text" value="0.5"/>	<input type="button" value="..."/>
--	---------------------	----------------------------------	------------------------------------

Choose the approximate edge length of the surface's triangles. This port corresponds to the number of points per 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

	Number of points per A^2:	<input type="text" value="2"/>	<input type="button" value="..."/>
--	---	--------------------------------	------------------------------------

With this port you define the approximate number of points per 2 and thereby the granularity of the surface.

Options

	Options:	<input type="checkbox"/> partial surface	<input type="checkbox"/> adjacent patches	<input type="checkbox"/> no duplicate points	<input type="checkbox"/> contract edges
--	-----------------	--	---	--	---

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

	Surface area:	<input type="checkbox"/> atom area	<input type="checkbox"/> residue area	<input type="checkbox"/> 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 appear allowing you to specify an arbitrary name for the attribute into which the surface areas will be written.

Attribute name

 **Attribute name:**

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.

Filter Options

 **Filter Options:** ignore water ignore HET

Allows common filters to be applied in addition to the filtering in the MolSelectionBrowser. The option *ignore waters* will remove all oxygens and hydrogens which are not bonded to another heavy atoms from the computation. The option *ignore het* ignores all HET groups (as defined in a pdb file).

2.8 ComputeHBonds

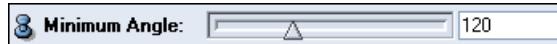
This module creates a molecule with a new level (the hydrogen bond level 'hBonds') containing possibly found hydrogen bonds. Chemical criteria are used to identify potential donors and acceptors. All non carbon heavy atoms with at least one hydrogen (explicit or implicit) are used as donors and all atoms with at least one lone electron pair are used as acceptors. Several geometric criteria are then checked including distance and angle between donor and acceptor atoms. 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**

8 **Minimum Angle:** 120

A slider control for the minimum angle. It has a blue track bar with a white triangular slider. To the left of the track bar is the number '8' and the label 'Minimum Angle:'. To the right is a text input field containing the value '120'.

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 (donor - acceptor - heavy atom bonded to the acceptor).

Maximum Distance

8 **Maximum Distance:** 3.9

A slider control for the maximum distance. It has a blue track bar with a white triangular slider. To the left of the track bar is the number '8' and the label 'Maximum Distance:'. To the right is a text input field containing the value '3.9'.

This value determines the maximum distance between the donor and the acceptor atom.

Restrict

8 **Restrictions:** none no H2O no HET no H2O or HET enriched backbone only

A dropdown menu labeled 'Restrictions' with five options: 'none' (selected), 'no H2O', 'no HET', 'no H2O or HET', and 'enriched backbone only'. The 'none' option is highlighted with a green circle.

Specifies optional restrictions for the hydrogen bond calculation. Removing atoms from HET residues and/or water molecules may produce a cleaner result. You can also restrict the computation to the enriched backbone to obtain hydrogen bonds between the backbone carbonyl and amine groups only.

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

A slider interface for setting the minimum angle. It features a small icon of a person, a label "Minimum Angle:", a horizontal slider bar with a triangular thumb, and a numerical input field showing the value "80".

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

A group of three buttons labeled "Considered Atoms": "backbone-enriched" (highlighted with a green circle), "all" (highlighted with a blue circle), and "none" (highlighted with a grey circle).

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 group of three buttons labeled "Write HBonds": "all" (highlighted with a grey circle), "secondary structure" (highlighted with a green circle), and "none" (highlighted with a blue circle).

This port lets you specify which hydrogen bonds to write out.

Maximum HBonds per Atom

A slider interface for setting the maximum number of hydrogen bonds per atom. It features a small icon of a person, a label "Maximum HBonds per Atom:", a horizontal slider bar with a triangular thumb, and a numerical input field showing the value "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, *Vortex*.

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 precomputed 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

Specifies range of time steps for which the density should be computed.

Voxel Size

This value determines the size of a voxel, i.e., its height, width, and depth, of the field storing the probability density.

Grid Dimension

The dimension of the field grid resulting from the specified voxel size.

Alignment

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.

Shape

Shape: atom spheres bond cylinders

Choose the geometry used to represent the molecule. If you have a highly varying trajectory, the recommended shape is *cylinders*.

Radius Options

Radius Options: fixed 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

Atom Radius: 0.2

Radius of the atom spheres.

Bond Radius

Bond Radius: 0.1

Radius of cylinders representing bonds.

Field

Field: Color Field ▾

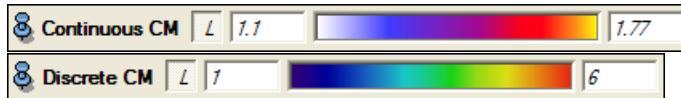
This menu contains two options, *Scalar Field* and *Color Field*. Choose the first if you only want the density. If you choose the second option, in addition to the density, the color information will be stored according to the color scheme (see *Atom Colors* port).

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



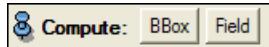
These two colormaps are used to color the atoms and sticks according to the selected color scheme in *Atom Colors*. See the description in the section on *display of molecules* for a detailed explanation.

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 HBondView

The *HBondView* module allows you to display hydrogen bonds using two different representations: dashed lines or tubes.

If the connected molecule does not contain information about hydrogen bonds, the bonds can be computed internally 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.

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

Source of hbonds

	HBond source:	<input type="radio"/> hBonds level	<input type="radio"/> self compute	<input checked="" type="radio"/> automatic
--	----------------------	------------------------------------	------------------------------------	--

Selects the source from where the hydrogen bond information is taken. HBonds level means that the level 'hBonds' in the molecule is used. This only works if such a level has been computed beforehand. Compute means that the hydrogen bonds are computed internally. This makes sense when your molecule is part of a trajectory for which the hydrogens bonds may change between timesteps. With 'automatic' the level 'hBonds' is used if it exists and else the bonds are computed internally.

Type

	Type:	<input type="radio"/> lines	<input checked="" type="radio"/> tubes
--	--------------	-----------------------------	--

Selects the display mode: *lines* or *tubes*.

Line width

	Line width:				<input type="text" value="2"/>
--	--------------------	--	--	--	--------------------------------

Specifies the width of the lines.

Complexity

 Complexity: 0.2

Specifies the complexity of the tubes. Higher complexity will produce more attractive results, but will take longer to draw.

Tube radius

 Tube radius: 0.05

Specifies the radius of the tubes.

Animation

 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.

Coloring

 Color:

 Continuous CM: 

 Discrete CM: 

 Define Color:

For an explanation of the four ports above, see the description in the general section on *display of molecules*.

2.12 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

 Time Steps: from to

Specify the range of molecules used for the computation.

Alignment

 Transformation: align to master 
 Select: 
 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

 Option: write result back to align master

Select this option if you want to iteratively improve the mean molecule.

2.13 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. You can read more about selecting atoms in the documentation of the *Molecule-View* and the *Selection Browser*. The order of the selected atoms is important for angles and dihedrals. It is determined by the sequence that the atoms were selected. If the Measurement module is connected to two molecules, the selected atoms of the first molecule come first.

Connections

Module [required]

The molecule in which you want to take measurements.

Data2 [optional]

You can connect a second molecule. This allows to take intermolecular measurements.

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

 **Selected atoms:** 7[Mol 1] 4[Mol 2] 5[Mol 2]

This port shows the indices and sequence of the atoms that are used for the measurement. If two molecules are connected it also shows the index of the molecule (1 or 2).

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



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



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



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



With the *Clear* button you can deselect all atoms.

Observable



Observable Name



If the molecule is part of a trajectory, then you can create an observable of the measurement, that contains the value of the measurement for each time step. This observable will be added to the trajectory data object. The observable's name needs to be entered in the text field.

2.14 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 which contains the atomic partial charges. MMFF94 partial charges may be computed with the MoleculeEditor. 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. The user should 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 close 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 $kg * \text{Angstroem}^2 * s^{-2} * \text{chargeunit}^{-1}$ for the potential field and $kg * \text{Angstroem} * s^{-2} * \text{chargeunit}^{-1}$ for the electric field, with *chargeunit* being the unit of the charges supplied by the user in the charge attribute.

Ports

Dielectrical Constant



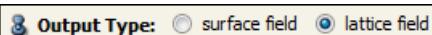
Sets the dielectric constant.

Calculation Type



Allows to switch between the calculation of the electrostatic field which will create a vector field object or potential field which will generate a scalar field object.

Output Type



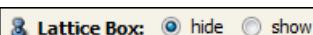
The field may either be generated on a lattice or on a surface. If surface computation is used the surface needs to be supplied as an input object.

Charge Attribute Name



The name of the float attribute that gives the atomic charges for the calculation.

Lattice Box



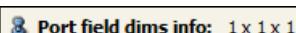
Toggles the dragger box that allows to resize the lattice. This port is only visible if the output type is 'lattice'.

Field Voxel Size



Allows to adjust the voxel size of the lattice. This port is only visible if the output type is 'lattice'.

Field Dims Info



Shows the number of grid points in x, y, and z direction. This port is only visible if the output type is 'lattice'.

2.15 MolOptimizer

This module optimizes the coordinates of a molecule by minimizing its MMFF94 force field energy.

To be able to start the minimization the MMFF parameterization needs to be available as attributes of the molecule. MMFF94 parameters can be computed with the MoleculeEditor or with the molecule tcl command addMMFFParameterization. The computation will create a MolTrajectory containing each minimization timestep. The trajectory will also contain a float observable with the name 'energy' which contains the total MMFF94 energy value (in kJ/mol) for each step.

Connections

Data [required]

The molecule that is optimized.

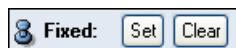
Ports

Method



Method defines the minimization method used. SD (steepest descent) is a first order gradient method following the negative gradient at each step. BFGS (Broyden-Fletcher-Goldfarb-Shanno) is a second order gradient method which works well for small molecules but will fail for large ones because of its n^2 storage requirements. CG (conjugate gradient) is a second order gradient method which has less storage demand than BFGS and is therefore better suited for larger molecules with the tradeoff of slower convergence. All three methods are local minimizers, i.e. they will find the closest local minimum.

Fixed



With the fixed port the user can set the current selection of atoms as fixed during the minimization. Fixed atoms will not move but still influence the atoms surrounding them.

Max Steps



The maximum number of steps during the minimization. The minimization will be terminated after this step except if another termination criteria is reached.

Cutoff



Cutoff is the nonbonded cutoff distance in Angstrom.

Min Diff



The minimum difference of energies (in kJ/mol) between two consecutive steps. When the difference becomes smaller the minimization will be terminated.

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 Option'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]

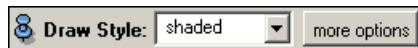
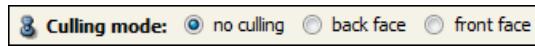
Object of type molecule. Needed if information about the surface is required.

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****Culling mode****Colormap****BaseTrans**

See the description of MolSurfaceView's *base class* for an explanation of the first four ports.

Color Mode

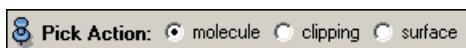
There are three color modes. The first mode uses default colors to color the outside triangles differently than the inside triangles. If one or two molecules are connected to the *MolSurfaceView*, you can choose the second mode to color the molecular surface according to the *coloring* component(s). 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 *Color-Field* 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



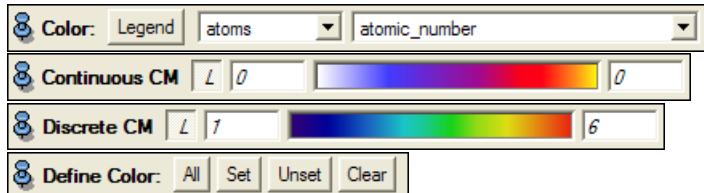
This port provides functionality for selecting (or highlighting) atom surface patches in the viewer window. For more information, see the general section on *highlighting*.

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 MolTrajBundleConverter

This module functions as a converter between the three molecular data types that are used by Amira: *Molecule*, *MolTrajectory*, and *MolTrajectoryBundle*.

It allows an arbitrary number of input objects to be attached as upstream connections and converted into a single output object.

Conversion into a Molecule:

When using *Molecules* as inputs, the output will be the union of all groups. When using *MolTrajectories* or *MolTrajectoryBundles*, each timestep is converted into a *Molecule* and then added, so the output *Molecule* will contain all timesteps at once.

Conversion into a MolTrajectory:

This process requires that all input objects have an identical number of atoms. The topology (i.e. bonds, residues ...) of the first object will be used as the topology of the output object. Of the other objects, only the atomic coordinates are used to add timesteps.

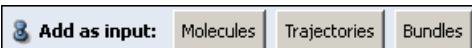
Conversion into a MolTrajectoryBundle:

This conversion is straightforward, since each input *Molecule* and *MolTrajectory* will appear as *MolTrajectory* in the output *MolTrajectoryBundle*.

Connections

input1 [required]

The input data objects. Whenever you add a new input object, a new available input port will be added.

Ports**Add all**

These buttons allow to add all objects (in the object pool) of the specified type as input objects for the conversion.

Result type

Specifies the type of the result object.

2.18 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.

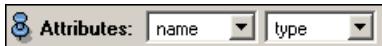
Ports

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 *is visible* 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.
```

2.19 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

Mode: balls sticks balls and sticks

Select one of three main display modes: balls, sticks, or balls and sticks. The default is sticks.

Quality

Quality: fast correct

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: display 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: [triangle up] [triangle down] 0.1 [button]

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: [triangle up] [triangle down] 0.05 [button]

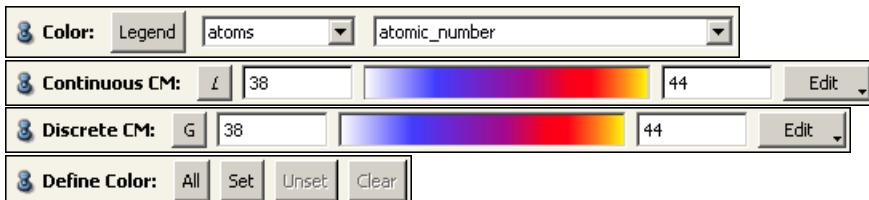
Radius of cylinders representing bonds (only visible in *correct* display mode).

Twist factor



Allows adjustment of the twist of the coil.

Coloring



The four ports above are described in the general section on *visualization of molecules*.

Highlighting



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*.

Highlight Size



This port provides functionality for selecting the intensity of highlight.

Buffer



This port is the front end to a filter that determines the set of atoms being displayed by the *MoleculeView*. 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.

2.20 NoncovalentInteractionGrid

This module generates a grid which contains the noncovalent interaction energy between the input molecule and a probe atom at each grid point. The energy will be computed with the MMFF94 forcefield in kJ/mol. Grid dimensions and atom parameters for the probe atoms can be chosen by the user. The molecule must have MMFF94 parameters assigned. You can add parameters by using the 'Add MMFF Parameterization' menu entry in the MoleculeEditor or with the addMMFFParameterization tcl command of the molecule.

Connections

Data [required]

The molecule for which the interaction energy will be computed.

Ports

Atom Type



Allows to select the MMFF atom type of the probe atom. This atom type determines all van der Waals Parameters.

Atom Charge



For the MMFF94 forcefield the charge of the atom is a function of the atoms formal charge and surrounding atoms by adjusting the charge according to bond charge increments. The probe atom has no bonds so the charge value needs to be adjusted by the user. Whenever the user chooses a new atom type, the value will be set to its formal charge. This parameter has no effect if the Coulomb energy is deactivated.

Energy



Sets which noncovalent energy components are computed.

Lattice Box



Allows to adjust the bounding box of the grid.

Port field voxel size



Allows to adjust the voxel size of the grid

Port field dims info



Shows the dimensions of the field. Large fields will take a long time to compute and computation may fail due to insufficient memory.

2.21 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]

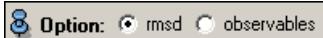
The molecular trajectory for which properties should be displayed.

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



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



Choose the type of property that should be plotted.

Action



Plot the *Observable Type* within a new window.

Plot Option



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



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 *Molecule View* module.

2.22 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*.

2.23 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 *Voltex* 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



The *down factor* influences the width of the Gaussian functions. The smaller the value, the larger the width of the Gaussian function.

Minimal Value



The *minimal value* determines the cut off distance for each Gaussian function. The smaller the value the smoother the density will look.

Scalarfield dimensions

 **Scalarfield dimensions:** 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

 **Cutting distance:** 2.628261

Maximal cutting distance of the Gaussian functions according to the specified *down factor* and *minimal value*.

2.24 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.

AlignSlave [optional]

This connection is only required if you want to search the trajectory according to the *rmsd* value with respect to a reference molecule. If the reference

molecule has a different topology than the time steps of the trajectory, an atom correspondence is needed for the alignment and the rmsd computation. To define this correspondence, one needs a selection in the align master and the align slave, where the align slave should be a time step of the trajectory.

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 precomputed 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



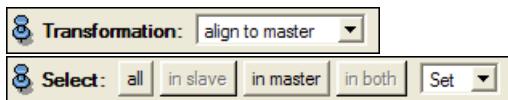
Three options are possible. The first two allow you to search for a single time step with either minimum or maximum value. The last option, *all sorted*, gives you the most flexibility and is therefore the default.

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.

2.25 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



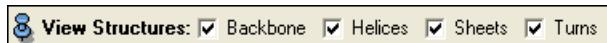
This slider lets you adjust the width of the threads-lines.

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



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



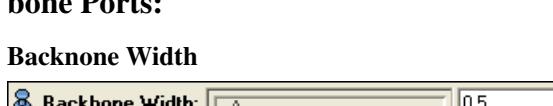
Highlighting



Highlighting Color



Backbone Width



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

Backbone Eccentricity: 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

Backbone Tube Radius: 0.2

This port lets you change the backbone radius.

Coloring

Backbone Color: residues name

Cont. CM: 1 200

Disc. CM: 0 1

Define Color: All Set Unset Clear

The four ports above are described in the section on *visualization of molecules*. Similar ports exist for helices, sheets, and turns.

Helix Ports:

Helix shape

Helix shape: Tubes Cylinders Ribbons

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 similar to the tube representation except that the backbone is shown using a surface whose normals are set perpendicular to the peptide planes.

Helix Radius

Helix Radius: [0.5]

With this port you can adjust the radius of the shown tubes, or cylinders.

Sheet Ports:

SheetShape

Sheet shape: Tubes Arrows

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

Sheet Arrow Width: [1.5]

This port is visible if you view the sheets as arrows. It determines the width of the arrows.

Sheet Arrow Height

Sheet Arrow Height: [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

Sheet Arrow Options: broad head 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



If you view the sheets as tubes, this port lets you configure the radius of the tubes.

Turn Ports:

Turn radius



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> to the value of <transparency> which must be in the interval [0,1].

```
setSecStructTransparency <secIndex> <transparency>
```

Same as above but will affect all residues of the secondary structure <secIx>.

```
setResidueColor <residueIndex> <r> <g> <b>
```

Will set the color of the residue with index <residueIndex> 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>.

```
setSpeculatiry <s>
```

Set the specularity to <s> where 0 is minimal and 1 is maximal specularity.

2.26 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

 **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

 **Distance Cutoff:**

All atoms that are further away than the specified value will not be connected.

Radius



This port specifies the radius of the tube.

Interpolation Method



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



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



You will find the description of the four ports above in the section on *visualization of molecules*.

Highlighting



This port lets you select tube sections with a lasso style selection mechanism or a ROI box.

Highlighting Color



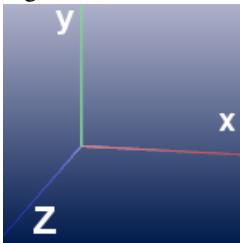
Tube sections that belong to atoms that have been highlighted with the *Selection Browser* will be shown in this color.

3 Virtual Reality Option

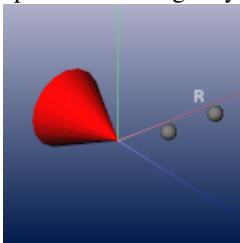
3.1 ShowConfig

This module displays a model of the currently loaded Virtual Reality Option 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.

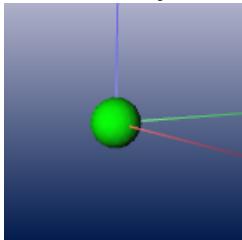
Objects associated with a local coordinate system are drawn with an axis consisting of three colored line segments. The x axis is red, y is green, and z is blue.



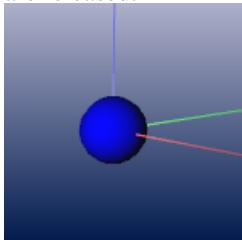
The camera position is marked with a red cone and two small gray spheres, one for each eye. The eye spheres are translated according to the configured eye offset value. One eye sphere is marked with the letter R which means that this is the sphere for the right eye.



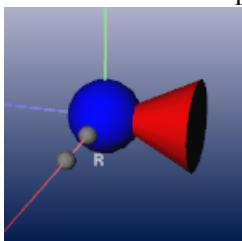
The default object- or scene position is marked with a green sphere.



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 *VRSettings control module*.

3.2 VRSettings

The VRSettings module is the central control module of the *Virtual Reality Option Virtual Reality Extension*. It allows you to activate different screen configurations specified in config files located in `$AMIRA_ROOT/share/config/vr` or `$AMIRA_LOCAL/share/config/vr`. 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 Virtual Reality Option documentation. In the following we assume that you are already familiar with the basic concepts of Virtual Reality Option.

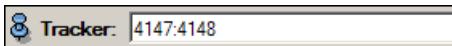
Ports

Config



This port provides a list of all config files found in `$AMIRA_ROOT/share/config/vr` and `$AMIRA_LOCAL/share/config/vr`. 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.

Tracker



Specifies how the connection to the tracking system should be established. In order to connect to a running trackd daemon, 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 connect to a VRPN server, the value of this port should be `vrpn <trackerDeviceName[@host[:port]]> <buttonDeviceName[@host[:port]]>`, with tracker and button device names as specified in the *vrpn.cfg* server configuration file. Specifying a host and a port, where a host can be an IP address or a DNS hostname, is optional. If no host has been specified Amira will try to connect to an VRPN server running on the localhost using the standard VRPN port 3883.

In order to activate a simple *tracker emulator*, you can type in *test* into the text field.

Action

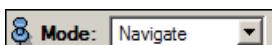


The *connect* button allows you to connect to the tracking system. Once you are connected to the tracking system, the button label changes to *disconnect*. Pressing the button then disconnects from the tracking system.

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 *Calibrating the Tracking System* of the Virtual Reality Option user's guide.

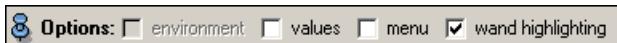
The *pick wand* button allows you to change the offset between the 3D input device and the visual representation of the virtual wand. If you press the button, the wand is displayed half-way between the current eye position and the default object position. You can then pick the virtual wand in the way 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.

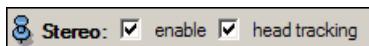
The *wand highlighting* toggle can be turned off to gain some performance. This is particularly useful when the scene contains a large number of interactive objects.

Adjust size



This slider allows you to change the overall scaling of the scene.

Stereo



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 *Flat Screen Configurations* of the Virtual Reality Option user's guide).

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



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



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



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



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 (exponent controlling the blending curve).

`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 Virtual Reality Option config file. The default camera position is used if head tracking is turned off.

`emulateWandButton <0/1> [key-code]`

Activates the wand button emulator. Useful when the wand has no button. [key-code] is an optional decimal value. It refers to the hexadecimal Key enum of the Open Inventor SoKeyboardEvent class. Default key is [ENTER] (code 0xFF0D in OIV, i.e. 65293 in Amira).

4 Very Large Data Option

4.1 ArithmeticRendering

The *Arithmetic* module performs calculations on up to three input data objects according to a user-defined arithmetic expression. This module computes the arithmetic expression on the fly. Slices, Voltex and ROI can be connected to this module to display its result.

Connections

Input a [optional]

The first data set to work with. A value in the user expression.

Input b [optional]

The second data set to work with. B value in the user expression.

Input c [optional]

The third data set to work with. C value in the user expression.

Ports

Expression



This port specifies the mathematical expression used to compute the result.

4.2 CastLattice

This is a computational module that allows you to change the primitive data type of a *LDA* data object. 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, CastLattice is also able to convert a *LDA* data object 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 *LDA* field to be converted.

Ports

Info

 **Info:** <http://amira.zib.de/types#uint8> (0...255) -> <http://amira.zib.de/types#uint8> (0...255)

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

 **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

 **Scaling:** scale offset

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 `inmin ... inmax` into a range `outmin ... outmax`, `SCALE` and `OFFSET` are determined like this:

$$\text{SCALE} = (\text{outmax} - \text{outmin}) / (\text{inmax} - \text{inmin})$$
 and

$$\text{OFFSET} = \text{outmin} / \text{SCALE} - \text{inmin}$$

4.3 ConvertToLargeDiskData

This module allows the conversion of large image data to *LDA file format* (default) or to *LargeDiskData*.

Conversion to LDA format can be applied to original data of the following formats: AmiraMesh, Raw Data, Stacked Slices (stacks of SGI, TIFF, GIF, JPEG, BMP, PNG, JPEG2000, PGX, PNM, and RAS raster files).

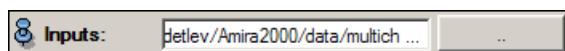
To create a *ConvertToLargeDiskData* module, use the *Create* menu's *Data* submenu. Select the input files with the *Inputs* port, specify the output with the *Output* port. Press the *Apply* button to start the conversion process.

Depending on the input format, you may be asked to supply parameters as described in *TIFF format* or *Raw Data* during the conversion process.

After a successful conversion, a new data object appears in the Pool.

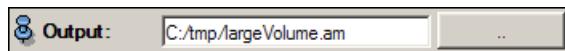
Ports

Inputs



The list of input files.

Output



One output file.

Commands

```
forceAmira311Format {0|1}
```

Provided for compatibility with Amira 3.1 and earlier. Lets you select the output data format.

`0 = LDA file format,`

1 = *LargeDiskData* = Amira 3.1 format.

Note1: An Amira Large Data Pack Very Large Data Option license is not required to use the Amira 3.1 format conversion.

4.4 IsosurfaceRendering (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.

If the isosurface is hidden in one viewer it is also hidden in all viewers (the visibility is global to all viewers)

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



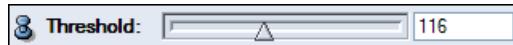
Port to select a colormap.

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.

4.5 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



- **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



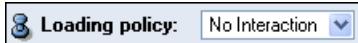
- **Slice Equal Resolution:** If checked, all tiles on a slice will be taken from the same resolution level.
- **Move Low Resolution:** If checked, LDA will use a lower texturing resolution when the user is interacting with the scene.
- **Ignore Fully Transparent Tiles:** If checked, LDA will not load fully transparent tiles (i.e., if all voxels within the tile have an alpha value of 0).

Num loading threads



Number of separate threads used for tile loading. Using more threads can increase performance on multiprocessor machines.

Loading policy



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



If set, LDA will only load tiles from the specified resolution level (specified using the *Fixed resolution level* port).

Fixed resolution level

Fixed resolution level:

Resolution level for the fixed resolution mode.

Main memory:

Amount in MB

Amount in MB: 128

Specifies how much main memory (in MB) LDA can use to cache data.

Notification rate

Notification rate: 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

Amount in MB: 64

Specifies how much texture memory (in MB) LDA can use for the volume (3D textures).

Texture load rate

Texture load rate: 1

Specifies the maximum number of textures to load into texture memory between two frames.

Texture memory (slices):

Number of textures

Number of textures: 256

Specifies how much texture memory is allowed in MB for slices (2D textures).

Texture load rate

Texture load rate: 64

Specifies the maximum number of textures to load into texture memory between two frames.

4.6 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.

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.

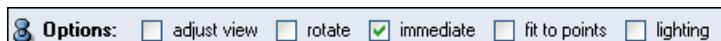
Ports

Orientation



This port provides three buttons for resetting the slice orientation. *Axial/xy* slices are perpendicular to the z-axis, *coronal/xz* slices are perpendicular to the y-axis, and *sagittal/xz* 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. Please note also that *adjust view* is enabled by default when the module is the first planar module in the pool.

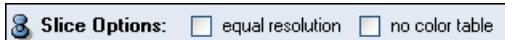
With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off.

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.

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

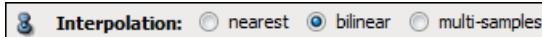
If the *lighting* toggle is set, the slice is lit.

Slice Options



If *equal resolution* is set (unset by default), all ortho slices and oblique slices are drawn using the same resolution. When *no color table* is not set (default), when editing the colormap, the rendering speed is faster and the color interpolation is better. This option uses shaders so it may be very slow on old graphic boards.

Interpolation



Specifies interpolation type (nearest value or linear interpolation). *Multi samples* computes a weighted value from 12 neighbors.

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 when using *linear mapping* or that only fully transparent parts are transparent while others are fully opaque when using *colormap mapping*.. *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



This port is displayed if embossing is enabled. 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 displayed 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.

4.7 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. The default colormap is a linear grey ramp, so the two threshold values determine which data values are mapped to black and which are mapped to white. Alternatively, any other colormap may also 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 colormap can 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 [required]

Colormap used to map scalar data to colors, by default a linear grey ramp. See also *Colormap*.

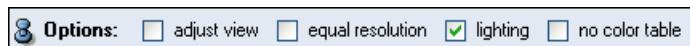
Ports

Orientation



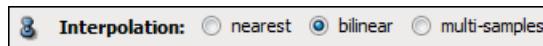
This port provides three buttons for resetting the slice orientation. *Axial/xy* slices are perpendicular to the z-axis, *coronal/xz* slices are perpendicular to the y-axis, and *sagittal/yz* slices are perpendicular to the x-axis.

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. When *lighting* is set (set by default), the display appears lit. When *no color table* is not set (default), when editing the colormap, the rendering speed is faster and the color interpolation is better. This option uses shaders so it may be very slow on old graphic boards. Please note also that *adjust view* is enabled by default when the module is the first planar module in the pool.

Interpolation



Controls whether interpolation is *nearest* or *bilinear*. *Multi samples* computes a weighted value from 12 neighbors.

Colormap



Choose the colormap which is used to map the data values. The two threshold sliders allow you to restrict the range of visible data. For example, when using

a linear grey ramp, all values below the lower bound are mapped to black, while 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*).

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 when using *linear mapping* or that only fully transparent parts are transparent while others are fully opaque when using *colormap mapping*. *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



This port is displayed if embossing is enabled. 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 displayed 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

```
createImage <image base name>
```

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

4.8 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

	Controlling:	<input checked="" type="radio"/> ROI box	<input type="radio"/> Subvolume box
--	---------------------	--	-------------------------------------

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:**

Minimum x-, y-, and z-coordinates of the region-of-interest.

Maximum

 **Maximum:**

Maximum x-, y-, and z-coordinates of the region-of-interest.

Options

 **Options:** Show box

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

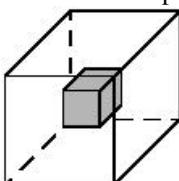
 **Cropping:**

The crop box is defined by three sets of parallel planes that define three slabs:

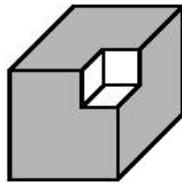
- The xmin and xmax planes define the X slab.
- The ymin and ymax planes define the Y slab.
- The zmin and zmax planes define the Z slab.

Four cropping options are available: *Subvolume*, *Exclusion*, *Cross*, and *Fence*.

Subvolume Displays voxels inside the ROI.

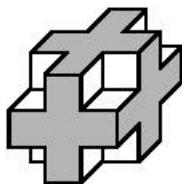


Exclusion Displays voxels outside the ROI.



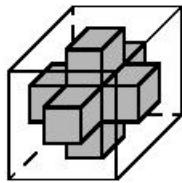
Fence Displays voxels between:

- (X_{\min}, X_{\max}) , or
- (Y_{\min}, Y_{\max}) , or
- (Z_{\min}, Z_{\max}) .



Cross Displays voxels between:

- (X_{\min}, X_{\max}) and (Y_{\min}, Y_{\max}) , or
- (X_{\min}, X_{\max}) and (Z_{\min}, Z_{\max}) , or
- (Y_{\min}, Y_{\max}) and (Z_{\min}, Z_{\max}) .



(Figures courtesy Real Time Visualization)

Action



Size



Size of data in the region of interest in MBytes

4.9 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.

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]

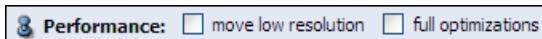
Connection to a module providing a region-of-interest, like *SelectRoi*. If such a module is connected, only the selected part of the volume will be displayed.

Colormap [optional]

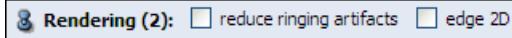
Colormap used to visualize the data.

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.

Performance

Move low resolution enables rendering at a lower resolution when moving, resulting in a smoother animation. *Full optimizations* discards completely transparent volume area and enables some more advanced fragment discarding (early-z...)

Rendering

Illumination activate port *Illumination*.

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. 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. 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 true.

Hi-quality decrease slicing effect by preintegrated rendering.

Reduce ringing artifacts decrease ringing artifacts by adding some jittering.

Edge 2D enables 2D edges detection.

Composition



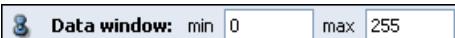
Specifies color composition type. *Alpha* composes the slices by blending the R, G, B components for each pixel based on their alpha values. *Sum* adds slice colors along the viewing axis. (Not available on VolumePro hardware.) *Max* draws the maximum intensity for each pixel drawn along the viewing axis. *Min* draws the minimum intensity for each pixel drawn along the viewing axis.

Interpolation



Specifies volume data interpolation type (*nearest* value or *linear* interpolation or *cubic* 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 colormap's alpha value is used for both absorption and emission. In *LumAlpha* mode, the colormap's 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.

Tile refinement



The refinement factor used when projection is active.

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.

Illumination



Lighting indicates if lighting is required. Default is false. Note that activating or deactivating lighting when using 2D/3D texture rendering might force the textures to be recreated, which may be slow. *limitations:* Underlying VolumeViz rendering engine does handle only a single light. So only the first light (ie. the

headlight if enabled or the first created light if not) will be used. More over only directional lights are supported.

Edge enhancement fakes lighting by detecting edge.

Boudary opacity fakes lighting by darkening volume boundaries.

Illumination options



Illumination quality goes from *low* to *high*, it is used by illumination rendering (lighting, edge enhancement and boundary opacity). Computational cost increase with chosen quality.

Bright lighting enables brighter lighting model (enable only when using *lighting*).

Edge coloring



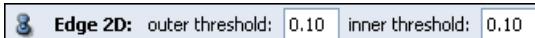
Gradient threshold controls whether or not a voxel is darken. Areas facing the camera will have an unmodified color, whereas areas where the normal is more perpendicular to the view direction will tend towards black. Lower gradient threshold gives stronger edges.

Boundary opacity



Boundary opacity increases opacity depending on the length of the gradient vector. Areas with large gradient changes (greater than *threshold*) will have their opacity increased according to given *intensity*.

Edge 2D



Apply an image space filter to the renderer volume in order to detect edges, which will be highlighted.

Commands

```
verbose {0|1}
```

If value is 1, additional messages for debugging are printed.

5 Mesh Option

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 and corresponding points on the reference surface. This is often referred to as Procrustes analysis. 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.

ROI [optional]

Connection to a module defining a region of interest.

Weights [optional]

This port can be connected to a field of type *HxSurfaceScalarField* defined on

the nodes (vertices) of the surface to be transformed. The values will be evaluated during iterative optimization so that vertices with a high weight have a greater influence on the alignment than those with a low weight.

Ports

Options

Options: iterate use correspondence use weights

Iterate: If checked [default], the closest point algorithm is repeated until either criterion defined in port *Stop* is reached. If unchecked, the closest point algorithm is performed only once.

Use correspondence: This option is available only if both reference and surface to be transformed have the same number of vertices. If checked, the algorithm assumes that the vertex indices on both surfaces correspond.

Use weights: This option is available only if a surface scalar field is connected to port *Weights*. Its values will be evaluated during iterative optimization so that vertices with a high weight have a greater influence on the alignment than those with a low weight. Points can be ignored by using a weight with value 0.

Trafo

 **Trafo:** rigid rigid + uniform scale affine

Changes the number of degrees of freedom to be optimized. *AlignSurfaces* performs linear alignment between two surface meshes x and y. The expression $|y - (Ax + b)|$ is minimized over all transformations (A,b) where A is a matrix and b the translation vector.

- **rigid:** A is a rotation matrix only
- **rigid + uniform scale:** A contains rotations and a uniform scale factor
- **affine:** A is an arbitrary matrix (includes non-uniform scaling)

Stop

 **Stop:** relative RMS max iter

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 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.

Connections

Data [required]

The tetrahedral grid to be visualized.

Ports

Bound.cond.id



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.

5.3 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 *Vortex*.

Connections

ComplexField [required]

Complex scalar field for which a real quantity should be computed. Only float and double are supported as primitive types. Consider using *CastField* to convert your data first.

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 π .

5.4 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.

Notice that ConePlot is only provided for backwards compatibility since it has been replaced by the *ParticlePlot* module.

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.

$$y(t) = \cos(\text{phase})\text{Re}(x(t)) + \sin(\text{phase})\text{Imag}(x(t)) \quad (5.1)$$

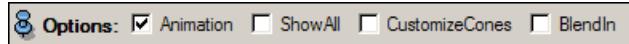
A value of 0 degree corresponds to the real part, while a value of 180 degrees corresponds to the imaginary part. The display can be animated with respect to the phase by the *Animate* port. This way polarization properties of the field can be revealed or wave phenomena become visible.

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.

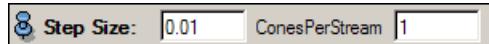
Animation: If this option is set, then two additional ports are displayed. The *Time* port allows you to set a time frame (from $-m > 0$ to n frames). The *Step size* port allows you to set the time between successive time steps. Currently we use an Euler integration. The smaller the step size, the more 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...50 is interpreted as 50 time steps for the forward integration, or -30...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.

5.5 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

SelectBy



Currently there are two selection modes in order to choose contours:

Material: Choose contours with respect to bordering materials. *Index*: Choose contours according to their index.

Materials



Display contours adjacent to all specified materials.

Index



Choose index of contour to display. If -1 is chosen, all contours will be shown.

5.6 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.

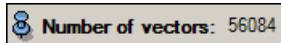
Connections

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).

5.7 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 field's 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]

Either an optional scalar field which can be used to control the distribution of seed points, or a vertex set that is used as 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

Color



This port is visible if no color field is connected to the module and it allows to change the color of the ISLs. Clicking on the icon opens a Color Dialog where the color can be set.

Colormap



The optional colormap. This port is visible only if a color field is connected to the module.

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



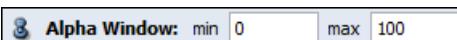
Base opacity factor of the lines. A value of 1 produces completely opaque lines, while 0 results in fully transparent lines.

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.

Seed



This port defines the type of the random number generator that distributes the seed point. The generation can be random or constant.

Options

	Options:	<input checked="" type="checkbox"/> fade	<input checked="" type="checkbox"/> lighting	<input type="checkbox"/> animate	<input type="checkbox"/> DEC
--	-----------------	--	--	----------------------------------	------------------------------

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`.

DEC: Activates directionally encoded colors. The colors are generated from the tangent directions of the lines. Directions x, y, and z are mapped to colors red, green, and blue.

Seed Box

	Seed Box:	<input type="radio"/> XformBox	<input type="radio"/> TabBox	<input checked="" type="radio"/> Hide
--	------------------	--------------------------------	------------------------------	---------------------------------------

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

	Distribute:	homogeneous	
--	--------------------	-------------	--

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.

Box

	Display box:	<input type="checkbox"/> Active	<input type="checkbox"/> Visible	Reset
--	---------------------	---------------------------------	----------------------------------	--------------

This port controls the projected seed box of the lines. The seed box is composed of a black shape representation and green little boxes that handle the manipulation. Pressing reset resizes the box to the bounding box of the connected data.

When using projection (see HxProjection), only the lower left corner of the green boxes is used to compute the projected shape. Other boxes won't be synchronized with the box geometry.

Box center



Box center:	x	0.000000	y	0.000000	z	0.000000
-------------	---	----------	---	----------	---	----------

This port controls the center of the seed box. When using projection, it can happen that the value entered be erroneous. In that case, the old value is displayed again.

Box center



Box scale:	x	0.1	y	0.1	z	0.1
------------	---	-----	---	-----	---	-----

This port controls the scale factor of the seed box. When using projection, it can happen that the value entered be erroneous. In that case, the old value is displayed again.

Commands

`createLineSet`

Can be used to create a lineset object from the currently displayed lines.

5.8 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.

5.9 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.

5.10 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.

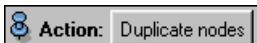
Connections

Data [required]

A tetrahedral grid without duplicated nodes.

Ports

Action



Button *duplicate nodes* causes a new output grid with duplicated nodes to be computed.

5.11 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

Output

	Output:	<input checked="" type="checkbox"/> evec1	<input checked="" type="checkbox"/> evec2	<input checked="" type="checkbox"/> evec3	<input checked="" type="checkbox"/> evals
--	----------------	---	---	---	---

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

	MaskExpr:	A>0
--	------------------	-----

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.

5.12 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 in 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 for resetting the slice orientation. *Axial/xy* slices are perpendicular to the z-axis, *coronal/xz* slices are perpendicular to the y-axis, and *sagittal/yz* 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.

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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 *PortRadioButton*.

- **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.

5.13 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 for resetting the slice orientation. *Axial*/xy slices are perpendicular to the z-axis, *coronal*/xz slices are perpendicular to the y-axis, and *sagittal*/yz 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 you release the mouse button.

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

Translate



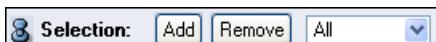
This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

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.

5.14 Gradient

The *Gradient* module computes the gradient of a field defined on a regular (i.e. uniform, rectilinear, or curvilinear) grid or on a tetrahedral grid . The output is a vector or tensor field defined on the same grid. The orientation of the gradient (vector) depends on the setting of port *Output*. If *Force* is selected the negative gradient vector is computed.

$$\text{grad}F = \pm \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)$$

Press the *Apply* button to start the computation.

Connections

Data [required]

Field defined on a regular grid or a vertex based tetrahedral grid.

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.

5.15 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.

Culling mode



Please refer to the *ViewBase* documentation.

Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

Buffer



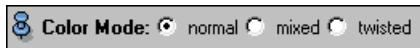
The Buffer buttons give you some control of the internal face buffer of the Grid-Boundary module. Only the faces present in the buffer are displayed according to the current drawing style. The *Add* button adds the highlighted faces to the buffer. The *Remove* button removes highlighted faces from the 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) 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, also see *PortButtonList*.

Color Mode



Here you may choose among several color modes. For details see the module *SurfaceView*.

Commands

