Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

DETLEV STALLING, MARTIN SEEBASS, MALTE ZÖCKLER,
HANS-CHRISTIAN HEGE

Hyperthermia Treatment Planning with HyperPlan - User's Manual

ZIB-Report 00-27 (October 2000)

Hyperthermia Treatment Planning with HyperPlan - User's Manual

Detlev Stalling, Martin Seebass, Malte Zöckler, Hans-Christian Hege

Abstract

HyperPlan is a software system for performing 3D-simulations and treatment planning in regional hyperthermia. It allows the user to understand the complex effects of electromagnetic wave propagation and heat transport inside a patient's body. Optimized power amplitudes and phase settings can be calculated for the BSD radiowave applicators Sigma 60 and Sigma 2000 (eye-applicator). HyperPlan is built on top of the modular, object-oriented visualization system Amira. This system already contains powerful algorithms for image processing, geometric modelling and 3D graphics display. HyperPlan provides a number of hyperthermia-specific modules, allowing the user to create 3D tetrahedral patient models suitable for treatment planning. In addition, all numerical simulation modules required for hyperthermia simulation are part of HyperPlan. This guide provides a step-by-step introduction to hyperthermia planning using HyperPlan. It also describes the usage of the underlying visualization system Amira.

Contents

Part I

HyperPlan Documentation

Chapter 1

Hyperthermia Planning

HyperPlan is a software system for performing 3D-simulations and treatment planning in regional hyperthermia. It allows the user to understand the complex effects of electromagnetic wave propagation and heat transport inside a patient's body. Optimized power amplitudes and phase settings can be calculated for the BSD radiowave applicators Sigma 40, Sigma 60 and Sigma 2000 (the so-called eye-applicator). The planning system as well as the underlying numerical algorithms have been developed at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) in cooperation with the Rudolf-Virchow Klinikum, Berlin. The work has been supported by Dr. Sennewald Medizintechnik GmbH, Munich.

HyperPlan is built on top of the modular, object-oriented visualization system Amira. This system already contains powerful algorithms for image processing, geometric modelling and 3D graphics display. HyperPlan provides a number of hyperthermia-specific modules, allowing the user to create 3D tetrahedral patient models suitable for treatment planning. In addition, all numerical simulation modules required for hyperthermia simulation are part of HyperPlan.

The first part of this guide provides a step-by-step introduction to hyperthermia planning using HyperPlan. The underlying software system Amira is described in the second part. Before reading the following sections it is a good idea to become more familiar with *Amira*. At least, one of the introductory *Amira tutorials* should have been worked through.

An overview of the planning process is given in section *Overview*. Technical prerequistes like file name conventions used throughout the guide are described in section *Filename Conventions*. The following sections are devoted to *segmentation*, *grid generation*, *E-field simulation*, *temperature simulation*, and *optimization issues*.

1.1 Overview

Before discussing the technical aspects of HyperPlan, let us first discuss some basic principles of hyperthermia simulation. In regional hyperthermia two different physical processes have to be considered. The first one is electromagnetic wave propagation inside the patient's body. Simulating this process means to solve Maxwell's equations numerically. For every point inside the patient we need to know the dielectric constant and the electric conductivity. Given these parameters, it is possible to obtain an approximate E-field pattern as well as a corresponding power absorption, also called SAR (Specific Absorption Rate). However, from the therapeutic point of view the power deposition is less important than the resulting temperature increase. To compute temperature increase, the transport of heat inside the body has to be simulated as well. Unfortunately, there is no equivalent to Maxwell's equations which describes the phenomenon exactly. Instead, in HyperPlan a simplified model is used, the bio-heat transfer equation. This equation does not take into account individual blood vessels. Instead, it assumes that heat exchange between tissue and blood occurs only in the capillaries. Thus, the thermal effect of blood flow is described by an isotropic term. Relevant parameters are the blood perfusion as well as the thermal conductivity.

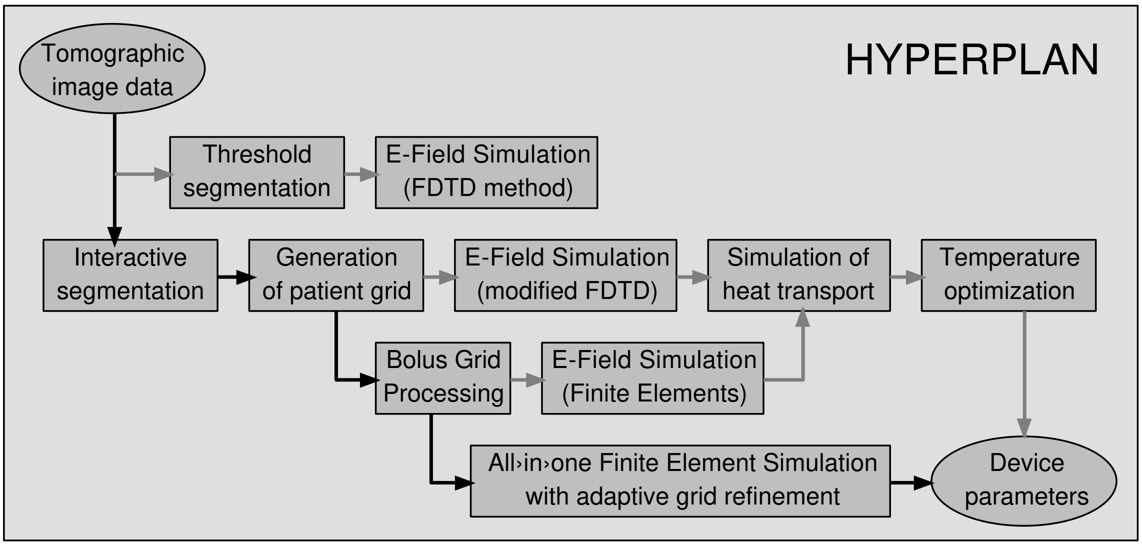


Figure 1.1: Work flow in hyperthermia simulation. The system reads tomographic image data of a patient and outputs optimized amplitudes and phase settings for the hyperthermia applicator. Several different simulation methods are provided. All data can be analysed using powerful visualization methods.

Figure 1.1 illustrates the different algorithmic steps of hyperthermia simulation as implemented in HyperPlan. First of all, a stack of computer tomographic images of a patient is loaded into the system. In these images regions of different tissue type as well as the tumor have to be identified. This task is called image segmentation. For each region specific values of the electric and thermal parameters are assumed. The next step is to solve Maxwell's equations and to compute an electromagnetic field distribution. In HyperPlan all channels of the radiowave applicator are simulated separately. This makes it possible to obtain the resulting field for any set of amplitudes and phase shifts by simple linear superposition. HyperPlan provides an FDTD method which directly works on a segmented image volume or so-called label field. This method can be used to investigate the resulting SAR distribution without generating a geometric patient model. However, the use of a voxel grid imposes some artificial rasterization effects. To obtain more accurate results the interfaces between different tissue compartments have to be represented more precisely. In HyperPlan this is achieved by using a three-dimensional tetrahedral patient model. Based on such a tetrahedral model an E-field simulation can be performed using a modified FDTD method or using a high-precision finite element method. When the E-fields have been computed, a resulting temperature distribution can be simulated. For this a fast multi-level finite-element method is used. Similar to the E-fields, a set of independent temperature fields is computed. This makes it possible to change amplitudes and phases without rerunning the whole simulation. Finally, an optimization module is provided which computes individually optimized device parameters. The optimization procedure tries to maximize the heating of the tumor while not exceeding user-defined maximal temperatures in the other tissue compartments.

HyperPlan also provides an all-in-one simulation module which performs E-field simulation, temperature simulation, and optimization at once. This has the advantage that adaptive grid refinement takes place at locations required by the optimized temperature distribution. Using this approach is the recommended way of performing hyperthermia simulations because it is the most accurate one. In Figure 1.1 the corrsponding work flow is marked by solid black lines while alternative paths are shown in light gray.

1.2 Filename Conventions

Some of the planning steps described in the following require a number of sub-steps. We recommend to group all planning data for a single patient into a separate directory. Furthermore, we recommend to compose filenames used to store intermediate planning results as follows:

name.filetype.version-number

The meaning of the different parts of the name is described below:

- *name* might be the patient's name or a short form of it. Make the *name* part of the filename a unique identification of the patient whose treatment is simulated.
- *filetype* describes the data type. Usually, when data sets are created, HyperPlan already suggests a suitable value for this part of the name. For example, if you perform a threshold segmentation on a data set called *Exam.ct-data* the resulting label field storing the segmentation results will be called *Exam.LabelField*. In particular, the following type names should be used (details are given in subsequent chapters):

`ct-data`	for the patient's CT data
`LabelField`	for segmentation results
`surf`	for triangular surface models
`grid`	for tetrahedral patient models
`EFieldVals`	for the result of an E-Field calculation
`ThermVals`	for the result of a temperature calculation
`Plan`	for the result of an optimization of treatment parameters

- *version-number* allows you to distinguish between different simulations for the same patient. You might for example perform several E-field calculations with different patient positions within the hyperthermia applicator, or several temperature calculations assuming different thermal tissue properties. HyperPlan takes the version-number into account by propagating it along proposed filenames for result data of successive planning steps.

1.3 Segmentation

In this section we describe how to import CT images in HyperPlan and how to segment them, i.e., how to separate different tissue types. Segmentation is performed in two steps. First, an automatic thresholding operation is applied. Then the results of thresholding are refined interactively. Both steps are described in separate sections.

1.3.1 Importing CT-Data

Before you can perform any kind of planning for a new patient you first have to load a stack of the patient's CT images into HyperPlan. Use the file dialog for loading the data, i.e., select *Load...* from the *File* menu of HyperPlan's main window.

At the hospital, when the CT scans are taken, make sure that all images have the same pixel size. Although not strictly required, it is also advantageous to choose equal slice distances throughout the whole image stack. If you are going to use one of the FDTD-based simulation methods equal slice distances are obligatory. In case of the Sigma 60 or Sigma 2000 eye applicator the area covered by the image stack should have an extension of at least 50 (better 60) cm in the direction of the patient's axis. The tumor should be centered somewhere in the middle of the image volume. We recommend to use a slice distance of 1 cm or smaller.

The ACR-NEMA file format and its successor the DICOM format are widely used to store medical images of various types. In HyperPlan the import of stacks of axis-aligned CT or MR images stored in these formats is supported. The read routine checks whether the spacing between subsequent slices is equal or not. In the first case an image volume with uniform coordinates is created, otherwise stacked coordinates are used. To check which coordinate type is used select a data set by clicking on the corresponding green data icon and look at the text line labeled *Info*. In order to create a uniform data set from a stacked one you may use the *arithmetic module*. Choose the *regular* option of the arithmetic module and adjust the resolution of the resulting uniform field as required. Afterwards you will have to convert to data values of the resulting field back to 8-bit or 16-bit integers by using the *cast field module*.

In ACR-NEMA or DICOM usually each slice is stored in a separate file, but HyperPlan requires the whole image volume to be loaded at once, so you have to select multiple files in the file dialog. This is done by clicking the first file and then shift-clicking the last one. The *file type filter* can help to specify the selection. You may enter arbitrary wildcard patterns here. Only files matching the pattern will be displayed. HyperPlan assumes that a file is a ACR-NEMA or DICOM file if the file name suffix is `.ima`, `.ani`, `.dcm` or `dc3` or if the file name matches a DICOM unique instance ID, e.g. `1.3.12.2.1107.5.1.2.20395.19980429221554713.4`. Files are automatically uncompressed if they were compressed by either `gzip` (Unix and Windows) or `compress` (Unix only).

Sometimes CT data are stored in *raw format*. Raw data files only contain the image data, but no header information like pixel size or image spacing. If the data files have a suffix `.raw` or if no other format can be determined, the files will be loaded in raw mode. When you click *Ok* in the file dialog, an additional dialog box will appear. In this dialog you can enter the dimensions of your image volume and of the bounding box. The bounding box should be entered in centimeters. If no more information is available like CT table position, you may simply set the lower left front corner of the box to (0,0,0), while the upper right back corner is determined by the physical size of the scanned image volume. Usually CT data are stored in two byte format using 12 bits of gray level information. Therefore the primitive data type should be set to *short*.

Many CT scanners produce images with a resolution of 512 x 512 pixels per slice. In HyperPlan such images should be resampled to 256 x 256 pixels per slice. The smaller resolution is well suited for treatment planning. It results in smaller data files and requires less memory.

To resample your images, select the *Resample* module from the popup menu of the green data icon representing the CT data. The Resample module appears as a red icon. When you click on it, the user interface of the Resample module appears in HyperPlan's work area. In the line labeled *Resolution* you can enter the desired resolution. Type in 256 for both x and y resolution. Click on the *DoIt* button to start resampling. After a few seconds a second green data icon representing the resampled volume should appear in the network area. You can investigate the effect of resampling by attaching an *OrthoSlice* module to the CT-Data (Tip: you can pick the connection line of OrthoSlice and attach it to either of the two CT data sets).

Once you have loaded a CT data set for a new patient into HyperPlan and resampled it to 256 x 256 pixels per slice, it is a good idea to save the resampled image volume to file. To do so, select the resampled volume by clicking on it. Then choose *Save Data As* from the file menu and type in a new name in the file browser. For convenience the file type should be `ct-data`.

1.3.2 Labeling CT-Data

After the CT images of a patient have been read, next different tissue types have to be identified in these images. This task is called *image segmentation*. To simulate the electromagnetic fields inside the patient's body at least fat tissue, muscles, and bones have to be separated. The *LabelVoxel* module provides a simple thresholding algorithm, which can be used to perform this separation. It is contained in the *Compute* submenu of the CT data icon's popup menu.

The LabelVoxel module lets you select at least three different thresholds which define the boundary between exterior and fat, fat and muscle, and muscle and bone, respectively. To verify these boundaries you might have a look at the Hounsfield histogram (press button *Histo*). In the histogram two peaks should be clearly visible, one for fat and one for muscle. You should choose the threshold separating these tissue types in the middle between both peaks.

In some CT images a couch may be visible in the same Hounsfield range as human tissue. Switch on the *remove couch* toggle to prevent that the couch is assigned to either fat, muscle, or bone. If this toggle is set, the biggest connected component in the 3D volume is searched for. This is assumed to represent the patient. All other voxels are set to *air*.

After thresholding the voxels labeled as *air* are separated into two classes: the *exterior* of the patient and air-filled cavities within the patient's body, called *bubbles*. Such cavities occur within the intestine and the stomach; the lung is also classified this way. Note that the separation between *exterior* and *bubbles* doesn't take place if the *bubbles* option of the *LabelVoxel* module is unset. For hyperthermia planning always ensure that this option is set.

Push the *DoIt* button to actually perform the labeling. A new data object of type *HxUniformLabelField3* will be created. It is a regular cubic grid with the same dimensions as the underlying CT data set. For each voxel it contains a label indicating the tissue type. You may visualize the result of the labeling operation by attaching an *OrthoSlice* module to it.

1.3.3 Manual Segmentation

Of course, simple thresholding in general cannot produce satisfactory results. In most cases, additional interactive segmentation is necessary. You may wish to merge segments, split segments, remove islands, etc. HyperPlan includes an interactive editor for 3D image data, called the *image segmentation editor*. This editor lets you display and browse through the 2D slices of the data set. You can activate it by first selecting the green icon representing a *LabelField* object and then pressing the *pencil-shaped edit* button. For a detailed description of the *image segmentation editor*, please refer to the Amira reference manual.

When the editor is activated, HyperPlan checks whether a CT data set with the same dimensions as the LabelField has been loaded. If such a CT data set exists, it is displayed as a background in the 2D viewing window of the segmentation editor. Without a background image segmentation doesn't make much sense. The background image may also be set manually by connecting the *image port* of the LabelField to the corresponding CT data set (click with the right mouse button into the small rectangle left of the LabelField icon and choose *ImageData*).

Before starting the manual segmentation, at least the following two filters should be applied:

- *Remove Islands*. This filter removes small segments by assigning them to their surrounding segments. In order to activate it choose *Remove Islands ...* from the *Label Filter* menu. This causes a small dialog window to appear which allows you to adjust the maximal size (in pixels) of segments to be deleted as well as some percentage value. Islands will be assigned to the tissue type that most of its neighbouring pixels belong to, but will not be changed if the relative incidence of that tissue is below the given percentage. There are three ways of applying the filter: *slice* refers to the 2D version of the filter which is applied to the currently selected slice only, *all slices* means application of the 2D filter to all slices, and *3D volume* is for the 3D filter. We recommend to use the second variant (*all slices*).

- *Fill Holes*. This filter fills holes in the interior of the currently selected tissue type. It should be applied to *Bone* in order to remove marrow regions falsely assigned to muscle or fat. Select *Bone* in the material list, then choose *Fill Holes* and *all slices* from the *Label Filter* menu of the segmentation editor.

Now you are ready to start manual segmentation. For this task you generally proceed as follows: First select an arbitrary area in a slice. Selected pixels are highlighted in red in the 2D window. Then change the labeling for the selected pixels, either by adding them to the currently selected region (button '+' or key 'a') or by subtracting them from this region (button '-' or key 's'). In the latter case, pixels are assigned to an automatically determined background region. There are different ways to perform a selection: simply clicking on a segment, using a lasso tool, a brush or a picker. The functionality of the selection tools is very similar to pixel-oriented painting programs. For a detailed description refer to the Amira reference manual.

You may add some more tissue types to the ones created by *LabelVoxel*. For this, press the right mouse button over the list of tissue types to activate a popup menu. In this menu select the entry *New Material*. You can edit the name of the newly created material by selecting the entry *Rename Material* from its popup menu. If you later want to perform an optimization of treatment parameters, you should at least mark the tumor region and name it *Target*. **Important:** Optimization will not work without a region called *Target* or *Tumor*.

There are a number of predefined tissue types for which HyperPlan's database contains electrical and thermal properties: Fat, Muscle, Bone, Target, Intestine, Abdomen, Bladder, Kidney, Liver, Spleen, Stomach, Pancreas, Heart, Lung, Arteries, Veins. You should preferably use these names. In order to check the contents of the database choose *Database* from the *Edit* menu of the main window. In the dialog which is popped up then open the folder called *Materials*. Each material may contain a blank-separated list of alternative names in a parameter called *name*. In order to extend or modify the database permanently edit the file *HyperPlan/share/materials/database.hm* with an arbitray text editor.

One remark should be made about the names 'Intestine' and 'Abdomen'. If you perform a detailed segmentation of the abdominal region separating fatty and 'real' intestinal regions, you should use the name 'Intestine'. If you perform a less detailed segmentation and mark the whole abdominal region as one tissue type, you should name it 'Abdomen'. The electrical and thermal parameters assigned to 'Abdomen' are mean values of those for fat and muscle tissue.

We strongly recommend to apply the *smooth filter* as a final step before leaving the segmentation editor, preferably the 3D variant. In order to activate the smooth filter choose *Smooth Labels ...* from the *Label Filter* menu of the editor. In the dialog choose *3D volume* and press *Apply*. The smoothing filter may sligthly change the labeling. Besides that it calculates a probability for each voxel to belong to the assigned tissue type. The probability is 1 in the interior of a region, and decreases at the region's boundaries. Subsequent planning steps (applying the generalized marching cubes procedure) can take the probabilities into account in order to create better results.

Manual image segmentation may take a considerable amount of time, especially for an unexperienced user. Therefore it is a good idea to save the segmentation results frequently. For this, choose the *Save Data As* item from the *File* menu. For convenience, we recommend the filetype `LabelField`. Once you have stored the segmentation results for the first time, you can use *Save Data* or simply *Ctrl-S* to save the data again under the same filename. If you have applied the smoothing filter, the LabelField file will not only contain the labels, but also the probabilities for each voxel. You can leave the image segmentation editor by clicking on the pencil icon again.

1.4 Grid Generation

You may skip this section if you want to perform an E-Field simulation on a LabelField using the FDTD voxel method (upper branch in Figure figure 1.1).

A *tetrahedral patient model* is needed for most of the numerical simulation methods included in HyperPlan. To generate such a model from the results of segmentation you have to carry out the three steps described in the sequel.

1.4.1 Extracting Surfaces from Label Fields

In the first step the segmentation results are converted into surfaces separating the different tissue types. The surfaces are composed of *triangular surface patches*, each separating two different compartments. In HyperPlan a method called *generalized marching cubes (GMC)* is used for surface generation. It is a generalization of the *marching cubes* algorithm which is well-known in computer graphics.

Load a *LabelField*, i.e., the result of a segmentation operation, into HyperPlan. If the LabelField has x- and y-dimensions 256 or larger, it is recommended to resample it to a lower resolution, otherwise the number of resulting surface triangles will be very large. Select the *Resample* module from the popup menu of the LabelField. If the Resample module is attached to a LabelField, the different parameter ports are used than in the case of ordinary image data. You can choose integer compression factors for x-, y-, and z-direction. Choose a compression factor yielding a resolution of about 128 in the x- and y-direction, e.g. a factor of 2 to reduce dimensions from 256 to 128. If the slice distance is 0.5 cm or larger, choose a factor of 1 (no compression) for the z-direction. (Hint: You can determine the slice distance by clicking on the green icon representing the LabelField. The voxel size is displayed in HyperPlan's work area. The voxel size in z-direction is the slice distance.) Press the *DoIt* button to perform the actual resampling operation.

Now apply the *Generalized Marching Cubes* algorithm. Select the *GMC* module from the popup menu of the resampled LabelField. Be sure that the toggle *add border* is set. This ensures that the surfaces being created will be closed. If the LabelField contains probability information (see Section 1.3.3) the toggle *sub-voxel accuracy* should be set as well. If the *sub-voxel accuracy* can't be set then you probably have forgotten to call the smooth filter at the end of manual image segmentation. Press the *Triangulate* button of the GMC module to start surface extraction.

Surface extraction using the GMC module will take about 30 seconds. Therefore, it is not really necessary to save the results to a file. In case you want to do this anyway you should use the filetype `gmc-surf`.

1.4.2 Simplifying Surfaces

The surfaces generated by the GMC module typically consist of 200.000 to 400.000 triangles. Tetrahedral grids created from such surfaces are much too large to be suitable in subsequent simulation steps. Thus the number of triangles must be reduced drastically. This step is called *surface simplification*.

Click on the icon representing the surface to be simplified. Then invoke the *Surface Simplification Editor* by pressing of the button displaying the mesh-like icon. The simplification algorithm included in HyperPlan repeatedly collapses surface edges to points, essentially starting from the edge causing the least significant modification of the original surface.

The requested number of triangles can be entered in field *faces*. In addition, in the field *max dist* the maximal edge length (in cm) can be entered. Simplification stops if either the desired number of faces is reached or no edge shorter then *max dist* exists anymore. For hyperthermia simulation we recommend to set the number of triangles to 0 and to specify a maximum edge length of about 2.5cm to 3cm (smaller numbers resulting in more triangles). You may also choose whether to preserve the slice structure of the GMC surface or not by selecting the corresponding option toggle. Preserving the slice structure only refers to the outer boundary of the patient model, i.e., to all patches with outer region equal to *Exterior*. If this option is set, the patient model later can be combined with the hyperthermia applicator more easily (so-called bolus grid processing). However, the option can only be selected if the surface was created from image data with uniform slice distance.

Press button *Simplify* to perform the simplification. This operation will take about 5 minutes depending on the number of triangles that have to be deleted. The progress bar at the lower border of the main window will give you an idea how long you will have to wait.

Button *Flip Edges* allows you to improve the quality of the resulting surface triangulation. HyperPlan searches pairs of triangles with a common edge and large obtuse angles and tries to remove them by flipping edges. In the Console window you get an information about the number of bad triangles and how many of them were removed.

If you want to visualize the simplified surfaces, select module *SurfaceView* from the popup menu of the surface data icon. For a detailed description of this module, refer to the documentation under Section *SurfaceView* in the Amira reference manual.

Save the simplified surface data into a file with filetype `surf`.

1.4.3 Checking Surface Quality

Before starting tetrahedron generation, it is a good idea to check if the surfaces contain *intersecting triangles*. We have integrated strategies to avoid intersections into the simplification algorithm, so this should not occur too often. Click on the green icon representing the surface. Then invoke the *Surface Editor* by clicking on the button displaying the pencil icon. Press the *compute* button with the check option set to *Intersections* in order to perform the intersection test. The number of intersections is shown in HyperPlan's Console window.

If there are intersections, they are displayed in the 3D viewer. You can try to remove them by editing the surfaces manually. There are three possible *editing operations*: moving surface points, flipping edges and collapsing edges. In the Console window you find a short description how to use the mouse buttons and the Shift- and Ctrl-key to perform those operations. In any case the viewer must be in interaction mode, i.e., the mouse cursor must be an arrow and not a hand.

Button 1: select triangle group

Shift-Button 1:	deselect single triangle
Ctrl-Button 1:	deselect triangle group
Button 2:	activate point dragger, highlight edge
Shift-Button 2:	flip edge
Ctrl-Button 2:	collapse edge

Once you have repaired an intersection, press the *Next* button until the next intersection is displayed (Sometimes, a single intersection will be reported multiple times since three or more triangles were involved). When you have repaired all intersections, press button *Intersections* to repeat the intersection test.

There is another test that should be performed before tetrahedron generation, namely the orientation test. Choose *Orientation* from the check mode menu and again press the *compute* button of the Surface Editor. If triangles with wrong orientation exist, they are displayed in the same way as the intersecting triangles before. For simple cases wrong orientations are repaired automatically. Otherwise you can try to repair them manually. **Important:** A prerequisite for the orientation test is that the outer triangles of the surface are 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 report falsely oriented triangles.

The options *Aspect Ratio*, *Dihed Angle*, and *Tetra Quality* sort all surface triangles according to different quality measures. The worst triangle according to the selected quality measure is displayed, and the worst quality is printed in the console window. Using the buttons of port *Display* you can cycle through all other triangles and edit the surface, if necessary. In order to generate high quality tetrahedral grids afterwards, the following quality numbers should be achieved:

Aspect ratio at least 10, better 20.
Dihedral angle at least 10, better 5.
Tetra quality at least 50, better 25.

If no intersections or wrong orientations are reported anymore, and if you have removed all bad quality triangles, you may leave the editor by clicking on the pencil icon. If you have edited the surfaces, save them into a file again.

1.4.4 Tetrahedron Generation

The last step is generation of a *volumetric tetrahedral grid* from the surfaces. Each tissue type is treated separately for this purpose. First a closed surface is constructed from all triangles corresponding to one tissue type. Then the volume enclosed by that surface is filled with tetrahedra.

Select module *TetraGen* from the popup menu of the simplified and edited surface. Tetrahedron generation typically takes 10 - 15 minutes CPU time. It is performed as a *batch job*, so that you can continue working with HyperPlan while the job is running.

The *Region* port allows you to specify whether tetrahedra should be generated for all tissue types or just for one selected tissue type. The latter choice is needed for testing purposes only. In port *Grid* you can enter the filename for the resulting tetrahedral grid. We recommend the filetype `grid`. If you press button *Materials*, a special dialog window is invoked. It allows you to define a desired mesh size (in cm) for each tissue type. Normally the predefined values should work well.

Press button *Run batch* to submit the batch job. If a file with the name as the specified grid already exists, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. Then press *Run batch* again. Afterwards the *job dialog* window should appear showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running. You can bring up the job dialog at any time by selecting *Jobs* from the *File* menu.

If tetrahedron generation succeeded, the job is marked as being completed. If HyperPlan is still running or is running again, the resulting tetrahedral grid is automatically loaded.

If tetrahedron generation failed, the job is also marked as being completed, but some kind of error message is displayed. If you want to determine the reason for a failure, select the job and press the button *View Output*. Look at the last lines of output created by the tetrahedron generation program.

- If you find the following message:

  ```
  *------------------------------------------------------------*
  * ERROR
  * surface of current region is not closed
  * surface can't be filled with tetrahedra
  *------------------------------------------------------------*
  ```

  an 'open' surface occured for some tissue type. The most probable reason for this failure is that toggle *add empty slices* was not set in the *GMC* module.

- If the message is

  ```
  *------------------------------------------------------------*
  * ERROR
  * surface of current region contains intersecting triangles
  * surface can't be filled with tetrahedra
  *------------------------------------------------------------*
  ```

  some intersecting triangles have been generated during surface simplification. If the surface intersection test has been passed successfully (see Section 1.4.2) this error should not occur.

- If the message is

  ```
  *------------------------------------------------------------*
  * ERROR
  * surface triangles of current region don't have
  * a uniform orientation
  * surface can't be filled with tetrahedra
  *------------------------------------------------------------*
  ```

  some triangles with wrong orientation have been generated during surface simplification. If the surface orientation test has been passed successfully (see Section 1.4.2) this error should not occur.

- If none of the above three error messages occurs, we would ask you to send an e-mail to amira@zib.de reporting on this error. If possible, determine the Cmd Directory of the batch job (it is shown in the job dialog window when you select the job) and attach all files existing in that directory to the e-mail.

1.5 E-Field Simulations

For E-Field simulation HyperPlan offers three alternative approaches. They differ with respect to the underlying geometric model (regular cubic grid or tetrahedral grid) and the numerical method used (FE or FDTD).

Each of HyperPlan's E-field calculation modules computes a separate field for each channel of the hyperthermia applicator. In particular, 4 channels are computed for the Sigma 40 and Sigma 60 applicators, and 12 channels are computed for the Sigma 2000 eye applicator.

1.5.1 General Remarks About the FDTD Method

The current release of HyperPlan contains two different FDTD modules which use different geometry models as input. The first method directly works on a *LabelField*, while the second one requires a tetrahedral patient grid. Internally, both methods operate on a cartesian grid consisting of cubic cells (so-called Yee cells). Both methods currently only support the Sigma 60 applicator.

- FDTD on label fields

  This algorithm merely requires roughly segmented CT data as input. After importing the CT data and resampling it to 256 x 256 pixel resolution, the *LabelVoxel* module should be activated. This module transforms the Hounsfield values of the CT images pixel per pixel into values representing the four tissue compartments *exterior*, *fat*, *muscles* and *bones*. These labels are directly used as input for the FDTD module. The size of the Yee cells has to be adapted to the distance between the CT slices (typically 1 cm). The size is automatically set to either the slice distance (option *fine lattice*) or twice the value of it (option *coarse lattice*). A size of about 1 cm is recommended. Note: Whether you have to enter '1' or '2' to achieve 1 cm cells depends on the slice distance.

- FDTD on tetrahedral grids (modified FDTD Solver)

  This algorithm requires a tetrahedral patient model as input. The tetrahedral grid is used to create a region-based regular geometry model. The solver takes into account the field discontinuities at tissue boundaries. You can choose the size of the cubes as you like. A size of about 1 cm is recommended.

There are some restrictions that you have to keep in mind when working with the FDTD modules. The maximal possible number of cubes (Yee cells) is 82 in x and y directions and 106 in z direction. For modelling the Sigma 60 applicator, the number of cubes should be chosen such that 5 - 10 cells of background medium lie between the applicator/patient bounding box and the boundary of the FDTD lattice (e.g., the number of cubes should be set to about 80 for a cube size of 1 cm and about 48 for a size of 2 cm). The program will stop if the chosen number of cubes is too small to cover the patient's and the applicator's geometry. If necessary, check if the chosen value is not too small. This is the case, if the power deposition pattern in the patient is independent of variations of the number of cubes. The number of iterations should be chosen as at least 2500 for a size of 1 cm and at least 1500 for a size of 2 cm. These iteration numbers prove to be sufficient for modelling the Sigma 60 applicator in the frequency range 80-120 MHz.

Furthermore, the FDTD-inherent resolution lies (because of storage restrictions) in the range of about 1 cm, which is a value much bigger than the pixel resolution of the CT-slices of about 1 mm. Both the voxel based and the modified models are idealizations and cannot contain the full patient geometry information. Keep in mind this important fact especially if the FDTD results are visualized over the CT-background (e.g. using the colorwash option). The relative coarseness of FDTD lattice with respect to the real patient geometry requires averaging and interpolating procedures. Thus solutions obtained may differ depending on the FDTD version used. How to achieve a better field resolution within the patient's body and a better antenna modelling is still a subject of research.

1.5.2 FDTD On Label Fields

Once the CT volume has been labeled, you can start an E-field simulation. From the popup menu of the green LabelField icon select the *Antenna* module. This module defines the hyperthermia applicator. Note: Currently the FDTD method supports only the Sigma 60 device for simulation. However, you can also display the geometry of the Sigma 40 and the Sigma 2000 eye applicator. You can change the position of the applicator relative to the patient.

Next, select the *FDTD* solver from the Antenna's popup menu. The interface of the FDTD module lets you select between two different resolutions. By default a mesh is used with twice the spacing of the CT slices. However, the resulting E-field is interpolated back to the finer resolution. The small resolution should only be used if insufficient memory is available on the computer, i.e., less than about 128 MB, depending on the size of the data set.

You have to type in a file name under which you want to store the resulting E-Fields. HyperPlan makes a suggestion for you, but you can change it if you want. We recommend to include the filetype `EFieldValsVX` in the filename.

If you press the button *Materials*, an editor window is invoked that allows you to change the electric properties of the different tissue types. Note: *The electrical parameters stored in HyperPlan's material database refer to a frequency of 90 MHz.*

E-field calculation will take ca. 40-60 minutes CPU time per applicator channel. It is performed as a *batch job*. Press the *Submit* button to submit the job. If the output file did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, the *job dialog* window appears showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

More information is given in the reference documentation of the *FDTD module*.

1.5.3 FDTD On Tetrahedral Grids

This section describes how to start HyperPlan's FDTD solver for tetrahedral grids.

1. Load one of the tetrahedral patient models (filetype `grid`) into HyperPlan. Select module *DuplicateNodes* from the popup menu of the green data icon representing the patient model. The popup menu is activated using the right mouse button. Press the button *Do it*. After a few seconds a new green icon appears. It represents essentially the same tetrahedral grid as the original one, but now containing duplicated nodes at the boundaries between tissue compartments. These duplicated nodes are necessary for representing the discontinuities of the electromagnetic field. Save the new grid to a file. We recommend to use the filetype `EFieldGridFD`.
2. Load one of the tetrahedral patient models named `EFieldGridFD` into HyperPlan. To visualize the patient model in the 3D viewer, select a *GridVolume* module from the popup menu of the green data icon representing the patient model. The popup menu is activated using the right mouse button.
3. Select the *Antenna* module from the grid's popup menu. A geometric representation of the applicator appears in the 3D viewer. Select the Antenna icon. You may change parameters like position and inclination angle of the applicator. The relative position of the applicator is specified in centimeters. Caution: The *Sigma Eye* button merely changes the geometric representation of the applicator. Currently the FDTD solver only works for the Sigma 60 applicator.
4. Attach the FDTD solver to the applicator by selecting *FDTD* from the popup menu of the Antenna icon. A red icon appears, representing a computational module. Select the icon to bring up the user interface of the FDTD solver. If you want to change dielectric constants or electric conductivities for some tissue types, click on button *Materials* and enter the new values in the editor window. Note: *The electrical parameters stored in HyperPlan's material database refer to a frequency of 90 MHz.*
5. In the field labeled *EFieldVals* you can specify the name of the file containing the simulation results. We recommend to include the filetype `EFieldValsFD` into the filename.
6. An E-field simulation is expected to run several hours on a workstation. Therefore it is performed as a *batch job*. Press the *Submit* button to submit the job. If the output file did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, the *job dialog* window appears showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.
7. It is a good idea to run simulations overnight. You may quit HyperPlan while a job is running. If you restart HyperPlan again, select *Jobs* from HyperPlan's File menu to open the job dialog window again. It will show the current state of the job queue.

More information is given in the reference documentation of the *FDTD module*.

1.5.4 E-Field Calculation Using the FE Method

This section describes how to perform an E-field calculation using the finite element method. This method is the preferred one since it is more accurate than an FDTD simulation. Also, it can be used for the Sigma 40 and Sigma 2000 applicators, not only for the Sigma 60. The whole task can be separated into two different parts, namely in converting the patient model into a so-called *extended grid* and in applying the simulation code itself.

Creating an Extended Grid

Before starting the E-field calculation, you must extend the tetrahedral patient model. Grids have to be added representing the *water bolus* of the hyperthermia applicator and parts of the *surrounding medium*, i.e., air. If you want to perform several simulations with different relative positions of patient and applicator, you must create such an extended grid for each simulation.

1. Load a *tetrahedral patient model* (filetype `grid`) into HyperPlan. To see the model in the 3D viewer, select a *GridVolume* module from the popup menu of the green data icon representing the patient model. The popup menu is activated using the right mouse button. Press button *Buffer Add* in order to see the entire model.
2. Select the *Antenna* module from the model's popup menu. A geometric representation of the applicator appears in the 3D viewer. Select the Antenna icon. You may change parameters like position and inclination angle of the applicator. The relative position of the applicator is specified in centimeters. At port *Applicator* you can select the Sigma 40, the Sigma60, or the Sigma Eye applicator. Simulation of all is supported by the FE method.
3. Select module *BolusGrid* from the antenna's popup menu. Click on the BolusGrid icon. Parameter *Offset* lets you control how the bolus surface bends inwards to fit to the patient's body. Press button *Compute*. If the tetrahedral patient model contains duplicated points, an error message will be issued. *Tetrahedral grids containing duplicated points are only needed if E-field calculation is being performed using the FDTD method*. The error message will include a hint how to remove the duplicated points. Once you have removed them, press button *Compute* again.

   After a few seconds a new green icon will appear. It represents a data object containing the surface of the water bolus (adapted to the patient model) and two spherical surfaces enclosing both the patient model and the water bolus. The spherical surfaces describe the outer boundary of the whole grid required for finite element calculation. If you want to visualize the surfaces, select module *SurfaceView* from the surfaces' popup menu.
4. Before starting tetrahedron generation, it is a good idea to check if the surfaces contain *intersecting triangles*. Click on the green icon representing the surfaces. Then invoke the *Surface Editor* by clicking on the button with the pencil icon. Press *Compute* in order to perform the intersection test. The number of intersections is shown in HyperPlan's Console window. Click again on the pencil icon to leave the editor.

   If there are intersections, you can try to remove them by going back to the *Antenna* module and slightly changing the applicator position. You can also change the *Offset* in module *BolusGrid*. Normally a smaller offset value reduces the probability of intersections.

   Caution: If you want to use surface editing operations to remove the intersections as described in Section 1.4.2), you have to be *very careful*. You must not change any triangles belonging to the patient's surface at this time because they are prescribed by the patient model. If such triangles are changed, the bolus grid and the patient model will not fit together later on.
5. If there are no intersections, select the *TetraGen* module from the surface's popup menu. This module will generate a volumetric tetrahedral grid from the surfaces.

   Tetrahedron generation is performed as a *batch job*. Press button *Run* to submit the batch job. If a file with the name defined at port *Grid* already exists, a warning occurs. If you don't want to overwrite the file, press *Cancel* and change the filename. Then press button *Run* again. The job dialog window appears that shows you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

   When tetrahedron generation is finished and HyperPlan is still running, the tetrahedral grid is automatically loaded.
6. The tetrahedral grid generated in the previous step only covers the water bolus and the surrounding medium (air). You must combine this grid with the tetrahedral patient model from which you started. Select the icon representing the patient model. Then point to the icon of the bolus grid and press the right mouse button. Select module *Combine* from the popup menu. Make sure that *Combine* is connected to both tetrahedral grids. If not, you can point to the small rectangle at the left hand side of the *Combine*

icon and press the right mouse button. Now you can create connect the inputs of the *Combine* module to both grids manually.

We recommend that you use the default settings (all toggles are set). Just press button *DoIt*. After a few seconds a new green icon will appear which represents the combined tetrahedral grid. Save the combined grid into a file. We recommend to use the filetype `ExtendedGrid`.

Applying the FE Method

Now you are ready to start the E-field calculation. Load the *extended tetrahedral grid* (filetype `ExtendedGrid`) into HyperPlan. Select the module called *E-Field Simulation* from the grid's popup menu.

In this module you must enter two filenames, one for the output grid and one for the file containg the E-field values which are going to be computed. The output grid is very similar to the original tetrahedral patient grid. However, it additionally contains a list of the edges of all tetrahedra. HyperPlan suggests two filenames. Check if these are ok. It is recommended to include the filetypes `EFieldGridFE` and `EFieldValsFE` in the filenames.

If you press the *Materials* button, a dialog window is popped up that allows you to change the electric properties of the different tissue types. Note: The electrical parameters stored in HyperPlan's material database refer to a frequency of 90 MHz. The material database itself can be modified permanently by editing the file `HyperPlan/share/materials/databsse.hm`.

E-field calculation will take ca. 10-20 minutes CPU time per applicator channel. It is performed as a *batch job*. Press the *Submit* button to submit the job. If one of the output files does already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, the *job dialog* window appears showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

When the E-field calulation is finished and HyperPlan is still running, the EFieldGrid and EFieldVals files are automatically loaded.

1.6 Temperature Simulation

This section describes how to perform a *temperature simulation* on a tetrahedral patient model. It requires that an E-Field simulation already has been performed, either using the FDTD method as described in Section 1.5.3 or using the FE method as described in Section 1.5.4. Like for the electromagnetic fields a set of independent temperature channels is computed. From these channels the actual temperature distribution for any set of antenna amplitudes and phases can be obtained by superposition very quickly.

1. First you have to load a tetrahedral patient model as well as the corresponding E-field values into HyperPlan. If the FDTD method was used for E-field calculation, the patient models are called `EFieldGridFD`, while the files containing the E-field values have a suffix `EFieldValsFD`. If the FE method was used, the suffixes are `EFieldGridFE` and `EFieldValsFE`, respectively.
2. From the popup menu of the *EFieldVals* icon select the module called *Static Heat*. In this module you can enter the names of the files for the temperature grid and the temperature channel data which are going to be computed. The temperature grid is very similar to the initial electromagnetic coarse grid. However, it does not contain any duplicated vertices, nor does it contain information about grid edges. We recommend to use the filetypes `ThermGridFD` and `ThermValsFD`, or `ThermGridFE` and `ThermValsFE`.

   If you press the *Parameters* button, a dialog window is popped up allowing you to change some global parameters for temperature simulation. You can also change the thermal properties of the different tissue types. For certain tissue types default values are taken from the HyperPlan material database.
3. Temperature calculation will take ca. 10 minutes CPU time. It is performed as a *batch job*. Press the *Submit* button to submit the job. If one of the output files did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been

submitted, the *job dialog* window appears showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

When temperature calulation is finished and HyperPlan is still running, the ThermGrid and ThermVals files are automatically loaded.

1.7 Temperature Optimization

This section describes how to perform a temperature optimization. The result of temperature optimization is a so-called *plan file* containing the power amplitudes and phase settings of the individual channels of the applicator. Before optimization can be started a temperature simulation on a tetrahedral patient grid has to performed, as described in the previous section.

1. From temperature simulation you will get two files, a temperature grid and a temperature channel file. The filetypes are `ThermGridFD` and `ThermValsFD`, or `ThermGridFE` and `ThermValsFE`, respectively. Load both of these files into HyperPlan in order to compute the optimal antenna settings.
2. Check if the patient model contains a region named 'Target'. Optimization will not work without a *Target* region.
3. Select the *Opt Temp* module from the popup menu of the temperature channel file. The optimization module will compute a *plan file*, which contains the optimal amplitudes and phases of the radiowave antennas. As with the other simulation modules you have to specify the name of the result file before the computation can be started. HyperPlan suggests a name, but be sure to check if this name is ok. We recommend to use the filetype `PlanFD` or `PlanFE`, respectively.
4. If a simulation of the Sigma-Eye applicator is to be performed, you can set the number of available power generators – 4 or 12 – at the port *Power Generators*.
5. Press the *Submit* button to start the optimization. Usually temperature optimization takes only a few seconds. Also it does not require much memory. Therefore there is no need to leave HyperPlan before starting the batch queue.
6. When optimization has been finished, the resulting plan file is automatically loaded into HyperPlan. If you want to visualize the optimal temperature distribution, select the *Superpose* module from the popup menu of the *ThermVals* icon. Change the menu at port *Action* from 'Default' to the name of the plan file. Now the Superpose module computes the optimal temperature distribution. You can visualize it by creating an *Isosurface* module or a *FieldCut* module from the popup menu of the *Superposed* icon.

1.8 All-in-one Simulation

HyperPlan also provides a module for performing all simulation steps, namely *E-Field Simulation*, *Temperature Simulation*, and *Temperature Optimization* at once using the *Finite Element* method. For E-Field Simulation adaptive mesh refinement is used. The corresponding module is called *Refined E-Field Simualtion*. It can be attached to an extended grid. For details of how to create such a grid see Section 1.5.4. Details about the *Refined E-Field Simualtion* module are described in the Amira reference manual. In principle the module provides a combination of the input ports of the ordinary *E-Field Simulation* module and the *Static Heat* module as described in Sections 1.5.4 and 1.6. In addition the number of refinement steps can be specified. In most cases a single refinement step yields sufficiently good results. In addition to the *EFieldGrid* and *EFieldVals* files a number of additional data files are written to the output directory. These also includes temperature values and a plan file containing optimized antenna parameters.

Part II

Amira User's Guide

Chapter 2

Introduction

Amira is a 3D visualization and modelling system. It allows you to visualize scientific data sets from various application areas, e.g. medicine, biology, chemistry, physics, or engineering. 3D objects can be represented as grids suitable for numerical simulations, notably as triangular surface and volumetric tetrahedral grids. Amira provides methods to generate such grids from voxel data representing an image volume, and it includes a general purpose interactive 3D viewer.

Section 2.1 (Overview) provides a short overview of the fundamentals of Amira, i.e. its object-oriented design and the concept of data objects and modules.

Section 2.2 (Features) summarizes key features of Amira, for example direct volume rendering, image processing, and surface simplification.

2.1 Overview

Amira is a modular and object-oriented software system. Its basic system components are modules and data objects. Modules are used to visualize data objects or to perform some computational operations on them. The components are represented by little icons in the *object pool*. Icons are connected by lines indicating processing dependencies between the components, i.e., which modules are to be applied to which data objects. Data objects of specific types are created automatically from file input data when reading such or as ouput of module computations, modules matching an existing data object are created as instances of particular module types via a context-sensitive popup menu. Networks can be created with a minimal amount of user interaction. Parameters of data objects and modules can be modified in Amira's interaction area.

For some data objects such as surfaces or colormaps there exist special-purpose interactive editors that allow the user to modify the objects. All Amira components can be controlled via a Tcl command interface. Commands can be read from a script file or issued manually in a separate console window.

The biggest part of the screen is occupied by a 3D graphics window. Additional 3D views can be created if necessary. Amira is based on the latest release of the TGS Open Inventor graphics toolkit. In addition, several modules apply direct OpenGL rendering to achieve special rendering effects or to maximize performance. In total, there are more than 120 data object and module types. They allow the system to be used for a broad range of applications. User-defined extensions are facilitated by the Amira developer version.

2.2 Features

Amira provides a large number of module types allowing you to visualize various kinds of scientific data as well as to create polygonal models from 3D images. All visualization techniques can be arbitrarily combined to produce a single scene. Moreover, multiple data sets can be visualized simultaneously, either

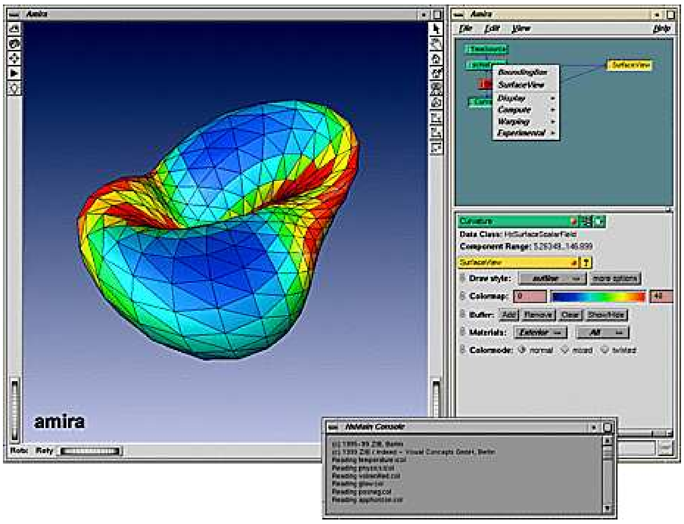


Figure 2.1: Data objects and modules are represented as little icons in the object pool (top right). In the 3D graphic window a surface colored according to its curvature is shown. Curvature information has been computed by a computational module and is stored as a separate data object. In the mid right window the parameters of selected modules (here: Curvature and SurfaceView) are shown. The window at the bottom provides a Tcl-command shell.

in several viewer windows or in a common one. A built-in transformation editor makes it easy to register data sets with respect to each other or to deal with different coordinate systems.

