UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MEDIVIS, INC.
Petitioner

v.

NOVARAD CORP.
Patent Owner

Inter Partes Review No. IPR2023-00042
US Patent No. 11,004,271

Inter Partes Review No. IPR2023-00045
US Patent No. 10,945,807

DECLARATION OF TODD W. PRICE

control_number	begattach	file_path	capture_url
PV_C1171489F	PV_C1171489F	PVOD Delivery #7433\1171489F-amira-users-guide-pdf.pdf	https://web.archive.org/web/20121004141444if_/http://www.amira.com/images/stories/pdf/522/AmiraUsersGuide.pdf
PV_C1171774F	PV_C1171774F	PVOD Delivery #7433\1171774F-3-d-visualization-dicom-sonia-pujol-pdf.pdf	https://www.slicer.org/w/img_auth.php/3/3a/3DVisualizationDICOM_SoniaPujol.pdf

original_capture_url	filename	document_title	capture_time
https://web.archive.org/web/20121004141444if_/http://www.amira.com/images/stories/pdf/522/AmiraUsersGuide.pdf	1171489F-amira-users-guide-pdf.pdf		2023-05-16T21:13:22.000Z
https://www.slicer.org/w/img_auth.php/3/3a/3DVisualizationDICOM_SoniaPujol.pdf	1171774F-3-d-visualization-dicom-sonia-pujol-pdf.pdf		2023-05-17T17:36:19.000Z

hash	file_size	unique_id	file_download_type	screenshot_server_ip	screenshot_load_time	screenshot_page_length	file_uploaded_at	file_description	post_datetime	parent_est_post_datetime	post_author	post_reaction_count	post_comment_count
96ef775455b1391aab0012fcd008237df1dc9ebf455b643e4d8fb99ba960d45d	8037594	2tyHxejmUsmPcFuW8MChpb	application/pdf										
0b439f4d5c3d8bcce022259c0041fcfc9f7fe0605ecdf33b9271e49e6a1b4148	53938773	wcjgKWTtZMcQqMBrMZSRZ7	application/pdf										

Affidavit of Todd W. Price

I, the undersigned, do hereby certify and affirm that:

1. I am over the age of 18 and am not a party to this action. All forthcoming facts are based on my personal knowledge, which I would be willing to testify competently to if called as a witness at trial.

2. I am the Chief Technology Officer at Page Vault Inc., headquartered at 401 E. Ontario St., Chicago, Illinois, 60611. I have been so employed since March 2013.

3. Page Vault is a cloud-based service that enables Users to make captures of Web Content, including webpages, online media files such as videos, and online documents.

4. When capturing Web Content, a User provides URLs (Uniform Resource Locators) associated with the Web Content to be captured by the Page Vault Capturing Server (PVCS). This is accomplished either through the Page Vault Browser (PVB) interface or through the Page Vault Portal (PVP) interface.

5. When using the PVB interface, a User establishes a connection from their computer to the PVCS. This connection operates over an encrypted channel and allows the User to remotely control the Page Vault Browser (PVB) software that is installed on the PVCS.

6. The PVB is a proprietary browser that utilizes a Chromium browser engine. The PVB restricts the User from modifying the Web Content, accessing or modifying a webpage's source code, installing browser plugins, or otherwise altering the contents of the Web Content or the internet traffic between Web Servers (WS) and the PVCS.

7. As a User controls the PVB, the PVCS establishes an active web browsing session with various WS.

8. Remotely controlling the PVB, a User can direct the PVCS to create Captures of Web Content associated with a URL.

9. When using the PVP (Page Vault Portal) interface, a User establishes an authenticated session to the PVCS and provides a URL that is associated with Web Content to be collected.

10. Through the PVP interface, the User directs the PVCS to initiate a secure connection to WS and create Captures of the Web Content associated with the provided URL.

11. Captures can include the way the Web Content for the specified URL appeared in the PVB interface along with associated metadata. Captures are digitally signed and hashed.

12. Page Vault securely stores Captures, which Users can access, review, and download.

13. Attached hereto are true and accurate copies of Captures of the PVCS.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed this 19th *day of* May, 2023 *at* Evanston, Illinois.

Todd W. Price

Amira 5

User's Guide

Amira® 5

Amira User's Guide

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

Copyright Information
©1995-2009 Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany
©1999-2009 Visage Imaging

All rights reserved.

Trademark Information:
Amira is being jointly developed by Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Visage Imaging.
Amira ® is a registered trademark of Konrad-Zuse-Zentrum für Informationstechnik Berlin and Visage Imaging.
HardCopy, MeshViz, VolumeViz, TerrainViz, ScaleViz are trademarks of Mercury Computer Systems S.A.S.
Mercury Computer Systems S.A.S. is a source licensee of OpenGL®, Open Inventor® from Silicon Graphics, Inc.
OpenGL® and Open Inventor® are registered trademarks of Silicon Graphics, Inc.
All other products and company names are trademarks or registered trademarks of their respective companies.
This manual has been prepared for Visage Imaging licensees solely for use in connection with software supplied by Visage Imaging and is furnished under a written license agreement. This material may not be used, reproduced or disclosed, in whole or in part, except as permitted in the license agreement or by prior written authorization of Visage Imaging. Users are cautioned that Visage Imaging reserves the right to make changes without notice to the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic or listing errors.

Contents

I Amira User's Guide 1

1 Introduction 3
- 1.1 Overview . . . 4
- 1.2 Features overview . . . 4
  - 1.2.1 Data import . . . 5
  - 1.2.2 Viewing, navigation, interactivity . . . 6
  - 1.2.3 Visualization of 3D Image Data . . . 6
  - 1.2.4 Image processing . . . 7
  - 1.2.5 Model reconstruction . . . 8
  - 1.2.6 Visualization of 3D models and numerical data . . . 9
  - 1.2.7 General Data Processing and Data Analysis . . . 10
  - 1.2.8 Matlab integration . . . 12
  - 1.2.9 High Performance Visualization . . . 12
  - 1.2.10 Automation, Customization, Extensibility . . . 12
- 1.3 Application Areas . . . 12
- 1.4 Options . . . 13

2 First steps in Amira 17
- 2.1 Getting Started . . . 18
  - 2.1.1 Loading Data . . . 19
  - 2.1.2 Invoking Editors . . . 21
  - 2.1.3 Visualizing Data . . . 22
  - 2.1.4 Interaction with the Viewer . . . 23
- 2.2 How to load image data . . . 26
  - 2.2.1 The Amira File Browser . . . 26
  - 2.2.2 Reading 3D Image Data from Multiple 2D Slices . . . 27
  - 2.2.3 Setting the Bounding Box . . . 28
  - 2.2.4 The Stacked Slices file format . . . 29

2.2.5 Working with Large Disk Data 30
2.3 Visualizing 3D Images 32
2.3.1 Orthogonal Slices 33
2.3.2 Simple Data Analysis 33
2.3.3 Resampling the Data 34
2.3.4 Displaying an Isosurface 36
2.3.5 Cropping the Data 36
2.3.6 Volume Rendering 37
2.4 Segmentation of 3D Images 40
2.4.1 Interactive Image Segmentation 40
2.4.2 Volume Measurement 42
2.4.3 Threshold Segmentation 43
2.4.4 Refining Threshold Segmentation Results 43
2.5 Surface Reconstruction from 3D Images 45
2.5.1 Extracting Surfaces from Segmentation Results 45
2.5.2 Simplifying the Surface 45
2.6 Creating a Tetrahedral Grid from a Triangular Surface 47
2.6.1 Simplifying the Surface 47
2.6.2 Editing the Surface 49
2.6.3 Generation of a Tetrahedral Grid 51
2.7 Warping and Registration Using Landmarks 53
2.7.1 Displaying Data Sets in Two Viewers 54
2.7.2 Creating a Landmark Set 54
2.7.3 Registration via a Rigid Transformation 57
2.7.4 Warping Two Image Volumes 57
2.8 Registration of 3D image data sets 58
2.8.1 Basic Manual Registration 59
2.8.2 Automatic Registration 60
2.8.3 Image Fusion 61
2.9 Alignment of 2D Physical Cross-sections 62
2.9.1 Basic Manual Alignment 63
2.9.2 Alignment Via Landmarks 65
2.9.3 Optimizing the Quality Function 67
2.9.4 Resampling the Input Data 67
2.9.5 Using a Reference Image 68
2.10 Visualization of Vector Fields 69
2.10.1 Simple Vector Representation 69
2.10.2 Illuminated Stream Lines 71