```
setSelectAllNew {0|1}
```

If set to 1 all tetrahedra are selected when a new data set has been connected to the *Data port*. Useful in a network together with *TimeSeriesControl*.

5.16 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.

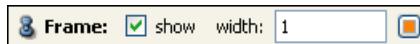
Connections

Data [required]

The tetrahedral grid to be visualized.

Ports

Frame



This port is used to display or hide the frame, set the frame width and color.

Orientation



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

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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.

Export



Creates a *Surface* object containing the set of currently visible triangles.

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.

5.17 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.

The *fit to points* toggle lets you reset the plane by clicking on at least 3 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 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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*.

5.18 GridToSurface

This computational module parses the connected *TetraGrid* object type and converts it into *Surface* type when you press the Apply button. A green icon representing the generated surface should appear in the Pool.

Connections

Data [required]

Connect a data object of type *TetraGrid* object type to be converted.

Ports

Action



By pressing the button "Create Surface" the surface is created in the Pool. Once the new object has been created, it is possible to set the boundary IDs back into the input data source by clicking the button "Set boundary ids of input".

5.19 GridView (Block Structured)

The *GridView* module displays a *block-structured grid* or parts of it.

Connections

Data [required]

The *block-structured grid* to be shown.

ColorField [optional]

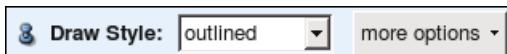
Arbitrary scalar field which is mapped onto the grid volume using pseudo-coloring.

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, the description can be found there.

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 faces (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*.

Options



Toggle viewing the boundary of the blocks.

5.20 GridVolume (Hexahedra)

The *GridVolume* 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.

Texture [optional]

The texture field must be an HxUniformColorField3 2D slice. When connected,

the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

Ports

Draw Style



Please refer to the *GridVolume* documentation.

Colormap



Please refer to the *GridVolume* documentation.

Texture wrap



Please refer to the *GridVolume* documentation.

Culling mode



Please refer to the *ViewBase* documentation.

Buffer



Please refer to the *GridVolume* documentation.

Materials



Please refer to the *GridVolume* documentation.

Color mode



Please refer to the *GridVolume* documentation.

5.21 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.

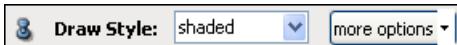
Colormap [optional]

The colormap is used for pseudo-coloring the grid volume.

Texture [optional]

The texture field must be an *HxUniformColorField3* 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

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



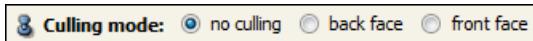
This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

Texture wrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

Culling mode



Please refer to the *ViewBase documentation*.

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 data set 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.

5.22 GridVolume (Tetrahedra)

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 must be in 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.

Texture [optional]

The texture field must be an HxUniformColorField3 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis.

A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

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*.

Texture wrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

Buffer



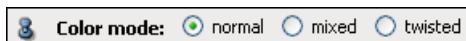
This port lets you *add* and *remove* highlighted triangles (being displayed in red wireframe) to an internal buffer. For a further description and for the functionality of each of the port buttons see *ViewBase*.

Materials



This port provides 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`.

`setSelectAllNew {0|1}`

If set to 1 all tetrahedra are selected when a new data set has been connected to the *Data port*. Useful in a network together with *TimeSeriesControl*.

5.23 HexToTet

The *HexToTet* module converts a hexahedral grid to an equivalent tetrahedral grid. Each hexahedron is converted to between six and twelve tetrahedra which have the same material ID as the "parent" hexahedron. The new tetrahedral grid has the same points as the source hexahedral grid; for some hexahedrons, an additional

inner point is added to ensure that all tetras exactly match each other at their faces. The module also converts the data objects connected to the hexahedral grid.

Note : Boundary conditions are not converted.

Press the *Apply* button to start the computation.

Connections

Data [required]

Unstructured hexahedral grid to be converted.

ROI [optional]

Optional region to restrict the conversion.

Ports

Options



Toggles whether the data objects connected to the hexahedral grid are also converted or not.

5.24 HexaQuality

The *HexaQuality* module computes different quality measures for hexahedral grids. For this purpose it can be attached to a *HexaGrid* object as well as a *GridVolume* module. Quality is calculated for all hexahedra in the first case, and for all hexahedra selected by the *GridVolume* module in the second case. The output is a scalar field defined on the hexahedral grid and equal to 0 on the non-selected hexahedra.

The user can create a histogram of the quality for the hexahedral grid. By default, the histogram is shown with a logarithmic scale to direct the focus on the hexahedra with worst quality.

Press the *Apply* button to start the computation.

Connections

Data

The tetrahedral grid.

Grid volume

A *GridVolume* module that selects the hexhedra for which the quality is calculated.

Either the **Data** or **Grid volume** connection is required. The other one then becomes optional.

Ports

Quality Measure



The option menu lets you choose between different quality measures:

- **Squish:** the cell squish measures the angular divergence between the vector $\vec{d}_{cc \rightarrow fc_i}$ pointing from the cell centroid to each face centroid and the face normal vector \vec{n} pointing out of the cell:

$$\max_i \left[1 - \frac{\vec{n} \cdot \vec{d}_{cc \rightarrow fc_i}}{|\vec{n}| |\vec{d}_{cc \rightarrow fc_i}|} \right]$$

In the best case the cell squish is equal to 0, whereas it is close to 1 in the worst case where the cell is degenerate.

- **Equiangle skew:** the equiangle skew measures the cell quality on the basis of its angles and is defined as follows:

$$\max \left[\frac{\theta_{max} - \theta_e}{180 - \theta_e}, \frac{\theta_e - \theta_{min}}{\theta_e} \right]$$

where θ_{max} is the largest cell angle, θ_{min} is the smallest cell angle and θ_e is the equiangular cell angle (equal to 90 here, in the hexahedral grid case). In the best case the cell is equiangular and the skew is equal to 0, whereas it is close to 1 in the worst case where the cell is degenerate. A cell with an equiangle skew lower than 0.5 is considered as good and as acceptable if the skew is lower than 0.75.

- **Maximum angle:** largest cell angle.
- **Minimum angle:** smallest cell angle.
- **Aspect ratio:** the aspect ratio measures the cell stretching and is defined as the ratio between the maximum cell centroid to face centroid distance and the minimum node to node distance. In the best case the ratio is equal to 0.5 and an aspect ratio smaller than 5 is generally considered as acceptable.

Samples



This slider lets you select the number of samples for the histogram. The default values should be adequate.

Histogram



If you select this toggle, a plot window will appear, once you've pressed *Apply*, showing the histogram of qualities for all selected hexahedra. If no hexahedra have been selected, the plot window will not be shown.

5.25 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 the two input data sets must have the same number of points. 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 preceding 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



Info: no input objects

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



Options: interpolate mesh too physical time

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.

5.26 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.

Texture [optional]

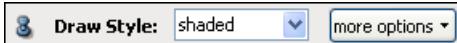
The texture field must be an HxUniformColorField3 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

ROI [optional]

Connection to a module defining a region of interest.

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



To use this port, you must first create a dragger. To do so, type the command `Isosurface showBox`. Then the buffer port must be enabled explicitly with the command `Isosurface buffer show`. 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.

TextureWrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

Culling mode



Please refer to the [ViewBase documentation](#).

Average



Averages cell based fields when selected.

5.27 Isosurface (Polyhedra)

This module computes an isosurface for a scalar field defined on a three-dimensional polyhedral 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. Data must be stored per nodes.

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.

Texture [optional]

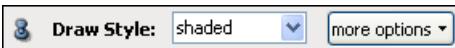
The texture field must be an HxUniformColorField3 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

ROI [optional]

Connection to a module defining a region of interest.

Ports

Draw Style



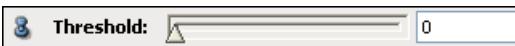
The draw style port is inherited from class *ViewBase*. For a description of this port see *ViewBase*.

Colormap



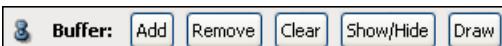
If a colormap is connected to the isosurface module, this colormap will be shown here. If no colormap is connected to the module, the port's default color is used. To change this color, click into the color bar with the left mouse button. This will bring up the color selection dialog. To connect the 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.

Buffer



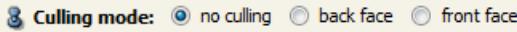
To use this port, you must first create a dragger. To do so, type the command `Isosurface showBox`. Then the buffer port must be enabled explicitly with the command `Isosurface buffer show`. For a detailed description see the *ViewBase* documentation.

TextureWrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

Culling mode



Please refer to the *ViewBase documentation*.

5.28 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)*.

Texture [optional]

The texture field must be an HxUniformColorField3 2D slice. When connected, the texture is mapped onto the surface geometry along the texture normal axis. A transformation can be applied to the data to change the texture projection axis and texture coordinates scale. Note that texture projection override pseudo-color mode.

PointProbe [optional]

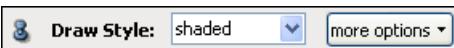
If this port is connected to a *PointProbe* module as isovalue the value at a certain point within the scalar field will be chosen. For details see *PointProbe*.

ROI [optional]

Connection to a module defining a region of interest.

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



To use this port, you must first create a dragger. To do so, type the command `Isosurface showBox`. Then the buffer port must be enabled explicitly with the command `Isosurface buffer show`. For a detailed description see the *ViewBase* documentation.

TextureWrap



Wrap mode used for the texture projection on the surface. There are two wrap modes: *Repeat*: The texture is repeated outside its 0-1 texture coordinate range. *Clamp*: Clamps texture coordinates to lie within 0-1 range. This port is hidden if there is nothing connected to the *Texture* connection port.

Culling mode



Please refer to the *ViewBase* documentation.

5.29 Lambda2

The *Lambda2* module computes the second largest eigenvalue of the matrix $L = S^2 + \Omega^2$ where S and Ω 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 regular grid (*RegVectorField3*). The vector components must be floats.

5.30 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 *Vertex View*.

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.

5.31 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]

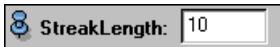
The surface where the streaks will be put on.

VectorField [required]

The vector field from which the streaks will be computed. May be either a surface vector field or a 3D vector field.

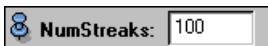
Ports

StreakLength



Relative length of the field line segments. The length is scaled by the length of the diagonal of the surface's bounding box.

NumStreaks



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.

5.32 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.

5.33 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



Number of vectors:

Shows number of input vectors and the vector dimension for computing the magnitude field.

Mode



Mode: vector normal tangent

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.

5.34 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:

$$(t - 1) * (\text{expression1}) + t * (\text{expression2}). \quad (5.2)$$

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]

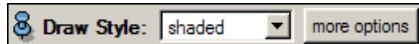
An additional variable which can be used in all the expression fields. It can be used to generate animations.

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*.

Culling mode



Please refer to the *ViewBase documentation*.

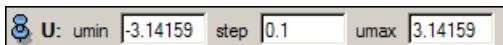
Pre-defined surfaces



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.

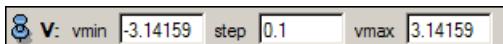
As a last entry you will find a Random generator which will generate random entries for the x, y and z ports, i.e. a random mapping. Currently there is a maximum depth of 10 for the randomly generated expressions.

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

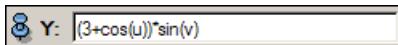


This port works the same way as the 5.34 port for the *v* parameter of the initial mesh.

X



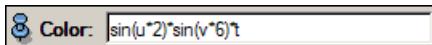
Enter an expression to be evaluated in order to describe the *x*-position of a mesh in 3D space. Use the 5.34 port to see some useful settings for this port.

Y

Enter an expression to be evaluated in order to describe the y-position of a mesh in 3D space. Use the 5.34 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 5.34 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.

5.35 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. 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



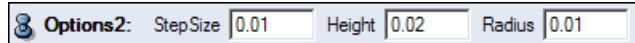
A colormap which specifies the pseudo-coloring used. Please note that the maximum magnitude detected is printed out by the module after each *Apply* button action.

Options



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



This port specifies the initial step size of the Runge-Kutta solver and the height

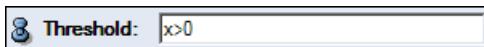
and radius of the cones. If a shape object is attached to the module, only the height setting will be used to specify its isotropic scaling.

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



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 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.

5.36 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.

ROI [optional]

Connection to a module providing a region-of-interest, like *SelectRoi*.

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*.

ColorField2 [optional]

A second scalar field which may be used for pseudo-coloring with two color fields.

Colormap2 [optional]

Colormap used for pseudo-coloring with a combination of two color fields. If no colormap is connected the default color of the colormap port will be used.

Ports

Orientation



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

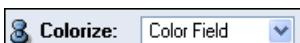
The *fit to points* toggle lets you reset the plane by clicking on at least 3 different points in the scene. After enabling this toggle, you should switch to interaction mode by pressing the ESC key inside the viewer. Then click 3 times at different points on any geometry in the scene. The plane then will be automatically adjusted to match these points. The virtual trackball - visible when *rotate* toggle is active - will be moved to the center of the picked points. After 3 points have been picked, this toggle is automatically unchecked, unless you pressed the shift key. Shift-clicking allows you to select more than 3 points. In this case the plane that best fits the selected points will be computed.

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. There is a second mode 2 *color fields multiplied*. This mode makes it possible to visualize two color fields at the same time. The color fields have to be attached to the ports *ColorField* and *ColorField2*. The colors of each field are multiplied and then applied to the final image.

Colormap



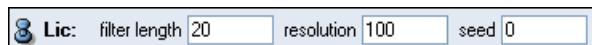
Port to select a colormap.

Colormap2



Port to select the second colormap that is applied to the second color field.

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

Hide undefined values



If checked, locations where there is no data value or where extracted data values are equals to the undefined value are displayed as transparent pixels.

If a *TetraComplexVectorField3*, a *TetraVectorField3*, an *EdgeElemVectorField3* or an *EdgeElemComplexVectorField3* is connected to the module, only locations where there is no data value will be discarded.

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.

5.37 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]

Vector field defined on a uniform grid (*UniformVectorField3*) and that describes the space distortion. The vector components must be floats.

5.38 RemeshSurface

This module remeshes triangular surfaces to improve triangle quality. The module implements two approaches for surface remeshing. The first approach uses explicit regularization of the triangles around a vertex and, therefore, generates triangular surfaces with *high regularity*. A highly regular mesh is a mesh for which most of its vertices have 6 neighbors.

The second approach is based on *Lloyd relaxation* and does not explicitly regularize the vertex connectivity. This approach attempts an *isotropic vertex placement* in order to achieve a high triangle quality.

The more general approach is that of an *isotropic vertex placement*, why it is set the default. However, for smooth surfaces the *high regularity* objective might also produce very good results. At the end of this module's documentation you see the remeshing results for both objectives compared to the original surface.

Note that the algorithm is both time and memory consuming. For example, when remeshing a surface with 1,300,000 triangles, about 2GB of memory will be consumed. The time varies according to the size it will be remeshed to. For example, when remeshing the surface to 50 percent of its original size, it will take about 15 minutes.

Remeshing is started by pressing the *Apply* button.

Connections

Data [required]

Input surface to be remeshed.

DensityField [optional]

Surface scalar field used for modulating locally vertex density. If no field is connected the curvature is used for weighting the density. See also port *Density contrast*.

SurfacePathSet1 [optional]

A *surface path* set that can be considered while remeshing to preserve certain features. Please also see the *Surface path options* port. Note that more than one surface path set can be connected to the module. If the first surface path set is connected, a new connection will be created.

Ports

Advanced options

	Advanced options:	<input checked="" type="checkbox"/> show
--	--------------------------	--

By default, advanced options are hidden. They will be displayed if the *show* toggle is checked.

Objective

	Objective:	Best isotropic vertex placement	
--	-------------------	---------------------------------	--

This port determines the objective of the remeshing:

High regularity: the remesher tries to generate a mesh such that for most of the vertices the number of neighboring vertices is 6.

Best isotropic vertex placement: the remesher attempts to generate a mesh with vertices placed isotropically across the surface. Since vertex placement is modulated by either the curvature or a density field, isotropy of vertex placement will only hold with respect to the modulated vertex distribution.

Triangle area opt.

	Triangle area opt:	nPasses	5	nAreaSteps	3	nEdgeFlips	3
--	---------------------------	---------	---	------------	---	------------	---

This port lets the user decide how much effort is put into the optimization of the triangle area. The desired triangle area in a certain surface region is a function

of the given density. If no *density field* is given, the curvature field will be used instead, which is computed internally. The goal is to optimize the correct triangle area distribution. If no weighting is used, the areas of all triangles should be similar. The first parameter specifies the number of passes used for optimizing the triangle area. The *nAreaSteps* parameter specifies the number of steps per pass that are used for optimizing the triangle area by moving the vertices. The last parameter, *nEdgeFlips* specifies the number of edge flips performed per pass for optimization. The default parameters should be suitable for most applications.

Vertex valence opt.

	Vertex valence opt.:	nPasses	<input type="text" value="1"/>
--	-----------------------------	----------------	--------------------------------

This parameter allows specifying the number of passes used for improving the vertex valence, i.e. the regularity. This option is only visible, if the objective is *high regularity*. A mesh is considered as being regular, if the number of neighbors of each vertex in the mesh is 6. So, the higher the *nPasses* value, the more the remesher will try to generate a regular mesh.

Triangle quality opt.

	Triangle quality opt.:	nPasses	<input type="text" value="10"/>
--	-------------------------------	----------------	---------------------------------

This parameter specifies the number of passes being used for triangle quality optimization. Here, the objective is to obtain triangles that are as equilateral as possible, i.e. all triangle angles should be similar. This option is only visible, if the objective is *high regularity*.

Lloyd relaxation

	Lloyd relaxation:	nPasses	<input type="text" value="40"/>
--	--------------------------	----------------	---------------------------------

This parameter becomes visible if the objective is *best isotropic vertex placement*. It determines the number of passes used for relaxing the points. The default should be suitable in most cases.

Defaults

	Defaults:	<input type="button" value="Set"/>
--	------------------	------------------------------------

With this button, the user can quickly reset the parameters to the default.

Desired size

	Desired size:	#vert.	<input type="text" value="35309"/>	#tris	<input type="text" value="70618"/>	%	<input type="text" value="50"/>
--	----------------------	--------	------------------------------------	-------	------------------------------------	---	---------------------------------

The values of this port determine the number of vertices and triangles in the final surface. Note, that the actual number of vertices and triangles might be slightly different from this number. The three values in this port influence each other. Setting one of the values will result in a modification of the other values. In regular surfaces, the number of triangles is twice as large as the number of vertices. Hence, setting the number of triangles will result in half the number of vertices. However, it is inherent to the algorithm to consider the number of vertices and not the number of triangles. Hence, the resulting mesh will be closer to the specification of the number of vertices than to that of the number of triangles. The last value is provided for convenience only. It allows setting the percentage of the original mesh size.

Error thresholds

	Error thresholds:	smoothness	<input type="text" value="0"/>	distance	<input type="text" value="0"/>
--	--------------------------	------------	--------------------------------	----------	--------------------------------

Two error thresholds can be set. Both thresholds are angle thresholds and have a range between -1 and 1. If the error threshold is 1, no changes are allowed, because each remeshing will violate this threshold. If the value is -1, no error will be taken into account. Since the thresholds are given as the cosines of angle thresholds, their range is -1 to 1.

The *smoothness* error measures the difference between the vertex normals of neighboring vertices. Hence, a vertex can only be moved if the smoothness criterion is preserved. As a result, regions of the original surface that are not smooth will usually stay as they are.

The *distance* error measures the distance between the original and the modified surface in terms of normal distance. Hence, a remeshing step can only get accepted, if the distance between the normals of the original and the remeshed surface is smaller than the given threshold.

Density contrast

	Density contrast:	<input type="text" value="0"/>
--	--------------------------	--------------------------------

The density contrast determines the influence of the density field or the surface curvature on the triangle area in certain regions of the surface. It is used as an exponent of the values of the density field, so be careful when choosing a value. A value between 0 and 2 might be appropriate.

Density range

	Density range:	min	<input type="text" value="0.00200396"/>	max	<input type="text" value="8911e+07"/>
--	-----------------------	-----	---	-----	---------------------------------------

This is the density range to be taken into account. Values below and above will be clipped.

Interpolate orig. surface

	Interpolate orig. surface:	<input type="radio"/> smoothly	<input checked="" type="radio"/> none
--	-----------------------------------	--------------------------------	---------------------------------------

When the first option, *smoothly*, is selected, the original surface will be smoothed internally, and this smoothed surface will be used to move the vertices. If the second option is chosen, no interpolation is done and the vertices of the remeshed surface will be exactly on the original piecewise linear surface.

Remesh options (1)

	Remesh options (1):	<input type="checkbox"/> fix contours	<input checked="" type="checkbox"/> contract boundary edges
--	----------------------------	---------------------------------------	---

Contours can either be given by surface paths or implicitly at non-manifold edges. With the first option, these contours can be fixed, which means that no changes are allowed for contours. With the second option, one enables contraction of edges on the contours. Note that this option will only be considered if the first option is not set.

Surface path options

	Surface path options:	<input type="checkbox"/> use surface paths	<input checked="" type="checkbox"/> fix control points of surf. path
--	------------------------------	--	--

The remeshing algorithm can be used in conjunction with surface paths, i.e. the user can specify surface paths which will be considered separately. If the second option is selected, control points, i.e. distinguished vertices on the surface path, will not be moved.

Remesh options (2)

	Remesh options (2):	<input checked="" type="radio"/> whole surface	<input type="radio"/> only around contours
--	----------------------------	--	--

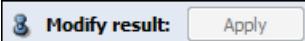
This option lets the user decide whether the whole surface should be remeshed or whether the remeshing should be restricted to the area around contours, where contours are the borders of patches.

Contour layers

	Contour layers:	4
--	------------------------	---

If the surface should only get remeshed around contours, this parameter enables the user to determine how large the area in number of triangles should be.

Modify result



This button allows modifying the result. If one has already remeshed the surface but would like to improve the result, hitting this button allows modifying the surface starting with the previously computed result. This functionality might save a lot of time if only minor changes are desired.

Examples

5.39 SampleScalarField

This module can be used to sample a scalar field at the cells of a tetrahedral grid. The result is either a scalar field storing the mean value or a vector field where minimum, mean and maximum for each cell are the components.

The module lets the user determine the sampling level, i.e. how many values are sampled for each tetrahedron and sample either the entire grid or a user defined selection.

Connections

Tetra grid [required]

Connect this port to the *tetrahedral grid*.

Scalar field [optional]

Connect this port to the *scalar field* to be sampled.

Ports

Info



This port provides information on the current *Sampling Level*.

Sampling level



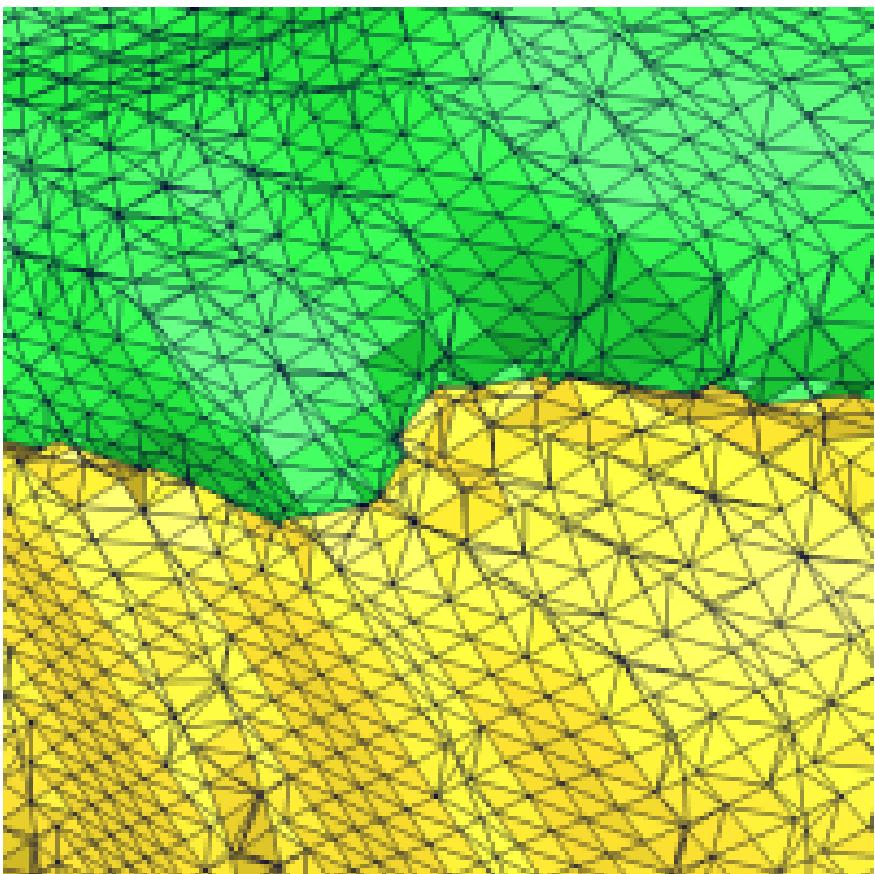


Figure 5.1: Surface before remeshing.

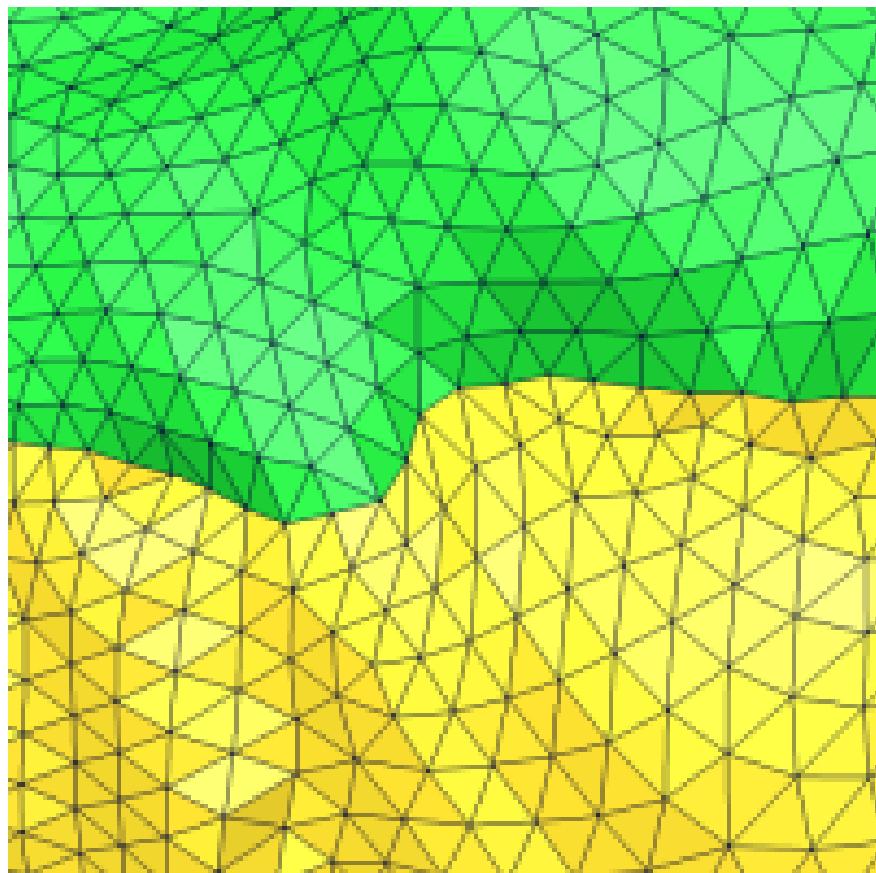


Figure 5.2: Surface remeshing with *high regularity*.

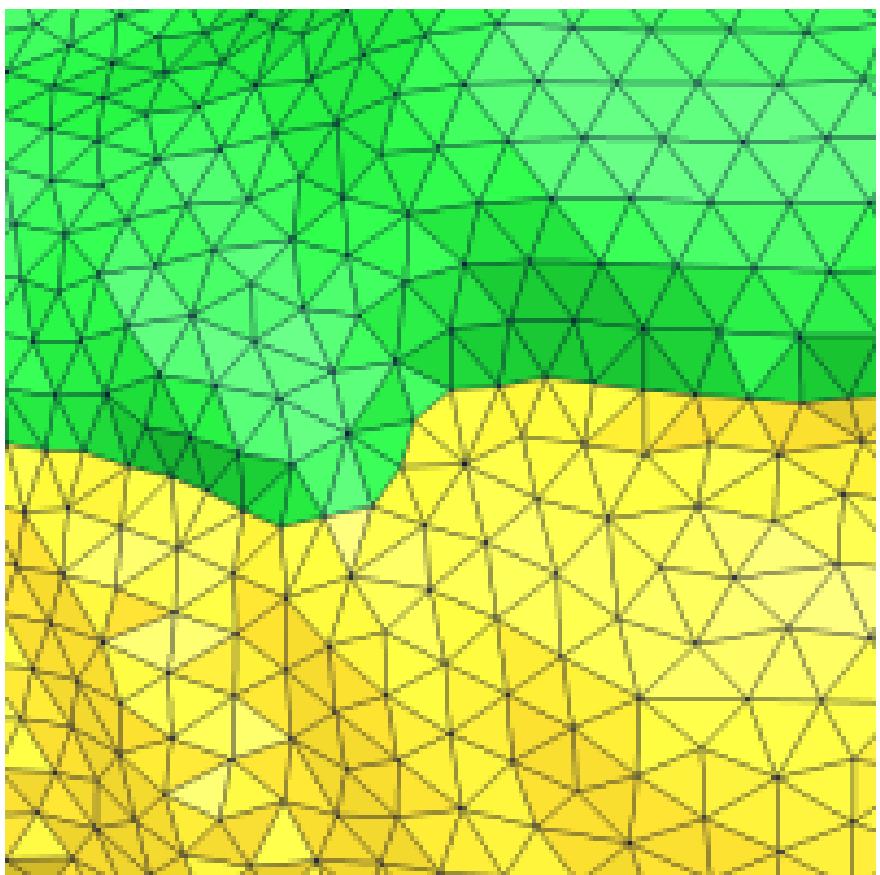
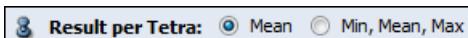


Figure 5.3: Surface remeshing with *best isotropic vertex placement*.

This port lets the user specify the number of samples to be taken from each tetrahedron. With *Sampling level 1* one sample is taken at the tetrahedron's barycenter. With *Sampling level 2*, and *3* the tetra cells are subdivided into 8 or 64 subcells, respectively, and samples are taken at the sub-cell's barycenters.

Result per Tetra



Select *Mean* if a scalar field storing the mean value is written as result or select *Min, Mean, Max* if a vector field with min, mean and max values as components shall be written.

Default value



The value specified at this port is written to all cells where either the scalar field is not defined or which have not been visible when *Sample: Selection only* has been used (see below). If *Min, Mean, Max* is selected at port *Result type* 3 textfields will be available to enter the defaults for the Min, Mean, and Max component, respectively.

Sample



Press the *Entire Grid* button to sample the scalar field for all tetrahedra or press the *Selection only* button to sample only at tetra cells currently visible. Use the *Buffer* port of *GridVolume* to modify the set of visible tetrahedra.

5.40 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

Surface [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



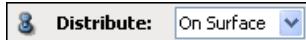
The length of the field lines, or more precisely, the number of atomic line segments, in forward respectively backward direction. The lines may stop earlier if a singularity (i.e. zero magnitude) is encountered or if the field's domain is left.

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.

5.41 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.

ROI [optional]

Connection to a module defining a region of interest.

Colormap [optional]

The colormap is used for pseudo-coloring the clouds.

Ports

Alpha Scale



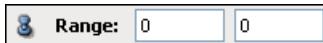
A global factor used to change the overall transparency of the individual splats. Higher values produce denser clouds.

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.

Optical Model



Switches between different optical models: only light emission or emissive and absorptive light model or absorbing light only.

Colormap



In case a colormap is connected to the splats 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*.

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*.

5.42 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 Virtual Reality Option 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

 Resolution:  0

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

 Density:  0

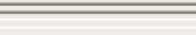
Logarithmic slider allowing to adjust the density of stream lines or stream ribbons. The higher the value the more lines or ribbons are traced.

Width

 Width:  1

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

 Length:  0

Adjusts the length of the stream lines or stream ribbons.

Seed type

 Seed type: 

Defines the seed type: along a straight line segment, along a circular line, or inside a square regions.

Mode

 Mode: lines ribbons

Defines if simple stream lines or stream ribbons should be traced.

Dragger

 Dragger:

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.

5.43 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.

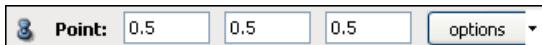
Ports

Colormap



Port to select a colormap.

Origin



This port defines the seed point of the stream surface. By pressing the menu button *options* 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



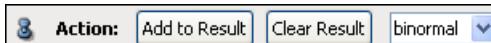
Controls the resolution of the discretized stream surface or, more precisely, the preferred edge length of the triangulation. Smaller values results in more details.

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`.

5.44 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

Surface [required]

Surface for which a LIC texture is to be computed.

VectorField [required]

Surface vector field or 3D vector field to be visualized.

ColorField [optional]

An optional scalar field which can be used for pseudo-coloring.

Color field2 [optional]

An optional scalar field which can be used for pseudo-coloring. The colors are multiplied with the colors from the first colorfield

Colormap [optional]

Colormap used for pseudo-coloring. If no colormap is connected the default color of the colormap port will be used.

Colormap2 [optional] Colormap used for pseudo-coloring using the second scalar field. If no colormap is connected the default color of the colormap port will be used.

Ports

Colormap



Port to select a colormap.

Colormap2



Port to select the second colormap.

ColorMode



Four 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. Third, 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. Finally, if *2 ColorFields multiplied* is selected, the LIC texture will be colored with two colors. The colors are chosen according to the values of the two scalar fields. The colors are multiplied.

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.

5.45 TensorDisplay

This module displays symmetric second order tensors using tensor glyphs. You can select either an ellipsoid, a cylinder, a cone, three lines, or superquadrics to indicate the tensor shape in a slice through the volume.

In order to render a glyph to visualize the tensor at a specific location the module maps the tensors eigenvalues and eigenvectors to a rotation and an anisotropic scaling of a default geometry, e.g. a sphere is deformed into an ellipsoid. The eigenvector's components are directly mapped to world-coordinate components.

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.

The glyphs are distributed in 2d mode based on a regular sampling grid. Its density can be controlled by the Resolution port. As locations on the grid in general do not co-incide with locations of the underlying data an interpolation of the tensors will be performed prior to display. This interpolation is implemented as a tri-linear interpolation on the tensor components.

In 3d mode the glyphs are located at the grid locations. The visibility of the glyphs can be controlled for example by a connected mask field.

Surface structures can also be used to influence the sampling of tensors. The location of surface vertices together with a distance value are used to generate a spatially distributed point cloud for tensor display. Again a tri-linear interpolation on tensor components is used to sample the tensor from the underlying field.

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 $A > 0$ to disable the display of tensors in the exterior part of the label field.

Shape data [optional]

Any currently displayed geometry can be connected to this port to be used as a replacement for the standard glyphs. A surface for example can be provided by connecting a SurfaceView display module. The shape is assumed to be oriented initially towards the positive y-direction (global axis) and is rotated based on the eigenvectors. After the rotation the positive y-direction of the shape will point towards the direction of the principal eigenvector. The eigenvalues are used for scaling the geometry.

Surface [optional]

Connect a vertex set (like a surface) and the vertex positions of the surface will be used as sample positions for the tensor field. If an offset value is provided the sample position is moved in the direction of the surface normal by the amount specified in the offset port (using the units of voxel size).

ROI [optional]

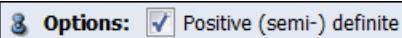
If a *SelectROI* module is connected the tensor values are only displayed in the region of interest.

Ports**Display mode**

If *2D* is selected the locations of the tensor glyphs are sampled based on the connected *1.10EmptyPlane* module. If *3D* mode is selected the tensors are sampled directly on the grid locations specified in the data (ignores the information provided by *EmptyPlane* and the *Resolution* port). This might produce a large amount of geometry. Use a connected label field to limit the amount of tensor glyphs generated.

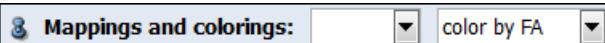
Display

Select either ellipsoids, cylinders, cones, lines or superquadrics 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 eigenvalue decomposition 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. Another option is provided that does not provide any mapping (as is mapping of eigenvalues). Visual artifacts might be produced if this option is used and negative eigenvalues are present.

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



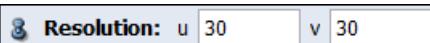
The tensor glyphs are scaled according to the value of their eigenvalues. If Eval2 and Eval3 are disabled, the tensor is drawn with a default scaling of 1 to 1/2 for the first principal direction to the second and third principal direction.

Display complexity



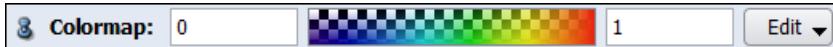
Values larger than 0.2 will increase the number of triangles used to display the glyphs which improves the display quality.

Resolution



The number of glyphs sampled over the size of the 1.10EmptyPlane. Larger numbers result in more dense tensor displays.

Colormap



MaskExpr



Offset



Scale



Commands

```
set TensorDisplayNoWarning <value>
```

If this global Tcl variable is defined and set to a value larger than 0, no warning dialog is displayed in case of a large number of objects.