2.2.1 Direct Volume Rendering

One of the most intuitive and most powerful techniques for visualizing 3D image data is *direct volume rendering*. Light emission and light absorption parameters are assigned to each point of the volume. Simulating the transmission of light through the volume makes it possible to display your data from any view direction without constructing intermediate polygonal models. By exploiting modern graphics hardware, Amira is able to perform direct volume rendering almost in realtime, even for data volumes of 40 megabytes and more. Volume rendered images can be combined with any type of polygonal display. This improves the usefulness of this technique significantly. Moreover, multiple data sets can be volume rendered simultaneously – a unique feature of Amira. Transfer functions with different characteristics required for direct volume rendering can either be generated automatically or edited interactively using an intuitive colormap editor.

2.2.2 Isosurfaces

Isosurfaces are most commonly used for analyzing arbitrary scalar fields sampled on a discrete grid. Applied to 3D images, the method provides a very quick, yet sometimes sufficient method for reconstructing polygonal surface models. Beside standard algorithms, Amira provides an improved method which generates significantly fewer triangles with very little computational overhead. In this way large 3D data sets can be displayed interactively even on smaller desktop graphics computers. Like other polygonal models, isosurfaces can be colored in order to visualize a second independent data set. Another highlight comprises the realistic view-dependent way of rendering semi-transparent surfaces. By correlating transparency with local orientation of the surface relative to the viewing direction, complex spatial structures can be understood much more easily.

2.2.3 Segmentation

Amira also provides a component for 3D image segmentation with several special-purpose features. This component is called image segmentation editor. It offers a large set of segmentation tools, ranging from purely manual to fully automatic. Among others, the following tools are provided: brush (painting), lasso (contouring), magic wand (region growing), thresholding, intelligent scissors, contour fitting (snakes), contour interpolation and extrapolation, various filters including smoothing, cleaning, and connected component analysis. Although the display is slice-oriented, many tools can be applied in both 2D and 3D. Since the editor does not store contours surrounding regions but region labels, a unique and well-defined classification is guaranteed.

2.2.4 Surface Reconstruction

Once the interesting features in a 3D image volume have been segmented, Amira is able to create a corresponding polygonal surface model. The surface may have non-manifold topology if there are locations where three or more regions join. Even in this case the polygonal surface model is guaranteed to be topologically correct, i.e. free of self-intersections. Fractional weights which are automatically generated during segmentation allow the system to produce smooth boundary interfaces. This way realistic high-quality models can be obtained, even if the underlying image data are of low resolution or contain severe noise artifacts. Making use of innovative acceleration techniques, surface reconstruction can be performed very quickly. Moreover, the algorithm is robust and fail-safe.

2.2.5 Surface Simplification

Surface simplification is another prominent feature of Amira. It can be used to reduce the number of triangles in an arbitrary surface model according to a user-defined value. Thus, models of finite-element grids, suitable for being processed on low-end machines, can be generated. The underlying simplification algorithm is one of the most elaborate ones available. It is able to preserve topological correctness, i.e., self-intersections commonly produced by other methods are avoided. In addition, the quality of the resulting mesh, according to measures common in finite element analysis, can be controlled. For example, triangles with long edges or triangles with bad aspect ratio can be suppressed.

2.2.6 Generation of Tetrahedral Grids

Amira allows you not only to generate surface models from your data but also to create true volumetric tetrahedral grids suitable for advanced 3D finite-element simulations. These grids are constructed using a flexible advancing-front algorithm. Again, special care is taken to obtain meshes of high quality, i.e., tetrahedra with bad aspect ratio are avoided. Several different file formats are supported, so that the grid can be exported to many standard simulation packages. In the developer version additional file formats can easily be added by the user.

Chapter 3

First steps in Amira

This chapter contains step-by-step tutorials illustrating the use of Amira. The tutorials are almost independent of each other, so after reading the basics in the Getting Started section it is possible to follow each tutorial without knowing the others. If you go through all tutorials you will get a good survey of Amira's basic features.

In all tutorials the steps to be performed by the user are marked by a dot. If you only want to get a quick idea how to work with Amira you may skip the explanations between successive steps and just follow the instructions. But in order to get a deeper understanding you should refer to the text.

Section 3.1 (Getting started) explains some very basic principles of Amira. You will learn how to load a data set from a file, and how to create a module in order to visualize it.

Section 3.2 (Scalar field visualization) demonstrates how to read and manipulate 3D image data. You will learn how to display isosurfaces and how to apply direct volume rendering.

Section 3.3 is devoted to *vector field visualization*. You will explore interesting visualization methods such as line integral convolution (LIC) and illuminated field lines.

Section 3.4 (Grid generation) demonstrates all the steps required to reconstruct a polygonal model from 3D image data. The tutorial also shows how to convert a surface model into a volumetric tetrahedral grid suitable for finite-element simulations.

Finally, Section 3.5 explains how to *warp two 3D objects* into each other by defining landmarks.

Note: If you want to visualize your own data, please first refer to Section 5.1. This section contains some general hints on how to import data sets into Amira.

3.1 Getting Started

In this section you will learn how to

1. start the program
2. load a demo data set into the system
3. invoke editors for editing the data
4. connect visualization modules to the data
5. interact with the 3D viewer.

The following text has the form of a short step-by-step tutorial. Each step builds on the steps described before. We recommend that you read the text online and to carry out the instructions directly on the computer. Instructions are indicated by a dot so you can execute them quickly without reading the explanations between the instructions.

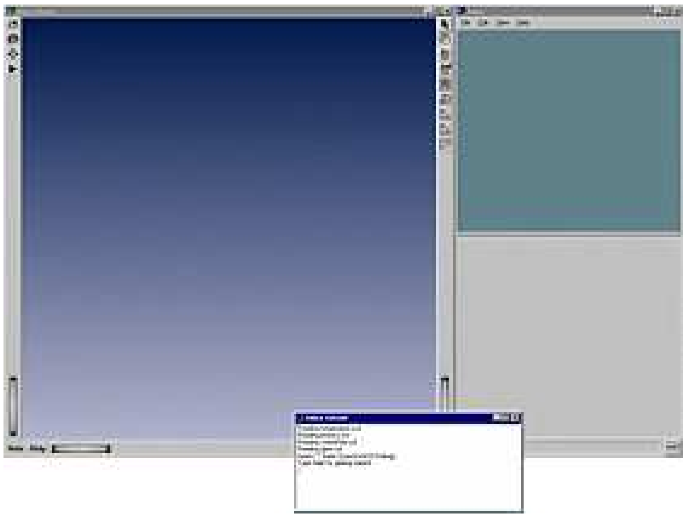


Figure 3.1: The Amira user interface consists of four major parts: the 3D viewer (1), the object pool (2), the working area (3), and the console window (4).

- On a Windows system, select the Amira icon from the start menu. On a Unix system, start Amira by entering `amira` in a shell window.

If there is no such command, the software has not been properly installed. In this case try to execute the script `bin/start` located in the Amira root directory.

When Amira is running, windows like those shown in Figure 3.1 appear on the screen. The user interface is divided into four major parts. The 3D viewer window displays visualization results, e.g. slices or iso-surfaces. The object pool will contain small icons representing data objects and modules. The working area displays interface elements (*ports*) associated with Amira objects. Finally, the console window prints system messages and lets you enter Amira commands.

Amira provides an integrated hypertext browser. You may use this browser to read the user's guide online. In order to activate the browser, type `help` in the console window or select *User's Guide* from the *Help* menu of Amira's main window.

3.1.1 Loading Data

Usually, the first thing you will do after starting Amira is to load a data set. Let's see how this can be done:

- Choose *Load ...* from the *File* menu.

After selecting this menu item, the file dialog appears (see Figure 3.2). By default the dialog displays the contents of the the first directory defined in the environment variable `AMIRA_DATADIR`. If no such variable exists the contents of Amira's demo data directory are displayed. You can quickly switch to other directories, e.g. to the current working directory, using the directory list located in the upper part of the dialog window.

Amira is able to determine many file formats automatically, either by analyzing the file header or the file name suffix. The format of a particular file will be printed in the file dialog right beside the file name.

Now, we would like to load a scalar field from one of the demo data directories contained in the Amira distribution.

- Change to the directory `data/tutorials`, select the file `lobus.am` and press *OK*.

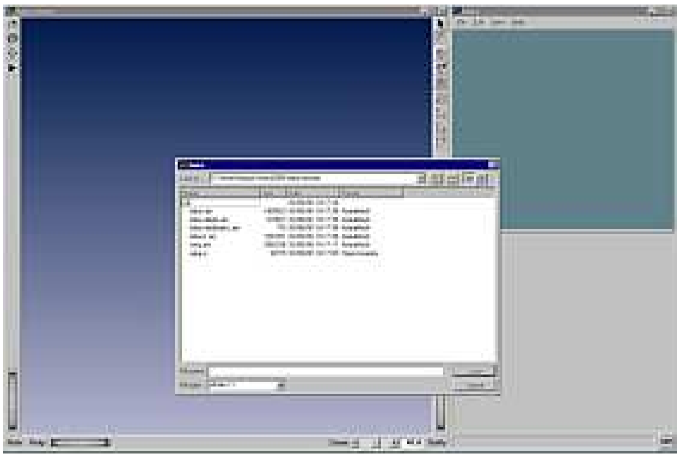


Figure 3.2: Data sets can be loaded into Amira using the file browser. In most cases, the file format can be determined automatically. This is done by either analyzing the file header or the file name suffix.

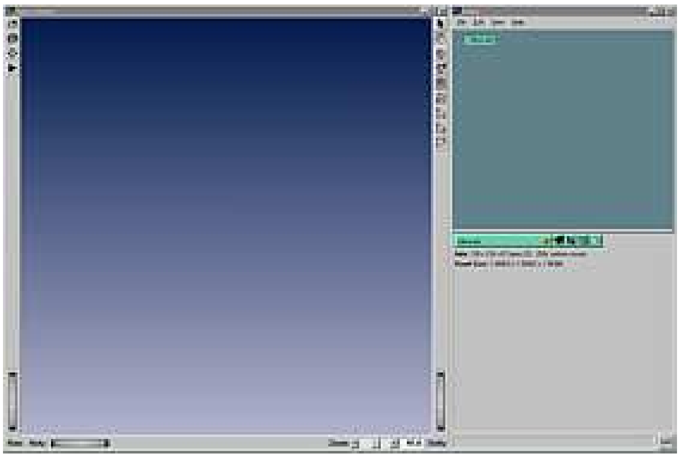


Figure 3.3: Data objects are represented by little green icons in the object pool. Once an icon has been selected information about the data set such as its size or its coordinate type is displayed in the working area.

The data will be loaded into the system. Depending on its size this may take a few seconds. The file is stored in Amira's native *AmiraMesh* format. The file `lobus.am` contains 3D image data of a part of a fruit fly's brain, namely an optical lobe, obtained by confocal microscopy. This means the data represents a series of parallel 2D image slices across a 3D volume. Once it has been loaded, the data set appears as a little green icon in the object pool. In the following we call this data set "lobus data seta".

- Click on the green data icon with the left mouse button to select it.

This causes some information about the data record to be displayed in the working area (Figure 3.3). In our case we can read off the dimensions of the data set, the primitive data type, the coordinate type, as well as the voxel size. To deselect the icon, click on an empty area in the object pool window. You may also pick the icon with the left mouse button and drag it around in the object pool.

3.1.2 Invoking Editors

After selecting an object, in addition to the textual information, some buttons appear in the working area, right next to the data object's name. These buttons represent *editors* which can be used to interactively

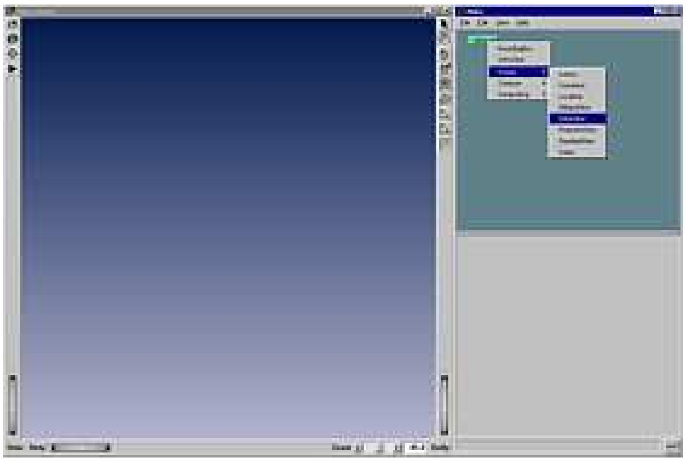


Figure 3.4: In order to attach a module to a data set, click on the green icon using the right mouse button. A popup menu appears containing all modules which can be used to process this particular type of data.

manipulate the data object in some way. For example, all data objects provide a *parameter editor*. This editor can be used to edit arbitrary attributes associated with the data set, e.g. filename, original size, or bounding box. Another example is the *transform editor* which can be used to translate or rotate the data in world coordinates. However, at this point we don't want to go into details. We just want to learn how to create and delete an editor:

- Invoke one of the editors by clicking on an editor icon.
- Close the editor by clicking again on the editor icon.

Further information about particular editors is provided in the user's reference manual.

3.1.3 Visualizing Data

Data objects like the lobus data can be visualized by attaching *display modules* to them. Each icon in the object pool provides a popup menu from which matching modules, i.e., modules that can operate on this specific kind of data, can be selected. To activate the popup menu

- click with the right mouse button on the green data icon. Choose the entry called *BoundingBox*.

After you release the mouse button a new *BoundingBox* module is created and is automatically connected to the data object. The *Bounding Box* object is represented by a yellow icon in the object pool and the connection is indicated by a blue line connecting the icons. At the same time, the graphics output generated by the *BoundingBox* module becomes visible in the 3D viewer. Since the output is not very interesting, in this case we will connect a second display module to the data set:

- Choose the entry called *OrthoSlice* from the popup menu of the lobus data set.

Now a 2D slice through the optical lobe is shown in the viewer window. Initially, a slice oriented perpendicular to the z-direction and centered inside the image volume is displayed. Slices are numbered 0,

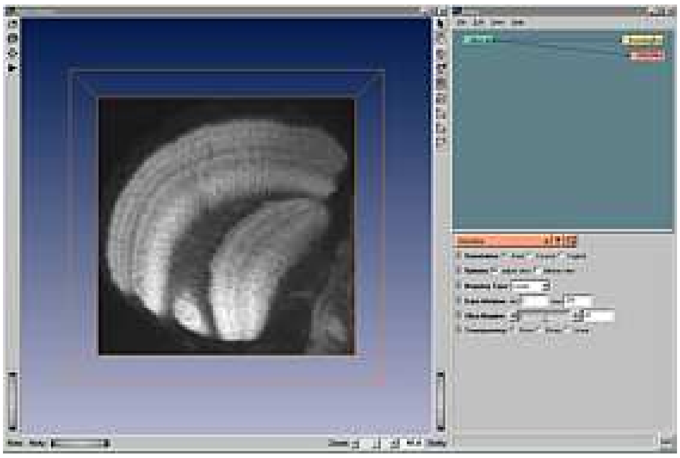


Figure 3.5: Visualization results are displayed in the 3D viewer window. Parameters or ports of a module are displayed in the working area after you select the module.

1, 2, and so on. The slice number as well as the orientation are parameters of the *OrthoSlice* module. In order to change these parameters, you must select the module. Like for the green data icon, this is done by clicking on the *OrthoSlice* icon with the left mouse button. By the way, in contrast to the *BoundingBox*, the *OrthoSlice* icon is orange, indicating that this module can be used for clipping.

- Select the *OrthoSlice* module.

Now you should see various buttons and sliders in the working area, ordered in rows. Each row represents a *port* allowing you to adjust one particular control parameter. Usually, the name of a port is printed at the beginning of a row. For example, the port labeled *Slice Number* allows you to change the slice number via a slider.

- Select different slices using the *Slice Number* port.

By default, *OrthoSlice* displays slices with axial orientation, i.e., perpendicular to the z-direction. However, the module can also extract slices from the image volume perpendicular to x- and y-direction. These two alternate orientations are referred to as *sagittal* and *coronal* (these are standard phrases used in radiology).

3.1.4 Interaction with the Viewer

The 3D viewer lets you look at the model from different positions. Moving the mouse inside the viewer window with the left mouse button pressed lets you rotate the object. With the middle mouse button you can translate the object. For zooming press both the left and the middle mouse button at the same time and move the mouse up or down.

Notice that the mouse cursor has the shape of a little hand inside the viewer window. This indicates that the viewer is in viewing mode. By pressing the `ESC` key you can switch the viewer into interaction mode. In this mode, interaction with the geometry displayed in the viewer is possible by mouse operations. For example, when using *OrthoSlice* you can change the slice number by clicking on the slice and dragging it.

- Select different buttons of the *Orientation* port of the *OrthoSlice* module.
- Rotate the object in a more general position.
- Disable the *adjust view* toggle in the *Options* port.
- Change the orientation using the *Orientation* port again.
- Choose different slices using the *Slice Number* port or directly in the viewer with the interaction mode described above.

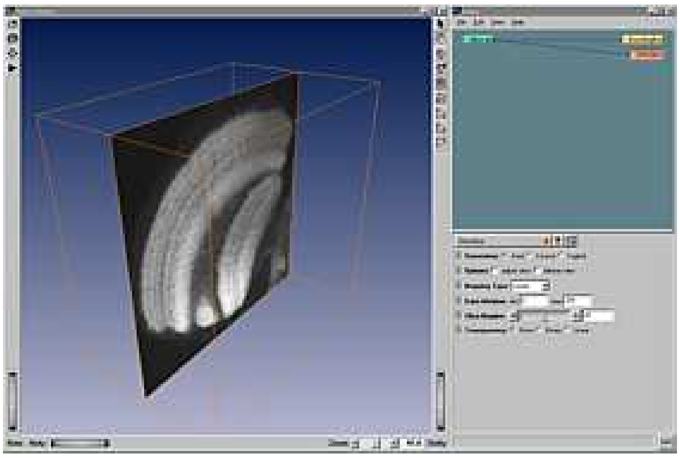
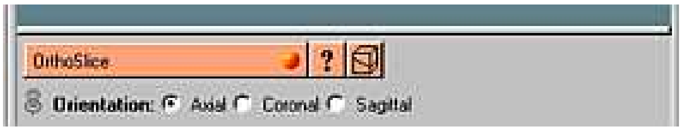


Figure 3.6: The *OrthoSlice* module is able to extract arbitrary orthogonal slices from a regular 3D scalar field or image volume.

Each display module has a *viewer toggle* by which you can switch off the display without removing the module. This button is attached to the colored bar where the module name is shown as illustrated below.

- Deactivate and activate the display of the *OrthoSlice* or *BoundingBox* module using the *viewer toggle*.



If you want to remove a module permanently, select it and choose *Remove* from the edit menu. Choose *Remove All* from the same menu to remove all modules.

- Remove the *BoundingBox* module by selecting its icon and choosing *Remove* from the *Edit* menu.
- Remove all remaining modules by choosing *Remove All* from the same menu.

Now the object pool should be empty again. You may continue with the next tutorial, i.e., the one on *scalar field visualization*.

3.2 Visualization of Scalar Fields

This chapter provides a step-by-step introduction to the visualization of regular scalar fields, e.g. 3D image data. Amira is able to visualize more complex data sets, such as scalar fields defined on curvilinear or tetrahedral grids. Nevertheless, in this section we consider the simplest case, namely scalar fields with regular structure. Each step builds on the step before. In particular, the following topics will be discussed:

1. orthogonal slices
2. simple threshold segmentation
3. resampling the data
4. displaying an isosurface
5. cropping the data
6. volume rendering

We start by loading the data you already know from Section 3.1 (Getting Started): a 3D image data set of a part of a fruit fly's brain. The data set has been recorded with a confocal laser scanning microscope at the

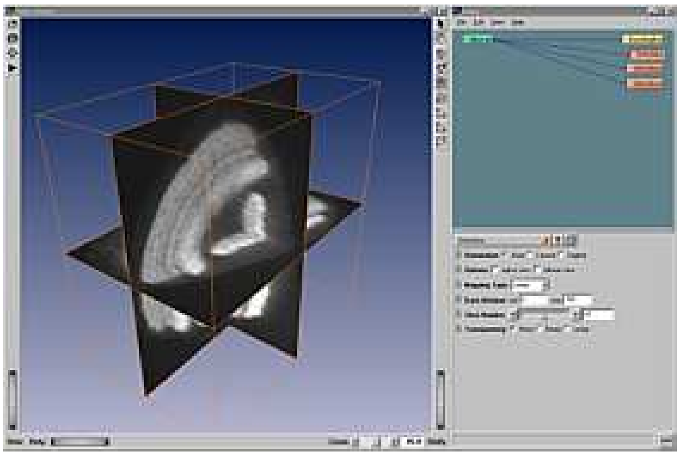


Figure 3.7: Lobus data set visualized using three orthogonal slices.

University of Wuerzburg.

- Load the file `lobus.am` located in subdirectory `data/tutorials`.

3.2.1 Orthogonal Slices

The fastest and in many cases most "standard" way of visualizing 3D image data is by extracting orthogonal slices from the 3D data set. Amira allows you to display multiple slices with different orientations simultaneously within a single viewer.

- Connect a *BoundingBox* module to the data (use right mouse on lobus.am).
- Connect an *OrthoSlice* module to the data.
- Connect a second and third *OrthoSlice* module to the data.
- Select *OrthoSlice2* and press *coronal* in the orientation port.
- Similarly, for *OrthoSlice3* choose *sagittal* orientation.
- Rotate the object in the viewer to a more general position.
- Change the slice numbers of the three *OrthoSlice* modules in the respective ports or directly in the viewer as described in section Getting Started.

In addition to the *OrthoSlice* module, which allows you to extract slices orthogonal to the coordinate axes, Amira also provides a module for slicing in arbitrary orientations. This more general module is called *ObliqueSlice*. You might want to try it by selecting it from the lobus data popup menu.

3.2.2 Simple Data Analysis

The values of the *data window* port of the *OrthoSlice* module determine which scalar values are mapped to black or white, respectively. If you choose a range of e.g. 30...100, any value smaller or equal to 30 will become black, and all pixels with an associated value of more then 100 will become white. Try modifying the range. This port provides a simple way of determining a threshold, which later can be used for segmentation, e.g. in biology or medicine to separate background pixels from anatomical structures. This can be most easily done by making the minimum and maximum values coincide.

- Remove two of the *OrthoSlice* modules.
- Select the remaining *OrthoSlice* module.
- Make sure that the *mapping type* is set to *linear*.

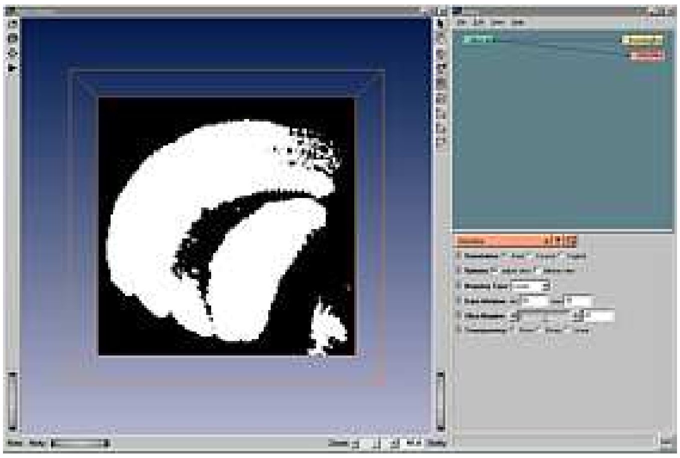


Figure 3.8: By adjusting the data window of the *OrthoSlice* module a suitable value for threshold segmentation can be found. Intensity values smaller than the min value will be mapped to black, intensity values bigger than the max value will be mapped to white.

- Change the minimum and maximum values of the data window port until these values are the same and a suitable segmentation result is obtained. For this data set 85 should be a good threshold value.

A more powerful way of quantitatively examining intensity values of a data set is to use a data probing module *PointProbe* or *LineProbe*. However, we will not discuss these modules in this introductory tutorial.

3.2.3 Resampling the Data

Now we are going to compute and display an *iso-surface*. Before doing so, we will resample the data. The resampling process will produce a data set with a coarser resolution. Although this is not necessary for the iso-surface tool to work, it decreases computation time and improves rendering performance. In addition, you will get acquainted with another type of module. The *Resample* module is a computational module. Computational modules are represented by red icons and typically have a *DoIt* button to start the computation. After you press this button they produce a new data object containing the result.

- Connect a *Resample* module to the data and select it.
- Enter values for a coarser resolution, e.g. x=64, y=64, z=43.
- Press the *DoIt* button.

A new green data icon representing the output of the resample computation named *lobus.Resampled* is created. You can treat this new data set like the original lobus data. In the popup menu of the resampled lobus you will find exactly the same attachable modules and you can save and load it like the original data.

You may want to compare the resampled data set with the original one using the *OrthoSlice* module. You can simply pick the blue line indicating the data connection and drag it to a different data source. Whenever the mouse pointer is over a valid source, the connection line appears highlighted in yellow.

3.2.4 Displaying an Isosurface

For 3D image data sets, isosurfaces are useful for providing an impression of the 3D shape of an object. An isosurface encloses all parts of a volume that are brighter than some user-defined threshold.

- Turn off the viewer toggle of the *OrthoSlice* module.
- Connect an *Isosurface* module to the resampled data record and select it.
- Adjust the threshold port to 85 or a similar value.

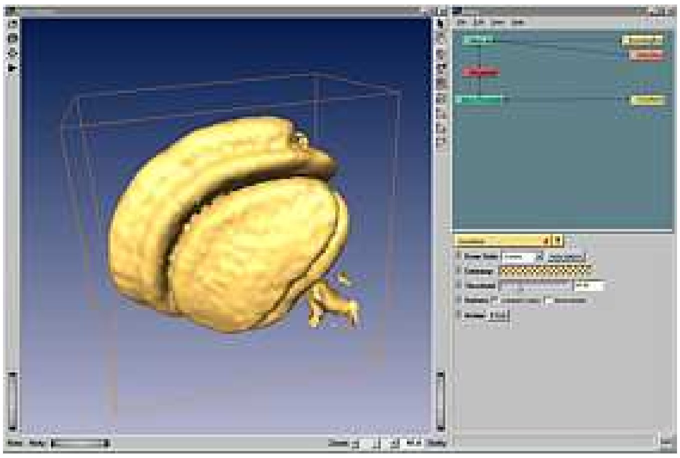


Figure 3.9: Lobus data set visualized in 3D using an isosurface.

- Press the *Do It* button.

3.2.5 Cropping the Data

Cropping the data is useful if you are interested in only a part of the field. A crop editor is provided for this purpose. Its use is described below:

- Remove the resampled data *lobus.Resampled*.
- Activate the display of the *OrthoSlice* module.
- Select the *lobus.am* data icon.
- Click on the green cropping icon in the working area (■).

A new window pops up. There are two ways to crop the data set. You can either type the desired ranges of x, y, and z coordinates into the crop editor's window or put the viewer into interaction mode and adjust the crop box using the green handles directly in the viewer window.

- Put the viewer into interaction mode.
- With the left mouse button, pick one of the green handles attached to the crop volume. Drag and transform the volume until the part of the data you are interested in is included.
- Press *OK* in the crop editor's dialog window.

The new dimensions of the data set are given in the working area. If you want to work with this cropped data record in later sessions you should save it by choosing *Save Data As ...* from the *File* menu.

As you already might have noticed, the crop editor also allows you to rescale the bounding box of the data set. By changing the bounding box alone, no voxels will be cropped. You may also use the crop editor to enlarge the data set, e.g. by entering a negative value for the *k min* number. In this case the first slice of the data set will be duplicated as many times as necessary. Also, the bounding box will be updated automatically.

3.2.6 Volume Rendering

Volume Rendering is a visualization technique that gives a 3D impression of the whole data set without segmentation. The underlying model is based on the emission and absorption of light that pertains to every voxel of the view volume. The algorithm simulates the casting of light rays through the volume from pre-set

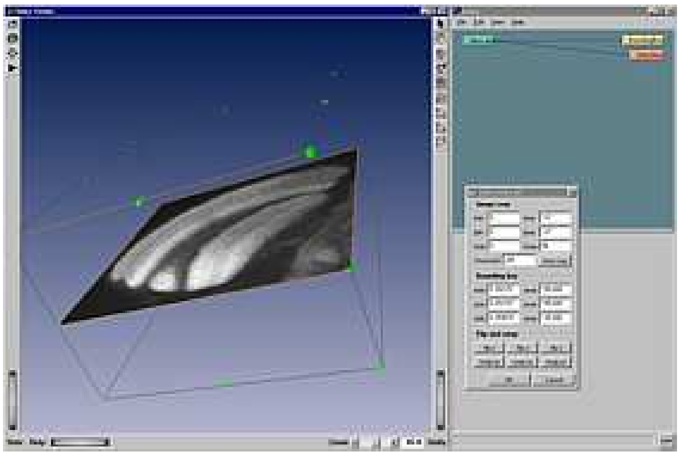


Figure 3.10: The crop editor works on uniform scalar fields. It allows you to crop a data set, to enlarge it by replicating boundary voxels, or to modify its coordinates, i.e. to scale or shift its bounding box.

sources. It determines how much light reaches each voxel on the ray and is emitted or absorbed by the voxel. Then it computes what can be seen from the current viewing point as implied by the current placement of the volume relative to the viewing plane, simulating the casting of sight rays through the volume from the viewing point.

You can choose between two different methods for these computations: *maximum intensity* projections or an ordinary emission-absorption model.

- Remove all objects in the Object Pool other than the *lobus.am* data record.
- Connect a *Voltex* module to the data.
- Select the data icon and read off the range of data values printed on the first info line (22...254).
- Select the *Voltex* module and enter the range in the *Colormap* port.
- Click the *DoIt* button in order to perform some texture preprocessing which is necessary for visualizing the data.

By default emission-absorption volume rendering is shown. The amount of light being emitted and absorbed by a voxel is taken from the color and alpha values of the colormap connected to the *Voltex* module. In our example the colormap is less opaque for smaller values. You may try to set the lower bound of the colormap to 40 or 60 in order to get a better feeling for the influence of the *transfer function*. In order to compute *maximum intensity* projections, choose the *mip* option of port *Mode*.

Internally, the voltex module makes heavy use of OpenGL texture mapping. Both textures modes, 2D and 3D, are implemented. 3D textures yield slightly better results. However, this mode is only supported on high end graphics workstations such as SGI High or Maximum Impact, SGI Octane SSI, MXI or MXE, SGI Infinite Reality, HP fx/4 and fx/6. In particular, it is currently not supported under Microsoft Windows. The 3D texture mode requires you to adjust the number of slices cut through the image volume. The higher this number the better the results are.

Alternatively, 2D textures can be used for volume rendering. In this case slices perpendicular to the major axes are used. You may observe how the slice orientation changes if you slowly rotate the data set. The 2D texture mode is well suited for mid-range graphics workstations with hardware accelerated texture mapping. If your computer does not support hardware texture mapping at all, you should use visualization techniques other than volume rendering.

- Set the *em/ab* button of port *Mode*.
- If you are using 3D texture mode, choose about 200 slices.

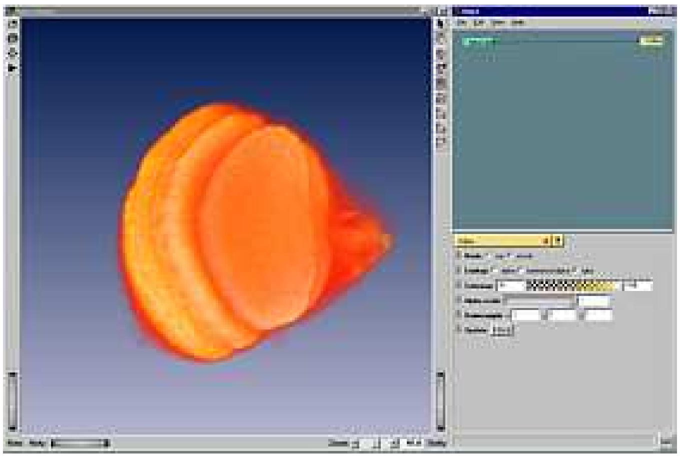


Figure 3.11: The *Voltex* module can be used to generate maxmimum intensity projections as well as volume renderings based on an emission-absoprtion model. In both cases, 2D or 3D texture mapping techniques can be applied applied.

- Click with the right mouse button on port *Colormap* and choose *volrenRed.icol*.
- Set *Lookup* to RGBA and change the min and max values of the colormap to 40 and 150.
- Finally, press *Do It* in order to initialize the *Voltex* textures.

Whenever you choose a different colormap or change the min and max values of the colormap, you must press the *Do It* button again. This causes the internal texture maps to be recomputed. An exception are SGI systems with Infinite Reality graphics. On these platforms a hardware-specific OpenGL extension is exploited, causing colormap changes to take effect immediately.

3.3 Visualization of Vector Fields

This step-by-step-tutorial briefly explains some Amira modules for vector field visualization. The use of these modules is explained by way of data representing the flow around an airfoil. The two methods referred to in steps 2 and 3 are independent of each other. These topics will be covered:

1. Loading the Wing and the Flow Field
2. Line Integral Convolution
3. Illuminated Stream Lines

3.3.1 Loading the Wing and the Flow Field

As in the previous tutorials, we use the file dialog to import data.

- Import the geometry of the wing by loading the file `wing.iv` from the directory `data/tutorials`.
- Attach an *IvDisplay* module to this data object.
- Load the vector field data set called `wing.am` from the directory `data/tutorials`.

The extension *.iv* indicates that the wing geometry is defined in the Open Inventor file format. The *IvDisplay* module can be used for displaying geometry in this format. The vector field itself is stored in Amira's native *AmiraMesh* file format. The data represents the flow around an airfoil computed on a regular grid with curvilinear coordinates. By selecting the green data icon *wing.am*, you can find out that the number of grid nodes in x,y,z-direction is 125 x 41 x 21.

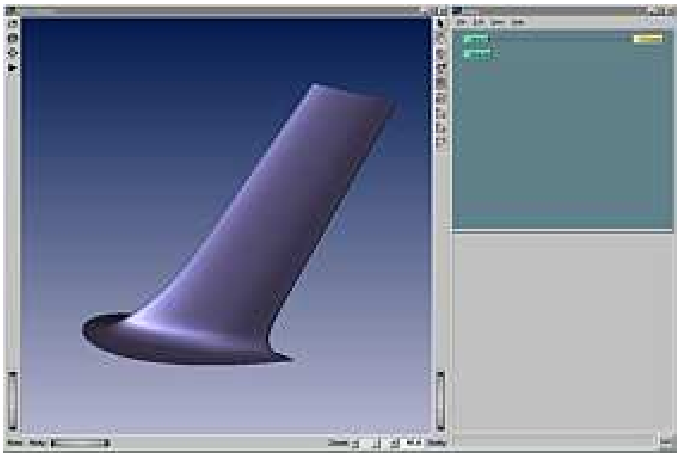


Figure 3.12: Open Inventor geometry of the airfoil.

3.3.2 Line Integral Convolution

Line Integral Convolution (LIC) is a method for visualizing 2D vector fields, i.e., depicting the vector field direction for a suitably sampled subset of points in the 2D domain. The direction is represented by local streamlines, i.e. by curves whose tangent vectors coincide with those of the given vector field. Local streamlines are computed such that all image pixels are covered. The streamlines are projected onto a random noise input texture map of the same size as the vector field domain. The projection step involves summations of texture pixel intensities along streamline paths by way of a convolution integral with a filter kernel. This causes pixel intensities in the resulting image to be highly correlated along individual streamlines but statistically independent perpendicular to the latter. Thus the directional structure of the vector field becomes clearly visible.

Here we use the 3D vector field that we have already loaded and visualize two-dimensional slices:

- Connect a *PlanarLIC* module to the vector field *wing.am* and select it.
- Choose the slice number 22 in the *Translate* port.
- Set filter length to 40 and resolution to 200.
- Press the *Do It* button of the *Action* port.

Whenever the projection plane of the *PlanarLIC* module is changed or other values for filter length or resolution are taken, the LIC texture must be recalculated, i.e., the *Do It* button must be pressed again. Otherwise, a checkerboard pattern will be displayed. Experiment with different filter lengths and resolution values to see what kinds of textures can be produced.

To get an impression of the magnitude of the vectors, you can apply a colormap to the image. Some default colormaps are already loaded when Amira starts up. The corresponding icons usually will be hidden. In order to use one of the colormaps, you have to select it explicitly:

- From the main window's *Edit Show* menu select *temperature.icol* or any other colormap: the corresponding icon becomes visible.
- Select *Magnitude* from port *Colorize* of the *PlanarLIC* module. A new port labeled *Colormap* appears in the working area.
- Connect the colormap port to *temperature.icol* by clicking with the right mouse button in the red rectangle and choosing *temperature.icol* from the appearing popup menu.

The two integer values of the colormap port specify the range of values to which the colormap is applied. Vector magnitudes within this range are depicted symbolically by coloring streamline pixels such that each

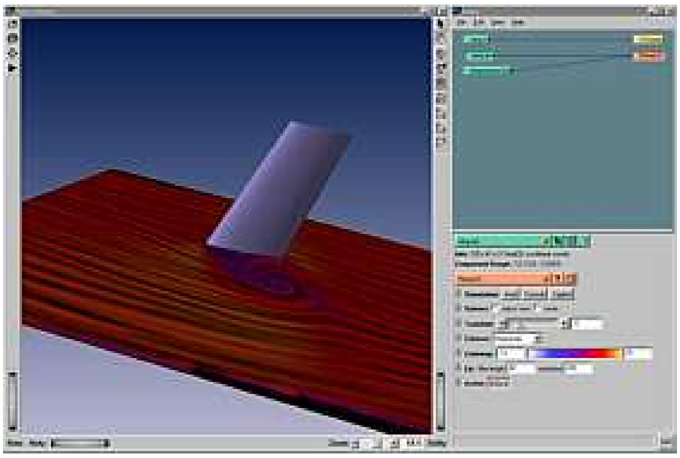


Figure 3.13: Air flow around the wing visualized using Line Integral Convolution

pixel gets the unique color associated with the magnitude value by the *temperature.icol* colormap.

- Select the *wing.am* icon and read off the magnitude range.
- Shift-select the *PlanarLIC* icon and enter the magnitude range into the *Colormap* port.

3.3.3 Illuminated Stream Lines

Illuminated Stream Lines is a technique for interactive 3D vector field visualization which makes use of large numbers of properly illuminated stream lines. A realistic shading model is employed which significantly increases realism of the resulting images and enhances spatial perception.

Now you will learn which tools are used for illuminated stream line visualization and how to use them to get a 3D impression of our airflow vector field.

- Remove the *PlanarLIC* module or disable its display.
- Connect a *DisplayISL* module to the vector field *wing.am*.
- Set *Num Lines* to 300.
- Press the *DoIt* button of the *Distribute* port.
- Click on the *TabBox* button of port *Box* and zoom out the viewer if you cannot see a box.

A *TabBox* appears in the viewer. Only stream lines flowing through this box are visible. The green ticks at the corners and edges of the box allow you to change the dimensions of the box.

- Switch the viewer into interaction mode.
- Try out what happens if you click with the right mouse button on one of the green ticks on the corners or edges of the *TabBox* and drag them around.
- To move the whole *TabBox*, click with the right mouse button in the box and move it.
- Try to get the *TabBox* into a shape and position as shown in the image.
- Press *DoIt* button again in order to recalculate the stream lines.

Now some information about Console Window commands. Most Amira modules provide more control features than those that are available by the ports displayed in the Working Area. All of them are available by commands that you type in the console window. You can get a list of commands associated with a particular module currently in use just by entering its name. Now we will use such commands to form and position the TabBox exactly as in the image above.

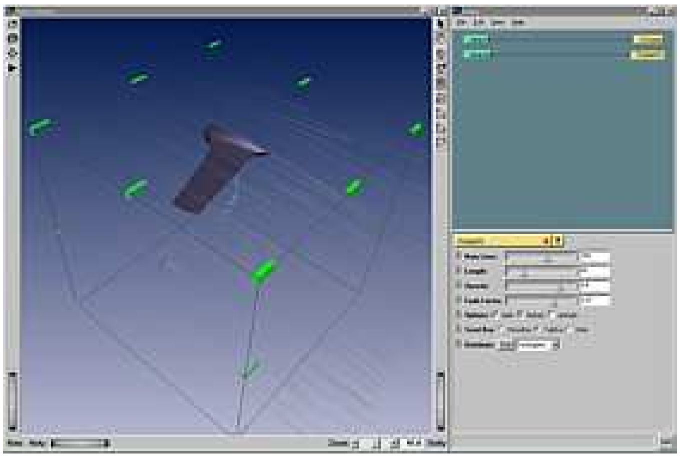


Figure 3.14: Illuminated streamlines around a wing

- Type “DisplayISL” into the Console Window.
- Type “DisplayISL getBox” into the Console Window.

The first command lists all commands of the *DisplayISL* module and the second shows the scale and translation of the current TabBox. The first word of a command always has to be the name of a module as shown on its icon. Note that module commands are not recognized unless corresponding modules have been loaded into the object pool. However, you do not need to select a module for typing in its commands.

- Type “DisplayISL setBoxTranslation -0.31 0.00 0.17”
- Type “DisplayISL setBoxScale 0.11 0.04 0.14”

Now the sub-field that corresponds to the clipping of the vector field as implied by the settings of the *TabBox* should look like the one shown in the image above.

3.4 Grid Generation from 3D Image Data

By following this step-by-step tutorial you will learn how to generate a surface grid and a volumetric tetrahedral grid for an object embedded in a voxel data set. A surface grid is suitable for producing a 3D view of the object’s surface, a volumetric tetrahedral grid is the basis for producing various views of inner parts of the object, e.g. cuts through it, and both kinds of grids are frequently used for numerical simulations.

The generation process consists of these steps:

1. Threshold Segmentation
2. Refining Segmentation Results
3. Extracting Surfaces from Segmentation Results
4. Simplifying the Surface
5. Generation of a Tetrahedral Grid

3.4.1 Threshold Segmentation

The first step is to separate the object from the background. This is done by segmenting the volume into exterior and interior regions on the basis of the voxel values.

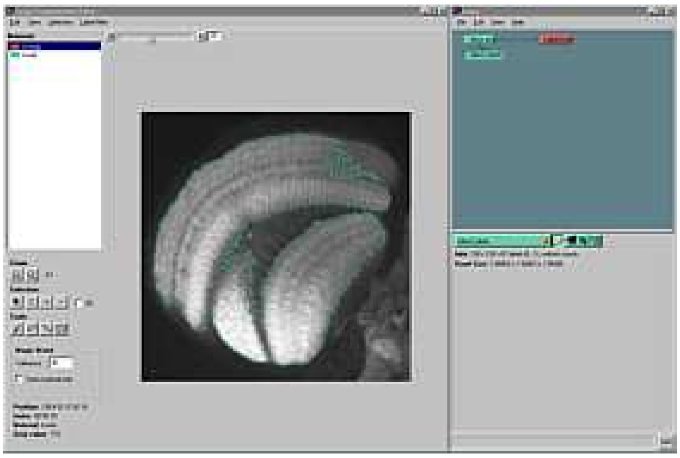


Figure 3.15: Data from confocal microscopy is segmented using Amira's image segmentation editor.

- Load the *lobus.am* data record from the directory *data/tutorials*.
- Attach a *LabelVoxel* module to the data icon and select it.
- Type 85 into the text field of port *Exterior-Inside*. You may also determine some other threshold that separates exterior and interior as described in the tutorial on Scalar Field Visualization.
- Choose the *remove couch* option from port *Options*.
- Press the *DoIt* button of port *Action*.

By this procedure each voxel having a value lower than the threshold is assigned to *Exterior* and each voxels whose value is greater than or equal to the threshold is assigned to *Interior*. This may, however, cause artifacts that are not part of the object, but have voxel values above the threshold, being assigned to the interior. This can be suppressed by setting the *remove couch* option which assures that only the biggest coherent area will be labeled as the interior and all other voxels are assigned to the exterior.

After pressing the *DoIt* button a new data object is computed and its icon appears in the Object Pool. The data object is denoted *lobus.Labels*. It is of type *LabelField*, represents a cubic grid with the same dimensions as *lobus.am*, and contains an interior or exterior label for each voxel according to the segmentation result.

3.4.2 Refining Segmentation Results

You can visualize and manually modify a *LabelField* by using Amira's *image segmentation editor*. A more detailed description of this tool is contained in the User's Reference guide. Here, we use the image segmentation editor to smooth the data in order to get a nicer looking surface of the object.

- Select the *lobus.Labels* icon and click on the icon in the green title bar in the Working Area that shows a pencil.

In response the image segmentation editor is popped up.

- Change the *slice slider* in the upper part of the editor's window to slice 39.
- Choose a magnifiaction ratio of 4:1 by pushing the zoom-up button in the left part of editor's window.

The image segmentation editor shows the image data to be segmented (*lobus.am*) as well as green contours representing the borders between interior and exterior regions as contained in the *lobus.Labels* data object. As you can see, the borders are not so smooth and there are many little islands, bordered by green contours. This is what we want to improve now.

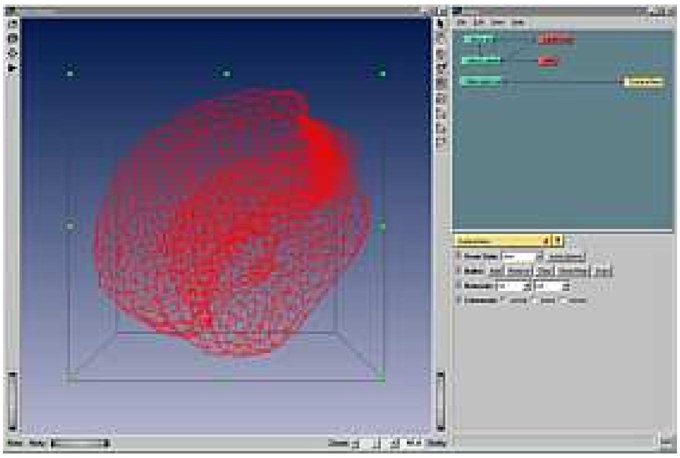


Figure 3.16: Surface representation of optical lobus as triangular grid

- Choose *Remove Islands* from the editor's *Label Filter* menu. In response, a little dialog window appears.
- In the dialog window select the *all slices* mode. Then press *Apply* in order to apply the filter in all slices. Note how the segmentation results become less noisy.
- To further clean up the image, choose *Smooth Labels* from the editor's *Label Filter* menu. Another dialog box appears.
- Select the *3D volume* mode and push the *Apply* button in order to execute the smoothing operation.
- To examine the results of the filter operations, browse through the label field slice by slice. In addition to the slice slider you may also use the cursor-up and cursor-down keys for this.
- Click onto the pencil icon in the Working Area to close the image segmentation editor.

3.4.3 Extracting Surfaces from Segmentation Results

Now we let Amira construct a triangular surface of the segmented object. A method called *generalized marching cubes* (GMC) is used for this purpose. This method can handle probabilities associated to the voxels. They were calculated when you applied the *smoothing filter* of the image editor and can also be computed by the *LabelVoxel* module. If the *sub-voxel accuracy* option is set, the GMC algorithm takes the probabilities into account to compute a smoother surface.

- Connect a *GMC* module to the *lobus.Labels* data.
- Make sure that the toggles *sub-voxel accuracy* and *add border* are set.
- Push *Triangulate* in the *Action* port.

The option *add border* ensures that the created surface is closed. A new data object *lobus.gmc-surf* is generated. Again, it is represented by a green icon in the Object Pool.

3.4.4 Simplifying the Surface

Usually the number of triangles created by the *GMC* module is far to high for subsequent operations. Thus, the number of triangles has to be reduced in a *surface simplification* step. In Amira a *Surface Simplification Editor* is provided for this purpose.

- Select the surface *lobus.gmc-surf*.
- Click on the triangle mesh icon (second from the right in the title bar) in the Working Area.
- Set the desired number of faces to 3500 in port *Simplify*.

- Turn on the *fast* toggle in port *Options*. This option disables some time-consuming intersection tests.
- Push the *Simplify* button in port *Action*.

The number of triangles is reduced to about 3500 now. The progess bar tells you how much of the simplification task has already been done.

To examine the simplified surface attach a *SurfaceView* module to the *lobus.gmc-surf* data object.

- Attach a *SurfaceView* module to the *lobus.gmc-surf* surface. More closely investigate the simplified geometry in the 3D Viewer.
- Before making the next step, deactivate the display of the *SurfaceView* module by hitting the *Viewer Toggle* located in the module's yellow title bar.