Medivis Exhibit 1023

2.10.3 Animated Particle Plot 72
2.11 Creating animated demonstrations 73
2.11.1 Creating a Network 74
2.11.2 Animating an OrthoSlice Module 74
2.11.3 Activating a Module in the Viewer Window 77
2.11.4 Using a Camera Rotation 78
2.11.5 Editing or Removing an Already Defined Event 79
2.11.6 Overlaying the Bone with Skin 79
2.11.7 Using Clipping to Add the Skin Gradually 80
2.11.8 More Comments on Clipping 83
2.11.9 Breaks and Function Keys 84
2.11.10 Loops and Go-to 85
2.11.11 Storing and Replaying the Animation Sequence 86
2.12 Creating movie files 86
2.12.1 Attaching MovieMaker to a Camera Path 87
2.12.2 Attaching MovieMaker to DemoMaker 89
2.13 Using MATLAB Scripts 90
2.13.1 Lowpass Filtering on Images 90
2.13.2 Thresholding on a Volume 93
2.14 Introduction to the Filament Editor 93
2.14.1 Exploration of the volume data 94
2.14.2 Automatic extraction of the dendritic tree 94
2.14.3 Filament Tracing 97
2.14.4 Visualize the network 98
2.14.5 Alternative way to extract a network using the Segmentation Editor 99
2.15 Introduction to the Multi-planar Viewer 100
2.15.1 Exploration of the volume data 101
2.15.2 Explore two data sets using fusion mode 102
2.15.3 Manually register two data sets 104

3 Program Description 107
3.1 Interface Components 107
3.1.1 File Menu 107
3.1.2 Edit Menu 110
3.1.3 Pool Menu 112
3.1.4 Explorer Menu 114
3.1.5 Create Menu 114

Medivis Exhibit 1023

3.1.6 View Menu 115
3.1.7 Online Help 118
3.1.8 Main Window 121
3.1.9 Viewer Window 131
3.1.10 Console Window 136
3.1.11 File Dialog 137
3.1.12 Job Dialog 139
3.1.13 Preferences Dialog 141
3.1.14 Snapshot Dialog 146
3.1.15 System Information Dialog 147
3.2 General Concepts 148
3.2.1 Class Structure 148
3.2.2 Scalar Field and Vector Fields 150
3.2.3 Coordinates and Grids 151
3.2.4 Surface Data 152
3.2.5 Vertex Set 153
3.2.6 Transformations 153
3.2.7 Parameters 154
3.2.8 Shadowing 154
3.2.9 Sub-application Concept 155

4 Technical Information 157
4.1 Data Import 157
4.2 Command Line Options 158
4.3 Environment Variables 159
4.4 User-defined start-up script 161
4.5 System Requirements 162
4.5.1 Troubleshooting 163
4.5.2 Microsoft Windows 164
4.5.3 Linux 164
4.5.4 Mac OS 165
4.6 Amira License Manager 166
4.6.1 Contents 166
4.6.2 About Password Protection 166
4.6.3 About the Amira License Manager 166
4.6.4 Troubleshooting 167
4.6.5 Using TGS_LICENSE_DEBUG 168
4.6.6 Contacting the License Administrator 170

Medivis Exhibit 1023

4.6.7 Contacting Technical Support 170
4.7 Notes for Mac users 171
4.8 Amira and the /3GB switch 171
4.8.1 The Problem 172
4.8.2 How to activate the /3GB switch 172

5 Scripting 175
5.1 Introduction 175
5.2 Introduction to Tcl 176
5.2.1 Tcl Lists, Commands, Comments 176
5.2.2 Tcl Variables 177
5.2.3 Tcl Command Substitution 178
5.2.4 Tcl Control Structures 178
5.2.5 User-Defined Tcl Procedures 180
5.2.6 List and String Manipulation 181
5.3 Amira Script Interface 182
5.3.1 Predefined Variables 184
5.3.2 Object commands 185
5.3.3 Global Commands 185
5.4 Amira Script File 199
5.5 Configuring Popup Menus 200
5.6 Registering pick callbacks 203

6 Demo Framework 207
6.1 Directory Structure and Files 207
6.1.1 Directories and Files in src/ 208
6.1.2 Data Storage 214
6.2 Selecting Demos 215
6.3 Prerequisites 218

II Molecular Option User's Guide 219

7 Molecular Option Introduction 221
7.1 First Steps with Molecular Visualization in Amira 221
7.1.1 Getting Started with Molecular Visualization 222
7.1.2 Selection, Labeling, and Masking 225
7.1.3 Alignment of Molecules 232

Medivis Exhibit 1023

7.1.4 Molecular Surfaces 237
7.1.5 Sequential and Structural Alignment 240
7.1.6 Editing of molecules 242
7.1.7 Molecular Interfaces 245
7.1.8 Measurement 248
7.2 Molecular Data Structures 249
7.2.1 Internal Structure of Molecules 249
7.3 Displaying Molecules 249
7.3.1 Coloring Molecules 250
7.3.2 Selecting and Filtering atoms 252
7.4 Aligning Molecules 254
7.4.1 Alignment of Trajectories 254
7.4.2 Mean Distance Alignment 256
7.4.3 Sequence alignment 256
7.5 Visualizing Molecular Trajectories and Metastable Conformations 257
7.6 Atom Expressions 257
7.6.1 Overview 257
7.6.2 Grammar 258
7.6.3 Literals 258
7.6.4 Operators 259
7.6.5 Shortcuts 261
7.6.6 Further Examples 261

III Virtual Reality Option User's Guide 263

8 Virtual Reality Option User's Guide 265
8.1 Virtual Reality Option Essentials 266
8.1.1 Virtual Reality Option configurations 266
8.1.2 Immersive interaction and the trackd daemon 266
8.1.3 Multiple displays and parallel rendering 267
8.2 Using Virtual Reality Option on a multi-pipe system 267
8.2.1 Configuration 267
8.2.2 Starting Amira 268
8.3 Using Virtual Reality Option on a cluster 268
8.3.1 Overview 268
8.3.2 Requirements 269
8.3.3 Preparing cluster slave nodes 269

Medivis Exhibit 1023 16

8.3.4 Running Virtual Reality Option on cluster 272
8.3.5 Limitations 273
8.3.6 Troubleshooting cluster configurations 274
8.4 Flat Screen Configurations 274
8.4.1 Example: A 2-channel Passive Stereo Configuration . . . 275
8.4.2 Example: A Super-wide Configuration with Soft-edge Blending 279
8.4.3 Example: A Tiled 4-channel 2x2 Monitor Configuration . 282
8.5 Immersive Configurations 284
8.5.1 Example: A Workbench Configuration 285
8.5.2 Example: A Holobench Configuration 287
8.5.3 Example: A 4-side CAVE Configuration 290
8.6 Calibrating the Tracking System 293
8.6.1 Overview 293
8.6.2 Tracking system essentials 294
8.6.3 Calibration step by step 298
8.7 3D User Interaction 301
8.7.1 The 3D Menu 302
8.7.2 User-defined 3D Menu Items 303
8.7.3 3D Module Controls 307
8.7.4 Navigation Modes 307
8.7.5 Tcl Event Procedures 308
8.7.6 The 2D Mouse mode 309
8.8 Writing Virtual Reality Option Custom Modules 310
8.9 Config File Reference 313
8.9.1 Tracker Emulator 322

IV Very Large Data Option User's Guide 325

9 Very Large Data Option User's Guide 327
9.1 Working with out-of-core data files (LDA) 327
9.1.1 Tune the Size Threshold to Allow Conversion 327
9.1.2 Load the Out-of-core Data 328
9.1.3 Raw Data Parameters 329
9.1.4 Out-of-core Conversion 329
9.1.5 Display an Orthoslice, an Oblique Slice, and a 3D Volume 332

Medivis Exhibit 1023

V Quantification+ Option User's Guide 335

VI Microscopy Option User's Guide 339

10 Deconvolution 341
10.1 General remarks about image deconvolution 342
10.2 Data acquisition and sampling rates 343
10.3 Standard Deconvolution Tutorial 345
10.4 Blind Deconvolution Tutorial 350
10.5 Bead Extraction Tutorial 354
10.6 Performance issues and multi-processing 359

11 Working with Multi-Channel Images 365
11.1 Loading Multi-Channel Images into Amira 365
11.2 Using OrthoSlice with a MultiChannelField 367
11.3 Using ProjectionView with a MultiChannelField 368
11.4 Using Voltex with a MultiChannelField 368
11.5 Saving a MultiChannelField in a Single AmiraMesh File 370

VII Skeleton Option User's Guide 371

12 Skeleton Option User's Guide 373
12.1 Importing your Image Data 373
12.2 Arranging the Bricks 374
12.3 Aligning Bricks 375
12.4 Filtering, Correcting Z-Drop, Resampling 375
12.5 Creating the Large Disk Data 377
12.6 Accessing the Large Disk Data 378
12.7 Computing directly on the Large Disk Data 379
12.8 Region of Interest 381
12.9 Check Network 383
12.10 Coloring a Lineset According to its Depth Value 383

Medivis Exhibit 1023 18

Part I

Amira User's Guide

Medivis Exhibit 1023

1 Introduction

Amira is a 3D data visualization, analysis and modelling system. It allows you to visualize scientific data sets from various application areas, e.g. medicine, biology, bio-chemistry, microscopy, biomed, bioengineering. 3D data can be quickly explored, analyzed, compared, and quantified. 3D objects can be represented as image volumes or geometrical surfaces and 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.
Amira is a powerful, multifaceted software platform for visualizing, manipulating, and understanding Life Science and bio-medical data coming from all types of sources and modalities. Initially known and widely used as the 3D visualization tool of choice in microscopy and biomed research, Amira has become a more and more sophisticated product, delivering powerful visualization and analysis capabilities in all visualization and simulation fields in Life Science.