```
set TensorDisplayNoWarningThreshold <value>
```

If this global Tcl variable is defined and set, the warning dialog will only be displayed if more objects than specified by the value would be created. The default threshold value is 65536.

5.46 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.

5.47 TetraCombine

This module takes two tetrahedral grids as input, puts them together and creates a new combined grid. *TetraCombine* 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]

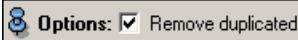
The first input grid.

GridB [required]

The second input grid.

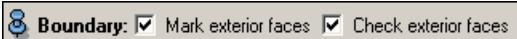
Ports

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.

5.48 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 for filling 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 *IvTo-Surface* does not. You must edit the region names before applying *TetraGen* to a surface created by the latter module.

It is recommended to study the tutorial *Grid generation*, which describes how to prepare a surface for tetrahedron generation.

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.

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 be displayed 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 including a suffix `grid` in 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.

If there is no predefined value, the mesh size will be initially set to 0, and the mean edge length of the surface triangles of the specific region will be taken as the desired mesh size. If the prescribed mesh size differs from the mean edge length of the surface triangles, the edge length of the generated tetrahedra will increase/decrease towards the center of the region in order to approach the prescribed mesh size.

If the prescribed mesh size differs too much from the mean edge length, this may imply generation of very distorted tetrahedra, or may even cause a failure in tetrahedron generation. If the prescribed mesh size is larger than three times or smaller than half of the mean edge length, a warning will be issued when pressing either the *Check* or the *Run now* button. You can decide whether you want to adapt the prescribed mesh size to the mean edge length, or keep the prescribed mesh size as is.

If you press the *Check* button, the input surface will be checked for some pre-conditions for tetrahedron generation, e.g. the surface must not self-intersect. The result of the check will be written to a spreadsheet report. The report will contain one row for each region, and 11 columns. The tests in columns 2 to 5 are all mandatory for tetrahedron generation. In columns 6 to 8, some quantitative measures are given. If they exceed certain values, this may imply generation of very distorted tetrahedra, or may even cause a failure in tetrahedron generation. If any of these tests fails, invoke the *Surface Editor* for the input surface. That editor provides the same tests and several tools for fixing the failures. Columns 9 to 11 of the report show the mean edge length of the surface triangles, the prescribed mesh size, and the estimated number of tetrahedra, which is computed from the volume enclosed by the region's surface, and the prescribed mesh size.

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 *online*. This may take some time for large surfaces. The progress bar indicates the current region and which part of its volume is already filled with tetrahedra.

5.49 TetraQuality

The *TetraQuality* module computes different quality measures for tetrahedral grids. For this purpose it can be attached to a *Tetra Grid* object as well as a *Grid Volume* module. Quality is calculated for all tetrahedra in the first case, and for all tetrahedra selected by the *Grid Volume* module in the second case. The output is a scalar field defined on the tetrahedral grid and equal to 0 on the non-selected tetrahedra.

The user can create a histogram of the quality for the tetrahedral grid. By default, the histogram is shown with a logarithmic scale to direct the focus on the tetrahedra with worst quality.

Press the *Apply* button to start the computation.

Connections

Data

The tetrahedral grid.

Grid volume

A *Grid Volume* module that selects the tetrahedra for which the quality is calculated.

Either the **Data** or **Grid volume** connection is required. The other one then becomes optional.

Ports

Quality Measure



This option menu lets you select between different quality measures:

- **Diameter Ratio:** ratio of diameters of circumscribed and inscribed sphere of a tetrahedron. 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. The optimal value, obtained for all dihedral angles in an equilateral tetrahedron, is $2\arcsin(1/\sqrt{3})$ (about 70.5 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. The optimal value, obtained for all solid angles in an equilateral tetrahedron, is $3\arccos(1/3) - \pi$ (about 31.6 degrees).
- **Edge Length:** length of all edges in a tetrahedron.

Select Angle



If the quality measure is dihedral or solid angle, you can select

- all angles: the scalar output is the mean angle in each tetrahedron and the histogram represents all angles in the grid ;
- the minimal angle: in each tetrahedron the scalar output is the minimal angle, which is represented on the histogram ;
- the maximal angle: in each tetrahedron the scalar output is the maximal angle, which is represented on the histogram ;
- minimal and maximal angles: in each tetrahedron the scalar output is their half sum and in the histogram both minimal and maximal angles are represented for each tetrahedron.

Select Edge



If the quality measure is edge length, you can select

- all edges: the scalar output is the mean length in each tetrahedron and the histogram represents the length of all edges in the grid ;
- the minimal edge length: in each tetrahedron the scalar output is the minimal edge length, which is represented on the histogram ;
- the maximal edge length: in each tetrahedron the scalar output is the maximal edge length, which is represented on the histogram ;

- minimal and maximal edge lengths: in each tetrahedron the scalar output is their half sum and in the histogram both minimal and maximal edge lengths are represented for each tetrahedron.

Samples



This slider lets you select the number of samples for the histogram. The default values should be adequate.

Histogram



If you select this toggle, a plot window will appear, once you've pressed *Apply*, showing the histogram of qualities for all selected tetrahedra. If no tetrahedra have been selected, the plot window will not be shown.

5.50 TetraVectors

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*) or 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 vector-field in to account.

Connections

Data [required]

The 3D tetrahedral vector field to be visualized. TetraVectors can only visualize vectors defined on the vertices (nodes) of tetrahedrons. If the vectors are defined on the tetrahedrons themselves, TetraVectors cannot be attached to the data.

PortModule [required]

The *GridVolume* or *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.

5.51 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*.

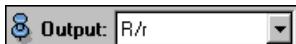
Ports

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.
- **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.

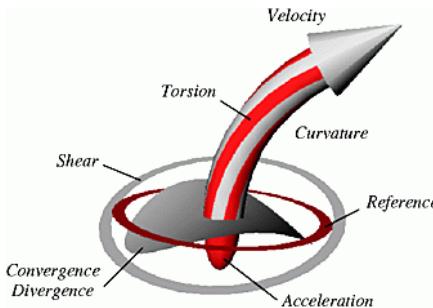


Figure 5.4: Components of the vector field probe.

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.

Seeds [optional]

Seed positions for vector arrows. If a *VertexSet* is connected to this port, the vertices are used as seed positions for the vector arrows.

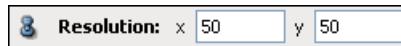
Ports

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



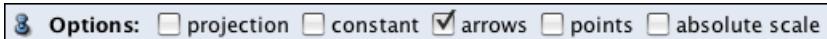
Scaling factor used to control the length of the vector arrows.

ScaleZ



Scaling factor used to control the z-coordinate 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.

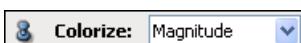
Constant: If this option is set then all arrows will be of equal length. Otherwise, the length of the arrows is chosen to be proportional to vector magnitude.

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.

Absolute scale: If this option is set then vector length are mapped directly to physical space. If not set, a data-dependent scaling is performed first.

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/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 view [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. *Tangent:* (Only for vector fields on triangle center) If this option is set then only the tangent component of the vector is shown.

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.

6 Microscopy Option

6.1 BeadExtract

This module can be used to resample and average the image of one or multiple beads, i.e., fluorescing sub-resolution microspheres, thereby obtaining an approximation of a point spread function (PSF) required for non-blind deconvolution.

The module must be connected to the image data set containing the measured beads as well as to a landmark set indicating the center positions of all beads to be resampled and averaged. The whole process of obtaining a point spread function from a bead measurement is described in a separate *tutorial on bead extraction*. Please refer to this tutorial for more information on how to use the module.

Press the *Apply* button to actually extract the beads with the center of the resulting data set corresponding to the current positions of the landmarks. The image volume around every landmark is extracted and resampled using a Lanczos filter (compare the *Resample* module). The resampled bead images are then added to the result object. The final PSF is not normalized.

Connections

Data [required]

Connection to a uniform image data set containing measured beads.

Landmarks [required]

Connection to a landmark set indicating the center positions of the beads to be resampled and averaged.

Ports

Info



Displays the number of beads to be resampled and averaged.

Resolution



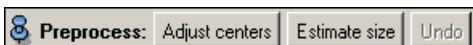
Specifies the number of voxels of the final PSF image to be generated. If a PSF image is connected as a result object to this module, the port becomes insensitive and the number of voxels of the result object are used.

Voxel size



Specifies the voxel size of the final PSF image to be generated in micrometers. If a PSF image is connected as a result to this module, the port becomes insensitive and the voxel size of the result object is used.

Preprocess



Adjust centers: If this button is pressed, the landmark positions of the data set connected to port *Landmarks* are shifted to the center of gravity of the corresponding bead. It is required that the initial landmark positions be inside the bead images and that neighboring beads do not overlap significantly. Otherwise incorrect center positions may be computed.

Estimate size: If this button is pressed, the desired number of voxels of the final PSF image specified in port *Resolution* are computed automatically. Before this is done the beads' center positions already should have been adjusted. The estimate is computed by determining the extent of the biggest spot around any landmark. Again, it is required that neighboring beads do not overlap significantly. The *Estimate size* button becomes insensitive if a result object is connected to the module. In this case always the size of the existing PSF image will be used.

Undo: Undoes the effect of any of the previous two buttons. For example, if wrong center positions have been computed after pressing *Adjust centers*, the original landmark positions can be restored using the *Undo* button.

6.2 Convolution

This module convolves two uniform 3D data objects with each other by Fourier transforming the two inputs, multiplying them, and then transforming them back. The module is part of the *Amiradeconvolution modules*. It can be used for example to verify the results of image deconvolution.

The bounding box and voxel sizes of the input data set and the convolution kernel are ignored by this module. Use the *Resample* module to make sure that the resolution of both inputs is identical.

Press the *Apply* button to start the computation.

Note: The Convolution module currently only supports data sizes up to 536.870.911 voxels (including border voxels).

Connections

Data [required]

The data set to be convolved.

Kernel [required]

The convolution kernel.

Ports

Border width

A dialog box titled "Border width" with three input fields: x: 12, y: 12, z: 30. Each field has a spin button on the left and a text input field on the right.

Defines the size of the border region. A border region is necessary if the input data set doesn't fade out to black at the boundaries.

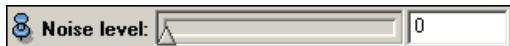
Options

A dialog box titled "Options" with two checked checkboxes: "normalize kernel" and "add noise".

If *normalize kernel* is selected, the integral of the convolution kernel (connected to port *Kernel*) will be normalized to one. If this is not the case, the intensities of the convolved data set will be scaled by the actual integral.

If *apply noise* is selected, the values of the convolved data set will be multiplied by random numbers uniformly distributed around 1 (white noise).

Noise level



This port will only be shown if the *apply noise* option of the *Options* port has been selected. It specifies the amount of noise applied to the output, i.e., the range of the random numbers around 1, by which the result is multiplied.

6.3 CorrectZDrop

This module lets you fix artifacts in 3D microscopic images caused by light absorption in other slices. If such artifacts are present, the average intensity in lower slices seems to be decreased. This so-called *z-drop* or *intensity attenuation* can be corrected automatically by fitting an exponential curve to the average intensities in each of the slices, or manually by providing a user-defined formula.

Press the *Apply* button to start the computation.

Connections

Data [required]

The image data exhibiting a z-drop artifact. Scalar fields with uniform or stacked coordinates as well as *multi-channel fields* are supported.

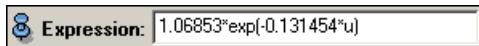
Ports

Mode



Lets you select between *automatic* mode and *manual* mode.

Expression



This port is only available if *manual* mode has been selected. It provides a text field where you can enter a formula specifying a factor used to multiply the intensity values in each slice. Within the formula the variable *u* specifies the slices. *u* will take the value 0 for the first slice and 1 for the last slice. For the other slices it takes intermediate values depending on the actual slice

location (this makes support of stacked coordinates easy). In automatic mode the following formula $a \cdot \exp(b \cdot u)$ will be used, where a and b are fitted automatically. If you first perform an automatic z-drop correction and then switch to manual mode, the fitted exponential will be displayed in the port's text field.

6.4 DataPreprocess

This module can be used to apply both a background and a flatfield correction to a raw 3D image stack.

For the background correction, a single background image must be provided at the *background* port of the module. The background image should be a nearly black image recorded with the camera's shutter closed. This image is subtracted from all slices of the 3D input data set, thus compensating for any dark current of the camera's CCD detector.

For the flatfield correction, a single flatfield image must be provided at the *flatfield* port of the module. The flatfield image should be an unfocused almost white image taken from a drop of homogeneously fluorescing dye. The intensities of the 3D input image are then scaled according to the normalized intensities of the flatfield images. The input image gets brighter at pixels where the flatfield is dark and vice versa. In this way non-uniform sensitivity of the camera's CCD detector is compensated for. If both a flatfield and a background image are present, the background is subtracted from the flatfield too.

Press the *Apply* button to start the computation and produce a corrected output data set.

Connections

Data [required]

The raw 3D image stack to be corrected. Any regular scalar field with uniform coordinates is supported.

Background [optional]

An optional 2D background image with the same number of voxels in the x and y directions as the 3D input image. If an input is present at this port, a background correction is performed (see above).

Flatfield [optional]

An optional 2D flatfield image with the same number of voxels in the x and y directions as the input image. If an input is present at this port, a flatfield correction is performed (see above).

Ports

Background



Background: mean=9.11829 deviation=1.6651

Displays the mean value and the standard deviation of the background image, if such an image is present. Only the first slice is considered.

Flatfield



Flatfield: No flatfield image connected

Displays the mean value and the standard deviation of the flatfield image, if such an image is present. Only the first slice is considered.

6.5 Deconvolution

This module is the front-end for deconvolving 3D microscopic images. Two different iterative maximum-likelihood image restoration algorithms are provided, a non-blind one and a blind one. For a general description of the deconvolution process, please refer to the provided *tutorials*.

The resulting deconvolved data set will be stored in the Pool. If no input PSF is specified or if the blind deconvolution algorithm has been selected, the estimated PSF will be stored in the Pool also.

Press the *Apply* button to start the deconvolution process. Since deconvolution is a time consuming operation, it optionally can be performed as a batch job (see the *Action* port below).

Note: The Deconvolution module currently only supports data sizes up to 536.870.911 voxels (including border voxels).

Connections

Data [required]

The data set to be deconvolved.

Kernel [optional]

The point spread function (PSF) to used for deconvolution. If no PSF is specified, an estimated PSF is calculated automatically based on the numerical aperture of the microscope, the wavelength of the emitted light, and the refractive index. If a blind deconvolution is to be performed, an input PSF (if connected) will be used as an initial estimate.

Ports**Border width**

A control panel with a blue icon and the text "Border width: x 8 y 8 z 67". The "x" and "y" fields are highlighted in red, while the "z" field is greyed out.

Defines the size of the border region. A border region is necessary if the input data set doesn't fade out to black at the boundaries. For performance reasons it might advisable to choose values such that the sum of the size of the input data set and the border width results in a power of two. For example, if the data set consists of 118 slices, a border width of 10 slices in z direction would be a good choice.

In the case of widefield data it is sometimes advisable to have the border width in z direction exactly as large as the data set itself. If this is the case, the border region will be initialized by mirroring the values from the actual data volume. Otherwise, the values of the first slice and of the last slice will be interpolated linearly.

Iterations

A control panel with a blue icon and the text "Iterations: 1 ▲ 60". The slider is currently at position 60.

Specifies the number of iterations of the deconvolution procedure.

Initial estimate

A control panel with a blue icon and the text "Initial estimate: const input data previous result". The "const" option is selected.

Specifies the initial estimate of the deconvolution algorithm. If *const* is chosen, a constant image is used initially. Often, this yields smoother results than the second option, namely *input data*. The third option (*previous result*) is only available if the input data set has already been deconvolved previously. Use this option if you want to apply some additional deconvolution iterations.

Overrelaxation

	Overrelaxation:	<input checked="" type="radio"/> none	<input type="radio"/> fixed	<input type="radio"/> optimized
--	------------------------	---------------------------------------	-----------------------------	---------------------------------

Overrelaxation is a technique to speed up the convergence of the iterative deconvolution process. In most cases *fixed* overrelaxation is a good choice. For non-blind deconvolution also an *optimized* overrelaxation technique is available. This method further accelerates convergence but is more memory and time consuming.

Regularization

	Regularization:	<input checked="" type="radio"/> none	<input type="radio"/> intensity based
--	------------------------	---------------------------------------	---------------------------------------

This port specifies if you want *intensity-based* regularization or not.

Method

	Method:	<input checked="" type="radio"/> standard	<input type="radio"/> blind
--	----------------	---	-----------------------------

This port specifies whether standard (non-blind) or blind deconvolution should be used.

PSF Parameters

	PSF Parameters:	NA <input type="text" value="1.35"/>	λ <input type="text" value="0.52"/>	n <input type="text" value="1.516"/>
--	------------------------	--------------------------------------	---	--------------------------------------

Parameters for calculating the point spread function. This port will only be shown if standard (non-blind) deconvolution has been selected and no input PSF was specified, or if blind deconvolution has been selected, in which case these parameters act as constraints.

NA denotes the numerical aperture of the microscope. *lambda* denotes the wavelength of the emitted light in micrometers with the voxel sizes also being interpreted in micrometers. Finally, n denotes the refractive index of the specimen.

Microscopic Mode

	Microscopic Mode:	<input checked="" type="radio"/> widefield	<input type="radio"/> confocal
--	--------------------------	--	--------------------------------

Selects whether the input image has been recorded using a widefield microscope or using a confocal microscope. This is important for the PSF generation as well as for the selection of appropriate constraints during blind deconvolution.

Action



Since deconvolution is a time consuming operation, it optionally can be performed as a batch job. A batch job can be submitted using the *Batch job* button. If this button is pressed, first a dialog is popped up allowing you to specify the filename of the final deconvolved data set as well as the number of optional check point files. A check point stores an intermediate result obtained after a certain number of iterations have been performed. The actual deconvolution job is started via the job dialog accessed by *File / Jobs* menu item. The job dialog is popped up automatically after the job has been finally submitted, but this may take a few seconds if there are currently no jobs running.

6.6 DistanceMap

This module computes a 3D distance field of a 3D object. Each voxel will be assigned a value depending on the distance to the nearest object boundary. The boundary voxels of the object are assigned a value of zero whereas the assigned value increases as the distance increases.

To use this module it must be connected to a uniform label field where each voxel with a nonzero value is assumed to belong to the object.

Press the *Apply* button to start the computation.

Connections

Data [required]

Labelfield from which the distance map is computed.

Ports

Type



You may either choose a true Euclidean distance metric or an approximation based on a 3x3x3 chamfer metric. The latter is much faster to compute and accurate enough for most applications. Single Seeded computes a distance

map which expresses the distance from a single seed point rather than from the boundary.

Chamfer Weights



This port is only available in chamfer mode. Different chamfer metrics are available. The 1-2-3 metric is equivalent to only considering a 6-neighborhood when propagating the distance value, whereas the 3-4-5 considers a 26-neighborhood and is a better approximation of the Euclidean distance metric. Float also corresponds to a 26-neighborhood but the resulting field will have float data type instead of short int.

Region



This port is not available in Single Seeded mode. Choose in which region the distance field will be computed:

- **Inside:** Inside the object (outside will be set to zero).
- **Outside:** Outside the object (inside will be set to zero).
- **Both (unsigned):** Inside and outside the object. A positive distance is computed whether the position is inside or outside the object.
- **Both (signed):** The distance value will be negative at a position inside the object and positive outside the object.

Point



This port is only available in Single Seeded mode. Specifies the seed point for the distance map in world coordinates. You can use a dragger to adjust it.

6.7 FourierTransform

This module computes a discrete forward or backward Fourier transform from a scalar input data set with uniform coordinates. Alternatively, the power spectrum, i.e., the squared magnitude of the Fourier transform can be computed.

The origin of the input data set will be ignored by this module. Also, instead of being expressed in wave numbers, the bounding box of a Fourier transformed data set will be the same as the bounding box of the input.

Press the *Apply* button to start the computation.

Note: The FourierTransform module currently only supports data sizes up to 536.870.911 voxels (including border voxels).

Connections

Data [required]

The input data to be Fourier transformed.

Ports

Mode



Option menu specifying the action to be performed.

If a real scalar field is connected to the module, three options are available, namely *forward*, *forward complex*, and *power spectrum*. If *forward* is selected, the output is stored in a special so-called half-complex format. Such an output can be back-transformed, but not many other operations can be performed on it. If *forward complex* is selected, the output will be a complex-valued scalar field with the same number of voxels as the input object. Finally, if *power spectrum* is selected, the output is a real-valued scalar field with the same number of voxels as the input object containing the squared magnitude of the Fourier transform.

If a complex scalar field in half-complex format is connected, the only option is *backward*, indicating a backward Fourier-transform.

If an ordinary complex scalar field is connected to the module, the three options *forward*, *backward*, and *power spectrum* are available. The first two options specify a forward and backward Fourier transform, respectively. The output is a complex scalar field with the same resolution as the input. If *power spectrum* is selected, the output is a real-valued scalar field with the same number of voxels as the input object containing the squared magnitude of the Fourier transform.

6.8 PSFGen

This module can be used to compute a point spread function (PSF) for deconvolution of widefield and confocal microscopic image data. PSF computation is based on electromagnetic vector theory. It can be created by selecting it from the *Create / Others* menu of the main window.

Press the *Apply* button to compute the PSF.

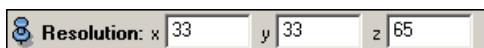
Connections

Data [optional]

A uniform scalar field (usually the image data to be deconvolved) can be connected to this port. The values of the *Resolution* and *Voxel size* ports will be set automatically to the values of the input field then.

Ports

Resolution

A horizontal input field labeled "Resolution" with three spinners for x, y, and z dimensions. The x and y spinners both show the value 33, while the z spinner shows 65.

Resolution:	x 33	y 33	z 65
-------------	------	------	------

The number of voxels of the PSF image to be generated.

Voxel size

A horizontal input field labeled "Voxel size [um]" with three spinners for x, y, and z dimensions. All three spinners show the value 0.04.

Voxel size [um]:	x 0.04	y 0.04	z 0.04
------------------	--------	--------	--------

The voxel size in microns of the PSF image to be generated.

PSF Parameters

A horizontal input field labeled "PSF Parameters" with three spinners for NA, lambda, and n. The NA spinner shows 1.35, the lambda spinner shows 0.52, and the n spinner shows 1.516.

PSF Parameters:	NA 1.35	λ 0.52	n 1.516
-----------------	---------	----------------	---------

Parameters for calculating the point spread function. *NA* denotes the numerical aperture of the microscope. *lambda* denotes the wavelength (as measured in a vacuum) of the emitted light in micrometers. For confocal data the excitation and emission wavelength are assumed to be identical. In this case it might prove useful to compensate by supplying a value between excitation and emission wavelength as parameter. Finally, *n* denotes the refractive index of the specimen.

Microscopic Mode



Specifies whether the PSF of a widefield microscope or of a confocal microscope should be computed.

7 Quantification+ Option

7.1 Quantification

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. See *Quantification+ Option User's Guide* for more information.

Connections

Data [required]

Initial input image.

Ports

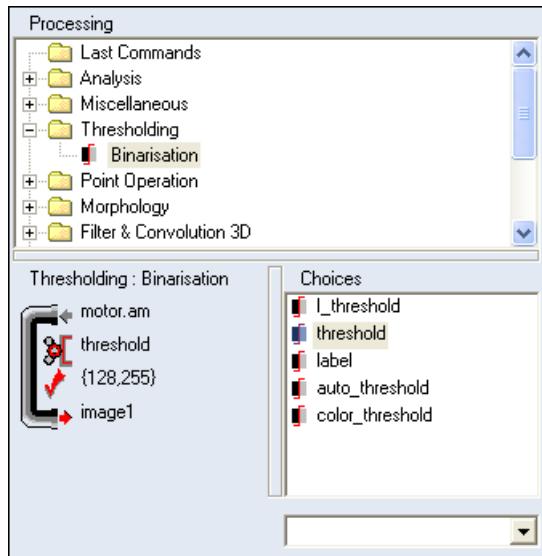
Action



Pressing the first button launches the Visilog result viewer. The two following buttons bring up the Visilog documentation within a PDF viewer installed on the system:

- click on the first one to launch the Visilog's Users Guide,
- click on the second one to get context sensitive help on the current Visilog command (also accessible when clicking the F1 button on the Visilog port).

Visilog



This port is identical to the Visilog user interface, in which the command is chosen from the command tree.

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=imagine1 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 fields	Calibration,

printText <text>

Print text in a notepad of ResultViewer

setCommand <command>

Initialize the Visilog Module 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\_ima"
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"  
Visilog exeIp "ip=IpCopy input=grey\\_ima output=ima2"
```

7.2 Quantification-Analyse

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.

7.3 Quantification-Threshold

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 is not visible when the tool is called from Visilog.

Connections

Data [required]

Image data the thresholds are applied to.

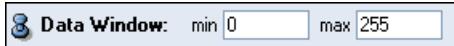
Ports

Orientation



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

Data Window



It allows you to restrict the range of visible data values. Values below the lower

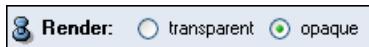
bound are mapped to black, values above the upper bound are mapped to white.

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.

8 Neuro Option

8.1 ComputePerfusion

Computes mean transit time (MTT), blood flow (CBF), and blood volume (CBV) from a perfusion weighted image series (MR or CT). The different computations for CT or MRI images are selected based on the DICOM tag *echo time* (0018,0081). For MR data this tag should contain a value larger than 0.

Load your (DICOM) images as a time series and connect this module to the *Time-SeriesControl* module. The module requires a spreadsheet as secondary input. This table can be created by the module using the sample button functionality. Press a sample button (e.g. AIF) and in interactive mode select a vessel for arterial inflow or venous outflow using the middle mouse button. A spreadsheet object should be created which contains the sampled time-density curve at that location.

The algorithm implemented in this module performs the perfusion computation based on deconvolution.

Connections

Data [required]

Connect this port to a *TimeSeriesControl* module. The connected time series will be used for computation.

AIF [optional]

This port can be connected to a SpreadSheet object. The spreadsheet should contain a single float-valued column "AIF" (arterial input function) with a sample for each time point in the series. A second column "VOF" can be provided for the venous output function.

Mask [optional]

If a mask dataset is connected, the computation is restricted to areas that are not marked as Exterior.

Ports

Options

 Options: extended

This checkbox enables the ports described below.

Sample

 Sample:

The sample port provides two functions, one for sampling the arterial input and one for the venous output function. The data should be visualized using an *OrthoSlice* module (or similar) to provide the point of interest in 3D space. Press one of the buttons and, in the interactive mode (cursor shows arrow), select a point on the slice using the middle mouse button. This 3D location is used as a seed point for a search algorithm that looks for the voxels with the strongest signal within a radius specified in port *Search Radius*. An average over several of the voxels is used to calculate the time-density function from the connected time series. The result is stored in a *SpreadSheet* object connected to the AIF port of this module.

The voxel positions used for the computation of the input and output functions can be stored in a *LabelField*. The creation of such a *LabelField* can be enabled with the Tcl command `createLabelfield 1`. It includes 5 materials: "Exterior", "AIF_search_area", "AIF_measurement", "VOF_search_area" und "VOF_measurement"

Inter-frame time

 Inter frame time: 0.5

This port specifies the time between image frames in seconds.

Smoothing

 Smoothing:

Activates Gaussian smoothing.

Smoothing Kernel Size

 Smoothing kernel size: x y

Specifies the size of the Gaussian filter kernel.

Smoothing Kernel Sigma



Specifies the sigma of the Gaussian filter kernel.

Smoothing kernel sigma: x [0.4] y [0.4]

Phases in baseline



The number of images used to compute the pre-contrast baseline. Adjust this value if less than the default number of phases before contrast adaptation are recorded.

Phases in baseline: 6

Note that for MRI data the first image will always be removed from the computation as it usually contains an artificially high signal intensity.

Limit to phase



With this port the computation can be restricted to a number of volumes smaller than the full recorded time series. In some cases this can reduce the effect of noise on the computation and limit the effect of recirculation on the computation.

Limit to phase: 49

Density factor



This value describes the density of brain tissue. The value should not be changed for normal operation.

Density factor: 1.05

Hematocrit large vessel



A ratio which is used by the algorithm. The value should not be changed for normal operation.

Hematocrit large vessel: 0.4

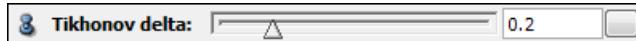
Hematocrit small vessel



A ratio which is used by the algorithm. The value should not be changed for normal operation.

Hematocrit small vessel: 0.7

Tikhonov delta



Tikhonov delta: 0.2

This value sets a regularization factor used during the deconvolution step. As a matrix inversion is required, this regularization attempts to reduce the influence of noise on the result of the reconstruction. Use a lower value to regularize more.

Signal-to-noise



A warning message is produced if the signal-to-noise ratio of the images in the baseline (before contrast) is smaller than this value. In case this warning is produced, the value of the phases in baseline should be adapted to not include the initial inflow of contrast. The warning might also be produced if the data contains more than the normal amount of noise.

Search Radius



This port specifies the radius in mm around the seed point that is searched for optimum signal strength of the AIF and VOF. Note, the measurement unit is only in mm if the voxel size or bounding box of the data set is specified in mm.

Filter Voxel



This port enables averaging of the arterial input and venous output functions.

Minimum Uptake Difference



The minimum uptake difference can be adjusted, which is required to use a voxel for the AIF/VOF calculation. The difference is calculated between voxel values at the first time point and all consecutive time points.

Maximum Starting Value



The maximum value that a voxel of the AIF/VOF can have to be a valid seed point.

8.2 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 module will attempt to read the gradient direction from the parameter section of the attached data files. The user might supply known gradient directions if this extraction fails.

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 ($b0$). Usually this is a dataset with higher brightness compared to the gradient weighted images. The module only supports a single $b0$ volume. If more $b0$ volumes are available they should be averaged prior to using this module.

If the necessary input data objects do not fit into the available main memory, consider using the *ComputeTensorOutOfCore* module.

Connections

S0 [required]

This port must be connected to the data set representing the measurement without Gradient ($b0$).

S1,...,S7 [required]

These ports must be connected to the gradient weighted images. Further connection ports will be generated dynamically, when these first seven ports are all connected.

Ports

Diffusion weighting factor

	Diffusion weighting factor:	750
--	-----------------------------	-----

The diffusion weighting factor used in this study.

Options

	Options:	<input type="checkbox"/> smoothing	<input type="checkbox"/> correct oblique
--	----------	------------------------------------	--

If the *smoothing* option is enabled, two more ports will allow you to request a

Gaussian filtering of the raw data prior to the computation of the tensor field. The *correct oblique* option will enable a correction of the gradient directions that are read from the input data if an oblique slice scan is detected. If the gradient directions are already corrected, this option should remain off.

Size

Size: x [3] y [3] z [3]

A horizontal input field with three separate text boxes for 'x', 'y', and 'z' dimensions, each containing the value '3'. A small icon of a 3D cube is positioned to the left of the text 'Size'.

This port is visible only if smoothing is enabled. It specifies the size of the Gaussian kernel used for smoothing.

Sigma

Sigma: x [0.6] y [0.6] z [0.6]

A horizontal input field with three separate text boxes for 'x', 'y', and 'z' dimensions, each containing the value '0.6'. A small icon of a 3D cube is positioned to the left of the text 'Sigma'.

This port is visible only if smoothing is enabled. It specifies the variance used by the Gaussian filter for smoothing.

Buttons

Attach selected data Import gradients Store gradients in files

A horizontal row of three buttons: 'Attach selected data', 'Import gradients', and 'Store gradients in files'. The first button has a small icon of a 3D cube next to it.

This port supports the workflow of the ComputeTensor module.

- *Attach selected data:* The ComputeTensor module needs to be connected to many gradient weighted image dataset all loaded into Amira. Select the loaded gradient weighted images in the object pool and press this button. All selected datasets will be added as gradient weighted images.
- *Import gradients:* The gradient for each gradient weighted image can be imported into the user interface from a text file. The file should contain the gradients as single lines with 3 components each. After pressing this button the gradients are displayed in the gradient ports of this module.
- *Store gradients in files:* Gradients specified in the user interface can be permanently stored in the data objects of the pool. This button will add the gradient to each gradient weighted image as a parameter entry called *DTIGradient*. If a data set with such an entry is connected to ComputeTensor its value is used as gradient.

G1,G2,...

G1: [1] [0] [1]

A horizontal input field with three separate text boxes for 'G1', 'G2', and 'G3' values, each containing the value '1'. A small icon of a 3D cube is positioned to the left of the text 'G1'.

Ports allow you to supply user-defined Gradients which are 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.

8.3 ComputeTensorOutOfCore

This module computes a symmetric second order (diffusion) tensor from a set of N ($N \geq 7$) 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:/MyData/",
        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 Benger 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



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.

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.

8.4 ConvertTalairach

In brain research it is often desired to describe the locations of brain structures independent from individual differences in the size and overall shape of the brain. In the Talairach coordinate system the anterior commissure and posterior commissure lie on a straight line along the y-axis. This coordinate system is completely defined by requiring the midsagittal plane to be vertical and the superior part of the brain having positive Z values. When specifying a location in the structure in 3D space, the anterior commissure is used as a reference point and origin of the coordinate system. The transformation of brain data into Talairach coordinates simplifies registration and spatial warping of image data from the brain.

Connections

Data [required]

Connect MRI, PET, CT, SPECT etc. brain image data in a uniform scalar field to this *Data* input port.

Ports

Slice Number



Move the slice to different positions in the image data set.

Show



Enable/Disable visualization of the slice through the data and the chosen landmarks.

Select



Use these buttons to indicate the anterior commissure (AC), posterior commissure (PC), and a location on the midsagittal plane (MP).

AC Location



Coordinates of the anterior commissure.

PC Location



Coordinates of the posterior commissure.

Mid-Plane



Coordinates of a point on the midsagittal plane.

Transform



Apply the computed transform and resample the data set.

Apply

Compute the transformation.

8.5 CreateGradientImage

Module computes the bi-linear form of a given vector and an input tensor field. The bi-linear form is transformed to a diffusion weighted image that is suitable as input to ComputeTensor.

Connections

Data [required]

Connect a symmetric second order tensor field.

Mask [optional]

Specifies which voxel (non-Exterior) should be used to compute the bi-linear form.

Ports

Gradient

	Gradient:	<input type="text" value="1"/>	<input type="text" value="0"/>	<input type="text" value="1"/>
--	------------------	--------------------------------	--------------------------------	--------------------------------

Specify a gradient direction. The gradient direction is normalized to one before used by the module.

BValue

	BValue:	<input type="text" value="1"/>
--	----------------	--------------------------------

A value that is multiplied to the bilinear form.

Mode

	Mode:	<input type="radio"/> Input	<input checked="" type="radio"/> 16bit
--	--------------	-----------------------------	--

Either the data input type or a 16bit integer is used to exported the data.

Scale output

	Scale output:	Scale <input type="text" value="4000"/>	Intercept <input type="text" value="0"/>
--	----------------------	---	--

The data can be scaled before export $A * \text{scale} + \text{Intercept}$.

MaskExpr

	MaskExpr:	<input type="text" value="A>0"/>
--	------------------	-------------------------------------

This expression is evaluated for each voxel. If true (!0) the bi-linear form is evaluated.

8.6 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) of a tensor field.

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 *MaskExpr* below.

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 fourth material from the material list of the Segmentation Editor display of the mask.

Exposure

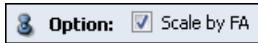


Controls the exposure e of the color representation. Larger values will make the images brighter. It is computed as:

$$C_{r=x,g=y,b=z} = (1 - \exp(-|e_1(x,y,z)|FA) e) 255 \quad (8.1)$$

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.

Option



If enabled colors will be scaled by the fractional anisotropy FA . Thus the visibility of areas with a high fractional anisotropy will be enhanced.

8.7 FiberTracking

This module implements the tensorline algorithm and computes line sets from tensor fields by numerical integration. The input 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 with at least one label that fills the area in which fibers are expected. The fiber tracking will not continue into areas assigned to the Exterior material. The module will produce one line set for each label where the label is used as the seed area for the fiber tracts.

Note that the number of line segments produced may be very large. The *Select-Lines* 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 seed points and therefore the number of lines produced.

Connections

Data [required]

A vector field 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 treshold) and if the module should check for how straight the path of integration is (direction threshold).

Options2

Options2: f as CL

Port which when enabled replaces the weighting parameter f by the linear tensor shape measure for each voxel.

FA threshold

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

Tensor weight g: 1

Port to specify the weighting between STT and TEND fiber tracking.

The weighting factors f and g should both 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

Tensor weight f: 1

Port to specify the weighting between STT and TEND fiber tracking.

Direc threshold

Direc threshold: 45

The angle in degree at which the fiber tracking is stopped.

Step size

Step size: 1

The step size of the fiber tracking in units of the voxel size.

Length



The length of the fiber tracked in units of the voxel size.

8.8 SegmentBrain

This module takes a T1 or T2 MRT scan of the human head as input and creates from it a label field that masks out non-brain tissue. Such a mask can be useful for visualization where it allows to visualize the surface of the brain in isolation as well as for particular analyses of the human brain such as *Diffusion Tensor Imaging* and *brain mapping*.

SegmentBrain typically works without adjusting parameters. If the result label field has inner holes it may be necessary to increase the value in port *Smoothing*. Inner holes can also be the result of gradients imposed by an inhomogeneous magnetic field. In that case one could use *NormalizeImage* or *CorrectZDrop* (requires a Microscopy Option license) to correct for the gradient prior to using *Segment-Brain*.

Connections

Data [required]

A uniform scalar field containing the MRT scan.

Ports

Export



Check this box if a surface representation of the mask is wanted in addition to the label field.

Smoothing



By default, *Smoothing* is set to 0. If inner holes occur in the result label field this value can be increased. For some images, values between 10 and 20 are sufficient, for others *Smoothing* must be set to 200 to get satisfactory results.

Since computation times are typically short it is recommended to increase the value successively until a satisfactory segmentation is achieved.

9 DICOM Reader

9.1 DicomSend

This module sends all slices of a HxUniformScalarField3 to a remote DICOM node. Destination nodes have to be configured. This can be done by clicking on the edit button. Before the images are sent, a dialog box is opened allowing the user to edit DICOM relevant information. If the image stack has a parameter bundle **DICOM**, *DicomSend* will look for available DICOM tags to pre-fill those fields.

Connections

Data [required]

The image data set to be sent.

Ports

Dicom node



Pull-down menu to select the destination DICOM node.

Action



Sends the images.

Edit Dicom Nodes



Opens a dialog box to add, edit, or delete DICOM node configurations.

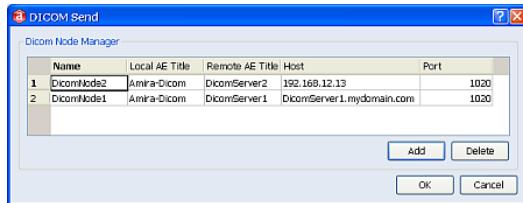


Figure 9.1: Edit DICOM nodes dialog.

A table displays the current settings for each DICOM node. The user can edit the information in the table by clicking on the desired table cell and typing in the new information. The columns and their meaning are:

- **Name** This is the name of the DICOM node as it appears in the pull-down menu of port *Dicom node*.
- **Local AE title** This is the name of the local DICOM node. Typically this name needs to be known by the destination DICOM node.
- **Remote AE title** This is the name of the destination DICOM node.
- **Host** This is the name or IP of the host where the remote DICOM node is running.
- **Port** This is the port number on the remote host that accepts DICOM images.

To add a DICOM node press the *Add* button and edit the fields in the table. To delete a DICOM node select a row in the table and press the *Delete* button. To accept the current settings press *OK*. This writes the configuration into \$AMIRA_ROOT/share/dicom/DicomNodes.cfg. Press *Cancel* to leave the dialog and discard any changes.

Part II

Alphabetic Index of Ports

10 Amira

10.1 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 the port to remain visible even if the object the port belongs to is deselected.

Connections are special ports, allowing you 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, all commands of the port's base class are available (this is the same as for Amira modules and data classes) also. 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.

Two ports of the same type can be interconnected with the TCL command `connect`: if a port P0 is connected to a port P1, any new value applied to P0 is automatically applied to P1, and reverse. The syntax of the command is:

```
<modulename 1> <P0 name> connect < modulename 2> [<P1 name>]
```

If `<P1 name>` is not given, the command will take `<P0 name>` to retrieve the port of module 2.