For an explanation of the buffer concept employed in the *SurfaceView* module see the short description of the *GridVolume* module below.

3.4.5 Generation of a Tetrahedral Grid

The last step is the generation of a *volumetric tetrahedral grid* from the surface. This means that the volume enclosed by the surface is filled with tetrahedra.

Because the computation of the tetrahedral grid is time consuming it may be performed as a *batch job*. You can then continue working with Amira while the job is running. However, in this case we want to compute the grid right inside Amira.

- Connect a *TetraGen* module to the *lobus.gmc-surf* surface.
- Clear the text field called *Grid*. If the text field is not empty it specifies a filename under which the resulting tetrahedral grid is stored automatically.
- Push the *Run now* button in port *Action*. A pop-up dialog appears asking you whether you really want to start the grid generation. Click *Continue* in order to proceed.

Once grid generation is running, the progress bar informs you about the number of triangles which already have been created. In some situations grid generation may fail, for example, if the input surface intersects itself. In this case try to create a simplified surface with the *fast* toggle of the *Surface Simplification Editor* disabled. You can also invoke the *Surface Editor* to interactively fix any intersections.

After the tetrahedral grid has been successfully created a new icon called *lobus.grid* will be put in the Object Pool. You can select this icon in order to see how many tetrahedra the created grid contains. If grid generation takes too long on your computer you may also load the pre-computed grid *lobus.grid* from the *data/tutorials* directory.

As the very last step you may want to have a look at the fruits of your work:

- Attach a *GridVolume* module to the *lobus.grid*.
- Select the *GridVolume* icon and push the *Add to* button of port *Buffer*.

The *GridVolume* module maintains an internal buffer and displays all tetrahedra stored in this buffer. By default the buffer is empty, but all tetrahedra are highlighted, i.e., they are displayed using a red wireframe representation. The *Add to* button causes the highlighted tetrahedra to be added to the buffer. You may easily visualize a subset of all tetrahedra using a 3D selection box or by drawing contours in the 3D viewer.

3.5 Registration and Warping Two 3D-Objects Using Landmarks

This is an advanced tutorial. You should be able to load files, interact with the *3D viewer*, and be familiar with the 2-viewer layout and the viewer toggles.

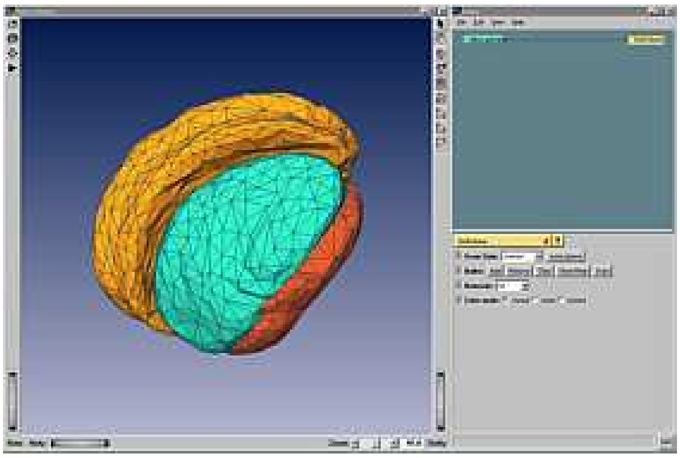


Figure 3.17: Volumetric representation of optical lobe as tetrahedral grid

We will transform two 3D-objects into each other by first setting landmarks on their surfaces and then defining a mapping between the landmark sets. As a result we shall see a rigid transformation and a warping which deforms one of the objects to match it with the other. The steps are:

1. Displaying Data Sets in Two Viewers.
2. Creating a Landmark Set.
3. Alignment via a Rigid Transformation.
4. Warping Two Image Volumes.

3.5.1 Displaying Data Sets in Two Viewers

The data we will be working with in this tutorial are of the same kind you have already seen before: Two optical lobes of a drosophila's brain.

- Load the two lobes by executing the script share/examples/landmark.hx.

This script will load two data sets called *lobus.am* and *lobus2.am*. In addition, two isosurface modules connected to each of the data sets will be created. In the viewer the two lobes are visualized by isosurfaces, the first in yellow and the second in blue. As we can see, the lobes are orientated differently. We want to look at each lobe in its own viewer.

- Choose *2 Viewers* from the *View Layout* menu.

You can see the two lobes in both viewers.

- Visualize the first lobe (yellow) in the first viewer and the second lobe (blue) in the second viewer by switching off the viewer toggles in the *isosurface* modules.

3.5.2 Creating a Landmark Set

Now, let us create a landmark set object.

- From the main window's *Edit* menu select *Create Landmarks*

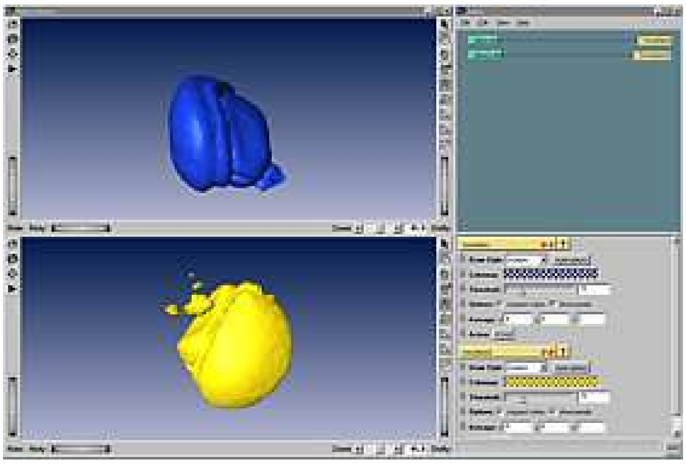


Figure 3.18: Two lobes visualized with isosurfaces in 2-viewer layout.

in order to create an empty set of landmarks. The new object will show up in the object pool. We are going to match two objects by means of corresponding landmarks, i.e., we actually have to produce two landmark sets. Make this number known to the *Landmarks* object.

- Select object *Landmarks*.
- Type *Landmarks setNumSets 2* in the console window. Instead of manually typing the name of the object you want to send commands to, e.g., *Landmarks*, you may simply press the tab key on the empty command line.
- Start the *Landmark Editor* by clicking on the button with the disk-shaped icon.

When starting the editor, a *LandmarkView* module is automatically created and connected to the *Landmarks* data object. As indicated on the info line, two empty landmark sets are availabe now. We use the editor to define some markers in both objects. For the following, it is useful to pin the three ports on the landmark editor. In order to do so, select the grey pin toggles left from the port's labels.

- Connect a second *LandmarkView* module to the *Landmarks* object.
- Select the first *LandmarkView* module and choose *Point Set*: *Point Set 1*.
- Shift-select the second *LandmarkView* module and choose *Point Set*: *Point Set 2*.
- Adjust the viewer toggles of the two display modules such that the first is visible in the first viewer and the second in the second viewer.

Before starting to set landmarks it is helpful to rotate the two lobes in their viewers such that they are approximately aligned. This will make it easier to locate corresponding features in the two objects and to select reasonable positions for landmarks.

- Rotate the two lobes to align them roughly.

Now, we are ready to define and set corresponding landmarks. Select the *LandmarkSet* object if necessary and choose the option

Edit mode: *Add*.

To set corresponding landmarks simply click with the left mouse button anywhere on the surface in the first viewer first and click on the surface in the second viewer subsequently. The landmarks are visualized as small spheres, the first landmark in yellow and the corresponding landmark in blue. Make sure to always set the first landmark on the first (yellow) surface and the second landmark on the second (blue) surface!!

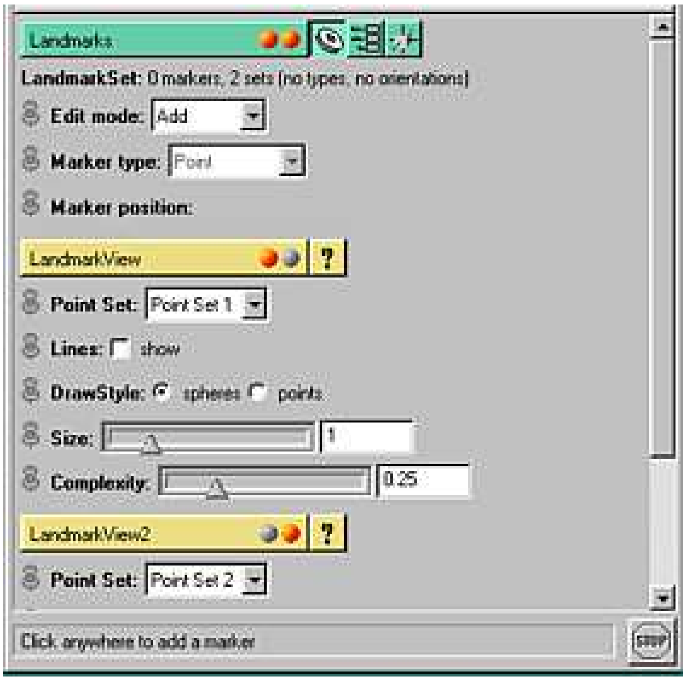


Figure 3.19: The image shows how the viewer toggles and *Point Set* ports should be set.

If you want to change the position of an existing landmark set

Edit mode: *Move*

and select the respective landmark (blue or yellow) by clicking on it with the left mouse button. Then just click at the desired position.

You can also delete existing landmarks by setting

Edit mode: *Remove*

and clicking on the respective landmark. Both corresponding landmarks (blue and yellow) will be removed, no matter which one was selected.

You should now be able to create several landmarks. You may want to change the view of the objects to set landmarks on the back. In case you have problems to define landmarks you may use an existing set of landmarks by loading the file `landmarkSet` from the directory `data/tutorials`. Once landmarks have been created, the next step is to transform the two objects into each other.

3.5.3 Registration via a Rigid Transformation

To register one object to the other connect a *LandmarkWarp* module to the *LandmarkSet* object by clicking with the right mouse button on the *LandmarkSet* icon in the object pool and selecting *Compute LandmarkWarp*.

We want to perform an alignment of the first lobe to the second. Therefore the *LandmarkWarp* module must be connected to the image data of the first lobe (use the right mouse button in the tiny rectangle of the *LandmarkWarp* icon).

The *LandmarkWarp* module is able to perform several transformations. We start with a purely rigid transformation to match the corresponding landmarks as good as possible by perfoming only rotations and translations of the first object. To do that choose

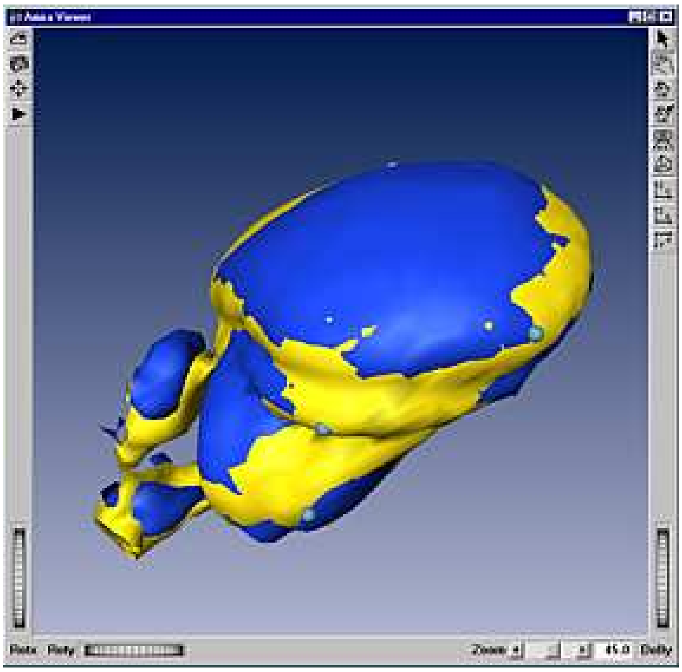


Figure 3.20: Result of landmark-based elastic warping using the Bookstein method.

Method: *Rigid*

in the *LandmarkWarp* module and make sure that *Direction* is set to

Direction: $1 \rightarrow 2$.

Then press *DoIt* to start the computation. The module creates a new data object named *lobus.Warped*. To visualize the result, connect the isosurface that was initially connected to the data of the first lobe to the result, select it and press the *DoIt* button. In order to compare the result with the second lobe, adjust the viewer toggle of its *Isosurface* module to display it in the first viewer. You should see that the result of the transformation fits the second object quite well.

3.5.4 Warping Two Image Volumes

Using the rigid transformation the object will not be deformed. To perform a deformation and obtain a better fit we can use another transformation method of the *LandmarkWarp* module. Select the latter and choose

Method: *Bookstein* and press the *DoIt* button.

To visualize this result, the isosurface has to be recomputed. Having done that you can see both the deformed and the second lobe in the first viewer. To merely see the resulting deformation in the first viewer, switch off the viewer toggle of the second lobe's *Isosurface* module. Only a little deformation will be seen because the two original objects were rather similar. Using more different data sets results in larger deformations.

We hope you have had some fun with our tutorials, got to know the basic features of Amira and learned to use them. For details and more information about other features see chapter 3, the *Program Description*.

Chapter 4

Program Description

This chapter contains a detailed description of Amira interface components and data types. No in-depth knowledge of Amira is required to understand the following sections, but it is a good idea to have a look at one of the tutorials contained in Chapter 3, particularly the very first one described in Section 3.1 (Getting Started).

4.1 Interface Components

In this section the following interface components are described:

- *File Menu*, *Edit Menu*, *View Menu*, *Help Menu*
- *Main Window*, *Viewer Window*, *Console Window*
- *File Dialog*, *Job Dialog*, *Snapshot Dialog*

4.1.1 File Menu

The file menu lets you load and save data objects as well as Amira network scripts. In addition, it gives you access to Amira's job dialog and allows you to quit the program. In the following text, all menu entries are discussed separately.

Load

The *Load* button activates Amira's *file dialog* and lets you import data sets stored in a file. Most file formats supported by Amira will be recognized automatically via the file header or the file name extension. For each file, the file dialog will display its format. If you try to load a file for which the format couldn't be detected automatically, an additional dialog pops up asking you to select the format manually. You may also manually set the file format for any file by selecting the file, activating the file dialog's popup menu using the right mouse button, and then choosing the *Format* option.

A list of all *supported file formats* is contained in the reference manual. Hints on how to import your own data sets are given in Section 5.1.

If you select multiple files in the file dialog, all of them will be loaded, provided all of them are stored in the same format. 2D images stored in separate files usually will be combined into a single 3D data object. On the other hand, there are some file formats which cause multiple data objects to be created. Finally, you can also import and execute Amira network scripts using the *Load* button.

Save Data

The *Save Data* button allows you to save a single modified data object again using the same filename previously chosen under *Save Data As*. The button will only be active if the data object to be saved is selected and if this data object already has been saved using *Save Data As*. A common application of the *Save* button is to store intermediate results during manual segmentation in Amira's *image segmentation editor*.

Save Data As

This button lets you write a data object into a file. To do so you must first select the data object (click on the corresponding green data icon). Then choose *Save Data As* to activate Amira's *file dialog*. The file dialog presents a list of all formats suitable for saving that data object. Choose the one you like and press *OK*. Note that you must specify the complete file name including the suffix. Amira will not automatically add a suffix to the file name. However, it will update the suffix whenever you select a new format from the file format list. Also, Amira will ask you before it overwrites an existing file.

Some file formats create multiple files for a single data object. For example, each slice of a 3D image data set might be saved as a separate raster file. In this case, the file name may contain a sequence of hashmarks. This sequence will be replaced by consecutive numbers formatted with leading zeros.

If no file format at all has been registered for a certain type of data object, the *Save as* button will be disabled. It will also be disabled if more than one data object is selected in the object pool.

Save Network

You may save the complete network of icons and connections shown in the object pool using the *Save Network* button. The file dialog will appear and you will carry out the same steps as for single data files. This feature is useful for resuming work at a point where it was left in a previous Amira run. When loading the network script, all data objects and modules will be restored automatically.

Note that usually all data objects must have been stored in a file in order to be able to save the network. If this is not the case, a dialog is popped up listing all the data objects that still need to be saved. In the dialog you can also specify that all required data objects should be saved automatically in the same directory where the network script will be written (auto-save feature).

Jobs

This button brings up Amira's *job dialog* which is used to control the execution of batch jobs running in the background. For example, tetrahedral grids can be generated in a batch job (see module *TetraGen*). However, for most users the batch queue will be of minor interest.

Quit

This button terminates Amira. The current network configuration will be lost unless you explicitly save it using *Save Network*.

4.1.2 Edit Menu

The *Edit* menu provides control on the visibility of object icons and lets you delete and create objects. Depending on how many icons are selected in the Object Pool, some menu options might be disabled.

Hide

The *Hide* button hides all currently selected objects. The object's icons are removed from the Object Pool but the objects themselves are retained. You get the same effect by pressing the `[Alt] [d]` keys. Hidden objects can be made visible again using the *Show* or *Show All* option.

Remove

The *Remove* button deletes all selected objects and removes the corresponding icons from the Object Pool. You can get the same effect by pressing the `[Alt] [x]` keys. If you want to reuse a data object later, be sure to save it in a file before deleting it.

Duplicate

The *Duplicate* button creates copies of all selected data objects. For each copy a new data icon is put in the Object Pool. The name of a duplicated data object differs from the original one by one or more appended digits. The duplicate option will be unavailable if you have selected icons that do not represent data objects.

Show

The *Show* button lets you select a currently hidden object for having its icon displayed in the Object Pool. Among the hidden objects there are usually some colormaps which are loaded at start-up. This option will be unavailable if there are no hidden objects.

Show All

The *Show All* button makes all currently hidden objects visible, displaying their icons in the Object Pool. This option will be unavailable if there are no hidden objects.

Remove All

The *RemoveAll* button deletes all currently visible icons and the associated objects from the Object Pool. A pre-loaded colormap that is currently visible is also deleted, but all hidden objects are retained.

Database

The *Database* button activates the *Parameter Editor* and allows you to manipulate Amira's global material database. The material database contains various material parameters of which most are required for certain numerical simulations to be performed on segmented volumetric grids representing specific physical objects (e.g. E-field simulations on tetrahedral patient models in medical applications) such that parameter values are fixed within segments but vary across segments, notably object colors, density, heat capacity. Each data object containg materials, e.g. a *label field* or a *tetrahedral grid*, may have its own copy of these values. However, if a value is not found locally it is looked up in the global material database. The material database is stored in the file `share/materials/database.hm` located in the Amira installation directory. If you want to make permanent changes in the database, you must edit this file.

Create

The *Create* button lets you create modules or data objects that cannot be accessed via the popup menu of any other object. For example, it allows you to create a procedurally defined scalar field from scratch. The icon of a newly created object will not be connected to any other object in the Object Pool. In order to establish connections later on, use the popup menu over the small rectangular connection area of the object's icon.

4.1.3 View Menu

The *View* menu provides control over several *Viewer* options affecting the display independent of the *Viewer* input.

Layout

The *Layout* button lets you select between one, two, or four 3D viewers. All viewers will be placed inside a common window using a default layout. If you want to create an additional viewer in a separate window, choose *Extra Viewer*. You may create even more viewers using the Tcl command `viewer <n> show`. Starting from *n=4*, viewers will be placed in separate windows.

Background

The *Background* button lets you select between the three background styles *Uniform*, *Gradient*, *Checkerboard*. The effect becomes immediately visible, independent of the scene being displayed. Two different background colors are used by the gradient and the checkerboard background. The option *Swap colors* lets you interchange these colors. The option *Reset colors* resets the default background colors (which are a dark blue and a very light blue).

In order to change the background color via the command interface use the viewer commands `viewer <n> setBackgroundColor` and `viewer <n> setBackgroundColor2`. The command interface also allows you to place an arbitrary raster image into the viewer background (see Section 4.1.6, viewer commands).

Transparency

The *Transparency* button controls the way of calculating pixel values with respect to object transparencies during the rendering process.

- *Screen Door:* Transparent surfaces are approximated using a stipple pattern.
- *Add:* Additive alpha blending.
- *Add Delay:* Additive alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Add Sorted:* Like *Add Delay*, but transparent objects are sorted by distances of bounding box centers from the camera and are rendered in back to front order.
- *Blend:* Multiplicative alpha blending.
- *Blend Delay:* Multiplicative alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Blend Sorted:* Like *Blend Delay*, but transparent objects are sorted by distances of bounding box centers from the camera and are rendered in back to front order.

Lights

The *Lights* menu lets you activate different light settings for the 3D viewer. By default, the viewer simply uses a single headlight, i.e., a directional light pointing in the same direction as the camera is looking. This setting may be restored using the *Standard* option. Two more options, *Blue-Red* and *Fancy*, let you activate other light settings.

At any time, additional lights can be created via the *Create light* option. Except for the viewer's default headlight, all lights are represented by little blue icons in the Object Pool, just like ordinary data objects or modules. Any hidden light icons are listed at the end of the *Lights* menu (and not in the *Edit Show* submenu). In order to make all hidden light icons visible, use the *Show all icons* option. For more information about *lights*, please refer to the Reference Section of this manual.

Fog

The *Fog* button introduces a fog effect into the displayed scene and controls how opacity increases with distance from the camera. The fog effect will only be seen on a *uniform* background. More fine tuning is provided by the *fogRange* Viewer command.

- *None:* No fog effect (default).
- *Haze:* Linear increase in opacity with distance.
- *Fog:* Exponential increase in opacity with distance.
- *Smoke:* Exponential squared increase in opacity with distance.

Axis

The *Axis* button creates an *Axis* module named *GlobalAxis* which immediately displays a coordinate frame in the viewer window. This button is a toggle, so clicking on it again deletes the *GlobalAxis* module and removes the coordinate frame from the viewer window. The axes will be centered at the origin of the world coordinate system. You may also create local axes by selecting the appropriate entry from a data object's popup menu.

Fading effect

The *Fading effect* toggle lets you switch on a fading effect which is applied to all kinds of scene movements. Before a new image is rendered only a certain fraction of the background will be cleared. In this way older images remain visible until they fade out after a while. Note that this mode requires single buffer rendering, and therefore, flickering may be visible in some cases.

4.1.4 Online Help

Amira user's documentation is available online. You can access it via the *User's Guide* entry of the main window's *Help* menu. The user's guide contains some introductory chapters, as well as a reference part containing documentation for specific

- modules,
- data types,
- editors,
- file formats,
- and other components.

You may access the documentation of any such object via a separate index page accesible from the home page of the online help browser. Amira modules also provide a question mark button in the working area. Pressing this button directly pops up the help browser for the particular module.

Going through the online documents is similar to text handling within any other hypertext browser. In fact, the documentation is stored in HTML format and can be read with a standard web browser as well. Some specially marked (colored and underlined) text items allow you to jump quickly to related or referenced topics, where blue items point to unread sections, and red items to already viewed sections. Use the *Backward* and *Forward* buttons to scroll in the document history and *Home* to move to the first page.

Searching the online documentation
The online help browser provides a very simple interface for a full text search. For example, if you are looking for information about the *surface editor*, type these two words into the text field in the upper part of the help window. Then, press the search button in order to perform the search. If you want to look for multiple words, you must prepend them with a plus sign, e.g. *surface +editor*.

Running demo scripts
In the demo section of the on-line manual you can easily start any demonstration just by clicking on the marked text. The script will be loaded and executed immediately. You may interrupt running demo scripts by using the stop button in the lower right of the Amira main window.

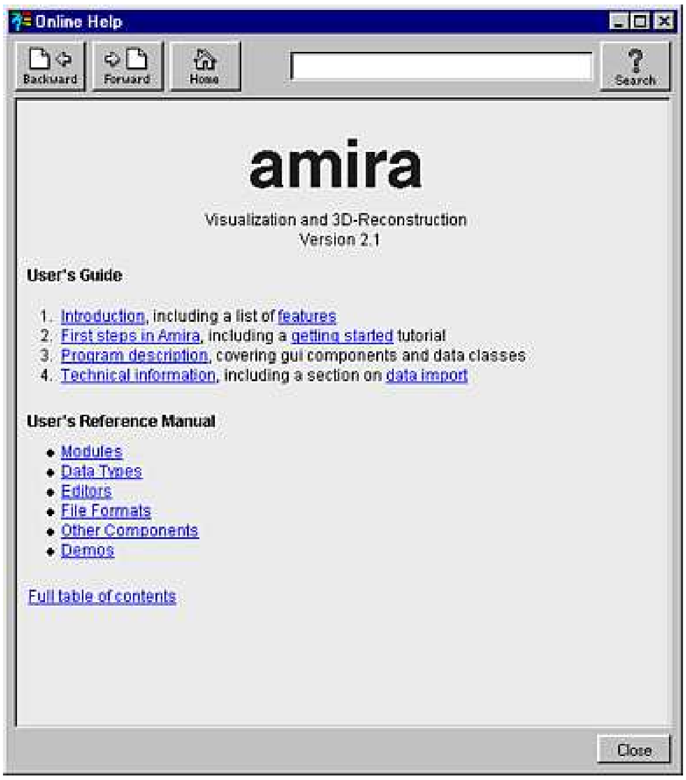


Figure 4.1: Amira's help window.

Commands

`help`
Makes the help dialog appear and loads the home page of the online help.

`help getFontName`
Returns the name of the font of the browser.

`help setFontName`
Sets the font of browser.

`help getFontSize`
Returns the size of the font of the browser.

`help setFontSize`
Sets the size of the font of the browser. In order to permanently change the font size, put this command in the `.Amira` file in your home directory or in an `Amira.init` file in the current working directory. For details see Section 5.4.

`help load` *`file.html`*
Load the specified hypertext document in the file browser. Note that only a subset of HTML is supported.

`help reload`
Reload the current document.

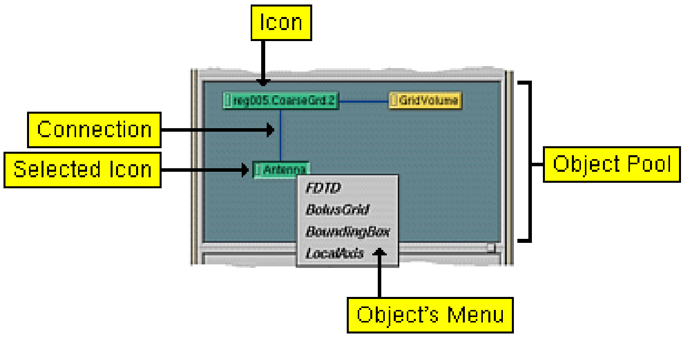


Figure 4.2: The object pool contains data objects and module icons.

4.1.5 Main Window

Amira's main window consists of two components, the *Object Pool* in the upper part of the window and the *Working Area* in the lower part. The Object Pool contains icons representing data objects and modules currently in use as well as lines connecting icons indicating dependencies between objects and modules. The Working Area is the place where the user interface of selected objects is displayed. Typically, the interface consists of buttons and sliders arranged in *Ports*. They can be thought of as ports because the user can pass information to a module solely through them.

Object Pool

Once a data object has been loaded or a module has been created it will be represented by an icon in the Object Pool. Some objects, especially colormaps, may not be visible here. Such hidden objects are listed in the *Edit Show* menu of the main window. Selecting an object from this menu causes the corresponding icon to be made visible in the Object Pool.

Icon colors indicate different object types. Data objects are shown in green, computational modules in red, and visualization modules in yellow. Orange icons represent visualization modules of *slicing* type. Such modules may be used to clip the graphical output of any other module. Connections between data objects and processing modules, shown as blue lines, represent the flow of data. You may connect or disconnect objects by picking and dragging a blue line between object icons.

As you might expect, not all types of processing modules are applicable to all kinds of data objects. If you click on an icon with the right mouse button, a menu pops up that shows all types of modules that can be connected to that object. Selecting one of them will automatically create an instance of that module type and connect it to the data object. A new icon and a connecting line will appear in response. This way you can set up a more or less complex network that represents the computational steps required to carry out a specific visualization task and is indeed used to trigger them.

If you look closer at an object's icon you will notice a tiny rectangle on its left. If you click on it with the right mouse button a menu pops up that shows all connection ports of that object. As mentioned above, for most objects the required connections are automatically established on creation. However, in order to set up optional connections you must use the connection popup menu. For example, you may attach an optional scalar field to an Isosurface module in order to let its values on an existing isosurface be encoded by colors. The colormap used for pseudo-coloring is specified by another connection port.

Once you have selected an entry from the connection popup menu, you can choose a new input object for that port. In order to do so, click on the input object's icon in the Object Pool. The blue connection line will become yellow if the connection port can be connected to the chosen object. In order to disconnect an input object click on the icon of the module the port belongs to. Data objects possess a special connection port called *Master*. This port refers to a computational module or editor the data object is attached to. It indicates that the computational module or editor controls the data object, i.e., that it may modify its contents.

Each object has an associated control panel containing buttons and sliders for setting or changing additional parameters of the object. The control panel becomes visible once the object has been selected, i.e. by clicking on its icon with the right mouse button. In order to select multiple objects you must shift-click the corresponding icons. Clicking on the icon of a selected object deselects it again. Clicking somewhere on the background of the Object Pool causes all selected objects to be deselected. An icon may be dragged around in the Object Pool by clicking on it and moving the mouse pointer while holding down the mouse button.

Working Area

Once an object has been selected, its input controls will be displayed in the Working Area below the Object Pool. Each object has a specific set of controllable parameters or options. These are described in detail for each module in the *index section* of the reference manual. Computational modules and visualization modules also provide a question mark button which lets you access the documentation of that module directly.

At the top of an object's control panel its name is displayed and a number of additional control buttons are provided. All objects have one or more orange viewer buttons for each 3D viewer. These buttons control whether any graphical output of an object is displayed in a particular viewer or not. For example, if you have two viewers and two isosurface modules you may want to display one isosurface in each viewer.

Display modules of *slicing* type (orange ones) provide a clip button. Clicking this button will cause the graphical output of any other module to be clipped by that slice. Clipping does not affect modules with hidden geometry or modules that are created after the clip button has been pressed.

Data objects provide a number of additional *editor* buttons. Editors are used in order to modify the contents of a data object interactively. For example, you can perform manual segmentation of 3D image data by editing *label fields* using the *image segmentation editor*. Some editors display their controls in the working area like all other objects, while others use a separate dialog window that allows you to perform object manipulations.

As already mentioned, specific input controls of an object or a module are organized in *Ports*. Each port has a pin button on its left. If a port is pinned it will still be visible even when the object is deselected. The ports are composed of various widgets that reflect an operational meaning, e.g. a value is entered by a slider, a state is set by radio buttons, a binary choice is presented as a toggle button. The control elements have a uniform layout and are divided into several basic types. A description of the basic port types is contained in the *component index* section of the User's Reference Manual.

4.1.6 Viewer Window

The 3D viewer plays a central role in Amira. Here all geometric objects are shown in 3D space. The 3D viewer offers powerful and fast interaction techniques. It can be regarded as a virtual camera which can be moved to an arbitrary position within the 3D scene. The left mouse button is used to change the view direction by means of a virtual trackball. The middle mouse button is used for panning, while the left and the middle mouse button pressed together allow you to zoom objects.

Sometimes you need to manipulate objects directly in the 3D viewer. For example, this technique, called 3D interaction, is used by the *transform editor*. The editor provides special draggers that can be picked and translated or rotated in order to specify the transformation of a data object. Before you can interact with these draggers, you must switch the viewer into *interaction mode*. This is done by clicking on the arrow button in the upper right corner. If the viewer is in interaction mode, the mouse cursor will be an arrow instead of a hand symbol. You can use the `[ESC]` key in order to quickly switch between interaction mode and viewing mode. If the viewer is in interaction mode, use the `[Alt]` key to temporarily switch to viewing mode.

More than one viewer can be active at a time. Standard screen layouts with one, two, or four viewers can be selected via the *view menu*. Additional viewers can be created using the Tcl command `viewer <n> show`, where `<n>` is an integer number between 0 and 15. While viewers 0 to 3 will be placed in a common

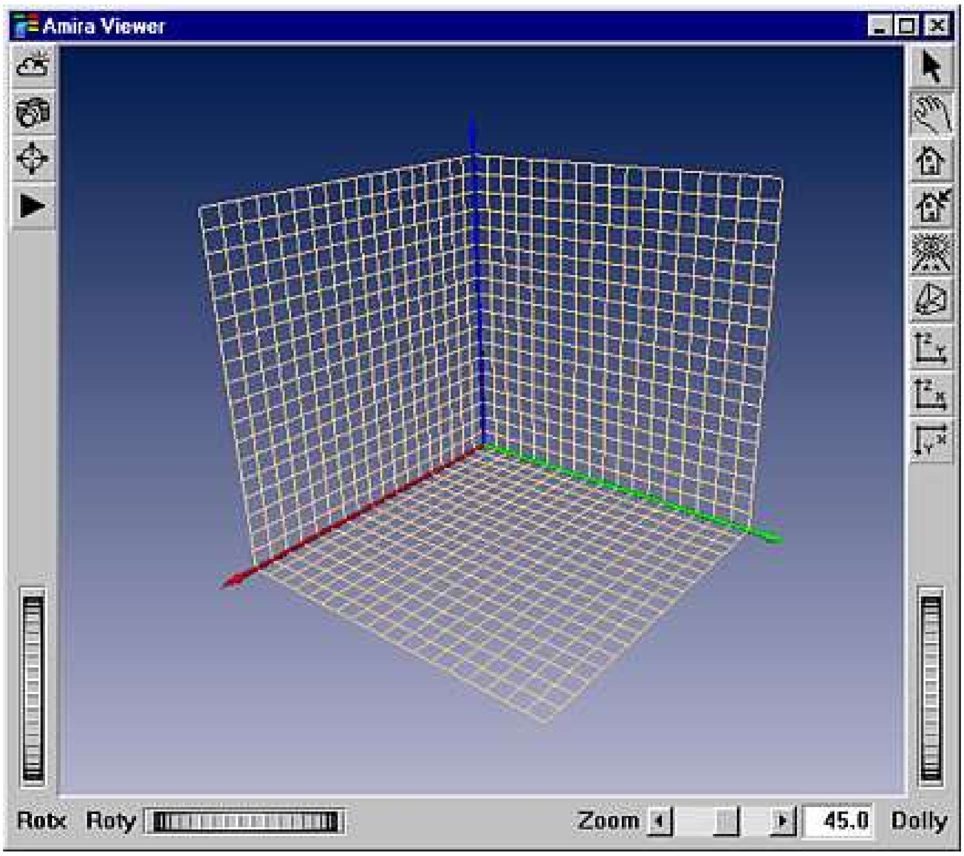


Figure 4.3: Amira's viewer window provides a virtual trackball for easy navigation. The decoration frame contains several controls, allowing you for example to switch between viewing mode and interaction mode, to choose certain orienations, or to take snapshots.

panel window, viewers 4 to 15 will create their own top-level window. For more specific control, the viewer provides an extensive command set, which is documented *below*.

The decoration of the viewer window provides several buttons and controls, see Figure 4.3. The precise meaning of these controls is described below.

- *Edit Background Color:* Pops up a color editor which allows you to change the main background color of the viewer. In order to change the secondary background color which is used if a non-uniform background pattern has been selected, use the *Swap colors* option of the main winodw's *View Background* menu (see Section 4.1.3). Alternatively, you may use the *viewer command* `setBackgroundColor2`.
- *Snapshot:* Takes a snapshot of the current rendering area and saves it in a file. The filename as well as the desired output format have to be entered through the *snapshot dialog*. Snapshots may also be taken using the *viewer command* `snapshot`.
- *Seek:* Pressing the seek button and then clicking on an arbitrary object in the scene causes the object to be moved into the center of the viewer window. Moreover, the camera will be oriented parallel to the normal direction at the selected point. Seeking mode may also be activated by pressing the `[S]` key in the viewer window.
- *Animate Camera:* If this button is pressed, the first camera path object found in the *Object Pool* will be used for animation. No action is performed if no camera path is found. In order to create a camera path, select *KeyframeCamPath* from the *Edit Create* menu and invoke the *camera path editor*.
- *Pick:* Switches the viewer into interaction mode. You can also use the `[ESC]` key to toggle between viewing mode and interaction mode.
- *View:* Switches the viewer into viewing mode. You can also use the `[ESC]` key to toggle between

interaction mode and viewing mode.

- *Home:* Resets camera to the home position.
- *Set Home:* Sets the current position as the new home position.
- *View All:* Repositions the camera so that all objects become visible. The orientation of the camera will not be changed.
- *Perspective/Ortho:* Toggles between a perspective and an orthographic camera. By default, a perspective camera is used. You may want to use an orthographic camera in order to measure distances or to exactly align objects in 3D space.
- *YZ-, XZ- and XY-Views:* Adjusts the camera according to the specified viewing direction. The viewing direction is parallel to the coordinate axis perpendicular to the specified coordinate plane. Medical doctors are used to viewing series of tomographic images parallel to the XY-plane with the y-axis pointing downwards. This convention is followed by the XY-button. The opposite view direction is used if the Shift key is pressed.
- *Rotate:* Rotates the camera around the current view direction. By default, a clockwise rotation of one degree is performed. If the Shift-key is pressed while clicking, a 90 degree rotation is done. If the Ctrl-key is pressed, the rotation will be counterclockwise.

Viewer Commands

Commands to a viewer can be entered in the console window. The syntax is

`viewer [<number>] command`,

where `<number>` specifies the viewer being addressed. The value 0 refers to the main viewer and may be omitted for convenience.

Commands

`viewer [<number>] snapshot [-offscreen [<width> <height>]] <filename>`

This command takes a snapshot of the current scene and saves it under the specific filename. The image format will be automatically determined by the extension of the file name. The list of available formats includes: TIFF (`.tif,.tiff`), SGI-RGB (`.rgb,.sgi,.bw`), JPEG (`.jpg,.jpeg`), PNM (`.pgm,.ppm`), BMP (`.bmp`), PNG (`.png`), and Encapsulated PostScript (`.eps`). If the viewer number is not given, the snapshot is taken from all viewers, if you have selected the 2 or 4 viewer layout from the *View menu.*

If the `-offscreen` option is specified, offscreen rendering with a maximum size of 2048x2048 is used. In this case the viewer number is required even if viewer 0 is addressed. If the width and height is not specified explicitly, the size of the image is the current size of the viewer.

Caution: If you have more than one transparent object visible in the viewer and you want to use offscreen rendering set the transparency mode to *Blend Delayed* and check to see if all objects are rendered properly prior to taking a snapshot.

`viewer [<number>] setPosition <x> <y>`

Sets the position of the viewer window relative to the upper left corner of the screen. The allowed range of *x* and *y* values is defined by the resolution of the screen. Typically this will be 1280 x 1024.

`viewer [<number>] getPosition`

Returns the position of the viewer.

`viewer [<number>] setSize <width> <height>`

Sets the size of the viewer window. The visible part of the 3D window exactly matches the given size if decorations are switched off (see below).

`viewer [<number>] setVideoFormat pal|ntsc`
Sets the size of the viewer window according to PAL 601 or NTSC 601 resolution, i.e., 720x576 pixels or 720x486 pixels. The current setting of the decoration is taken into account.

`viewer [<number>] setVideoFrame <state>`
If *state* is 1, a frame is displayed in the overlay plane of the viewer. This frame depicts the area where images recorded to video are safely shown on video players. Setting *state* to 0 switches the frame off. Note: Objects displayed in the overlay planes are not saved to file with the `snapshot` command (see above).

`viewer [<number>] getSize`
Returns the size of the viewer window.

`viewer [<number>] getCamera`
This command returns the current camera settings, i.e., camera position, camera orientation, and camera focal distance. The values are returned as Amira commands, which can be executed in order to restore the camera settings.

`viewer [<number>] setCameraPosition <x> <y> <z>`
Defines the position of the camera in world coordinates.

`viewer [<number>] setCameraOrientation <x> <y> <z> <a>`
Defines the orientation of the camera. By default, the camera looks in negative z-direction with the y-axis pointing upwards. Any other orientation may be specified as a rotation relative to the default direction. The rotation is specified by a rotation axis *x y z* followed by a rotation angle *a* (in radians).

`viewer [<number>] setCameraFocalDistance`
Defines the camera's focal distance. The focal distance is used to compute the center around which the scene is rotated in interactive viewing mode.

`viewer [<number>] setAutoRedraw <state>`
If *state* is 0, the auto redraw mode is switched off. In this case the image displayed in the viewer window will not be updated, unless a redraw command is sent. If *state* is 1, the auto redraw mode is switched on again. In a script it might be useful to disable the auto redraw mode temporarily.

`viewer [<number>] redraw`
This command forces the current scene to be redrawn. An explicit *redraw* is only necessary if the auto redraw mode has been disabled.

`viewer [<number>] rotate <degrees>`
Rotate the camera around the current up-direction.

`viewer [<number>] rotateVD <degrees>`
Rotate the camera around the current view-direction. This does the same as the rotate button in the user interface.

`viewer [<number>] decoration <state>`
The decoration is an extended window frame that serves as a user-interface. It contains buttons and thumb wheels for adjusting the view or switching between interaction and viewing mode. The `decoration` command can be used to show or hide the decoration area. Hiding the decoration is useful when multiple viewers are open and the size of a single viewing window is rather small.

`viewer [<number>] 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`.

`viewer [<number>] setBackgroundColor2 <r> <g> <b>`
Sets the secondary background color which is used by non-uniform background modes.

`viewer setBackgroundMode <mode>`
Allows you to specify different background patterns. If mode is set to 0 a uniform background will be displayed. Mode 1 denotes a gradient background, while mode 2 causes a checkerboard pattern to be displayed. This might help to understand the shape of transparent objects.

`viewer setBackgroundImage <imagefile>`
This command allows you to place an arbitrary raster image into the center of the viewer's background. The image must not be larger than the viewer window itself. The format of the image file will be detected automatically by looking at the file name extension. All formats mention for the `snapshot` command are supported except of Encapsulated Postscript.

`viewer [<number>] saveScene <filename>`
Saves all of the geometry displayed in a viewer in Open Inventor 3D graphics format. Warning: Since many Amira modules use custom Inventor nodes, the scene usually can not be displayed correctly in external programs like *ivview*.

`viewer [<number>] viewAll`
Resets the camera so that the whole scene becomes visible. This method is called automatically for the first object being displayed in a viewer.

`viewer [<number>] show`
This command opens the specified viewer and ensures that the viewer window is displayed on top of all other windows on the screen.

`viewer [<number>] hide`
This command closes the specified viewer.

`viewer [<number>] setTransparencyType <type>`
This command defines the strategy used for rendering transparent objects. The argument *type* may be a number between 0 and 6, corresponding to the entries *Screen Door*, *Add*, *Add Delay*, *Add Sorted*, *Blend*, *Blend Delay*, and *Blend Sorted* as described for the *View menu*.

Most accurate results are obtained using mode 6, which is the default. However, some objects may not be recognized correctly as being transparent. In this case you may switch them off and on again in order to force them to be rendered last. Also, if lines are to be rendered on a transparent background problems may occur. In this case, you may use transparency mode 4 and ensure the correct rendering order manually.

`viewer [<number>] getTransparencyType`
This command returns the current transparency type as a number in the range 0...6. The meaning of this number is the same as in `setTransparencyType`.

`viewer [<number>] fogRange <min> <max>`
Sets a range of attenuation for the fog affect that can be introduced into a viewer scene by the *View menu*. The default range is $[0, 1]$. Values within this range correspond to distances of scene points from the camera, such that points nearest to the camera have value zero and those farthest away have value one. Restricting the range of attenuation means that attenuation will start at points where the specified minimum is attained and reach its maximum at points where the specified maximum is attained. Maximum attenuation by fog is equivalent to invisibility, thus all points beyond that maximum will appear as background.

4.1.7 Console Window

The *Console* window is the user interface to Amira's advanced controlling features. It serves two purposes. First, it gives you some feedback on what is currently going on. Such feedback messages include warnings,

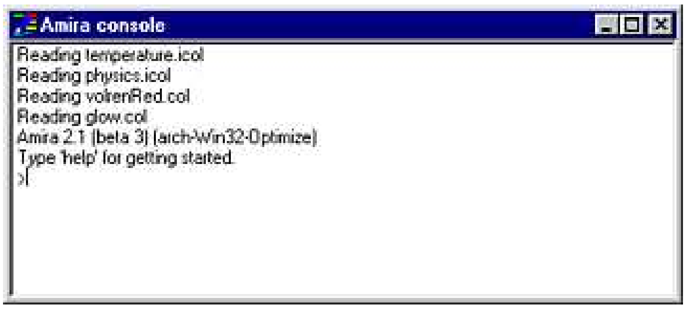


Figure 4.4: Amira's console window displays info messages and lets you enter Tcl-commands.

error indications and notes on problems as well as information on results. Second, it provides a command line interface, i.e., a shell where Amira commands can be entered.

Amira's console commands are based on *Tcl* (Tool Command Language). This means you can write command scripts using Tcl with Amira specific extensions. For a complete description of Tcl and its control structures we refer to the text book *Tcl and the Tk Toolkit* by *John K. Ousterhout*, the creator of Tcl. Like many other books about Tcl this also covers the Tk GUI toolkit. Note that Tk is not used in Amira.

Concerning Amira extensions there are some fixed command names like `load` or `viewer` and temporary ones which are associated to modules and data objects. The general format of the temporary commands is:

object command-word target-value

In particular, when creating an *OrthoSlice* module, a command with the same name is introduced, e.g. *OrthoSlice* or *OrthoSlice2* or *OrthoSlice3* ... depending on how many instances of *OrthoSlice* modules already exist, and removing the module from the *Object Pool* also removes the command. As long as the module exists, all its associated command words are available. By these commands you may set the more specialized functional options that the module provides. The command interface of a particular module is described in the User's Reference Guide. A help function that displays all functions provided by a module is also available. You activate it by entering the module command with the `help` option.

Because of the object-oriented concepts underlying Amira's design you should think of a module or data object as an instance of an Amira class - e.g. there is a class *HxOrthoSlice* of which *OrthoSlice* and *OrthoSlice2* are instances - and of commands as associated to classes. Due to Amira's class hierarchy all commands associated to a particular class are also associated to all of its sub-classes, i.e., a module command is also available for all other modules that are instances of a sub-class of that module's class, and the same applies to data classes. Moreover, there is a class *HxObject* with sub-classes *HxModule* and *HxData* from which all modules respectively data classes are derived. Therefore the commands provided by *HxObject*, listed below as *Object* commands, are available for all modules and all data objects.

To execute a single console command just type in its name and arguments and press 'Enter'. If you select an object and then press the `[TAB]` key on the empty command line in the console window then the name of the object will be automatically inserted. Often, this saves a lot of typing. The *Console* supports a command history mechanism. Use 'up arrow' and 'down arrow' to scroll up and down in the history list. To execute a file containing many Tcl commands use '`load` *filename*' or load the script file via Amira's file dialog from the file menu. For advanced script examples take a look at Amira's demo files.

Commands

`help`
Shows the *help* window.

`load` *command-file*
Loads and executes the command file. The same command is used to load ordinary data objects into Amira.

`quit`
Quit the application.

Object `allPorts`
Lists all ports of *Object*.

Object `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.

Object `deselect`
Deselects an object, i.e., the object's widgets vanish from the Working Area.

Object `destroy`
The object is removed, as well as certain dependent objects.

Object `fire`
Calls the internal update and compute methods of the object.

Object `getIconPosition`
Returns the position of the object's icon in the Object Pool display area.

Object `getTypeId`
Returns the type of the object.

Object `help`
Displays all commands specific to that object.

Object `hideIcon`
Hides the icon of a specific object.

Object `select`
Selects an object, i.e., the object's widgets appear on the Working Area.

Object `setIconPosition` *xPos yPos*
Sets the position of the object's icon in the Object Pool display area to *xPos yPos*.

Object `setLabel` *newlabel*
Changes the name of the object to *newlabel*. If already some other object with the same name exists, *newlabel* will be automatically modified.

Object `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.

Object `showIcon`
Shows a specific object as an icon in the *Object Pool* display area.

Object `unclipGeom` *PlaneModule*
Undos the effect of the `clipGeom` command described above.

Object `update`
Calls the internal update method of the object.

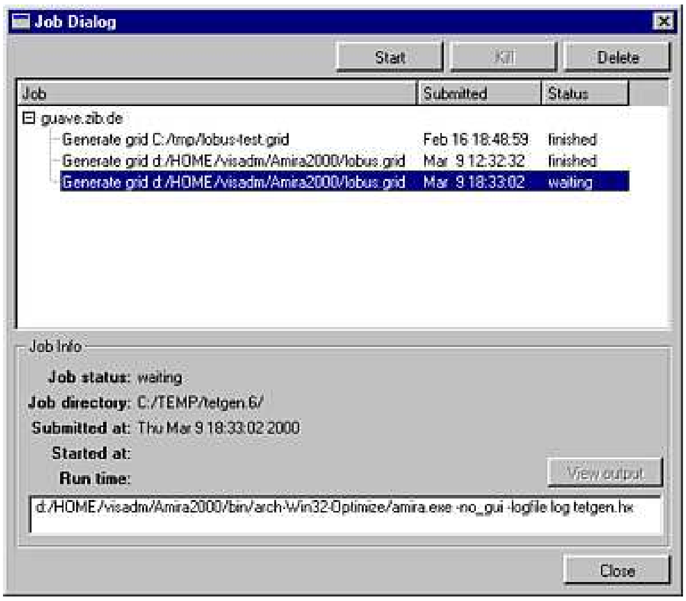


Figure 4.5: The job dialog lets you start, stop, examine, and delete batch jobs.