- Multi purpose - One platform for visualizing, analyzing and presenting
- Flexible - Option packages to configure Amira to your needs
- Efficient - Takes advantage of the latest graphics cards and processors
- Easy to use - Intuitive user interface and great documentation
- Cost effective - Multiple options and flexible license models
- Handling large data - Very large data sets are easily accessible with specific readers
- Extensible - C++ coding wizard for technical extension and customization
- Support - Direct customer support with high level of interaction
- Innovative - Technology always updated to the latest innovation

Section 1.1 (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 1.2 (Features) summarizes key features of Amira, for example direct volume rendering, image processing, and surface simplification.

Medivis Exhibit 1023

Section 1.3 (Application Areas) illustrates some typical application areas of Amira and shows what kinds of problems can be solved using the system.
Section 1.4 (Options) shortly describes optional extensions available for Amira and what they can be used for.

1.1 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 *Pool*. Icons are connected by lines indicating processing dependencies between the components, i.e., which modules are to be applied to which data objects. Alternatively, modules and data objects can be displayed in a *tree view Explorer*. Modules from data objects of specific types are created automatically from file input data when reading or as output 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 Open Inventor from Mercury. In addition, several modules apply direct OpenGL rendering to achieve special rendering effects or to maximize performance. In total, there are more than 270 data object and module types. They allow the system to be used for a broad range of applications. Scripting can be used for customization and automation. User-defined extensions are facilitated by the Amira developer version.

1.2 Features overview

Amira provides a large number of data and module types allowing you to visualize, analyze and model various kinds of 3D data. The Amira framework is ideal to

Medivis Exhibit 1023

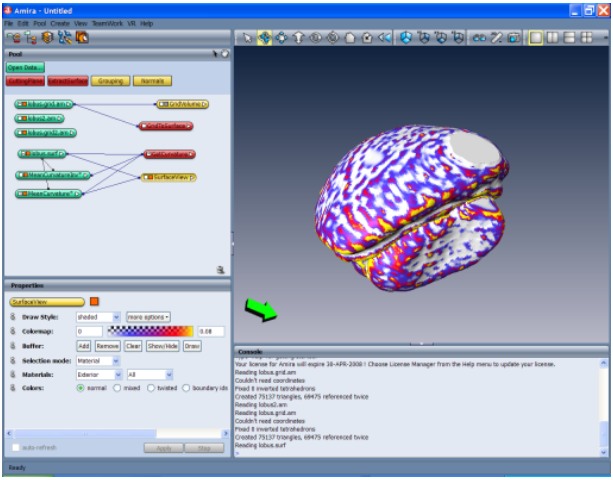


Figure 1.1: Data objects and modules are represented as little icons in the Pool (top left). In the 3D graphic window a surface colored according to its curvature is shown. Curvature information has been computed by a computational module and is stored as a separate data object. In the left window the parameters of selected modules are shown. The lower right pane provides a Tcl-command shell as well as access to the help browser.

integrate the data from multiple sources into a single environment.
This section summarizes the main features of Amira software suite. For more complete information you may browse indexes for data types, file formats and modules in the Reference Guide. This is accessible from the home page of the online help browser.
Section 1.4 describes the Amira optional extensions.

1.2.1 Data import

Amira can load directly different types of data, including:

- 2D and 3D image and volume data
- Geometric models such as point sets, line sets, surfaces, grids
- Numerical simulation data

- Time series and animations

A large number of file formats are supported in the standard edition or through specific optional readers. For an introduction to data import, see Chapter 2. For more details, see section 4.1.

1.2.2 Viewing, navigation, interactivity

All visualization techniques can be arbitrarily combined to produce a single scene. Moreover, multiple data sets can be visualized simultaneously, either in several viewer windows or in a common one. Thus you can display single or multiple data sets in a single or multiple viewer windows, and navigate freely around or through those objects.
A built-in spatial *transformation editor* makes it easy to register data sets with respect to each other or to deal with different coordinate systems. Automatic registration of volume or geometric data is also possible.
Direct interaction with the 3D scene allows you to quickly control regions of interest, slices, probes and more.
Combinations of data sets, representation and processing features can be defined with minimal user interaction for simple or complex tasks. See section 1.1.

1.2.3 Visualization of 3D Image Data

1.2.3.1 Slicing and Clipping

You can quickly explore 3D images looking at single or multiple orthographic or oblique sections. Multiple data sets can be superimposed on slices, or displayed as height fields. You can cut away parts of your data to uncover hidden regions. Curved or cylinder slices are also available.

1.2.3.2 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 in real time, even

Medivis Exhibit 1023

on very large data when using the Very Large Data Option. Thus volume rendering can instantly highlight relevant features of your data. 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 be either generated automatically or edited interactively using an intuitive colormap editor.

1.2.3.3 Isosurfaces

Isosurfaces are most commonly used for analyzing arbitrary scalar fields sampled on discrete grids. 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.

1.2.3.4 Large Volume Data

With the Very Large Data Option, even very large data sets that cannot be fully loaded in memory can be manipulated at interactive speed. Multi-resolution techniques can manage and visualize extremely large amounts of volume data of up to hundreds of gigabytes. You can then, for instance, quickly select a region of interest and extract down-sampled or partial data for further processing.

1.2.4 Image processing

1.2.4.1 Alignment of image slices

The image *slice aligner* enables you to build a consistent stack of images with manual or automatic tools, if, for instance, physical cross-sections have been shifted during image acquisition.

1.2.4.2 Image filters

Image features can be enhanced by applying a wide range of filters for controlling contrast, smoothing, noise reduction and much more. See *Image Filters* and Quantification+ Option.

1.2.4.3 Image segmentation

Segmentation means assigning labels to image voxels that identify and separate objects in a 3D image. Amira offers a large set of segmentation tools, ranging from purely manual to fully automatic: brush (painting), lasso (contouring), magic wand (region growing), thresholding, intelligent scissors, contour fitting (snakes), contour interpolation and extrapolation, wrapping, smoothing and de-noising filters, morphological filters for erosion, dilation, opening and closing operations, connected component analysis, images correlation, objects separation and filtering (see Quantification+ Option), etc. See section 2.4 for a tutorial about image segmentation.

1.2.5 Model reconstruction

1.2.5.1 Surface generation

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 that are automatically generated during segmentation allow the system to produce optionally 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.

1.2.5.2 Surface Simplification and Editing

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.

Medivis Exhibit 1023

A surface editor is also available for smoothing or refining surface in whole or part, cutting and copying parts of surfaces, defining boundary conditions for further numerical simulation, checking and modifying surface triangles.

1.2.5.3 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.

1.2.5.4 Point Clouds/Scattered Data

Amira can also reconstruct surfaces from scattered points (see *Delaunay* and *PointWrap* modules).

1.2.5.5 Skeletonization

A set of tools is included for reconstructing and analyzing a dendritic, porous or fracture network from 3D image data.

1.2.6 Visualization of 3D models and numerical data

1.2.6.1 Point sets, line sets

Amira can visualize arbitrary functional data given on 3D *point sets* or *line sets*.

1.2.6.2 Polygonal models

A number of drawing styles and coloring schemes help to yield meaningful and informative visualizations of polygonal models, whether generated from image data or imported from CAD or simulation package. Surface and 3D grid meshes can be colored or textured in order to visualize a second independent data set.
Another Amira feature 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.

1.2.6.3 Numerical data post-processing

Amira allows you to analyze results of numerical simulations. It supports polygonal surfaces such as triangular meshes, 3D lattices with uniform, rectilinear of curvilinear coordinates, and polyhedral 3D grids such as tetrahedral or hexahedral grids. Most general purpose image visualization techniques and analysis tools can be applied, e.g.: slice extraction, computation of isolines or isosurfaces, data probing and histograms. In addition, scalar quantities can be visualized with pseudocolors on the grid itself.
Beside visualization, data representations such as isosurfaces, grid cuts or contour lines can be extracted as first class data objects.
Displacement vectors can be visualized on grids or applied as grid deformation that can be animated.

1.2.6.4 Flow Visualization

Amira provides many advanced tools for vector fields and flow visualization. Vector arrows can be drawn on a slice, within a volume, or upon a surface. The flow structure may be better revealed by representations such as fast Line Integral Convolution on slices or arbitrary surfaces, illuminated and animated streamlines, stream ribbons, stream surfaces, particle animations, synthetic *vector probe...* All of these stream visualization techniques are highly interactive. While seedpoint distributions can be automatically calculated, you can also select and interactively manipulate seed points and structures, thus supporting the investigation the flow field and highlighting of different features. Amira can also support six-component complex vector fields and phase visualization, e.g. electromagnetic fields.
See tutorial section 2.10 (visualization of vector fields).

1.2.6.5 Tensor Data

Amira has support for iconic visualization of tensor field, extraction of eigenvalues, computation of rate of strain tensor, gradient tensor.

1.2.7 General Data Processing and Data Analysis

1.2.7.1 3D registration of multiple data sets

Multiple data sets can be combined to compare images of different objects, or images of an object recorded at different times or with different imaging modalities

Medivis Exhibit 1023

such as X-ray CT and MRI. In addition, fusion of multi-modal data by arbitrary arithmetic operations can be performed to increase the amount of information and accuracy in the models. Amira allows manual registration through interactive manipulators, automatic rigid or non-rigid registration through landmarks, and automatic registration using iterative optimization algorithms (see *AffineRegistration* module).
Surfaces can also be registered using rigid or non-rigid transformations, based on landmarks sets warping, alignment of centers or principal axes, or distance minimization algorithms.