Available ports for interconnection are of the following type :

- HxPortButtonList
- HxPortButtonMenu
- HxPortColorList
- HxPortFloatSlider
- HxPortFloatTextN
- HxPortGeneric
- HxPortIntSlider
- HxPortMultiMenu
- HxPortMultiOptions
- HxPortRadioBox
- HxPortRangeSlider
- HxPortTabBar
- HxPortText
- HxPortTextEdit
- HxPortToggleList
- HxPort3DPointList

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 the 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.

`connect`

Connects the port to another port of the same type.

10.2 Connection

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. 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.

10.3 MasterConnection

This port represents a special connection port which is provided by every data object. It allows connection of 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.

10.4 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 must put the viewer into interaction mode (press the **ESC** key). Then move the mouse over one of the dragger's crosshairs and press 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 when the dragger is shown and the viewer is in *selection mode* to cycle through *Append new point* and *Modify current selected point* mode.

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.



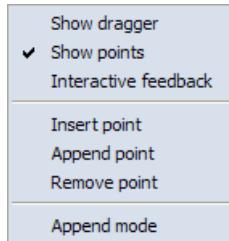


Figure 10.1: Popup menu for multiple points management.

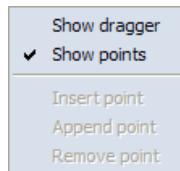


Figure 10.2: Popup menu for single point management.

This port is used for example by the *data probing* modules.

When you click on the *options* button, the following popup menu appeared:

The *Append mode* notify the *Port3DPointList* that each middle mouse button click will have to be interpreted as "add selected point to the end of the points list".

If the module which contains this port isn't able to handle more than one point (like the *point probe* module), *Append mode* and *Interactive feedback* options will be hidden :

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 numbers, 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 is 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.

displayInteractiveFeedback {0|1}

Display or not the interactive feedback.

10.5 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>]`

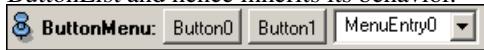
Check whether the specified button is snapped or not.

`wasHit [<index>]`

Returns 1 if the specified button was hit or if its snap toggle is down.

10.6 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 *DemoMaker*.

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>.

`setOptLabels <menu> <list>`

Sets the labels of the option menu item.

`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.

10.7 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)

10.8 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.

```
setAlphaEnabled {0|1}
```

If alpha is enabled, the dialog shows the alpha channel and allows to change it.

```
getAlphaEnabled
```

Checks whether the alpha channel is editable.

10.9 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 object icon 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 by right-clicking the colormap area of the port or by pushing the *Edit* button.

Some modules with this port in addition provide range sliders to conveniently adjust minimum and maximum of the data window. In order to facilitate adjustment the histogram of the connected data field is displayed as overlay on the colormap gradient. Minimum and maximum can be adjusted independently by dragging the respective handle along the intensity scale. Alternatively, the handles can be moved coordinately by dragging in the space between the handles. If minimum and maximum handle touch, i.e. $\text{minimum} = \text{maximum}$, the sliders can be moved by dragging the circle shown at the top of the handles.

The context menu item *Adjust to data range* allows you to automatically adjust the colormap range to the connected data range.

If the connected data provides a histogram, the context menu shows three additional items:

- Adjust to histogram: The colormap range is adjusted so that the minimum is set to one percent of the data values and the maximum is set to 99 percent. Both *Adjust to data range* and *Adjust to histogram* won't change the current zoom of the colormap display, unless the minimum or maximum values aren't covered by the colormap display. In that case the zoom of the colormap display will be adjusted so that the minimum and maximum values are covered by the colormap display.
- Zoom in range: The colormap display is zoomed so that the whole colormap display covers the range currently adjusted by the minimum and maximum values. Minimum and maximum values remain the same.
- Show full range: The colormap display is zoomed so that the entire data range is covered by the colormap display. Minimum and maximum values remain the same.

In addition the colormap display can be zoomed using the mouse wheel.

If no colormap is connected to the port, still a default color and a default alpha value can be defined. This corresponds to the *Constant* item in the context menu. 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.

Colormaps present (yet possibly hidden) in the Pool can be selected in the *Loaded colormaps* item. You can also use *Load colormap...* to get a colormap from a specific directory. When a colormap is selected, it can be edited. To that end, click on *Edit colormap...* in the context menu and the *ColormapEditor* will appear.



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 disable 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.

`getLocalMaxValue`

Returns 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.

`getMinValue`

Returns the lower bound of the range used for mapping data values to colors. If local range is enabled or if no colormap is connected the the lower bound of

the local range is returned. Otherwise the the lower bound of the range of the connected colormap is returned.

`getMaxValue`

Returns the upper bound of the range used for mapping data values to colors. If local range is enabled or if no colormap is connected the the upper bound of the local range is returned. Otherwise the the upper bound of the range of the connected colormap is returned.

`setNumColumns <columns>`

Sets the number of digits of the minimum and maximum text fields (default is 8).

`getNumColumns`

Returns the number of digits of the minimum and maximum text fields.

10.10 PortDoIt

This port is almost exactly the same as *PortButtonList*. The primary difference is that all buttons are initialized to have a snap toggle and that by default it has only one button.



or



This port is mainly used by compute modules.

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's control panel, open the *Edit/Preferences* dialog, click on the *Layout* tab, then check the *Show "DoIt" buttons* box.

Note also that the *Apply* button corresponds to the first button of the *DoIt* port. If a *DoIt* port has multiple buttons, in order to display all of them, you must check the *Show "DoIt" buttons* box as described above.

Commands

Inherits all commands of *PortButtonList*.

10.11 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`).

10.12 PortFilename

This port is used to define a file or directory name. The 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 five modes, one allowing to select any file, one allowing to select one existing file only, one allowing to select multiple existing files, and two modes to specify a load or save directory. However, if the 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. A value of 3 is returned if a directory can be set for loading, a value of 4 if a directory for saving can be selected (see description above).

```
setMode {0|1|2|3|4}
```

Sets the mode of the file browser (0 for *any file*, 1 for *existing file*, 2 for *multiple files*, 3 for *load directory*, 4 for *save directory* (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.

10.13 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),

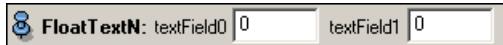
```
hasSubRangeButtons
```

Checks whether the subrange buttons are enabled or not.

10.14 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. In-sensitive 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.

10.15 PortFontSelection

This port provides an access to a font object which can be selected via a font selection dialog. Another button allows selection of the color of the current font. Currently, the *Strikeout* and the *Underline* options of the font selection dialog are not supported.



This port is used for example by the *Annotation* module.

Commands

Inherits all commands of *Port*.

`getFontName`

Returns the selected font name.

`getFontSize`

Returns the selected font size.

`isBoldFont`

Returns true if the font is bold.

`isItalicFont`

Returns true if the font is italic.

`getFontColor`

Returns the current font color.

`setFontName <FontName>`

To set the current font name.

`setFontSize <FontSize>`

To set the current font size.

`setBoldFont <0|1>`

To set a bold font or not.

`setItalicFont <0|1>`

To set an italic font or not.

`setFontColor <R> <G> `

To set the current font color.

10.16 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 Option* 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}
```

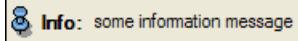
Sets the sensitivity of an element. If an element is insensitive, it is grayed out and accepts no user input. By default, all new elements are sensitive.

```
getSensitivity <id>
```

Returns the sensitivity of an element.

10.17 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`.

10.18 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*.

10.19 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*.

10.20 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.

10.21 PortMultiMenu

This port provides up to three 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.

10.22 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 *Port-MultiMenu* 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.

10.23 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*.

10.24 PortSharedColormap

This port provides a connection to a *colormap* object. Like the *PortColormap*, this port has a user interface and the contents of the connected colormap as well as its range are shown. The port also has an Edit context menu (that pops up when pressing the right mouse button over the colormap area of the port).

In contrast to the *PortColormap*, the shared port has the property to propagate the colormap connected to an object to all other objects that have a *PortColormap* and that are connected to the object possessing the shared port.

If a colormap is connected to the object, it appears in the port. Otherwise, only the Edit button is visible. The Edit context menu items are similar to the ones in the PortColormap, except for the *Disable colormap* item, which leaves the object connected to no colormap. There is no concept of *Constant* colormap for the PortSharedColormap.



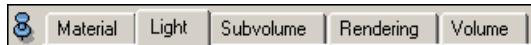
See also: PortColormap, Colormap, ColormapEditor

Commands

Inherits all commands of PortColormap.

10.25 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 Volren.

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.
```

10.26 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.

10.27 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 is 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.

`step [-f(forward) |-b(ackward)]`

Plays the animation by one time step forward or backward.

`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.

10.28 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.

`isToggleVisible <index>`

Checks whether toggle button <index> is visible or not.

`setToggleVisible <index> {0|1}`

Show or hide toggle button <index>.

Part III

Alphabetic Index of File Formats

11 Amira

11.1 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 Amira user's guide (*Scripting* chapter). 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`.

11.2 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.

11.3 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.

Note on binary encodings Data contained in an *AmiraMesh* file can be encoded in ascii or in binary format (see below). For binary encodings both big-endian mode and little-endian mode are supported. The default mode for *AmiraMesh* binary export was big-endian in previous product versions, but has been changed to little-endian in Amira 5.0. The behaviour can be configured using the following Tcl command:

```
amiramesh -endianess {big|little|native}
```

If you prefer to export AmiraMesh binary files in big-endian mode, you can add the above Tcl command with the argument `big` in the `Amira.init` file in the Amira installation directory under `share/resources`.

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 specified in this line, all data arrays are stored in plain ASCII text. If the tag `BINARY` is specified, the data arrays are stored in IEEE big-endian binary format. If the tag `BINARY-LITTLE-ENDIAN` is specified, the data arrays are stored in IEEE little-endian binary format. The header section of an AmiraMesh file is always stored as ASCII text, even if the data itself is binary encoded.

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*
- *Spatial Graph*

11.3.1 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 $(xmax-xmin)/(dims[0]-1)$. For degenerated 3D data sets with one dimension being 1 choose equal minimum and maximum coordinates in that direction.

11.3.2 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 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 $(xmax-xmin)/(dims[0]-1)$. For degenerate 3D data sets with one dimension being 1, choose equal minimum and maximum coordinates in that direction.

11.3.3 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.

11.3.4 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.

11.3.5 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.

11.3.6 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.

11.3.7 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.

11.3.8 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.

11.3.9 Spatial Graph

The data type *Spatial Graph* is used to store data organized in three dimensional networks. *Points*, *Nodes*, and *Segments* are the three basic structural components used in this module. The *Nodes* are the branching points and endpoints of the network, and the *Segments* are the lines connecting the nodes. *Segments*, in turn, consist of *Points* describing the three-dimensional course of a segment. Each *Point* is identified by spatial coordinates. A set of Segments connected by Nodes will be termed a Graph and a SpatialGraph data object can store even several Graphs. One or more scalar data items can be optionally stored for each network item, while labels can be associated only with *Nodes* and *Segments*.

In order to store Spatial Grpah sets in an AmiraMesh file, three 1D arrays have to be defined, namely *EDGES*, used to store the Segments indices, *VERTEX*, used to store the Nodes coordinates, and *POINT*, for the Points. Scalar values can be attached to Segments, Nodes and Points, while labels and label groups only to Segments and Nodes. Multiple values should be distinguished by denoting them Data2, Data3, and so on. Here is an example with 5 Nodes, 4 Segments and 242 Points, with one label (LabelGroup0) associated to the Segments and one scalar (Data0) to the Points:

```
# AmiraMesh 3D ASCII 2.0

define VERTEX 5
define EDGE 4
define POINT 242

Parameters {
```

```
LabelGroup0 { Id 0, Color 0.8 0.16 0.16 }
ContentType "HxSpatialGraph"
}

VERTEX { float[3] VertexCoordinates } @1
EDGE { int[2] EdgeConnectivity } @2
EDGE { int NumEdgePoints } @3
EDGE { int LabelGroup0 } @4
POINT { float[3] EdgePointCoordinates } @5
POINT { float Data0 } @6

# Data section
@1      # Coordinates Nodes
132.078 164.902 101.783
141.066 186.003 116.899
176.625 138.721 111.86
141.066 187.176 117.907
152.788 199.68 130

@2      # To which Nodes the Segments belong
0 1
2 1
1 3
3 4

@3      # Number of Points belonging to Edges
59
143
4
36

@4      # Label of the Edges
0
0
0
0

@5      # Coordinates Points
```

```
132.078 164.902 101.783
132.354 165.293 102.499
132.619 165.684 103.136
...
@6      # Scalar associated Points
0.345
0.463
0.865
...
```

Once the Nodes are defined, the Segments follow using Nodes indices as shown above. For each Segment the number of included Points and their coordinates should be specified. The index of the first item is 0. An index value of -1 indicates that a item should be terminated.

11.4 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.

11.5 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.

Note that if the input contains values that cannot be represented as numbers (NaN, Not a Number) a label field is created as additional output. The label Inside will indicate the region with valid data.

11.6 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.

11.7 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 the TIFF section.

11.8 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*.

11.9 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.

11.10 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/HxIndexDemo.html`.

11.11 Hoc

NEURON is a simulation environment for developing models of single neurons and networks of neurons. *Hoc* is a user extendable language, which is used in *NEURON* as scripting language. The file extension `.hoc`, therefore, refers to script files used in *NEURON*.

Only a set of commands defines the geometry and topology of a neuron model in the `.hoc` files. Just a restricted subset of keywords is actually required to reconstruct a basic spatial model. This reader scans the *hoc* file, identifies the commands and reads the parameters:

- `pt3dclear` - Destroy the 3d location info in the currently accessed section.
- `pt3dadd` - Add the 3d location and diameter point at the end of the current `pt3d` list.
- `connect` - The first form connects the section at end 0 or 1 to the currently accessed section at position `x`.
- `create` - Create a list of section names.
- `access` - Declare the default reference section.

The above commands are specific for the NEURON environment, please refer the NEURON documentation for a detailed code reference.

Once the data are loaded, a new *SpatialGraph* object is created in the pool and the diameter data are stored in the "thickness" scalar attribute. The values of "thickness" are read together with the coordinates when Amira identifies the command `pt3dadd` in the *hoc* file.

Amira is also able to automatically read topological information and convert them into label attributes. If the network described in the *hoc* file is organized in a correct tree-like form, Amira reads the hierarchical dependencies and stores them in the label attribute "Ranks". Additionally, the name of the sections will be read automatically, if the *hoc* file is organized in the following format:

```
{create <section1> [N] }  
{create <section2> [M] }
```

```
<...>

{access <section1>[N1] }
<...>
{access <section1>[N2] }
<...>

{access <section2>[M1] }
<...>
{access <section2>[M2] }
<...>
```

In this specific format Amira is able to recognize the N elements belonging to *section1*, the M elements belonging to *section2*. The topological information are stored in the "topology" label attribute.

In the following lines you can find a simple example, you can copy and paste it in a text file ending with the extension ".hoc" and load it into Amira. This *hoc* script describes one soma and two dendrites, one of them has a bifurcation.

```
{create soma[1.]}
{create dendrite[4.]}

{access soma[0.]}
{pt3dclear()}
{pt3dadd(0.,0.,1.,10.0)}
{pt3dadd(0.,0.,0.,10.0)}

{soma[0.] connect dendrite[0.](0), 1}
{access dendrite[0.]}
{pt3dclear()}
{pt3dadd(0.,0.,0.,10.0)}
{pt3dadd(68.,73.,-75.,5.0)}

{dendrite[0.] connect dendrite[1.](0), 1}
{access dendrite[1.]}
{pt3dclear()}
{pt3dadd(68.,73.,-75.,5.0)}
```

```

{pt3dadd(100.,100.,-100.,8.0) }

{soma[0.] connect dendrite[2.](0), 1}
{access dendrite[2.]}
{pt3dclear()}
{pt3dadd(68.,73.,-75.,5.0)}
{pt3dadd(103.,108.,-150.2,5.0)}

{soma[0.] connect dendrite[3.](0), 1}
{access dendrite[3.]}
{pt3dclear()}
{pt3dadd(0.,0.,0.,10.0)}
{pt3dadd(10.,200.,100.,8.0)}

```

You can also save the *SpatialGraph* object in a *hoc* file, but you have to perform a "Set Root" operation before (see *Filament Editor*). The "Set Root" operation ensures that the network does not contain loops. Amira recognizes the ranking labels and writes the segments in such a way that the coding requirements expected in the NEURON environment will be satisfied.

11.12 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 hashmark (#) 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 `Vertices` 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 Materials3
Patches 2
    3   -5
}
```

11.13 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 structure of an *Icol* format can be immediately understood by looking at the examples in Amira's demo data directory.

11.14 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.

11.15 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.

11.16 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.

11.17 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 *Calculus-Matlab* in order to load, save, and execute M-files.

11.18 Nifti

Nifti is a file format for the exchange of neuro imaging data.

With this reader and writer you can import and export files in the Nifti format in Amira. Not all possible Nifti formats are supported (see Limitations).

Nifti data objects can 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.

A dedicated file extension .nii for storing the header and the data in the same file is available. This is the default format for Nifti file export. Optionally, the file

can be compressed. In order to enable compression for file export, you need to choose a file name with the extension `.nii.gz` in the Amira file dialog.

The reader will scan the data directory for an additional file with the extension `.atlas.xml`. This file is assumed to contain additional XML encoded information about the names and colors of materials stored in the nifti data file. Such a file will also be written if a LabelField is exported as Nifti file.

Limitations:

- Only data sets with up to 4 dimensions are supported. `nu`, `nv`, `nw` have to be 0 or 1. The time steps are imported as a dynamic time series in Amira.
- Supported data types are: unsigned char, signed short, unsigned short, signed int, unsigned int, float, double, RGBA colorfield (read only).
- If the `scl_slope` variable is defined `scl_slope` and `scl_inter` will be used to scale the data while reading it. Please note, that this may introduce a loss of precision depending on the data type.
- Due to the way coordinates are encoded in a Nifti file, the origin of the bounding box of an imported data set will be always set to (0,0,0). Any translations will be encoded in the object's transformation matrix.
- Only 3D scalar image with uniform coordinates can be exported in Nifti format.
- Currently only `.nii` and `.nii.gz` files can be exported, but not `.hdr` and `.img` files.

11.19 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, `Amirasurfaces` can be exported in Open Inventor format.

11.20 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.

11.21 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.

11.22 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 *Cluster*.

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.

11.23 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.

11.24 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:

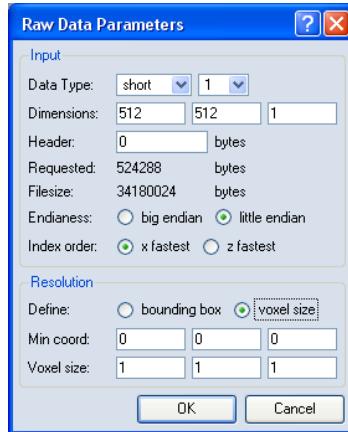


Figure 11.1: Amira's raw data read dialog.

- **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/MaxCoords.** 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
- **Endianess.** 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 Endianess IndexOrder  
  DataType nDataVar dimX dimY dimZ  
  xMin xMax yMin yMax zMin zMax
```

where

- **FileName** is the name of the file to load,
- **Endianess** 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 endianess, 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.

11.25 Raw Data as LargeDiskData

This file format allows a subvolume to be loaded from 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* object. The file is opened readonly. You cannot change the image data, nor add or modify parameters.

11.26 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 the TIFF section.

11.27 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.

11.28 SWC

The *SWC file format* refers to the *Cvapp* software application developed by Robert Cannon at the University of Southampton. *Cvapp* allows editing of neuron morphologies, cleaning up and optimizing morphology files, and the interchanging of various morphology file formats between simulation platforms (e.g. between Neurolucida to NEURON or GENESIS and *vice versa*).

The SWC format expresses each point of a network on a separate line by seven quantities:

- an integer code for the point
- a code for its type (axon, apical dendrite etc)
- three spatial coordinates

- the diameter
- the code of its parent (the next point toward the soma)

11.29 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 (#).

11.30 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). 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 YCbR 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 11.2.

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

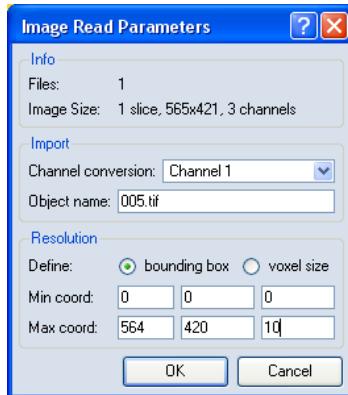


Figure 11.2: Amira’s channel conversion dialog.

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*.

If the reader finds a tag defined by FEI (www.fei.com) in the TIFF file, an additional dialog is displayed that allows the user to adjust the stage tilt value. This adjustment will change the voxel size in the y-direction to correct for an artifact of image acquisition. If the user presses Skip, no adjustment is done.

The user can set a default value for the tilt correction by defining an environment variable called "AMIRA_STAGE_TILT_CORRECTION_DEFAULT".

11.31 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. ($I>1 \&& J>1 \&& K>1$). 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 Hx-Hexahedron 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.

11.32 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.

11.33 Vevo Mode Raw Images

This reader adds support for images acquired using a Vevo 770 ultrasound scanner from Visual Sonics. The reader imports the **B-Mode**, 3D **B-Mode** and 3D **PW Doppler** images exported in the .rdi / .rdb raw data file format of the scanner.

To load the data, select either the .rdi or the .rdb file in the *File/Open data ...* dialog. **B-Mode** images will be imported as a 2D slice, 3D **B-Mode** images will be imported as single channel uniform scalar fields while and 3D **PW Doppler** images are imported as 2-channel fields. In the latter case the **B-Mode** image is assigned to the first channel and will be connected to a gray colormap and the **PW Doppler** image is assigned to the second channel and is connected to the doppler.am colormap. Both colormaps, if they do not already exist, will be loaded into the workspace from the /data/colormaps directory.

11.34 Wavefront Technologies 3D Geometry (.obj)

This file format was developed by Wavefront Technologies and defines a geometry in 3D space. The open file format is used by various computer graphics software. It supports description of vertex positions, texture coordinate vertex positions, normals and polygon faces defined as a list of vertices and texture vertices. Counter-clockwise storage of vertices eliminates the need of explicit declaration of normals.

12 Molecular Option

12.1 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 simultaneous)
- *.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 simultaneous)

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/>

12.2 AMF

AMF is an XML based data description format for molecular data in Molecular Option.

It is able to store all data contained in the Molecule data-structure in a human readable format. It thus should be the format of choice if you want to edit add or remove levels, attributes, or observables by hand.

By default, AMF stores the xml file in gzip compression. If you want to edit the contents of the file, you may decompress it with the gzip program. To do so, you first need to rename the file to end on gz:

```
mv Molecule.amf Molecule.amf.gz  
gzip -d Molecule.amf.gz
```

Both uncompressed and compressed files can be loaded. Amira detects amf files by the ending and the file header.

AMF can be used to store 3 different data types which will be described in the following sections:

- Molecule
- MolTrajectory
- MolTrajectoryBundle

Each of these entries, which is encountered in a file, will be loaded into a separate data module in Amira.

AMF stores the topology of each structure by defining levels (e.g. atoms and bonds) and groups in that level (e.g. a single bond in the bond level). Each level has a reference level (bonds reference atoms) and each group contains indices of groups of the reference level (e.g. a bond contains two atom indices). Attribute values are defined in the order of the groups in the level. Note that unlike in the rest of Amira where counting of level, group and attribute indices starts with 1, in AMF each index starts with 0, therefore indices will be shifted by -1 compared with what you see after loading it into 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. This is a special case and the GROUP element can be omitted. Bonds always reference two atoms and are therefore of fixed group size 2. The indices can therefore be read as consecutive groups of

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). These levels represent the 3 possibilites of defining a level: Variable group size means that the number of group elements occurs in the list before each group, fixed group size of n means that each group is an n-tuple in the list, and the atom level is a special case because it is the root level and atoms do not reference any other groups. The coordinates are in a different section and the list contains consecutive 3-tuples (x,y,z) for each atom in the order of their index.

```
<?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. The list starts with the first timestep and gives all coordinates and continues with the second timestep and so on. 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.

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 AMP (MOLTRAJECTORYBUNDLE*, MOLTRAJECTORY*, MOLECULE*)>
<!ATTLIST AMP
    version CDATA #REQUIRED>

<!ELEMENT MOLTRAJECTORYBUNDLE (MOLTRAJECTORY*)>
<!ATTLIST MOLTRAJECTORYBUNDLE
    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
    reflevel 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>
```

12.3 DX

The *dx* file format contains the electrostatic field generated by APBS [1-2]. The reader was written to work with the output of APBS version 0.5.1.

References

- 1 <http://apbs.sourceforge.net>
- 2 Holst M, Baker N, Wang F. Adaptive multilevel finite element solution of the Poisson-Boltzmann equation I: algorithms and examples. *J Comput Chem*, 21, 1319-1342, 2000.

12.4 GROMACS

Gromacs is a format consisting of three different file formats for storing structures of biomolecules and molecular simulations such as 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>

12.5 MAP

The *map* file format contains the interaction grids generated by Autogrid which is part of the Autodock suite [1-2]. The import routine was written for output generated by Autogrid version 4.0.

References

- 1 <http://autodock.scripps.edu>
- 2 Huey, R., Morris, G. M., Olson, A. J. and Goodsell, D. S. (2007), A Semiempirical Free Energy Force Field with Charge-Based Desolvation J. Computational Chemistry, 28: 1145-1152.

12.6 MDL

The *sdf* file format is part of the set of MDL file formats used for saving one or multiple small chemical structures [1].

Amira uses and writes the information of the atom and bond blocks and data fields. Molecules with more than 999 atoms cannot be saved as sdf.

If more than one molecule record is found when reading a file, the data is loaded as a TrajectoryBundle. Saving a TrajectoryBundle as sdf will write one molecule for each trajectory by using the coordinates of the first timestep.

References

- [1] http://en.wikipedia.org/wiki/SD_format

12.7 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 HETATOM 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 HETATOM 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 modeling 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/>. If you have an active network connection and know the PDB-Id, you can also load the structure directly from rcsb.org. To do so use the Tcl command

```
newMolFromPDB <PDB-ID>
```

where PDB-ID is the ID of the structure (for example 1HVR). The command works asynchronously. Depending on file size and bandwidth the process may take a few seconds up to several minutes. The requested pdb entry will appear as a new object in the object pool.

12.8 PHI

The *phi* file format is generated by the Poisson Boltzmann solver of CONGEN [1-2]. The file import routine was written for output generated by Congen version 2.1.2178.

References

- 1 http://www.congenomics.com/congen/congen_toc.html
- 2 R. E. Brucoleri "Application of Systematic Conformational Search to Protein Modeling", Molecular Simulations 10, 151-174 (1993)

12.9 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.

12.10 Tripos

The *mol2* file format is used to save Tripos Sybyl molecule information [1]. If more than one molecule record is found when reading a file, Amira loads the data as either a Trajectory (if all molecules have the same number of atoms and bonds), or as a TrajectoryBundle (if at least 2 molecules differ in their number of atoms or bonds). Amira uses and writes the information given in the molecule atom and bond records.

References

[1] <http://www.tripos.com/data/support/mol2.pdf>

12.11 UniChem

The UniChem file format (extension *uni*) is used by the UniChem molecular modeling 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.

12.12 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 **typebase** 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 inter-

preted 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*.

13 Very Large Data Option

13.1 LDA

This is the native VolumeViz file format for storing hierarchical multi-resolution volume data. 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.

13.2 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.

13.3 Stacked-Slices as LargeDiskData

Stack of image files can be loaded as LargeDiskData. 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 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.

14 Mesh Option

14.1 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`.

14.2 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.

14.3 Abaqus format

The *Abaqus* file format (<http://www.simulia.com/>) is a file format used by the Abaqus program (version 6.7) to encode CAD/CAM models and FEM simulation results. The format is able to store different data entities and FEM cell types. In Amira tetrahedral meshes can be exported together with sampled density fields.

The file name extension of this format is .inp. For *Abaqus* files with some other extension the file format has to be specified manually via the popup menu of the file dialog.

A special mode is enabled if density fields are saved in the .inp format. The data attached to the nodes is binned given its maximum and minimum value. For each bin a different material is created. The number of bins (default 20) can be controlled with a TCL variable ABAQUS_NUM_BINS which needs to be defined before exporting in the console window. If the TCL variables ABAQUS_MIN_VALUE and ABAQUS_MAX_VALUE are set to some floating point values these values will be used as the minimum and maximum value of the histogram. Values outside the defined range will be assigned to the first and last bin.

14.4 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.

14.5 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 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.

14.6 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.

14.7 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 data set 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.

14.8 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 -plot3dggrid` 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 (brackets 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 (brackets indicate one record):

(Q1 is density (RHO), may be complex)

[q11, q12,...,q1NPOINTS,

NPOINTS floating-point Q2 values:
(Q2 is x momentum (RHO*U), may be complex)

q21,q22,...,q2NPOINTS,

NPOINTS floating-point Q3 values:
(Q3 is y momentum (RHO*V), may be complex)

q31,q32,..., q3NPOINTS,

NPOINTS floating-point Q4 values:
(Q4 is z momentum (RHO*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]
```

15 Microscopy Option

15.1 Bio-Rad Confocal Format

The Bio-Rad confocal file format is used to store 3D image data from confocal microscopy. It essentially consists of a 76 byte header section followed by the image data in big endian raw format. Amira recognizes Bio-Rad files automatically by the suffix `.pic`. In order to load Bio-Rad files from the command line use `load -biorad <filename>`.

Since the header section of the format doesn't contain full information about the voxel size, the bounding box of the 3D image has to be adjusted manually for the resulting uniform scalar field using Amira's *crop editor*. Note that Bio-Rad confocal files can only be read but not be written by Amira.

15.2 FEIStackedScalarField3

Inherits of *StackedScalarField3*. See also file format *MRC*.

15.3 FEIUniformScalarField3

Inherits of *UniformScalarField3*. See also file format *MRC*.

15.4 Leica 3D TIFF

This reader is able to read 3D TIFF files containing a whole stack of 2D images. In particular, the 3D TIFF format is used by newer Leica laser scanning microscopes. In addition to the image data itself special parameters like pixel size or slice distances may be stored in a 3D TIFF file. If such information is found it will be interpreted in order to create a uniform scalar field of the proper type. However, if no bounding box information is encountered, the *channel conversion dialog* described in the 2D TIFF section will be popped up.

15.5 Leica Binary Format (.lei)

This is the Leica binary file format used by Leica laser scanning microscopes. It consists of an *lei* file as well as several TIFF slices. In order to read these files, select only the *lei* file. Parameters like pixelsize or slice distance are read from the *lei* file.

15.6 Leica Image Format (.lif)

The *Leica Image File (LIF)* format is used by Leica's LAS AF software which is shipped with all Leica laser scanning microscopes starting with May 2006. It consists of a single binary XML files that contains one or multiple image data objects. Selecting a *.lif file in the *Open data ...* file browser brings up a dialog that lists all data sets contained in the file. The user may highlight one or multiple entries and click OK to load the corresponding images, volumes, or series into the Amira workspace. Currently the LIF import supports the following image types: 2D images, z-stacks, 2D time lapse series and 3D (z-stacks) time lapse series.

15.7 Leica Slice Series (.info)

This is the file format used by the older Leica laser scanning microscopes. It consists of an *info* file as well as several raw or TIFF slices, which all must reside in the same directory. In order to read these files, select only the *info* file. Parameters like pixelsize or slice distance are read from the *info* file. If the file contains colormaps, they will be read, too.

15.8 MRC

MRC is a file format for the exchange of electron microscopy data.

With this reader you can import and export files in the MRC file format into Amira. Recognized file extensions for the reader are .mrc, .rec and .ali.

This reader and writer supports only the extended header of FEI (www.feicompany.com). FEI MRC files are detected by an existing extended header and either one of the following two strings: "Fei Company", "FEI Electron Optics" in the first label of the standard MRC header structure.

15.9 Metamorph STK Format

The MetaMorph Stack (STK) file format is used to encode 3D image data, e.g. from confocal microscopy. It is a special version of the TIFF file format. Thus, STK files are indicated as TIFF files in the format column of the file dialog.

STK files can be read just as ordinary 3D TIFF files. The *channel conversion dialog* is popped up, letting the user decide how to proceed with multiple channel images and letting him define the bounding box of the 3D image. Note that size hints stored in the STK file itself are currently not interpreted. Also note that Amira can only read but not write STK files.

15.10 Olympus (.oib/.oif)

The Olympus OIB/OIF file format is used by Olympus FluoView 1000 confocal microscopes to store single and multi-channel image stacks, single images, and time lapse series. Two variants of the format are available. One (file extension "oib") uses a single binary file to store images and meta information. Another (file extension "oif") stores all frames of an image stack as separate files in a directory hierarchy. A simple ASCII text file is used for storing the meta information. The latter version exists primarily to work around the 4GB file size limit on some operating systems. To import Olympus OIB/OIF data, load the .oib/oif file with *File->Open Data* from the main menu.

Note: Because this file format uses Microsoft OLE2 Compound Document Format this reader is only available on the Windows platform.

15.11 Zeiss LSM

This format stores 3D image data from Zeiss LSM(TM) confocal laser scanning microscopes in a single file. An LSM image stack can store up to four separate channels or an RGB(A) color.

16 DICOM Reader

16.1 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.

16.2 DICOM export

The DICOM data format and its predecessor the ACR-NEMA format are widely used to exchange medical image data, provided by various modalities, but also in many other areas dealing with volume image data. 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.

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.

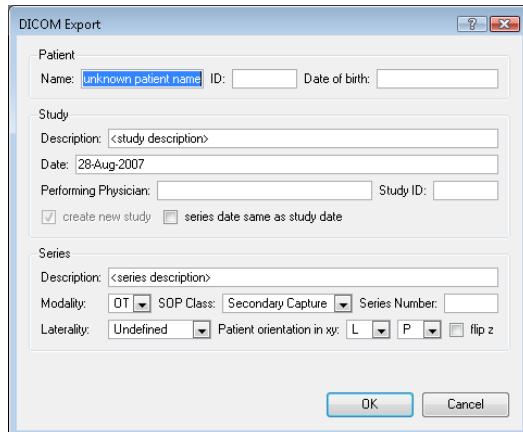


Figure 16.1: DICOM Export dialog.

16.3 DICOM import

The DICOM data format and its predecessor the ACR-NEMA format are widely used to exchange medical image data, provided by various modalities, but also in many other areas dealing with volume image data. DICOM stands for **Digital Imaging in Communications and Medicine**, 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.

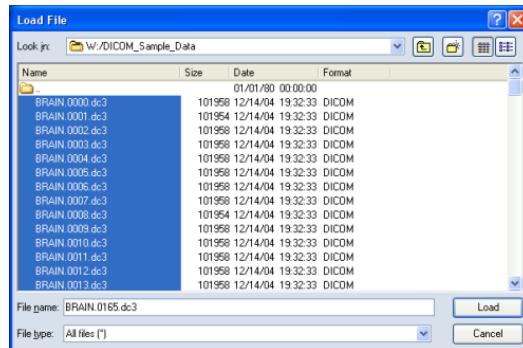


Figure 16.2: Selecting multiple DICOM files in the file dialog.

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 specialties. 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.

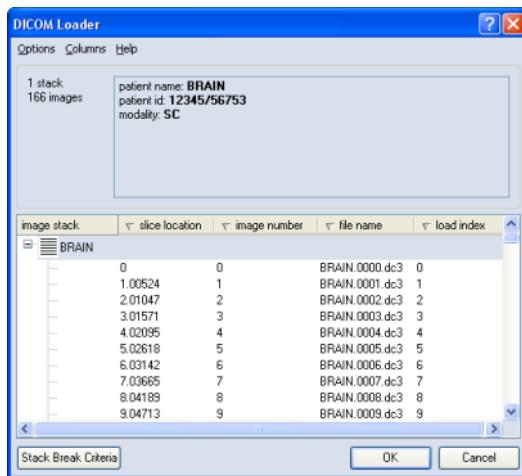


Figure 16.3: The DICOM load dialog.

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.

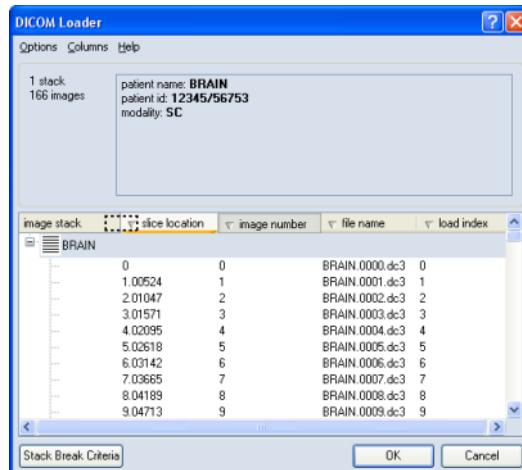


Figure 16.4: Reordering columns in the load dialog.

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 press the *Stack Break Criteria* button on the DICOM Loader.

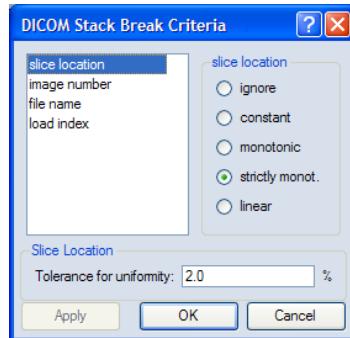


Figure 16.5: Dialog for specifying stack break criteria.

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 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.

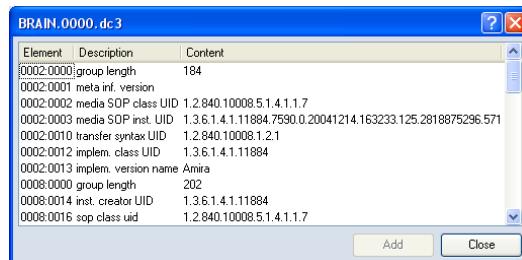


Figure 16.6: Dialog listing all DICOM parameters.

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 image distribution.

Part IV

Alphabetic Index of Data Types

17 Amira

17.1 AnnaScalarField3

This data class represents a user-defined 3D scalar field based on an arithmetic expression. It provides a port *Expression* by which an arithmetic expression depending on Cartesian coordinates x, y, z can be entered that defines the value for each point in space. The range of values with respect to the unit cube (default) domain is indicated as *Component Range*. A data object of type *AnnaScalarField3* has three additional input ports that can be connected to other data objects representing 3D fields, e.g., to image data objects. The predefined variables *a*, *b* and *c* are available for referencing such connected data objects in the arithmetic expression, this way a scalar field depending on other fields can be defined. Whenever an expression evaluation is triggered to compute the value for a point (x,y,z), the variables *a*, *b* and/or *c* 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.

If inputs contain undefined values, each of them will be used during the expression evaluation. If an undefined value is found, 0 will be used during this evaluation instead of the real undefined value. Therefore, the undefined value isn't propagated to the result.

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 *Field A*, *Field B* or *Field C* are connected to port *InputA*, *InputB* and *InputC* respectively, the following variables are also defined:

- **a:** the values of input object A

- **b:** the values of input object B
- **c:** the values 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 field.

InputB [optional]

Optional second scalar field.

Ports

Expr



Expr:

Input field for arithmetic expression defining the scalar field values.

Domain



Domain: unrestricted dependent field

This port is only visible if other fields are connected as input a, b or c. These input fields might not be defined everywhere. If set to *unrestricted* the field can be evaluated everywhere, a value of 0 is used for the input fields of a, b or c if evaluated outside their domain. If set to *dependent field*, the field is only evaluated inside all domains of the dependent fields a,b and c.

17.2 AnnaVectorField3

This data class represents a user-defined 3D vector field based on an arithmetic expression. It provides ports *X*, *Y*, *Z* by which arithmetic expressions can be entered that define the component mappings with respect to 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)*.

$Z(x,y,z)$ 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. 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*, , *cy*, *cz* for x-, y-, and z-components respectively of vector fields. This way a vector field depending on other 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*.

If inputs contain undefined values, each of them will be used during the expression evaluation. If an undefined value is found, 0 will be used during this evaluation instead of the real undefined value. Therefore, the undefined value isn't propagated to the result.

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

An input field with a small icon of a gear and the label "x:" followed by a text box containing the value "5".

Input field for arithmetic expression defining the x-component field values.

Y

An input field with a small icon of a gear and the label "y:" followed by a text box containing the value "10".

Input field for arithmetic expression defining the y-component field values.

Z

An input field with a small icon of a gear and the label "z:" followed by a text box containing the value "0.5".

Input field for arithmetic expression defining the z-component field values.

Domain

A button labeled "Domain:" followed by two radio buttons. The first radio button is checked and labeled "unrestricted". The second radio button is labeled "dependent field".

This port is only visible if other fields are connected as input a, b or c. These

input fields might not be defined everywhere. If set to *unrestricted* the field can be evaluated everywhere, a value of 0 is used for the input fields of a, b or c if evaluated outside their domain. If set to *dependent field*, the field is only evaluated inside all domains of the dependent fields a,b and c.

17.3 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*.

17.4 BlockStructuredGrid3

A data objects of type *BlockStructuredGrid* represents a mesh consisting of multiple blocks. Each block is a regular grid, see section *Coordinates and Grids* in chapter *Program Description* of the Amira user's guide.

17.5 BlockStructuredScalarField3

A data object of type *BlockStructuredScalarField* represents scalar data on a block-structured grid. The object pool needs to contain at least one appropriate *BlockStructuredGrid*.

Connections

Shared colormap [optional]

The colormap that is associated with this data.

Grid [required]

The *grid* that the scalar data is stored on.

Ports

17.6 BlockStructuredVectorField3

A data object of type *BlockStructuredScalarField* represents vector data on a block-structured grid. The object pool needs to contain at least one appropriate *BlockStructuredGrid*.

Connections

Grid [required]

The *grid* that the vector data is stored on.

Ports

17.7 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.

17.8 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 *ClusterStringLabels*.

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.

17.9 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 *Vortex* 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 $0.3*R + 0.59*G + 0.11*B$, 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 $0.3*R + 0.59*G + 0.11*B$.

`swapRGBA`

Swaps ABGR tuples into RGBA tuples or vice versa.

17.10 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.

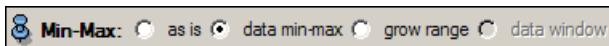
Ports

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* in chapter *Program Description* of the Amira user's guide).

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> <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.

`interp1 <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.

17.11 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.

`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>`.

`setDefaultFileFormat [format]`

Sets the default file format for the data object. The default file format is used when the object's save method is called without a format string, or when a data object needs to be stored in order to create a network script. The default file format will also be used to initialize the file format combo box of the file dialog when choosing "Save Data As..." from the main menu. For most native Amira data objects the default file format is the AmiraMesh format. The format itself should be specified by the format name (same name as shown in the file type combo box of the file dialog). If called with a parameter the method returns the current default file format.

`setEditor [editor]`

Attaches the given editor to the data object. If called without arguments the currently attached editor is detached. For example, the following command opens the parameter editor for the data object lobus.am: `lobus.am setEditor [create HxParameterEditor]`

`getEditor`

Returns the name of the editor currently attached to the data object. If no editor is attached, an empty string is returned.

17.12 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.

`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.

17.13 HexaGrid

A data object of type *HexaGrid* represents an unstructured finite-element grid composed of hexahedrons. The geometric information is stored in terms of vertices, edges, faces, and hexahedrons. Like a *LabelField* with its uniform hexahedral grid structure a hexahedral 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.

Commands

`hasMaterial <name>`

Returns true if the specified material is defined in the material section of the *HexaGrid*.

`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 hexahedrons from an other hexahedral grid into this one.

`removeHexa <n>`

Marks the hexahedral cell specified by `<n>` as obsolete.

`cleanUp`

Removes all obsolete hexahedrons from the grid.

`fixOrientation`

Fixes the orientation of all hexahedrons so that the enclosed volume is positive.

17.14 HexaScalarField3

This is a class of 3D scalar fields defined on hexahedral grids. See also its base class *ScalarField3*.

Ports

17.15 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*.

17.16 LandmarkSet

This data type represents specific points or markers in 3D space. It can be used to flag markers in 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 *Console Window* in chapter *Program Description* of the Amira user's guide.

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 moves 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.