4.1.8 Job Dialog

Certain time-consuming operations in Amira can be performed in batch mode. For this purpose Amira provides a job queue, where jobs like generation of a tetrahedral grid can be submitted. You can inspect the current status of the job queue, start and delete jobs from the queue by selecting *Jobs* from Amira's file menu. This will bring up the *Job Dialog*.

In the upper part of the job dialog the current list of jobs of a user is shown. For each job a short description is displayed, as well as the time when the job has been submitted and the current state of the job. A job may be waiting for execution, running, finished, or it may have been killed.

The job directory

For each job a temporary directory is created containing any required input data, scripts, state information, and log files. On Unix systems this directory is created at the location specified by the environment variable `TMPDIR`. If no such variable exists, `/tmp` is used. On Windows systems the default temporary directory is used. Typically this will be `C:/TEMP`.

Controlling the job queue

A job's state may be manipulated using the action buttons shown above the job list. In order to start the job queue select the first job waiting for execution and then press the *Start* button. Note that only one job can be executed at a time. In order to kill a running job, select it in the job list and press the *Kill* button. You may delete a job from the job queue using the *Delete* button. When deleting a job the temporary job directory will be removed as well.

Information about a job

Once you have selected a job in the job queue, more detailed information about it will be displayed in the lower part of the dialog window, notably the state of the job, the temporary job directory, the submit time, the time when the job has been started, the run time, and the name of the command to be executed. Any console output of a running job will be redirected to a log file located in the temporary job directory. Once such a log file exists and has non-zero size you may inspect it by pushing the *View output* button.

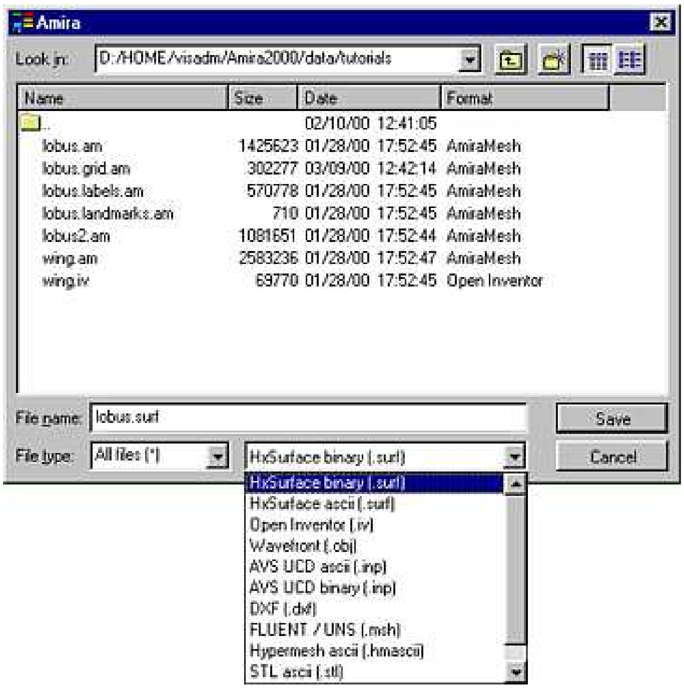


Figure 4.6: Amira's file dialog.

Commands

`job submit` *cmd info [tmpdir]*
Submits a new job to the job queue. `command` specifies the command to be executed. `info` specifies the info string displayed in the job dialog. `tmpdir` specifies the temporary job directory. If this argument is omitted a temporary job directory is created by Amira itself. In any case, the directory will be automatically deleted when the job is removed from the job queue. Example: `job submit "clock.exe" "Test job"`

`job run`
Starts the first job in job queue pending for execution. When a job is finished, execution of the next job in the queue starts automatically, thus all jobs in the queue will be executed successiveley by `job run`.

4.1.9 File Dialog

The *File Dialog* is the user interface component for importing and exporting data into resp. from Amira. It is used at several places in Amira, most prominently by the *Load*, *Save Data As*, and *Save Network* items of the main window's *File* menu.

The dialog provides two modes of information, a detail mode and a multi-column mode. In the detail mode, which is active by default, some file data are shown next to each filename, namely the file size, the file's last modification time, and the file format. You may sort the file list according to each of these properties by clicking on the particular column's header bar. Subdirectories will always be displayed first. In multi-column mode only the file name is displayed. You may switch between both modes using the tool buttons in the upper right part of the dialog window.

Most file formats supported by Amira will be recognized automatically, either by analysing the file header or by looking at the file name suffix. A list of all *supported file formats* is contained in the reference section of this manual. You may manually set the format of a file by means of the dialog's popup menu (see below).

Figure 4.7: The snapshot dialog allows you to save or print the contents of a viewer window.

Changing Directories

You can change the current directory by double-clicking a subdirectory in the file list or by entering a new directory in the dialog's path list. By default, the path list contains the current directory, the directory containg the demo data sets provided with Amira, as well as all directories defined by the environment variable `AMIRA_DATADIR`. In `AMIRA_DATADIR` multiple directory names have to be separated by colons `[:]` on Unix systems or by semicolons `[;]` on Windows systems. In addition, on Windows system the names of the twelve most recently visited directories are stored in the path list.

Selecting Files

To select a single file just click on it or type in its name in the file name text field.

In some cases you might want to select more than one file at once, e.g. when loading a 3D image data set as a series of single 2D images. You can do this by selecting the first file first and then shift-selecting the last file. Then all intermediate files will be selected as well. Moreover, you may ctrl-click a file in order to toggle its selection state individually.

Using the Filename Filter

The filename filter is visible when the dialog is in import mode (*Load File*). It is useful to restrict the list of filenames to a subset matched by the filter expression. The filter expression may contain the wildcard characters `?` (matches any character) and `*` (matches an arbitrary character sequence). For example, the expression `*.img` matches all filenames with the suffix `.img`.

The File Dialog's Popup Menu

The file dialog provides a popup menu which may be activated by pressing the right mouse button over the file list. Among others, this menu lets you rename or delete files or directories, provided you have the permission to do that. Note that you may only delete empty directories.

Using the *Format* option of the popup menu you may manually set the format to be used when loading a file into Amira. This option is useful if for some reason the wrong format has been detected automatically, or if no format at all could be detected. Note however, that any format specification set manually will be overwritten when the directory is reread the next time.

4.1.10 Snapshot Dialog

The *Snapshot Dialog* provides the user interface of the viewer's snapshot facility. You get the dialog by clicking on the camera icon in the left toolbar of the *viewer*.

You can either send the grabbed image directly to the printer or save it into a file in order to include it into other documents. In the "to printer" mode you first have to select and configure a printer by pushing the *Configure* button. In addition, you may enter an arbitrary text string which is printed as an annotation text below the snapshot image.

In the "to file" mode type in a filename or use the *File Dialog* (*Browse* button) to choose the output file.

The format option lets you select the file format (EPS or Raster Image) to be produced for file output. If raster image has been selected the file format will be determined from the file name suffix. The following formats are supported: TIFF (`.tif,.tiff`), SGI-RGB (`.rgb,.sgi,.bw`), JPEG (`.jpg,.jpeg`), PNM (`.pgm,.ppm`), BMP (`.bmp`), PNG (`.png`), and Encapsulated Postscript (`.eps`).

Note: On many computers you may get better quality snapshots if you first switch the viewer into *single buffer mode*. You can do this using the right mouse button popup menu in the viewer window. By default, the viewer uses *double buffer mode*. This mode avoids flicker effects by rendering new images in the back buffer before copying the final image into the front buffer for visibility. However, since two buffers are used, on some platforms the color depth is reduced and dithering is applied. This may deteriorate image quality.

In *single buffer mode* you can observe some flickering resulting from images being rendered in the front buffer whose content becomes visible immediately. On the other hand, on some platforms the color depth in single buffer mode is higher than in double buffer mode. Therefore, when taking snapshots choose single buffer mode to obtain best quality.

4.2 General Concepts

This section contains some general comments on how data objects are organized and classified in Amira. In particular, the following topics are discussed:

- *Amira Class Structure*
- *Scalar and Vector Fields*
- *Coordinates and Grids*
- *Surface Data*
- *Vertex Set*
- *Transformations*
- *Parameters*

4.2.1 Class Structure

In this section we discuss the object-oriented design of Amira in a little more detail. You already know that data objects, e.g. grey level image data or vector field sets, appear as separate icons in the *Object Pool*. You also know that there are certain display modules which can be used to visualize the data objects. While some modules can be connected to many different data objects, e.g. the *Bounding Box* module, others cannot, e.g. the *Ortho Slice* module. The latter can only be connected to voxel data or to scalar distributions on voxel grids. The reason is that internally both are represented as a scalar field with uniform cartesian coordinates. Consequently, the same visualization methods can be applied to both. On the other hand, for example a volumetric tetrahedral grid model of the object of interest usually looks completely different. But since it is also a 3D data object, the same *Bounding Box* module can be connected to it.

In summary, there are Amira data objects that might be conceived of different type, but with respect to mathematical structure, applicability of viewing and other processing modules, as well as programming interface design have many common properties. Obeying principles of object-oriented design, the data types of Amira are organized in class hierarchies where common properties are attributed to 'higher up' classes and inherited to 'derived' classes, as sub-classes of a class are commonly referred to. Conceptually each object occuring in Amira is an instance of a class and each of its predecessors in the hierarchy that the class belongs to. The classes and their hierarchies are defined within Amira. As the user you normally deal with instances of classes only. For instance, there is a class called "HxObject" with sub-classes "HxData" and "HxModule". "HxData" comprises the types of data associated with data objects used for modeling the objects of interest, e.g. volumetric tetraedral grids or surfaces. "HxModule" comprises data types that have been assigned to display and other processing modules, again in accordance with principles of object-oriented design. This is why Amira's data objects and processing modules are commonly referred to as "objects".

There are also classes in Amira that are not derived from "HxObject" and constitute other data types, and there are several independent class hierarchies. E.g. there is a class called "HxPort" from which all classes supporting the operation and display of interface control elements are derived (see section *Working Area* and the *List of Ports* in the index section of the user's guide).

A single class hierarchy is usually figured as an upside-down tree, i.e. with the root at the top. Thus the *data* class tree is the one to which the information as to which processing module is applicable to which data object is hooked. Its classes reflect the mathematical structure of the object models supported by Amira. For example, scalar fields and vector fields are such structures and derived from a common "field" class which represents a mapping $R^3 \rightarrow R^n$. Deriving a sub-class from this base class requires a value to be specified for n.

At the same time fields defined on cartesian grids are distinguished from fields defined on tetrahedral grids, i.e., this distinction is part of the classification scheme that gives rise to branches in the *data* class subtree. In the next section of this chapter you will learn more about the *data* class hierarchy. In the second section we discuss how some data types frequently used for various visualization tasks fit into it.

Internally, all class names begin with a prefix *Hx*. However, you don't have to remember these names, unless you want to use the command shell to create objects. For example, a bounding box is usually created by choosing the *BoundingBox* item from the popup menu of a data object that is to be visualized, but you may also create it by typing `create HxBoundingBox` in the command window.

4.2.2 Scalar Field and Vector Fields

The most important fields in Amira are three-dimensional ones. These fields are defined on a certain domain $\subseteq \mathbb{R}^3$. A field can be evaluated at any point inside its domain. If the field is defined on a discrete grid, this ususally involves some kind of interpolation.

Scalar Fields

A 3D scalar field is a mapping $R^3 \rightarrow R$. The base class of all 3D scalar fields in Amira is *HxScalarField3*. Various sub-classes represent different ways of defining a scalar field. There are a number of visualization methods for them, for example pseudo-coloring on cutting planes, iso-surfacing, or volume rendering. However, many visualization modules in Amira rely on a special field representation. Therefore, they can only operate on sub-classes of a general scalar field. Whenever a given geometry is to be pseudo-colored, any kind of scalar field can be used (cf. *Colorwash*, *GridVolume*, *Isosurface*).

The class *HxTetraScalarField3* represents a field which is defined on a tetrahedral grid. On each grid vertex a scalar value, e.g. a temperature, is defined. Values associated to points inside a tetrahedron are obtained from the four vertex values by linear interpolation. This class does not provide a copy of the grid itself, instead a reference to the grid is provided. This is indicated in the Object Pool by a line which connects the grid icon and the field icon. As a consequence, a field defined on a tetrahedral grid cannot be loaded into the system if the grid itself is not already present.

The class *HxRegScalarField3* represents a field which is defined on a regular cartesian grid. Such a grid is organized as a three-dimensional array of nodes. In the most simple case these nodes are axis-aligned and have equal spacings. The coordinates of such a uniform grid can be obtained from a simple bounding box containing the origin vector and increments for all directions. Stacked coordinates are another example. Here the spacing in z-direction between subsequent slices may be different. In any case scalar values inside a hexahedral grid cell are obtained from the eight vertex values using trilinear interpolation. While the *OrthoSlice* module can only be used to visualize scalar fields with uniform or stacked coordinates, other modules like *ObliqueSlice* or *Isosurface* work for all scalar fields with regular coordinates.

Yet another example of a scalar field is the class *HxAnnaScalarField3*. It represents an analytically defined scalar field. To create such a field, select *ScalarField* from the *Edit Create* menu of Amira's main window. You have to specify a mathematical expression which is used to evaluate the field at each requested position.

Up to three other fields can be connected to the object. These can be combined to a new scalar field, even if they are defined on different grids.

Vector Fields

As for scalar fields Amira provides a number of vector field classes, these are derived from the base classes *HxVectorField3* and *HxComplexVectorField3*. While ordinary vector fields return a three-component vector at each position, complex vector fields return a six-component vector. Complex vector fields are used for encoding stationary electromagnetic wave pattern as required by some applications. Usually complex vector fields are visualized by projecting them into the space of reals using different phase offsets. The Vectors module even allows you to animate the phase offset. In this way a nice impression of the oscillating wave pattern is obtained.

4.2.3 Coordinates and Grids

Amira currently supports two important grid types, namely grids with hexahedral structure (regular grids), and unstructured tetrahedral grids. Other types, e.g. unstructured grids with hexahedral cells or block-structured grids will be added in future releases of Amira.

Regular Grids

A regular grid consists of a three-dimensional array of nodes. Each node may be addressed by an index triple *(i,j,k)*. Regular grids are further distinguished according to the kind of coordinates being used. The most simple case comprises *uniform* coordinates, where all cells are assumed to be rectangular and axis-aligned. Moreover, the grid spacing is constant along each axis. A grid with *stacked* coordinates may be imagined as a stack of uniform 2D slices. However, the distance between neighbouring slices in z-direction may vary. In case of *rectilinear* coordinates the cells are still aligned to the axes, but the grid spacing may vary from cell to cell. Finally, in case of *curvilinear* coordinates each node of the grid may have arbitrary coordinates. Grids with curvilinear coordinates are often used in fluid dynamics because they have a simple structure but still allow for accurate modeling of complex shapes like rotor blades or airfoils.

Tetrahedral Grids

The *TetraGrid* class represents a volumetric grid composed of many tetrahedrons. Such grids can generally be used to perform finite-element simulations, e.g. E-field simulations.

A considerable amount of information is maintained in a *TetraGrid*. For each vertex a 3D coordinate vector is stored. For each tetrahedron the indices of its four vertices are stored as well as a number indicating the segment the tetrahedron belongs to as obtained by a segmentation procedure. Beside this fundamental information a number of additional variables are stored in order for the grid being displayed quickly. In particular all triangles or faces are stored separately together with six face indices for each tetrahedron. In addition for each face pointers to the two tetrahedrons it belongs to are stored. This way the neighborhood information can be obtained efficiently.

When simulating E-fields using the finite-element method, the edges of a grid need to be stored explicitly, because vector or Whitney elements are used. These elements and its corresponding coefficients are defined on a per-edge basis. When a grid is selected information on the number of its vertices, edges, faces, and tetrahedrons is displayed.

4.2.4 Surface Data

Amira provides a special-purpose data class for representing triangular surfaces, called *HxSurface*. This class is documented in more detail in the index section of the user's guide. For the moment, we only mention that the class maintains connectivity information and that it may represent manifold as well as non-manifold topologies.

The surface class provides a rich set of Tcl commands. It is a good example of an Amira data class that does not simply store information, but allows the user to query and manipulate the data by means of special-purpose methods and interfaces.

4.2.5 Vertex Set

Another example of data abstraction and inheritance is the *VertexSet* class. Many data objects in Amira are derived from this class, e.g., landmark sets, molecules, surfaces, or tetrahedral grids. All these objects provide a list of points with x-, y-, and z-coordinates. Other modules which require a list of points as input only need to access the *VertexSet* base class, but don't need to know the actual type of the data object.

One such example of a generic module operating on *VertexSet* objects is the *VertexView* module. This module allows you to visualize vertex positions by drawing dots or little spheres at each point.

4.2.6 Transformations

Data objects in Amira can be modified using an arbitrary affine transformation. For example, this makes it possible to align two different data objects so that they roughly match each other. Internally, affine transformations are represented by a 4x4 transformation matrix. In particular, a uniform scalar field remains a uniform scalar field, even if it is rotated or sheared. Display modules like *OrthoSlice* still can exploit the simple structure of the uniform field. The possible transformation is automatically applied to any geometry shown in the 3D viewer.

In order to interactively manipulate the transformation matrix use the *Transform Editor* (documentation is contained in the index section of the user's guide).

Be careful when saving transformed data sets! Most file formats do not allow to store affine transformations. In this case you have to apply the current transformation to the data. This can be done using the Tcl-command `applyTransform`. In case of *vertex set objects* the transformation is applied to all vertices. Old coordinates are replaced by new ones, and the transformation matrix is reset to identity afterwards. After a transformation has been applied to a data set, it cannot be unset easily anymore.

If a transformation is applied to uniform fields, e.g. to 3D image data, the coordinate structure is not changed, i.e., the field remains a uniform one. Instead, the data values are resampled, i.e., the transformed field is evaluated at every vertex of the final regular grid. The bounding box of the resulting grid is modified so that it completely encloses the transformed original box.

4.2.7 Parameters

For any data object an arbitrary number of additional parameters or attributes may be defined. Parameters can be interactively added, deleted, or edited using the *parameter editor*. Parameters are useful for example to store certain parameters of a simulation or of an experiment. In this way the history of a data object can be followed.

There are certain parameters which are interpreted by several Amira modules. The meaning of these parameters is summarized in the following list:

- `Colormap name`
  This specifies the name of the default colormap used to visualize the data. Some modules automatically search the object pool for this colormap and for example use it for pseudocoloring.
- `DataWindow minVal maxVal`
  This indicates the preferred data range used for visualizing the data. The OrthoSlice module automatically maps values below `minVal` to black and values above `maxVal` to white.
- `LoadCmd cmd`
  This parameter is usually set by import filters when a data object is read. It is used when saving the current network into a file and it allows to restore the object automatically. Internal use only.

Note that there are many file formats which do not allow to store parameters. Therefore, information might get lost when you save the data set in such a format. If in doubt, use the Amira specific *AmiraMesh* format.

Chapter 5

Technical Information

This chapter contains technical information about Amira which is not covered in the previous chapters.

- *Data Import*
- *Command Line Options*
- *Environment Variables*
- *Amira start-up script*
- *Frequently Asked Questions*
- *System Requirements*
- *Acknowledgements and Copyrights*

5.1 Data Import

Usually, one of the first things Amira users want to know is how to import their own data into the system. This section contains some advice intended to ease this task.

In the simplest case, your data is already present in a standard file format supported by Amira. To import such files, simply use the *File Load* menu. A *list of all supported formats* can be found in the index section of the user's guide. Usually, the system recognizes the format of a file automatically by analyzing the file header or the filename suffix. If a supported format is detected, the file browser indicates the format name.

Often, *3D image volumes* are stored slice by slice using standard 2D image formats such as TIFF or JPEG. In case of medical images, slices are commonly stored in ACR-NEMA or DICOM format. If you select multiple 2D slices simultaneously in the file browser, all slices will automatically be combined into a single 3D data set. Simultaneous selection is most easily achieved by first clicking the first slice and then shift-clicking the last one.

If your data is not already present in a standard file format supported by Amira you will have to write your own converter or export filter. For many data objects such as 3D images, regular fields, or tetrahedral grids Amira's native *AmiraMesh* format is most appropriate. Using this format you can even represent point sets or line segments for which there is hardly any other standard format. The *AmiraMesh documentation* explains the file syntax in detail and contains examples of how to encode different data objects. One important Amira data type, triangular non-manifold surfaces, cannot be represented in a *AmiraMesh* file but has its own file format called *HxSurface* format.

Finally, in case of images or regular fields with uniform coordinates you may also read *binary raw data*. Note that for raw data the dimensions and the bounding box of the data volume must be entered manually in a dialog box which pops up after you have selected the file in the file browser.

5.2 Command Line Options

This section describes the command line options understood by Amira. In general, on Unix systems Amira is started via the `start` script located in the subdirectory `bin`. Usually, this script will be linked to `/usr/local/bin/amira` or something similar. Alternatively, the user may define an alias `amira` pointing to `bin/start`.

On Windows systems Amira is usually started via the start menu or via a desktop icon. Nevertheless, the Amira executable may also be invoked directly by calling `bin/arch-Win32-Optimize/amira.exe`. In this case, the same command line options as on a Unix system are understood.

The syntax of Amira is as follows:

```
amira [options] [files ...]
```

Data files specified in the command line will be loaded automatically. In addition to data files, script files can also be specified. These scripts will be executed when the program starts.

The following options are supported:

- `-help`
  Prints a short summary of command line options.
- `-version`
  Prints the version string of Amira.
- `-no_stencils`
  Tells Amira not to ask for a stencil buffer in its 3D graphics windows. This option can be set to exploit hardware acceleration on some low-end PC graphics boards.
- `-no_overlays`
  Tells Amira not to use overlay planes in its 3D graphics windows. Use this option if you experience problems when redirecting Amira on a remote display.
- `-no_gui`
  Starts up Amira without opening any windows. This option is useful for executing a script in batch mode.
- `-logfile filename`
  Causes any messages printed in the *console window* also to be written into the specified log file. Useful especially in conjunction with the `-no_gui` option.
- `-depth_size number`
  This option is only supported on Linux systems. It specifies the preferred depth of the depth buffer. The default on Linux systems is 16 bits.
- `-style={windows | motif | cde}` This option sets the display style of Amira's Qt user interface.
- `-debug` This options applies to the developer version only. It causes local packages to be executed in debug version. By default, optimized code will be used.
- `-cmd command [-host hostname] [-port port]`
  Send Tcl command to a running Amira application. Optionally the host name and the port number can be specified. You must type `app -listen` in the console window of Amira before commands can be received.

5.3 Environment Variables

In order to execute Amira no special environment settings are required. The Amira start script automatically detects in which directory Amira has been installed. If necessary, certain shell variables, for example the shared library path or the X11 application resource path, will be set for you.

Nevertheless, some environment variables may be set in order to control certain features. These variables are listed below and their effects are explained. Environment variables may be set using `setenv` (C-shell) or `export` (Bourne shell).

- `AMIRA_DATADIR`
  A list of data directory names separated by semicolons `[;]` on Windows systems and colons `[:]` on Unix systems. The first directory will be used as the default directory of the file dialog. Other directories are quickly accessible via the file dialog's path list.
- `AMIRA_MESA`
  If this variable is set, it is assumed that OpenGL software rendering is used, i.e. by means of the Mesa library. In this case, no limits to the size of texture memory are set. The *Voltex* module does not apply a bricking technique.
- `AMIRA_TEXMEM`
  Specifies the amount of texture memory in megabytes. If this variable is not set some heuristics are applied to determine the amount of texture memory available on a system. However, these heuristics may not always yield a correct value.
- `AMIRA_MULTISAMPLE`
  On high-end graphics systems like SGI Onyx2, a multi-sample visual is selected by default. In this way, efficient scene anti-aliasing is achieved. If you want to disable this feature, set the environment variable `AMIRA_MULTISAMPLE` to 0.
- `AMIRA_NO_OVERLAYS`
  If this variable is set, Amira will not use overlay planes in its 3D graphics windows. The same effect can be obtained by means of the `-no_overlays` command line option. Turn off overlays if you experience problems with redirecting Amira on a remote display.
- `AMIRA_LOCAL`
  Specifies the location of an additional local Amira package. Any objects redefined in the local package will replace the ones defined in the main Amira package. Moreover, a local Amira package may be used to install additional objects locally.
- `AMIRA_SMALLFONT`
  Unix systems only. If this variable is set a small font will be used in all ports being displayed in the *Working Area* even if the screen resolution is 1280x1024 or bigger. By default, the small font will be used only in case of smaller resolutions.
- `AMIRA_XSHM`
  Unix systems only. Set this variable to 0 if you want to suppress the use of the X shared memory extension in Amira's *image segmentation editor*.
- `TMPDIR`
  This variables specifies in which directory temporary data should be stored. If not set, such data will be created under `/tmp`. Among others, this variable is interpreted by Amira's job queue.

5.4 User-defined start-up script

Amira may be customized in certain ways by providing a user-defined start-up script. The default start-up script, called `Amira.init`, is located in the subdirectory `share/resources` of the Amira installation directory. This script is read each time the program is started. Among other things, the start-up script is responsible for registering file formats, modules, and editors and for loading the default colormaps.

If a file called `Amira.init` is found in the current working directory, this file is read instead of the default start-up script. If no such file is found, on Unix systems it is checked if there exists a start-up script called `.Amira` in the user's home directory. Below an example of a user-defined start-up script is shown:

```
# Execute the default start-up script
source $AMIRA_ROOT/share/resources/Amira.init 0

# Set up a uniform black background
```

```
viewer 0 setBackgroundMode 0
viewer 0 setBackgroundColor black

# Choose non-default font size for the help browser
help setFontSize 12

# Restore camera setting by hitting F2 key
proc onKeyF2 { } {
    viewer setCameraOrientation 1 0 0 3.14159
    viewer setCameraPosition 0 0 -2.50585
    viewer setCameraFocalDistance 2.50585
}
```

In this example, first the system's default start-up script is executed. This ensures that all Amira objects are registered properly. Then some special settings are made. Finally, a hot-key procedure is defined for the function key F2. You can define such a procedure for any other function key as well. In addition, procedures like `onKeyShiftF2` or `onKeyCtrlF2` can be defined. These procedures are executed when a function key is pressed with the Shift or the Ctrl modifier key being pressed down.

5.5 Frequently Asked Questions

Questions

General

1. *What is Amira?*
2. *What is the latest version of Amira?*
3. *How can I try Amira? Are there demo or evaluation keys?*
4. *Is Amira Y2K compliant?*

Installation, hardware and platform related questions

5. *What are the supported platforms for Amira?*
6. *Is Amira available on Windows 98/NT ?*
7. *Compatibility between Windows and Unix version ?*
8. *Can I display Amira on a remote screen?*
9. *Do I need to have root or administrator privileges in order to install the product?*
10. *What are the software and hardware requirements?*
11. *What is the minimum configuration required for my platform?*
12. *What is the recommended hardware for my purpose?*
13. *What is the resource consumption of Amira, for memory, disk, cpu, graphics?*
14. *Are there any limits to the size of textures that graphics boards can use?*
15. *Does Amira support the VolumePro volume rendering hardware?*

Resources, examples, documentation

16. *Where should I start to learn how to use Amira?*
17. *How long do I need to learn how to use Amira?*
18. *Can I get training courses?*
19. *What is the relevant documentation available?*
20. *How do I see what command line options Amira accepts?*
21. *How can I check the version of my Amira package?*
22. *Are there any examples or demos?*

23. *Is there a specific newsgroups?*
24. *Is there a mailing list?*
25. *Is there a web site?*
26. *Where can I find (free) modules for Amira? Is there a public repository for 3rd party contributions?*
27. *How can I learn Tcl?*

Technology

28. *What graphics libraries are used by Amira ?*
29. *What is Open Inventor?*
30. *What is OpenGL?*
31. *What is Tcl?*
32. *What is Qt?*
33. *Is Amira dataflow oriented?*
34. *How do modules communicate?*
35. *What is the firing order of modules?*

Data input/output, printing

36. *What are the supported data formats (input and output)?*
37. *How can I use Amira to import/export image formats other than the Amiira image format?*
38. *How can I define the pixel size for my 3D image volume ?*
39. *What are the data/mesh/UCD cells types supported by Amira?*
40. *How can I read my data (with some specific file format)?*
41. *How can I interface to my database?*
42. *How can I reuse my work with Amira? Can I compose modules?*
43. *How can I print with Amira? How can I take a snapshot of the viewer window?*
44. *What image formats are supported for snapshots?*
45. *How can I create printed reports including Amira images?*
46. *How can I publish Amira images or animations on the Web?*

Visualization

47. *Can I display axes?*
48. *Can I do image processing with Amira?*
49. *Does the surface reconstruction support non-manifold topologies?*
50. *Is it possible to start Amira up with NO display in order to do batch processing of data or to generate pictures and plots without displaying anything on the console?*
51. *How does Amira behave with large data sets?*
52. *How can I change the background color of the viewer?*
53. *Can I display a colormap in the viewer window as legend?*
54. *How can I adjust color and transparency of individual parts of a surface?*
55. *How can I create an iso-surface with fewer polygons than the iso-surface module extracts?*
56. *How do I visualize data with holes in it?*
57. *How can I read a series of single image files such that I get a 3D stack?*
58. *How can I quickly switch between two different data sets?*
59. *How can I compare two data sets?*

Specific features

60. *Does Amira support Stereo viewing?*
61. *Does Amira support VR devices, such as 3D mouse, head-mounted displays or CAVE systems?*
62. *Can I use antialiasing?*

Answers

General

1. What is Amira?

   Amira is a professional general-purpose visualization and 3D reconstruction software. Visualization means that you can display various data sets, notably 3D image data, vector fields, and finite element data. 3D reconstruction means that you can create polygonal surface models as well as tetrahedral grids from 3D image data.

2. What is the latest version of Amira?

   The latest version is Amira 2.2.

3. How can I test Amira? Are there demo keys available?

   For evaluation purposes a fully functional version of Amira can be downloaded from `www.amiravis.com` after electronic registration. This version will have a time-limited license key.

4. Is Amira Y2K compliant?

   Yes.

Installation, hardware and platform related questions

5. What are the supported platforms for Amira?

   Amira 2.2 runs on Microsoft Windows 9x, Microsoft Windows NT4, Microsoft Windows 2000, HP-UX 10.20, SGI Irix 6.5.x, Sun Solaris 2.7, and on Linux (RedHat 6.2, Suse 6.3). Details are described in the user's guide in section *System Requirements*.

6. Is Amira available on Windows 98/NT ?

   Yes. Amira is available on Microsoft Windows 9x, Microsoft Windows NT4 and Microsoft Windows 2000.

7. Compatibility between Windows and Unix version ?

   Both, the Windows version and the Unix version contain exactly the same set of data objects, editors, and modules. Data files can be exchanged between Windows and Unix without limitations. There are only minor differences between different version which are due to differences in the underlying hardware, e.g. the availability of 3D textures for volume rendering.

   The user interface of Amira 2.2 is written using the Qt toolkit. This ensures that the GUI looks similar and has the same functionality on both Windows and Unix systems.

8. Can I display Amira on a remote screen?

   In general, it is not recommended that you use a remote display for demanding interactive 3D graphics applications like Amira.

   However, in principle you can redirect the output of the Unix version to a remote display. For IRIX, HP-UX, and SunOS, the remote X server needs to support the GLX extension. Call `xdpyinfo` to find out whether your computer has that extension installed. Then simply set the `DISPLAY` variable and start Amira.

   If heterogeneous platforms are involved problems with the overlay planes may occur. If the viewer window remains black try using the Amira command line option `no_overlays` or set the environment variable `AMIRA_NO_OVERLAYS`.

9. Do I need to have root or administrator privileges in order to install the product?

   No. On Windows systems you can run the setup tool without having administrator privileges. On Unix systems simply extract the provided tar file.

   However, on Sun and HP-UX it is recommended that you set the default visual of the X server to 24-bit true color. This may require root privileges. In addition, on some HP-UX systems it is recommended that you increase certain kernel parameters like process data size or stack limit. This requires root privileges as well.

10. What are the software and hardware requirements?

    Software and hardware requirements are described in the user's guide in section *System Require-*

ments.

11. What is the minimum configuration required for my platform?

    You need a graphics board with at least 24 bits of color per pixel (16 bits for Linux). At least 64 MB of main memory are required, 256 MB or more are recommended.

12. What is the recommended hardware for my purpose?

    In principle, all features are available even on low-end graphics boards. However, if software emulation is used, rendering performance may drop severely. Many visualization techniques applied in Amira make use of *texture mapping*. Therefore it is highly recommended to have a graphics board which supports texture mapping in hardware.

    Amira offers direct volume rendering based on 3D textures. This feature is either not available or is very slow on many PC graphics cards. In order to use it on Unix systems a high-end graphics board is recommended, e.g. a SGI Onyx2, a SGI Octane MXE, or a HP C3000 Visualize fx/6 or fx/10 with texture option.

13. What is the resource consumption of Amira, for memory, disk, CPU, graphics?

    Amira needs roughly about 60 MB of disk space. Memory, CPU and graphics performance (of course) depend on the kind of data you are going to visualize. CPU speed is less critical than graphics performance. Most Amira modules are single-threaded. Therefore multi-processor systems are not a big advantage. Enough memory should be available in order to completely store the data to be visualized.

14. Are there any limits to the size of textures that graphics boards can use?

    Yes. The exact value depends on the graphics board. A typical limit will be 2048x2048 pixels per texture. On some architectures there is also a limit to the total amount of texture memory available. This is of special importance for texture-based volume rendering. Details are given in the documentation for the *Voltex* module.

15. Does Amira support the VolumePro volume rendering hardware?

    Yes, Amira supports the VolumePro board on the Windows NT and Windows 2000 platforms. If you need VolumePro support for other platforms, please contact us.

Resources, examples, documentation

16. Where should I start to learn how to use Amira?

    Probably, the best starting point are the *tutorials* in the user's guide. They provide a step-by-step learning-by-doing introduction.

17. How long does it take to learn how to use Amira?

    Amira is easy to use. After going through one of the tutorials for 15 minutes you will have an idea of the basic functionality. Usually, this is sufficient in order to be able to do first visualizations of your own data. Of course, becoming an Amira wizard and becoming familiar with all the features available in the system will take substantially more time.

18. Can I get training courses?

    Yes, training courses and consulting services are available. For more information refer to the web sites `www.amiravis.com` or `www.tgs.com`.

19. What is the relevant documentation? Where can I find help?

    The primary source for documentation is the Amira user's guide. This guide, as well as additional information, is provided on the Amira web site `www.amiravis.com` and also on `www.tgs.com`. If you have specific questions you will find contact information on these web sites.

20. How do I see what command line options Amira accepts?

    Command line option are documented in the user's guide in Section 5.2. In addition, starting Amira with `-help` gives you a short summary of options.

21. How can I check the version of my Amira package?

    To get the version of Amira type `app -version` in Amira's *console window*. In order to see when Amira was compiled use `app -built`. Please indicate the version string or compilation date whenever your report bugs or problems.

22. Are there any examples or demos?

    Yes. The Amira 2.2 distribution contains tutorials and example with demo data. The tutorials are contained in Chapter 3 of the user's guide. *Demos* are listed in the reference section of the user's guide.

23. Is there a specific newsgroups?

    There is no Amira specific newsgroup yet.

24. Is there a mailing list?

    Not yet.

25. Is there a web site?

    For technical information refer to `www.amiravis.com`. For sales and marketing contact `www.tgs.com`.

26. Where can I find (free) modules for Amira? Is there a public repository for 3rd party contributions?

    With the availabilty of the developer version a public repository for 3rd party contributions will be set up at `www.amiravis.com`.

27. How can I learn Tcl?

    There are many good Tcl books. For example, you can try *Tcl and the Tk Toolkit* by *John K. Ousterhout*, the creator of Tcl. Like many others this book also covers the Tk GUI toolkit. Note that Tk is not used in Amira.

Technology

28. What graphics libraries are used by Amira ?

    Amira is based on the Open Inventor graphics toolkit. Furthermore, Amira contains a number of custom Inventor nodes which implement special visualization techniques. These nodes apply direct OpenGL rendering.

29. What is Open Inventor?

    Open Inventor is a C++ library allowing you to describe and render 3D scenes. Open Inventor is built on top of OpenGL. This guarantees portability and hardware-accelerated performance across a wide range of platforms.

30. What is OpenGL?

    OpenGL is a library for rendering 3D graphics. OpenGL is the industry standard for professional 3D graphics. It is supported by all professional graphics hardware and by an increasing number of consumer graphics cards.

31. What is Tcl?

    Tcl is a popular scripting language. Tcl has a simple syntax, so you can learn Tcl in one afternoon. Amira has a built-in Tcl interpreter. This way Amira is scriptable.

32. What is Qt?

    Qt is a multi-platform GUI software toolkit developed by Troll Tech (`www.troll.no`). An application written with Qt can be compiled on Unix/X11 as well as on Windows. While the user interface of Amira 2.0 was based on Motif, all releases of Amira 2.1 and later (including the Windows version) are based on Qt. For the end-user this guarantees that the set of features and the user interface will be compatible across all platforms.

33. Is Amira dataflow oriented?

    No, Amira is not dataflow oriented. Amira is object oriented. Data objects are persistent in memory and represented in the user interface. Data are accessed by the modules using the C++ interfaces of the data classes.

34. How do modules communicate?

    Modules are loaded into a common process space at runtime, by using shared libraries. This way they can communicate like C++ objects in a normal C++ program. There is no overhead for module communication.

35. What is the firing order of modules?

    Most modules are fired in downstream order. If you create a new module from the popup menu of an existing one the new module will be downstream. Amira networks are typically much less complex than in dataflow-oriented visualization systems. Therefore the firing order is usually not of concern for the end-user.

Data input/output, printing

36. What are the supported data formats (input and output)?

    A *list of supported file formats* is contained in the index section of the user's guide.

37. How can I use Amira to import/export image formats other than the Amira image format?

    Amira supports several standard image formats such as TIFF, JPEG, SGI-RGB, ACR-NEMA, or DICOM. When Amira reads data it can usually determine the file format automatically. You simply select the file in the file browser and click OK. When Amira writes data, the file browser presents an option menu containing all file formats which can be used to export that data. Use this menu to select a non-default format.

38. How can I define the pixel size for my 3D image volume ?

    There is a difference between the number of pixels in a 3D image volume (e.g. 512x512x200) and its physical bounding box (e.g. 30cm x 30cm x 20cm). Often voxels are even not equally sized in all directions. Many 2D image formats do not contain this extra information. When reading images, you can supply this information in Amira's image input dialog. You can also change this information later by selecting the data set (green icon) and choosing the *Image Crop Editor* button.

39. What are the data/mesh/UCD cells types supported by Amira?

    The list of supported data types includes

    - 2D and 3D grayscale images (8, 16, and 32 bits)
    - 32 bit RGBA-color images
    - segmentation results based on labeled voxels
    - unstructured tetrahedral meshes
    - unstructured hexahedral meshes
    - scalar and vector fields defined on uniform, stacked, rectilinear, curvilinear, tetrahedral, or hexahedral grids
    - manifold and non-manifold surfaces
    - colormaps including transparency values
    - Open Inventor scene graphs

40. How can I read my data (with some specific file format)?

    Amira supports a number of standard file formats. Therefore it is likely that you can find a converter if your file format is not supported. For image data, Amira provides a powerful Raw-Data interface, which can handle most simple binary file formats with some additional manual work.

    In order to implement custom I/O methods, the entensible version of Amira called AmiraDev is required.

41. How can I get access to my database?

    Data I/O is handled via files. The developer version, of course, allows the user to add any database interface he/she wants.

42. How can I reuse my work with Amira? Can I compose modules?

    You can save networks, and you can save data objects that have been created or modified. In order to build new modules, you must use the developer version, or write script objects in Tcl.

43. How can I print with Amira? How can I take a snapshot of the viewer window?

    In the viewer, on the left hand side, there is an icon showing a photo camera. This allows you to write snapshots of the 3D scene to a file or to a printer. If no printers show up in the list then probably they are not properly installed. In the latter case you can still print to a PostScript file on disk and print that file from a different computer.

You can also use the command line interface to make snapshots. This is useful for generating animations via a Tcl script. The syntax is `viewer <n> snapshot <filename>`, where `<n>` denotes the viewer window to be captured. The format of the output file is determined automatically from the file name suffix.

44. What image formats are supported for snapshots?

    Snapshots can be stored in TIFF, JPEG, SGI-RGB, PNM, BMP, PNG, or EPS format. The file type is determined automatically from the file name suffix.

45. How can I create printed reports including Amira images?

    You may use any desktop publishing or word processing system of your choice. Probably all of them allow you to import either TIFF or JPEG or EPS files.

46. How can I publish Amira images or animations on the Web?

    Make snapshots and save them as JPEG images, or create animation sequences as described below.

Visualization

47. Can I display axes?

    Yes. Use the menu entry *Axis* in the view menu. This will display global axes located at the origin of the world coordinate system. You may also attach local axes to any data object by selecting *Display LocalAxis* from the object's popup menu.

48. Can I do image processing with Amira?

    Basic image processing functionality is provided although Amira is not a dedicated image-processing program. For example, the *Image Filters* editor supports smoothing, sharpening, as well as certain morphological operations.

49. Does the surface reconstruction support non-manifold topologies?

    Yes. In contrast to many other products, non-manifold topologies are handled in a consistent way by Amira.

50. Is it possible to start Amira up with NO display in order to do batch processing of data or to generate pictures and plots without displaying anything on the console?

    You can start Amira with the `-no_gui` command line option in order to execute scripts in batch mode.

51. How does Amira behave with large data sets?

    Amira is an interactive visualization system. Therefore, data sets must be loaded into main memory in order to be processed. In some projects very large dynamic data sets (up to several 100 GB) have been visualized with Amira. In this case special reader modules have been used which only read subsets of the data at once.

52. How can I change the background color of the viewer?

    There are three different background modes, namely *uniform*, *gradient*, and *checkerboard*. These modes can be set for all viewers via the *View Background* menu of the main window or via the command `viewer <n> setBackgroundMode <mode>` for a particular *viewer*. The primary background color can be adjusted via the camera icon in the upper left corner of the viewer window or via the command `viewer <n> setBackgroundColor <color>`. The secondary color used in gradient and checkerboard mode can be adjusted via the command `viewer <n> setBackgroundColor2 <color>`.

    You can also place an arbitrary raster image in the background using the command `viewer <n> setBackgroundImage <filename>`. Any image file in TIFF, SGI-RGB, JPEG, PNM, BMP, or PNG format can be read. However, note that the image is not shown if its size is greater than that of the viewer window.

53. Can I display a colormap in the viewer window as legend?

    First make the colormap icon visible in the Working Area. This can be done by selecting the *Show* or *Show All* item of *Edit* in the menu bar of the Working Area. Then click on the green colormap icon with the right mouse button and select *Show Colormap*.

54. How can I adjust color and transparency of individual parts of a surface?

    A surface object may consist of multiple patches refering to different materials. The color of each material can be adjusted using the Tcl command `setColor` described in *Surface*. Likewise, for each material a specific transparency value may be set using the command `setTransparency`. In this way certain parts of a surface may be highlighted. Note that you must choose draw style *transparent* in order to enable transparencies. Also note that color mode *mixed* is most appropriate for transparent surfaces in terms of performance and meaning.

55. How can I create an iso-surface with fewer polygons than the iso-surface module extracts?

    First of all, the iso-surface module provides a special option called compact cubes which produces about 40 percent fewer triangles than standard methods. Moreover, very large data sets may be downsampled on-the-fly during isosurface generation.

    If you need more flexibility, you can create a separate surface object by selecting *create surface* from the *more options* menu of the iso-surface module. You may then use the simplification editor in order to remove as many triangles from the surface as you want. You can display the resulting simplified surface using the SurfaceView module.

56. How do I visualize data with holes in it?

    There are several choices. You can apply a slicing module such as OrthoSlice or ObliqeSlice. Alternatively, you can clip away parts of a 3D geometry using an arbitrary slicing module. Slicing modules are indicated by an orange icon. Such modules provide a little push button that must be pressed in order to activate clipping. An empty clipping module can be created via the Edit Create menu of the main window.

    If you want to visualize surfaces or finite-element grids you can also use the selection box feature of the corresponding viewing modules. Most of these modules such as *SurfaceView* or *GridVolume* support a buffer concept which allows you to select which parts of the object should be displayed. Even the *Isosurface* module has such a buffer. For this module it can be enabled using the Amira command `Isosurface showBox`.

57. How can I read a series of single image files such that I get a 3D stack?

    Select all image files in the file browser at once. This can be done by clicking the first file and then shift-clicking the last one. Individual files can be selected and deselected by ctrl-clicking. After pressing the Ok button all images will be combined in a single 3D data stack. Note that the images should be of the same size.

58. How can I quickly switch between two different data sets?

    Click with the left mouse on the blue line connecting one of the data icons and the visualization module icon in the Working Area and - holding the left mouse button down - move the line to the icon of the other data object.

59. How can I compare two data sets?

    One solution is to display each data set in a different viewer. You can activate up to four viewers via the *View Layout* menu. If two viewers are visible attach a display module to each data set. You can control in which of the viewers the output of a module is displayed by selecting the module and setting or unsetting the orange viewer toggles. If you are using two *OrthoSlice* modules, make sure that the same slice is displayed in both viewers.

    In order to get the same camera settings in both viewers use the Tcl command `viewer 0 setSlaveViewer 1`. Whenever you navigate in viewer 0 the camera of viewer 1 will be adjusted as well.

    An alternative method to compare two different data sets is to compute and visualize the difference of both. To subtract two fields from each other use the *Arithmetic* module. Connect the module to both data sets by activating the popup menu over the small rectangle of the module's icon. Then enter an expression like A-B in order to compute the difference. You can visualize the result of the *Arithmetic* module by any of the ordinary display modules.

Specific features

60. Does Amira support Stereo viewing?

Yes. You will need special shutter glasses, e.g. Stereographics Crystal Eyes. Stereo viewing is successfully being used on SGI and HP-UX systems. The Windows version is also be stereo enabled. You can use red/blue stereo as well as shutter stereo, if your hardware supports stereo for OpenGL applications.

61. Does Amira support VR devices, such as 3D mouse, head-mounted displays or CAVE systems?

    The public version of Amira does not yet support these features. However, custom solutions have already been implemented, e.g. to support tracking devices. Contact `indeed@zib.de` if you have any requirements in this field.

62. Can I use anti-aliasing?

    Yes. On SGI Onyx2 Infinite Reality systems Amira will automatically use a multi-sample visual. On other systems you may use the command `viewer 0 antiAlias 3` to enable 3-pass jittered rendering. To reset, type `viewer 0 antiAlias 1`.

5.6 System Requirements

Amira 2.2 runs on Microsoft Windows 9x, Microsoft Windows NT, HP-UX 10.20, SGI Irix 6.5.x, Sun Solaris 2.7, and on Linux (RedHat 6.x, Suse 6.3).

Amira relies on fast hardware-accelerated OpenGL 3D graphics. We strongly recommend hardware texture mapping, since many visualization tools in Amira rely on it. Hardware texture mapping is available, for example, on SGI O2, Octane and Onyx systems, on HP workstations with fx/4 or fx/6 graphics, or on Sun Creator 3D and Elite graphics boards. For details on hardware acceleration, see below.

Apart from 3D graphics hardware probably, the most important system parameter is main memory. You should have at least 128 MB, preferably 256 MB or more. Amira can also be started with only 64 MB but working with large data sets definitely requires more main memory. Keep in mind that a single 3D image data set can easily occupy 60 MB of memory (240 slices with 512 x 512 pixels of 1 byte).

The speed of the processor of course is also an important parameter. However it is less critical than the graphics system and the main memory size. For the PC versions, we recommend at least a 500 MHz PIII processor.

5.6.1 On System Stability

Amira is a very demanding application that extensively uses high-end features. Experience shows that such applications tend to reveal instabilities in system hardware, hardware drivers, and the operating system. A common problem is insufficient main memory. We recommend you configure enough swap memory in addition to physical memory. The total amount of virtual memory should be at least 512MB. 1GB would be even better.