1.2.7.2 Operating on 3D data

Many utilities are available for data processing. Here are some important ones. Resampling can reduce or enlarge the resolution of a 3D image or data sets defined on regular grids, and different sampling kernels are supported. Data can be cropped or regions of interest can be defined. Data can be converted to any supported primitive type, from byte to 64-bits floating point numbers. Multi-component data such as multi-channel images or vector data can be composed or decomposed. Standard 3D field operators such as scalar field gradient or vector field curl are available. Surface curvatures and distances between surfaces can also be computed, as scalar or vector information. The powerful *Arithmetic* module allows to perform calculations on data sets with user-defined expression, and can be used to interpolate data between regular grids and polyhedral grids. Data sets can also be created from arithmetic expressions.

1.2.7.3 Measurements, quantification

You can query the exact values of your data sets at arbitrary locations specified interactively by a mouse click, or along user-defined lines and spline curves. Probe points can serve to set interactively isosurfaces. You can plot or export the data for further processing with spreadsheet or plotting applications, with probing, measuring, counting, and other statistical modules quantify densities, distances, areas, volumes, mean value and standard deviation, ...
Histograms of values can be computed and plotted, possibly restricted to a region of interest. *Volume/value* diagrams can be also plotted.
The Quantification+ Option provides an extensive set of intensity and geometrical measurements on image or label data, either for individual labelled particles or as statistics. See the section dedicated to Quantification+ Option.

Medivis Exhibit 1023

1.2.8 Matlab integration

You can integrate complex calculus using Matlab software from The Mathworks, Inc. by means of the *CalculusMatlab* module. This module allows for a connection to your Matlab server from your Amira session, and executes Matlab computations directly on your Amira data. It is also possible to import and export Matlab matrices to and from Amira, and export Amira surfaces to Matlab surfaces. See section 2.13.

1.2.9 High Performance Visualization

Amira makes extensive use of graphics hardware for optimal performance and rendering quality on your system. Moreover, the Virtual Reality Option allows for combining multiple graphics engines for high-performance requirements.

1.2.10 Automation, Customization, Extensibility

Tcl scripting
All Amira components can be controlled via a Tcl command language interface. Tcl scripts are used for saving your work session. Tcl scripts also allow the advanced user to automate or customize tasks with Amira for routine workflows, without the need for C++ programming. Custom Amira modules with user interface can even be created as Tcl scripts. Amira module behaviour and 3D interaction can be customized by using Tcl. Amira can also be used for batch processing.
See Chapter 5, including a short introduction to the Tcl scripting language.
C++ programming
With the Virtual Reality Option, Amira can also be extended by programmers. The Developer Option allows for the creation of new custom components for Amira such as file readers and writers, computation modules, and even new visualization modules, using the C++ programming language. New modules and new data classes can be defined as subclasses of existing ones. In order to simplify the creation of new custom extensions, a development wizard is included.
See the Developer Option User's Guide for detailed information.

1.3 Application Areas

Amira is successfully being used in a number of different application areas. Among these are:

Medivis Exhibit 1023

- Medicine
- Biology
- Microscopy
- Computational Fluid Dynamics
- Neuroscience

Examples from these disciplines are illustrated by several *demo scripts* contained in the online version of the user's guide.

1.4 Options

Amira Options are additional sets of modules providing solutions for dedicated application areas. Options can be added to a standard Amira installation at any time. For each option a separate license is required. Currently, the following options are available for Amira:

- **Amira Developer Option** allows you to develop your own custom modules, file readers, and file writers using the C++ programming language.

- **Amira Molecular Option** adds advanced tools for the visualization of molecules. It combines Amira's strong capabilities for 3D data visualization such as hardware-accelerated volume rendering, with specific tools for molecular visualization and data analysis, such as molecular surfaces, sequence alignment, configuration density computation, and molecule trajectories. Amira Molecular Option includes a very powerful molecule editor.

- **Amira Mesh Option** supports mesh generation for flow, stress, and thermal analysis; for export of surface or 3D meshes to solvers; and for post-processing of data coming back from these solvers, providing very powerful visualization on scalar, vector, and tensor fields.

- **Amira DICOM reader** allows for import/export of DICOM data in Amira, making Amira the perfect application for medical data analysis.

- **Amira Microscopy Option** includes readers for most important microscopy file formats, 3D image deconvolution and and a dedicated filament editor.

Medivis Exhibit 1023

- **Amira Skeleton Option** supports reconstruction and analysis of porous, vascular and dendritic networks.

- **Amira Virtual Reality Option** is designed to enable the use of Amira's advanced data visualization and analysis features on immersive VR systems and tiled screen configurations. It has built-in support for efficient multi-threaded rendering on multipipe systems and for distributed rendering on cluster systems using application-level distribution. This approach leads to optimal performance with minimal bandwidth requirements. Tracking capabilities allow for a true immersive experience as well as interaction with the visualization.

- **Amira Very Large Data Option** manages and visualizes very large amounts of volume data, up to hundreds of gigabytes. The multi-resolution technique used in this option allows for interactive visualization and navigation through vast amounts of data.

- **Amira Quantification+ Option** includes new image processing capability as well as image analysis and quantification tools. Very powerful for material sciences, this option allows for efficient statistical analysis of data in this application area.

- **Specialized Readers** For the following file formats / applications specialized readers are available: **SEG-Y, CATIA 4, CATIA 5, IGES, STEP, TNO Madymo, and Mecalog Radioss**

Table 1.1 shows the license keyword associated with each of the Amira Options. For additional information about Amira and its options, please refer to the Amira web site, *http://www.amira.com/*.

Medivis Exhibit 1023

Amira Option	**License keyword**
Developer Option	AmiraDev
Molecular Option	AmiraMol
Mesh Option	AmiraMesh
DICOM reader	AmiraDicomReader
Microscopy Option	ResolveRT
Skeleton Option	AmiraSkel
Virtual Reality Option	AmiraVR
Very Large Data Option	AmiraVLD
Quantification+ Option	AmiraQuant
Specialized readers:	
SEGY reader	AmiraSegyReader
Catia5 reader	AmiraCatia5Reader
Catia4 reader	AmiraCatia4Reader
IGES reader	AmiraIgesReader
STEP reader	AmiraStepReader
Madymo reader (TNO)	AmiraMadymoReader
Mecalog Radioss reader	AmiraRadiossReader

Table 1.1: Amira Options and their associated license keywords

Medivis Exhibit 1023

2 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 particular, these topics will be covered:

- *Getting started* - the basics of Amira
- *Reading images* - how to read images
- *Visualizing 3D images* - slices, isosurfaces, volume rendering
- *Image segmentation* - segmentation of 3D image data
- *Surface reconstruction* - surface reconstruction from 3D images
- *Grid generation* - creating a tetrahedral grid from a triangular surface
- *Warping* - how to work with landmark sets
- *3D image registration* - how to register 3D image data sets
- *Alignment of 2D Physical Cross-Sections* - how to reconstruct a 3D model
- *Vector fields* - stream lines and other techniques
- *Filament Editor* - filament tracing for neurons and vessels images
- *The DemoMaker module* - creating animations with the DemoMaker module
- *Creating movie files* - how to use the MovieMaker module
- *Using MATLAB* - how to use the CalculusMatlab module
- *Trace Filaments* - how to extract a network model from a neuron 3D image data
- *Multi-planar Viewer* - visualize and register multiple volumes

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.

Note: If you want to visualize your own data, please first refer to Section 4.1. This section contains some general hints on how to import data sets into Amira.
Note for Mac users: Please read Section 4.7 before starting the tutorials.

2.1 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 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.

- Windows, select the Amira icon from the start menu.
- Mac, select the Amira icon from the Application menu.
- Unix-Linux, 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, a window like the one shown in Figure 2.1 appears on the screen. The user interface is divided into four major regions. The 3D viewer window displays visualization results, e.g., slices or isosurfaces. The Pool will contain small icons representing data objects and modules. The Properties Area displays interface elements (*ports*) associated with Amira objects. Finally, the lower left pane is shared by the console and Amira's integrated hypertext help browser. Click on the Console or Help tab to select which window you want to view. The console prints system messages and lets you enter Amira commands. You can use the help browser to read the user's guide online.
You can also activate the help browser by pressing F1, selecting *User's Guide* from the *Help* menu of Amira's main window, or by typing `help` in the console window.