```
computeLinearTransform [<src-set> <dst-set>  
<transform-type>]
```

Computes the linear transformation that moves 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. The possible types of linear transformations, specified by `<transform-type>`, are: 0=rigid (default), 1=rigid+iso-scale, 2=affine.

```
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.

17.17 LargeDiskData

A LargeDiskData object is useful for working with large image data. It allows you to extract subvolumes loaded as a normal field object. In this way Amira can manage data that is larger than main memory. Note that only uniform coordinates are supported.

A selection of file formats can be loaded as LargeDiskData: (*AmiraMesh*, *Raw Data*, *Stacked-Slices*, and *LargeDiskData*).

To access the data, you must 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.

17.18 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, compare section *Coordinates and Grids* in chapter *Program Description* of the Amira user's

guide. In Amira uniform, stacked, rectilinear, and curvilinear coordinates are supported. *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 Option 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 = rectilinear, 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.

17.19 Light

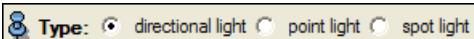
Light objects are used to define additional lights in the Amira viewer windows. Actually light objects are neither modules nor standard data objects. Nevertheless, they are displayed in the Pool. Like modules they provide some ports allowing the user to adjust the object's properties. In particular, 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 activation of 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.

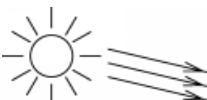
Ports

Type

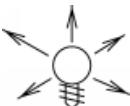


This radio box determines the light type. Three different types are supported:

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 a 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.

17.20 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 *Vertex Set* in chapter *Program Description* of the Amira user's guide. 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.

`removeDuplicatePoints [<tolerance>]`

Removes duplicate points in a line set. If tolerance is not specified, only exact matches are considered. Data values are taken from the last occurrence of a duplicate vertex.

`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>`

Smoothes 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.

17.21 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

 **Number of streams:** 

Select the requested number of streams. After this, the appropriate number of the following file name fields will appear.

Stream1

 **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

 **Stream2:** 

Identical to above for stream number 2.

Stream3

 **Stream3:** 

Identical to above for stream number 3.

Stream4

 **Stream4:** 

Identical to above for stream number 4.

Type

 **Type:** 

Select here how the *MoviePlayer* module has to interpret and render the final image sequence.

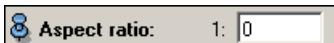
- *mono* - every image forms a single frame.
- *stereo* - two images form a stereo frame.
- *stereo(interlaced)* - every image forms a stereo frame. The left eye channel is stored in the even lines and the right in the odd lines. Internally the module resorts the lines to get an image of the type *stereo(up/down)* .
- *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



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 its *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



Force 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



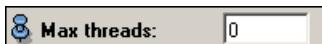
Flip the movie in X- and/or Y-direction.

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.

17.22 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.

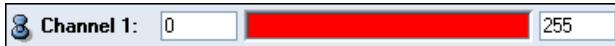
Connections

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.

17.23 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.

duplicate

Duplicates the object and returns the name of the duplicated object.

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>`

Undo the effect of the `clipGeom` command described above.

`setPickable <bool>`

Defines if a rendering module is pickable.

`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.

17.24 RawAsExternalData

This file format allows a subvolume to be loaded from 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* object.

17.25 RegScalarField3

This is a class of 3D scalar fields defined on regular lattices. See also *Lattice3* and its base class *ScalarField3*.

Ports

17.26 ScalarField3

This class is the base class for all 3D scalar fields in Amira. See also its base class *Field3*.

Ports

17.27 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 the *Scripting* chapter in the Amira user's guide). 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* in the *Scripting* chapter of the Amira user's guide). 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.
- `$this proc compute {} { ... }` defines a method that is called whenever a user interface component of the script object is changed by the user. See examples below.
- `$this proc savePreparation {<datadir> <savingFlags>} { ... }` defines a method that is called before a session is stored in a network file. So some clean up can be done here.

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

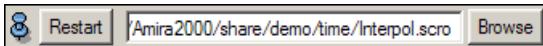
Connections

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 *filename port*.

`newPortFloatSlider <name>`

Creates a new *float slider port*.

`newPortFloatTextN <name> <number-of-fields>`

Creates a new *float text port*.

`newPortMultiMenu <name> <num-options-1>`

`[<num-options-2> [<num-options-3>]]`

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.

```
setVarSerialize <variable> 0|1
```

Variables created with `setVar` are usually serialized when storing a session in a network file. This behavior can be disabled for variables whose values are only meaningful within one session.

```
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 command 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.*

17.28 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*, animated using a *TransformAnimation* module, or modified 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 numbers are returned. The first three numbers 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 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 numbers 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.

`hasUndefinedValue`

Indicates if an undefined has been set or not.

`getUndefinedValue`

Return the undefined value. If `hasUndefinedValue` returns 0, the result of this command is undefined.

`setUndefinedValue <undefinedValue>`

Set the undefined value. This method should be called immediately after having read associated data. Normally, the reader should do this job.

`hasDataWindow`

Indicates if a data window has been set or not.

`setDataWindow <min> <max>`

Set the data window.

`removeDataWindow`

Remove the data window.

`getRange [-d|-w]`

This method is provided here for convenience because many spatial data objects (although not all) contain data values for which min/max values can be computed. If such data is available the minimum and maximum data component is computed. For fields with more than one data variable, for example vector fields, this is not the magnitude range of the field. Details of the behavior might depend the specialization of this module and would be noted there.

If the spatial data object didn't contain any data, result values are 1 0 (The minimum value will be greater than the maximum).

The optional argument `-d` indicates that the returned values do not include the undefined value.

The optional argument `-w` indicates that the returned values correspond to the data window. If no data window is found, the returned values correspond to the raw data range.

`touchMinMax`

This method is provided here for convenience because many spatial data ob-

jects (although not all) contain data values for which min/max values can be computed. Invalidates cached min and max values.

17.29 SpatialGraph

The *SpatialGraph* data type is designed to store data that can be represented as curved lines in 3D space and that are eventually organized in networks of multiple such lines.

Terminology: Branching or endpoints of the network are called *Nodes*, the curved lines connecting nodes are called *Segments*. The three dimensional course of a segment is given by a sequence of *Points* in 3D space. A set of segments connected by nodes will be termed a *graph*, and a *SpatialGraph* data object can store several graphs.

For each segment, node, and point one or more scalar data items can be stored. Nodes and segments can, in addition, be annotated with labels. Labels may be grouped in *label groups*, so that a set of several labels can be used to tag structures that are conceptually connected. In order to visualize the network of a SpatialGraph object the module *SpatialGraphView* can be used. *SpatialGraph* objects can be stored using the *AmiraMesh* file format. The *Filament Editor* sub-application is the dedicated tool to edit *SpatialGraph* objects.

Commands

```
merge <spatialgraph2>
```

Merges spatialgraph2 into the spatial graph.

17.30 SpreadSheet

This data type represents a spreadsheet. A spreadsheet will be created e.g., by the module *MaterialStatistics*.

17.31 StackedLabelField3

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.

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.

17.32 StackedScalarField3

This is a class of 3D scalar fields consisting of parallel slices. See also its base class *RegScalarField3*.

Ports

17.33 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. 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 a 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 exists already, one will be created.

merge <surface2>

Adds surface2 to the surface. A return value of 1 indicates success.

refine

Refines the surface by subdividing all edges. After this operation the surface will contain four times the number of triangles.

getPatchArea

Returns the surface area of a given patch.

assignInsideMaterial <patchID> <ExteriorMaterialID>

Assigns a new inside material to a patch.

assignOutsideMaterial <patchID>

<ExteriorMaterialID>

Assigns a new outside material to a patch.

17.34 Surface Path Set

This module represents one or more paths on a surface, consisting of one or more nodes connected by line segments. The three types of nodes are:

- **Vertex nodes:** can only lie on a vertex of the surface
- **Edge nodes:** must lie on an edge of a triangle
- **Triangle nodes:** can lie everywhere on the surface

Paths can be created using the *Surface Path Editor*. The idea is to define points on the surface, so called **control points** which are connected to each other via path nodes using various strategies like Dijkstra or shortest geodesic.

Connections

Surface [required]

A path set must be connected to a *Surface* on which it is defined.

17.35 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 *Creating a Tetrahedral Grid from a Triangular Surface* in the *First steps in Amira* chapter of the Amira 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.

17.36 TetraScalarField3

This is a class of 3D scalar fields defined on tetrahedral grids. See also its base class *ScalarField3*.

Ports

17.37 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.

17.38 UniformLabelField3

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.

17.39 UniformScalarField3

This is a class of 3D scalar fields defined on a uniform lattice. See also *Lattice3* and its base class *RegScalarField3*.

Ports

17.40 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 *Vertex View* 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 change 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.

17.41 ViewBase

This module is the base class for several other Amira modules displaying a set of triangles, like *Isosurface*, *Surface View*, *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 a 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

ColorField

Scalar field to be visualized via pseudo-coloring. The field will be evaluated at the vertex positions of the surface and a color will be mapped onto the surface using the colormap connected to connection port *Colormap*. 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

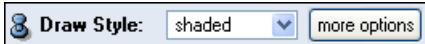
Colormap used for pseudo-coloring (see *Color Field*). To change the colormap right-click the colormap window or reconnect (drag) the blue connection line with another colormap data object. To change the port's default color double-(left-)click the colormap window and select a color from the *Color Dialog*. See also *Colormap*.

Texture [optional]

This port allows connection of a 2D image that will be mapped onto the surface as a texture. The image must be a single slice of type *ColorField*. A transformation can be applied to the texture image in order to change the texture projection axis and texture coordinates scale. Note that texture projection overrides pseudo-color mode.

Ports

Draw Style



This port determines the draw style of the surface. Five major styles may be selected from 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.

More options: Pressing this button opens a pull-down menu that may be used to fine-tune draw style. This menu contains 4 groups of items:

The first group deals with surface shading and pseudo-color mapping.

- *Specular*: 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 interpolated colors are drawn inside the triangles. See also ->*Texture*.
- *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. See also ->*Gouraud*.

The second group of items refers to transparency. 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 (except for *Sorted Layers* and *Sorted Layers Delayed* transparency types), 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.

- *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 opacity values of the triangles 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 centers of the triangles are presorted along the major coordinates axes. With *Sorted Layers* and *Sorted Layers Delayed* transparencies, this option has no effect.

The third group of items control which side of a triangle is rendered. With *Culling* enabled either the front or the back *face* of a triangle is rendered. This may be used to improve rendering performance on some older graphics boards. Note that with modern graphics boards the difference is usually negligible. Three modes can be selected:

- *Both faces*: Both faces are rendered.
- *Front face*: Only front faces are rendered.
- *Back face*: Only back faces are rendered.

The following group of items is available for *SurfaceView* only. The settings here affect the computation of the surface normals used for shading.

- *Triangle normals*: Enables per-triangle normals. Shading will be discontinuous at the triangles' edges so that faces appear flat.
- *Vertex normals*: Enables per-vertex normals. An average is computed from neighboring normals for all triangle vertices. This gives the surface a smooth, continuous appearance.
- *Direct normals*: An average is computed from neighboring normals for all triangle vertices. No averaging is performed if two neighboring triangles form an angle greater than the crease angle. The default for this angle is 1.0472 radians (= 60 degrees) and can be changed using the command `setCreaseAngle` in the console.

All ViewBase derived display modules allow export of a surface object.

- *Create surface*: Creates a *Surface* object containing the set of currently visible triangles.

Buffer



This port can be used to modify the list of currently visible triangles. All visible triangles are stored in an internal buffer. To modify the contents of this buffer you need to first *select* triangles (selected triangles are drawn in red wireframe) and then add or remove them using the buttons described below. Triangles can be selected either by manipulating a tab-box dragger, by using the *Materials* (*Patch* or *Boundary id*) ports, or by drawing a contour in the viewer.

- *Add*: Adds highlighted triangles to the buffer. If the Shift key is held down while the button is pressed, 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 contour in the viewer. All triangles within this contour will be highlighted. If CTRL is pressed while drawing, the triangles within the curve are deselected.

TextureWrap



Wrap mode used for the texture projection on the surface when a texture is applied to the surface. This port is hidden if no texture data is connected to the *Texture* connection port. There are two wrap modes:

- *Repeat*: The texture is repeated outside its 0-1 texture coordinate range.
- *Clamp*: Clamps texture coordinates to lie within 0-1 range.

Commands

```
createSurface [name]
```

Converts the set of visible triangles into a *surface* object.

```
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*). See also the description of port *Draw Style*.

```
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 when *points* are selected in port *Draw Style*.

```
setPolygonOffsetMode {auto|on|off}
```

If lines are to be drawn on top of a surface, the surface must be rendered with an offset to avoid artifacts. This can be done automatically (auto), always (on), or never (off).

```
setOutlineColor <color> | <r><g><b>
```

Sets the line color in *outlined Draw style*.

```
setLineWidth <width>
```

Sets the width of the lines in *outlined* and *lines Draw Style*.

```
setHighlightColor <color> | <r> <g> <b>
```

Sets wireframe color of selected triangles.

```
setEmissiveColor <color> | <r> <g> <b>
```

Sets the emissive color of the surface.

```
setSpecularColor <color> | <r> <g> <b>
```

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 size and distinctness of the specular highlight in *Specular* surface rendering. The parameter <shininess> can take values between 0 and 1, 0.2 being the default.

`showBox`

Shows box that is used for selecting triangles. See also the description of port *Buffer*.

`hideBox`

Hides box that is used for selecting triangles. See also the description of port *Buffer*.

18 Molecular Option

18.1 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.

18.2 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.

Timestep editing commands:

`getNumTimeSteps`

Returns the number of time steps in the trajectory.

`getCoordinate <stepIx> <atomIx>`

Returns the coordinate of atom `<atomIx>` of timestep `<stepIx>`. No return value.

`setCoordinate <stepIx> <atomIx> <x> <y> <z>`

Sets the coordinate of atom `<atomIx>` of time step `<stepIx>`. No return value.

```
removeTimestep <stepIx>
```

Removes stepIx'th timestep. No return value.

```
removeTimestepRange <firstStepIx> <lastStepIx>
```

Removes all timesteps starting with the <firstStepIx>'th end ending with the <lastStepIx>'th. No return value.

```
restrictToTimestepRange <firstStepIx> <lastStepIx>
```

Restricts trajectory to timesteps within the range <firstStepIx> and <lastStepIx>. No return value.

```
appendTrajectory <trajectory2>
```

Appends all timesteps of a trajectory of given name in the object pool. This only works if both trajectories have the same topologies and observables. No return value.

```
splitByObservable <observableIx>
```

Splits the trajectory by creating a trajectory for each different value found in the observable. Each new trajectory will contain all timesteps which had the specific observable value. Return the name of the new trajectory bundle object containing all trajectories.

Topology editing commands:

```
addHydrogens
```

Add hydrogens until the valences of all atoms are saturated. No return value.

```
addMMFFParameterization
```

Adds parameters of MMFF94 force field. No return value.

```
removeWater
```

Removes all water molecules and unbonded oxygens and hydrogens. No return value.

```
addHydrogens
```

Add hydrogens until the valences of all atoms are saturated. No return value.

```
removeHydrogens
```

Removes hydrogens. No return value.

```
removeNonPolarHydrogens
```

Removes hydrogens on non polar heavy atoms. No return value.

generateSmiles

Returns SMILES string of molecule.

assignKekuleBondOrders

Converts all aromatic bond orders to single and double to create a kekule structure. No return value.

assignNonKekuleBondOrders

Reverses the kekule structure assignment. No return value.

Observables commands:

getNumObservables

Returns number of observables.

getObservableNames

Returns the names of all observables as a list.

getObservableIx <obsName>

Returns index of observable with given name. Returns 0 if no such observable exists.

getObservableName <obsIx>

Returns name of observable with given index.

addFloatObservable <name>

Adds a new float observable with the given name. Initial values will be 0. Returns index.

addIntegerObservable <name>

Adds a new integer observable with the given name. Initial values will be 0. Returns index.

setObservableValue <obsIx> <stepIx> <value>

Sets value of <stepIx>'th timestep of observable with index <obsIx> to value. If stepIx is 0 the value will be set for all timesteps. No return value.

getObservableValue <obsIx> <stepIx> <value>

Returns value of <stepIx>'th timestep of observable with index <obsIx>.

sortByObservable <obsIx> [a/d]

Sorts timesteps of the trajectory so that the values of the given trajectory are in

ascending (a) or descending order. The sort order is an optional argument and is ascending. No return value.

```
computeObservableRange <obsIx>
```

Returns range (minimum and maximum) of values of given observable.

```
computeObservableStatistics <obsIx1> [<obsIx2>]
```

Computes variance and mean of the given observable. If a second observable is specified it will also compute the correlation and covariance between the two observables.

Alignment:

```
alignTimestep <stepIx> <alignToStepIx>
```

Aligns coordinates timestep <stepIx> to coordinates of timestep <alignToStepIx> by minimizing RMSD. No return value.

```
alignTimesteps <alignToStepIx>
```

Aligns coordinates of all timesteps to coordinates of timestep <alignToStepIx> by minimizing RMSD. No return value.

Clustering commands:

```
clusterTimestepsKMeans <clusterNumber>
```

```
[observableName]
```

Applies K-Means clustering to the timesteps of the trajectory with the rmsd between the coordinates of two timesteps used as distance metric. The parameter <clusterNumber> determines how many clusters will be created. The command will generate an integer observable in the range of [1...clusterNumber] containing the cluster index of each timestep. If no observable name is given, the name clusterIx will be used. The K-Means algorithm is non deterministic. No return value.

```
clusterTimestepsQT <clusterRadius> <minMemberNum>
```

```
[observableName]
```

Applies QT clustering to the timesteps of the trajectory with the rmsd between the coordinates of two timesteps used as distance metric. Unlike K-Means clustering, QT clustering does not have a predefined cluster number. Instead the parameter <clusterRadius> determines the rmsd threshold for a cluster and the larger this radius, the larger the clusters will be and the smaller the number of clusters. To avoid a large number of small clusters and focus on significant

clusters the parameter <minMemberNum> restricts the clustering to clusters which have this minimum number of members. The command will generate an integer observable in the range of [0...clusterNumber] containing the cluster index of each timestep whereby 0 means that the timestep was not assigned to any cluster. If no observable name is given, the name clusterIx will be used. The QT algorithm is deterministic. QT-Clustering is significantly more computationally demanding than K-Means. The latter algorithms should be thus preferred for large trajectories. No return value.

Miscellaneous commands:

`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.

`loadIntoMemory`

Makes sure that all coordinates of the Trajectory are stored in local memory.

18.3 MolTrajectoryBundle

An object of type *MolTrajectoryBundle* represents a set of *MolTrajectories*.

Commands

`getNumTrajectories`

Returns the number of trajectories in the bundle.

`getTrajectoryIx <name>`

Returns index of trajectory with given name. 0 if no such trajectory.

`setTrajectoryName <index> <name>`

Renames the index'th trajectory with the given name. No return value.

`findDuplicates`

Searches for identical topologies within the bundle. Prints a list which shows

for each trajectory the list of trajectories that have an identical topology. If no duplicates were found in the bundle, 0 is returned.

Editing:

`addTrajectory <object-name>`

Adds the trajectory with the given name in the object pool to the bundle. Removes all trajectories with the given indices. No return value.

`replaceWithTrajectory <index> <object-name>`

Repalces the index'th trajectory with the trajectory with the given name in the object pool. No return value.

`removeIx <indices>`

Removes all trajectories with the given indices. The list of indices needs to be seperated by white space and enclosed in curly brackets. No return value.

Example:

- `removeIx {1 5 6}`

`removeDuplicatesByData <name>`

If several trajectories have the same value for the given data field, this command will keep only the first occurence and remove the others. If a trajectory has no field of the given name it will not be considered in the operation (i.e. it will always be kept regardless whether there are other ones without this field).

`copyIx <indices>`

Copies the trajectories with the given indices into a new bundle, which will be added to the object pool. The order of the trajectories in the new bundle will be as given by the indices. The list of indices needs to be seperated by white space and enclosed in curly brackets. Returns label of new object in object pool.

`removeNames <names>`

Removes all trajectories of the given names. The list of names needs to be seperated by whitespace and enclosed in curly brackets. No return value.

`restrictToIx <indices>`

Removes all trajectories in the bundle except the ones with the given indices. The indices need to be seperated by a white space and enclosed in curly brackets. No return value.

`restrictToNames <names>`

Removes all trajectories in the bundle except the ones with the given names. The names need to be separated by a white space and enclosed in curly brackets. No return value.

`restrictRandom <number>`

Restricts the bundle to a random set of <number> trajectories. No return value.

`applyTransform <matrix>`

Applies given 4x4 transformation matrix to all trajectories in the bundle. No return value.

`addHydrogens`

Adds hydrogens to all trajectories in the bundle. No return value.

`removeHydrogens`

Removes hydrogens for all trajectories in the bundle. No return value.

`addMMFFParameterization`

Adds MMFF94 parameterization for all trajectories in the bundle. No return value.

`computeBonds`

Computes bonds based on an average bond length table for all trajectories in the bundle. No return value.

`loadIntoMemory`

For some file types containing trajectories, only the currently used timestep is used in memory. This means that most of the editing commands will not work. This command allows each trajectory in the bundle to be loaded into memory. No return value.

`alignBySmartsMatching`

Aligns each trajectory in the bundle to <mol2> by first matching the given smarts string to both molecules and then using this matching for computing the transformation yielding the lowest rmsd between the matched groups. If either of the molecules did not have a matching nothing will be done. No return value.

Sorting:

```
sortByName
```

Sorts trajectories in bundle by the name. No return value.

```
sortByData <name> <type>
```

Sort the trajectories in the bundle by the given data entry. Type may be "string", "float" or "int". No return value.

```
reverseOrder
```

Reverses order of trajectories in bundle. No return value.

Searching:

```
match <atomexpression>
```

Returns a list of indices of trajectories for which at least one atom matches the given *atom expression*.

Importing/exporting data entries:

```
readData <filename> <dataname> <datatype>
```

Reads data entry from a file. Each data value needs to be in a seperate line and the number of lines must be the same as the number of trajectories in the bundle. No return value.

```
writeData <filename> <dataname>
```

Writes data entry to a file. There will be one line per trajectory containing the data value for the trajectory. No return value.

```
writeAllData <filename>
```

Writes all data entries to a csv file. There will be one line per trajectory containing first the index of the trajectory and then the data values for the trajectory, all comma separated. The first line will contain the field names. No return value.

18.4 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 loaded from a file or generated by several *global tcl commands*. They can be edited by either using some of the data objects tcl commands which are described in the following sections or by using the *Molecule-Editor*, *Attribute-Editor*, or *Data-Editor*.

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

defregexp <levelName/groupName> <atomExpr>

Define a new group by using *atom expressions*. The new group will only contain atoms, i.e., no higher level groups.

applyTransform

The current transform is applied to the molecule, i.e., the original atomic positions are replaced by the transformed positions.

generateSmiles

Returns SMILES string of molecule.

alignBySmartsMatching

Aligns molecule to <mol2> by first matching the given smarts string to both molecules and then using this matching for computing the transformation yielding the lowest rmsd between the matched groups. If either of the molecules did not have a matching nothing will be done and 0 is returned. Else 1 is returned.

computeRMSD

Returns the root mean square distance between the molecule and another molecule. The molecules must have identical topologies (this means the sequence of the atoms must be identical too).

```
savePQR
```

Saves the molecule in PQR format (pdb format with added partial charges and radii). If no attribute names are supplied as parameters, the default values 'charge' and 'radius' will be used.

```
getMolWeight
```

Returns the molecular weight of the molecule. The computation will consider implicit hydrogens in the atoms/implicit_hnum attribute. **Viewer Commands:**

The following commands allow to connect different kinds of viewer modules to the molecule from the command line:

```
showSurface
```

Creates CompMolSurface and MolSurfaceView modules to show the molecular surface. No return value.

```
showSticks
```

Creates a MoleculeView module to show the molecule in sticks representation. No return value.

```
showHBonds
```

Creates a HBondsView module to show all hydrogens bonds. No return value.

```
showHBondsTo
```

Create a new molecule in the object pool containing all hyrdogens bonds between <molecule> and <molecule2>. A HBondView will be connected to this molecule to show the hydrogen bonds. Returns name of new molecule. **Selection Commands:**

```
showBrowser
```

Shows the selection browser. No return value.

```
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
```

There is also a *global command* 'sel' which applies the atom expressions to all molecules in the object pool.

`desel <atomExpr>`

Lets you remove atoms from the selection. Like for 'sel' there is also a global 'desel' command operating on all molecules.

`selAtom <atomIx>`

Selects atom with given index.

`deselAtom <atomIx>`

Deselects atom with given index.

`isAtomSelected <atomIx>`

Returns 1 if atom with given index is selected, 0 otherwise.

`getNumSelection`

Returns number of selected atoms.

`invertSelection`

Atoms which are selected will become unselected and vice versa.

`selWithinSelection <objectname> <range>`

Atom expressions only work on single molecule objects. This command allows the WITHIN operator of atom expressions to be used in relation to another molecule object in the object pool. The command will select all atoms that are within <range> Angstroem from any atom selected in the molecule <objectname>.

`selWithinCoordinate <x> <y> <z> <range>`

Will select all atoms which are within <range> Angstroem from the given coordinate.

`expandSelectionToGroupsOfLevel <levelIx>`

Will expand selection to all groups of given level for which at least a single atom is selected.

`selFromFilter [<objectName>]`

Selects all atoms which are not hidden in the filter of the given downstream object. If the name of the object is omitted it will select all atoms which are not hidden in any of the downstream object filters. No return value.

`selToFilter [<objectName>]`

Sets filter of the given object to the currently selected atoms. If the name of the object is omitted it will apply to all downstream objects. No return value.

---

```
printSelection
```

Prints a lists of all atoms which are selected.

```
printSelectionByAtoms
```

Same as printSelection.

```
printSelectionByResidues
```

Prints a lists of all residues which are entirely or partially selected.

```
printSelectionByLevel <levelName>
```

Prints a lists of all groups of the given level which are entirely or partially selected.

```
getSelectionCenter
```

Computes center of gravity of selected atoms and returns x,y, and z coordinate.

```
getSelectionBBox
```

Computes the bounding box surrounding all selected atoms and returns it as xmin,xmax,ymin,ymax,zmin,zmax.

```
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>. This set can be used to return to the current selection at a later time. Note that there is also a *global command* which will apply addSet to all molecules in the object pool.

```
removeSet <name>
```

Remove set of name <name> 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>. No return value. Like for addSet, there is also a global useSet command. **Labeling commands:**

The following command allow to modify labels of groups. They work on all downstream MolLabel modules.

```
setGroupLabel <levelIx> <groupIx> <label>
```

Attaches a label to a group. This is done by setting the label in a create MolLabel module. If no module exits, a new MolLabel will be created. No return value.

```
setGroupLabelSize [<levelIx>] <size>
```

This command works on all attached MolLabel modules and sets the font size of all labels for the given level. If the level index is omitted, it will be applied to all levels. No return value.

```
setGroupLabelColor [<levelIx>] <red|green|blue|yellow|white|bla
```

This command works on all attached MolLabel modules and sets the color of all labels for the given level. If the level index is omitted, it will be applied to all levels. No return value.

```
setGroupLabelColorRGB [<levelIx>] <r> <g>
```

Same as setGroupLabelColor but instead of using a text string defining a color, the red green blue value of the color (each within the range [0,1]) is specified. No return value. **Editing levels:** The following commands allow to modify the level hierarchy of the molecule. The commands work index based. Removing levels may invalidate locally stored level indices.

```
getLevelIx <levelName>
```

Returns index of level with name <levelName>, 0 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>. Returns 0 if it does not reference another level. **Editing groups of a level:** The following commands allow to modify the groups that are part of a level. The commands work index based. Removing groups may invalidate locally stored group indices.

```
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 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.

```
removeGroups <levelIx> <groupIx1> <groupIx2> ...
```

Remove the given groups from level <levelIx>. Will also remove all dependant groups and attribute entries. No return value.

```
restrictToGroups <levelIx> <groupIx1> <groupIx2>
```

```
...
```

Remove all groups of level <levelIx> except for the given groups. Will also remove all dependant groups and attribute entries. No return value.

```
removeSelection
```

Remove all atoms which are selected and all dependant groups. No return value.

`restrictToSelection`

Remove all atoms which are not selected and all dependant groups. No return value.

`copySelection`

Copies the selected atoms into a new Molecule object. The label of the new object in the object pool will be returned.

`splitSelection`

Copies the selected atoms into a new Molecule object and removed them from the current. The label of the new object in the object pool will be returned.

`addMolecule <objectname>`

Adds the molecule object in the object pool with the given name.

`addMoleculeSelection <objectname>`

Adds all selected groups of the molecule object in the object pool with the given name.

`addGroup <levelIx>`

Adds group to level `<levelIx>`. Returns index of new group.

`addGroupSelection <levelIx>`

Adds group to level `<levelIx>` and sets its group elements to the currently selected atoms. Returns index of new group. **Editing attributes:** The following commands allow to modify attributes of groups. The commands work index based. Removing attributes may invalidate locally stored attribute indices.

`getNumAttributes <levelIx>`

Return number of attributes of level `<levelIx>`.

`getAttributeIx <levelIx> <attrName>`

Return index of attribute `<attrName>` of level `<levelIx>`.

`getAttributeName <levelIx> <attrIx>`

Return name of attribute `<attrIx>` of level `<levelIx>`.

`addStringAttribute <levelIx> <attrName>`

Adds a string attribute of name `<attrName>` to level `<levelIx>`. Returns the index of new attribute. Returns 0 if not successful.

```
addIntegerAttribute <levelIx> <attrName>
```

Adds an integer attribute of name <attrName> to level <levelIx>. Returns the index of new attribute. Returns 0 if not successful.

```
addFloatAttribute <levelIx> <attrName>
```

Adds a float attribute of name <attrName> to level <levelIx>. Returns the index of new attribute. Returns 0 if not successful.

```
removeAttribute <levelIx> <attrIx>
```

Remove attribute <attrIx> from level <levelIx>. No return value.

```
setAttributeName <levelIx> <attrIx> <attrName>
```

Sets name of attribute <attrIx> of level <levelIx> to <attrName>. No return value.

```
getAttributeValue <levelIx> <attrIx> <groupIx>
```

Returns value of attribute <attrIx> of level <levelIx> for the group <groupIx>.

```
setAttributeValue <levelIx> <attrIx> <groupIx>
<value>
```

Sets value of attribute <attrIx> of level <levelIx> for the group <groupIx> to <value>. **Editing coordinates:**

```
getCoordinate <atomIx>
```

Returns the coordinate of atom <atomIx>. No return value.

```
setCoordinate <atomIx> <x> <y> <z>
```

Sets the coordinate of atom <atomIx>. No return value.

```
perturbCoordinates [<max>]
```

Applies a random perturbation to each coordinate. The perturbation will be uniformly distributed in the range [-max,max]. If the parameter is omitted 0.001 will be used. No return value. **Editing data entries:** The following commands allow to modify data entries of the molecule. The commands work either by using the data entry's index or its name. Removing entries may invalidate locally stored data indices.

```
setData <dataID> <value>
```

Adds a data entry with the given name and value. The name must contain at least one letter. If the entry already exists <dataID> can be either its name or its index and the value will be reset to the new value. No return value.

`removeData <dataID>`

Removes a data entry. The `<dataId>` may either be the entry's name or its index.  
No return value.

`getDataValue <dataID>`

Returns the value of a data entry. The `<dataId>` may either be the entry's name or its index.

`getDataName <dataIx>`

Returns the name of a data entry with the given index.

`getDataIx <dataName>`

Returns the index of a data entry with the given name.

`getNumData`

Returns the number of data entries. **Miscellaneous editing commands:**

`addHydrogens`

Add hydrogens until the valences of all atoms are saturated. No return value.

`removeHydrogens`

Removes hydrogens. No return value.

`removeNonPolarHydrogens`

Removes hydrogens on non polar heavy atoms. No return value.

`splitByLevel <levelIx>`

Splits molecule by putting each group of the given level into a separate MolTrajectory. Returns the name of the MolTrajectoryBundle object which is created.

`splitByAttribute <levelIx> <attributeIx>`

Splits molecule by putting all groups of the given level that have the same attribute value into a separate MolTrajectory. If the attribute is the index attribute of the level this command does the same as the splitByLevel command. Returns the name of the MolTrajectoryBundle object which is created.

`cleanBonds`

Some imported molecule structures have duplicate entries for the same bond. This leads to problems in some algorithms. This command removes such duplicates. No return value.

---

```
addMMFFParameterization
```

Adds parameters of MMFF94 force field. No return value.

```
addMassAttribute [<name>]
```

Adds an atom attribute containing the atomic mass. Specifying the name of the new attribute is optional. If no name is given 'mass' will be used. No return value.

```
addRadiusAttribute [<name>]
```

Adds an atom attribute containing the standard van der Waals radius. Specifying the name of the new attribute is optional. If no name is given 'radius' will be used. No return value.

```
removeWater
```

Removes all water molecules and unbonded oxygens and hydrogens. No return value.

```
assignKekuleBondOrders
```

Converts all aromatic bond orders to single and double to create a kekule structure. No return value.

```
assignNonKekuleBondOrders
```

Reverses the kekule structure assignment. No return value.

```
computeHBonds [<maxDistDA> <minAngleDAX>]
```

Will create a level 'hBonds' containing potential hydrogen bonds. The optional parameters <maxDistDA> determines the maximum distance between donor and acceptor atom and <minAngleDAX> the minimum angle between donor, acceptor, and a heavy atom bonded to the acceptor. The default values are 3.9 and 120. For more fine grained control and more details look at the documentation of the *ComputeHBonds* module. No return value. **Protein specific editing commands:**

```
alignToProtein <mol2>
```

Aligns this molecule to another molecule in the object pool with the given label. The alignment will be accomplished by residue sequence alignment with the CompSeqAlign module. Both molecules need to contain the residues/type attribute with 3 letter code amino acid types. No return value.