Especially on PC platforms, OpenGL drivers today are often not as robust as desired. Also, system crashes due to bad memory chips or unstable power-supply are not rare. If you experience problems or instabilities with Amira on your Windows platform, we recommend that you follow these steps:

1. Click on all the demo scripts in the Online User's Guide. If the system crashes, turn off hardware acceleration (choose the *extended* button from the Windows display settings dialog) and try again. If this eliminates the problem, there is a bug in your OpenGL driver. Try to get a new driver from the web site of the manufacturer of your graphics board.
2. Try using a different color depth in the Windows display settings dialog. Try 16, 24, or 32 bit.
3. Load the `lobus.am` data set and visualize it with a Voltex module. Turn on the spin rotation (turn it with the mouse in the viewer and release the mouse button while moving the mouse, so that the object continues moving). Let it run over night (turn off the screen saver). If the system has crashed or frozen the next morning, you probably have a hardware problem.

If this does not help, or if a reproducible error occurs on different computers, then it might be a bug in the

Amira software itself. Please report such bugs so that they can be eliminated in the next release.

5.6.2 Microsoft Windows

Amira runs on Intel or AMD-based systems with Microsoft Windows 98, Windows NT (NT4, Service Pack 4 or later), and on Windows 2000. If you are using Windows 95, be sure that the *Windows Socket 2 Update* is installed on your computer. Otherwise, the system will complain that `ws2_32.dll` is missing. You may download the update from `www.microsoft.com/windows95/downloads`. We recommend using Windows 98 or Windows 2000 instead of Windows 95.

Graphics Hardware: You should use a graphics board with OpenGL support (which is the case for almost all newer boards) and texture mapping capabilities.

5.6.3 Silicon Graphics

Graphics Hardware: To get optimal graphics performance, the machine should support *texture mapping in hardware*. Currently this is the case for all O2 systems, and for Octane systems with High Impact and Maximum Impact graphics (not Solid Impact). Amira provides a number of modules which make use of texture mapping, e.g. slicing, pseudo-coloring, or volume rendering. On machines without hardware texture mapping, these modules either run much slower or may not work at all. The advantage of the Octane is a higher speed in polygon rendering. For a complex model with an isosurface of 100,000 triangles, the frame rate is 10 per second for an Octane, compared to 3 per second for an O2. The MXE and MXI Octanes have larger texture memory than the SSE and SSI. Thus the MXE and MXI enable a direct volume rendering using 3D textures, which is not possible on an SE, SI or O2.

Software: The current version of Amira requires IRIX 6.5.x or higher. We recommend you install the newest version of the operating system (see `http://www.sgi.com/support`).

5.6.4 HP-UX

Amira performs pretty well on HP workstations equipped with Visualize fx/4, fx/6+, and fx/10 graphics cards under HP-UX 10.20. Probably it will run on other machines as well provided the OpenGL runtime environment has been installed. In any case we recommend to install the texture acceleration option for your graphics system (hardware texture mapping), especially if you intend to work with large 3D image data sets.

Important: By default some HP workstations are configured with a data size limit of 64 MB for each process. In order to load reasonable data sets, you should increase this value to 1 GB and the stack size to 128 MB. Do this by modifying the values in sam/Kernel Configuration `maxdsiz=0x80000000, ssiz=0x8000000, tssiz=0x8000000`.

The graphics patches PHSS_17160 (10.20 3D Common Runtime patch) and PHSS_17166 (10.20 OpenGL 1.1 Runtime patch) or later should be installed on your system, otherwise X Errors and program crashes can occur. You may download these patches from a HP support server, e.g. from `http://europe-support.external.hp.com` or `http://us-support.external.hp.com`.

Finally, Amira requires version 1.21 of /usr/lib/libCsup.sl to be installed on the system. Otherwise, an unresolved symbol error will occur. If you have an older version of this library, please install patch PHSS_17872 or some later replacement.

5.6.5 SunOS

Amira runs on Sun workstations with SunOS 5.7. Note that 1.2 OpenGL 1.1.1 106022-05 for Solaris or some newer version of OpenGL must be installed on the system. You may check the version using the following shell command:

```
strings /usr/lib/libGL.so | grep OpenGL
```

Amira is sucessfully being used on systems with Creator 3D and Elite 3D graphics boards. It runs on a simple Creator graphics boards as well. However, since no hardware texturing is available, performance is limited.

5.6.6 Linux

Amira 2.2 runs on glibc 2.1-based Linux systems. It has been tested on the following distributions: RedHat 6.2, Suse 6.3.

3D graphics hardware acceleration on Linux is a rather new and quickly evolving topic. Amira is known to work with the *3D Accelerated-X* servers from XI graphics `http://www.xig.com`, with the *XFree86 4* drivers from nVidia `http://www.nvidia.com` for TNT and GeForce based cards, and with SGI's drivers for their *SGI 230* linux workstation. In principle, Amira should work with other hardware accelerated OpenGL implementations as well.

The basic recipe to get hardware acceleration is to remove (or rename) the libraries libGL.so.x and libGLU.so.x in the `Amira/lib/arch-Linux-Optimize` directory, so that the hardware-accelerated libraries will be used instead. Sometimes not all features are accelerated. E.g. it is sometimes necessary to disable the stencil buffers (by starting Amira with the option -no_stencils).

In any case Amira requires an X server resolution of at least 1024x768 and at least 15 bit of color depth.

Please note that if software rendering is used, rendering performance may drop significantly, especially for visualization techniques like volume rendering.

5.7 Acknowledgements and Copyrights

We thank the Department of Genetics at the University of Würzburg, the Hermann Föttinger Institut of the Technical University of Berlin and the Rudolf-Virchow Klinikum Berlin for providing demo data sets.

The Amira software project was started at the scientific visualization group at Konrad-Zuse-Zentrum Berlin (ZIB). Since 1999 Amira is being jointly developed further by ZIB and by Indeed - Visual Concepts GmbH, a spin-off company from ZIB. Amira is based on the latest release of the Open Inventor toolkit from TGS Template Graphics Software, Inc.

Indeed - Visual Concepts GmbH has selected TGS Template Graphics Software, Inc., as the worldwide distributor for Amira.

Copyright (C) 1995-2000 Konrad-Zuse-Zentrum Berlin (ZIB)
Copyright (C) 1999-2000 Indeed - Visual Concepts GmbH, Berlin
Copyright (C) 1999-2000 Template Graphics Software, Inc., USA

Amira uses the following non-commercial libraries:

- The Tcl library developed by John Ousterhout, subject to following license terms:

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS

SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

- The libtiff library developed by Sam Leffler, subject to the following license terms:

Copyright (c) 1988-1997 Sam Leffler Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- The zlib library developed by Jean-Loup Gailly and Mark Adler.
- The JPEG input and output routines are based in part on the work of the Independent JPEG Group.
- On Linux, the Mesa OpenGL library developed by Brian Paul is used. The library is subject to the GNU Library General Public License (`http://www.gnu.org/copyleft/lesser.html`). It is linked as a shared object so you may replace or upgrade it without recompiling Amira.

Amira is a registered trademark of ZIB, Berlin.

OpenGL, Open Inventor, Silicon Graphics, and IRIX are registered trademarks of Silicon Graphics Inc.

Sun and Solaris are registered trademarks of Sun Microsystems, Inc.

Part III

Amira Reference Manual

Chapter 6

Alphabetic Index of Modules

6.1 Annotation

This module displays a 2D text in the 3D viewer. Position and color of the text can be specified by the user. The module is useful to annotate snapshots. For color legends refer to module *DisplayColormap*.

Ports

Color

Choose text color with the *ColorDialog*.

Position

Position of the text, measured in pixels from the lower left corner. If negative values are specified, position is measured from the top or right border, respectively.

Text

The text to be displayed.

6.2 Antenna

The *Antenna* module represents a treatment unit for regional hyperthermia. Currently, two different device types are supported, the BSD Sigma-60 applicator and the BSD Sigma-Eye applicator. However, note that the Sigma-Eye applicator cannot be used together with the *FDTD* module.

You may attach the antenna module either to a *LabelField* or to a *TetraGrid*. In the former case you can use the *FDTD* module to compute a SAR distribution using a straight-forward finite difference method. In the latter case you may also use the *FDTD* module and invoke a modified finite difference method. Alternatively, you may proceed with the *BolusGrid* module, which allows you to perform highly accurate treatment planning using a finite element method, c.f. *E-Field Simulation*.

In any case, in order to obtain realistic results the applicator should not exceed the upper and lower bound of the patient model. In addition, the tumor should be located somewhere in the middle of the applicator.

Connections

Data
A LabelField or a tetrahedral grid which defines the patient model to be used in a subsequent E-field simulation step.

Notice, that a tetrahedral grid must contain duplicated nodes if you want to use the *FDTD* module later on. Therefore, you might have to use module *Duplicate Nodes*. On the other hand, if you want to proceed with *BolusGrid*, the tetrahedral grid must not contain duplicated nodes!

Ports

X-Shift

The x position of the center of the applicator in cm.

Y-Shift

The y position of the center of the applicator in cm.

Z-Shift

The z position of the center of the applicator in cm.

Incline

Inclination angle of the applicator around the x-axis. This is measured in degrees. When using large inclination angles, make sure that the applicator doesn't intersect with the patient model.

Applicator

This port lets you switch between the BSD Sigma-60 applicator and the BSD Sigma-Eye applicator.

Show

This port provides two toggle buttons which allow you to individually turn on and off display of antennas and water bolus. Notice, that these buttons have no effect if the main viewer toggle of the module is off (sphere icon right from module name must be orange).

Reset

Resets the applicator shifts so that the treatment unit is centered around the middle of the patient model.

6.3 Arbitrary Cut

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

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

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

Connections

Data `[required]`

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

Ports

Orientation

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

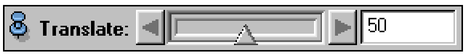
Options

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

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

Finally, 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.

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 port *Data*.

`getPlane`

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

`setPlane <origin uVec vVec>`

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

6.4 Arithmetic

The computational module *Arithmetic* does calculations on up to three input (data) objects according to a user-defined arithmetic expression. The result is written into a new data object named *Result*. The calculations are triggered by a *Doit* button. The arithmetic expression is evaluated either on appropriate points of the domain underlying the first input data object (e.g. for points of a grid on which a scalar field is defined) or on a regular 3D unit grid for which the number of points can be set by the *Resolution* port. Input data are referenced by variables A, B, and C, this implies pointwise evaluation such that the result for a point (x,y,z) depends on input values for 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 input object A.

B The values of input object B.

C The values of input object C.

I For the resulting object based on a regular grid, these are the cell numbers of the x dimension, numbered 0, 1, ... In case of a tetrahedral grid I denotes the tetrahedra numbers.

J The cell numbers of the y dimension.

K The cell numbers of the z dimension.

X, Y, Z The X-, Y-, and Z-coordinates in the resulting object.

R The radius $r = \sqrt{X^2 + Y^2 + Z^2}$

This is a list of available mathematical and logical operators:

`+ - / *` The basic mathematical operators.
`!` unary negation.
`-` unary minus.
`%` The modulo operator.
`> < <= >= != ==` The basic comparison operators stand for *greater, less, less equal, greater equal, not equal, equal*. If the comparison is true the result of these operators is 1 otherwise 0.
`&& || ^` The logical operators *and, or, xor*, the result is either 1 or 0.

There are also some builtin functions:

pow() Power function (with two arguments), note that there is no power operator.

sin() cos() tan() Trigonometric functions.

asin() acos() atan() Inverse trigonometric functions.

sqrt() Square root.

ln() exp() Logarithm and exponent.

For better understanding some examples on the usage of the *Arithmetic* module follow:

A - B The result is the difference between input objects A and B.

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.

A The result is a copy of input object A. If A is based on a tetrahedral grid and the result type is set to *regular* together with an appropriate resolution this expression leads to a conversion from an unstructured tetrahedral grid to a structured regular grid.

Connections

Data `[required]`

Input data object of type *Scalar Field*. InputA is the master input object, i.e. the type of the resulting object is determined by the type of the object connected to port InputA.

InputB `[optional]`

Input data object of type *Scalar Field*.

InputC `[optional]`

Input data object of type *Scalar Field*.

Ports

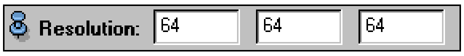
Expr

Here you can type in an expression for the computation of the resulting object.

ResultType

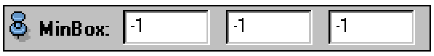
With this radio box the type of the resulting object can be set either to the type of the object connected to port InputA or to a regular scalar field.

Resolution



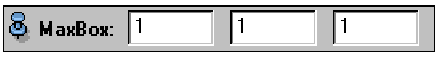
If port *Result Type* is set to *regular* the resolution of the regular field to be generated is set to the values given here.

MinBox



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

MaxBox



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

Action

Pushing this button triggers the computation.

6.5 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 object pool.

Connections

Data [optional]

Data object the axes should be adapted to.

Ports

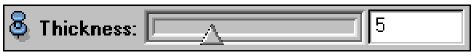
Axis

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. On default, the x-, y-, and z-axes are drawn in red, green, and blue, respectively.

Commands

getFontName

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

setFontName <name>

Lets you change the text font.

getFontSize

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

setFontSize <size>

Lets you change the size of the text font.

getBoundingBox

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

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

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

6.6 BolusGrid

In order to simulate electromagnetic fields in regional hyperthermia using the finite element method, a tetrahedral grid is required which not only covers the patient area but also the surrounding space, which includes the water bolus of the hyperthermia applicator. In order to create such an *extended grid* three steps have to be performed:

1. Create a surface representation of area outside the patient.
2. Fill the area outside the patient with tetrahedrons.
3. Combine patient grid and grid of outside area, this yields an *extended grid*.

The *BolusGrid* module lets you perform the first step. It has to be attached to an *Antenna* module. Taking into account the type of the applicator as well as its specific adjustment the module generates a *surface*, which contains the water bolus and a surrounding sphere and which exactly fits to the patient model that the applicator is attached to.

Sometimes it happens that in the resulting surface the bolus end caps intersect triangles of the patient model boundary. You can check this using the *Surface Editor*. In case you found intersections you may try to reduce the value of port *Offset*. Smaller offsets typically produce less intersections. You may also try to reposition the applicator sligthly. In general, intersections occur less often if in creating the patient model the option *preserve slice structure* of the *Simplification Editor* has been selected.

Caution: If you want to use the *Surface Editor* to remove the intersections, you have to be *very careful*. You must not change any triangles of the patient's surface at this time because they are prescribed by the patient model.

Connections

Data
This port connects the *BolusGrid* module with an *Antenna* module.

Ports

Smooth

This port controls two different smoothing strategies. If toggle *boundary* is set, the curve where the water bolus is attached to the patient is smoothed somewhat. If toggle *End caps* is set, the triangles of the bolus end caps are smoothed, i.e., vertices are shifted such that the triangles become more equilateral. Usually, both toggles should be on.

Option

If toggle *Flip Edges* is set, the triangular grids of the bolus end caps are improved by edge flipping.

Offset

This defines where the water bolus touches the patient's skin. The curve where this happens is shifted inwards by the given distance. The offset is measured in centimeter.

Action

The *compute* button of the action port initiates computation of the bolus surface.

6.7 Boundary Conditions

The *Boundary Conditions* module visualizes boundary conditions defined in a tetrahedral grid for a *finite elements* simulation.

Visible faces are stored in an internal buffer similar to the *GridVolume* module. Likewise, the selection domain can be restricted interactively by adjusting a selection box. Ctrl-clicking on a face makes it invisible.

In Amira there are predefined boundary condition ids with special meanings. The meaning of an id depends on the type of simulation to be performed on the grid (E-field or temperature simulation).

- predefined boundary condition ids for E-field simulation

id 1: outer boundary ('open' boundary conditions)
id 11-59: antenna triangles (metallic boundary conditions)
id 61-99: triangles of antenna junctions

- predefined boundary condition ids for temperature simulation
  id 1: predefined temperature 37°C (Dirichlet condition)
  id 2: insulating surface (Neumann condition)
  id 3: boundary to water bolus (Cauchy condition)
  id 5: boundary to air (Cauchy condition)

Connections

Data `[required]`
The tetrahedral grid to be visualized.

Ports

Bound.cond.id

This port provides a menu where all boundary condition ids occuring 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.

6.8 Bounding Box

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

Connections

Data `[required]`
Any spatial data object.

Ports

Text

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

6.9 CalcAreaVolume

Calculates the area and volume of materials or patches.

Connections

Data `[required]`

... description ...

Ports

Mode

Allows the choice between a spreadsheet representing the materials or the patches a surface-type object consits of. The choice of the material port results in a spreadsheet exhibiting the names of the materials, the number of triangles they consist of, the surface-area of those triangles and the volume of the materials. The choice of patches results in a spreadsheet showing the patchnumber, the materials they form part of, the number of triangles and the surface area of the patch.

Action

... description ...

6.10 CastField

This is a computational module that allows you to change the primitive data type of a regular 3D scalar field or of an RGBA color field. A new scalar field will be created having the same dimensions and the same coordinates as the incoming one, but the data values will be shifted, scaled, and casted 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 occuring in the incoming field a corresponding material will be created. Material colors may be defined via an optional colormap.

Connections

Data `[required]`

The scalar field or color field to be converted.

Colormap `[optional]`

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

Ports

Info

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

Output Datatype

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

Scaling

Defines a linear transformation which is applied before the data values are clamped and casted to the output datatype. The transformation is performed as follows: `output = SCALE*(input+OFFSET)`
If you want to convert data with a range `inmin ... inmax` into a range `outmin ... outmax`, `SCALE` and `OFFSET` are determined like this: `SCALE = (outmax - outmin) / (inmax - inmin)` and `OFFSET = outmin / SCALE - inmin`

Options

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

Colormap

Port to select a colormap.

Color Channel

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

Action

Start data conversion.

6.11 Clipping Plane

The clipping plane module is an instance of the *Arbitrary Cut* module. Refer to its *documentation*.

6.12 ColorCombine

This module combines up to three source fields into a HxUniformColorField3. The resulting field has the same dimensions as the field connected to source1. Therefore it is imperative that there is a source1. The CombineField connects to colorfields, scalarfields and scalarfields with a colormap. Relative scale of the input fields is determined by their bounding boxes. The module supports three different ways of merging the input data. ”Average” is the simple geometrical average of the inputs. ”Add & Clamp” adds the values, making sure they do not exceed 255. ”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. 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.

Connections

Data `[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.

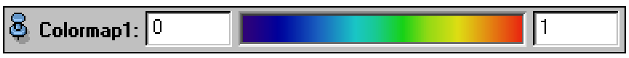
Colormap1 `[optional]`
The colormap for source field 1.

Colormap2 `[optional]`
The colormap for source field 2.

Colormap3 `[optional]`
The colormap for source field 3.

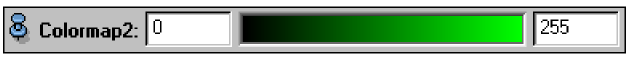
Ports

Colormap1



Colormap input port for source field1.

Colormap2



Colormap input port for source field2.

Colormap3

Colormap input port for source field3.

Mode

This option menu lets you choose the combining algorithm. The choices are: geometrical average, simple addition and a addition weighted in relation to the corresponding alpha values. This port is only visible, if Colormode 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, B compoment of the result, respectively.

Action

Pushing this button triggers the field computation.

6.13 Colorwash

The *Colorwash* module helps you to visualize two scalar fields in combination, e.g., medical images like CT data and a temperature distribution. It can also be used for *fusion* of medical images from different modalities. The first scalar field is mapped to a greyscale, the second is mapped to colors.

Apply an *OrthoSlice* or an *ObliqueSlice* module to the first scalar field, which must be a uniform or stacked field. Select the green icon of the second scalar field and invoke *Colorwash* from the popup menu of the *OrthoSlice* or *ObliqueSlice* module.

The map used to determine the colors is a succession of three maps, namely a (linear ramp) map of the scalar value range or a subrange thereof to an index set, e.g. [0, ..., 255], a map of the index set to RGB values by a *Colormap* connected to the *Colorwash* module by an input port, and a multiplication of an RGB value with a grey value from the underlying *OrthoSlice* module. In the example above the predefinied colormap *temperature.icol* is connected to *Colorwash* automatically, but the connection is not shown in the Object Pool display area. This happens whenever one of the predefined colormaps is used, otherwise you have to load your own colormap and connect it to *Colorwash*. If you want to make changes in the input colormap invoke the *Colormap Editor* on the *Colormap* data object, the *Colormap* port of *Colorwash* lets you set or replace the current input *Colormap* data object only; it also provides two sliders for setting *Min* and *Max* values that define the subrange of the scalar data as required in the first mapping step.

Connections

Data `[required]`

The second scalar field to be visualized via a colormap.

Module `[required]`

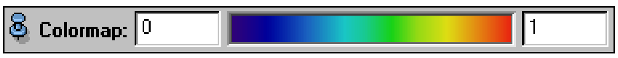
Connection to the *OrthoSlice* or *ObliqueSlice* module.

Colormap `[optional]`

Connection to a *Colormap*.

Ports

Colormap



The input *Colormap*. If no *Colormap* is connected to the module the port’s default color is used, i.e. a *Colormap* with constant value. To change this color click into the color bar with the left mouse button, this brings up a color selection dialog. To connect the port to a *Colormap*, use the popup menu under the right mouse button. See also *PortColormap*.

Mode

The Colorwash module offers two modes: In the (default) *multiply* mode, the RGB values from the colormap are multiplied with the grey-values in the OrthoSlice as described above. In the replace mode, the grey values in the OrthoSlice are completely replaced by the colors. it is sometimes useful to switch fore and back between these two modes.

6.14 CombineLandmarks

This module merges an arbitrary number of *HxVertexSet* objects into one *HxLandmarkSet*.

Connections

Data `[required]`

Accepts a VertexSet object. As soon as this port gets connected to a data object, an additional source port will appear. This allows the merging of a practically unlimited number of data objects.

Source2 `[optional]`

See above.

Ports

Unify Sets

If this option is chosen, the resulting landmark set will only contain one set of landmarks. Otherwise, the input vertices of the individual input data sets will be mapped to different sets.

Action

Triggers the computation.

6.15 ComponentField

This class is a special data object which helps you to analyze uniform complex scalar fields. The class itself is a uniform real scalar field. However, it can be attached to a complex scalar field like an ordinary display module. The component field provides an option menu allowing you to select which real quantity should be computed from the complex input values. These real quantities may be visualized using standard modules like *OrthoSlice*, *Isosurface*, or *Voltex*.

Connections

ComplexField `[optional]`

Complex scalar field for which a real quantity should be computed.

Ports

Component Field

Specifies the real quantity to be computed. Possible choices are *Real Part*, *Imaginary Part*, *Magnitude*, and *Phase*. The phase is defined in radians ranging from $-\pi$ to π.

6.16 Compute SAR

In simulating regional hyperthermia, the electric field created by a hyperthermia applicator can be calculated by superposition of the fields corresponding to the single channels (see module *Superpose*). Module 'Compute SAR' computes the SAR distribution corresponding to an electric field. Strictly speaking, the ARD distribution (power per volume) is calculated.

Connections

Data

An electric field of a hyperthermia applicator, as created by module *Superpose*.

Ports

Mode

Select between output of the magnitude (absolute value) of the electric field and output of the corresponding SAR distribution.

Action

Press button 'DoIt' to start the calculation.

6.17 ComputeContours

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

Connections

Data `[required]`

The label set to extract contours from.

Ports

Orientation

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

Options

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

Materials

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

Action

Triggers the computation.

6.18 Connected Component Analysis

This module searches for 6-connected regions in an image volume. Regions are detected based on thresholding. A region is a set of adjacent voxels, with a certain intensity. This module is e.g. useful to count cells on a dark background.

Only input fields of type byte are supported.

Connections

Data `[required]`

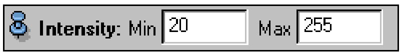
Image dataset to be analyzed.

Ports

Info

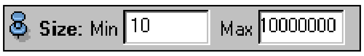
After hitting *DoIt*, this port will display the number of detected regions, the volume range and the average volume.

Intensity



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

Size



Minimal and maximal size in voxels of regions to be considered. Regions smaller or larger than this range are considered part of the background.

Output

If *Region field* is checked, a scalar field of type byte will be created, in which those voxels, which were considered to belong to a region are set to non-zero, while background voxels are set to zero. Voxels belonging to non-connected regions will be assigned different colors. The detected regions can be visualized in 3D by using e.g. an *Isosurface* module with threshold *0.5* on this output. Alternatively, the result can be casted to a Labelfield (using the *CastField* module) and then the *GMC* can be applied.

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

Action

Triggers the computation.

6.19 ContourView

This module displays contours of a surface. Contours are typically computed automatically and they are defined to be the boundaries of patches. If a surface contains no contours, this module will not display anything. You can automatically compute contours by using the `recompute` command of the *Surface*.

This module is mostly useful for developers.

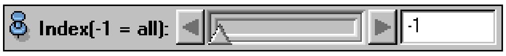
Connections

Data `[required]`

The underlying surface data.

Ports

Index)

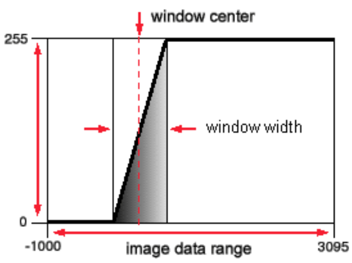


Choose index of contour to display. If -1 is choosen, all contours will be shown.

6.20 ContrastControl

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

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



The *window*'s center and width can be modified with the two sliders in the working 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 a fast modification of the *window*'s center *and* width, experienced users can directly use the mouse. After putting the viewer into interaction mode (indicated by an arrow shaped cursor) 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.

For an overview of the current mapping, a graphical representation of the mapping function can be displayed within the viewer (see *PortWindow*). Therefore the toggle button in the working 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* .

Connections

Module `[required]`

Connection to the *OrthoSlice* module.

Data `[required]`

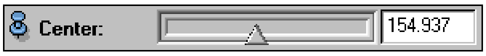
Connection to the scalar field is automatically established via the OrthoSlice module.

Ports

Linear mapping

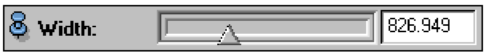
This port displays information on the image data range. In verbose mode the current data window will be displayed, too.

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.

Commands

`verboseMode {0|1}`
Setting verboseMode to a value not equal to zero leads to a permanent refresh of the current data window settings via the mapping port. Due to the possibilities of changing the window settings quickly this leads to flickering results and furthermore reduces the interactive adjustment speed. A value of 0 is the default.

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

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

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

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

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

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

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

6.21 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.

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

Connections

Data `[required]`
Vector field defined on a uniform grid (*UniformVectorField3*). The vector components must be floats.

Ports

This module has no ports.

6.22 DataProbe

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

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

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

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

Plane

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

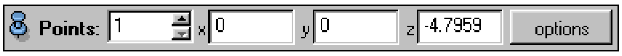
Options

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

Evaluate

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

Points



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

Control

With this radio buttons you can choose which of the following two ports are taken to compute the sample points. This port is not shown for module *PointProbe*.

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.

Show

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

Commands

getInterpol
Returns the currently used interpolation method.

setInterpol {none|linear|spline}
Sets the interpolation method. none means no interpolation at all and the sample values are taken at the controlpoints only.

getOrder
Returns the order of the spline probe.

setOrder <value>
Sets the order of the spline probe.

getPoint [<index>]
Returns the coordinate of the requested controlpoint. Index defaults to 0.

setPoint [<index>] <x> <y> <z>
Sets the coordinates of controlpoint index. Index defaults to 0.

getSamplePoints
Returns the coordinates of all points where samples are taken.

getSampledValues
Returns all sampled values.

setImmediate {0|1}
Switches the immediate mode on or off, i.e. data is shown while the probe line resp. point is being moved.

setOrtho {0|1}
Switches the orthogonal mode on or off, i.e. all controlpoints of a probe line are moved in sync or not.

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

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

appendPoint <x> <y> <z>
Appends a controlpoint with the given coordinates.

insertPoint <index> <x> <y> <z>
Inserts a controlpoint with the given coordinates as the indexth point.

removePoint <index>
Removes the given controlpoint.

6.23 Delaunay2D

This module takes a set of 3D vertices and produces a Delaunay triangulated surface with 2D topology. In order to do so the vertices are temporarily being projected into the xy-, xz-, or yz-coordinate plane. The incoming data object must be derived from a vertex set, cf. Section 4.2.5. Note, that the Delaunay algorithm may take some time, especially for large data sets. If the input is not of planar topology, like e.g. a sphere, the result will probably not be useful. If multiple 3D vertices project to the same 2D vertex, the result is undefined. In the latter case, it might be help to issue a jitterPoints 0.001 0.001 0.001 command on the command line.

Connections

Data [required]

The vertex set to be triangulated.

Ports

Projection Plane

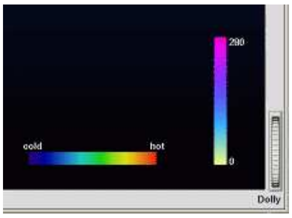
Lets you select whether the triangulation should be computed in the xy-, xz-, or yz-coordinate plane. You may use the axis module in order to find out which plane is most appropriate for your data.

Action

Starts computation.

6.24 DisplayColormap

This module allows you to position a colormap-icon onto the 3D viewer. This is useful e.g. to produce snapshots, that shall 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 Edit/Show menu of the main window to bring their icons up.



Connections

Data `[required]`

The colormap to be displayed.

Ports

Options

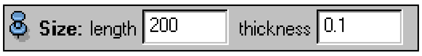
This port provides the following three toggles:

- **custom text** Whether or not custom text should be displayed. See below for details.
- **vertical** Vertical orientation instead of horizontal.
- **relative size** Change size of displayed colormap when viewer size changes.

Position

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

Size



Size in pixels (or relative if option is selected) and thickness of colorbar.

Custom Text

This is only available if *custom text* option is selected. You may specify a space separeted list of values, that shall be displayed. In addition you may specify a text, which 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 one of the labels should contain blanks you have to enclose the text in double quotes, e.g. `100/"very hot"`.

6.25 DisplayISL

This module visualizes a 3D vector field using so-called illuminated field lines. This technique was first presented on the *Visualization 96* conference. More details are described in *M. Zöckler, D. Stalling, H.-C. Hege, Interactive Visualization of 3D-Vector Fields using Illuminated Streamlines, Proc. IEEE Visualization '96, Oct./Nov. 1996, San Francisco, pp. 107-113, 1996.*

The module computes a large number of field lines, by integrating the vector field starting from random seed points. The lines are displayed using a special illumination technique, which gives a much better spatial understanding of the fields structure than ordinary constant-colored lines. In order to execute a script demonstrating this module click here.

The functionality of *DisplayISL* can be extended by means of the *SeedSurface* module.

Connections

Data `[required]`

The vector field to be visualized. Since a procedural data interface is used all types of vector fields are supported (e.g. fields on regular or curvilinear grids, as well as procedurally defined fields).

ColorField `[optional]`

An optional scalar field which, if present, will be used for pseudo-coloring. Often it is useful to create a scalar field from the vector field using the *Magnitude* module.

AlphaField `[optional]`

An optional scalar field which can be used to control the lines' opacity locally.

Distribution `[optional]`

An optional scalar field which can be used to control the distribution of seed points.

Colormap `[optional]`

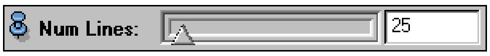
The *colormap* used to encode the color field. Will only be visible if a color field is connected to this module.

Ports

Colormap

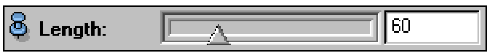
Optional colormap.

Num Lines



Number of field lines to be displayed. The technique typically gives the best results with a rather large number of lines (e.g. 100-500). Note however that on systems with slow 3D graphics without texture hardware, a 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. On default lines are traced the same distance in forward and backward direction. You may use the command `setBalance` to change this behavior.

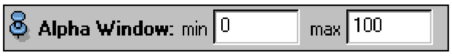
Opacity

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 or smaller than *min* field lines will be completely transparent. Likewise, at locations where the alpha field is equal or bigger than *max* field lines will be completely opaque.

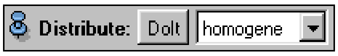
Options

Fade: Enables fading mode as described above.
Texture: Controls illumination of lines. If off, constant colored lines are drawn (flat shading).
Animate: Activates particle-like animation. The animation speed may be controlled via the command `setAnimationSpeed`.

Seed Box

Clicking on *XformBox* or *TabBox* brings up a 3D dragger in the 3D Viewer. This allows you to restrict the region of interest, i.e. the region in which seed points are placed, by interactively transforming the dragger (Remember to switch the *viewer* into interaction mode by hitting ESC). After changing the box you have to hit *DoIt* to trigger recalculation.

Distribute



On the one hand, this port provides a *DoIt* button which 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 have to press *DoIt* in order to update the display.

On the other hand, the port also provides an option menu specifying the way how seed points are distributed inside the seed box. On 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.

6.26 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.

$$\mathrm{div} V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$$

Connections

Data `[required]`

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

Ports

This module has no ports.

6.27 Duplicate Nodes

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.

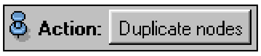
Connections

Data `[required]`

A tetrahedral grid without duplicated nodes.

Ports

Action



Button *duplicate nodes* causes a new output grid with duplicated nodes to be computed.

6.28 E-Field Simulation

The *E-Field Simulation* module performs an E-field calculation using the *finite element* method. Before starting the E-field calculation, you must create an *extended tetrahedral grid* by combining a patient grid and a grid representing the exterior space. This can be done using module *Combine*. The grid representation of the exterior space itself is generated using modules *Antenna* and *BolusGrid*.

When you have created an extended grid, load it into HyperPlan. Then you can select the *E-Field Simulation* module from the grid's popup menu. E-field calculation will take ca. 10-20 minutes CPU time per channel of the applicator. It is performed as a *batch job*. The status of the job queue can be controled in the *Job Dialog*.

Connections

Data

An extended tetrahedral grid. Usually such a grid has the suffix `ExtendedGrid`.

Ports

Directory, EFieldGrid, EFieldVals

The *E-Field Simulation* module creates two output files, an E-field grid and a file containing a set of E-field distributions The E-field grid is very similar to the tetrahedral patient model (suffix `grid`). However, it additionally contains a list of the edges occuring in the grid.

This ports allows you to specify the names of the output files. HyperPlan makes a suggestion for you, but you can change it as you please. It is recommended to use the suffixes `EFieldGridFE` and `EFieldValsFE` in the filenames.

Frequency

Here you can enter the frequency (in MHz) of the electromagnetic waves. Recommended values are 90 MHz for the Sigma-60 and 100 MHz for the BSD Sigma-Eye applicator.

Precision

This port is currently not used.

Options

If toggle *Save total field* is set, two additional files are created (`EFieldGrid` and `EFieldVals`) which contain the fields in the whole extended grid. The filenames contain an additional suffix `Total`. This option is for testing purposes only.

Action

If you press the *Materials* button an editor window appears where some important material constants for the E-field calculation are presented. Note: *the electrical parameters stored in HyperPlan's material database refer to a frequency of 90 MHz*. Changes can be made by simply selecting an item and typing a new value. The 'ok' button in the editor window saves the changes. After you have setup the simulation, you can commit it by pressing the *Submit* button. If one of the specified output files did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, 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.

6.29 FDTD

The FDTD module calculates an electric field using a finite difference algorithm and stores the result in a given file. Since an E-field simulation takes several hours of computing time the calculation is performed as a batch job. The status of the job queue can be controled in the *Job Dialog*.

The results of E-Field simulation can be visualized later on. See section 1.5.1 for an overview on performing E-Field simulations using the finite difference method. The FDTD module works on both voxel grids *(FDTD Voxel-Solver)* as well as on tetrahedral grids *(Modified FDTD-Solver)*. It must be attached to an *Antenna* module which in turn is connected to the voxel grid resp. tetrahedral grid. Currently only the BSD Sigma-60 applicator is supported by this module.

Note: The maximal possible number of cubes (Yee cells) is 82 in x and y directions and 106 in z direction. The number of cubes should be chosen such that 5-10 cells of background medium lie between the applicator/patient bounding box and the boundary of the FDTD lattice, i.e., the number of cubes should be set to about 80 for a cube size of 1 cm and about 48 for a size of 2 cm.

The program will stop if the chosen number of cubes is too small to cover the patient's and the applicator's geometry. The number of iterations should be chosen as at least 2500 for size of 1 cm and at least 1500 for a size of 2 cm. These iteration numbers proved to be sufficient for modelling the BSD Sigma-60 applicator in the frequency range 80-120 MHz.

Connections

Data

Connection to the *Antenna* module.

Ports

Directory, EFieldVals

These ports allow you to specify the directory and the file name under which the resulting E-fields will be stored. HyperPlan makes suggestions for you, but you can change them as you please. It is recommended to use the suffix `EFieldValsFD` in the filename.

Options

In the first text field (*max iter*) you can type in the number of time steps for the FDTD calculation.

For restrictions and recommendations, see above. In the second text field (*MHz*) you can define the frequency (in Mhz). It is best to choose a frequency between 80 and 120 Mhz.

Cubes

Here you can type in the number of the cubic cells (Yee cells) in the directions x, y and z. For restrictions and recommendations, see above.

Cube Size

Here you can specify the size of the cubic FDTD-cell (so-called Yee cell) in centimeters. For the FDTD Voxel-Solver the size is restricted to the single or double distance between the CT-slices. If the modified FDTD on a tetrahedral grid is used, the size can be set arbitrarily. However, it is recommended to use a size of 1 cm. For restrictions and recommendations, see above.

Action

If you press the *Materials* button an editor window appears showing the material constants for the E-field calculation. Note: The electrical parameters stored in HyperPlan's material database refer to a frequency of 90 MHz. Changes can be made by simply selecting an item and typing a new value. The *Ok* button of the editor window saves your changes.

After you have setup the simulation, you can commit it by pressing the *Submit* button. The *Job Dialog* window should appear, showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

Display

A bounding box of the calculation domain with size given by NumCubes times Cube Size is shown in the 3D viewer by pressing the *box* toggle.

6.30 Field Cut

The *Field Cut* module is a tool for visualizing cross sections of a 3-dimensional scalar field defined on a tetrahedral grid. In general such grid volumes are made up of several materials. A (2-dimensional) cutting plane has to be defined for this purpose and may be shifted through the (3-dimensional) tetrahedral grid in orthogonal direction, an *Orientation* port is provided for setting the orientation of the cutting plane perpendicular to one of the main axes and a *Translate* port for specifying an orthogonal translation. *FieldCut* renders a slice with the values of the scalar field mapped to colors of a connected colormap. This is called *pseudo coloring*. The cutting plane is usually clipped to a rectangle. With the 3D viewer turned into interactive mode you can pick one of its edges and shift it through the grid volume. To define arbitrary cutting planes you have to set the *rotate* toggle which makes a rotation handle appear in the center of the cutting plane. Having turned the viewer into interactive mode you can pick the handle with the left mouse button and rotate the plane as you please.

Connections

Data `[required]`

The field of type *TetraScalarField3*

Colormap `[optional]`

An optional colormap used for pseudocoloring the values of the scalar field on the cutting plane.

Ports

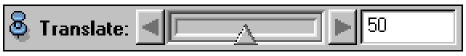
Orientation

This port provides three buttons to reset the slice orientation. Axial slices are perpendicular to the z-axis, frontal slices are perpendicular to the y-axis, and sagittal slices are perpendicular to the x-axis. See also *PortButtonList*.

Options

If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

Colormap

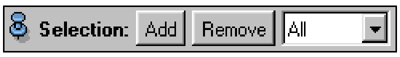
See also *PortColormap*.

Interpolation

In pseudo-color mode you can switch between two rendering methods by the radio buttons *Gouraud* and *Texture*. See also *PortRadioBox*.

- **Gouraud** shaded faces have their colors assigned at their vertices. The colors in between are linearly interpolated. Therefore misleading colors occasionly 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.

6.31 GMC

This module computes a triangular approximation of the interfaces between different tissue types in a *LabelField* with either uniform or stacked coordinates. This is done by means of a generalized marching cubes algorithm, which explains the letters GMC. The resulting surfaces can be non-manifold surfaces.

Depending on the resolution of the incoming LabelField the resulting triangular surface may have a huge amount of triangles. Therefore the LabelField should be resampled to 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 GMC module can use the probability information to generate smoother surfaces.

Connections

Data `[required]`

LabelField from which the interfaces should be extracted.

Ports

Options

This port provides two toggles. If *sub-voxel accuracy* is on the probability information of the incoming LabelField will be used to generate smoother surfaces. If this option is off or if the LabelField does not contain probabilities, no such smoothing is performed.

The second 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.

Border

This port will only be visible if *add border* has been selected. It provides two toggle buttons labeled *adjust coords* and *extra material*.

It the first option is selected points belonging to triangles adjacent to boundary voxels will be moved exactly onto the nearest boundary face of the bounding box. In this way the resulting surface appears to be sharply cut off at the boundaries.

The second toggle indicates that triangles adjacent to boundary voxels will be inserted into separate patches. The outer region of these patches will be called *Exterior2*. If the toggle is off no such extra material will be created. Instead, the boundary is assumed to be labeled with 0, which usually corresponds to *Exterior*.

Minimal edge length

A non-vanishing value indicates that short edges of the final GMC 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. On default, values between 0 and 0.8 can be entered. Typically, a value of 0.4 already yields good results. However, note that intersections may be introduced during edge contraction. If you want to avoid this try to use the *simplification editor*. The editor applies some special strategies in order ensure topological consistency.

Action

This port has a single button *Triangulate* which is used to start surface extraction. Depending on the size of the LabelField the algorithm requires up to a minute to finish.

6.32 GetCurvature

This module computes curvature information for a discrete triangular surface of type *HxSurface*. 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.

Connections

Data `[required]`

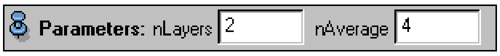
The surface for which curvature information should be computed.

Ports

Method

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

Parameters



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

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

Output

This menu controls the output of the curvature module. If *curvature* is selected then a surface scalar field is generated containing the maximal principal curvature of a triangle or of a point. If *1/curvature* is selected then the curvature values are inverted. In this case the output values have the dimension of a length, indicating the radius of a sphere locally fitting the surface. If *max direction* is selected then a surface vector field is computed indicating the direction of maximum principal curvature. The length of a directional vector is equal to the corrsponding curvature value.

Action

Press this button to start computation.

6.33 Gradient

The *Gradient* module computes the gradient of a scalar field consisting of floats defined on a uniform grid. The output is a uniform vector field. The direction of the gradient vector depends on the setting of port *Output*. If *Force* is selected the negative gradient vector is computed.

$$\mathrm{grad}F = \pm\left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)$$

Connections

Data `[required]`

Scalar field defined on a uniform grid (*UniformScalarField3*). The scalar values must be floats.

Ports

Output

If *Force* is selected the negative gradient vector is computed.

6.34 Grid Boundary

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]`

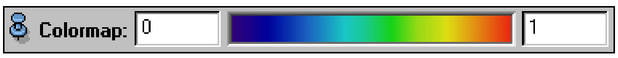
A colormap is used to map the data values of the optional scalar field.

Ports

Draw Style

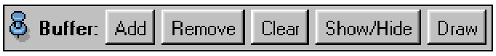
This port is inherited from the *ViewBase* class and therefore the description will be found there. In contrast to other modules derived from the *ViewBase* class this module does not provide the possibility to consider vertex and direct normals. The triangles will always be drawn flat.

Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

Buffer



The Buffer buttons give you some control of the internal face buffer of the GridBoundary module. Only the faces present in the buffer are displayed according to the current drawing style. The *Add* button adds the highlighted faces to the buffer. The *Remove* button removes highlighted faces from the buffer. The *Clear* button removes all faces from the buffer. The *Hide* button deselects all faces but does not change the buffer. See also *PortButtonList*.

Materials

This port provides two menus where all different materials of the input grid are listed. Faces which belong to the boundary between the selected materials are displayed using a red (highlighted) 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*.

6.35 Grid Cut

The *Grid Cut* module is a tool for visualizing cross sections of labeled tetrahedral volume grids made up of different 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. *GridCut* renders a slice with the material compartments in their associated colors. The cutting plane is usually clipped to a rectangle, with the 3D viewer turned into interactive mode you can pick one of its edges and shift it through the volume grid. 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 labelled tetrahedral volume grid to be visualized.

Ports

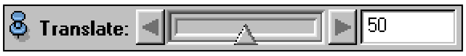
Orientation

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

Options

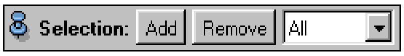
If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch on the rotate handle for the cutting plane and off again. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

Selection



This port maintains a list of materials 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.

6.36 Grid Volume

The *Grid Volume* module is a powerful tool for visualizing labelled tetrahedral volume grids, e.g. tetrahedral patient models. Views on various sub-grids can be specified, e.g. a view on a sub-grid corresponding to one of the tissue compartments of a patient model. The module maintains an internal buffer which contains all currently visible tetrahedra. Tetrahedra belonging to a particular compartment can be selected and added to the buffer. Selected tetrahedra are displayed using a red wireframe. Having been added to the buffer they are displayed in their associated colors. The selection can be restricted by means of an adjustable selection box. In addition, individual tetrahedra can be removed from the buffer by Ctrl-clicking on one of their faces, but notice that the viewer has to be turned into interactive mode for this. Similarly, a simple click on a face adds the tetrahedra in front of it to the buffer.

Connections

Data `[required]`

The labelled *tetrahedral grid* to be visualized.

ColorField `[optional]`

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

Colormap `[optional]`

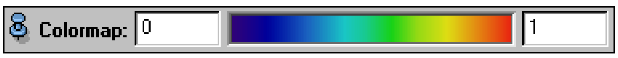
The colormap is used for pseudo-coloring the grid volume.

Ports

Draw Style

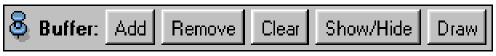
This port is inherited from the *ViewBase* class and therefore the description will be found there. In contrast to other modules derived from the *ViewBase* class this module does not provide the possibility to consider vertex and direct normals. The triangles will always be drawn flat.

Colormap



This port becomes visible only if a scalar field has been connected to the *ColorField* port. For further details see section *Colormap*.

Buffer



This port lets you *add* and *remove* highlighted triangles (being displayed in red wireframe) to an internal buffer. For a further description and for the functionality of each of the port buttons see *ViewBase*.

Materials

This port provides a menu where all materials of the input grid are listed. If you choose a material, all tetrahedra of that material are selected and displayed using a red wireframe model. You may crop the selection domain by turning the viewer into interactive mode and moving the green handles of the bounding box to a position of your choice. Only the tetrahedra inside the bounding box can be added to the buffer.

Color mode

Here you may choose among several color modes. For details see the module *SurfaceView*.

Commands

`getSelectedTetra`

Displays a run-length encoded list of all currently selected tetrahedra. The list consists of pairs of integer numbers. The first number is the index of a selected tetrahedron. The second value denotes the number of subsequent selected tetrahedra.

`setSelectedTetra <list>`

Adds some tetrahedra to the internal buffer so that they become visible. The argument is a run-length encoded list like the one displayed by getSelectedTetra.

6.37 GridView

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

Connections

Data `[required]`

The regular field to be investigated.

Ports

Orientation

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

Geometry

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

Slice

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

6.38 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.

Colormap `[optional]`

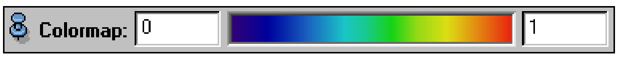
The colormap used to map data values to colors.

Ports

Orientation

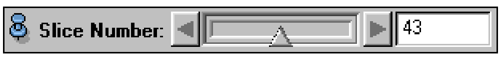
This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

Colormap



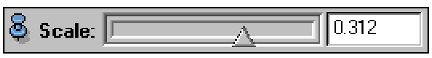
Choose colormap for the visualization.

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].

6.39 Histogram

The *Histogram* 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 *Grid Volume* 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]`

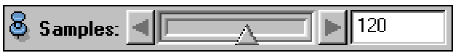
A *Grid Volume* module that selects the tetrahedra for which the histogram is calculated.

Ports

Histograms

If you select one of these toggles, a plot window appears showing the corresponding histogram for all tetrahedra selected by the Grid Volume 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.

6.40 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.

Ports

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.

6.41 Intersection

The *Intersection* module shows the intersection of a tetrahedral grid and a cutting plane. Depending of the setting of the *All* toggle only material boundaries are shown or the boundaries of all cut tetrahedra. Whenever this module is invoked from a popup menu of a data object, it comes together with a module *EmptyPlane* that constructs a cutting plane.