Medivis Exhibit 1023

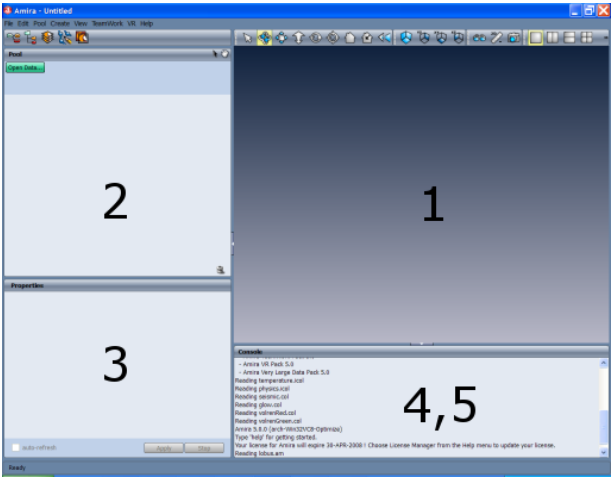


Figure 2.1: The Amira user interface consists of five major parts: the 3D viewer (1), the Pool (2), the Properties Area (3), the console window (4), and the help browser (5).

2.1.1 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 *Open Data ...* from the *File* menu.

After selecting this menu item, the file dialog appears (see Figure 2.2). By default the dialog displays the contents of 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.
At the top of the Pool is an Open Data button which is a shortcut to the File/Open Data dialog. You may use it for opening data files in the tutorials that follow. However, the tutorials will instruct you to use the File/Open Data command.
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

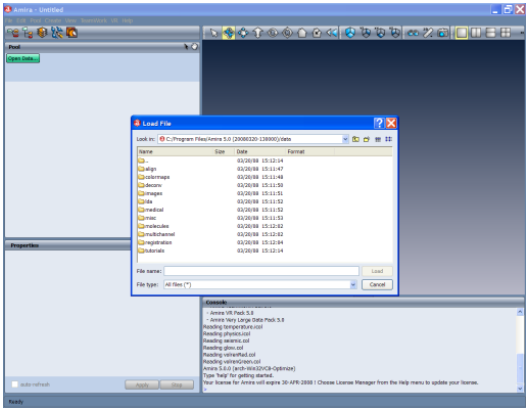


Figure 2.2: Data sets can be loaded into Amira using the file browser. In most cases, the file format can be determined automatically. This is done by either analyzing the file header or the file name suffix.

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*.

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 green icon in the Pool. In the following we call this data set "lobus data set".

- 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 Properties Area (Figure 2.3). 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

Medivis Exhibit 1023

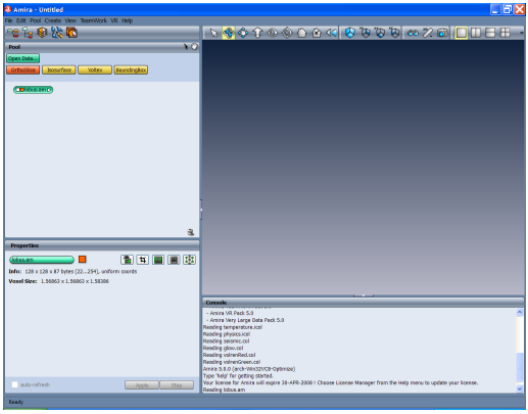


Figure 2.3: Data objects are represented by green icons in the Pool. Once an icon has been selected information about the data set such as its size or its coordinate type is displayed in the Properties Area.

the icon, click on an empty area in the Pool window. You may also pick the icon with the left mouse button and drag it around in the Pool.
Clicking on an object typically causes additional buttons to be displayed in the button area at the top of the Pool. These buttons are convenience buttons allowing easy one-click access to the modules most frequently used by the selected object. The tutorials, however, will have you to access modules via the menu interface to help familiarize you with the organization of modules within Amira.

2.1.2 Invoking Editors

After selecting an object, in addition to the textual information, some buttons appear in the Properties Area, to the far right of the data object's name. These buttons represent *editors* which can be used to interactively 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:

Medivis Exhibit 1023

- 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.

2.1.3 Visualizing Data

Data objects like the lobus data can be visualized by attaching *display modules* to them. Each icon in the 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 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, 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. As 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.

Medivis Exhibit 1023

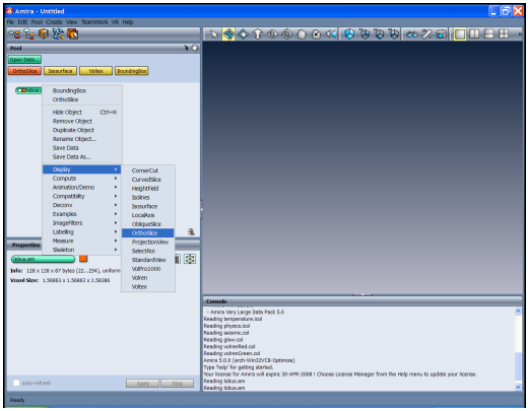


Figure 2.4: In order to attach a module to a data set, click on the green icon using the right mouse button. A popup menu appears containing all modules which can be used to process this particular type of data.

- Select the *OrthoSlice* module.

Now you should see various buttons and sliders in the Properties 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 sometimes referred to as *sagittal* and *coronal* (these are standard phrases used in radiology).

2.1.4 Interaction with the Viewer

The 3D viewer lets you look at the model from different positions. If you click on the *Trackball* button in the viewer toolbar, moving the mouse inside the viewer

Medivis Exhibit 1023

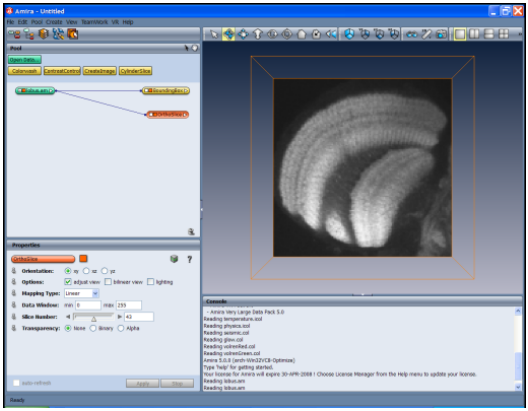


Figure 2.5: Visualization results are displayed in the 3D viewer window. Parameters or ports of a module are displayed in the Properties Area after you select the module.

window with the left mouse button pressed lets you rotate the object. If you click on the *Translate* or the *Zoom* buttons, you can translate or zoom the object. (For zoom, move the mouse up and down.)

Alternatively, with the middle mouse button pressed you can translate the object. For zooming press both the left and the middle mouse buttons 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

Medivis Exhibit 1023

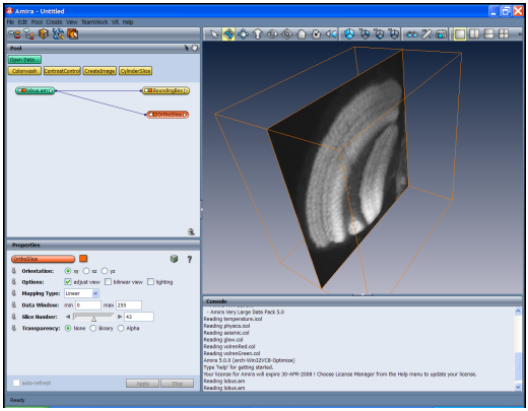


Figure 2.6: The *OrthoSlice* module is able to extract arbitrary orthogonal slices from a regular 3D scalar field or image volume.

with the interaction mode described above.

Each display module has a *viewer toggle* by which you can switch off the display without removing the module. This button is just to the right of 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 Object* from the *Pool* menu. Choose *Remove All* from the same menu to remove all modules.

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

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

2.2 How to load image data

Loading image data is one of the most basic operations in Amira. Other than with 2D images, there are not many standardized file formats containing 3D images. This tutorial guides you by means of examples on how to load the different kinds of 3D images into Amira. In particular this tutorial covers the following topics:

1. Using the *File/Open Data...* browser and setting the file format.
2. Reading 3D image data from multiple 2D slices.
3. Setting the bounding box or voxel size of 3D images.
4. The *Stacked Slices* file format.
5. Working with LargeDiskData.

2.2.1 The Amira File Browser

Image data is loaded in Amira with the *File/Open Data...* dialog. All *file formats* supported by Amira are recognized automatically either by a data header or by the file name suffix. What follows is only of concern in these cases:

- The automatic file format detection fails.
- 3D image data is stored in several 2D files.
- The data is larger than the available main memory.

Setting the file format
In most cases the format of a file is determined automatically, either by checking the file header or by comparing the file name suffix with a list of known suffixes. In the load dialog the file format is displayed in a separate column in detail view.

Example:

- Files containing the string `AmiraMesh` in the first line are considered *AmiraMesh* files.
- Files with the suffix .stl are considered STL files.

Medivis Exhibit 1023

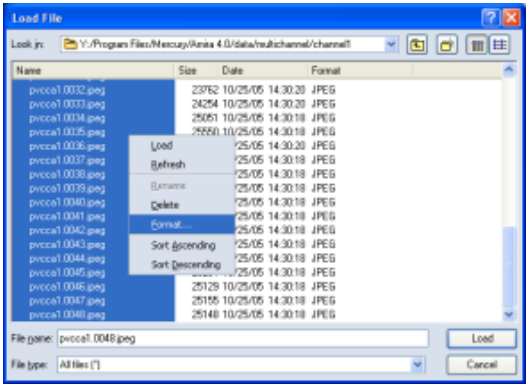


Figure 2.7: The *Format* option of the file browser

If automatic file format detection fails, e.g. because some non-standard suffix has been used, the format may be set manually using the *Format* entry in the pop-up menu of the *Load* dialog (right mouse button).