```
alignProtein <mol2>
```

Aligns a molecule in the object pool with the given label to this molecule. Both

molecules need to contain the residues/type attribute with 3 letter code amino acid types. No return value.

#### applyBiomt

PDB files may contain information how to generate multimers from a single unit in BIOMT REMARK records. These records are read in as a PDB\_REMARK data field. The applyBiomt command will parse all BIOMT matrices in this field, duplicate the structure, and apply the specified matrix transformations.

#### appendAminoAcid <index> <terminus> <code>

Append a new amino acid to the amino acid that is the index'th group in the residues level. The type of the new amino acid must be given in the 3 letter code. Terminus must be 'C' or 'N' depending on which terminus the old index'th residue the new residue is appended. For this command to work, the atoms/type attribute must contain the pdb atom types and the residues/type and bonds/type attribute must exist. No return value.

#### substituteAminoAcid <index> <code>

Substitutes the amino acid that is the index'th group in the residues level with the amino acid of the given 3-letter code. For this command to work, the atoms/type attribute must contain the pdb atom types and the residues/type and bonds/type attribute must exist. No return value.

#### mutatePDB <mutationString>

This command acts similar as the substituteAminoAcid command except that the syntax of the substitution definition differs to make this command more easily usable with pdb proteins. The mutation string is a concatenation of the original type, the pdb index and the new type (example LYS34ASP). Both 3 or 1-letter code may be used (L34H). The index refers to the pdb\_index attribute. If the original type is left out, the amino acid with the given index will be substituted regardless of type (example 34H). If the index is left out, all amino acids of the given original type will be substituted with the new type. In this latter case, 3-letter codes need to be used (example LYSASP). A set of substitutions can be applied at once by separating them by whitespaces and enclosing the set in curly brackets (example {L23H G24S}). No return value.

#### getAminoAcidChirality [<index>]

Returns whether amino acid <index> is an 'L' or 'D' isomer. Returns 'N' if residue is not an amino acid or if required attributes are missing. If <index> is omitted a string containing 'L', 'D' or 'N' for each residue will be returned.

For this command to work, the atoms/type attribute must contain the pdb atom types and the residues level must exist containing the amino acids.

`setAminoAcidChirality <index> <chiralCode>`

Sets amino acid to 'L' or 'D' isomer. For this command to work, the atoms/type attribute must contain the pdb atom types and the residues level must exist containing the amino acids. No return value.

`getProteinFASTASequence`

Returns the protein sequence in FASTA format for all convertible chains. The topology must have chains and residues levels and contain the attributes chains/name, residues/type, and residues/name. All residues whose name starts with 'HET' will be cut out. If a chain contains any other residues whose type cannot be converted from 3 letter code to 1 letter code, it will be skipped.



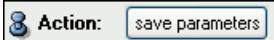
# 19 Very Large Data Option

## 19.1 VolumeDataObject

This data type is an uniform scalar field specialized for the out-of-core management. The whole data set does not need to fit in memory to be visualized. Typically, the file extension associated to this data type is the *LDA* file format.

### Ports

#### Action



If data parameters have been edited via the parameter editor, use this button to save them into the *lda* file.



# 20 Mesh Option

## 20.1 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



The mass is a parameter of the Schwarzschild and Kerr metrics.

### Angular moment



The angular moment used in the Kerr metric.

## 20.2 PbScalarField3

This is a class of 3D scalar fields defined on polyhedral grids. See also *PolyhedralGrid* and its base class *ScalarField3*.

### Ports

## 20.3 PolyhedralGrid

A data object of type *PbMesh3D* represents an unstructured finite-element grid composed of polyhedra. The geometric information is stored in terms of vertices, edges, faces, and polyhedra. For instance such data objects are useful as patient models. A polyhedral grid 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.

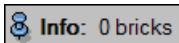
# 21 Skeleton Option

## 21.1 Mosaic

The Mosaic data class can be used as a container for other *SpatialData* objects. It stores a reference to the location on file and the bounding box information. A useful application is to arrange several bricks of overlapping image data and use a *MosaicToDiskData* to convert them into one large image stored on disk.

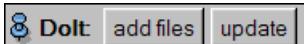
### Ports

#### Info



Provides information about the mosaic.

#### DoIt



Press *add files* to select files on disk which will be added to the mosaic.

Use *update* to reread all files from disk and update the information stored in the mosaic. You should force an update if you change the data objects referenced by the mosaic.



## **Part V**

# **Alphabetic Index of Editors**



# 22 Amira

## 22.1 Advanced Colormap Editor



To modify existing colormaps push the edit button which is visible when the colormap is selected.

As you can see, there is a menu bar near the top border of the window with the submenus *Edit*, *Mode*, and *Brush*, which will help you to control the behaviour 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.

## 22.1.1 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

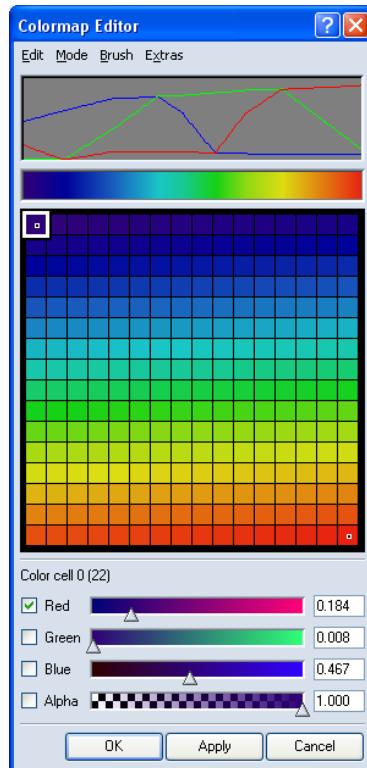


Figure 22.1: Amira colormap editor

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*.
- **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 imme-

dately 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 epifluorescence 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.

### 22.1.2 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 be 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.

## 22.2 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,

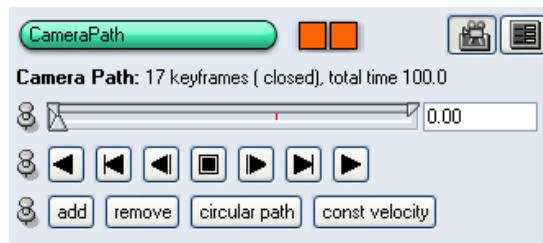
choose an appropriate view in the original viewer (to learn how to do that, see section *Viewer Window* in chapter *Program Description* of the Amira user's guide) and click on the *add* button. Your first keyframe has been saved. The time slider should contain one red line and it should have advanced 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.

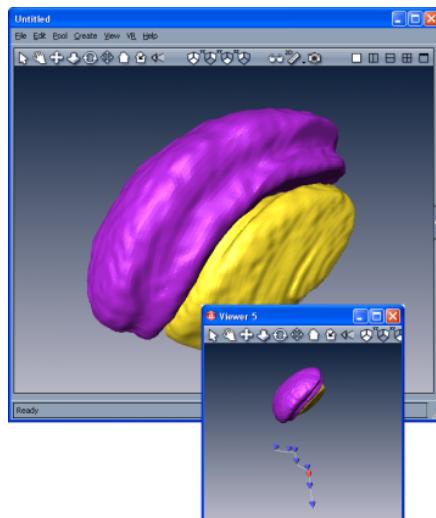
To change the time of a keyframe, jump to this frame, click on *remove*, type in the new time, and click on *add* again.

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.

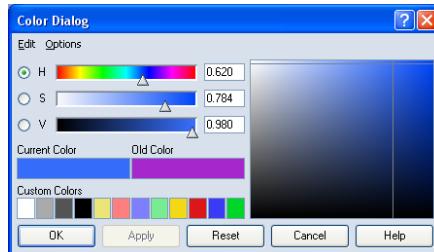
**Caution:** Pressing the *circular path* button will create a new camera path and destroy the old one. The new camera path lies on a circle around one of the major axes with the center at the point the camera in the original window is looking at. The radius is determined by the distance of the camera position and the point the camera looks at.



**Figure 22.2:** 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 22.3:** 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 22.4:** Amira color dialog

## 22.3 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, 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.

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.

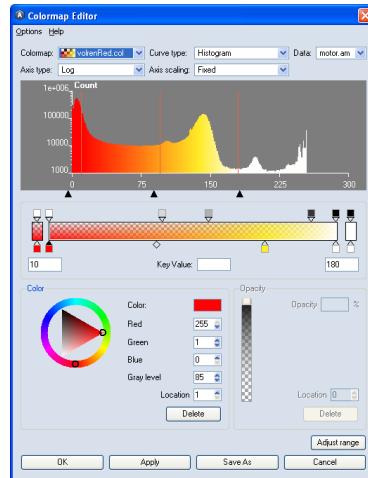
**Color Cells:** The color cells are subdivided into three different groups: One cell named *Current Color* displays the color depending on the current settings. Each color manipulation is shown by this cell immediately. The cell named *Old Color* displays the original color the dialog has been invoked 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 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.



**Figure 22.5:** Amira colormap editor.

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.

## 22.4 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.

## 22.4.1 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.

## 22.4.2 Menu Bar

The menu bar contains two items: a *Help* button which brings up this help topic in the help browser, and the *Options* menu which contains the following 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.
- **Show Alpha Curve:** If this toggle is active, the alpha values of the colormap are edited by changing the shape of a curve rather than by adjusting opacity markers. Opacity markers are described in the *Gradient Editor* text below, the alpha curve in the *Alpha Curve* text. **Note:** Unlike most attribute settings, this setting is persistent across Amira sessions.

## 22.4.3 Global Settings

### 22.4.3.1 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.

### 22.4.3.2 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.

#### 22.4.3.3 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.

#### 22.4.3.4 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.)

#### 22.4.3.5 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.

### 22.4.4 Histogram

The purpose of the *Histogram* is to graphically summarize the distribution of values within the selected data set, colorized 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.

## 22.4.5 Gradient Editor

The gradient editor allows you to modify the colormap using color markers and either opacity markers or an *alpha curve* depending on the setting of the *Show Alpha Curve* toggle of the Colormap Editor's *Options* menu. The color is linearly interpolated between two consecutive markers. Color and alpha channels are treated separately.

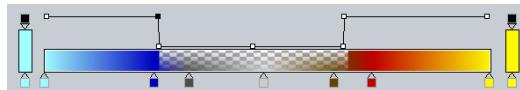
When selecting a colormap, the markers are automatically extracted. To modify the gradient, you can add or remove markers. To add a new marker, click on the area containing the color or opacity marker at the desired position. A new marker is added with the interpolated value. You can move this marker 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. Or right click on the marker.

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.



**Figure 22.6:** Amira Gradient Editor showing alpha curve

## 22.4.6 Alpha Curve

In many cases it may be more convenient to control the transparency of the colormap via the alpha curve than by using the opacity markers described above. The color gradient is updated continuously while you adjust the alpha curve.

To add a point to the curve, click where you want the point added. It is not necessary to click exactly on the curve.

To move an existing point, click on it to select it (it will be highlighted in black), then drag it with the mouse. It can also be moved using the *Opacity Chooser*.

To remove a point, click on it with the right mouse button.

To reset the alpha curve definition to its default state (full opacity, no intermediate points), press the Delete key.

## 22.4.7 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.

## 22.4.8 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.

### 22.4.9 Control Buttons

- **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.

## 22.5 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



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.

#### Samples



Enter the number of samples.

#### Mode



In *interpolation* mode, the curve goes through the specified points. In *approximation* mode, the curve approximates the polygon spanned by the points.

#### Editing Mode



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



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. Furthermore, the *Interactive Feedback* can be activated to see a preview of the future curve before appending a new point.

### Point Size



Specifies the size of points.

### Display



Specifies the display mode. These options are not mutually exclusive. Checking off the 'points' will display points of the BSpline and checking off the 'lines' will display lines between the points. At this time 'curve' option is not available.

### Curve

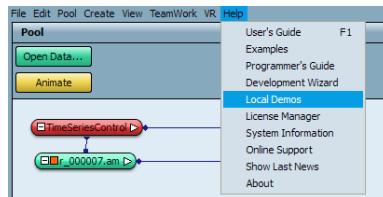


Resets the curve to its original shape by removing changes made to it with the Curve Editor

## 22.6 Demo GUI

The *Demo GUI* is a special-purpose system for managing demos. Demos to be selected with the Demo GUI must have been previously prepared as described in the *Demo Framework* documentation.

To use the Demo GUI, you must set the environment variable \$AMIRA\_DEMOS to the directory which contains the prepared demos. Within Amira you will find



**Figure 22.7:** Amira Help menu with Demo GUI item

an item *Local Demos* in the *Help* menu. If this item is inactive (grayed out), \$AMIRA\_DEMOS is not set. If no valid demos are found in the specified directory, an error popup will be displayed.

The Demo GUI window consists of two tabs:

**Preselection tab:** On this tab you can perform a filter operation on all demos available in your demos directory. The filter result is shown in the right list.

**Selection tab:** Here you can select the demos you want to use for your presentation from the filter result (The content of the right list of the preselection tab is identical with the left list shown on the selection tab).

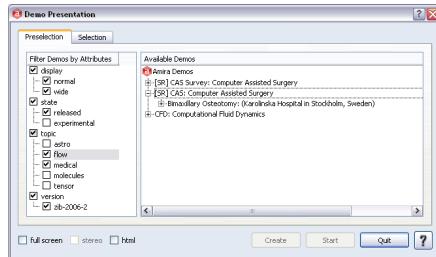
## Preselection tab

On the preselection tab you can select your desired attributes from the attributes checkbox tree on the left and the result of your search will be shown in the right list. If you alter your search query the result list will be updated automatically.

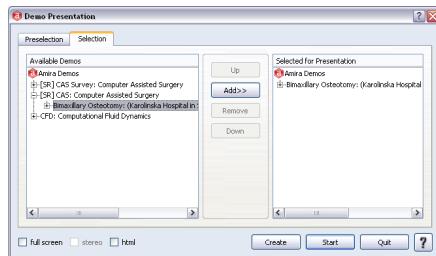
## Selection tab

The selection tab contains two lists:

**Left List:** List of all available demos. Here the demos are organized according to the directory structure under \$AMIRA\_DEMOS. To expand, i.e., to go down in the directory structure, click on the + icon. Demos can be expanded up to the level of steps.



**Figure 22.8:** Preselection tab with the result of the filter operation on the right



**Figure 22.9:** Selection tab with one selected demo

**Right List:** List of all selected demos. If you selected and added a demo at a higher level (a group of demos), they are expanded up to the demo level on the left list.

To select demos, click with the left mouse button on a demo on the left list and add it with the *Add* button to the right list, the selection list. Once a demo has been put into the right list, it can be selected again and then it is possible to change its position within the list (*Up/Down* buttons) or remove it again. Steps within a demo may be deleted, for example, to shorten a demo.

For more information about a particular demo, just click with the right mouse button on a demo on either the left or right list. A popup window containing some information and thumbnail images will appear.

## Starting Demos

The three toggles below the tabs may be used as follows:

**full screen:** If on, the demos are displayed in a full screen viewer. Since the Amira main window is obscured, it is only possible to steer the demos with the help of the *function keys*.

**stereo:** This toggle is only activated if the current machine or graphics card has OpenGL stereo capabilities. If set, the Amira script for the demos contains a Tcl variable STEREO which in turn is used by the underlying individual demo scripts to switch stereo on or off.

**html:** If this item is checked and the *Start* button has been pressed, the Amira help window pops up and the selected demos can be started and steered also from this window.

The *Create*, *Start*, and *Quit* buttons provide the following functionality:

**Create:** Creates a network with only the *DemoSequence* and the *DemoStatus-Display* module. In this state you can save the network in order to use the compiled demo later on. To really start the demo, press the *Start* button in the *DemoSequence* module or see below in Figure 22.10

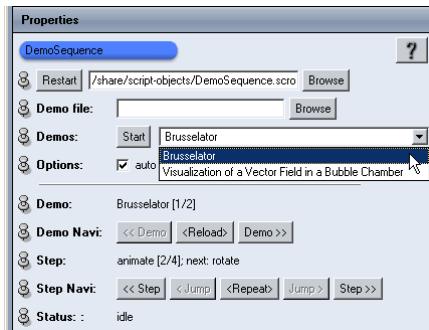
In addition, a directory selection is created under the \$AMIRA\_DEMOS directory which contains the selected demos in the same ways as is done by the *prepareDemos.tcl* commandline tool.

**Start** : Creates the network for the first selected demo and starts its first step.

**Quit:** Exit the Demo GUI window.

## Rearrange Demos

After the selected demos have been executed, or a demo network compiled with either the Demo GUI or the command-line tool has been loaded, you may start the



**Figure 22.10:** *DemoSequence GUI*

Demo GUI again by selecting the item in Amira's *Help* menu, then rearrange the previously selected demos.

## 22.7 Digital Image Filters

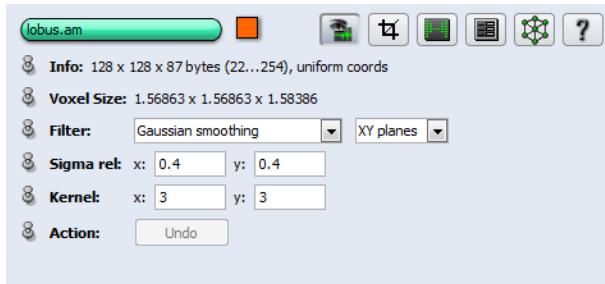


This editor provides digital image filters for 3D image data sets, such as smoothing, unsharp masking, and morphological operations. Some filters operate in 3D while others are applied to two-dimensional slices. In the latter case, the orientation of the 2D slices can be selected via an option menu. Press the *Apply* button to start the computation.

**Note** All image filters are executed using multithreaded computation. See the *Edit/Preferences* to select the number of cores to be used.

Currently, the following filters are supported:

- *Noise reduction minimum filter*
- *Noise reduction maximum filter*
- *Unsharp masking*
- *Laplacian edge detection filter*
- *Noise reduction median filter*
- *Gaussian smoothing filter*



**Figure 22.11:** Editor for applying digital image filters.

- *Sobel edge detection filter*
- *Equalize filter*
- *Edge-Preserving filter*
- *Non-local means filter*
- *Resampling/Low pass filter*
- *Intensity remapping filter*
- *Brightness/Contrast filter*
- *Statistical feature detection filter*
- *Lighten/Darken filter*

## 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.

### 22.7.1 Noise reduction minimum filter (Minimum)

This filter replaces the value of a pixel by the smallest value of neighboring pixels covered by an NxN 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.

### 22.7.2 Noise reduction maximum filter (Maximum)

This filter replaces the value of a pixel by the largest value of neighboring pixels covered by an NxN 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.

### 22.7.3 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. The parameter *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.

### 22.7.4 Laplacian edge detection filter (Laplacian zero-crossing)

This 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.

## 22.7.5 Noise reduction median filter (Median)

This 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 NxN 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.

## 22.7.6 Gaussian smoothing filter (Gauss)

The Gaussian filter smoothes 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). The minimum kernel size is 1 which means that the image is not blurred at all in that direction. The text fields labeled *sigma rel* allow you to adjust the width of the Gauss function relative to the kernel size.

## 22.7.7 Sobel edge detection 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.

## 22.7.8 Equalize filter (Histogram)

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.

**Note** that in the 3D mode the computation is not multithreaded so it can take a rather long time if the data is large. A faster preview is always possible by switching to the 2D mode.

## 22.7.9 Edge-Preserving smoothing

This is a smoothing filter that models the physical process of diffusion. Similar to the Gaussian filter, it smoothes 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 is not multithreaded so it 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 smoothes 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.

## 22.7.10 Non-local means filter

This module implements the windowed non-local means algorithm for denoising scalar volume data. The original non-local means algorithm did not use a search window, but compared each voxel with each other voxel. Of course, this leads to very long running times which are unfeasible. Hence, in general a windowed version of the non-local means algorithm, as implemented in this module, is used. In order to determine the new value for the current voxel, the algorithm compares the neighborhoods of all voxels in a given search window with the neighborhood of the current voxel. The similarity between the neighborhoods determines the weight with which the value of a voxel in the search window will influence the new value of the current voxel. The final weights are determined by applying a Gauss kernel to the similarity values.

The non-local means algorithm works most effectively if the noise present in the data set is white noise. This is the assumption behind the algorithm. If this can

be assumed, the non-local means algorithm will naturally preserve most features present in the image, even small and thin ones. However, the algorithm must not be confused with a feature enhancement algorithm, such as edge enhancement. If edge enhancement is required, it is suggested to run non-local means filtering first followed by edge enhancement. This should give the best results.

This module implements a CPU and a GPU version of the non-local means algorithm for 2D, that is in each slice, and for 3D. While the results of the 3D version are slightly better, the 2D version is much faster and gives similar results. The module can manage data partitioning so the data can be processed even if it does not fit entirely in GPU memory, and also in order to avoid system timeouts that may apply on GPU used for display. For more details, see the explanations about CUDA Device, Options and Partition size ports.

**Note:** The GPU version of the algorithm requires a fairly new NVidia graphics board in order to run with adequate speed. It is recommended to use a graphics board of the latest generation.

For more information about the algorithm itself see:

A. Buades, B. Coll., and J.M. Morel, A non local algorithm for image denoising, in Proc. Int. Conf. Computer Vision and Pattern Recognition (CVPR), 2005, vol. 2, pp. 60-65.

## Ports

### Search Window



Search window size. The algorithm looks for matches within this area around each point. The value represents the diameter of the search window area in number of voxels. The larger the search window is, the better are usually the results. But the size of the search window also effects the running time significantly. The larger the search window, the longer the run time. This value has to be set to a large enough value so that similar structures can be found within the search window area. Too small values will result in simple blurring of the image because there is not enough structural information within the search window area. A search window size of 21 is usually a good choice.

### Local Neighborhood



Size of the neighborhood window. The influence of each point in the search

window on the base point is weighted by comparing the neighborhood window of this point with the neighborhood window of the base point of the search window. The value represents the diameter of the neighborhood area in number of voxels, and affects the quality of the result as well as the run time. If this value is either much smaller or much larger than fine structures in the data the algorithm shows little or no effect at all. The larger this value, the longer the run time.

### Similarity Value

 **Similarity Value:**

The similarity value determines the similarity weight assigned to each voxel in the search window. The larger the value, the higher the weight assigned to a voxel. As a result, the larger the value, the more the resulting image will be smoothed. In mathematical terms, the squared similarity value is proportional to the standard deviation of the assumed Gaussian noise of the image. The similarity value does not affect the calculation time. To achieve similar results with the 2D and 3D implementations, the similarity value must be larger when using the 2D implementation.

### Adaptive

 **Adaptive:**  false  true

Adaptive choice of lambda. If adaptive is set to true, lambda will be adjusted for each voxel using the standard deviation of each search window. This accounts for different contrast levels across the data set. It is generally recommended to use the adaptive mode.

### Device

 **Device:**  GPU  CPU

Choose whether the CPU or the GPU (CUDA device) is used for calculation. On modern graphics cards, the GPU calculation is generally much faster than the CPU calculation, even if run on several processors.

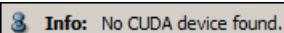
### CUDA device

 **CUDA device:** GeForce GTX 285 [ID: 0] 

This port shows the list of available CUDA devices. The devices that have no kernel timeouts are first in the list. Note that if there is only one CUDA device and no other graphic card, the computation time is limited to approximately 5

seconds if it uses hardware acceleration at the same time. On Windows this is generally the case; on Linux you can use SSH-tunneling and/or VNC on a machine not running any X server to use your local machine for graphics output. If the computation time exceeds 5 seconds, the module might terminate the computation raising a CUDA error. To avoid timeouts, please enable *automatic partition size* in the *Options* port.

### Info



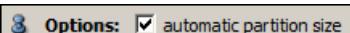
This port is only visible if no CUDA device is available. In this case it displays a message that no CUDA device is available.

### Number of Threads



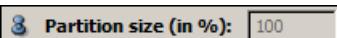
This port is visible if the selected device is CPU and multithreaded calculation is available. The port allows the user to set the number of threads used for the calculation.

### Options



If this option is selected, a sensible partition size will be automatically determined. Switching off this option might result in a CUDA error due to too large block sizes.

### Partition size



This port is only visible if the selected device is GPU. This value allows the user to scale the blocks that are being sent to the graphics card. If this value is 1, the block size is chosen small enough to run the algorithm on the GPU. When computing NLM on a single GPU, however, it might be necessary to reduce the block size further to allow the computation for a single block to be finished within 5 seconds. This can be done with this parameter. A heuristic tries to find a good value for this parameter. If you think the parameter is too small, then you can manually tune it. This, however, may result in CUDA driver crashes which will require restarting the application.

### GPU Memory

 GPU Memory: 2020 MB

This port displays the usable size of the memory on the CUDA device.

#### Max. Block Size

 Max. Block Size: 808 MB

This port displays the maximum block size currently used. This value is determined by the partition size and other constraints.

### 22.7.11 Resampling/Low pass filter (Lanczos, Phase 0)

This 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.

### 22.7.12 Intensity remapping filter (Sigmoid)

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 it width  $\alpha$ . The target image range is given by the interval  $[min, max]$ .

$$f(z) = (max - min) \cdot (1 + \exp(-\frac{z - \beta}{\alpha}))^{-1} + min$$

### 22.7.13 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 ( $x_{av}$ ).

$$f(x) = x_{av} + \text{Contrast} \cdot (x + \text{Brightness} - x_{av}),$$

where

$$x_{av} = \text{Brightness} + Image_{min} + \frac{(Image_{max} - Image_{min})}{2}$$

### 22.7.14 Statistical feature detection filter (Moments)

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 *Cast-Field* to do a conversion of your data set to floats or doubles first.

### 22.7.15 Lighten/Darken filter (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.

## 22.8 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 crop such a field, i.e., to discard all voxels lying outside of a specified subvolume in index space. Alternatively, you



**Figure 22.12:** The ImageCrop editor allows you to crop a 3D image and to modify its bounding box or voxel size.

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 22.12 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 *Crop list* allows you to select a ROI and use its box coordinates as crop coordinates.

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.

### 22.8.1 Cropping an Image by Dragging and Moving the Box

Some experience with manipulating *draggers* 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.

### 22.8.2 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.

### 22.8.3 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.

#### 22.8.4 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.

#### 22.8.5 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.

#### 22.8.6 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 data set.

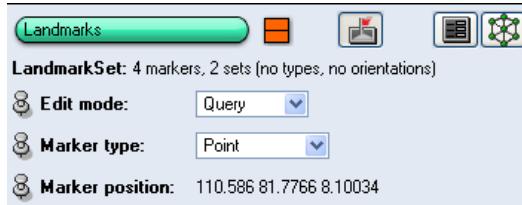


Figure 22.13: User interface of the landmark editor.

## 22.9 Landmark Editor

 This component allows you to edit *landmark sets*. It operates by directly interacting with the 3D geometry in the viewer window, cf. section *Viewer Window* in chapter *Program Description* of the Amira user's guide. 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:

- **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 pre-defined specialized marker types can be selected. In contrast to point-type markers, specialized markers have fixed predefined sizes (world coordinates

assumed to be given in centimeters).

See also documentation for *Landmark Set*.

## 22.10 LineSet Editor



This component allows you to edit *linesets*. It operates by directly interacting with the 3D geometry in the viewer window, cf. *sectionViewer Window* in chapter *Program Description* of the Amira user's guide. 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.
- **Draw.** To draw lines.

- **Values.** To set values at selected points.

## 22.11 Multiplanar Viewer



The **Multiplanar Viewer** is a sub-application designed to visualize and explore one or two image data sets and one label field (segmentation results) at the same time. In its default configuration, the viewer offers 3 slice oriented 2D viewers displaying orthogonal and/or oblique (thick) slices (MPR viewer) and one 3D viewer showing up to two image volumes with a volume rendering technique. Besides a rich set of visual manipulation and cropping functionality the viewer offers specialized tools for manual and automatic image registration.

The documentation of the *Multi-planar Viewer* is divided into the following parts:

- *Overview of the Multi-planar Viewer*
- *How to use the Multi-planar Viewer*

### 22.11.1 Overview of the Multiplanar Viewer

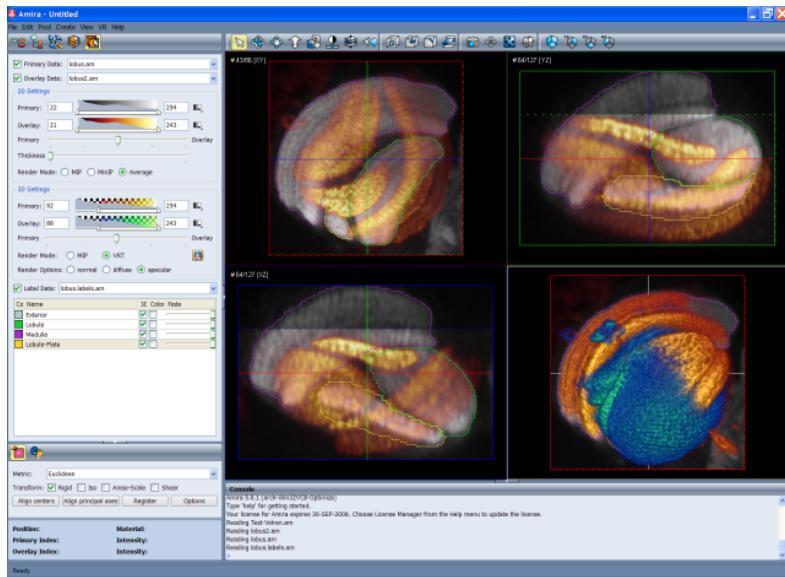
The Multi-planar Viewer is activated by pressing the Multiplanar Viewer button in the sub-application task bar. This will change the layout of the main and viewer window as shown in Figure 22.14. The main components of this layout are the viewer (MPR+3D viewer), the user interface to control the visualization, the tool-box, and information area.

#### The MPR+3D viewer

By default, the viewer is displayed in 4-viewer mode. If you prefer to work with one larger view rather than four smaller views, double-click on the viewer of your choice. To switch back to 4-viewer mode just double-click on the single viewer again. Right-clicking on a viewer activates the context-menu through which you can access tools and visualization options.

#### The User Interface

- **Data Selectors:** In the upper part of the main window two data selectors allow selecting the primary and secondary image volumes to be displayed. The pull-down menus will list all compatible data objects that are available in Amira's workspace (**Pool**). We define the terms "Primary" and "Overlay" in the context of the Multiplanar Viewer as:



**Figure 22.14:** The Multi-planar Viewer.

- **Primary** - Reference data set. This data set will be considered our principal data set to which a secondary data set could be eventually compared or registered. In the context of PET/CT fusion, the CT data set would be used as reference data set as it provides the best anatomical reference.
- **Overlay** - Secondary data set, which could be compared or registered to the Primary.
- **2D Settings:** controls for the colormap settings, the Primary-Overlay ratio (or the Alpha transparency if just one data set is displayed), the thickness and rendering mode of the thick slice in the 2D viewers. Thick slice rendering Modes:
  - **MIP** - Maximum intensity projection. The brightest data value along each ray of sight in the thick slice is displayed. This mode is especially useful for very sparse data sets, for example, angiographic data

or images of neurons.

- **MinIP** - Minimum intensity projection. Same as MIP but with the darkest value.
  - **Average** - The average data value along each ray in the thick slice is calculated and displayed.
- **3D Settings:** same as 2D Settings but applied to the 3D viewer. Volume Rendering Modes:
    - **VRT** - Texture-based volume rendering, with optional shading effects. If VRT is selected the user can optionally set the light effect to "normal", "diffuse", or "specular".
    - **MIP** - Maximum intensity projection. The brightest data value along each ray of sight is displayed instead of the result of the emission absorption model. If MIP is selected the user can optionally set the visualization effect to "normal" or "inverse".
    - **3D MPR button** - Turns on/off the MPR in the 3D viewer.
  - **Label Field:** In the central part of the editor window boxes allow to select the LabelField object to be visualized. This port refers to the Primary data set; to be activated the loaded LabelField must to have the same dimensions as the Primary data set. If the port is activated and the checked, the materials will be listed in the window below. You can optionally turn on/off the 3D rendering, select a constant color for rendering, or fade the labeled voxels.

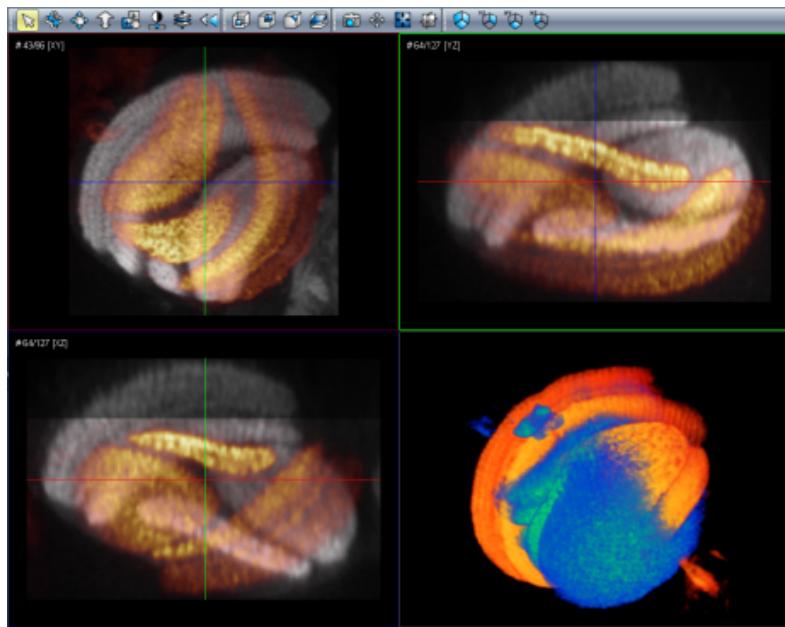
## The Toolbox

Below the label window, a toolbox provides access for the registration tools. Depending on the currently selected tool the area below the tool bar will show additional controls. The registration tools are activated only if the Primary and the Overlay are both available and the tools are applied to the Overlay data set. Two tabs are available in the toolbox: the Automatic and the Manual Registration.



### Automatic Registration

The interface and the functionalities of the Automatic Registration tools are the same as the *AffineRegistration* module. Please see its documentation for a complete description and tutorial. In addition the *Monitor* toggle allows the user to track the actual registration progress.



**Figure 22.15:** The Multi-planar Viewers.



### Transform Editor Options

The interface and the functionality of the Transform Editor Options corresponds to the dialog of the *Transform Editor*. Please refer to the *Transform Editor* documentation for a complete description.

### Information Area

At the bottom of the control panel a status area informs the user about the data being visualized.

## 22.11.2 The Viewers

The Multi-planar Viewer has three 2D viewers and one 3D viewer, similarly to the *Segmentation Editor*. By default, the viewer is displayed in 4-viewer mode, but the user can activate a 1-view only by double-click the viewer of choice. To switch back to 4-viewer mode just double-click on the single viewer again. Right-clicking on a viewer activates the context-menu through which the user accesses tools and visualization options:

- Navigation toolbar functionalities
- Tools and Cropping tools
- Display settings as 2D and 3D bounding boxes, crosshair, info tags

### MPR viewer

In this viewer the image data are displayed either as a conventional thin slice, i.e. data are shown from a definite plane within the volume, or as a thick slice, i.e. a slab with user defined-thickness. With thick slicing a maximum intensity projection of the gray values within the slab is displayed or the user can set the MinIP or Average (see Multi-planar Viewer overview). The thickness of the slab can be set with the slider in the control panel. The data window can be set with the *Window level* tool by clicking the correspondent icon in the toolbar. The slice number, the total number and the orientation of the slices are displayed in textual form at the top of each 2D slice window. The orientation of the slice can be changed via the View icons in the toolbar. The options in the panel "2D Settings" of the user interface control the visualization in the MPR viewer.

### 3D viewer

The 3D viewer of the Multi-planar Viewer is an independent 3D viewer used to display the data and the label fields together with an optional 3D representation of the MPR slices currently seen in the MPR viewer. The "3D MPR" button in the panel "3D Settings" controls the activation of the MPR slices in the 3D viewer. The border colors of the slices denote the different directions (red=x-axis, green=y-axis, blue=z-axis). The options in the panel "3D Settings" of the user interface control the visualization in the 3D viewer.

### Navigation Tools in MPR Viewer

 *Rotate*: rotate the thick slice around selected point or center of slice, if no point is selected.

 *Translate*: translate the slice.



*Zoom*: zoom in/out.



*Window level*: window leveling (brightness/contrast).



*Browse slices*: browse slices through the MPR.

### Navigation Tools in 3D Viewer



*Rotate*: rotate scene.



*Translate*: translate scene.



*Zoom*: zoom in/out.



*Window level*: window leveling (brightness/contrast).



*Browse slices*: N/A.

### 3D Clipping Tools reference



*Clip Plane*: clip a plane inside the volume.



*Clip Corner*: clip a corner of the bounding box (similar to *CornerCut*).



*Clip Slab*: clip a slab of the volume.



*Clip Box*: clip a box inside the volume.

### Registration



*Manual Registration*: activate the manual registration.

### 2D Slices Functionalities



*Center View*: center the view.



*Fit to Window*: fits the slices to the window.



*Reset Plane Position*: reset the position of the slices.