Connections

Data `[required]`

A tetrahedral grid has to be connected to this port.

Module `[optional]`

An empty plane is loaded automatically.

Ports

Lines

Toggles whether all or only material boundaries are shown.

Line Width

The linewidth of the boundary lines.

Selection

With the menu you can select the material which will be cutted on the cutting plane. Subsequent selections are added to the list of selected materials. With the *Clear* button the list can be resetted.

Commands

setColor <r> <g> <b>

Sets the color of the intersection lines.

6.42 Isolines

This module computes isolines for an arbitary 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]`

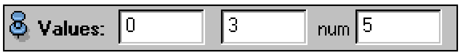
Colormap used for pseudo-coloring. If no colormap is connected all isolines have equal color.

Ports

Spacing

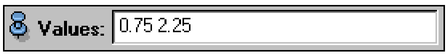
Control how isoline thresholds are being defined. In *uniform* a certain number of isovalues are distributed equidistantly within a user-defined interval. In *explicit* mode the threshold for each isoline has to be set manually.

Values



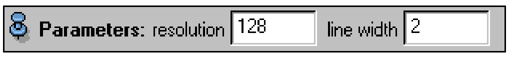
In *uniform* mode this port contains three text fields allowing the user to define the lower and upper bound of the isoline interval as well as the number of isolines being computed in this interval.

Values



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

Parameters



The *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

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

Port to select a colormap.

6.43 Isolines (Surface)

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

Connections

Data `[required]`

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

Colormap `[optional]`

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

Ports

Spacing

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

Values

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

Values

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

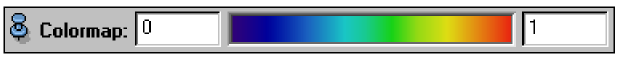
Parameters

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

Options

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

Colormap



Port to select a colormap.

6.44 Isosurface (Regular)

This module computes an isosurface within a three-dimensional scalar field with regular cartesian coordinates. The module contains optimized code for uniform coordinates, i.e. grids consisting of equidistant slices.

Very high-resolution datasets can be downsampled to reduce the number of polygons being produced. In addition, an internal polygon reduction method is provided that merges certain triangles of the original triangulation. This way, the number of triangles can be reduced up to 50%. A second independent scalar field may be connected to the module. This field determines how the isosurface is colored. If no color field is connected to the module the isosurface has constant color.

Connections

Data `[required]`

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

ColorField `[optional]`

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

Colormap `[optional]`

The colormap is used for pseudo-coloring the isosurface.

PointProbe `[optional]`

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

Ports

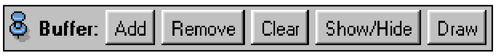
Draw Style

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

Colormap

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

Buffer



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

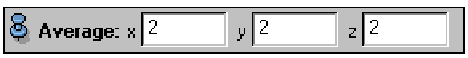
Threshold

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

Options

If the *compact cubes* 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.

Action

Press the *DoIt* button of this port to start the computation of the isosurface.

6.45 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.

Connections

Data `[required]`

The scalar field defined on a tetrahedral grid.

ColorField `[optional]`

See port *ColorField* in section *Isosurface (Hexahedra)*.

Colormap `[optional]`

See port *Colormap* in section *Isosurface (Hexahedra)*.

Ports

Draw Style

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

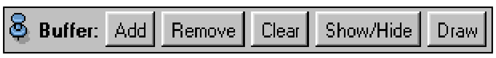
Colormap

See port *Colormap* in section *Isosurface (Hexahedra)*.

Threshold

Determines the value used for isosurface computation. The slider is automatically adjusted to cover the whole range of data values. If the threshold value is changed the computation of the new isosurface is immediately invoked.

Buffer



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

6.46 IvDisplay

The *IvDisplay* module renders an object containing *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 on of three draw styles to render the *Inventor* geometry.

6.47 IvToSurface

The *IvToSurface* computational module parses the connected *Inventor Scene Graph* (*Inventor* geometry data) and converts it into the surface format. This is done when you push the *DoIt* button. A green icon named *GeometrySurface* representing the generated surface should appear.

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

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

Connections

Data `[required]`

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

Ports

Relative Tolerance

Factor to determine duplicated points.

Options

If this toggle is set the triangle connectivity of the Surface data format is filled.

Action

The surface is generated if the *DoIt* button is pressed.

6.48 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 grey 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 grey 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 *DoIt* action 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 Editor* to further process this object.

Connections

Data `[required]`

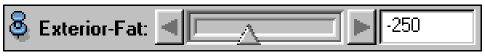
Image data to be segmented.

Ports

Regions

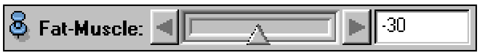
This port lets you specify the names of the different regions to be segmented. Between two and five regions may be specified. The indivdual region names must be separated by blanks.

Exterior-Fat



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

Fat-Muscle



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

Muscle-Bone

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

Options

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

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

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

Action

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

6.49 LandmarkView

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

Connections

Data `[required]`

The landmark set to be displayed.

Ports

Point Set

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

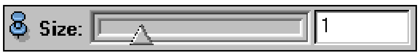
Lines

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 as spheres/geometry or as points. If the input contains a large number of points (several hundred, or thousands), point style can significantly improve display performance.

Size



Size of displayed spheres or points.

Complexity

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

6.50 Line Probe

See Section *Data Probing* for details.

6.51 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.

Connections

Data `[required]`

The line set to be displayed.

Colormap `[optional]`

Colormap used for pseudo-coloring.

Ports

Shape

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

ScaleMode

If the line set object contains additional data values per vertex 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.

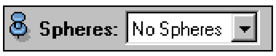
ScaleFactor

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

ColorMode

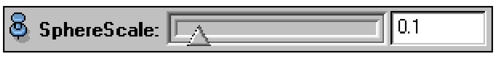
If the line set object contains additional data values per vertex this menu allows you to select one such variable which will be used to lookup vertex colors. If *No Color* has been selected the lines or tubes will be displayed in uniform default color. This color may be changed using the command `setLineColor`.

Spheres



On default *No Spheres* is selected. Changing this selection causes a sphere to be displayed at each line vertex. If the line set object contains additional data values per vertex there will be one entry for each 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

Like port *Color Mode*, but affects the sphere colors instead of the line colors.

Colormap

Optional colormap.

Commands

`setLineColor <color>`
Lets you adjust the default line color which is used if color mode is set to *No Color*. The color may be specified as an RGB color triple or as a X11 color name.

`setSphereColor <color>`
Lets you adjust the default sphere color which is used if sphere color mode is set to *No Color*. The color may be specified as an RGB color triple or as a X11 color name.

6.52 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.

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.

Action

Starts computation.

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.

6.53 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.

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.

Action

Pushing this button triggers the computation.

6.54 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.

Connections

Data `[required]`

Vector field or complex vector field defined on a regular grid (*RegVectorField3, RegComplexVectorField3*).

Vector field on an irregular grid, either specified by nodes of a surface or a tetrahedral grid (*TetraVectorField3, SurfaceVectorField*).

Ports

Number of Vectors

Shows number of input vectors for computing the magnitude field.

Compute

Pushing this button triggers the computation.

6.55 ObliqueSlice

The ObliqueSlice module lets you reconstruct arbitrarily oriented slices through 3D scalar fields of any type, as well as through 3D color fields. In the medical context such slices are also known as multi-planar reconstructions (MPR).

The module is derived from *ArbitraryCut*. See the documentation of the base class for details about how position and orientation of a slice can be changed. Like in the *OrthoSlice* module three different methods are supported to map scalar values to screen colors, namely linear mapping, monochrome mapping based on an adaptive histogram equalization technique, as well as pseudo-coloring.

Connections

Data `[required]`

The scalar or color field to be visualized.

Colormap `[optional]`

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

Ports

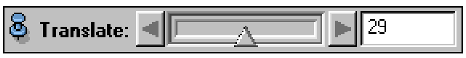
Orientation

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

Options

If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. With the *rotate* toggle you can switch the rotate handle for the cutting plane on and off. If the *immediate* toggle is set the slice is updated every time you drag it with the mouse in the 3D viewer. Otherwise only the bounding box of the cutting plane is moved and the update takes place when you release the mouse button.

Translate



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

MappingType

This option menu controls how scalar values are mapped to screen colors. In case of a *linear* mapping a user-defined data window is mapped linearly to black and white. If *historam* is selected, an adaptive histogram equlization technique is applied. This method tries to show all features of the data, even if a wide range of values is covered. Finally, *colormap* can be used to activate pseudo-coloring. Port *mapping* is hidden if the module is connected to a color field.

Data Window

This port is only visible 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.

ContrastLimit

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 only visible if *colormap* is selected. Choose a colormap to map data to colors.

Sampling

This port provides several toggles controlling the way how oblique slices are reconstructed. The first toggle labeled *fine* lets you switch between low and high sampling resolution. On default, 150x150 or 300x300 sample points are taken in these modes, respectively. These numbers can be changed via Tcl-commands. The next toggle, denoted *interpolate data*, will only be active if the scalar field to be visualized is defined on a uniform grid. In this case if the toggle is off nearest neighbor interpolation is used. If it is on or if the data is not defined on a uniform grid, then the field's native interpolation method is used, e.g. trilinear interpolation on regular grids. Finally, the toggle labeled *interpolate texture* controls how the slice is texture-mapped by the underlying OpenGL-driver. If the toggle is off, nearest-neighbor sampling is used. Otherwise, bilinear filtering is applied.

Overlay

This port determines how (optionally) attached *ColorWash* modules are mapped onto this slice.

6.56 OptTemp

The *OptTemp* module helps you to perform a temperature optimization for a tetrahedral patient model. The optimization procedure attempts to heat the entire tumor to 43°C and to keep the temperature in healthy tissue below 42°C. However in most cases these tasks cannot be fulfilled perfectly, i.e., there may be some underheated parts of the tumor and some hot spots in healthy tissue. For healthy tissue some absolute temperature limits are prescribed, which may depend on the tissue type. At the moment they are 44°C for most tissues and 42°C for some heat sensitive tissues like bladder and intestine.

Temperature optimization results in a set of power amplitudes and phase delays for the channels of the hyperthermia applicator, which are stored in a file (suffix `Plan`). Use module *Superpose* to load this file and inspect the optimized temperature distribution and the corresponding SAR distribution.

Summarized briefly, you have to perform the following steps for a temparatue optimization:

1. First you have to load a tetrahedral patient model as well as a set of temperature distributions into HyperPlan. Both are created by module *Static Heat*. The patient model files are called `ThermGridFD` or `ThermGridFE`, while the files containing the temperature distributions usually have a suffix `ThermValsFD` or `ThermValsFE`. The characters `FD` or `FE` represent the numerical method (*finite differences* or *finite element*) which has been used for electric field calculation.
2. Check if the patient model contains a region named 'Target', 'Tumor', or 'Tumour'. Optimization will not work if such a region does not exist.
3. From the popup menu of the `ThermVals` icon select module *Opt Temp*, this module lets you enter the name of the file containing the optimized parameters (amplitudes and phases) that are going to be computed. Depending on the name of the `ThermVals` file, it is recommended to use the suffix `PlanFD` or `PlanFE` in the filename.
4. If a simulation of the Sigma-Eye applicator is performed, you can select at port *Power Generators* whether 4 or 12 power generators are available.
5. Now you are ready to submit the job. Do this by pressing the *Submit* button. If the output file did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, the *job dialog* window appears showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

Connections

Data

A set of temperature distributions as created by module *Static Heat*. The data files usually have a suffix `ThermVals`.

6.57 OrthoSlice

The *OrthoSlice* module is an important tool for visualizing scalar data fields defined on uniform cartesian grids. For instance stacks of tomographic images are commonly displayed using OrthoSlice. Such data are visualized by extracting an arbitrary axial, frontal, or sagittal slice out of the volume. Within the slice the data are mapped to colors or grey levels by one of three mapping methods. The most simple mapping technique uses a linear grey ramp. In addition, a data range can be specified. This range determines which data values are mapped to black and which are mapped to white. Alternatively, a contrast limited histogram equalization technique may be applied. With this method, there is no unique correspondence between grey levels and data values any more. The method tries to visualize all features of an image. As a third mapping method, an external colormap can be used in OrthoSlice.

Connections

Data `[required]`

The scalar or color field to be visualized. Currently, uniform cartesian grids of type *HxUniformScalarField3*, *HxUniformStackedScalarField3*, and *HxUniformColorField3* are supported.

Colormap `[optional]`

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

Ports

Orientation

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

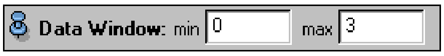
Options

If the *adjust view* toggle is set, the camera of the main viewer is reset each time a new slice orientation is selected. The *bilinear* toggle determines how color is interpolated within a slice. By default, the toggle is off and constant interpolation is used. With constant interpolation inividual grid cells become visible. Bilinear interpolation often produces somewhat blurry results.

Mapping Type

This option menu lets you select between the three different mapping methods, namely a *linear* grey ramp, *histogram* equalization, and *colormap* mode.

Data Window



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

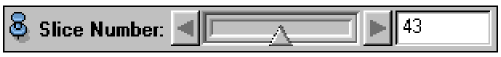
Contrast Limit

This port is only visible if *histogram* equalization is selected. The number determines the contrast of the resulting image. The higher the value, the more contrast is contained in the resulting image.

Colormap

Choose colormap if *mapping type* is set to *colormap*.

Slice Number



This slider allows you to select different slices. The slices may also be picked and dragged directly in the 3D viewer.

Transparency

This radio box port determines the transparency of the displayed slice. *None* means that the slices are fully opaque. *Binary* means that black parts are fully transparent while other parts are opaque. Finally, *linear* means that opacity is proportional to luminance.

6.58 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.

Connections

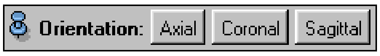
Data `[required]`
Vector field to be visualized.

ColorField `[optional]`
A scalar field which may be used for pseudo-coloring.

Colormap `[optional]`
Colormap used for pseudo-coloring. If no colormap is connected the default color of the colormap port will be used. The port is hidden if pseudo-color mode is set to *none*.

Ports

Orientation



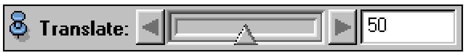
This port provides three buttons for resetting the slice orientation. *Axial* slices are perpendicular to the z-axis, *coronal* slices are perpendicular to the y-axis, and *sagittal* slices are perpendicular to the x-axis.

Options

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

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

Translate

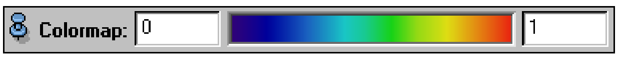


This port lets you translate the plane along its normal direction.

Colorize

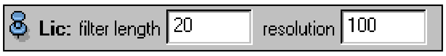
An option menu allowing to select different pseudo-color modes. If item *none* is selected the LIC texture will be displayed in greyscale only. If *magnitude* is selected each pixel of the LIC texture will be colored according to vector magnitude at this point. If *normal component* is selected then color denotes the signed length of the vector component perpendicular to the cutting plane. This length will be positive if the vector points upwards or negative if the vector points downwards. If *parallel component* is selected then color denotes the length of the vector component tangential to the plane. This length will always be greater or equal than zero. Finally, if *color field* is selected and if a scalar field is connected to port *ColorField* the external scalar field will be used for pseudo-coloring.

Colormap



Port to select a colormap.

Lic



The input denoted *filter length* controls the one-sided length of the filter kernel used for line integral convolution. The larger this value is the more coherent is the greyscale 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 or equal than the sampling resolution. For example, if the resolution is set to 128 the size of the LIC texure 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.

Phase

This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controling 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.

Action

Starts computation of the LIC texture. 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 *DoIt* button again in order to update the texture.

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 in increased computing time.

`writeTexture <filename>`
This command allows you to write the current LIC texture into a file in raw PPM format.

6.59 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* will be invoked implicitely by certain modules such as the *data probing modules* or the *LabelVoxel* module. Each of these modules generates 1-dimensional data such as a function plot along a line or a histogram. While the data-generating modules control the initial settings of the plot window the user can freely adjust these settings afterwards.

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

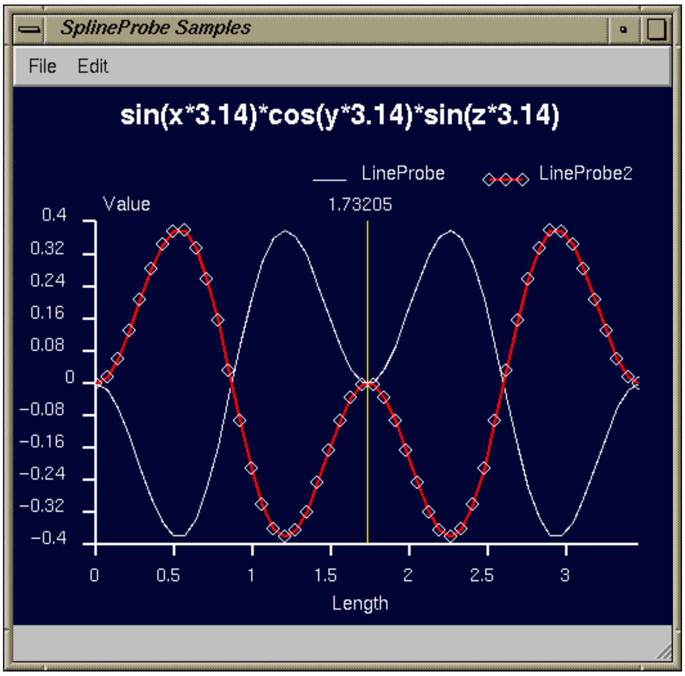


Figure 6.1: Amira's plot window.

- Curves
- Cartesian Axes
- Polar Axes
- Plot Areas
- Legends
- Annotations
- Markerlines

The plot engine processes the objects top to bottom (see Figure). The sequence of objects determine 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. An axis is placed in such an area and establishes together with the preceding PlotArea a window-viewport (world coordinates to normalized coordinates) transformation. Notice that the objects have to be kept in the following sequence: 1. *PlotArea*, 2. *Axis*, 3. *Curves* and *Markerlines*.

Every object is identified by a unique name. You need to know these names if you want to change certain object attributes via the command interface of the *Plot Tool*.

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. By selecting 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.

In the following sections all not self-describing parameters are documented.

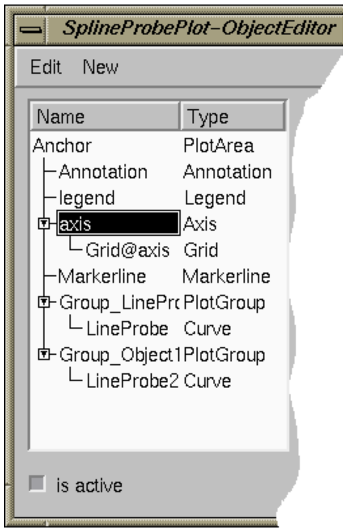


Figure 6.2: List to select a plot object.

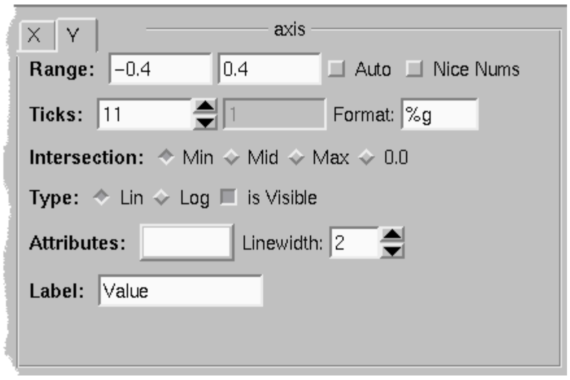


Figure 6.3: Editable axis parameters.

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 have to set off the *Auto* toggle. Now the range fields are 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* are set to -1.

It is possible to zoom interactively into your data using the mouse. For this purpose 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.

Editing annotation parameters

To position an annotion on your plot you can switch between world coordinates or normalized coordinates. The usage of world coordinates is useful if you like to annotate a certain feature e.g. an extrema of a curve. In this case you have to insert the annotion after that curve object. You can also position an annotation in the plot window interactivly by pressing the left mouse button on the annotation, move the mouse and release

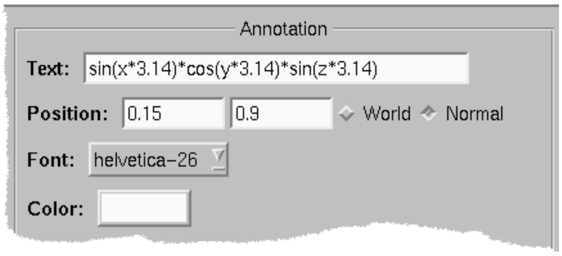


Figure 6.4: Editable annotation parameters.

it at the new position.

Editing legend parameters

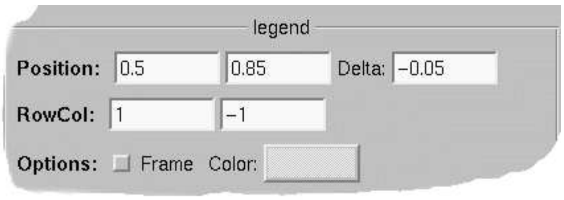


Figure 6.5: Editable legend parameters.

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. The RowCol parameter determines how many rows and columns are available for the legend items. The default is -1 and means an 'unlimited' number of items where the number of columns (2. parameter) counts first. E.g if you have 6 curves in your plot and you want the legend to have 2 columns and 3 rows set RowCol to 3 and 2. Mind the x position to make sure that all names are fully displayed. You can also position a legend in the plot window interactivly by pressing the left mouse button on one of the names in the legend, move the mouse and release it at the desired position.

Editing markerline parameters

The position of a markerline has to be given in world coordinates. Also be sure that the markerline is inserted after the appropriate axis. You can shift a markerline horizontally resp. vertically in the plot window by pressing the left mouse button on the markerline, move the mouse and release it at the new position.

Working with plot objects

Furthermore you can copy or delete plot objects. E.g. 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 have to select the object in the

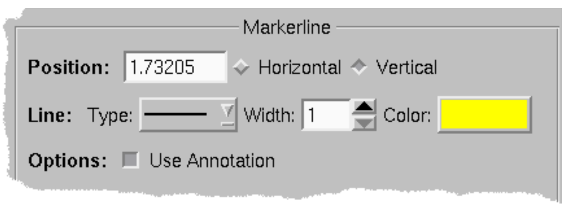


Figure 6.6: Editable markerline parameters.

list and then choose the *Edit->Copy* pulldown menu. After that choose *Paste* or *Append* in the menu and the object will be inserted at the current position (= position of the selected object) resp. the object will be placed behind the selected object.

With the *New* pulldown menu you can create new objects which will be inserted at the appropriate positions in the list of objects.

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.

Saving data

The *Save data...* item under the *File* menu lets you save the data of all curves in a file. The date is stored in a proprietary format subject to change in the future.

Commands

In the Amira environment 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.

`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`), JPEG (`.jpg,.jpeg`), and PNM (`.pgm,.ppm`).

`print <filename>`
Prints the plot window into a PostScript file (vector based).

`save <filename>`
Saves the data of all data based plot objects in a propriatary format.

`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.

6.60 Point Probe

See Section *Data Probing* for details.

6.61 PointWrap

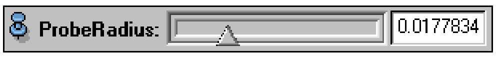
This algorithm performs a surface reconstruction from a set of unorganized points. It models a *probe sphere*, that is being 'dropped' onto and then 'rolled over' the set of points. Every three points the sphere rests on during this tour become a triangle in the resulting surface. The result is (almost) guaranteed to be an oriented manifold.

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.

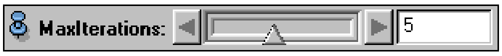
Probe Mode

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.

Action

The *DoIt* button triggers the computation.

6.62 Refined E-Field Simulation

The *Refined E-Field Simulation* module performs an E-field calculation using the *Finite Element* method with *adaptive grid refinement*. For this purpose it also performs a *temperature calculation* and a *temperature optimization* on each refinement level. See the descriptions of modules *E-Field Simulation*, *Static Heat*, and *OptTemp* for more details.

Before starting the E-field calculation, you must create an *extended tetrahedral grid* by combining a patient grid and a grid representing the exterior space. This can be done using module *Combine*. The grid representation of the exterior space itself is generated using modules *Antenna* and *BolusGrid*.

When you have created an extended grid, load it into HyperPlan. Then you can select the *Refined E-Field Simulation* module from the grid's popup menu. E-field calculation will take ca. 10-20 minutes CPU time per channel of the applicator on the starting grid. For each refinement level calculation time will increase approximately by a factor 2. The calculation is performed as a *batch job*. The status of the job queue can be controled in the *Job Dialog*.

Connections

Data

An extended tetrahedral grid. Usually such a grid has the suffix `ExtendedGrid`.

Ports

Directory, EFieldGrid, EFieldVals, Frequency, Precision, Options

Same functionality as in module *E-Field Simulation*.

Refinement Steps

Here you can set the number of adaptive grid refinements. It is recommended to perform at most 2 refinement steps.

Action

Button *El.Properties* invokes an editor window which allows you to specify the electric material properties (Same functionality as button *Materials* in module *E-Field Simulation*). Note: *the electric parameters stored in HyperPlan's material database refer to a frequency of 90 MHz.*

Buttons *Parameters* and *Therm.Properties* invoke editor windows which allow you to specify the thermal material properties (Same functionality as buttons *Parameters* and *Materials* in module *Static Heat*).

After you have setup the simulation, you can commit it by pressing the *Submit* button. If one of the specified output files did already exist, a warning message is issued. If you don't want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, 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.

6.63 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. The other ports depend on whether the input field contains ordinary numbers or labels as used in image segmentation (c.f. LabelField).

For non-labeled data fields you see the *Filter* and the *Resolution* port. In case of a labeled data field you see the *Average* port. In both cases the *DoIt* button completes the port list. Depending on the existence of labeled data the resampling operation is performed differently.

Resampling non-labeled data fields

For non-labeled input data two ports denoted *Filter* and *Resolution* will be shown.

Filter provides an option menu allowing you to specify the filter kernel for the resampling operation. Usually the default kernel, *Triangle*, yields sufficiently good results for both minifications and magnifications. The following filters are supported:

- *Box* - simple replication of scalar values, shows considerably tiling or jaggies
- *Triangle* - computationally simple, still sharp transition lines
- *Bell* - smoothing filter
- *B-Spline* - no sharp transitions, but its width causes excessive blurring
- *Lanczos* - excessive "ringing" effect
- *Mitchell* - no sharp transitions, good compromise between "ringing" and "blurring"
- *Minimum* - useful for down-sampling, preserves tiny dark features on a bright background
- *Maximum* - useful for down-sampling, preserves tiny bright features on a dark background

The *Minimum* and *Maximum* filters can only be used to downsample non-complex data fields.

Port *Resolution* lets you specify the desired resolution of the output field. To start resampling, press the *DoIt* button.

Resampling labeled data fields

Labeled data fields can only be down-sampled. Instead of *Filter* and *Resolution* a port denoted *Average* appears. This port allows you to enter the number of cells to average in every dimension. Note, that no enlargement is possible.

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

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.

Resampling in progress

While resampling, the *progress bar* at the bottom of the *work area* indicates the percentage of resampling that is done. You may cancel the resampling calculation any time by pressing the stop button on the right of the progress bar.

Connections

Data `[required]`
The underlying data field to be resampled.

Ports

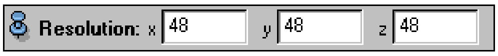
Input Resolution

Displays the resolution of the input data set.

Filter

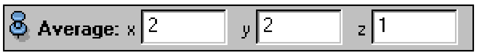
This lets you select one of the resampling filters mentioned above. This will only be visible if the input data set does not contain labels.

Resolution



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

Average



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

Action

Starts resampling.

6.64 ScriptObject

The Amira system is fully scriptable by its built-in TCL interface. The *ScriptObject* module allows the user or custom solution provider to make its scripts accessible from Amira's graphical user interface. Here is the most simple example of how this can be done:

Write a simple Amira script (using your favorite text editor) and put the special header for script objects in its first line.

```
# Amira-Script-Object V0.1

echo "Hello world, a script is called."
```

Load this file into Amira. A blue icon will appear. Each time you click on the *Call* button, the script will be executed and a message in Amiras console window appears.

It is possible to build more complex user interfaces with script objects, as shown in the second example:

```
# Amira-Script-Object V0.1

proc constructor { } {
    global this
    $this newPortIntSlider myValue
    $this myValue setLabel "Value:"
}

if { [ $this isFirstCall ] } {
    constructor
}

set val [$this myValue getValue]
echo The value is $val
```

Here, we use a number of special commands. The global variable always contains the name of the script object, which is currently executed. This allows to access its special commands. The command

```
$this isFirstCall
```

returns true only the first time the script is executed (typically when the script is loaded or when reset is hit). In the example, a TCL procedure called constructor is defined and called the first time the script is executed. Within this procedure, the script object is told to create a new port of type HxPortIntSlider, which will appear in the user interface. The port has the internal name myValue and its visible label is set to Value. Finally, the current value of the port is accessed using the command

```
this myValue getValue.
```

Whenever the slider is moved, the script will be called and the current value is printed.

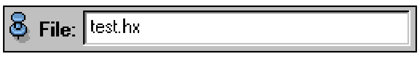
Each time the script is called, it is read from the disk, this means that you do not need to reload the data, if you have modified the script.

Connections

Data `[optional]`
Can be connected to any data object. Can be used by the script.

Ports

File



File name of the TCL script.

Action

Call simply executes the script (as a change of any other, dynamically created, port would do). *Restart* deletes all dynamically created ports, sets the `isFirstCall` flag to `1` and calls the script.

6.65 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 object 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*.

Connections

Data `[required]`

Module of type *DisplayISL* which is used to display the illuminated field lines.

DisplayISL `[required]`

The surface on which the field lines will be distributed.

Ports

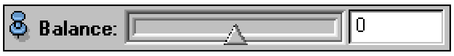
NumLines

Number of field lines to be displayed. This port will only be visible if distribution mode *on surface* has been selected. The distribution algorithm tries to achieve a constant seed density per surface area.

Length

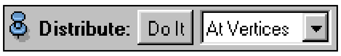
The length of the field lines, or more precisely, the number of atomic line segments, in forward respectively backward direction. The lines may stop earlier if a singularity (i.e. zero magnitude) is encountered or if the field's domain is left.

Balance



On 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



On the one hand, this port provides a *DoIt* button which 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 have to press *DoIt* in order to update the display.

On the other hand, the port also 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.

6.66 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.

Connections

Data `[required]`

Connects to uniform scalar fields.

Ports

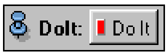
Info

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.

DoIt



Triggers the computation.

6.67 ShowContours

This module computes and shows contours for a 3D label field on a 2D cutting plane. The plane itself is defined by another module which must be connected to port *Module*. Since the label field has integer coordinates and from the intersection with an arbitrary cutting plane result real coordinates, the algorithm must make approximations. This approach may lead to artifacts if the sampling density is too low. The module computes contours between all the materials.

Connections

Data `[required]`

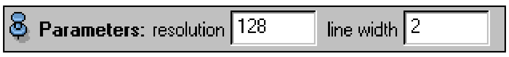
3D label field for which contours are to be computed.

Module `[optional]`

Module defining cutting plane used for contours computation.

Ports

Parameters



The *resolution* value determines the resolution of the sampling raster used for computing contours. The second input of this port determines the width of the contours in pixels.

6.68 SmoothSurface

This module performs a surface *smoothing* by shifting its vertices. Each vertex is shifted according to its neighbors' coordinates. 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 the boundaries are preserved.

The smooth 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.

Connections

Data `[required]`

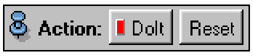
The surface to be smoothed.

Ports

Parameters

Contains two parameters: *Iterations* specifies the number of steps for the smoothing, *lambda* is a shifting coefficient which should be in the range (0..1)

Action



The *DoIt* button starts the smoothing, pressing the *Reset* button the original surface will be restored.

6.69 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 artifical discontinuities might be observed.

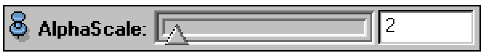
Connections

Data `[required]`

The tetrahedral scalar field to be visualized.

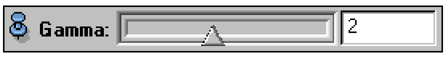
Ports

Alpha Scale



A global factor used to change the overall transparency of the individual splats. Higher values produce denser clouds.

Gamma

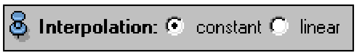


This value determines how the function value between the min and max values of port *Range* will be mapped to opacity and color. If the gamma value is 1 a linear mapping will be used. If the gamma value is smaller than 1 the overall appearance of the cloud gets more opaque. If the gamma value is bigger than 1 the cloud gets more transparent.

Range

Data range. Regions where the function value is below the min value will be completely transparent. Likewise, regions where the function value is above the max value will be opaque.

Interpolation



Lets you select the type of splats being used. *Constant* enables untextured splats. *Linear* enables textured splats. The latter setting will be able to correctly display linearly varying scalar fields.

Commands

```
setColor <color>
```

Lets you define the base color of the volume rendered clouds. On default an orange color will be used (0.8 0.6 0.1). The color may be specified by either an RGB triple in range 0...1 or by a common X11 color name, e.g. *green*.

6.70 Spline Probe

See Section *Data Probing* for details.

6.71 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 contain coronal, sagittal, and axial projections of the data, respectively. These projections correspond to standard xy-, yz, and xy-orientations. Note, that for the axial (xy) view the origin is in the upper left corner. This is the standard orientation used in radiology. In each 2D view the position of the two other slices is indicated by a blue cross hair. You may click at any point in a 2D view in order to reposition the two other slices.

Connections

Data `[required]`

The scalar field to be visualized.

Plane1 `[optional]`

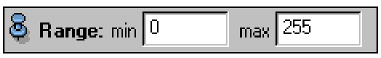
If this port is connected to an *OrthoSlice* module, which is connected to the same input, an additional brownish line is displayed in the 2D viewers, showing the position of the *OrthoSlice* module.

Plane2 `[optional]`

See above.

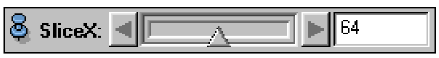
Ports

Range



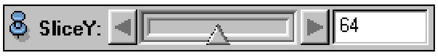
Control the mapping of input data to grey values. Values below *min* are mapped to black, values above*max* are mapped to white. Values between *min* and *max* are mapped linearly.

SliceX



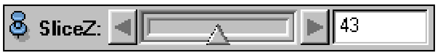
Number of slice currently displayed in sagittal (yz) window.

SliceY



Number of slice currently displayed in coronal (xz) window.

SliceZ



Number of slice currently displayed in axial (xy) window.

ZoomButtons

Allows you to decrease, reset, or increase the current image magnification factor. You may also `[Ctrl][z]` to zoom down or `[Ctrl][Shift][z]` to zoom up.

6.72 Static Heat

The *Static Heat* module performs a *temperature simulation* on a tetrahedral patient model. Like for the electromagnetic fields a set of independent temperature distributions are computed. For simulating an applicator with N independent channels the number of temperature distributions is N*N + 1. From these distributions the actual temperature distribution for any set of antenna amplitudes and phases can be obtained very quickly by superposition.

Note: Temperature simulation can only be done if the preceding E-field calculation was performed on a tetrahedral grid.

First you have to load a tetrahedral patient model as well as a set of E-field channels into HyperPlan. If the FDTD method was used for E-field calculation, the patient models are called `EFieldGridFD`, while the files containing the E-field channels have a suffix `EFieldValsFD`. If the FE method was used, the suffixes are `EFieldGridFE` and `EFieldValsFE` respectively.

Connections

Data
A set of E-field distributions as created by modules *FDTD* (FDTD method) or *E-Field Simulation* (FE method). The data files usually have a suffix `EFieldValsFD` or `EFieldValsFE`.

Ports

Directory, ThermGrid, ThermVals
The *Static Heat* module creates two output files, a temperature grid (`ThermGrid`) and a file containing a set of temperature distributions (`ThermVals`). The temperature grid is very similar to the initial electromagnetic coarse grid. However, it does not contain any duplicated vertices, nor does it contain information about the grid edges. By this ports you can specify the names of the output files. HyperPlan makes a suggestion for you, but you can change it as you please. It is recommended to use the suffixes `ThermGridFD` and `ThermValsFD`, or `ThermGridFE` and `ThermValsFE`, depending on the suffix of the `EFieldVals` input data.

Action
If you press the *Parameters* button an editor window appears where some global parameters for temperature simulation are presented. Changes can be made by simply selecting an item and typing a new value. The ok button in the editor window saves the changes. If you press the *Materials* button you can change the thermal properties of the different tissue types. After you have setup the simulation, you can commit it by pressing the *Submit* button. The *job dialog* window should appear, showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

6.73 Static Heat Nonlinear

The *Static Heat Nonlinear* module performs a *temperature simulation* on a tetrahedral patient model. The underlying heat transfer model is a *nonlinear* version of the *Bio-Heat Transfer Equation* taking into account that blood perfusion depends on temperature. The superposition principle (see module *Static Heat*) is no longer valid for the nonlinear case. Therefore module *Static Heat Nonlinear* calculates only one temperature distribution corresponding to a specific (optimal) set of antenna parameters which are also determined by this module.

Note: Nonlinear temperature simulation can only be done if the preceding E-field calculation was performed using the FE method (module E-Field Simulation).

On default, a nonlinear behaviour is assumed for the following tissue types: *Fat*, *Muscle*, and *Target*. For the temperature dependence of blood perfusion a *sigmoidal* curve is assumed, which is characterized by 4 parameters:

TLowForPerfusion	lower bound for temperature,
THighForPerfusion	upper bound for temperature,
PerfusionAtLow	perfusion at lower bound (and below),
PerfusionAtHigh	perfusion at upper bound (and above).

First you have to load a tetrahedral patient model as well as a set of E-field channels into HyperPlan. The patient models normally are called `EFieldGridFE`, while the files containing the E-field channels have a suffix `EFieldValsFE`.

Connections

Data

A set of E-field distributions as created by module *E-Field Simulation*. The data files usually have a suffix `EFieldValsFE`.

Ports

Directory, ThermGrid, ThermVals

The *Static Heat Nonlinear* module creates two output files, a temperature grid (`ThermGrid`) and a file containing a temperature distribution (`ThermVals`). The temperature grid is very similar to the initial electromagnetic coarse grid. However, it does not contain any duplicated vertices, nor does it contain information about the grid edges. By this ports you can specify the names of the output files. HyperPlan makes a suggestion for you, but you can change it as you please. It is recommended to use the suffixes `ThermGridNL` and `ThermValsNL`.

Action

If you press the *Parameters* button an editor window appears where some global parameters for temperature simulation are presented. Changes can be made by simply selecting an item and typing a new value. The ok button in the editor window saves the changes.

If you press the *Materials* button you can change the thermal properties of the different tissue types. If you want a tissue type different from *Fat*, *Muscle* or *Target* to show nonlinear behaviour, add the four parameters described above (*TLowForPerfusion*, *THighForPerfusion*, *PerfusionAtLow*, and *PerfusionAtHigh*) to the parameter list of that tissue type.

After you have setup the simulation, you can commit it by pressing the *Submit* button. The *job dialog* window should appear, showing you the status of the job queue. If you press the *Start* button, the first pending job of the queue starts running.

6.74 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. On default, the current stream surface will be display in wireframe mode and will be colored according to the arc-length of the stream lines forming the stream surface.

Ports

Colormap

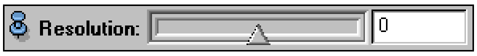
Port to select a colormap.

Origin

This port defines the seed point of the stream surface. By pressing the toggle *show* a cross hair dragger can be activated which allows you to change the seed point interactively in 3D.

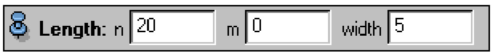
Actually, in order to compute a stream surface not only a seed point but a seed line is required. Starting from the seed point, such a line will be defined automatically taking into account the seed selection mode chosen in port *Action*.

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 conicide 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*.

6.75 Superpose (Electric Fields)

In simulating regional hyperthermia, a separate field is calculated in advance for each channel of the hyperthermia applicator (see modules *FDTD* and *E-Field Calculation*). Then the electric field for any set of antenna amplitudes and phases can be obtained very quickly by superposition.

Connections

E-Field Set
A set of electric fields corresponding to the channels of a hyperthermia applicator, as created by module *FDTD* or *E-Field Calculation*. The data files usually have a suffix `EFieldVals`.

Plan Data (optional)
A file containing antenna parameters (amplitudes and phases). Such files usually have a suffix `Plan`. The antenna parameters are used as default values for superposition. For more information see the *Plan Data* description.

Ports

Action
If you press the button *Save*, the actual antenna parameters (power amplitudes and phase delays) are stored into a file. If the button *Reset* is pressed, the last interactive changes of the antenna parameters are removed and the actual default values (synchronous mode or antenna parameters read from a file) are restored. The option menu on the right allows to select between different antenna steerings. *Default* means a synchronous steering of all antennas. If files containing antenna parameters (suffix `Plan`) have been loaded, their file names occur in the menu. In this way the E-field or SAR distribution corresponding to an optimized temperature distribution (see module *Opt Temp*) can be shown by loading the file containing the optimal antenna parameters and selecting its filename from the option menu.

Power and Scale Factor

The power amplitudes for the single channels (see below) should only be used to do a relative weighting between the channels. Using port *Factor* you can define an additional over-all scaling factor for all channels. Using port *Power* the scale factor can be defined indirectly by prescribing the total power absorbed in the patient.

Note: The value displayed at port Power means the total power absorbed in the patient model, not the total power emitted by the hyperthermia applicator.

Auto Scale

This toggle determines what will happen if the parameters for the single channels are changed. If the toggle is off, the scale factor is kept constant, and the total power may change. If a file containing antenna parameters is read and this file also contains a scale factor, this scale factor is applied. If the toggle is set, the total power is kept constant, and the scale factor may change. The scale factors contained in antenna parameter files are ignored.

If module *Superpose* is applied to results from a *finite element* E-field calculation, there is a second toggle *update power*. If this toggle is set, the power value is recalculated each time the antenna parameters are changed. If port *Output* is set to *E-Field*, antenna parameters can be changed much faster if the toggle is switched off.

Output

Select between output of the superposed electric field and output of the corresponding SAR distribution.

Other Ports

Antenna parameters for all channels of the hyperthermia applicator: *Power Amplitudes* (relative units) and *Phase Delays* (degree). If the number of channels is 4, it is assumed that the applicator Sigma60 is simulated and the channels are labeled *Left*, *Bottom*, *Right*, and *Top*.

Note: The order of the 4 channels differs from the order on the BSD 2000 console!

6.76 Superpose (Temperature Distributions)

The module Superpose also computes the temperature distribution corresponding to a given set of antenna parameters (power amplitudes and phase delays), if it is selected from the popup menu of a file containing a set of temperature distributions. Such files are created by module *Static Heat*. They must contain N*N + 1 temperature distributions for simulating an applicator with N independent channels.

Connections

Temperature Data Set

A set of temperature distributions as created by module *Static Heat*. The data files usually have a suffix `ThermVals`.

Plan Data (optional)

Same functionality as in module *Superpose* as applied to electric fields. For more information see the *Plan Data* description.

Ports

The number of parameter ports and their functionality differ slightly from module *Superpose* as applied to electric fields.

Action

Same functionality as in module *Superpose* as applied to electric fields.

Scale Factor

The power amplitudes for the single channels (see below) should only be used to do a relative weighting between the channels. Using this port you can define an additional over-all scaling factor to be applied to all channels.

Note: The scale factor cannot be changed if toggle Auto Scale *is set.*

Auto Scale

If this toggle is off the port *Scale Factor* is active. If a file containing antenna parameters is read and this file also contains a scale factor, this scale factor is applied. If the toggle is set the port *Scale Factor* is inactivated and the scale factors contained in antenna parameter files are ignored. The scale factor is calculated according to maximal temperatures defined for each type of healthy tissue. The maximal temperatures are defined in *HyperPlan's material database*. They are currently set to 44°C for most tissues and 42°C for some heat sensitive tissues like bladder and intestine. By default, the toggle is set.

Note: This toggle has a different meaning than AutoScale in module Superpose as applied to electric fields.

Other Ports

Same functionality as in module *Superpose* as applied to electric fields.

6.77 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.

Click here to execute a script demostrating the *SurfaceLIC* module. Computation of the surface LIC texture may take about half a minute.

Connections

Data `[required]`

Surface for which a LIC texture is to be computed.

VectorField `[required]`

Surface vector field or 3D vector field to be visualized.

ScalarField `[optional]`

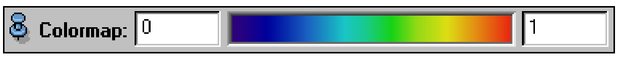
An optional scalar field which can be used for pseudo-coloring.

Colormap `[optional]`

Colormap used for pseudo-coloring. If no colormap is connected the default color of the colormap port will be used.

Ports

Colormap



Port to select a colormap.

ColorMode

Three different color modes are provided. If *Constant* is selected then a uniform overall base color is used for the LIC texture. This will be the default color of the colormap port or the left-most color of the colormap connected to this port, if any. If *Magitude* is selected then the LIC texture will be colored according to the magnitude of the vector field. Finally, if *Color field* is selected and a scalar field is connected to the module then the LIC texture will be colored according to the values of this scalar field.

Texture Interpolation

A radio box determining how the triangle textures are being filtered by the underlying OpenGL driver. Possible choices are *constant* or *bilinear* interpolation. Usually, you will not see a big difference unless you zoom up the image very much.

Contrast

This port provides two parameters controlling the amount of contrast of the final LIC texture. Input field *center* specifies the average grey value of the texture. Higher values result in brighter images. Input field *factor* determines the width of the grey 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 greyscale 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.

Action

Starts 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.

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.

6.78 SurfaceView

This module allows you to visualize triangular surfaces, i.e. data objects of type *Surface*. Derived from the generic *ViewBase* class, the module provides an internal buffer of visible triangles. You can add triangles to this buffer by means of two special option menus. For example, if your surface contains regions FAT and MUSCLE, you may first highlight all triangles separating these two regions by choosing FAT and MUSCLE in port *Materials*. Highlighted triangles are displayed in red wireframe. By pressing button *Add to* of port *Buffer* highlighted triangles can be added to the internal buffer, which causes them to be displayed in their own colors. You may restrict highlighting by means of an adjustable box. In order to resize the box pick one of the green handles at the corners of the box. Highlighted triangles may also be removed from the buffer by pressing button *Remove*. In addition, individual triangles may be removed from the buffer by *shift-clicking* them.

Connections

Data `[required]`

The surface to be visualized.

ColorField `[optional]`

In conjunction with a colormap an optional field can be visualized on top of the surface in pseudo-color mode. The field may be either of type *HxScalarField3* or of type *HxSurfaceScalarField*. In addtion to scalar surface fields also 3 and 4-component surface fields are supported. In this case, the field components are directly interpreted as RGB or RGBA values. The values should range from 0 to 1.

Colormap `[optional]`

The *Colormap* is used to visualize the data values of a scalar field connected to port *ColorField*.

Ports

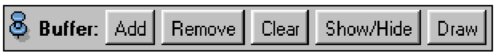
Draw Style

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

Colormap

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

Buffer



This port lets you *add* and *remove* highlighted triangles (being displayed in red wireframe) to an internal buffer. For a further description and for the functionality of each of the port buttons see *ViewBase*.

Materials

This port provides two option menus listing all regions defined in the surface. By setting the menus properly, you can highlight all triangles separating two different regions. Highlighted triangles are displayed in red wireframe. You may restrict the set of highlighted triangles by means of an adjustable box. Use port *Buffer* to add or remove highlighted triangles to the internal buffer.

Colormode

There are three different coloring modes:

Normal: Each side of a triangle is colored according to the color of the region it points to.

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: In the twisted mode the colors of the two triangle sides in *normal* mode are inverted.

6.79 Tetra Combine

This module takes two tetrahedral grids as input, puts them together and creates a new combined grid. The module can be used to create an *extended grid* from a patient grid and a grid representing the exterior space including parts of a treatment device or support to which the patient's body is attached. Triangles representing the boundary of the patient's body may be present in both input grids. *Combine* provides an option to remove any duplicated triangles and vertices from its output. The order in which the input grids are combined does not matter.

To make sure that both grids being combined fit exactly together, you can attach a *GridVolume* module to the combined grid. When invoking that module, all *exterior* triangles of the grid are highlighted, i.e., all triangles which are incident to only one tetrahedron. The displayed triangles should all be located at the outer boundary of the combined grid, not in its interior.

Connections

Data `[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.

Action

The *DoIt* button triggers computation.