2.2.2 Reading 3D Image Data from Multiple 2D Slices

A common way to store 3D image data is to write a separate 2D image file for each slice. The 2D images may be written in TIFF, BMP, JPEG, or any other supported image *file format*. In order to load such data in Amira, all 2D slices must be selected simultaneously in the file browser. This can be done by clicking the first file and shift clicking the last one.

- Open the *File/Open Data...* dialog.
- Browse to the `/Amira-5/data/multichannel1/channel/` directory.
- Select the first file `pvcca1.0001.jpeg`
- `Shift`-click the last file (`pvcca1.0048.jpeg`).
- Click *Load*.

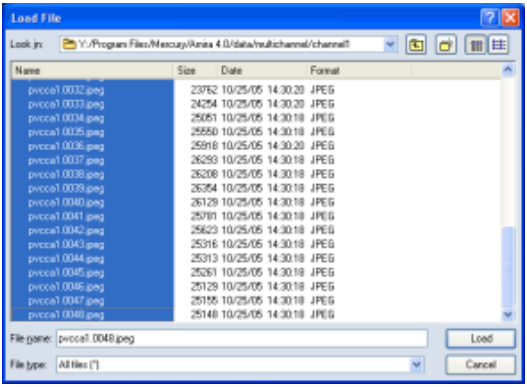


Figure 2.8: Loading multiple 2D images

2.2.3 Setting the Bounding Box

When loading a series of bitmap images, usually the physical dimensions of the images are not known to Amira. Therefore an *Image Read Parameters* dialog appears that prompts you for entering the physical extent of the *bounding box*. Alternatively, the size of a single voxel can be set. In Amira the bounding box of an object is the smallest rectangular, axis-aligned volume in 3D space that encompasses the object. *Note that in Amira the bounding box of a uniform data set extends from the center of the first voxel to the center of the last one. For example, if you have 256 voxels and you know the voxel size to be 1 mm, the bounding box should be set to 0 - 255 (or to some shifted range).*

- Enter 0.85 in the first and second text fields and 3.5 in third text field of the *Voxel Size* port.
- Click *OK*.

This method will always create a data set with uniform coordinates, i.e., uniform slice distance. In case of variable slice distances, the *StackedSlices* format should be used.

Medivis Exhibit 1023

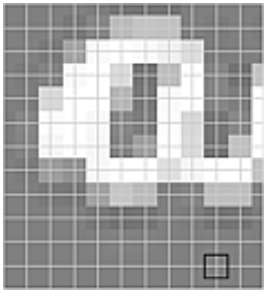


Figure 2.9: The definition of the bounding box in Amira. Different gray shades depict the intensity values defined on the regular grid (white lines). The black square depicts the extent of one voxel. The outer frame depicts the extent of the bounding box.

2.2.4 The Stacked Slices file format

Especially with histological serial sections it often happens that slices are lost during preparation. To handle such cases, Amira provides a special data type corresponding to a file format, called *Stacked Slices*. This file format allows a stack of individual image files to be read with optional z- values for each slice. The slice distance is not required to be constant. The images must be one-channel or RGBA images in an image format supported by Amira (e.g. TIFF). The reader operates on an ASCII description file, which can be written with any editor. Here is an example of a description file:

```
# Amira Stacked Slices
# Directory where image files reside
pathname C:/data/pictures
# Pixel size in x- and y-direction
pixelsize 0.1 0.1
# Image list with z-positions
picture1.tif 10.0
picture7.tif 30.0
picture13.tif 60.0
colstars.jpg 330.0
```


Medivis Exhibit 1023

```
end
```

Some remarks on the syntax:

- `# Amira Stacked Slices` is an optional header that allows Amira to automatically determine the file format.
- `pathname` is optional and can be included if the pictures are not in the same directory as the description file. A space separates the tag "pathname" from the actual pathname.
- `pixelsize` is optional, too. The statement specifies the pixel size in x- and y-directions. The bounding box of the resulting 3D image is set from 0 to pixelsize*(number_of_pixels-1).
- `picture1.tif 10.0` is the name of the slice and its z-value, separated by a space character.
- `end` indicates the end of the description file.
- Comments are indicated by a hash-mark character (#).

2.2.5 Working with Large Disk Data

Sometimes image data are so large that they do not fit into the main memory of the computer. Since the Amira visualization modules rely on the fact that data are in physical memory, this would mean that such data cannot be displayed in Amira. To overcome this, a special purpose module is provided that leaves most of the data on disk and retrieves only a user-specified subvolume. This subvolume can then be visualized with the standard visualization modules in Amira.

- Use the *File/Open Data...* dialog and go to c:/Program Files/Amira-5/data/medical/
- Right-click on the reg005.ctdata.am and select the *Format* entry from the pop-up dialog
- Select *AmiraMesh as LargeDiskData* as format and confirm your choice with *OK*.
- Press the *Load* button.

The data will be displayed in the Pool as a regular green data icon. The info line indicates that it belongs to the data class *HxRawAsExternalData*.

- Right mouse click, attach a *BoundingBox* module.

Medivis Exhibit 1023

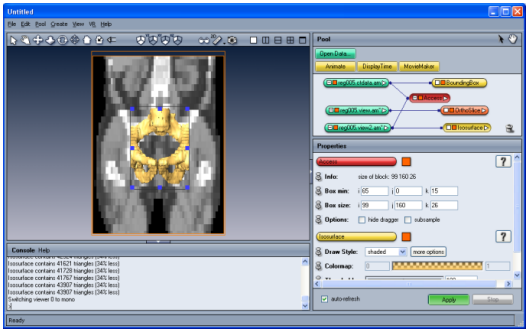


Figure 2.10: The usage of *AmiraMesh* as *LargeDiskData*. For instantaneous update, the *auto-refresh* check boxes of the *Access* and *Isosurface* modules have been checked

- Right mouse click, attach an *Access* module.
- Select the *Access* module in the Pool and enter 224, 161, and 59 into the *BoxSize* text fields.
- Check *Subsample* and enter 4 4 2 into the *Subsample* fields and press the *Apply* button.

This retrieves a down-sampled version of the data. Disconnect the *reg005.view.am* data icon from the *Access* module and use it as an overview (e.g. with *OrthoSlice*).

- Select the *Access* module in the Pool and deselect the *subsample* check box.
- Use the dragger box in the viewer to resize the subvolume.
- Press the *Apply* button.
- Attach an *Isosurface* module to the *reg005.view2.am* (set threshold set to 100).

Tip: To browse the data, check the *auto-refresh* check box for the *Access* and *Isosurface* modules. Now each time the blue subvolume dragger is repositioned, the visualization is updated automatically.
Loading *AmiraMesh*, *StackedSlices*, and *Raw* "*asLargeDiskData*" is a convenient and fast way of exploring data that exceed the size of system memory. However, especially with *StackedSlices*, it is not always the most efficient way of doing this. Amira can store the image data in a special format that facilitates the random

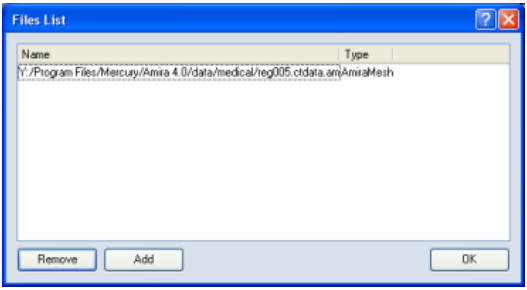


Figure 2.11: The *Input* dialog of the *ConvertToLargeDiskData* module.

retrieval of data from disk.

- Choose from the *Create/Data* menu *ConvertToLargeDiskData*
- Click *Browse* from the *Input* port.
- Click *Add*, go to `/Amira-5/data/medical/` and select `reg005.ctdata.am`, then click *Load*.
- Click *Browse* from the *Output* port.
- Go to `C:/tmp/` and enter a filename of your choice.
- Press the *Apply* button.

Although you will most likely not notice any difference with the small image data used in this tutorial, this method for retrieving large data significantly accelerates the *Apply* operation.

2.3 Visualizing 3D Images

This section 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

Medivis Exhibit 1023

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 2.1 (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 University of Wuerzburg.

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

2.3.1 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 *xz* or *coronal* in the *Orientation* port.
- Similarly, for *OrthoSlice3* choose *yz* or *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 Display submenu of the lobus data popup menu.

2.3.2 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

Medivis Exhibit 1023

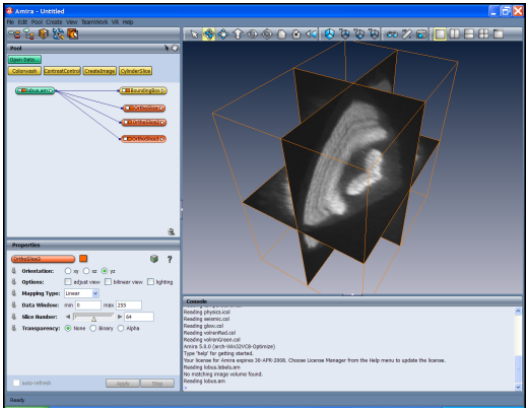


Figure 2.12: Lobus data set visualized using three orthogonal slices.