## 22.12 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 *Parameters* in chapter *Program Description* of the Amira 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.

## 22.13 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. For most of these modules it is possible to display the plot within Amira's viewer if one connects the *ViewerPlot* module to it. While the data-generating modules control the initial settings of the plot window, the user can freely adjust these settings afterwards.

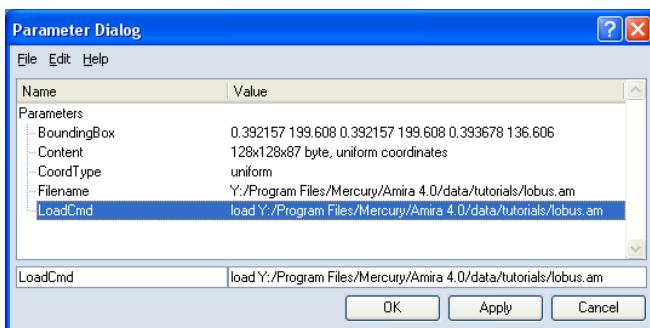
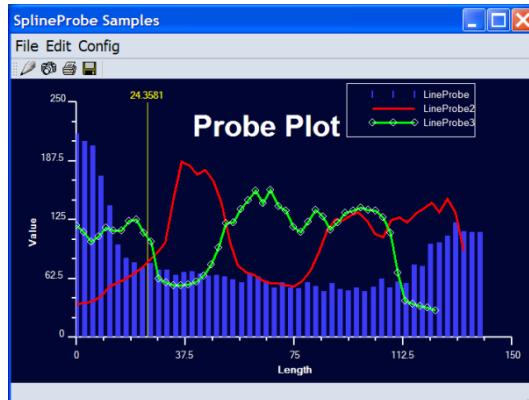


Figure 22.16: User interface of Amira's parameter editor.



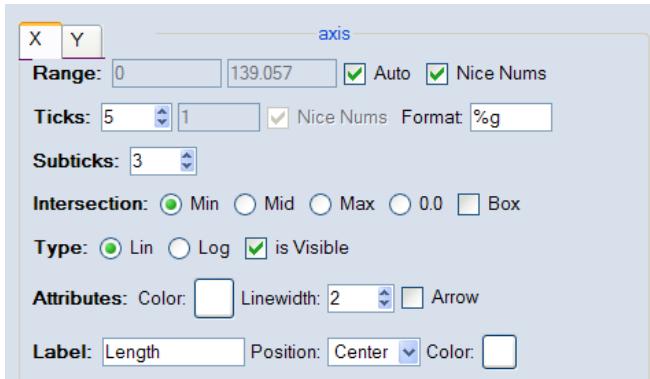
**Figure 22.17:** Amira’s plot window.

### 22.13.1 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 es-



**Figure 22.18:** Dialog for editing plot objects. On the right hand side the controls for editing axis parameters are shown.

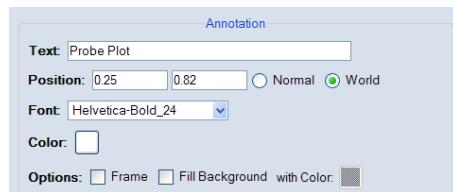
tabulates, together with the preceding *PlotArea*, a window-viewport (world coordinates to normalized coordinates) transformation. Note that the objects must be kept in the following sequence: 1. *PlotArea*, 2. *Axis*, 3. *Curves* and *Marker lines*. Legends display all succeeding curves in the sequence 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*.

## 22.13.2 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.



**Figure 22.19:** Editable annotation parameters.

### 22.13.2.1 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 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.

### 22.13.2.2 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 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.



Figure 22.20: Editable legend parameters.



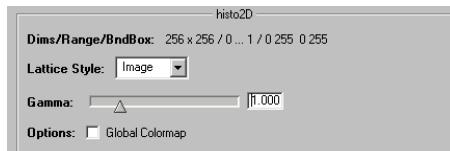
Figure 22.21: Editable marker line parameters.

### 22.13.2.3 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).

### 22.13.2.4 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 or vertically in the plot window by pressing the left mouse button



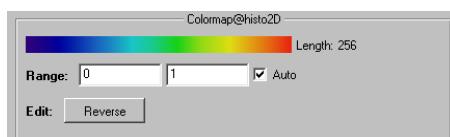
**Figure 22.22:** Editable lattice parameters.

on the marker line, moving the mouse, and releasing it at the new 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.

#### 22.13.2.5 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.

#### 22.13.2.6 Editing colormap parameters



**Figure 22.23:** 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.

#### 22.13.2.7 Editing (analytical) curve parameters

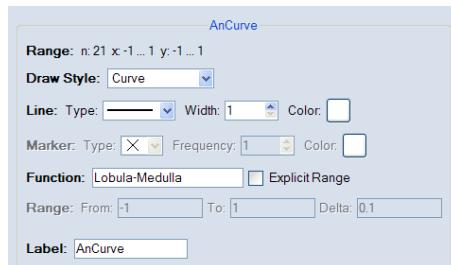


Figure 22.24: Editable (analytical) curve parameters.

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 range of the computation by activating the *Explicit Range* toggle and entering the appropriate range values. There are many built-in functions such as  $\sin$ ,  $\cos$ ,  $\tan$ ,  $\sqrt{x}$ ,  $\exp$ , ... which can be used in formulas.

#### 22.13.3 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) or 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.

### 22.13.4 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).

### 22.13.5 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.

#### 22.13.5.1 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.

### 22.13.6 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.

### 22.13.7 Specific Option

For the *SplineProbe* and *LineProbe*, a specific plot toolbar is available that let you choose the X axis type:

- *Length* : The X axis represents the length of the curve (default).

- *x-coord* : The X axis represents the x coordinate of the curve sampled points.
- *y-coord* : The X axis represents the y coordinate of the curve sampled points.
- *z-coord* : The X axis represents the z coordinate of the curve sampled points.

## 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.

-windowsize: 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>`

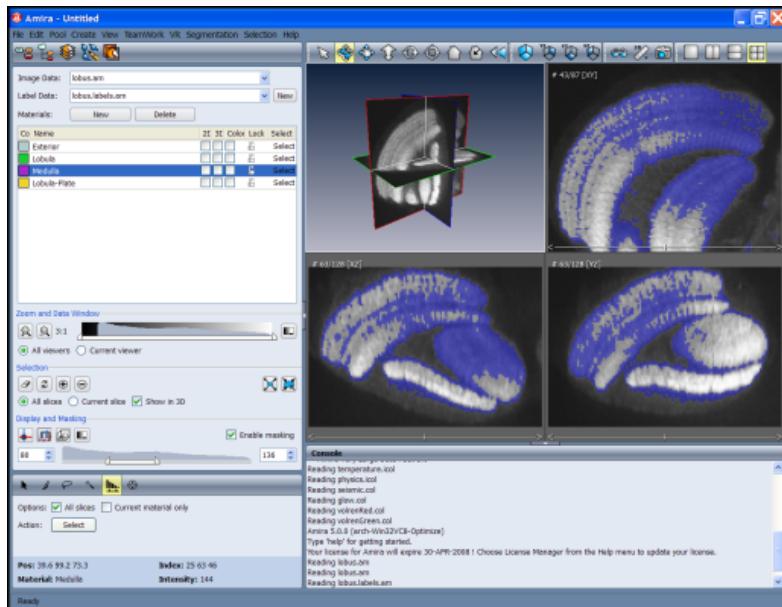
Sets the labels of the ticks along an axis. It is available for objects of type: {*Cartesian Axis, Polaraxis*}. If there are fewer labels than ticks the remaining ticks will be unlabeled.

## 22.14 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:



**Figure 22.25:** The Image Segmentation Editor.

- Overview of the segmentation editor
- Manipulating the material list
- Working in 4-viewer mode
- Selection tools
- Segmentation tools
- Selection filters
- Label filters
- Key bindings

#### 22.14.1 Overview of the Segmentation Editor

In order to activate the segmentation editor, press the Segmentation Editor icon in the sub-application toolbar or the button of a selected *Label Field*. Then an

instance of the editor will be created and a new window as shown in Figure 22.25 pops up. The major parts of this new window are:

- **Data area:** Two drop-down menus show the loaded Image Data and Label Field. You can load new data and label files directly from the menu bar. To create a new label field click on the *New* button.
- **Material List:** In the central part 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. For each material you can turn on/off the 3D and 2D view.
- **View, Selection and Display:** Below the material list several settings for viewing, voxel selection and masking are available. The settings control how the selected voxels are displayed, and activate also the mask for rapid region selection.
- **Tool Box:** Several tools for interactive manipulation of the segmentation can be selected here. 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 1-viewer or 2-viewer mode. 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 and 3D interaction. 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

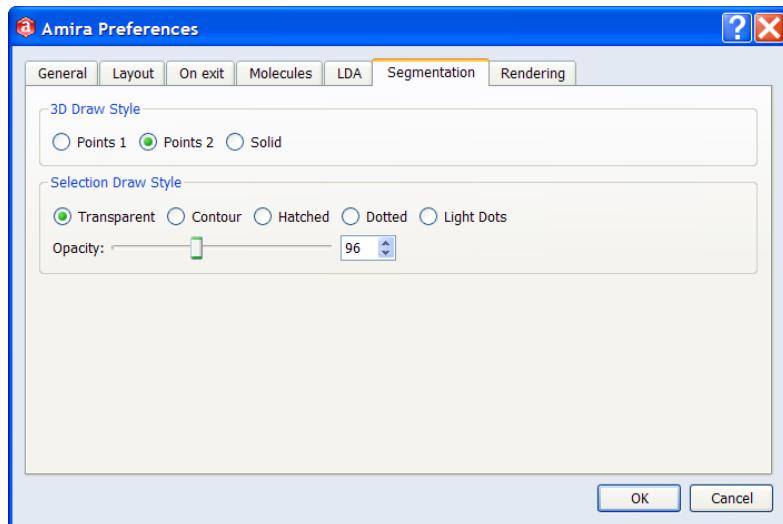
Two segmentation-specific menus are added to Amira's main menu bar: *Segmentation* and *Selection*. The entries of the *Selection* menu are described in a separate section below. Menu entries with dots (...) after their name open an additional dialog.

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.
- **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.
- **Current Viewer:** You can select the active viewer. To activate a viewer you can always just click it.
- **MaterialStatistics:** This entry brings up a dialog which lists the results computed by the MaterialStatistics module on the current data set and segmentation. For an explanation of the modes, see the documentation of the *MaterialStatistics* module.
- **Fill holes:** Invokes a label filter. See *Label filters* for more details.
- **Remove islands...:** Invokes a label filter. See *Label filters* for more details.
- **Smooth labels...:** Invokes a label filter. See *Label filters* for more details.

The *Selection* menu provides a number of filters and operators, which modify the current selection. They are described in the *Selection filters* section in details. Menu entries with dots (...) after their name open an additional dialog.

It is possible to specify various preferences in the *Edit/Preferences* menu of the menu bar.

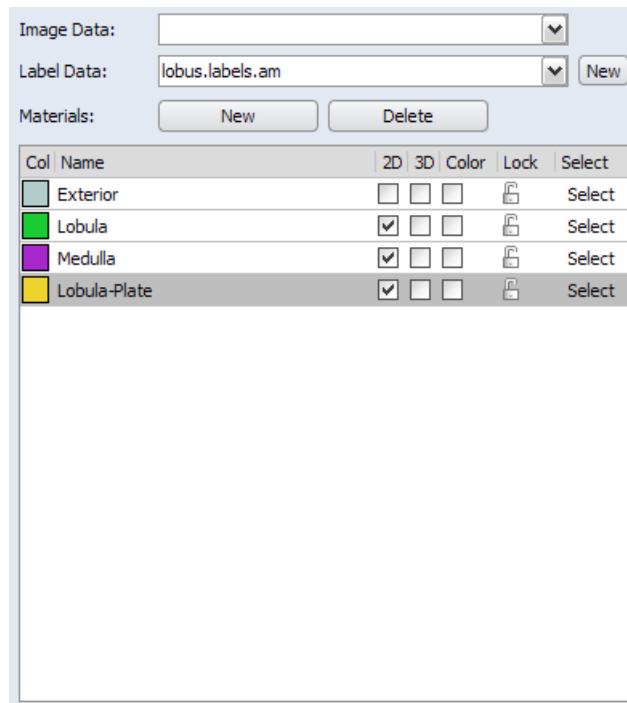


**Figure 22.26:** The Preferences of the Segmentation Editor.

- **3D Draw Style:** How the currently selected materials are displayed in the 3D viewer window, when the checkbox in the Material List is selected.
- **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.
- **Selection Draw Style:** How the currently selected materials are displayed in the 2D viewer window.
- **Transparent, Contour, Hatched, Dotted, Light Dots:** These radio buttons allow you to choose the default draw style used for drawing the currently selected voxels in the 2D viewer windows.
- **Opacity:** Set the opacity of the displayed regions in the 2D viewer window.

## 22.14.2 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.



**Figure 22.27:** The material list.

Press the *New* button to add a new material to the list (note that not more than 256 materials are accepted). 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. If the color checkbox is enabled on right of the material name, the voxels are rendered with the selected color.

Check the 2D and 3D checkboxes to the right of the material name to switch the material visible or invisible in the MPR and in the 3D viewer. The rendering windows and colormap can be set in the *Zoom and Data Window* and in the *Display and Data Masking* areas.

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 if *All slices* is selected or not).
- 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 if *All slices* is selected or not).
- 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.
  - *Preferences*: Opens the Segmentation Editor *Preferences*.
- **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.

- **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 two different materials with the same name.
- **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.
- **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.
- **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.
- **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.
- **Lock All:** This locks all materials.

### 22.14.3 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

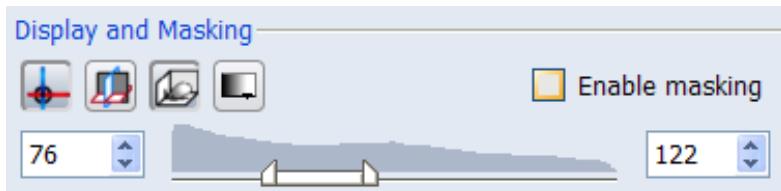


Figure 22.28: The Display and Masking area

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.

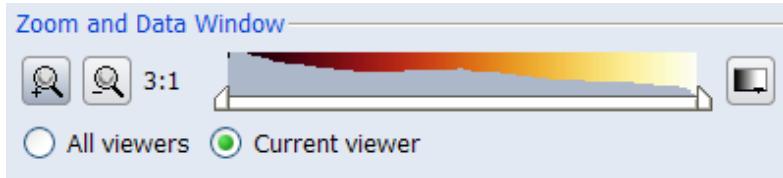
### The 3D viewer

The lower-right viewer is an independent viewer where several special segmentation actions and selections are supported. You can activate the 3D rendering of the data by clicking the *3D Volume Rendering* button of the *Display and Masking* area. Whereas the *3D MPR* button is a tri state button alternating the rendering of the 3D MPR slices from opaque to transparent and finally to turned off, by pushing the button repeatedly. Within the *Display and Masking* area you can also turn on/off the crosshair of the MPR, set the colormap and the 3D rendering window. The crosshair button activates the crosshair tool. It simply displays a crosshair in all three orthogonal viewers. The colors denote the different directions (red=x-axis, green=y-axis, blue=z-axis). You can click and grab one of the axes or their crossing point. The checkbox *Enable masking* enables the masking of the data, which is a useful feature for several segmentation tools.

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. Deselect the *Show in 3D* toggle to disable display of the highlighted region in the 3D viewer. In the *Material List*, toggle the 2D and 3D checkboxes to switch the material visible or invisible in the 3D viewer.

### 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



**Figure 22.29:** The Zoom and Data Window area

(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. If the *Current viewer* button is selected then the zoom buttons only apply to the currently active viewer.

#### Data range, histogram, and colormaps

The histogram displays the distribution of values in the 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. The range slider can be dragged beyond the minimum and maximum values of the image, causing the histogram to rescale accordingly. If the *Current viewer* button is checked, then the slider changes the data window of the currently active viewer only.

The colormap of the image can be changed from the default gray ramp to another colormap by pressing the *Select colormap* button.

#### 22.14.4 Selection Tools

The basic principle of the segmentation editor is to first select voxels and then to assign, subtract, and replace etc. the selected voxels to or from the current active material. A basic set of tools working with selections are described in the following:

**All slices / Current slice:** Lets you choose to operate on *All slices* or just the *Current slice* i.e., if you have selected *All slices*, all the selection tools described below will operate in 3D mode. For example, the *Clear* button clears the whole 3D selection, not only the current slice. In order to warn the user, all selection



Figure 22.30: The Selection area

tools will show red colored icons instead of gray ones indicating that voxels are selected, which aren't visible within the current *2D viewer* slices.

**Show in 3D:** Lets you display the current selection within the *3D viewer*. On slower computers or for very large data sets, this may become slow.

**Clear:** This button lets you clear the current selection. If *All slices* 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.

**Previous:** This button displays in the active *2D viewer* the slice previous to the current slice (lower slice number) that contains at least one selected voxel

(red overlay). If no such slice exists, clicking this button doesn't change the current *2D viewer* slice position at all.



**Next:** This button displays in the active *2D viewer* the next slice after the current slice (higher slice number) that contains at least one selected voxel (red overlay). If no such slice exists, clicking this button doesn't change the current *2D viewer* slice position at all.



**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. You can access this command also through the item *Selection* in the menu bar.



**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. You can access this command also through the item *Selection* in the menu bar.

## 22.14.5 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 *Edit/Preferences* menu. Alternatively, selected areas can be visualized using the same draw styles as ordinary labels. The different draw styles are listed under the popup menu clicking the right mouse button over the selected colour in the *Label List* (see also *Manipulating the Material List*). However, in most cases the default transparent display is most appropriate because then selected areas cannot be mistaken for labeled regions. If the middle mouse button is used the mouse cursor takes on the shape of a hand (only on the 2D slices), then the tool lets you translate the current slice or rendered volume.

You can also select a certain data range to be used as a mask for the segmentation tools. The mask allows you to focus your work on a specific range of the gray-values. This is very useful for brush, lasso and thresholding segmentation (see below). To enable the masking, check the *Enable masking* box in the *Display and*

### *Masking area.*

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. If the middle mouse button is used the mouse cursor takes on the shape of a hand. Then the tool lets you translate the current slice. If the *Hide cursor* option is selected, the cursor is hidden while moving over the current slice.

It is also possible to fence an area defined 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 the lines traced over with the brush.

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.

If the *Enable masking* option is selected, the brush will paint only pixels whose values fall between those in the range slider.

The *Square brush* option will paint, as the name suggests, a square region instead of the default circular region. All of the other options will work with the square brush if they are selected, like the limited range, etc.



### Lasso

The *Lasso* lets you define an area by generating a closed contour curve in 2D or 3D. Usually, if the *Freehand* button is selected, you draw such a curve freehand with the left mouse button pressed, but it is also possible to fence an area in the slice 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 (this is the default), the line segments are automatically fitted to 2D 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.

*2D Shape:* If the *Ellipse* or *Rectangle* buttons are selected, the lasso tool can be used to select an elliptic or a rectangular region in one of the 2D viewers. The *Auto trace* option is disabled for these shapes. If the *Alt* key is pressed, the shape starting point is the center instead of the upper left corner.

*3D Mode:* These buttons apply to the behavior of the lasso when used in the 3D viewer. If *Inside* mode is selected, lasso operations affect voxels inside the contoured region, while *Outside* mode does the same on voxels outside the contoured region (for the 2D viewers the lasso works always in *Inside* mode). In the 3D

viewer the default operation is to replace selected voxels if, *Inside* mode is enabled and to deselect selected voxels, if *Outside* mode is enabled. Press *Shift* to select all voxels outside the contour.

The default operation in the 2D viewers is replacing a selection, i.e. any contour drawn replaces an existing selection. Press *Shift* while drawing a contour to extend an existing selection. Press *Ctrl* to subtract from an existing selection.

All the operations described above can be constrained to the range of the masking slider, if the *Enable masking* option is selected.



### Magic Wand

This tool performs a so-called region growing either in 2D or in 3D. 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. It is possible to modify the range even after the seed voxel has been selected.

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.

If the *Enable masking* option is selected, this tool will select only pixels whose values fall between those in the range slider.



### Threshold

Selects voxels by threshold segmentation. The minimal and maximal image intensity can be specified in the *Display and Masking area*. When you click the *Select* button, all voxels falling into the interval are selected. If the *Current 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. Note that the *Enable masking* box has to be checked in order to use this tool.



### 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 field 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.

## 22.14.6 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 or in the whole volume. (The shortcut *I* works either on the current slice or the whole volume, depending on the state of the *All slices/Current slice* radio button in the *Selection* area.)

- **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.

- **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*.

## 22.14.7 Label Filters

Under the *Segmentation* item 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 *fraction* 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 fractions of the surrounding regions with respect to the total number of voxels adjacent to the island are calculated and compared with the *fraction* threshold. The island is assigned to the region with the highest fraction value greater than or equal to *fraction*. If no region exceeds the *fraction* threshold, the island remains untouched. Note that the *fraction* 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 fraction 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 protuberant 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.

## 22.14.8 Segmentation Editor - 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** - Threshold tool

**5** - Blow tool

**6** - Crosshair tool

- **Others**

**U** - Undo

**Q** - Toggle 3D button (switch between all slices and current slice)

**V** - Toggle 1- and 4-viewer mode

**L** - Draw limit line

. - 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)

## 22.15 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 22.31:** 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. Note that the port will not be created if during simplification errors occur. In this case a message in the console will be printed.

**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> <lambd>`

This command 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*.

## 22.16 Surface Editor



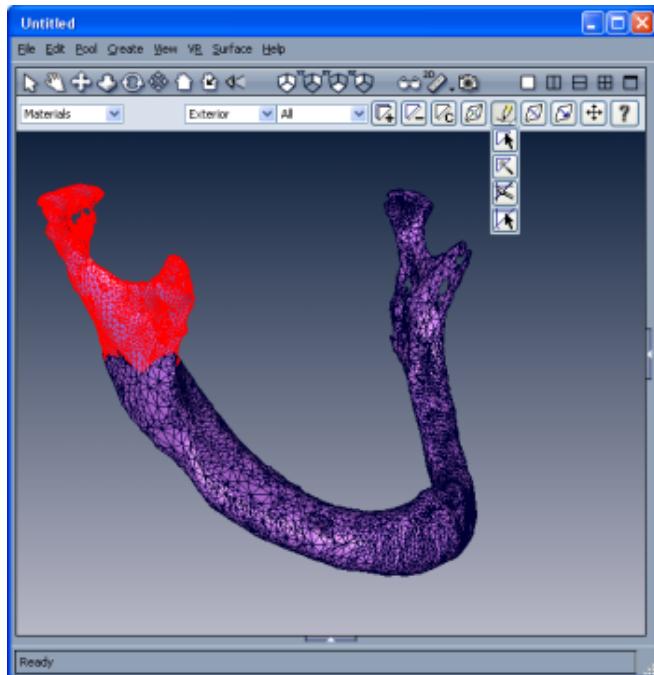
The surface editor allows you to modify a triangular surface in several ways, e.g., to remove or refine triangles, to flip edges or move points, or to set boundary ids for individual triangles. It also allows you to check a surface for intersecting triangles or for falsely oriented patches.

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 22.32. 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



**Figure 22.32:** User interface of the surface editor.

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*
- *mouse tools for selecting triangles and editing the surface*

## 22.16.1 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.

### 22.16.1.1 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*: Smoothes 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.
- *Fill Hole...*: This menu entry shows a drop down menu in the properties panel with a selection of hole fill algorithms. By pressing the fill button the currently displayed hole is triangulated using the selected hole fill algorithm.
- *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.

### 22.16.1.2 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.

### 22.16.1.3 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*.

#### 22.16.1.4 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.

### 22.16.2 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).

- *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.
- *Non-manifold test:* This test detects non-manifolds on surfaces and displays every non-manifold location including the incident triangles.
- *Holes test:* This test detects holes on triangular meshes, which can be closed by the triangulators available from the *Fill Hole...* menu entry.

### 22.16.3 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.

- *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.

## 22.17 Surface Path Editor



This editor allows creating and editing *paths* on triangular surfaces. Surface paths can currently be used only to measure distances on surfaces. To do so, right-click the surface object and select *CreateSurfacePath* from the *->Measure* submenu. This creates a new *SurfacePathSet* and automatically opens the *Surface Path Editor*. Note that the following operations can only be applied, if the editor is selected in the *Pool*. In this way, you may interact with the surface in the usual manner when deselecting the editor.

The basic operations are:

- **Creating a path:** Press Shift and left-click on the surface. A path can only be created if no other path is selected. Paths can be deselected by left-clicking onto the surface.

- **Adding a node to a selected path:** Press Shift and left-click on the surface. A node can only be added to a path if it is the only selected path. The node will always be added after the last added node.
- **Inserting a node into a selected path:** Press 'I' and left-click on the path at the point where you want to insert the node. The path must be the only selected path.
- **Selecting a path:** Left-click on the path.
- **Selecting a node:** Left-click on the node. A path must be selected to select one of its nodes.
- **Selecting more than one path:** Press 'S' and left-click on the next path you also want to select.
- **Deselecting a selected path:** Press 'S' and left-click on the path.
- **Moving a selected node:** Press Shift and left-click on the point of the surface you want to move the node to.
- **Delete selected items :** Press 'D' or the Delete button. If a node is selected, it will be deleted. Else all selected paths will be deleted.

Once a *SurfacePathSet* with one or more surface paths has been created their length can be queried by selecting a path and read its length from port *Info*. If multiple paths are to be measured at once the *SurfacePathSet* can be converted to a *SpatialGraph* object using *Compute->LineSetToSpatialGraph* and then analyzed using a *SpatialGraphStatistics* module.

## Ports

### Info



**Info:** Active 0: nPoints 548, length 343.851257, nCPs 5

This port informs the user about a selected path, if a single path is selected. If no path yet exists, the user is also informed how to create a new path.

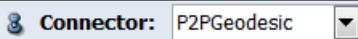
### Advanced options



**Advanced options:**  show

By default, advanced options are hidden. They will be displayed if the *show* toggle is checked.

### Connector



Select a strategy to connect two points. Several different connectors exist. Choose one from the list below.

- **Dijkstra**: connects two nodes using nodes computed with Dijkstra's shortest path algorithm. The path will therefore only cover edges and vertices of the surface. Note that if a surface scalar field is connected to the surface, the Dijkstra connector will use the edge weights provided by this scalar field instead of the edge lengths.
- **TriangleDijkstra**: connects two nodes using nodes computed with Dijkstra's shortest path algorithm. In contrast to the *Dijkstra* connector, the intermediate nodes generated by this connector lie in the middle of the surface edges. The segments cross the triangles, hence the name *TriangleDijkstra*.
- **PlaneCut**: uses the two triangle normals of the points that are to be connected to compute a plane. This plane is intersected with the surface and the shortest connection of the points within this intersection is taken as path. The plane is chosen such that the angle between the two surface normals and the plane are equal.
- **Geodesic**: computes the shortest geodesic path between two points. The used algorithm is extremely slow. If time restrictions are not critical, prefer this geodesic connector, otherwise try one of the following three connectors.
- **P2PGeodesic**: like Geodesic but runs a little faster.
- **ApproxGeodesic**: an approximation of the Geodesic algorithm that runs a lot faster.
- **LocallyShortest**: computes a geodesic between the two points. However, it might not be the shortest one, but only the locally shortest one.
- **NonManifold**: computes path along non-manifold edges of surface. A path can only be computed if the control points belong to the same non-manifold.

### Control point type



When a new control point is created by Shift-clicking onto the surface, the

control point will be of the type that is selected here. If *Vertex* is chosen, then the control point will be created on the nearest vertex on the surface, for *Edge* the nearest point on the closest edge will be chosen.

### Action



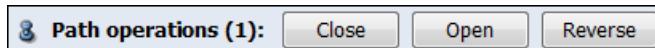
**Undo:** Undoes an editing step.

**Redo:** Redoes an operation after *undo*.

**Delete Path(s):** Delete all selected paths. The user can select all paths by pressing the *Select all* button.

**Delete Vertex:** Delete the selected vertex.

### Path operations (1)



**Close:** Connects the start and end node of the active path. As there is no last node, nodes cannot be appended to a closed path.

**Open:** If the selected path is closed it is either opened on the selected node or, if no node is selected, it is opened at the node that was the last node before the path was closed. If the selected path is already open and a node in this path is selected, the path is split in two at this node.

**Reverse:** Reverses the active path. After reversing a path, new nodes will be appended at the beginning and not at the end of the reversed path.

### Path operations (2)



**Merge:** Merge tries to connect endpoints of selected paths and make them one path.

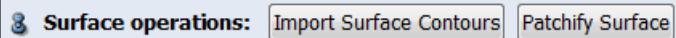
**Entangle:** Computes a node for each point of intersection of line segments in the path set. If two nodes of the paths are similar or equal in coordinate and type, they are replaced by a single node. Entangled nodes can be moved like normal nodes, but when moving them, all paths that belong to it are changed as well.

**Untangle:** Opposite to *Entangle*. If a node occurs in more than one path it is duplicated once for each path.

**Split at Intersections:** If two paths not only intersect in one point, but in a line segment, fix removes the overlapping part of one of the paths and splits them

into two at these points. In the resulting path set, all paths only intersect in points.

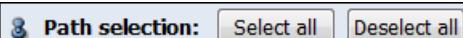
## Surface operations



**Import Surface Contours:** Creates paths along the boundary of surface patches.

**Patchify:** Decomposes surface into several patches if surface paths completely surround regions on the surface. If the path nodes do not lie on vertices only, this operation cannot be applied directly. Using the *Snap to edges* button, the user can convert the path into a vertex path for which patchification can be applied.

## Path selection



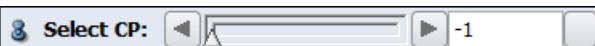
Select and deselect all paths. Since most operations work on selected paths only, these buttons enable the user to quickly select all paths. Deselection is equally important.

## Select path



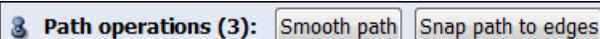
Select a specific path by setting the respective path id. A value of -1 deselects all paths.

## Select CP



Select a specific control point of the selected path by setting the control point id. A value of -1 deselects all control points.

## Path operations (3)



**Smooth path:** This option smooths a path, whereby the end nodes will be fixed.

**Snap path to edges:** Snap transforms nodes that are near another node type, i.e. edge or vertex, into this type. If, for example, a triangle-node is very close to an edge, then it is moved to the closest point on the edge and becomes an edge-node. Here, very close means, the difference of the coordinates of the nodes is

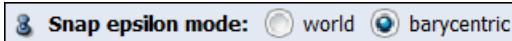
smaller than *epsilon*, where the epsilon can be defined using the *Snap epsilon* port. A triangle-node can become an edge or vertex node and an edge-node can become a vertex-node but not vice versa. Snapping paths to vertex nodes is necessary to directly allow patchification of the surface using the surface path.

### Snap epsilon



Defines the epsilon for snapping nodes to their nearest edges or vertices.

### Snap epsilon mode



When converting nodes to other types one can compare the absolute or relative coordinates. When this port is set to *world*, the epsilon is interpreted as an absolute value and when snapping, world-coordinates are compared. Else the snap-method compares barycentric coordinates.

## 22.18 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 22.33.

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.



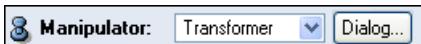
**Figure 22.33:** Dialog opened by the transform editor.

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 re-apply a transformation using the Tcl commands `getTransform` and `setTransform`.

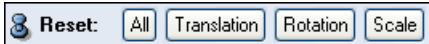
## Ports

### 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.



# 23 Molecular Option

## 23.1 MolTrajectoryBundle Editor



This tool allows you to edit the data entries contained in a *MolTrajectoryBundle* data object.

Each row shows a set of values for one trajectory in the bundle. The first column contains the current index of each trajectory. The second column contains the name. All other columns represent data entries. Double clicking one of the cells in the index columns will set the current trajectory in all downstream MolTrajectory objects to the clicked trajectory.

It is possible to sort according to each columns values by clicking on the header above the column. A certain sorting order can be applied to the indices of the trajectories with the *Trajectory/ApplyOrder* menu.

The names and data entry columns can be edited by double clicking the cell. The index column is not editable.

With CTRL-C the cell values of the current selection can be copied to the clipboard as a tab delimited string (which can be handled by most spreadhseet applications). CTRL-V overwrites the current selection with a tab delimited string in the clipboard. This requires that the number of lines in the clipboard is identical to the number of selected rows and that the number of tab delimited tokens in each row is identical to the number of cells that are selected for the corresponding row.

### Trajectory Menu:

New trajectories can be added to the bundle with the *Trajectories/Add* menu. A requester will appear that lets you pick any trajectory in the object pool. With the *Trajectory/Remove* menu you can remove all trajectories from the bundle for which at least one cell is currently selected, and with *Trajectory/Restrict* you can restrict the bundle to the selected trajectories.

### Data Menu:

New data can be added with the *Data/Add* menu. *Data/AddSelection* will create a new data column containing '0' for all unselected trajectories and '1' for all

selected ones. With the *Data/Remove* menu you can remove all data from the bundle for which at least one cell is currently selected.

#### Selection:

Items can be selected manually by single clicking a cell (SHIFT and CTRL can be used as modifiers). The section at the bottom of the editor allows you to select trajectories based on *atom expressions*. The *Add* and *Remove* buttons will add or remove the trajectories which are matched by the given expression to or from the selection. The *Repalce* button will replace the selection with the matching.

## 23.2 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.

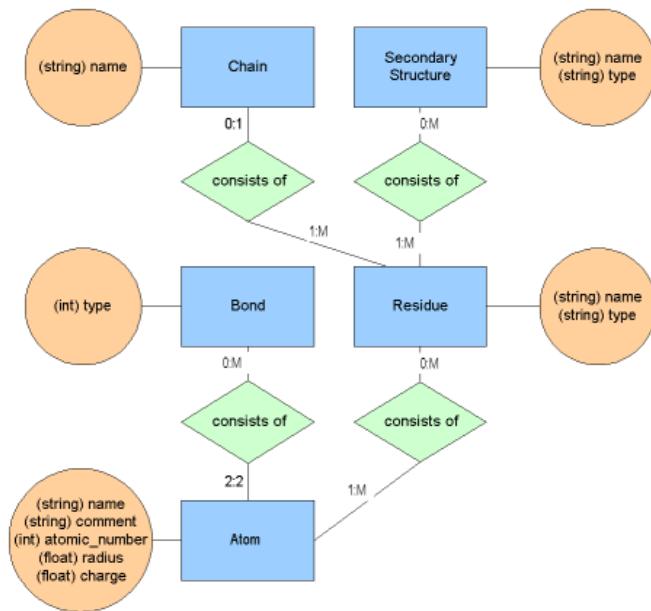
### The Level and Attribute Concept of Amira

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 23.1 shows the entity relationship model for the most common grouping levels.

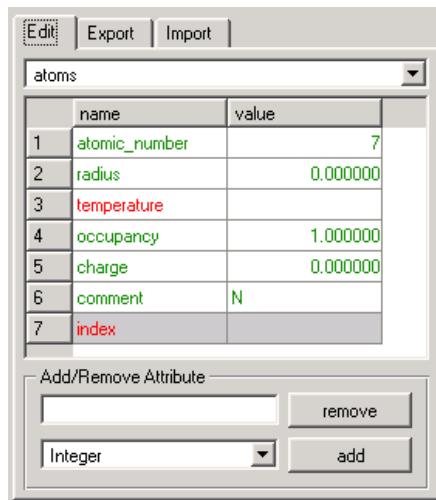
Each level has a set of attributes. Attributes are properties of groups of type string, integer, or float. The number of levels and attributes depends on the file format from which the molecule is read.

### The Edit Tab

The edit tab of the attribute editor (Figure 23.2) offers two different tools:



**Figure 23.1:** Entity relationship model of the most common levels

**Figure 23.2: Attribute Editor**

- addition, deletion, or renaming of attributes from a level
- 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 formatting 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=%(residues,index)
%()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 Option.

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] .r--" *%" "%(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.

## 23.3 Molecule Data Editor



This tool allows you to edit the data entries contained in a *Molecule* data object. The left column shows the name and the right column shows the value of each data element. Names and values in the table can be edited by double-clicking

into a cell. With the menu entry *Data/Delete* you can remove all data elements currently selected. *Data/New* will create a new element.

Outside of the editor, data element can be accessed with the `setData` and `getData` commands of the *Molecule*.

## 23.4 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 the set if selected atoms. Atoms can be selected by using the *selection browser* or by directly clicking on atoms in the viewer.

The molecule editor, Figure 23.3, 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 exists 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 moleclem which are derived from fixed properties representing the atomic configuration, namely the valency of the atoms, its atomic number and its formal charge. The atomic number is a mandatory attribute of the atom level and the valency is represented as explicit bonds in the molecular data structure and as an additional number of implicit hydrogens.

The implicit hydrogen number and the formal charge can be supplied as attributes of the atom level. When the editor starts, this information will be read from the `formal_charge` and `implicit_hnum` attributes. If no such attribute exists, they are

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. When the editor is closed by pressing the OK button the current state will be written into the atom level attributes.

## 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 hydrogen atoms. Additionally labels can activated which show important atomic properties:

- None: Labels deactivated.
- Index: The Amira internal atom index (counted from 0).
- Standard: Modes formal charge and radicals combined.
- Formal charge
- Implicit H-Num: Number of implicit hydrogens.
- Lone Pairs: Number of lone electron pairs.
- Atom Type: MMFF94[1] atom type. Atom typing for certain atoms may fail for rare atoms or uncommon or incorrect chemical environments.
- Oxidation number
- Radicals: Shows an asterisk for atoms that contain unpaired electrons.

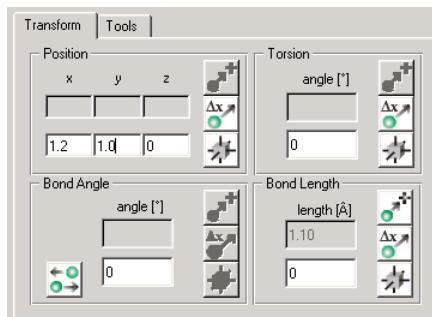
## The Tools Menu

- 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 non-polar H: Makes all non-polar hydrogens implicit.
- Strip H<sub>2</sub>O: Removes all water molecules (including unbound O or H atoms and OH molecules).
- Kekulize: Assign a Kekule structure to the molecule, in effect replacing all aromatic bond systems with alternating single and double bonds.
- Kekulize Non-Rings: Assigns a Kekule 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 system for aromaticity, 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.



**Figure 23.3:** Transform Tab

- *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. 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.

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:

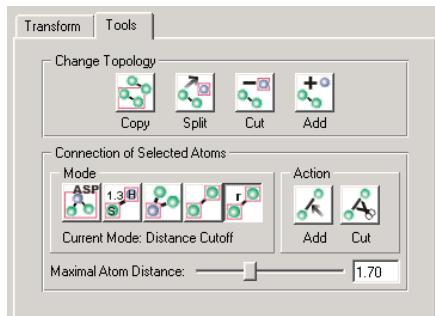


Figure 23.4: Tools Tab

## 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.

- *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 ' $\rightarrow$ ' 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 an 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<sub>2</sub>' 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 ' $\approx$ ' an aromatic bond. If the atoms were already connected the bond order will be simply changed. We advise against assigning aromatic bonds by hand. 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 Kekule 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

- *OK* will close the editor and accept all changes made to the molecule.
- *Cancel* returns from the editor restoring the last accepted state.
- *Reset* resets the molecule to the last accepted state without canceling the editor.
- *Apply* accepts all changes. This means that resetting will return the molecule

- to the current state.
- *Undo* undoes the last change. Currently, for efficiency reasons, this is only possible for coordinate transformations of the transform tab.



**Figure 23.5:** Button Group of the Molecule Editor

## Creating Molecules from Scratch

The editor can only be invoked when a molecule already exists. To create a molecule from scratch you can use the following Tcl command:

```
newMolFromSmiles <SMILES>
```

which will generate a molecule from a smiles string [2]. If you want to use square brackets '[' and ']' you need to enclose the SMILES in curly brackets to avoid Tcl command substitution. This is also necessary for the bond direction identifier 'wich is usually interpreted as an escape character. Examples:

```
newMolFromSmiles c1ccnc(C)c1
newMolFromSmiles {C[O-]}
newMolFromSmiles {F/C=C\F}
```

## Scripting Interface

Most functions of the MoleculeEditor are scriptable. An editor associated to a molecule may be created with the command

```
set <myEditor> [<moleculeObject> createMoleculeEditor]
```

where <moleculeObject> is the name of the molecule object as it appears in the objectpool and myEditor is an arbitrary tcl variable that will contain the name of the created editor object and which can then be used with the following commands:

```
addH
addPolarH
removeApolarH
removeH
removeWater
replaceAtomAtomicNum <atomicNumber>
replaceAtomSmiles <smilesStr>
assignFormalCharge
assignHNum
assignStandardHNum
incCharge
incValency
decCharge
decValency
dekekulize
kekulize
setBondAromatic
setBondDouble
setBondSingle
setBondTriple
minimize
reset
apply
copy
createAttributeMMFFCharge
createAttributeMMFFTType
createLevelAngles
createLevelDihedrals
createLevelOOPDihedrals
createLevelRingSystems
createLevelRings
createMMFFParameterization
```

## References

- [1] Halgren,T.A. Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94. J.Comp.Chem., 17, 490-519 1996.
- [2] Weininger,D. 'SMILES, a chemical language and information system. 1. Intro-

duction to methodology and encoding rules', J. Chem. Inf. Comput. Sci. 28, 31 - 36 1988.



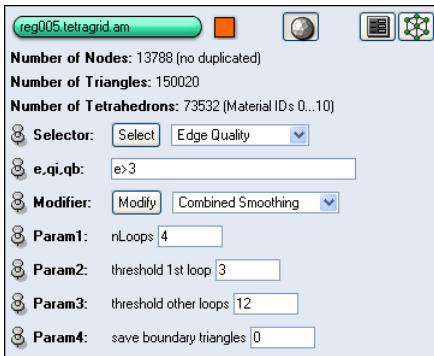
# 24 Mesh Option

## 24.1 Grid Editor

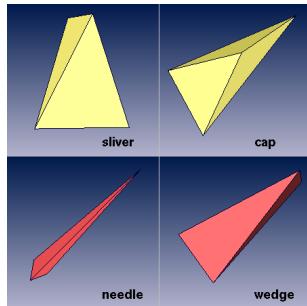


The Grid Editor allows you to analyze the quality of a *tetrahedral grid* according to different quality measures and to semi-automatically improve the grid quality.

To activate the Grid Editor, press the Grid Editor button of a selected tetrahedral grid. A user interface composed of different button controls (Figure 24.1) 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.



**Figure 24.1:** Grid editor for improving the quality of tetrahedral grids.



**Figure 24.2:** Four examples of distorted tetrahedra.

Tetrahedra can be distorted in different ways (see Figure 24.2):

- *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.
- **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.  $d < 0$  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 adj-

cent 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  $> 0$  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.

- **Automatic:** This tool tries to improve the quality of the whole grid automatically with no further user-interaction. If *save connectivity* is set to 1 (true) then it only applies optimization smoothing repeatedly. If it is set to 0, then several of the reconnection methods described above are used and the elements are flipped and smoothed until the algorithm converges.

These improvements can take a really long time to converge but you can feel free to interrupt anytime by pressing the stop-button: This tool uses a *floating threshold* which you can observe in the console-window. That means that the tetrahedrons with worst quality are considered first and the others later when the threshold decreases. When the process is stopped while using a threshold of e.g. 6.0 then all tetra-hedrons with quality worse than 6 have been improved.

If you still detect bad quality elements after the Automatic tool has finished then the source-grid didn't allow improvements there because of vertices fixed on boundaries.

# 25 Microscopy Option

## 25.1 Filament Editor



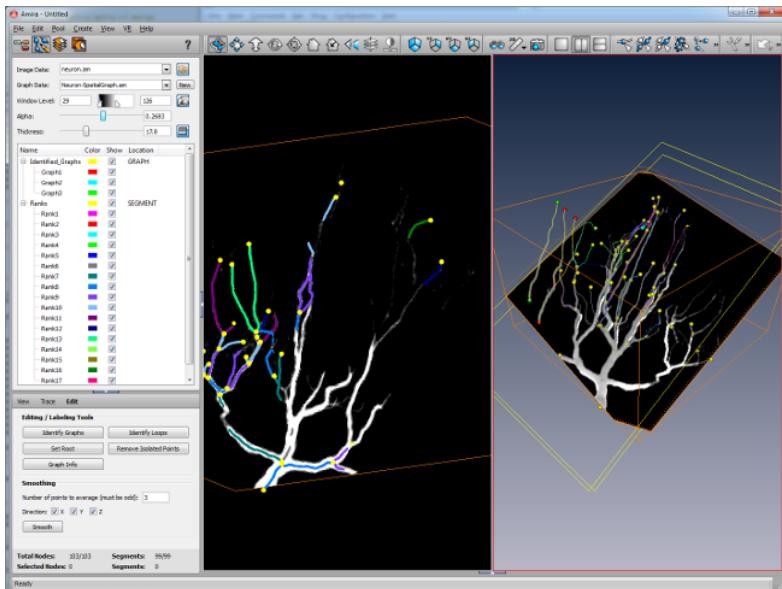
The Filament Editor is a sub-application designed to analyze and quantify 3D images of filamentous structures such as neurons and blood vessel networks. Fiber networks and other filamentous structure can be fairly well modeled as lines with certain thickness. Therefore, the analysis of a filamentous network consists of extracting the centerline of the fibers and measuring their thickness. Results of the analysis are stored efficiently in terms of line segments and branching points with additional information in *SpatialGraph* object. For this purpose the Filament Editor offers automatic and interactive tracing tools. Automatic tools can be conveniently used for images with good object-to-background contrast and will quickly arrive at the desired analysis. Usually, however, images suffer from degradations such as noise and blur so that an automatically traced image will be left with false connectivity and/or missing segments. Interactive tracing tools as well as a full-featured line editor can then be used to correct possible tracing errors. To facilitate interactive tracing the image data are displayed on a thick slice visualization using maximum intensity projection.

The documentation of the *Filament Editor* is divided into the following parts:

- *Overview of the Filament Editor*
- *Basic principles and terminology*
- *The Tool Box*
- *The Viewers*
- *The Label Editor*

### 25.1.1 Overview of the Filament Editor

The Filament Editor is activated by pressing the *Filament Editor* button in the sub-application taskbar. This will change the layout of the main and viewer window as



**Figure 25.1:** The Filament Editor.

shown in Figure 25.1. The main components of this layout are the following:

- **Data Selector:** In the upper part of the main window two drop-down menus allow you to select the image and *SpatialGraph* data objects to be edited. Data have to be loaded into the Amira workspace before they will be available in the *Image Data* and *Graph Data* drop-down menus.
- **2D Viewer Settings:** Controls for window level and thickness settings in 2D slice and 2D segment viewer. It also provides buttons to turn on/off volume rendering and/or slicing in the 3D viewer. When volume rendering is activated, the transparency can be changed using the *Alpha* slider.
- **Label Window:** The major part of the main control panel is occupied by the label window presenting a tree-like representation of the graph. Here the user may define and edit an arbitrary number of labels on different hierarchical levels of the graph data. Use the right mouse button to add, remove, and rename the labels.

- **Tool Box:** Below the label window a tool box provides access for the tracing tools and the graph display options. Depending on the currently selected tool, the area below the tool bar will show additional controls.
- **Info Area:** At the bottom of the control panel a box informs the user about the graph being edited. The information shown are the total number of node/segments and the number of selected nodes/segments.
- **Viewers:** The Filament Editor typically offers three viewers for editing and viewing the data: The 2D slice viewer displays the image data as an arbitrarily oriented slice with variable thickness. In contrast the 2D segment viewer displays the image data along the currently selected segment as a curved slice also with variable thickness. The 3D viewer is an instance of the familiar Amira viewer for 3D visualization of the graph data with optional 3D slice and volume rendering display.

### 25.1.2 Basic principles and terminology

Analyzing images of filamentous networks consists of extracting their skeleton in terms of line segments connected by nodes. Amira can store such data in a dedicated data type called *SpatialGraph*. A *SpatialGraph* object consists of **nodes** and **segments** where nodes are the branching points and endpoints, and segments are the lines connecting the nodes. Segments, in turn consist of **points** describing the three-dimensional course of a segment. A set of segments connected by nodes will be termed a **graph**. A *SpatialGraph* data object can store several graphs. Furthermore, *SpatialGraph* objects can hold scalar and label data on different levels. On the point level scalar data such as the thickness of the fiber can be stored. On the segment, node and graph levels also labels can be defined. Labels are useful to annotate the graph according to topological, structural, and/or functional properties. The Filament Editor provides automatic and interactive tools to extract the graph data from an image stack. The automatic tool (*Auto Skeleton*) takes segmented image stacks (i.e., label fields) as input and extracts the desired centerline using sophisticated thinning algorithms. Segmentation is either done by thresholding the image on the fly using the *Window Level* setting for the 2D slice viewer directly within the Filament Editor, or can be done using thresholding modules (*Compute->LabelVoxel*) or the *Segmentation Editor*. Automatically extracted graphs can then be post-processed using interactive tracing and/or line editing tools. In interactive tracing the user sets key points in the 2D slice viewer along a fiber and the tool finds the centerline connecting both points automatically.

When topology and geometry of the graph have been determined it can be labelled according to user specifications. An arbitrary number of labels can be inserted at the graph, node, and segment level. Each graph, node, or segment can belong to multiple label groups. This can be useful for visualization of complex fiber networks.

### 25.1.3 The Tool Box

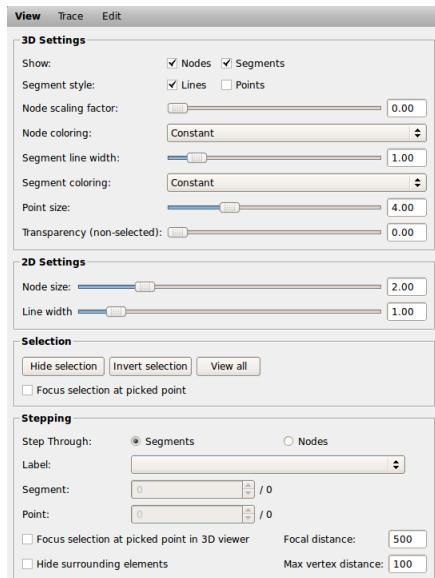


Figure 25.2: The View tab.

## The View tab

### 3D Settings:

The set of display options provided here constitute a subset of the most important ports of module *SpatialGraphView*. To get the full potential of this module you have to switch to the *Pool* and visualize your *SpatialGraph* objects directly there.

- **Show:** Check *Nodes* to display all nodes of the graph data, check *Segments*

to display segments.

- **Segment style:** Check *Lines* to display segments as lines, check *Points* to display the points, check both to display them as points and lines.
- **Node scaling factor:** Set the size of nodes.
- **Node coloring:** *Constant* displays all nodes with the default color. Select a label attribute for coloring, if available.
- **Segment line width:** Set the segment line width.
- **Segment coloring:** *Constant* displays all segments with the default color. Select a label attribute for coloring, if available.
- **Point size:** Set the size of the points.
- **Transparency (non-selected):** Set a constant transparency for nodes and segments that are not selected.

## 2D Settings:

- **Node size:** Set the size of the nodes in the 2D slice and 2D segment viewer.
- **Line width:** Set the line width of segments in the 2D slice and 2D segment viewer.

## Selection:

- **Hide selection:** Hides all selected nodes and segments.
- **Invert selection:** Inverts the selection.
- **View all:** Shows all nodes, segments.
- **Focus selection at picked point:** This check box has three states: When filled, selecting a node or segment in the 3D viewer moves the slice of the 2D slice viewer in the current orientation to the picked position. When checked, the slice moves to the picked position and is oriented tangential to the corresponding segment. When empty, the slice orientation and position are not affected by selecting an item in the 3D viewer.

## Stepping:

This tool steps through all *Nodes* or *Segments* and their corresponding *Points* while keeping the slice of the 2D slice viewer tangential to the current point or node. The following options are available:

- **Label:** Restrict the set of segments/nodes to step through by a label group. Within a label group select the *Not assigned* item to step through seg-

ments/nodes that are not assigned to a label of this label group. Select the *All* item to disable restriction to a label.

- **Node:** Step through all nodes of the current set of nodes. Next to the *Node* spin box the total number of nodes within the current set of nodes is displayed.
- **Segment:** Step through all segments of the current set of segments. Next to the *Segment* spin box the total number of segments within the current set of segments is displayed.
- **Point:** Step through all points of the currently selected segment. Next to the *Points* spin box the total number of points of the currently selected segment is displayed.
- **Focus selection at picked point in 3D viewer:** When checked, the element selected by the step tool (node or edge) is centered in the 3D viewer. The camera distance from the object can be adjusted with the **Focal distance** option. The distance is given in units.
- **Hide surrounding elements:** When checked, only elements close to the selected element are displayed. Elements to be displayed are selected by their vertices: For each vertex within a certain radius of a reference point the whole connected component attached to it is shown. Everything else is hidden in the 3D viewer. The reference point is the coordinate of the selected node if the node stepper is active. If the edge stepper is active, the middle between the edges end nodes is used as reference. The maximum distance for objects to be displayed can be adjusted with the **Max vertex distance** option. It is given in units.

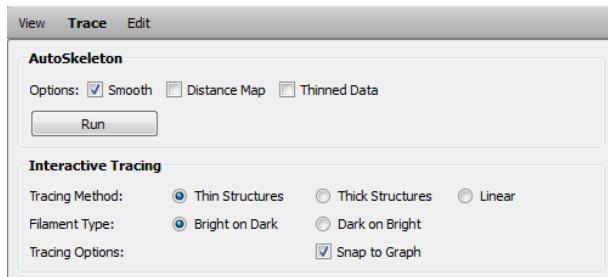
### The **Trace** tab

The *Trace* tab provides a set of tools to perform an automatic or interactive tracing on either the gray value image stack or a label field that was segmented previously using other methods. The tool always works on the data set currently selected in the *Image Data* drop-down menu, either image data or a label field.

#### *AutoSkeleton:*

The *Run* button triggers the computation of the graph. The result of the computation is a new SpatialGraph data object that will be automatically selected in the *Graph Data* drop-down menu. The following options are available:

- **Smooth:** If checked, the result will have reduced staircase artifacts.
- **Distance Map:** If checked, an intermediate *distance map* object will be exposed in the *Pool*.



**Figure 25.3:** The Trace tab.

- **Thinned Data:** If checked, the thinned data object (see *thinning* algorithm) will be exposed in the *Pool*.

#### *Interactive tracing:*

To activate interactive tracing, click the *Trace filament* icon in the toolbar. Note that the interactive tracing tool is always active while the *Trace filament* icon is highlighted and can be triggered only if the 2D slice viewer is active.

The tool offers two **tracing methods** that differ with respect to the thickness of the traced fibers. The *Thick Structures* tool is optimized for fibers that are several voxels in diameter. It also calculates on the fly a thickness estimate of the traced fiber and stores it in the *thickness* point attribute of the graph data. The *Thin Structures* tool can also handle very thin fibers. Note that the algorithm of the *thin* tracer is designed for fine and unbranched fibers; in case of thick branching sections the fiber diameter can be incorrectly estimated. The tool also offers a third option, *Linear*, which connects points by straight lines and does not estimate thickness. Attribute *thickness* is set to 1 for those segments. The *Linear* mode can be useful for low quality data for which the other modes fail.

Once the tracing tool is activated, the cursor of the 2D slice viewer will indicate if the current voxel's gray value is inside the data range specified by the *Window Level* control. You may limit this specified window even further by manually setting a range in the *Intensity range* control. Whenever a simple cross (outside cursor) appears, the voxel is outside the range. A circle with cross hair (inside cursor) indicates voxels inside the range.

Interactive tracing starts by clicking in the 2D slice viewer onto a foreground pixel which adds a node at the selected position. Once a node has been added, the

tracer is in *append mode*, which is indicated by the cursor turning red. The next click onto a foreground pixel adds a second node and starts the automatic tracing algorithm by calculating the shortest line connecting these two nodes within the current data range. Note that after the automatic tracing has finished, the tracer is still in *append mode*, so that the next click onto an inside voxel will append another point connected to the previous point, and so on. To terminate a tracing sequence, click onto the last added node or onto a pixel which is outside the specified range. The cursor color turns white again.

- **Tracing Methods:** The radio buttons allow you to select the *Thick Structure* or *Thin Structure* tracing algorithm, or to connect two points by a straight line (see below for a detailed explanation).

- **Filament Type:**

- **Bright on Dark** Foreground voxels have a higher intensity than background pixels (default).
- **Dark on Bright** Foreground voxels have a lower intensity than background pixels.

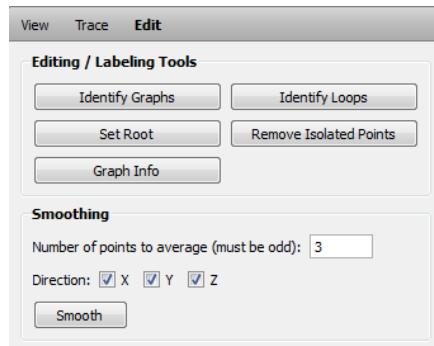
Note that toggling the *Filament Type* also affects the rendering in the 2D and 3D viewers. For dark filaments on a bright background the 2D viewers switches from maximum intensity projection (MIP) to minimum intensity projection (MinIP) and for volume rendering the transfer function (colormap) is inverted.

- **Tracing Options:** If *Snap to Graph* has been checked, picking a voxel while either P (point) or E (edge, the line between two points) is visible in the mouse cursor, the graph is branched by inserting a node into the graph at the picked position or, if N (node) is in the cursor, the graph is extended by using the node next to the cursor position.

## The *Edit* tab

### *Editing and Labeling Tools:*

- **Identify Graphs:** This button identifies graphs, that is all sets of connected segments. The results are stored as a label group *Identified\_graphs* in the *SpatialGraph* object. Use the Label Editor and the graph display options (*View tab*, *3D Settings*) to visualize identified graphs.
- **Identify Loops:** When clicking this button the graph is traversed to detect so called *loops*, i.e., all nodes that are connected to a particular root node



**Figure 25.4:** The Edit tab.

by more than one set of segments. This tool is useful when tracing neurons, where the dendritic arborizations must follow a strict tree-like topology. The result of the computation is stored as a label group *Identified\_Loops* in the *SpatialGraph* object. Use the Label Editor and the graph display options (View tab, 3D Settings) to visualize identified loops. Note that a segment is defined by two nodes. Therefore, when you remove a node, the connected segment will be also removed. If you delete a loop including its nodes, it might be possible that some connected segments will be also deleted. Hence the *Identified\_Loops* contain the looping segments, but not the nodes connecting them. However this may produce isolated points which can be removed by the *Remove Isolated Points* button.

- **Set Root:** Use this button to automatically determine the tree topology of a graph. To do so, the tool needs a root segment which can be set by selecting a segment and clicking this button. At the same time all other segments will be automatically labeled with a rank. With *rank* we denote the branching level with respect to the root. For example, beginning with the root segment (rank 0), every branching point increments the rank of the subsequent segments by 1 level. The results are stored as a label group *Rank* in the *SpatialGraph* object. Use the Label Editor and the graph display options to visualize the rank. Note that calculating the topology and thus determining the ranks will be aborted when loops are detected within the graph. If this happens, use the *Identify Loops* button to tag all loops and then use the graph

editing tools to resolve them.

- **Remove Isolated Nodes:** Press this button to remove nodes that are not connected to other nodes.
- **Graph Info:** Pressing this button triggers the computation of basic statistical information of the current graph. If parts of a graph are selected, the information is calculated for the current selection. If there is no selection, this is done for the whole graph. The information is displayed in a separate spreadsheet window (see *SpatialGraphStatistics*).

#### ***Smooth segments:***

To smooth the currently selected *SpatialGraph* press the *Smooth selection* button. The following options are available:

- **Number of points to average:** Number of neighboring points that should be averaged for each point.
- **Direction:** Restrict smoothing onto certain directions, e.g., if only *z* direction checkbox is checked, smoothing will only be applied in *z* direction of the *SpatialGraph*.

### **25.1.4 The Viewers**

The Filament Editor has three viewers:

- **2D slice viewer:** This viewer is used as the primary working area of the interactive tracing tool and displays the image data either as a conventional thin slice, i.e., data are shown from a definite plane within the volume, or as a thick slice, i.e., a slab with user-defined thickness. With thick slicing a maximum intensity projection of the gray values within the slab is displayed. The thickness of the slab can be set with the slider in the control panel. The data window can be set with the *Window level* tool. In addition, the tracing results within the slab are displayed on top of the gray data as green lines (segments) and blue points (nodes). Using the *Trackball* tool, the (thick) slice may take an arbitrary plane and using the *Browse slices* tool the position of the slice is alterable.
- **3D viewer:** This viewer is used to display the graph together with an optional 3D representation of the thick slice currently seen in the 2D slice viewer. The thick slice display is toggled by the button next to the *Thickness* slider. Together with the thick slice the bounding box and a yellow frame

bounding the area displayed in the 2D viewer is rendered. The 3D viewer also serves to edit the graph directly by selecting nodes and segments.

- **2D segment viewer:** This viewer also displays the image data either as a thin or thick slice. In contrast to the 2D slice viewer, the slice is located along a *selected* segment and only this segment (as green line) and its corresponding nodes (as blue points) are displayed. If no segment or more than one are selected nothing is shown. However, in the case a segment was selected and the new selection still contains parts of it, like nodes or points, the viewer keeps showing this segment and the corresponding slice. Note that modifications of the thickness or the data window affect both the 2D segment viewer and the 2D slice viewer.

## Viewer layout

The Filament Editor starts with the 2D slice viewer and the 3D viewer in a side-by-side layout. If no image data is loaded, the default layout is a single viewer. By default, the 2D segment viewer is hidden but the visibility and also the layout of the other viewers can be configured in the viewer toolbar. The 2D segment viewer is always located below the 2D slice and 3D viewer.



*Single viewer:* Show either the 2D slice viewer or the 3D viewer. If this button is pressed repeatedly the currently visible viewer is swapped with the other one



*Two viewer side-by-side or stacked:* Show the 2D slice viewer and the 3D viewer side-by-side or stacked



*Toggle viewer:* Show or hide the 2D segment viewer

## Viewer navigation

There are two basically different navigation modes: *Edit* and *Camera Navigation* mode. The two modes can be toggled using the ESC key. If no *SpatialGraph* object is present, *Edit* mode is not available and the icons in the viewer toolbar are not selectable. In the 2D slice viewer the *Edit* mode has three tools: *Select single*, *Move selected node* and the interactive tracing tool. In the 3D viewer default edit tool is *Select single*. Activating another selection tool (e.g., *Lasso tool*,

*Select connected*) makes this tool the standard selection tool. In *Camera Navigation* mode *Trackball*, *Translate* and *Zoom* have the corresponding effect in the two viewers. Also, like in the regular 3D viewer, the camera navigation tools are accessible through middle (translate) and middle+left mouse button (zoom). In the 2D segment viewer selection as well as editing are not provided and navigation is only possible by use of the *Translate* or the *Zoom* tool.

### **Navigation Tools in 2D Slice Viewer**

-  *Trackball*: Rotate the thick slice around selected point or center of slice, if no point is selected
-  *Translate*: Displace the slice laterally
-  *Zoom*: Zoom in/out
-  *Home*: Change viewer perspective to home (default perspective)
-  *Set Home*: Sets a new home perspective
-  *Window level*: Adjust data window (center/width). Drag the mouse vertically to move the center of the data window, drag it horizontally to change the width of the data window
-  *Browse slices*: Browse through slices in the 2D slice viewer

### **Navigation Tools in 3D Viewer**

-  *Trackball*: Rotate scene
-  *Translate*: Translate scene
-  *Zoom*: Zoom in/out
-  *Rotate*: Click button to rotate scene around viewer z axis
-  *Seek*: Center and zoom scene to the node or segment which has been clicked
-  *Home*: Change camera position to home (default camera)
-  *Set Home*: Sets camera to the home position



*Orthographic/Perspective:* Toggle between orthographic and perspective camera projection

## Selection Tools Reference



*Select single node, segment, or point:* Selects a single item. Using Ctrl+Shift the selection is extended to the whole connected component (shortcut: e).



*Select a connected component:* Selects a segment and all connected segments and nodes (shortcut: w).



*Select subtree:* Selects a whole sub-tree, or the shortest path to the root if the Shift key has been pressed. Available only if a root segment has been set, otherwise the tool will be disabled (shortcut: r).



*Draw a line to select nodes, segments, and points:* Lasso tool. Draw a free-hand contour in the viewer to select items (shortcut: q). Only available in 3D viewer



*Calculate the shortest path between a pair of nodes or segments:* Calculates the shortest path between an arbitrary pair of nodes or segments and returns it as selection (shortcut: f)



*Select all nodes, segments, and points:* Selects all items in the SpatialGraph (shortcut: a).



*Clear selection:* Deselects any items (shortcut: c).

## Edit Tools Reference



*Trace filament:* Activates the interactive tracing tools (shortcut: T).



*Move selected node:* Activates the interactive moving of points including re-tracing of connected edges. In order to move a node, you need to first select it, then click the *Move selected node* tool and finally click onto the position on the slice where you want to move the point to (only in 2D slice viewer)

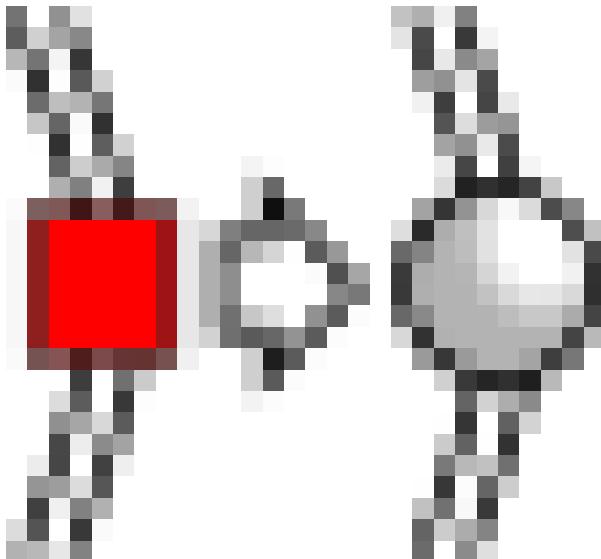


*Connect selected nodes, segments, or points:* Connects selected nodes, segments, or points (shortcut: S). In particular, when 2 nodes are selected, they are connected. When a node and a point are selected, it converts the point to a node and connects them. Otherwise it connects the selection in the following way:

1. for each selected element, find the connected component of which it is part,
2. determine all ending nodes of all those connected components,
3. find the pair of ending nodes in the other connected components that are closest and connect them,
4. update the connected components and start over from 2) until only one connected component is left.



*Remove intermediate nodes, leaving only branching and ending nodes:* Joins adjacent segments, if they are separated by a node with two incident segments, i.e., are neither branching nor ending nodes (shortcut: I).



*Convert a selected*

*point into a node:* Splits a segment at the selected point. This is useful to remove part of a segment: after the segment has been split at the selected point, the part to be removed can be selected and deleted. The selected point must not be the first or last point on a segment (shortcut: . (period)).



*Delete selected nodes, segments, and points:* Deletes selected nodes, segments, and points (shortcut: d). Selected nodes will be deleted only if all adjacent segments are selected as well



*Undo:* Undoes the last edit operation (shortcut: Ctrl+z)



*Redo:* Redoes the last edit operation (shortcut: Ctrl+y)

## Modifier Reference for Edit Tools

The behavior of the three main selection tools *Lasso*, *Select single*, and *Select connected* may be modified using one of the following *modifier keys* **Shift**, **Ctrl**, **Alt**, **Shift+Ctrl**, or **Alt+Ctrl**, as described below.

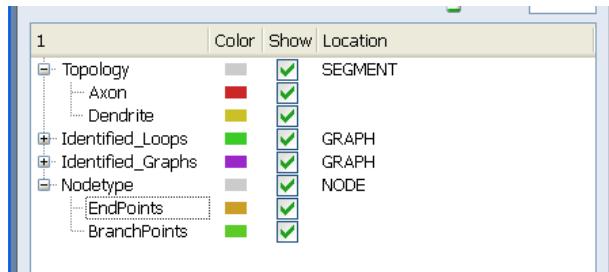
	Lasso tool	Select single node/segment	Select connected component
No modifier	Select all vertices/edges in contour. Deselect everything else.	Select clicked vertex, deselect everything else.	Select connected component while deselecting everything else.
Shift	Select all connected components that are completely inside contour. Deselect everything else.	Select clicked connected component while deselecting everything else.	Select clicked vertex while deselecting everything else.
Ctrl	Add all vertices/edges inside contour to selection.	Toggles selection of clicked vertex/edge/point while retaining the rest of selection.	Toggles selection of clicked connected component while retaining the rest of selection.
Alt	N/A	N/A	N/A
Shift+Ctrl	Add all connected components that are completely inside contour to selection.	Toggles selection of clicked connected component while retaining rest of selection.	Toggles selection of clicked vertex/edge/point while retaining the rest of selection.
Shift+Alt	N/A	N/A	N/A

### 25.1.5 Working with the Label Editor

The *Label Editor* is a window inside the Filament Editor's control panel allowing the user to edit the graph according to user-defined labels. Labels can be created to annotate the graph or parts of it to reflect, for example, anatomical or functional properties of the underlying biological object. Once a graph has been labeled, the Label Editor can be used to flexibly select, show, and hide parts of a graph.

- **Creating a label group**

To annotate a graph, create a label group by right-clicking into the label area and selecting the *Add node label group*, *Add segment label group*, or *Add graph label group* entry of the context menu. In the sub-menu you now need to select *Empty label group*. Segment labels are defined on segments of the graph, Node labels are defined on the nodes and graph level and Graph



**Figure 25.5:** The Label Editor Window.

labels are defined on both segments and nodes. Besides the different semantics, this distinction has implications for how the graph is displayed with e.g., *SpatialGraphView*.

- **Adding labels to a label group**

To add a label, right-click onto the root element of the label group in the tree window and select *Add label* from the context menu.

- **Importing label trees from other spatial graphs**

To import a label tree from an existing spatial graph, you need to right-click into the label area. The first step is then to decide whether you want to add the labels for nodes, segments or both. As result, a new sub-menu opens and you can choose a label tree from any of the appearing spatial graphs. For each spatial graph, all label trees are listed. Select the one you want. Note that the spatial graph you want to import the label tree from needs to reside in the *Pool*.

- **Renaming a label or a label group**

Right-click on the item to be renamed, select the *Rename [label | label group]* and begin to type where the cursor blinks.

- **Assigning nodes, segments, or both to a label group**

Select nodes, segments and graphs using the tools from the viewer toolbar or the *Label Editor* itself. Right-click on the label item and select *Assign selection* from the context menu.

- **Changing the color of a label**

Click on the colored rectangle in the label window to get a *color dialog*. Edit the desired color and click *OK*.

- **Changing the order of label groups and labels**

The order and hierarchy level of labels or label groups can be changed by clicking an item (label or label group) and dragging it into another position within the level of hierarchy or into another level. However, it is not possible to drag one label group into another one nor is it possible to drag a label from one label group into a different one.

- **Highlighting nodes and segments**

To highlight nodes and segments according to labels click the corresponding item in the *Label Editor* window. To select individual nodes/segments expand the label group item by clicking the [+] and selecting the label item to be highlighted. If a *Segment* label group has been selected, this highlights segments only, if a *Node* label group has been selected, only nodes are highlighted and if a *Graph* label group has been selected, nodes *and* segments are highlighted.

- **Showing and hiding nodes and segments**

Label groups and member labels can be hidden by unsetting the checkmark in the *Show* column of the label window. Unsetting the checkmark of a label group item hides all segment/nodes associated with that label group.

The fact that labels of different label groups are not necessarily independent, checking and unchecking a label of one label group may influence the visibility of other label groups labels as well. Thus tri-state checkboxes are used, introducing a *grayed out* third checkmark state indicating that not all segments and nodes associated with this label or label group are visible.

To hide individual label items of a label group, expand the label group and unset the checkmark of that label.

# Index

- Amira Script, 575
- 2DMesh, 3
- Abaqus format, 638
- Access LargeDiskData, 4
- ACR-NEMA, 647
- Advanced Colormap Editor, 747
- AffineRegistration, 6
- AlignMolecules, 301
- AlignPrincipalAxes, 10
- AlignSequences, 303
- AlignSlices, 11
- AlignSurfaces, 387
- AMBER, 617
- AMF, 618
- Amira Script Object, 575
- AmiraMesh as LargeDiskData, 590
- AmiraMesh Format, 575
- AnalyticTensorField, 741
- Analyze 7.5, 590
- AnalyzeAVW, 591
- Animate, 29
- AnnaScalarField3, 657
- AnnaVectorField3, 658
- Annotation, 30
- AnonymizeImageStack, 32
- ApplyTransform, 32
- ArbitraryCut, 35
- Arithmetic, 38
- ArithmeticRendering, 365
- AtomicMolecularDensity, 306
- AverageVolumes, 43
- AVS Field, 637
- AVS UCD Format, 637
- Axis, 45
- B-Splines, 661
- background correction, 493
- BeadExtract, 489
- Bio-Rad Confocal Format, 643
- BlockFaceCorrection, 46
- BlockStructuredGrid3, 661
- BlockStructuredScalarField3, 661
- BlockStructuredVectorField3, 662
- BMP Image Format, 591
- BondAngleView, 310
- BondCalculation, 312
- border width, 495
- BoundaryConditions, 389
- bounding box, 196, 613, 781
- BoundingBox, 47
- CalculusMatlab, 48
- CameraPath, 56
- CameraPath Editor, 754
- CameraRotate, 662
- CannyEdgeDetector, 57
- CastField, 58

- CastLattice, 365  
CCD detector, 493  
ChannelWorks, 60  
CityPlot, 61  
ClippingPlane, 64  
Cluster, 663  
ClusterDiff, 64  
ClusterFilter, 66  
ClusterGrep, 67  
ClusterSample, 68  
ClusterStringLabels, 69  
ClusterView, 70  
CollectiveTCL, 73  
Color Dialog, 757  
ColorCombine, 74  
ColorField3, 665  
Colormap, 666  
Colormap Editor, 759  
Colorwash, 76  
CombineLandmarks, 79  
CompareLatticeData, 80  
CompassPosition, 80  
CompMolInterface, 313  
CompMolSurface, 314  
ComponentField, 390  
ComputeContours, 82  
ComputeDistanceToPoint, 83  
ComputeHBonds, 318  
ComputePerfusion, 511  
ComputeSecondaryStructure, 319  
ComputeTensor, 515  
ComputeTensorOutOfCore, 517  
ConePlot, 391  
ConfigurationDensity, 320  
ConnectedComponents, 84  
Connection, 536  
ContourView, 394  
ContrastControl, 86  
ConvertTalairach, 519  
ConvertToLargeDiskData, 367  
Convolution, 491  
CornerCut, 90  
CorrectZDrop, 492  
CorrelationPlot, 91  
CreateCluster, 96  
CreateGradientImage, 520  
CreateSphere, 96  
Curl, 97  
Curve Editor, 764  
CurvedSlice, 98  
Cutting Plane, 101  
CylinderSlice, 102  
Data, 669  
DataPreprocess, 493  
DataProbe, 104  
Deconvolution, 494  
deconvolution, 494  
Delaunay2D, 110  
Demo GUI, 765  
DemoDirector, 112  
DemoMaker, 126  
DemoSequence, 139  
DICOM export, 647  
DICOM import, 648  
DicomSend, 529  
Digital Image Filters, 142, 769  
Displace, 395  
DisplayColormap, 142  
DisplayDate, 144  
DisplayISL, 396  
DisplayLogos, 146  
DisplayTime, 147  
DistanceMap, 497  
Divergence, 400  
DoseVolume (Tetrahedra), 401

- 
- DuplicateNodes, 402  
DX, 621  
DXF, 592  
  
EigenvectorToColor, 522  
Encapsulated PostScript, 592  
ExtractEigenvalues, 403  
ExtractSurface, 149  
  
FEIStackedScalarField3, 643  
FEIUniformScalarField3, 643  
FiberTracking, 523  
FIDAP NEUTRAL, 638  
Field3, 670  
FieldCut, 403  
FieldCut (Polyhedra), 406  
Filament Editor, 867  
FilteredObliqueSlice, 150  
flatfield correction, 493  
Fluent / UNS, 639  
FourierTransform, 498  
  
GetCurvature, 152  
Gradient, 408  
Grid Editor, 861  
GridBoundary, 409  
GridCut, 411  
GridCut (Polyhedra), 413  
GridToSurface, 414  
GridView, 154  
GridView (Block Structured), 415  
GridVolume (Hexahedra), 416  
GridVolume (Polyhedra), 418  
GridVolume (Tetrahedra), 420  
GROMACS, 622  
Grouping, 155  
  
HBondView, 323  
HeightField, 157  
  
HexaGrid, 671  
HexaQuality, 423  
HexaScalarField3, 672  
HexToTet, 422  
Histogram, 159  
Hoc, 593  
HTML, 592  
HxSurface, 595  
Hypermesh, 639  
  
Icol, 601  
IDEAS universal format, 639  
IlluminatedLines, 162  
ImageCrop Editor, 778  
initial estimate, 495  
Interfile, 601  
Interpolate, 425  
InterpolateLabels, 163  
Intersect, 164  
Isolines, 165  
Isolines (Surface), 168  
Isosurface (Hexahedra), 427  
Isosurface (Polyhedra), 429  
Isosurface (Regular), 170  
Isosurface (Tetrahedra), 431  
IsosurfaceRendering (LDA), 368  
IVData, 672  
IVDisplay, 172  
IVToSurface, 173  
  
job dialog, 497  
JPEG Image Format, 601  
  
LabelSpatialGraph, 174  
LabelVoxel, 175  
Lambda2, 433  
Landmark Editor, 782  
landmark set, 489

- LandmarkSet, 672  
LandmarkSurfaceWarp, 177  
LandmarkView, 177  
LandmarkWarp, 179  
LargeDiskData, 635, 675  
Lattice3, 675  
LatticeAccess, 181  
LatToHex, 433  
LDA, 635  
LDAExpertSettings, 369  
LegoSurfaceGen, 183  
Leica 3D TIFF, 643  
Leica Binary Format (.lei), 644  
Leica Image Format (.lif), 644  
Leica Slice Series (.info), 644  
Light, 677  
LineProbe, 183  
LineSet, 680  
LineSet Editor, 783  
LineSetProbe, 183  
LineSetToSpatialGraph, 185  
LineSetView, 185  
LineStreaks, 434
- MagAndPhase, 435  
Magnitude, 435  
MAP, 623  
masking, 41  
MasterConnection, 537  
MaterialStatistics, 189  
MATLAB Binary Format (.mat),  
    602  
MATLAB M-files Format (.m), 602  
maximum-likelihood method, 494  
MDL, 623  
MeanMolecule, 325  
Measurement, 192, 327  
Merge, 195
- Metamorph STK Format, 645  
microsphere, 489  
Molecule, 723  
Molecule Attribute Editor, 842  
Molecule Data Editor, 849  
Molecule Editor, 850  
MoleculeLabel, 336  
MoleculeView, 339  
MolElectrostatics, 329  
MolOptimizer, 331  
MolSurface, 715  
MolSurfaceView, 332  
MolTrajBundleConverter, 335  
MolTrajectory, 716  
MolTrajectoryBundle, 720  
MolTrajectoryBundle Editor, 841  
Mosaic, 743  
Movie, 682  
MovieMaker, 197  
MoviePlayer, 201  
MRC, 644  
MultiChannelField3, 685  
Multiplanar Viewer, 784
- Nifti, 602  
noise, 491  
NoncovalentInteractionGrid, 342  
NormalizeImage, 205  
numerical aperture, 495
- Object, 686  
ObliqueSlice, 206  
ObliqueSlice (LDA), 373  
Observables, 343  
Olympus (.oib/.oif), 645  
Open Inventor, 603  
OrthoSlice, 210  
OrthoSlice (LDA), 376

- 
- OrthoViewCursor, 213  
overrelaxation, 496
- ParallelMovieMaker, 215  
Parameter Editor, 789  
ParametricSurface, 436  
ParticlePlot, 439  
PbScalarField3, 742  
PDB, 623  
PHI, 626  
PlanarLIC, 442  
Plot 3D Single Structured, 640  
Plot Tool, 790  
PlotSpreadSheet, 218  
Ply Format, 606  
PNG Image Format, 604  
PNM Image Format, 604  
PointProbe, 219  
PointWrap, 219  
PolyhedralGrid, 742  
Port, 533  
Port3DPointList, 538  
PortButtonList, 542  
PortButtonMenu, 544  
PortChannelConfig, 545  
PortColorList, 546  
PortColormap, 547  
PortDoIt, 550  
PortDrawStyle, 551  
PortFilename, 552  
PortFloatSlider, 553  
PortFloatTextN, 556  
PortFontSelection, 558  
PortGeneric, 559  
PortInfo, 562  
PortIntSlider, 562  
PortIntTextN, 563  
PortMultiChannel, 563
- PortMultiMenu, 564  
PortRadioBox, 565  
PortSeparator, 566  
PortSharedColormap, 566  
PortTabBar, 567  
PortText, 568  
PortTime, 568  
PortToggleList, 571  
power spectrum, 498  
PrecomputeAlignment, 345  
ProbeToLineSet, 221  
ProjectionView, 221  
ProjectionViewCursor, 224  
PseudoElectronDensity, 346  
PSF, 489  
PSF/DCD (CHARMM), 626  
PSFGen, 500  
PSI format, 604
- Quantification, 503  
Quantification-Analyse, 508  
Quantification-Threshold, 509
- RankTimeStep, 347  
RateOfStrainTensor, 447  
Raw Data, 606  
Raw Data as LargeDiskData, 608  
RawAsExternalData, 689  
RefineTetras, 225  
refractive index, 495  
RegScalarField3, 689  
Relabel, 227  
RelabelTetras, 228  
RemeshSurface, 447  
Resample, 230  
resampling, 41
- SampleScalarField, 452

- ScalarField3, 689  
Scale, 234  
ScanConvertSurface, 236  
ScriptObject, 689  
SecStructureView, 349  
SeedSurface, 456  
Segmentation Editor, 800  
SegmentBrain, 526  
SelectLines, 237  
SelectRoi, 239  
SelectRoi (LDA), 379  
SeriesControl, 240  
SGI-RGB Image Format, 609  
Shear, 240  
ShortestEdgeDistance, 241  
ShowConfig, 357  
ShowViewerSpreadSheet, 242  
Simplification Editor, 820  
SmoothSurface, 243  
Sound, 244  
SpatialData, 694  
SpatialGraph, 698  
SpatialGraphStatistics, 245  
SpatialGraphToLineSet, 246  
SpatialGraphView, 246  
Splats, 458  
SplineProbe, 250  
SplitVolume, 250  
SpreadSheet, 698  
SQLite, 233  
stacked coordinates, 41  
Stacked-Slices, 610  
Stacked-Slices as LargeDiskData,  
    635  
StackedLabelField3, 698  
StackedScalarField3, 700  
StandardView, 251  
STL, 609  
StreamRibbons, 460  
StreamSurface, 462  
Surface, 700  
Surface Editor, 823  
Surface Path Editor, 832  
Surface Path Set, 703  
SurfaceArea, 253  
SurfaceCut, 254  
SurfaceDistance, 257  
SurfaceField, 259  
SurfaceGen, 259  
SurfaceIntersector, 262  
SurfaceLIC, 464  
SurfacePathView, 262  
SurfaceThickness, 264  
SurfaceView, 264  
SWC, 609  
Tecplot, 613  
TensorDisplay, 467  
TetraCombine, 472  
TetraGen, 473  
TetraGrid, 703  
TetraQuality, 476  
TetraScalarField3, 704  
TetraVectors, 478  
TetToHex, 472  
TIFF Image Format, 611  
Time, 704  
TimeSeriesControl, 268  
tracking, 360  
Trajectory, 270  
Transform Editor, 837  
TransformAnimation, 272  
TriangleDistortion, 273  
TriangleQuality, 480  
TridelityView, 274  
Tripos, 626

TubeView, 355  
UniChem, 627  
UniformLabelField3, 705  
UniformScalarField3, 707  
  
vector theory, 500  
VectorProbe, 481  
Vectors, 483  
Vectors/Normals (Surface), 485  
Vertex Morph, 280  
VertexDiff, 281  
VertexSet, 707  
VertexShift, 281  
VertexView, 282  
Vevo Mode Raw Images, 614  
ViewBase, 708  
ViewerPlot, 286  
Volren, 288  
Voltex, 292  
Voltex (LDA), 382  
VolumeDataObject, 739  
VolumeEdit, 296  
voxel size, 613, 781  
VoxelView, 298  
VRML, 614  
VRML-Export, 277  
VRSettings, 359  
  
Wavefront Technologies 3D Geome-  
try (.obj), 615  
wavelength, 495  
    emission, 500  
    excitation, 500  
  
Zeiss LSM, 645  
ZIB Molecular File Format, 627