6.80 TetraGen

The *TetraGen* module creates a *volumetric tetrahedral grid*. Its input is a description of the 3D geometry by triangulated surfaces. The *advancing front method* is applied to fill each region defined by the surface data with tetrahedra. Tetrahedron generation is performed 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 *GMC* module creates correct names ('Exterior' and 'Exterior2'), but the module *IvToSurface* does not. You must edit the region names before applying TetraGen to a surface created by the latter module.

Connections

Data `[required]`

The input surface file (suffix `surf`).

ExteriorSurface
`[optional]`
Additional outer bounding surface. Not needed in most applications.

Ports

Region

The menu of this port allows you to specify whether tetrahedra should be generated for all regions or just for one selected region. The latter choice is needed for testing purposes only.

Grid

Here you can define the filename for the resulting tetrahedral grid. We recommend to include a suffix `grid` into the filename.

Action

If you press the *Materials* button an editor window appears. It allows you to define a desired mesh size (in cm) for each region. Normally, the predefined values should work well. After you have set up the simulation, you can commit it by pressing the *Run* button. If the file specified at port *Grid* did already exist, a warning message is issued. If you do not want to overwrite the file, press *Cancel* and change the filename. When the job has been submitted, 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.

6.81 TetraQuality

The *TetraQuality* module creates a histogram of qualities for tetrahedral volume grids, e.g. tetrahedral patient models. For this purpose it has to be attached to a *Grid Volume* module. Quality is calculated for all tetrahedra selected by that module. On default, the histogram is shown with a logarithmic scale to direct the focus on the tetrahedra with worst quality.

Connections

Data `[required]`

The tetrahedral volume.

GridVolume `[required]`
A *Grid Volume* module that selects the tetrahedra for which the quality is calculated.

Ports

Quality Measure

This option menu lets you select between different quality measures:

- **Diameter Ratio:** Ratio of diameters of circumscribed and inscribed sphere. The optimal (minimal) value is 3.
- **Aspect Ratio:** Aspect ratio = 3 / diameter ratio. The optimal (maximal) value is 1.
- **Dihedral Angle:** For each tetrahedron edge the dihedral angle is defined as the angle between its adjacent faces. For an equilateral tetrahedron all dihedral angles are about 70 degrees.
- **Solid Angle:** For each tetrahedron vertex the solid angle is defined as the part of the unit sphere which is occupied by the tetrahedron. For an equilateral tetrahedron all solid angles are about 30 degrees.
- **Edge Length:** If you choose this measure, a histogram of edge lengths is created for all selected tetrahedra.

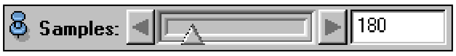
Select Angle

If the quality measure is dihedral or solid angle, you can select whether a histogram is created for

- all angles,
- the minimal angle,
- the maximal angle, or
- minimal and maximal angle

of each tetrahedron.

Samples



This slider lets you select the number of samples for the histogram. Normally, the default values should be adequate.

Histogram

If you select this toggle, a plot window appears showing the histogram of qualities for all tetrahedra selected by the *Grid Volume* module. If no tetrahedra have been selected, the plot window will not be shown.

6.82 TissueStatistics

This module takes a uniform *label field* as well as an optional scalar field as input and computes some statistical quantities for the regions defined in the label field. The results are stored in a *spreadsheet* object. The different columns of the spreadsheet have the following meaning:

- **Number:** Enumerates all materials of the label field.
- **Material:** Denotes which material is stored in a row.

- **Count:** Number of voxels contained in a region.
- **Volume:** Number of voxels times size of a single voxel.
- **CenterX:** X-coordinate of the region's center.
- **CenterY:** Y-coordinate of the region's center.
- **CenterZ:** Z-coordinate of the region's center.

If an additional scalar field is connected to the module, then four more columnns will be generated:

- **Mean:** Mean value of the field in a particular region.
- **Deviation:** Standard deviation of the field in a region.
- **Min:** Min value of the field in a particular region.
- **Max:** Max value of the field in a particular region.

The scalar field will be evaluated at the center of each voxel.

Connections

Data `[required]`

Label field defining the regions.

Field `[optional]`

Optional scalar field.

Ports

Action

Triggers the computation.

6.83 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.

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:

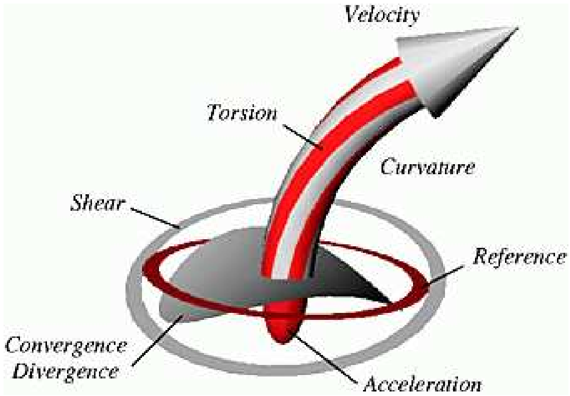


Figure 6.7: Components of the vector field probe.

- **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.

Action

Press the *DoIt* button to start calculation of triangle qualities.

6.84 VectorProbe

The *VectorProbe* module allows you to interactively investigate a 3D vector field by moving around a dynamic vector field probe. The probe displays certain quantities associated to the first order derivative of the field in an intuitive way. This kind of probe has been originally proposed by W.C. de Leeuw and J.J. van Wijk in *A Probe for Local Flow Field Visualization, Proceedings of Visualization'93, pp. 39-45*. It looks as follows:

Connections

Data `[required]`

The 3D vector field to be visualized.

Ports

Dragger

Shows or hides the dragger and the vector field probe attached to it. The dragger provides a cylinder handle and a square plate handle. The cylinder handle allows you to translate the icon along the center axis. The orientation of the cylinder can be changed using the `[Ctrl]` key while the mouse is located somewhere over the dragger.

Buffer

Adds the current vector field probe to an internal buffer. In this way multiple probes can be displayed at once.

Scale

Scales the overall size of the vector field probe.

Length

Adjusts the length of the probe's arrow part.

6.85 Vectors

This is a so-called *overlay module* which can be attached to any module defining a cutting plane, e.g. *OrthoSlice* or *ArbitraryCut*. Inside this plane a 3D vector field can be visualized using a regular array of vector arrows.

Connections

Data `[required]`
The 3D vector field to be visualized.

Module `[required]`
The module which defines the cutting plane where the arrows are placed.

Colormap `[optional]`
An optional colormap used for pseudo-coloring.

Ports

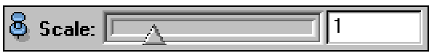
Colormap

Port to select a colormap.

Resolution

Provides two text inputs defining the resolution of the regular array of vector arrows in the plane's local x- and y-direction. The larger these values are the more arrows are displayed.

Scale



Scaling factor used to control the length of the vector arrows.

Options

This port provides the following toggle buttons.

Projection: If this option is set then 3D vectors are projected into the current plane. Otherwise, the arrows will indicate the true direction of the vector field.

Constant: If this option is set then all arrows will be of equal length. Otherwise, the length of the arrows is chosen to be proportional to vector magnitude.

Arrows: If this option is set then true arrows will be displayed. Otherwise, only simple line segments will be drawn.

Points: If this option is set then a little dot will be drawn at the bottom of an arrow. This is useful to highlight a sampling point at locations where the vector magnitude is so low that the arrow vanishes completely.

Colorize

An option menu allowing to control how arrows are colored. In order to see an effect a colormap must be connected to the module. If *Magnitude* is chosen then the vector magnitude is used to lookup a color from the colormap. If *Normal component* is chosen then color denotes the signed length of the vector component perpendicular to the cutting plane. This length is positive if the vector points upwards or negative if the vector points downwards. Finally, if *Parallel component* is chosen then *color* denotes the length of the vector component tangential to the plane. This length will always be greater or equal than zero.

Phase

This port will only be visible if a complex-valued vector field is connected to the module. It provides a phase slider controlling which part of the complex 3D vectors is visualized by the arrows. A value of 0 degree corresponds to the real part, while a value of 90 degrees corresponds to the imaginary part. The display can be animated with respect to the phase by the *cycle* button, this way polarization properties of the field can be revealed or wave phenomena become visible.

Commands

```
setLineWidth <value>
```

Allows to change the line width of the arrows. By default arrows are drawn two pixels wide.

6.86 Vertex View

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 color map. 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 color map to color the vertices according to the value of the scalar field at the position of a vertex.

Colormap `[optional]`

Used to color the vertices in connection with the *ColorField*.

Ports

Colormap

Port to select a colormap.

Draw Style

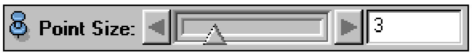
Vertices may be drawn in three different styles:

- *Spheres:* Points are drawn as triangulated spheres with equal radius.
- *Plates:* Points are drawn as quadrats with mapped-on image of a sphere.
- *Points:* Points are drawn as points. The size of the points does not differ according to the distance of the vertex from the viewer; they all have the same size.

SphereRadius

Specifies unique radius for all spheres. This port is only visible if *Spheres* or *Plates* is chosen as draw style.

Point Size



Size of points for draw style *Points*. Only visible in *Points* mode.

Complexity

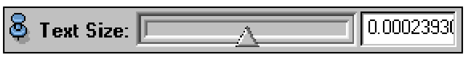
Set complexity of displayed spheres. Reducing the complexity leads to coarser spheres and improved rendering performance. If draw style is set to *plates* the complexity controls the size of the texture maps containing the sphere's images. The smallest texture size is 32x32, the biggest is 512x512. The port is not visible for draw style *points*.

Options:

A toggle list of options.

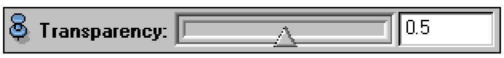
- *Show text:* If selected vertex numbers are drawn. This might be quite slow, especially for large vertex sets. Alternatively, you may select a vertex by clicking on it. Then its number is printed in the console window.
- *Transparency:* Optionally, the spheres and plates may be drawn transparent. Note that the vertices are not sorted along the z-axis. Thus the appearance of the vertex set may look flawed.
- *Buffer only:* The default is to display all spheres of the vertex set. By clicking this toggle you can display only those vertices that were previously added to the buffer.

Text Size



Specify textsize if *show text* is active.

Transparency



Allows you to specify the degree of transparency of the spheres.

Buffer:

A list of buttons that allow to manipulate the internal buffer. In order to actually see the buffer content activate the *buffer only* option.

- *Add:* adds selected vertices to buffer.
- *Remove:* removes selected vertices from buffer.
- *Clear:* removes all vertices from buffer.
- *Show/Hide Box:* controls box to select vertices.
- *Invert:* exchanges buffer content.

Commands

setHighlightColor <red> <green> <blue>
Set the color that is used to highlight selected spheres.

setDefaultColor <red> <green> <blue>
Set default color.

setColorHighlighted <red> <green> <blue>
Color all highlighted spheres.

setColor [<first-vertex-number> [<last-vertex-number>]] <red> <green> <blue>
Set color for all spheres from *first-vertex-number* to *last-vertex-number*. If *last-vertex-number* is omitted the color is set for the sphere with index *first-vertex-number*. If *first-vertex-number* is omitted too, the color for all spheres is set. *Red, green, and blue* range from 0 to 1.

highlight <first-vertex-number> [<last-vertex-number>]
Highlight spheres from *first-vertex-number* to *last-vertex-number*.

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.

6.87 ViewBase

This module is the base class for several other Amira modules displaying a set of triangles, like *Isosurface*, *SurfaceView*, *GridVolume*, and others. *ViewBase* is not useful on its own but provides special features common to all derived modules. In particular, these features comprise the following:

- A dedicated port allowing the user to modify the *draw style* of a triangular surface in an easy and consistent way. All modules derived from *ViewBase* thus have a similar GUI. Among the supported draw styles is a physically correct transparency mode.
- A generic *buffer* concept allowing the user to select triangles by means of an Open Inventor tab-box dragger. Only triangles added to the internal buffer will be displayed. Thus, complex surfaces may be decomposed into smaller pieces.
- An arbitrary 3D scalar field may be visualized on top of the triangular surface by means of pseudo-coloring. Pseudo-coloring may be achieved via Gouraud shading (mapped vertex colors will be interpolated) or, more accurately, via texture mapping (vertex values will be interpolated linearly and mapped to color afterwards).
- The set of triangles currently being visible can be converted automatically into a *Surface* object. This is useful for example in order to post-process isosurfaces or to extract parts from a bigger surface.
- Common *Tcl-commands* allow to control many parameters of modules derived from *ViewBase*. This includes line width, outline color, highlight color, specular color, shininess, alpha mode, normal binding, and more.

Connections

Color Field

Arbitrary scalar field to be visualized via pseudo-coloring. The color field will be evaluated at the surface's vertex positions and the vertex color will be set approriately.

Colormap

Colormap used for pseudo-coloring. To connect the port to a colormap use the popup menu under the right mouse button. To change the port's default color click it with the left mouse button. See also *Colormap*.

Ports

Draw Style

This port determines the draw style of the surface. Five major styles may be selected via an option menu:

Outlined: Opaque shaded display with edges superimposed.

Shaded: Opaque shaded display without edges being visible.

Lines: Shaded wireframe display.

Points: Triangle vertices only.

Transparent: Semi-transparent display.

Transparent mode implies physically correct transparencies, i.e., triangles appear more opaque if they are viewed under a small angle. It also implies approximate depth-sorting, i.e., triangles are roughly rendered from back to front in order to obtain correct blending results. Note, that in some cases visual artifacts may occur for long and thin triangles, for self-intersecting surfaces, or for multiple semi-transparent surfaces being displayed simultaneously.

The drawstyle may be fine-tuned by means of an additional popup menu which is activated by clicking on the button labeled *more options*. This menu contains the following items:

Shaded: Enables or disables specular highlights. Specular color and shininess may be changed via the Tcl-command interface.

Gouraud: Indicates that if a color field is connected pseudo-coloring is performed via Gouraud shading. First, colors are looked up at the traingles' vertices, then these colors are interpolated inside the triangles.

Texture: Indicates that if a color field is connected pseudo-coloring is performed via texture mapping. The color field's values are interpolated linearly before color lookup is performed. In this mode no specular colors can be used.

Opaque: All triangles will be rendered opaque.

Const alpha: Opacity values of a triangle will be taken as is. Usually, if no pseudo-coloring is done, all parts of the surface will have equal opacity.

Fancy alpha: Enables physically correct transparencies. The triangle's opacity values will be modified according to their orientation with respect to the viewing direction. Causes the silhouette of the surface to be fully opaque, thus enhancing perception for very transparent surfaces.

Depth sorting: Enables approximate depth sorting. The triangle's centers are presorted along the major coordinates axes.

Create surface: Creates a *Surface* object containing the set of currently visible triangles.

The following items are only present for a subset of derived modules, e.g. *Isosurface* or *SurfaceView*:

Both faces: Indicates that triangles are rendered both from back and front.

Front face: Enables back face culling. Increases rendering speed but may lead to artifacts for non-closed surfaces.

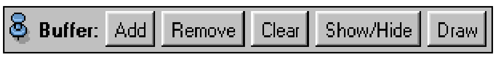
Back face: Enables front face culling. Increases rendering speed but may lead to artifacts for non-closed surfaces.

Triangle normals: Enables per-triangle normals. Shading will be discontinuous at the triangles' edges.

Vertex normals: Enables per-vertex normals. An average normal is computed for all triangle vertices.

Direct normals: An averaged normal is computed for every vertex a triangle. No averaging is performed if two neighboring triangles form an angle bigger than the crease angle set via the Tcl-command `setCreaseAngle`. The default is 30 degrees.

Buffer



This port can be used to modify the list of currently visible triangles. All triangles being visible are stored in an internal buffer. You may add or remove triangles from this buffer via an Open Inventor tab-box dragger. Triangles selected by this dragger will be highlighted, i.e., displayed in red wireframe. If the dragger is not visible, click on one of the buttons to activate it.

Add: Adds highlighted triangles to the buffer.

Remove: Removes highlighted triangles from the buffer.

Clear: Removes all triangles from the buffer.

Show/hide: Shows or hides the tab-box dragger without modifiying the internal buffer.

Commands

`createSurface [name]`
Converts set of visible triangles into a *surface*.

`setAlphaMode {opaque|constant|fancy}`
Triangles may be drawn either opaque or transparent. Two transparent modes are possible: with a constant alpha value *(constant)* or an alpha value varying according to triangle normal *(fancy)*.

`setNormalBinding {perTriangle|perVertex}`
Normals can be bound either per triangle or per vertex. In the first mode the triangles appear flat.

`setPointSize size`
Sets the size of *points*.

`setLineWidth width`
Sets the width of *lines*.

`setOutlineColor color`
Sets color of lines in *outlined* mode.

`setHighlightColor color`
Sets wireframe color of selected triangles.

`setEmissiveColor color`
Sets the emissive color of the surface.

`setSpecularColor color`
Sets the specular color of the surface. This will only take effect if specular lighting has been enabled in port *Draw Style*.

`setShininess shininess`
Sets the shininess of the surface.

`showBox`
Shows box that is used for selecting triangles.

`hideBox`
Hides box that is used for selecting triangles.

6.88 Voltex

Direct Volume Rendering is a very intuitive method to visualize 3D scalar fields. Each point in a data volume is imagined 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 *color map* 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 and 3D texture hardware, to allow for real-time rendering. Note that this currently is not supported by all grahics hardware. Currently hardware acceleration for 2D and 3D textures is available on e.g. SGI Octane (SSI, MXI, MXE), SGI Reality and InfiniteReality, SGI High/Max Impact, HP fx/4 and fx/6. The SGI O2 supports 2D texturing. Most PC graphics cards support 2D texture mapping. Older SGI systems, like Indigo2 Extreme, and the Linux platform currently do not offer hardware acceleration. Using this module on the latter platforms can be extremely slow.

Note that on some systems a significant slow down can occur, if the data set is larger than the available texture memory (which typically is 4 - 16 MB).

Connections

Data `[required]`

The 3D scalarfield to visualized.

Colormap `[optional]`

Colormap used to vizualize the data.

Ports

Mode

max intensity shows the brightest data value along each ray of sight. No absorption is used. This mode is especially useful for very "sparse" data sets, like e.g. angiographic data or images of neurons.

em/ab Uses emission and absorption as described above.

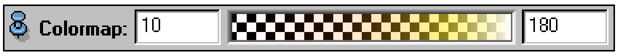
Range

If no colormap is connected, data values are mapped according to this range. Values smaller than the minimum are mapped to completely transparent (no absorption and no emission). Values larger than the maximum appear completely opaque and emit the maximum amount of light. Values in between are mapped proportionally.

Lookup

Only available if a colormap is connected. In *Alpha* mode, the colormaps alpha value is used for both absorption and emission. In *LumAlpha* mode, the colormaps alpha value is used for absorption, while the luminance of is taken for (uncolored) emission. In *RGBA* mode, colored images are generated by using all four channels of the colormap.

Colormap

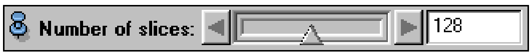


Port to select a colormap.

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, like e.g. SGI O2. *3D* mode needs less setup time and sometimes provides superior quality on high-end machines. 3D texture mapping is not available on the Windows platform.

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.

Update

Click on this button in order to trigger computations necessary to display the volume. Most parameter changes require hitting this button again.

6.89 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.

On 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.

Connections

Slice `[required]`

The *OrthoSlice* module which provides the slice where seed points have to be selected using the middle mouse button.

Data `[optional]`

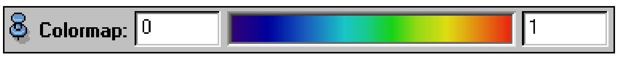
Optional scalar field. If set contiguous regions of this field will be displayed instead of regions of the field the *OrthoSlice* module is attached to.

Colormap `[optional]`

The colormap used to color the 3D regions.

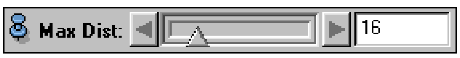
Ports

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.

Action

Provides a button to hide all selected 3D regions.

Chapter 7

Alphabetic Index of Data Types

7.1 Analytical Scalar Field

This data class represents a user defined 3D scalar field based on a regular grid. It provides a port *Expression* by which an arithmetic expression depending on uniform cartesian coordinates x, y, z can be entered that defines the value for each point of the unit cube. The range of values with respect to this (default) domain is indicated as *Component Range*. A data object of type *HxAnnaSccalarField3* has two additional input ports that can be connected to other data objects representing scalar fields on a regular grid, e.g. to image data objects. The predefined variables *a* and *b* are available for referencing such connected data objects in the arithmetic expression, this way a scalar field depending on other regular scalar fields can be defined. Whenever an expression evaluation is triggered to compute the value for a point (x,y,z), the variables *a* and/or *b* will be 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 *HxAnnaScalarField3* data object and entering the point's cordinates by the *Coord* port of that module.

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

- **x** - the x-coordinate of the current point
- **y** - the y-coordinate of the current point
- **z** - the z-coordinate of the current point

If data objects *ScalarField A* or *ScalarField B* are connected to port *InputA* resp. *InputB* the following variables are also defined:

- **a** - the values of input object A
- **b** - the values of input object B

The same C style mathematical and logical operators as well as built-in functions that are availabe for arithmetic expressions can be used here, see *HxArithmetic* module description.

Connections

InputA `[optional]`

Optional scalar field.

InputB `[optional]`

Optional second scalar field.

Ports

Expr

Input field for arithmetic expression defining the scalar field values.

7.2 Analytical Vector Field

This data class represents a user defined 3D vector field based on a regular grid. It provides ports *X*, *Y*, *Z* by which arithmetic expressions can be entered that define the component mappings with respect to uniform Cartesian coordinates x, y, and z. The (default) domain is the unit cube, thus for each point (x,y,z) in it, the associated vector (X,Y,Z) is given by scalar functions X(x,y,z), Y(x,y,z), and Z(x,y,z) which can be specified by the user in the same way as a function for a *HxAnnaScalarField3* data object.

The range of vector magnitudes, defined as Euclidean vectors lengths, is indicated as *Magnitude*.

A data object of type *HxAnnaVectorField2* has three additional input ports named *InputA*, *InputB*, *InputC* that can be connected to other data objects representing scalar or vector fields on a regular grid. There are some predefined variables for referencing such connected data objects in the arithmetic expressions, namely *a*, *b*, *c* for scalar fields and *ax*, *ay*, *az*, *bx*, *by*, *bz*, *cx*, *cy*, *cz* for x-, y-, resp. z-components of vector fields. This way a vector field depending on other regular scalar and/or vector fields can be defined. Whenever an evaluation of the three expressions is triggered to compute the vector associated to a point (x,y,z), each of the variables mentioned that occurs in them will be substituted by the corresponding input values at the same point (x,y,z). For instance the component values of a vector associated to a single point may be obtained by attaching a *PointProbe* module to the *HxVectorField3* data object, entering the point's cordinates by the *Coord* port of that module and setting the *Vector* toggle to *all*.

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

- **x** - the x-coordinate of the current point
- **y** - the y-coordinate of the current point
- **z** - the z-coordinate of the current point

If a scalar data object is connected to any of the ports *InputA*, *InputB*, *InputC* a corresponding variable for access to the data values is also defined, namely:

- **a** - the values of input object A
- **b** - the values of input object B
- **c** - the values of input object C

If a vector data object is connected to any of the ports *InputA*, *InputB*, *InputC* corresponding component variables for access to the data values are also defined, namely:

- **ax, ay, az** - the xyz vector components of input object A
- **bx, by, bz** - the xyz vector components of input object B
- **cx, cy, cz** - the xyz vector components of input object C

The same C style mathematical and logical operators as well as built-in functions that are availabe for arithmetic expressions can be used here, see *HxArithmetic* module description.

Connections

InputA `[optional]`

Optional scalar or vector field.

InputB `[optional]`

Optional second scalar or vector field.

InputC `[optional]`

Optional third scalar or vector field.

Ports

X

Input field for arithmetic expression defining the x-component field values.

Y

Input field for arithmetic expression defining the y-component field values.

Z

Input field for arithmetic expression defining the z-component field values.

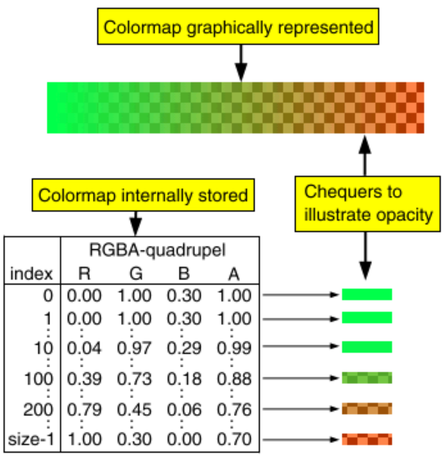
7.3 Colormap

A *Colormap* is a sequence of arbitrary RGBA-quadrupels, where every quadrupel specifes a color by its (R)ed, (G)reen and (B)lue value in the *RGB color model*, each value ranging from 0.0 to 1.0. The fourth value, the so-called (A)lpha value, is a measure for the *opacity*, also ranging from 0.0 to 1.0, where 0.0 means that the color cannot be seen, and 1.0 that the color does not shine through. Internally, these values are stored as *floats*.

Each colormap has a *size*, i.e. the number of RGBA-quadrupels in this colormap. Usually the *size* defaults to 256, but every other positive, non-zero value is possible.

Every RGBA-quadrupel in a colormap can be accessed by an *index*, starting with 0 and ending with *size*-1.

To make a long story short, take a look at this picture:



Beside the raw RGBA values the colormap also stores two *coordinates*, defining a range used for color interpolation. Color lookup requests for a parameter smaller than the minimum coordinates evaluate to the first colormap entry. Requests for a parameter greater than the maximum coordinate value evaluate to the last entry.

Connections

Master

The objects using the colormap.

Ports

Colormap

The *Colormap Port*.

7.4 E-Field Sets

During *E-field simulation* not only a single field is computed, but a separate field for each channel of the hyperthermia applicator. The fields for all channels are written into a single file. In HyperPlan the simulation results are represented as objects of type HxTetraEFieldSet or HxVoxelEFieldSet, depending on whether the simulation has been performed using a tetrahedral patient model or using the raw tissue labels. In both cases the SAR distribution for a given set of power amplitudes and phases can be obtained using the *Superpose* module.

In the tetrahedral case the Superpose module can also be used to compute the resulting electromagnetic field. The field will be stored as an object of type HxTetraComplexVectorField3. In case of a voxel-based simulation this is not possible, because the E-field components have already been pre-multiplied by $\mathrm{sqrt}(\sigma/2)$, where σ denotes the electric conductivity of the tissue. This makes it possible to compute SAR patterns without referencing the labeled volume which has been used for the simulation again. The amount of data to be stored is smaller and data management becomes easier.

7.5 Label Field

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 *Edit 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 object 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.

7.6 Landmark Set

This data type represents specific points or markers in 3D space. It can be used to flag markers in medical MRI images, or to specify pairs or n-tuples of corresponding points in multiple data sets.

An empty set of landmarks can be created by typing

`create HxLandmarkSet`

into the Amira console window, cf. Section 4.1.7. Individual landmarks can be interactively added, repositioned, or removed from a landmark set by means of the *landmark editor*.

Commands

`setNumSets <n>`
Set number of point sets contained in this data set. Upon creation a landmark set contains one set of points. If the landmarks are going to be used to specify corresponding points, it must contain two sets. This can be achieved by typing *Landmarks* `setNumSets 2`, where *Landmarks* is the name of your landmarks icon.

`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.

`appendLandmark <x> <y> <z>`
Appends a new marker to the data object. The new marker will have the same coordinates in all sets.

`setPosition <set> <index> <x> <y> <z>`
Sets the coordinate values of the specified marker `<index>` in a particular set.

7.7 Line Set

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 4.2.5. In order to visualize the line segements of a *LineSet* object the module *LineSetView* can be used. Finally, line sets can be stored using the *AmiraMesh file format*.

Commands

`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.

`addLine p1 [p2 [p3 ...]]`
Adds a new line segment to the line set. The line segment is defined by indices referencing elements of the vertex array.

`removeAllLines`
Removes all lines but leaves the vertex array unchanged.

7.8 Plan Data

Power amplitudes and phase shifts for the hyperthermia applicator may be stored as separate data objects of type HxPlanData. The plan data also contain a scaling factor which determines the total absorbed power inside the patient. A general problem is that the control units of the BSD Sigma applicators use a simplified steering, which allows the user to specify a subset of all possible amplitude and phase settings only. An appropriate conversion of the input parameters will be provided in future.

7.9 SAR Distributions

The specific absorption rate (SAR) is given by $\sigma/2\,|\mathbf{E}|^2$, where σ is the electric conductivity and $\mathbf{E}$ is a complex E-field vector. In Amira it is represented either as an object of type HxTetraScalarField3 or as an object of type HxUniformScalarField3 depending on the simulation method used. In both cases SI units are used, so the SAR is given in Watt/m^3.

7.10 SpreadSheet

This data type represents a spreadsheet. A spreadsheet will be created e.g. by the module *TissueStatistics*.

7.11 Surface

In Amira data objects of type *Surface* are used to represent non-manifold triangulated surfaces. Such surfaces are required as an intermediate step in generating a tetrahedral patient model from the results of segmentation of a 3D image stack.

Surfaces mainly consist of a list of triangles as well as a list of 3D coordinates. Each triangle is defined by three indices pointing into the list of coordinates. Moreover, triangles are grouped into so-called *patches*.

Conceptually, a patch describes the boundary between two adjacent regions (tissue types). These two regions, called *inner region* and *outer region*, are represented by indices into the surface's *material list*. Although required for grid generation, the patch structure of a surface does not necessarily define a valid space partitioning. However, any surface must have at least one patch.

Surfaces may also contain additional data, such as edges, boundary contours, or connectivity information. Because these data can be computed online they are usually not written into a file. If the data are not already present but a certain module requests them, they are recomputed automatically. You may notice a small time delay in this case. Recomputation of the connectivity information can be enforced by the Tcl-command `recompute`.

You may use the *GMC module* to extract boundary surfaces from a *LabelField* describing the results of s 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 enclosed by 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 by a floating point number in range 0...1.

`add -point <x> <y> <z>`
Adds a new point to the surface. The method returns the index of the new point.

`add -triangle <p1> <p2> <p3>`
Adds a new triangle to the surface and returns its index. The triangle will be inserted into the first patch of the surface. If no patch yet exists one will be created.

`refine`
Refines the surface by subdividing all edges. After this operation the surface will contain four times the number of triangles.

7.12 Temperature Sets

Temperature simulations can only be performed for tetrahedral patient models. Similar to the E-field simulation linear independent temperature fields are computed. The set of independent temperature fields is stored as an object of type HxTetraTemperatureSet. From these fields the temperature distribution for any setting of the applicator parameters can be computed by simple superposition. Again this is done by a *Superpose* module. Like the SAR pattern the final temperature distribution is represented as an object of type HxTetraScalarField3. However, notice that temperature distributions are defined on `ThermGrid` models, while E-fields and SAR distributions are defined on `EFieldGrid` models. The latter may contain duplicated vertices at the tissue boundaries to represent field discontinuities.

7.13 Tetrahedral Grid

A data object of type *TetraGrid* represents an unstructured finite-element grid composed of tetrahedra. The geometric information is stored in terms of vertices, edges, faces, and tetrahedra. For instance such data objects are useful as patient models. Like a *LabelField* with its uniform hexahedral grid structure a tetrahedral grid also contains a 'dictionary' of different material types or regions. In addition to the material names the dictionary may contain colors and other parameters related to material properties.

Amira is able to reconstruct tetrahedral grids from 3D image data. This procedure involves several steps, including image segmentation, extraction of boundary faces, surface simplification, and finally grid generation. The tutorial in Section 3.4 of the user's guide illustrated this process in more detail. The actual grid generation step is performed by the computational module *TetraGen*. The quality of a tetrahedral grid may be improved by applying certain operations provided by the *Grid Editor*.

Commands

`hasMaterial <name>`
Returns true if the specified material is defined in the material section of the *TetraGrid*.

`hasDuplicatedNodes`
Returns the number of duplicated nodes, i.e., nodes with exact identical coordinates. Such nodes may be used in order to represent discontinous 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.

7.14 Tetrahedral Patient Models

Tetrahedral patient models are represented as objects of type HxTetraGrid. This class stores geometric information in terms of vertices, edges, faces, and tetrahedra. Although the generation of patient-specific tetrahedral models adds some complexity to the hyperthermia planning procedure, such models nevertheless offer the opportunity for accurate E-field simulation and fast temperature computation. The boundaries between different tissue compartments can be represented accurately using a small number of elements. No artificial steps and rasterization effects occur as in cartesian grids.

Similar to a labeled volume a tetrahedral grid also contains a 'dictionary' of different tissue types. In addition to the tissue names the dictionary may also contain colors and default values for various simulation parameters.

For each patient model usually different versions are stored. The 'generic' patient models have a suffix `grid`. If you want to perform an E-field calculation on a tetrahedral grid using the *FDTD method*, you need a modified grid containing duplicated points at the tissue boundaries (suffix `EFieldGridFD`). The duplicated nodes are required to represent E-field discontinuities at the tissue boundaries. Use module *DuplicateNodes* to create such a modified grid. When you perform an E-field calculation using the *FE method*, another modification of the grid is stored as an output file (suffix `EFieldGridFE`). It is essentially the same as the original patient model, however, it additionally contains an edge list. This is needed for the special field representation of the finite element method (called *Vector elements*).

When you perform a temperature simulation, another modification of the grid is stored as an output file (suffix `ThermGridFD` or `ThermGridFE`). These grids don't contain any duplicated points, nor do they contain an edge list.

7.15 Tomographic Image Data

Tomographic images are the starting point to create an individual model of the patient as required in radiotherapy and more advanced methods in cancer therapy. Conceptually, tomographic images are interpreted as scalar fields in Amira. At each point in space a voxel intensity is defined. Usually, CT images are used as input because fat, muscle, and bone structures can easily be identified using this modality. In CT images pixel intensity corresponds to tissue density. The density is measured in Hounsfield units. In Amiranormalized Hounsfield units are used. A value of -1000 corresponds to air, and a value of 0 corresponds to water. Bones usually have a density value of 140 and higher.

When a set of equi-distant slices is loaded into Amira, an object of type HxUniformScalarField3 is created. For non-equidistant slices a HxStackedScalarField3 is used.

7.16 Vertex Sets

The *HxVertexSet* class is an abstract base class. It is derived by many other Amira data objects containing a list of 3D vertices, for example *landmark sets*, *surfaces*, or *tetrahedral grids*. *HxVertexSet* provides an interface allowing other modules to access the vertices in a transparent way.

In order to visualize the vertices of a vertex set you may use the *VertexView* module.

Commands

`applyTransform`
This command changes the coordinates of the vertices of the data object according to the object's current transformation matrix. This matrix can been 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.

Chapter 8

Alphabetic Index of File Formats

8.1 ACR-NEMA / DICOM

The ACR-NEMA file format and its successor the DICOM format are widely used to store medical images of various types. In Amira the import of stacks of axis-aligned CT or MRI images stored in these formats is supported. The read routine checks whether the spacing between subsequent slices is equal or not. In the first case an image volume with uniform coordinates is created, otherwise so-called stacked coordinates are used. Both, image data with uniform or stacked coordinates can be segmented using the *image segementation editor* and can be converted into polygonal models using the *GMC* module. If you want to create a uniform data set from a stacked one you may use the *arithmetic module*.

When reading ACR-NEMA or DICOM files all files of the data volume must be selected simulanteously in the file browser. This is done by clicking the first file and then shift-clicking the last one. Amira assumes that a file is a ACR-NEMA or DICOM file if the file name suffix is `.ima` or `.ani` or if the file name matches a DICOM unique instance ID, e.g. `1.3.12.2.1107.5.1.2.20395.19980429221554713.4`. Individual files are automatically uncompressed if they were compressed using `gzip` and if the file name ends with the suffix `gz`.

8.2 Amira Mesh 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 analysed and interpreted. For example, a tetrahedral grid is expected to have a one-dimensional array *Nodes* containing entries of type `float[3]` called *Coordinates*, as well as a one-dimensional array *Tetrahedra* containing entries of type `int[4]` called *Nodes*. If the Amiramesh file contains an entry *ContentType* in its parameter section the value of this parameter directly determines what kind of Amira data object is to be created.

A first example. In order to describe the syntax of an AmiraMesh file we first give a short example. This example describes a scalar field defined on a tetrahedral grid. *Concrete examples of how to encode other data objects are given below.*

```
# AmiraMesh ASCII 1.0

define Nodes 4
define Tetrahedra 1
```

```
Parameters {
    Info "This is an AmiraMesh example",
    Pi 3.1459
}

Materials { {
    Name "Stone",
    Color 0.8 0.3 0.1
} {
    Name "Water",
    Color 0 0.3 0.8
} }

Nodes { float[3] Coordinates } = @1
Tetrahedra { int[4] Nodes } = @4

Nodes { float Values } = @8
Tetrahedra { byte Materials } = @12

Field { float Example } = Linear(@8)

@1
0 0 0
1 0 0
0 1 0
0 0 1

@4
1 2 3 4

@8
0 0 0 1

@12
1
```

The first line of an AmiraMesh file should be a special comment including the identifier `AmiraMesh`. Moreover, if the tag `ASCII` is given in this line all data arrays are stored in plain ascii text. If the tag `BINARY` is given, the data arrays are stored in IEEE big-endian binary format. Note, that the header section of an AmiraMesh file is always given as ascii text.

The statement `define Nodes 4` defines a one-dimensional array of size 4. Later on, this array can be referenced using the name `Nodes`. Similarly, a statement `define Array 100 100` defines a two-dimensional array of size 100 x 100. The actual kind of data stored per array element will be specified later on.

The optional section `Parameters` allows the user to define arbitrary additional parameters. Each parameter consists of a name (like `Pi`) and a value (like `3.1459`). Values may be one or multiple integer or floating point numbers or a string. Strings have to be quoted using a pair of `"`-characters.

The optional section `Materials` allows the user to define additional material information. This is useful for finite element applications. The material section consists of a comma-separated list of parameters just as in the `Parameters` section.

The statement `Nodes { float[3] Coordinates } = @1` specifies that for each element of the array `Nodes` defined earlier three floating point numbers (floats) should be stored. These data are given the name `Coordinates` and a tag in this line and will appear below as a tagged block with the marker `@1`. Such data markers must always begin with the letter `@`.

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 compatibilty. 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*

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.

Fields with stacked coordinates

A field with stacked coordinates has uniform pixel spacing in x- and y-direction, but slices may be arranged arbitrarily in z-direction. This type of coordinates is commonly used to encode 3D medical images with non-uniform spacing.

In order to encode a 3D scalar or vector field with stacked coordinates you have to 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 degenerated 3D data sets with one dimension being 1 choose equal minimum and maximum coordinates in that direction.

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 explicitely 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 x-, y-, and z-direction. 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.

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 explicitely.

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.

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 occurence 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 is supposed to 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.

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.

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 arbitary number of additional vertex data values can be defined. Multiple values should be distinguished by denoting them `Data2`, `Data3`, and so on.

Colormaps

An Amira *colormap* consists of a one-dimensional array of RGBA components accompanied by two numbers *min max* specifying which data window should be linearly mapped to the RGBA values. The RGBA array should have 256 elements in order to be able to edit the colormap using the *colormap editor*.

Colormaps are encoded in an AmiraMesh file as follows:

```
# AmiraMesh ASCII 1.0

define Lattice 256

Parameters {
    ContentType "Colormap",
    MinMax 10 180
}

Lattice { float[4] Data } = @1

@1
1 0 0 0
1 0.00392157 0 0
1 0.00392157 0 0.00392157
1 0.0117647 0 0.00392157
1 0.0196078 0 0.0196078
1 0.027451 0.00392157 0.0196078
...
```

The RGBA values are stored in floating point format. A component value of 0 means no intensity (black), while a component value of 1 means maximum intensity (white). The fourth component denotes opacity (*alpha*). Here a value of 0 indicates that the color is completely transparent while a value of 1 indicates that the color is completely opaque.

8.3 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 is also described the TIFF section.

8.4 Encapsulated Postscript

Amira is able to save snapshots as well as individual slices of a 3D image data set in *Encapsulated Postscript* format. You may directly send EPS files to a Postscript printer, or you may include these files in many standard desktop publishing programs. The EPS files produced by Amira contain bitmaps rather than vector information.

The import of EPS files is not supported.

8.5 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 is 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 identifes 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 brackets { and }. Inside these brackets `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 make 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 differnt 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 `Outer-Region` 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 indentifier `Vertices` followed by the number of vertices. First branch-

ing points should be specified, then points on boundary curves, and then inner points of the patches. In the definition of boundary curves and patches the vertices are referenced by indices starting from 1, not from 0.

The following example describes the surfaces of three connected tetrahedra which are assigned to three different regions. Some of the definitions are optional. Really necessary are only the list of vertex coordinates as well as the definition of the patches. Boundary curves and surfaces may be reconstructed from this.

```
# HyperSurface ASCII

Vertices  6
     0.0  0.0  0.0
     0.0  0.0  1.0
     0.0  1.0  0.0
    -1.0  0.0  0.0
     1.0  0.0  0.0
     0.0 -1.0  0.0

NBranchingPoints 2
NVerticesOnCurves 2

BoundaryCurves 3
{
    Vertices 3
        1  3  2
} {
    Vertices 2
        1  2
} {
    Vertices 3
        1  4  2
}

Patches 5
{
    InnerRegion Material1
    OuterRegion OUTSIDE
    BranchingPoints 0
    BoundaryCurves 2
        1  -2
    Triangles 3
        5  1  3
        1  5  2
        5  3  2
} {
    InnerRegion Material2
    OuterRegion OUTSIDE
    BranchingPoints 0
    BoundaryCurves 2
        3  -1
    Triangles 2
        3  1  4
        2  3  4
} {
    InnerRegion Material3
```

```
        OuterRegion OUTSIDE
        BranchingPoints 0
        BoundaryCurves 2
            -3  2
        Triangles 3
            4  1  6
            2  4  6
            1  2  6
    } {
        InnerRegion Material1
        OuterRegion Material2
        BranchingPoints   0
        BoundaryCurves 2
            2  -1
        Triangles 1
            3  1  2
    } {
        InnerRegion Material2
        OuterRegion Material3
        BranchingPoints 0
        BoundaryCurves 2
            -3  2
        Triangles 1
            1  2  4
    }

    Surfaces 3
    {
      Region Material1
      Patches 2
         1  4
    } {
      Region Material2
      Patches 3
         2  -4  5
    } {
      Region Material3
      Patches 2
         3  -5
    }
```

8.6 Icol

This is a simple ASCII file format coming in two variants, indexed and non-indexed, to store colormaps. The *Icol* file format and the *Icol Colormap Editor* originate from the Graphics and Visualization Lab (GVL) of the Army High Performance Computing Research Center (AHPCRC), Minnesota. The *Colormap Editor* in Amira shares many ideas with the original AHPCRC tool. The structure of an *Icol* format can be immediatly understood by looking at the examples in Amira's demo data directory.

8.7 Inventor

The *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 Inventor. Since Amira is built on top of Inventor, it naturally supports this format. However, Amira has added a lot of special purpose components to Inventor. Therefore currently some geometries which can be displayed in Amira's 3D viewer cannot be saved in Inventor file format.

8.8 JPEG Image Format

JPEG is a standard format which can be used to store RGB and greyscale images in a highly compressed form. However, note that the compression algorithm is lossy. 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 is also described the TIFF section.

8.9 Leica Microscope 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.

8.10 Leica Microscope Slice Series

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.

8.11 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 is also described the TIFF section.

8.12 PNM Image Format

This format includes the PPM, PGM and PBM image formats. These formats are used to store RGB color images, greyscale 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.

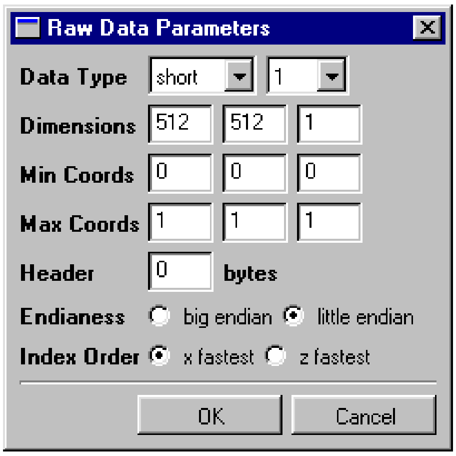


Figure 8.1: Amira's raw data read dialog.

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 also is also described the TIFF section.

8.13 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 Amiracan not recognize the file format automatically, a dialog will popup. Within this dialog choose *Raw Data* as file format and click *OK*. Since the raw data file does not contain any information on how to format the data, some user specifications are required. Amira will bring up a dialog:

Now adjust the parameters:

- **Data Types.** Primitive type of data: byte for 8 bit data, short for 16 bit data, int32 for 32 bit integer data, float for 32 bit float data, double for 64 bit floating point data.

  In addition the number of data values per data point has to specified: 1 for scalar data, 3 for vector data, 6 for complex vector fields.
- **Dimensions.** Size of the three dimensional array. If wrong dimensions are specified, the data will be scrambled.
- **Min/Max Coords.** The bounding box of the data. These parameters are not as critical as the other ones. In particular the bounding box can be adjusted afterwards using the crop editor.
- **Header.** Many file formats consist of a raw data block with a prepended header. If such files are read with this method, the size of the header can be specified here. The header will then simply be skipped, when reading the file
- **Endianess.** The byte order of data types other than byte is system dependend. If you read your files on the same type of processor as on which they have been produced, the default setting will be ok. But if you read data produced e.g. under Linux (little Endian), on an SGI (big Endian), you have to 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.

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 to short header. If most of the volume looks ok, but is shifted in x-direction, then you probably have specified a wrong header. If the data range looks ok, but the data seems scrambled, then you have specified wrong dimensions, or a wrong index order.

8.14 SGI-RGB Image Format

This is the SGI file format for RGB and greyscale 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 is also described the TIFF section.

8.15 STL

STL is a CAD format for Rapid Prototyping. It is a faceted surface representation, i.e. a list of the triangular surfaces with no adjacency information. Currently the ASCII version of the format is supported for writing Amira objects of type Surface.

8.16 Stacked-Slices

This file format allows to read a stack of individual image files with optional z-values per slice. The slice distance needn't 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.

Exemplary layout of a description file:

```
# Amira Stacked Slices
pathname /usr/data/pictures/
picture1.tif 10.0 #This is a comment
picture7.tif 30.0
picture13.tif 60.0
colstars.jpg 330.0
end
```

`# 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 or = has to separate the tag ”pathname” from the actual pathname. Default is the subdirectory of the description file.

`picture1.tif 10.0` is the name of the slice and its z-value, separated by a blank.

`end` defines the end of the description file.

With a # after z-values or the pathname comments can be included into the file.

8.17 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,

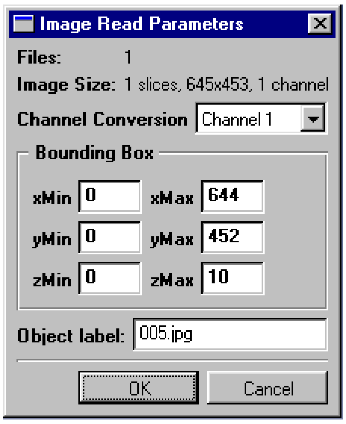


Figure 8.2: Amira's channel conversion dialog.

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, irrespectively by 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 if `image.####.tif` the files actually being written will be named `image.0000.tif`, `image.0001.tif`, and so on.

Note, that not all variants of the TIFF format are supported. In particular, the following limitations apply:

- The number of channels must be 1, 3 or 4.
- The number of bits per pixel must be 8 or 16 (1 channel images only).
- Images defined in YbCbR colorspace can not be read.
- Tiled images can not be read.
- Only scalar fields consisting of bytes can be saved.

The Channel Conversion Dialog

When reading 2D image files a special dialog window will be popped up. This dialog asks the user to specify how the 2D images should be converted into Amira data objects. In addition, the world coordinates of the resulting 3D data object can be adjusted. The channel conversion dialog looks as depicted in Figure 8.2.

First of all, the dialog displays the number of files to be read, the number of 2D slices (in most cases equal to the number files), the size of a 2D slice in pixels, as well as the number of channels stored in the files. An option menu lets you select whether a 1-component uniform scalar field should be created or a 4-component RGBA color field. Depending on the type of input not all options may be active. The meaning of the individual items is described below.