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*.
- 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.

2.3.3 Resampling the Data

Now we are going to compute and display an *isosurface*. 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 isosurface tool to work,

Medivis Exhibit 1023

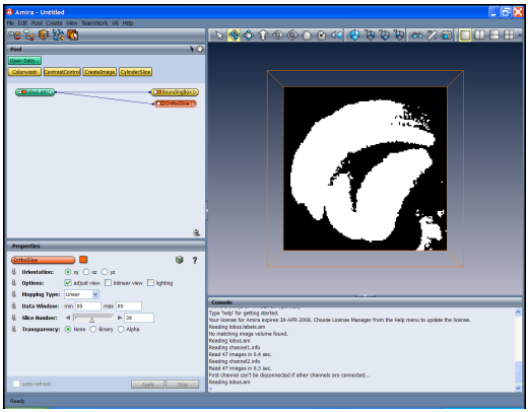


Figure 2.13: By adjusting the data window of the *OrthoSlice* module a suitable value for threshold segmentation can be found. Intensity values smaller than the min value will be mapped to black, intensity values bigger than the max value will be mapped to white.

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. Typically you must press the green *Apply* button at the bottom of the Properties Area 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 *Apply* 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

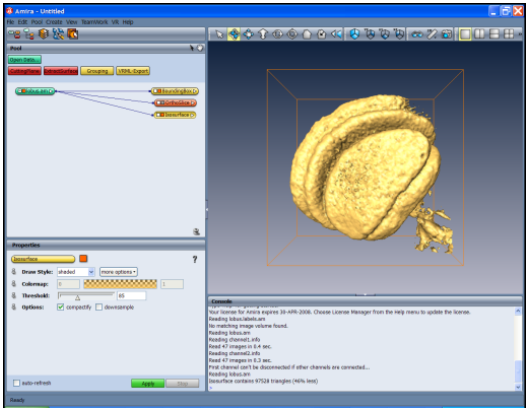


Figure 2.14: Lobus data set visualized in 3D using an isosurface.

valid source, the connection line appears highlighted in lighter blue.

2.3.4 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.
- Press the *Apply* button.

2.3.5 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.

Medivis Exhibit 1023

- Select the *lobus.am* data icon.
- Click on the Crop Editor button in the Properties 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 Properties 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.

2.3.6 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 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 Pool other than the *lobus.am* data record.

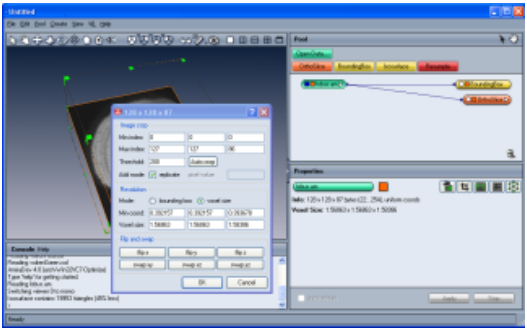


Figure 2.15: The crop editor works on uniform scalar fields. It allows you to crop a data set, to enlarge it by replicating boundary voxels, or to modify its coordinates, i.e. to scale or shift its bounding box.

- 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.
- Press the *Apply* 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 *Options*.

Internally, the voltex module makes heavy use of OpenGL texture mapping. Both texture modes, 2D and 3D, are implemented. 3D textures yield slightly better results. However, this mode is not supported by all graphics boards. 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 orienta-

Medivis Exhibit 1023

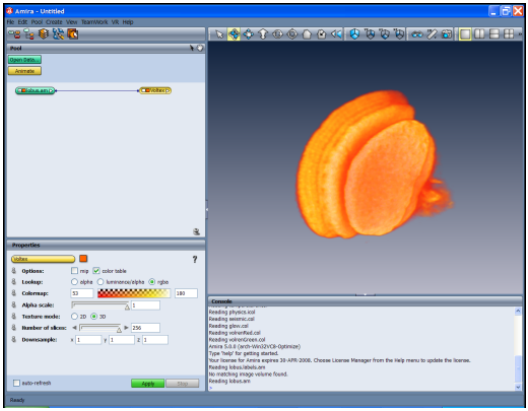


Figure 2.16: The *Voltex* module can be used to generate maximum intensity projections as well as volume renderings based on an emission-absorption model. In both cases, 2D or 3D texture mapping techniques can be applied applied.

tion 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.

- Make sure the *mip* button of port *Options* is unchecked.
- If 3D texture mode is enabled, choose about 200 slices.
- 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 *Apply* 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 *Apply* 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.

2.4 Segmentation of 3D Images

By following this step-by-step tutorial you will learn how to interactively create a segmentation of a 3D image. A segmentation assigns to each pixel of the image a label describing to which region or material the pixel belongs, e.g., bone or the kidney. The segmentation is stored in a separate data object called a *LabelField*. A segmentation is the prerequisite for surface model generation or accurate volume measurement.
This tutorial comprises the following steps:

1. Creation of an empty LabelField.
2. Interactive editing of the labels in the *Image Segmentation Editor*.
3. Measuring the volume of the segmented structures.
4. An alternative segmentation method: Threshold segmentation.

2.4.1 Interactive Image Segmentation

- Load the *lobus.am* data file from the directory *data/tutorials*.
- Right click on the green icon and choose *LabelField* from the Labelling section.

A new green icon appears, the LabelField that will hold the segmentation results. Simultaneously, the image segmentation editor is displayed in the 3D viewer pane. By default, the segmentation editor operates in 2-viewer mode . In this mode, one 2D viewers with different orientations and an additional 3D viewer are displayed.
The tools of the segmentation editor are displayed where the Pool and Properties Area are normally displayed. For this tutorial, you'll want to display the segmentation tools.

- Use the mouse to expand the viewer so that you have more room to maneuver in.
- In the lower right view, use the slider on the bottom to scroll through the slices. Go to slice 20. You see two bigger structures and one structure just appearing on the top.
- If necessary, click on the second button under the label *Zoom and Data Window* to zoom out the data so that you have a view of the entire slice.
- Click on the second button under the label *Tools*, the brush.

Medivis Exhibit 1023

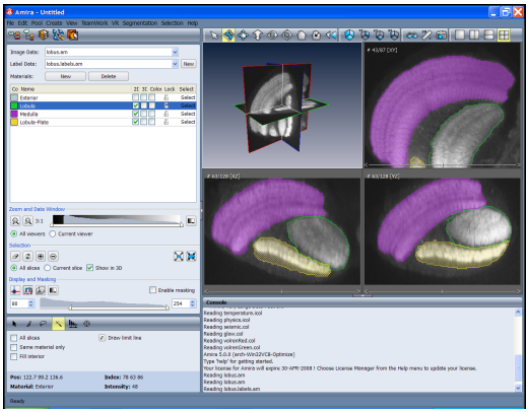


Figure 2.17: Image segmentation editor after selecting and assigning pixels for two structures in one slice

- Mark the rightmost structure with the mouse. Hold down the control button to unselect wrongly selected pixels if necessary.
- When done, select the entry *Inside* in the *Materials* list. Then click the **+** button under the *Selection* label.

The previously selected pixels are now assigned to the material Inside. You can right click on the entry *Inside* in the *Materials* list and choose a different draw style (for example, dotted).

- Click into the material list and choose *New Material* from the right button menu.
- Mark the middle structure using the brush, select the new material in the Materials list and assign the pixels to that structure.
- Go to slice 21 and practice by segmenting the two structures.

Hint: If you prefer to work with one larger view rather than four smaller views, click on the *Single Layout* button in the viewer toolbar. To cycle through each of the four views, press the *Single Layout* button repeatedly. To return to 4-viewer mode, press the *Layout4* button.

If a structure does not change a lot from slice to slice, you can use interpolation.

- Go to slice 22 and mark the right structure using the brush. Go to slice 31 and mark the same structure.
- Choose from the menu bar: Selection/Interpolate.
- Scroll through the data set. You should see that the in between slices 23 to 30 are selected too.
- In order to assign the selected pixels in all slices to the Inside material, select the Inside material in the list, then click the + button.

- Repeat the procedure between slice 32 and 50.
- Repeat the procedure for the middle structure.

Hints:

- It is highly recommended to frequently save the segmentation results while working. In order to do so, select the label field in the Amira main window and choose *Save* or *Save As...* from the Amira File menu.
- The brush is only the most basic segmentation tool. The segmentation editor provides many more functions that are described on its *reference page*.
- There are many useful key bindings, including SPACE and BACKSPACE to change the slice number or ”d” to toggle the draw style.
- Of course you can give the materials more meaningful names or colors using the context menu (right mouse button in the list).

At this point save the label field and leave the Editor. In the next tutorial you will learn how to create a 3D surface model from the segmentation results.

2.4.2 Volume Measurement

Once a structure is segmented, you can easily measure its volume:

- Right click on the LabelField’s green icon. Choose *Measure/MaterialStatistics*.
- Press the *Apply* button. A new icon appears.
- Select this icon and press the *Show* button.

The units in the volume column depend on the units you have specified the voxel size. In case of the *lobus.am*, the voxel size is in μm, therefore the volume is in μm^3.