- **Maximum.** The maximum value of the red, green, and blue channel is stored in a uniform scalar field (3 and 4 channel input only).
- **Weighted Average.** A grayscale uniform scalar field is created according to the NTSC formula, `I=.3*R+.59*G+.11*B` (3 and 4 channel input only).
- **Channel 1.** The first channel of the input is converted into a uniform scalar field (will always be active).
- **Channel 2.** The second channel of the input is converted into a uniform scalar field (3 and 4 channel

input only).

- **Channel 3.** The third channel of the input is converted into a uniform scalar field (3 and 4 channel input only).
- **Channel 4.** The fourth channel of the input is converted into a uniform scalar field (4 channel input only).
- **Color Field.** An RGBA color field is created (4 channel input or 1 channel input with additional colormap).

If the first image file contains a colormap the map will be loaded as a separate object if *Channel 1* is selected, or it will be used to compute an RGBA color field if *Color Field* is selected.

Moreover, the dialog allows you to adjust the bounding box of the resulting image data set. The bounding box specifies the world coordinates of the center of the lower left front voxel (min values) and the center of the upper right back voxel (max values). For example, if your input is 256 pixels wide and the size of each voxel is 1mm, then you may set *xMin* to 0 and *xMax* to 255. The bounding box of a data object may also be changed later using the *ImageCrop Editor*.

Chapter 9

Alphabetic Index of Editors

9.1 Color Dialog

The *Color Dialog* provides an interface for selecting colors. The dialog pops up whenever a single color shall be changed within Amira.

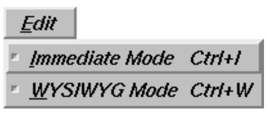
The *Color Dialog* therefore provides serveral different interface components: a menu bar, color sliders, color cells, a color picker, and five buttons.

The menu allows the user to set a few options for his work with the color dialog. The sliders (and the corresponding text fields and arrow buttons) let the user choose a color by hue, saturation, and value (HSV), or by the levels of red, green, and blue (RGB). The color picker allows the user to pick a color by sight. Moreover, the user can choose a predefined color from a palette of custom colors displayed by some of the color cells. The buttons are used to apply or reset changes and to quit the dialog.

Here is a detailed description of the interface components:

Menu Bar: The menu bar holds two pop up menus: the *Edit* and the *Options* menu.

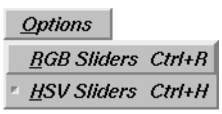
There are two items in the Edit menu allowing the user to toggle between different editing modes: the *Immediate Mode* and *WYSIWYG Mode*.



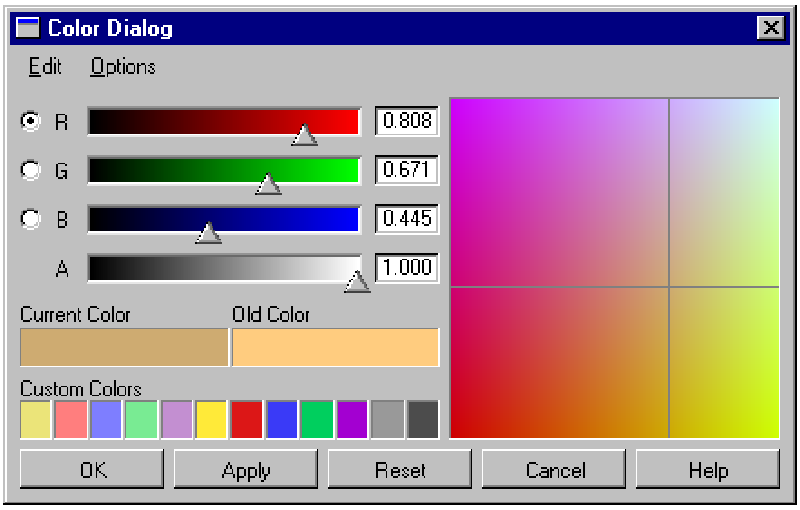
If Immediate Mode is selected, each modification of the color to be changed is effective immediately, e.g. if you are changing the background color of the 3D viewer and Immediate Mode is active, the effect of every slider or picker operation can be observed directly in the viewer.

The WYSIWYG (*W*hat *Y*ou *S*ee *I*s *W*hat *Y*ou *G*et) Mode 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.

The user has 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 Option Menu. Each of the color sliders is accompanied with a text edit field to view the exact value of the current color component and to set its numerical value. The component values are always in the range from 0 to 1. Furthermore, two arrow buttons can be used to move each slider respectively.

On the left side of each slider (except the alpha slider) an additional check button is displayed allowing the user 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 pop ups.

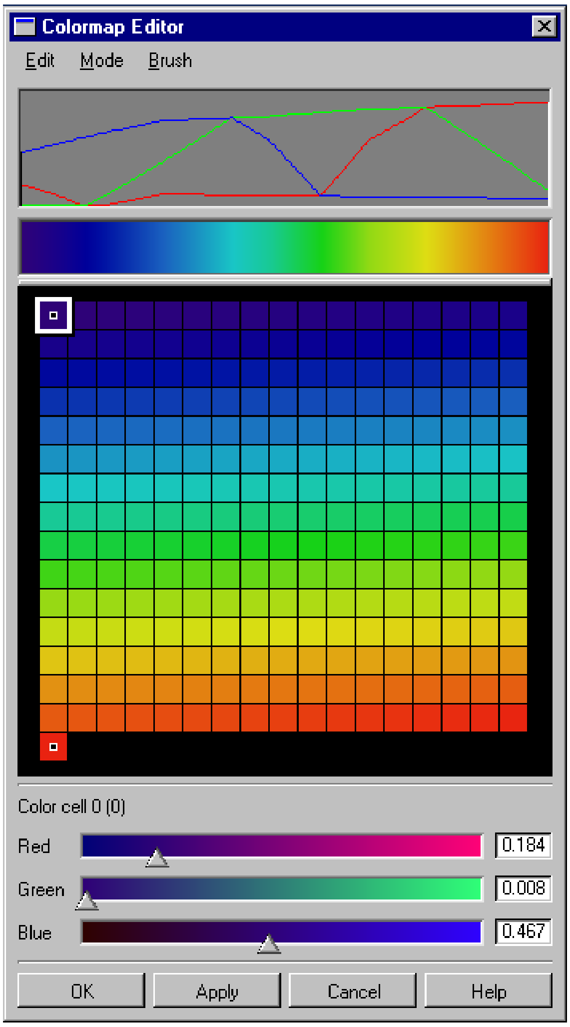
The Drag and Drop mechanism is applied to store and restore colors. The colors can be copied arbitrary between all cells (except that a color could not be stored to the old color cell).

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 (done by the toggle button on the left side of each slider) the two other components of



the color can be set with the picker. One component corresponds with the vertical extent of the color picker and the other with the horizontal. The selected component can not 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.

   As you move your mouse around the picker, the current color is set to the color beneath the intersection of the vertical and horizontal grey line.
3. release the mouse button when the grey cross is on top of the color you like.

9.2 Colormap Editor

To change existing *colormaps* push the edit button (pencil icon) which is visible when the colormap is selected. The colormap editor pops up. As you can see there is a menu bar near the top border of the window with the sub menus *Edit*, *Mode* and *Brush*, which help you to control the behavior of the colormap editor and to change components of the chosen colormap.

Below the menu bar a so-called ”color chart” is displayed. The red, green and blue lines are the graphs of the *color channel* components of the colormap, the underlying color model is RGB (at startup). The axes are not shown directly, the x-axis ranges from the lowest to the highest colormap index and the y-axis from 0.0 to 1.0 according to the RGB model. Thus, a point on a color line indicates the amount of color for the corresponding color channel and color index. You can manipulate the course of a line by setting a brush onto any of its points and moving it up or down. A brush is set by pressing a mouse button, the left one for the red line, the middle one for the green line and the right one for the blue line.

Below the color chart a ”color bar” is displayed which is a larger version of the one used for display in the colormap port. Its only function is to show you the actual appearance of the modified colormap in a smoother way by linearly interpolating the colors between the indices.

The largest area of the editor window is occupied by the ”color buttons” which represent the color in every position (index) of the chosen colormap, each position is called a ”color cell”. You can set the *focus* which is the target index of modifications applicable by the color sliders (see below), and it is also possible to modify the colormap by dragging a color cell. The index of the focus color cell is shown below the color buttons.

In the lower area of the window there are some ”color sliders”, one for each color channel by which you can modify color channel values with respect to the focus cell. This means that the values of neighboring cells are affected as well as you can see in the color chart. The sliders are manipulated by dragging the small triangles, alternatively values in fix-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.

The important elements for manipulation

1. **Focus**
   The focus marks the color cell for which you can change the value by the sliders. Its representation is a surrounding black-and-white box of a color cell and 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.
2. **Knot**
   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.

Description of every item

Let us first take a course in what the items of this colormap editor are and what they do.

A. **Menu Bar**
The menu bar consists of the three sub menus ”Edit”, ”Mode” and ”Brush”.

a) **Edit Menu**
This sub menu 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.

1. **Undo**
   If you think that your last changes to the colormap have been a mistake this entry lets you take back your last changes. You can easily go back to a point of editing when you were content with the colormap and

start editing again. The colormap editor memorizes up to 50 of your last changes.

2. **Redo**
   If you took back 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”.
3. **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.
4. **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.
5. **Snap Red — Snap 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.
6. **Snap Green — Snap 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.
7. **Snap Blue — Snap 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.
8. **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 neighbouring knots of the focus leaving the other color channels untouched.

b) **Mode Menu**
Here you control the appearance of the colormap in a certain color model or whether the alpha value (transparency) is shown or not.

1. **Red / Green / Blue**
   By this entry you select the RGB color model.
2. **Hue / Saturation / Value**
   When you activate this entry the display is changed according to the HSV color model.
3. **Show Alpha**
   Each color model has got a so-called alpha channel which determines the transparency of the colormap. When you handle a colormap with small alpha values you might be irritated by the transparency shown in the color bar and you perhaps will not be able to make out the correct color. By the toggle *Show Alpha* you control whether the alpha channel is shown or not. At startup it is not shown.
4. **Immediate Mode**
   If this mode is active all changes made to the current colormap are immediately passed to the colormap object that has been selected for editing with all impacts on dependent objects.

c) **Brush Menu**
This has only relevance to the color chart. Here you can set the brush type, which is one of the types shown in the ”Brush Menu”.

B. **Color Chart**
The color chart shows the graphs of the colormap’s colorchannel components. Four or three color channel

graphs are displayed (depending on whether *Show Alpha* is set or not). Three color channels are always displayed according to the currently set color model. The red graph shows the course of the first color channel (R / H), the green the course of the second (G / S) and the blue the third's course (B / V). If *Show Alpha* is enabled the alpha channel is shown in black color. 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 chequer. For an alpha value of 0.0 you see no color information but only the white and black chequer, for a value of 1.0 you do not see a chequer 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 botom. 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.

A knot is represented by a small black box in the middle of the color cell with white outlines, and the focus has additional black and white surroundings.

E. **Color Sliders**
In the lower area of the *Colormap Editor* window there are three or four color sliders which give you the opportunity to modify the values of the colormap cell with the focus on it for all color channels. You see four or three sliders depending on whether "Show Alpha" is set or not. The first three sliders correspond to the color model's channels and the fourth lets you modify the alpha value. On the left side of each slider you see the name of the corresponding color channel e.g. "Red", "Hue", etc. On the right side there is text entry which shows the actual value of the color channel for the color cell with the focus.

Then in the middle you see the true slider which is a rectangle with a color course depending on the model, e.g. in case of a red slider a course starting with black on the left (no additive red value) and going on to bright red on the right side (adding the full spectrum of red color). Attached to the rectangle is a small triangle which lets you manipulate the value by dragging it to the left or to the right (see "How to modify a colormap").

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.

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*
The only (but possibly most comfortable) manipulation you can do is to move a brush over a color channel and therefore raising / lowering it in the brush's region.

1. **Choosing the right brush**

You choose the brush in the menu bar's sub menu "Brush", where you see a choice of three brushes: a square, a circle and a diamond. By its form you determine how much a colormap entry will be raised/lowered, e.g. when you select the diamond brush all channels you touch will be raised/lowered by a small amount on the brush's edges corresponding to the sharp form of the diamond on the outer sides. But with a square brush all color channels will be raised/lowered by the same big amount in response to the square's form, i.e. all positions have the same height.

2. **Choosing the color channel**
   Now it's up to you to determine which color channel you want to edit. This is controlled by your mouse buttons. When you push the first mouse button, you edit the first color channel, with the second the second and by pushing the third button you modify the third channel. If you want to manipulate the alpha channel (provided that *Show Alpha* is set) you have to hold the Control key while pushing the first mouse button.

   This is summarized by the following table:

```
Color channel  Mouse button  Control key
first (R/H)    first         No
second (G/S)   second        No
third (B/V)    third         No
alpha          first         Yes
```

3. **Editing the channel**
   This is simply done by clicking with the desired channel's mouse button (including pressing the Control key if you want to edit the alpha channel) on the color chart. The brush appears and you can move it all around the color chart. The brush vanishes if you release the button. Depending on whether you start above or below the channel the brush is fixed to lower or raise the channel.

   When the brush touches the corresponding color channel (the others will not be affected) it moves the color channel up / down (depending on from which side it comes) to the first pixel outside the brush's area. No color channel can be moved below its minimum or beyond its maximum. When you move the color channel to its minimum / maximum it simply stops there and is not affected by further movements.

   When you want to raise / lower a channel already in its minimum / maximum you have to move with the mouse pointer (i.e. the future center of the brush) without editing the color channel (i.e. the mouse buttons not pressed) approximately over it. Only in this position it is possible to revert the usual behaviour, i.e. you can raise the channel although you are above the color channel and lower it although you are below it.

   The new colors are displayed in the color bar and in the color buttons. If you modify the color channel in the focus cell the new values will also appear in the color sliders i.e. as a new position of the small triangle and as a new value shown by the text entry.

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.

1. **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 disappers. 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.
2. **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.

3. **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.
4. **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 color chart is refreshed with the new color course. The color sliders stay the same because the focus cell's 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 into the true slider's area.

All modifications have 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. This could be best explained graphically (in the color chart).

1. **Setting a value explicitly in the text field**
   Simply do the same things as when working with an editor, i.e. click in the text field to activate the cursor, erase the old value and write a new value with the keyboard and press Enter to submit your value. This change will be visible in a corresponding position of the small triangle of the slider and in the color buttons and color chart as a changing of colors and a modified graph respectively.
2. **Setting a value by dragging the small triangle**
   You can see the modification interactively by dragging the small triangle all around the slider. The values and displays will be refreshed while you drag the small triangle. Modifying the focus cell this way lets you have an intuitive control of what new values will be the best for your needs.
3. **Setting a value by clicking in the true slider's area**
   Last but not least you can click on an arbitrary position in the color bar of the slider to set the new value. This is a very inaccurate way to modify the value because you can neither set the exact value nor watch how the position will modify the surrounding colormap entries.

The values and displays are refreshed in the same way as described in the previous sections.

9.3 Digital Image Filters

This editor allows you to apply certain digital image filters to 3D image data sets, such as smoothing, unsharp masking, or morphological operations. Some filters directly operate in 3D while others are applied

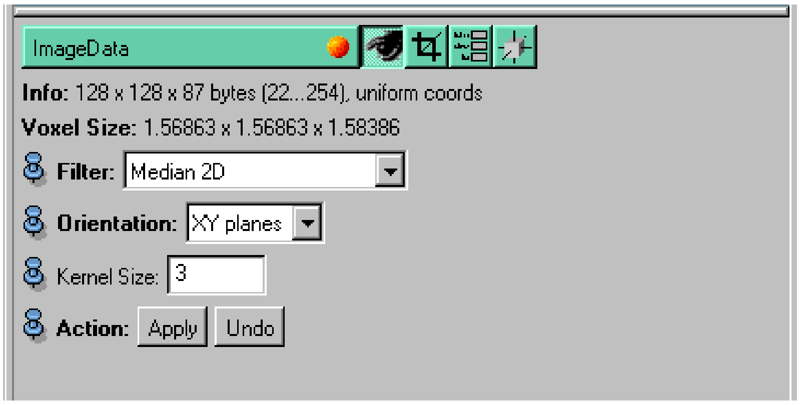


Figure 9.1: Digital image filters.

in two-dimensional slices only. In the latter case the orientation of the 2D slices can be selected via an option menu.

Currently, the following filters are supported:

- *2D Median Filter*
- *2D Minimum Filter*
- *2D Maximum Filter*
- *Unsharp Masking*
- *2D Histogram Filter*
- *3D Gauss Filter*
- *3D Lanczos Filter*

Ports

Filter Specifies which filter is to be applied to the image.

Orientation This port will only be visible if a 2D filter has been selected. It allows you select whether the filter should be applied in the XY, XZ, or YZ slices of the 3D image.

Action Starts computation.

2D Median Filter

The median filter is a simple edge-preserving smoothing filter. It may be applied prior to segmentation in order to reduce the amount of noise in a stack of 2D images.

The filter works by sorting pixels covered by a NxN mask according to their grey 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 mask. An odd value is required.

2D Minimum Filter

The minimum filter replaces the value of a pixel by the smallest value of neighboring pixels covered by a NxN mask. The size of the mask can be adjusted via the input field *kernel size*. A value of 3 denotes a 3x3 mask.

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.

2D Maximum Filter

The maximum filter replaces the value of a pixel by the largest value of neighboring pixels covered by a NxN mask. The size of the mask can be adjusted via the input field *kernel size*. A value of 3 denotes a 3x3 mask.

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.

Unsharp Masking

This filter sharpens a stack of 2D images using an unsharp mask. The unsharp mask is computed by a Gaussian filter of size *kernel size*. The scaled difference of the orginal image and the blurred image is then added to the original. Thereby the sharp features of the image get emphasized. The scaling factor determining the amount of sharpness can be adjusted via an text input field.

2D Histogram Filter

This filter performs a so-called *contrast limited adaptive histogram equalization (CLAHE)* on the data set. The CLAHE algorithm partitions the images into *contextual regions* and applies the histogram equalization to each one. This evens out the distribution of used grey values and thus makes hidden features of the image more visible.

3D Gauss Filter

The Gauss filter smooths or blurs an image by performing a convolution operation with a Gaussian filter kernel. The text fields labeled *kernel size* allow you to change the size of the convolution kernel in each dimension. A value of 3 denotes a 3x3 kernel. Odd values are required. The text fields labeled *sigma* allow you to adjust the width of the Gauss function relative to the kernel size.

3D Lanzcos Filter

The Lanzcos filter can be used to sharpen images. It performs a convolution with a Lanzcos kernel:

$$f(z) = \frac{\sin(\pi x)\sin(2\pi x)}{2\pi^2 x^2},\ |x| < 2$$

The kernel size in each dimension can be adjusted using the parameter inputs *kernel size*. A value of 3 denotes a 3x3 kernel. Odd values are required.

9.4 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 do this push the edit button of a selected tetrahedral grid (see the icon above). Then some additional button controls will appear in Amira's *Working Area*. The user interface is depicted in Figure 9.2. In case you select some other *Selector* or *Modifier* other ports might be shown. The editor works in conjunction

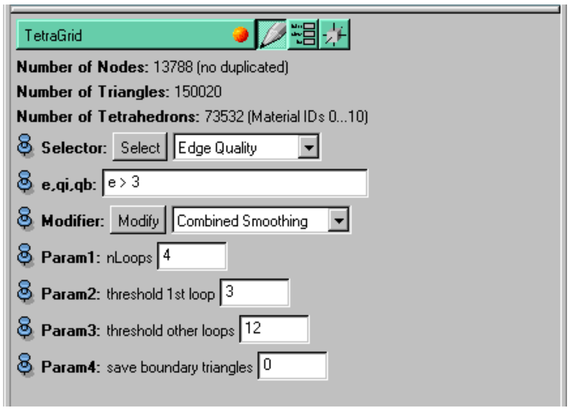


Figure 9.2: Tools for improving the quality of tetrahedral grids.

with the *GridVolume* module. If such a module is not already active, an instance of it will be automatically created when the editor is invoked.

Tetrahedra can be distorted in different ways:

- *Slivers* contain 4 nearly coplanar vertices forming a quadrangle.
- *Caps* consist of a triangle and a 4th vertex 'just above' it.
- *Needles* consist of a triangle and a 4th vertex 'far away'.
- *Wedges* are characterized by one sharp edge.

Using the tools of the *Select* menu

- **Index Selector**
  This tool selects tetrahedra according to their index (i).
- **Aspect Ratio Selector**
  This tool selects tetrahedra according to their determinant (d), diameter ratio (R), or other dimensionless quality measures (r1, r2).
- **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. For an equilateral tetrahedron all dihedral angles are ca. 70 deg, all solid angles are ca. 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**
  This tool selects tetrahedra according to their edge lengths.

Using the tools of the *Modify* menu

- **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 inner vertex is moved to that location if the quality of the adjacent tetrahedra is improved. Otherwise the midpoint between the old location and the center of mass is examined. Parameter *nLoops* defines the number of smoothing loops.
- **Optimization Smoothing**
  This tool improves mesh quality by moving inner vertices. For each inner vertex a new location is determined that optimizes the quality of the adjacent tetrahedra. Parameter *nLoops* defines the number of

smoothing loops, 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.

- **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 if the face is a boundary face and if 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 of the tetrahedra involved. Parameter *threshold* defines a threshold for tetrahedron quality. If all adjacent tetrahedra have a quality better than *threshold*, a face is not examined for flipping.
- **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.
- **Remove Inner Vertices**
  This tool improves mesh quality by removing inner vertices. If the number of tetrahedra adjacent to an inner vertex is 4, that vertex is removed. This should always improve mesh quality. If the number is 6, the vertex is removed if mesh quality is improved.
- **Bisect Inner 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 or Laplace 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', the other parameters define thresholds for tetrahedron quality which are applied in the first loop and the other loops, respectively.

We recommend to apply the *Combined Smoothing* and the *Remove Inner Vertices* methods.

9.5 Image Segmentation Editor

The *Image Segmentation Editor* (formerly called GI for General Image editor) is a tool for interactively segmenting 3D image data. Image segementation is the process of dividing an image slice or an image volume into different subregions (also called segments). In the bio-medical context, these segments can e.g. be different organs or tissue types. In Amira segmentation is done by assigning a label to each voxel. The volume of labels is stored in a *Label Field.* The *Image Segmentation Editor* lets you edit such a *Label Field*, which can e.g. serve as input for a *surface reconstruction.*

In order to activate the editor push the edit button of a selected *Label Field* (see the icon above). Then an instance of the editor will be created and a new window titled *Image Segmentation Editor* pops up. cf. Figure 9.3.

The major parts of this new window are:

- **Material List:** In the upper left of the ISE window a list of the materials is presented. Here you can add, remove, and rename materials, and select which material is currently active.
- **Tool box:** Below the material list, several tools for interactive manipulation of the segmentation can be selected. Depending on the currently selected tool, additional widgets show up in the options frame.
- **Info Area:** Below the tool box, some basic information (like cursor position) is displayed.

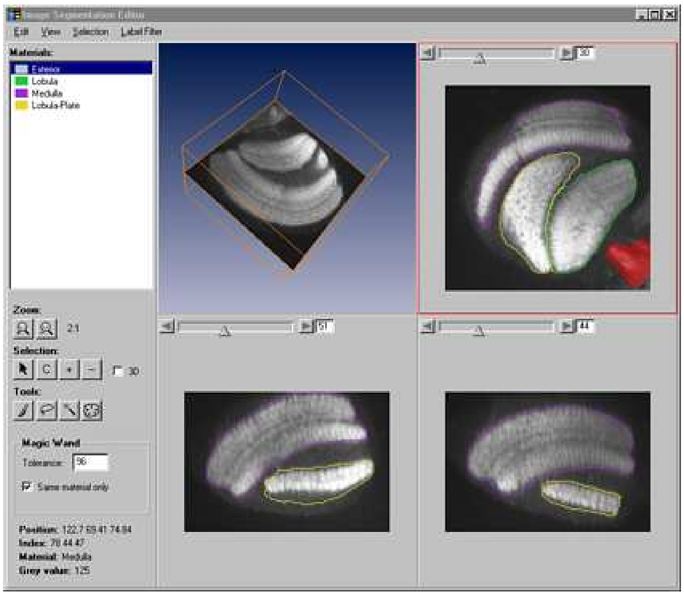


Figure 9.3: The Image Segmentation Editor window.

- **Menu Bar:** From the menu bar additional tools and filters can be accessed. Menu entries with (...) behind the name open an additional dialog window.
- **Image Viewer(s):** The biggest part of the window is covered by one or four image viewers, which display slices through the image data, a visualization of the labels and a current selection.

Basic principle for interactive segmentation

The basic idea of interaction in the ISE is a 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 using the right-mouse-popup menu in the material list.

Every pixel can only be assigned to one material.

Note that even with the advanced tools provided in Amira, image segmentation can be a time-consuming process. Due to limited main-memory and for performance reasons, there is only a limited undo space for 2D interaction and most 3D operations cannot be undone. Therefore it is highly recommended to **frequently save the label field** during the process of segmentation.

To get started it might be a good idea, to choose the image segmentation demo from the Users Guide's demo section and start by modifiying an existing label field.

Manipulating the material list

In the material list, the current or active material can be selected with the left mouse button. A special behaviour 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 already are assigned to the newly chosen material, are added to the current selection (in the current slice or in the whole volume, depending on the value of the *3D* toggle button in the Selection Command area). If Control is held down, the voxels with this material are deselected. Holding Alt does the same as Shift, except that it clears the selection before selecting the new voxels.

All other manipulation options can be accessed via a popup-menu, which shows up, when the right mouse button is pressed on a particular material entry. The menu entries are:

- **Draw Style**
  A sub menu offers different styles of appearance of the materials in the image viewer. the material you have clicked on, the present appearance is marked.

  The possible appearances are:

  - **Invisible** The material is not visible.
  - **Contour** The segments are enclodes by lines.
  - **Hatched** The segments are enclosed by lines and shown hatched.
  - **Dotted** The segments are enclosed by lines and filled with dots.
  - **Light Dots** The segments are enclosed by lines and filled with less dots.

- **Locate**
  This command sets all viewers to the slice with the largest region of this material. This is especially useful to navigate in large data sets with many materials.
- **Delete**
  If you choose *Delete*, the region you have clicked on will be deleted, assigning all of its voxels to the first material in the list (typically the background).
- **Rename**
  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.
- **Edit color**
  This command brings up a color editor in which you can easily change the color of this material.
- **Lock/Unlock Material**
  You can lock a material with this option. A key will be displayed on the right hand side of each locked material. If a material is locked, no voxels can be subtracted from this region, neither explicitly, nor implictly by adding them to another material.
- **New Material**
  By selecting *New Material* 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.

The Viewer Window
In the center of the ”Image Segmentation Editor” window you see the presently chosen slice with colored contours for the segments at startup. Near the top border you find a slider to control which slice of the 3D volume is currently displayed.
The orientation (e.g. XY, XZ, or YZ) of the slice can be changed with the menu entry **View/Orientation** in the menu bar.

As from version 2.1 of Amira, there are two different viewer arrangements: Single viewer, and **4 viewer** display. You can choose between those with the **View/Layout** menu. In the **4 viewer** mode, three 2D viewers with the major orientations and an additional 3D viewer are displayed. The orientation menu is disabled in this mode. You can arbitrarily switch between the viewers during work. One of the viewers is the active viewer. It is depicted with a red frame. A viewer can be made the active viewer by clicking on its title-bar, or into the viewer itself (be careful, when clicking into the image in 3D mode, if the Pick tool or the Magic Wand tool is active.).
All 2D operations like the ”+” or ”-” are allways performed in the active viewer (i.e. in the slice and orientation currently shown in the active viewer).
See also the description of the **Edit/Options** menu and the **Zoom** buttons in the toolbox.

Using the Zoom buttons These buttons control the size of the image. An info line on the right hand side of the buttons 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 pixel 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.

Using the tools of the *Selection* command area

Beside the *Selection* buttons there is a *3D* toggle button which extends an action to all slices if it is activated. Otherwise the present slice in the active viewer is affected.

- **Picker**
  This tool marks a connected area within one material.
  If the *3D* toggle button on the right is activated, the connectivity check is also made in the third dimension, marking affected voxels in all slices.
- **Clear (C)**
  This button lets you clear the current markings.
  If the *3D* toggle button is activated markings in all slices are cleared.
- **Add (+)**
  This button adds all selected voxels to the *active material*. Voxels that are currently assigned to a locked material are not affected. If the *Shift* key is held down while clicking, the voxels remain selected.
- **Subtract (-)**
  This button lets you subtract all selected voxels from the *active material*, if this one is not *locked*.

***Tools* for marking an image**

The various marking *Tools* offer you the opportunity to define areas in the present slice for subsequent edit operations. A tool is activated by clicking on its icon with the left mouse button. Selected areas are colored red.

The operation of most of the tools can be modified by pressing the Ctrl- or the Shift key. Ctrl typically deselects instead of selecting voxels, while Shift adds voxels to the selection. The description of the individual tools gives details.

- **Brush** 
  If this tool is activated, you can mark an area by simply ”brushing” voxels holding down the left or right mouse button. A textfield in the *option panel* controls the size of your brush in screen pixels.

  If the Ctrl-Key is held down, or if the middle mouse button is used, pixels are deselected instead of beeing selected.

  The right mouse button is a flood fill tool. You can e.g. sourround 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. deselects connected parts of the selection.
- **Lasso** 
  The *lasso* lets you define an area by generating a closed contour curve. Usually, you draw such a curve freehand with the the left mouse button pressed, but it is also possible to fence an area with line segments by pressing **Alt**, and clicking succesively to points with the left mouse button (holding down the Alt key all the time). Succesive 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 *autoTrace* is enabled (this is the default) the line segments are fitted to image edges. Click with the left mouse button to define a starting point. Dragging the mouse after releasing the button a contour curve automatically fitted to the closest image edge is drawn. Further mouse clicks mark fixed points on the contour. Contour parts drawn before the latest mouse click remain in place. Close the contour by clicking with the middle mouse button.
- **Magic Wand** 
  By clicking with the left mouse button on a voxel you get the largest connected area that contains this voxel and all voxels whose data values lie within a range defined by the one of the selected voxel and the *tolerance* value set in the *option panel*.

  Normally, the *magic wand* works on the present slice only, but if the *3D* toggle button on the right of the *Selection* tools is on, the adjacency of voxels is tested in the 3D image data volume.

  The *same material only* toggle restricts the search to voxels which currently belong to the same material as the start voxel. If e.g. a voxel currently belonging to Exterior is used as seed point, voxels which are allready assigned to different materials will not be selected.

- **Blowtool** 
  By clicking with the left mouse button on a voxel and dragging the mouse without releasing the button an initially circle-formed contour blows up. The greater the distance to the initial position of the mouse click gets the more the contour grows. The contour is designed to grow in areas with homogenious grey values and to stop where grey values change abruptely, i.e. at image edges.

  The tolerance slider in the *option panel* controls how sharp the image edge has to be in order to stop the contour from growing at each contour point. The smaller the tolerance the sharper the image edge has to be. If the tolerance is large, the contour will even stop at weak edges. After releasing 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.

The menu bar The menu bar has four sub-menus:

- **Edit.** This menu provides an **undo** entry which undoes the last operation. Succesive invokation of **undo** is possible, 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.

  The **options** entry brings up an additional dialog, in which you can control the data window and the opacity of the selection.
- **View.** This menu lets you change the **Layout** of the image viewer area. You can choose between either one single viewer, or four viewers (three slice viewers, one 3D viewer). Details on these two arrangements are described above.

  In the single viewer layout, you can choose the **Orientation** of the viewer.

  The *Info line* toggle lets you switch off the on-line information displayed in the lower left part of the window.
- **Selection.** This menu provides a number of filters and operators, which modify the current selection. They are described below. Menu entries with dots (...) behind their name open an additional dialog.
- **LabelFilter.** These filters and operators directly modify the label field, and not the selection.

Selection Filters

- **Grow/Shrink**
  This filter does a morphological dilatation or erosion of the current selection. The filter can be applied on the current **slice** in the active viewer, or on the whole **volume**.
- **Invert**
  This operator inverts the selection, i.e. selected voxels are deselected, while unselected voxels become seleceted. The filter can be applied on the current **slice** in the active viewer, or on the whole **volume**.
- **Threshold...** Select voxels by threshold segmentation. In the additional dialog, a minimal and a maximal image intensity can be specified. By clicking on *Apply*, 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.
- **Make Ellipse**

  By pressing this button the outline of a selected area is replaced by an ellipse that fits it. The filter operates in the current **slice** in the active viewer.
- **Fill**

  This selects all unselected pixels, which are comletely sourrounded by selected pixels in the current **slice** in the active viewer.
- **Interpol** Pressing this button interpolates selections between all slices where areas have been selected. Pressing this button again after corrections in certain slices will interpolate the selections between all

slices where changes have been made.

Label Filters

- **Fill holes**
  This filter removes islands of arbitrary size in the *current* material, i.e. an "island" is assigned to the active region if it is totally surrounded by a segment that belongs to the active material.
  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 the regions, depending on the *size* and *percent* values that are set in the filters dialog.
  An "island" is a connected area of voxels, in which the number of voxels is less than or equal to the *size* specified in the dialog.
  If such an "island" is encountered, the percentages of the surrounding regions with respect to the total number of voxels adjacent to the "island" are calculated and compared with the *percent* threshold. The "island" is added to the region with the highest percentage value greater than or equal to *percent*. If no region exceeds the *percent* threshold, this "island" remains untouched. **Note** that the *percent* value must be between 0.0 and 1.0. With the *Mode:* buttons you can control, whether the filter works in the current *slice*, in *all slices* or in 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 *Apply*.
- **Smooth...**
  This is a modified (Gauss) filter, which smoothes the region boundaries, therefore the labeling is slightly changed. 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 to make region boundaries appear smooth, e.g. in the *GMC* module, otherwise such may be displayed as zigzag lines.
  The *Size* option is for entering the size of the filter mask which is a square with dimensions *Size* x *Size*. 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.
- **Snakes...**
  The Snakes filter automatically adjusts the contour of a selected region to "edges" in the image. Edges are regions with a sharp contrast.

Key Bindings

If the mouse cursor is placed somewhere in "theGI" window some panel commands are available by keys, the key bindings are the following:

- **Changing the *slice number***
  By holding **Shift**, while you press one of these keys, the current markings are copied to the target slice.
  Space/CursorDown/CursorRight - Go to next slice
  Backspace/CursorUp/CursorLeft - Go to previous slice
  PgDn - Go forth over the next five slices
  PgUp - Go back over the previous five slices
  Home - Go to the first slice
  End - Go to the last slice
- **Selection**
  c - Clear all markings

i - Invert all markings

a - Add current marking to *active region*

s - Subtract current marking from *active region*

- **Tools**

  1 - Lasso

  2 - Brush

  3 - Magic Wand

  4 - Pick Tool

- **Zoom**

  z - Decrease zoom

  Z - Increase zoom

- **Display style**

  d - Change display style of the regions, i.e. change the appearance of **every** region to one of the above described, with every new key press cycling through the possible appearances

9.6 ImageCrop Editor

This editor works on a scalar data field defined on a uniform cartesian grid. It lets you crop such a field, i.e., discard all voxels lying outside of a specified subvolume. Alternatively, you may rescale the field by changing the values of the bounding box. Note that rescaling does not mean *resampling*, i.e., the number of voxels will not be changed.

The total data volume can be seen as consisting of *slices* in every dimension. Each slice in a dimension is referenced by an *index* ranging from zero to the number of slices minus one. This editor gives you the opportunity to add new slices on every side in every dimension. Last but not least you can simply flip these slices, i.e. revert the order in every dimension.

When clicking on the crop button (see the icon above, shown in the *Working Area*) of uniform scalar data field, the *ImageCrop Editor* pops up. It looks as shown in Figure 9.4.

In the *Viewer* you should see a grey box with green corners marking the bounding box of the scalar data field. The green corners are *draggers* by which you can control the data volume you want to preserve.

This means that there are two ways to crop such a scalar data field:

- Dragging and moving the box
- Setting indices explicitly in the cropping text fields

Adding slices is simply done by setting indices in the cropping text fields beyond the limits of the total data volume.

The only way to manipulate the bounding box is to set new values in the corresponding text fields.

For flipping slices, click on one of the three flip buttons, labeled *FlipX*, *FlipY*, and *FlipZ*, which on activation revert the order of slices in the desired dimension.

All changes take effect if you press the *OK* or *Apply* buttons, they are discarded by activating the *Cancel* button. As usual the editor is closed by pressing *OK* or *Cancel*, *Apply* will leave it open, *Cancel* restores the previous state.

NOTE:
All calculations in this dialog were designed with the assumption that at least two slices exist in all directions. A warning dialog appears if you want to run this editor on a smaller number in any direction. The consequences are not predictable.

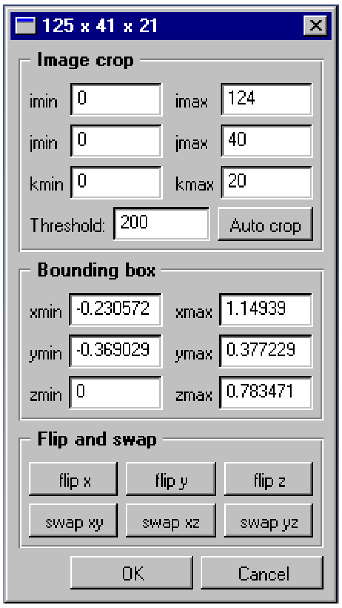


Figure 9.4: User interface of Amira's *ImageCrop Editor*.

Cropping an image by dragging and moving the box

A little experience with manipulating *draggers* and moving objects 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 have to be used.

Using the draggers you control the size of the subset of the data volume which has to be preserved, but the location of it in the total data volume is set by moving the box around to the desired location.

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 have to 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.

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 whether you really want to add new slices, and the slices are added on your confirmation to the volume at the specific 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.

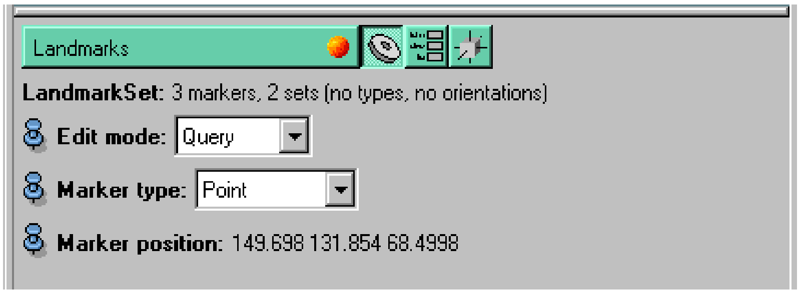


Figure 9.5: User interface of landmark editor.

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 boundig 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.

Flipping slices in one dimension

Invoke one of the three flip buttons to revert the order of slices in the corresponding dimension, e.g. click on *FlipX* to flip the slices in the x-dimension.

Exchanging two dimensions

Invoke one of the three exchange buttons to interchange the corresponding dimensions, e.g. click on *X-Y* to interchange the x- and y-dimension. CAUTION: The exchange operations are not rotations; they change the spatial orientation of the dataset.

9.7 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 4.1.6. In order to do so the viewer has to be switched to interaction mode, e.g. by hitting `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 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. 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 clicking on a marker its type and its xyz-position are shown. The marker type can be changed via an option menu. The default marker type is a *Point*, represented by a little sphere. In addition, a number of predefined medical marker types can be selected. In contrast to point-type markers, medical markers have fixed predefined sizes (world coordinates assumed to be given in centimeters).

See also documentation for *Landmark Set*.

9.8 LineSet Editor

This component allows you to edit *linesets*. It operates by directly interacting with the 3D geometry in the viewer window, cf. Section 4.1.6. In order to do so the viewer has to be switched to interaction mode, e.g. by hitting `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 pushing one of the buttons in the *Action* port. They are described below.

The *Selected* port informs you how many points and lines are selected at the moment.

In the *Display* port you may choose in which way the lineset is displayed. By clicking on the toggles optionally *all points*, *endpoints* and *lines* will be displayed in the viewer. Only displayed geometry is selectable. Default is endpoints and lines.

The *Select* port provides the following selection modes:

- **All** Clicking this button all lines will be selected.
- **By Length** Clicking this button an additional dialog pops up where you can enter the minimal and maximal numbers of points in lines that are to be selected. Pushing *OK* performs the selection.
- **Clear** Pushing this button the current selection will be cleared.

The *Action* port provides the following actions:

- **Connect** Pushing this button two lines can be connected. In order to do this exactly two points have to be selected, otherwise this button will not be active.
- **Delete** Selected points and lines will be deleted from the lineset.
- **Stretch** Pushing this button all selected lines will be smoothed.
- **Split** Pushing this button all lines will be splitted at selected points.

9.9 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 some extra information related to the data. Some parameters have a special meaning within Amira, cf. Section 4.2.7 of the user's guide.

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.

9.10 Simplification Editor

This editor can be used to reduce the number of triangles of a surface. In particular, the output of the *GMC* module has to be processed in this way before a tetrahedral patient model can be be generated. Surface simplification is done by means of an edge collapsing algorithm. Edges of the original surface

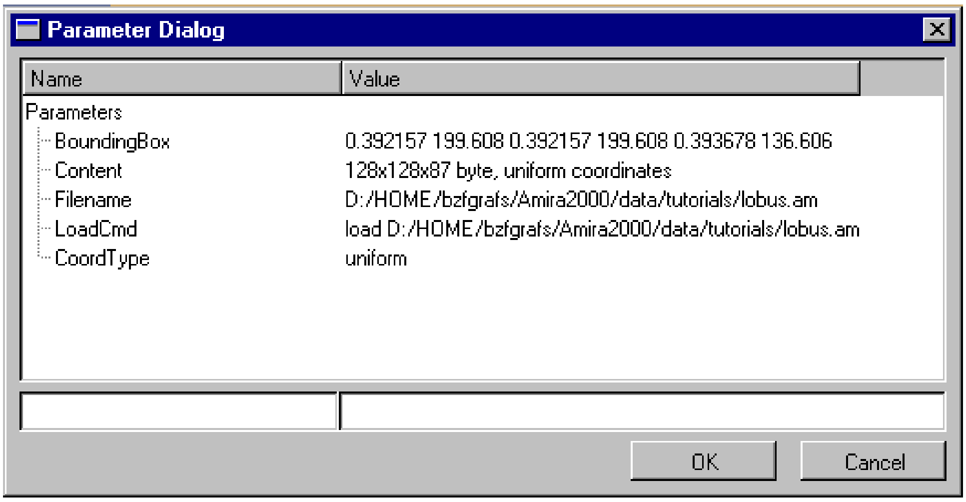


Figure 9.6: User interface of Amira's parameter editor.

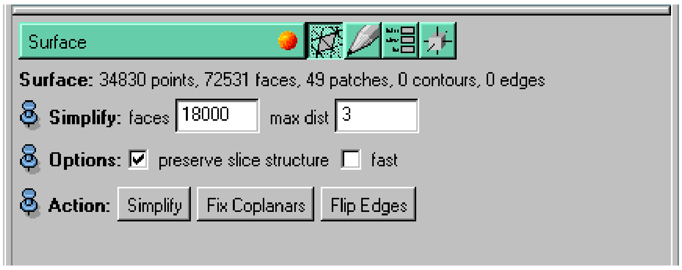


Figure 9.7: User interface of surface simplification editor.

are sucessively reduced to points. The shape of the original surface is preserved by minimizing a certain error criterium. 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*.

Ports

Simplify This port provides two 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 coplanar 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. Collapsing edges with a length exceeding this value will occur less likely. 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.

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.

Action Button *Simplify* starts surface simplification. The simplification process will take several minutes. You may interrupt it by hitting the stop button of the *progress bar*.

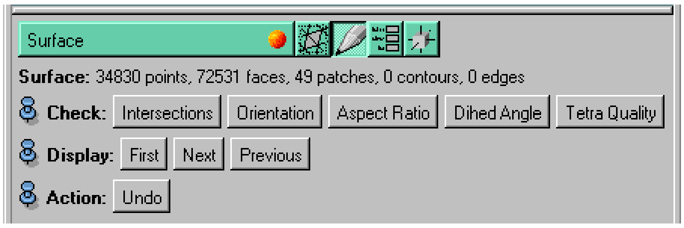


Figure 9.8: User interface of surface editor.

Button *Fix Coplanars* removes all coplanar triangles from a surface. This step is performed automatically after surface simplification.

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, coplanar triangles are removed automatically after this operation.

Commands

Simplifier setRadiusRatio <value>
This command defines a quality threshold for the edge flipping algorithm. Edges are flipped only if the radius ratio of a triangle exceeds the given value. By default a radius ratio of 20 is used.

Simplifier smooth <nSteps> <lambda>
This commands shifts the vertices of the surface so that it gets smoother. The value for *lambda* should be in the range of 0...1. This option is experimental and should not be used for routine work.

Simplifier setIntersectionTestStrategy <mode>
This command lets you choose between different intersection test strategies. *mode* is a three-digit number, for example 100. The digits specifiy 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.

9.11 Surface Editor

The *surface editor* allows you to check an arbitrary surface for intersecting triangles. It also provides some tools for interactively editing the surface. In this way, intersections may be removed manually. Also, the quality of the surface can be improved.

The editor works in conjunction with the *SurfaceView* module. If such a module is not already active, an instance of it will be automatically created when the editor is invoked. Then, in the viewer *interaction mode*, the following editing operations can be performed by clicking mouse buttons and using the Shift- or Ctrl-keys:

Button1
All triangles surrounding the picked triangle become visible.

Shift-Button1
The picked triangle is deselected and disappears.

Ctrl-Button1
The picked triangle as well as all its neighbors are deselected and disappear.

Shift-Ctrl-Button1
The picked triangle and all its neighbors become visible. All other triangles are deselected.

Button2
If a vertex is picked, a dragger is attached to that vertex. The dragger allows you to change the position of the vertex. If an edge is picked, the start and end points of the edge are displayed.

Shift-Button2
This operation 'flips an edge of the surface', i.e. a common edge of a pair of triangles is removed and the alternative triangle defining edge with respect to the corresponding quadrupel of surface vertices is introduced as a surface edge, thus the triangular layout is changed locally. 'Flipping the edge' a second time restores the original state.

Ctrl-Button2
Collapses an edge of the surface. Warning: There is no undo for this operation!

Ports

Check This port allows you to control the part of the surface being displayed.

Button *Intersections* checks the surface for intersecting triangles. The number of intersections is printed in Amira's console window. If intersections have been found, they can be subsequently viewed using port *Display*. Notice that a single intersection often will be reported multiple times because multiple triangles are involved.

Button *Orientation* checks the surface for correct orientation. 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 Amira's console window. If triangles have been removed automatically, you must subsequently select *Fix Coplanars* from the *Surface Simplication Editor* to remove them permanently. 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 are 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 report falsely oriented triangles.

Buttons *Aspect Ratio*, *Dihed Angle*, and *Tetra Quality* sort all surface triangles according to different quality measures. The aspect ratio is the ratio of the radii of the circumscribed and the inscribed circle 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 degree (better 10). Button *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 buttons of port *Display* you can cycle through all other triangles and edit the surface if necessary.

Display This port provides three buttons *First*, *Next*, and *Previous*. Depending on the last operation, these buttons allow you to cycle through the list of all triangles sorted by quality or through the list of intersecting triangles. These lists are not updated if you perform manual surface editing. Instead, you have to press one of the buttons of port *Check* again.

Undo Using this port you can undo the last editing operations (edge flip, edge collapse, and vertex movement) in reversed order. Currently, the undo buffer is constrained to a maximum of 100 operations.

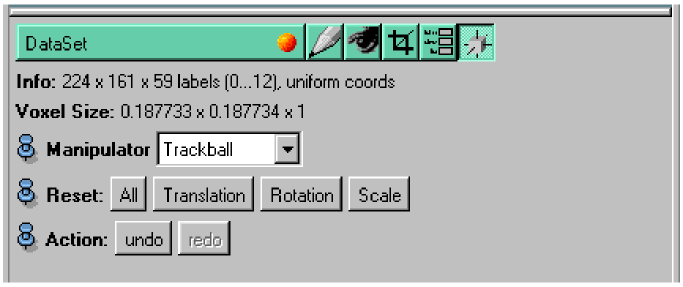


Figure 9.9: User interface of transform editor.

9.12 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 standard Inventor draggers. Make sure to switch the viewer to the interaction mode. Details can be found in the Inventor man pages `man SoDragger(3), SoTabBoxDragger(3), ....`

Display modules that are connected to a transformed data set will consider the transformation. Also some, but not all, computational modules that take more than one input support transformations.

Most data file formats do not support transformations. Therefore transformations are currently not saved along with the data. However, using the commands `getTransform, setTransform`, the transformation can be retreived as a homogeneous matrix. For details see `man SoTransform(3)`.

Index