Medivis Exhibit 1023

2.4.3 Threshold Segmentation

We now describe an alternative way of segmentation, which can require less manual interaction, but only works for images with good quality.
In some cases a satisfying segmentation can be achieved automatically based solely on the gray values of the image data set.
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.

- 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 Image Data Visualization.
- Press the *Apply* button.

By this procedure each voxel having a value lower than the threshold is assigned to *Exterior* and each voxel 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 which have voxel values above the threshold to be 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 you press the *Apply* button, a new data object is computed and its icon appears in the 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.

2.4.4 Refining Threshold 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 Image Segmentation icon in the green title bar in the Properties Area.

In response the image segmentation editor is popped up.

Medivis Exhibit 1023

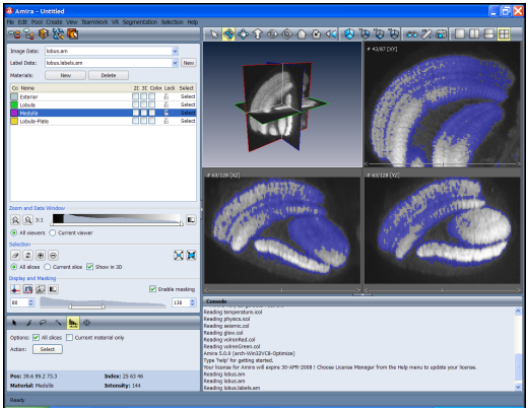


Figure 2.18: Data from confocal microscopy is segmented using Amira's image segmentation editor.

- In the lower right view, use the slider on the bottom to select slice 39.
- Choose a magnification ratio of 2:1 by pressing the zoom-up button in the Zoom and Data region of the editor.

The image segmentation editor shows the image data to be segmented (*lobus.am*) as well as brownish 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 brownish contours. This is what we want to improve now.

- Choose *Remove Islands* from the editor's *Segmentation* menu. In response, a little dialog window appears.
- In the dialog window select the *all slices* mode. Then press *Remove* 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 *Segmentation* menu. Another dialog box appears.
- Select the *3D volume* mode and press the *Apply* button in order to execute the smoothing operation.
- To examine the results of the filter operations, browse through the label field

Medivis Exhibit 1023

slice by slice. In addition to the slice slider you may also use the cursor-up and cursor-down keys for this.

- Leave the Segmentation Editor.

2.5 Surface Reconstruction from 3D Images

By following this step-by-step tutorial, you will learn how to generate a triangular surface grid for an object embedded in a voxel data set. A surface grid allows for producing a 3D view of the object's surface and can be used for numerical simulations.
The generation process consists of these steps:

1. Extracting surfaces from segmentation results
2. Simplifying the surface

As a prerequisite for the following steps, you need a label field, which holds the result of a previous image segmentation. You can either use the label field which you created in the previous tutorial or load the provided *lobus.labels* data set from the *data/tutorials* directory.

2.5.1 Extracting Surfaces from Segmentation Results

Now we let Amira construct a triangular surface of the segmented object.

- Connect a *SurfaceGen* module to the *lobus.Labels* data.
- Press the *Apply* button.

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

2.5.2 Simplifying the Surface

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

- Select the surface *lobus.surf*.
- Click on the Simplifier button (triangle mesh icon) in the Properties Area.

Medivis Exhibit 1023

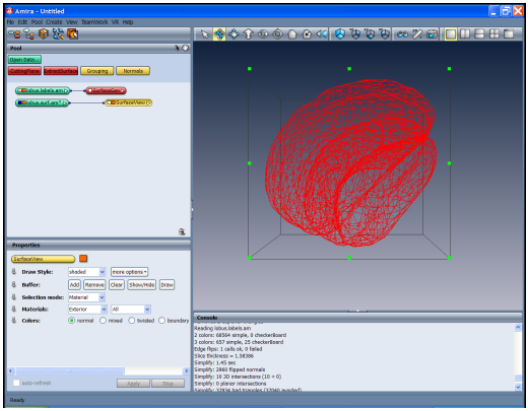


Figure 2.19: Surface representation of optical lobus as triangular grid

- Set the desired number of faces to 3500 in the *Simplify* port.
- Turn on the *fast* toggle in the *Options* port. This option disables some time-consuming intersection tests.
- Push the *Simplify now* button in the *Action* port.

The number of triangles is reduced to about 3500 now. The progress 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.surf* data object.

The *SurfaceView* module maintains an internal buffer and displays all triangles stored in this buffer. By default the buffer shows all triangles forming the boundary to the exterior. If you change the selection at the *Materials* port, the newly selected triangles are highlighted, i.e., they are displayed using a red wireframe representation. The *Add* and *Remove* buttons cause the highlighted triangles to be added to or removed from the buffer, respectively. You may easily visualize a subset of all triangles using a 3D selection box or by drawing contours in the 3D viewer. Press the *Clear* button of the *Buffer* port to see the display shown in Figure 2.19.

Medivis Exhibit 1023

2.6 Creating a Tetrahedral Grid from a Triangular Surface

By following this step-by-step tutorial, you will learn how to generate a volumetric tetrahedral grid from a triangular surface as created in the previous tutorial. A tetrahedral grid is the basis for producing various views of inner parts of the object, e.g., cuts through it, and is frequently used for numerical simulations.
The generation process consists of these steps:

1. Simplifying the surface
2. Editing the surface
3. Generating a tetrahedral grid

As a prerequisite for the following steps, you need a triangular surface, which is usually the result of a previous surface reconstruction. Load the supplied *lobus.surf* data set from the *data/tutorials* directory.

2.6.1 Simplifying the Surface

Usually the number of triangles created by the *SurfaceGen* module is far too large for subsequent operations, e.g., for a numerical simulation. Thus, the number of triangles should be reduced in a *surface simplification* step. In Amira a *Surface Simplification Editor* is provided for this purpose. There may be different goals for the simplification:

- In *computer graphics*, one wants to prescribe just the number of faces, because this determines the rendering speed.
- For a *numerical simulation*, one often wants to specify the maximum edge length occurring in the grid model.

This tutorial shows how the maximum edge length can be controlled during simplification.

- Select the surface *lobus.surf*.
- Click on the Simplifier (triangle mesh icon) in the Properties Area.
- Set the desired number of faces to 1000 and the desired maximal distance (i.e. edge length) to 10 in the *Simplify* port.
- Leave the *fast* toggle turned off in the *Options* port. This will cause intersection tests to be performed during simplification, which will considerably

Medivis Exhibit 1023

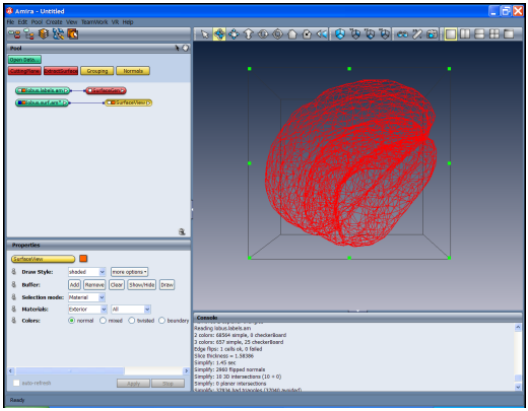


Figure 2.20: Surface representation of optical lobus as triangular grid

reduce the probability that the simplified surface contains self intersections.

- Press the *Simplify now* button in the *Action* port.

Simplification terminates when either of the limits given by the number of faces or the maximum distance is reached. The progress bar tells you how much of the simplification task has already been done. In this example the maximum distance will be the limiting factor, and the resulting surface will contain about 6000 faces.

Besides the maximum edge length, the minimum edge length occurring in the surface should also be controlled, because the ratio of maximum and minimum edge length will influence the quality of the resulting tetrahedral grid. This ratio should not be much larger than 10. If edges that are too short occur in the simplified surface, they can be removed as follows.

- Set the desired minimum distance to 2 in the *Simplify* port.
- Observe the number of faces as shown at the *Surface* port, and press the *Contract edges* button in port *Action*. All edges shorter than 2 will be contracted. In this example about 30 small edges will be detected. You will observe that the number of faces slightly decreases.

Medivis Exhibit 1023

2.6.2 Editing the Surface

As a second step of preparation for tetrahedral grid generation, invoke the *Surface Editor*.

- Select the surface *lobus.surf*.
- Leave the *Surface Simplification Editor* (Simplifier) by again clicking on the triangle mesh icon.
- Enter the *Surface Editor* by clicking on the Surface Editor button in the Properties Area.

Automatically, a *SurfaceView* module will be attached to the *lobus.surf* surface. For details about that module see its description.
When the *Surface Editor* is invoked, the *Surface* menu is added to Amira's menu bar and a new toolbar is placed just below Amira's viewer toolbar. The *Surface/Tests* menu contains 6 specific tests which are useful for preparing a tetrahedral grid generation. Each of the tests creates a buffer of triangles which can be cycled through using the back and forward buttons.

- Select *Intersection test* from the *Surface/Tests* menu. The total number of intersecting triangles is printed in the console window. Intersections shouldn't occur too often if toggle *fast* was switched off during surface simplification. In case they occur, the first of the intersecting triangles and its neighbors are shown in the viewer window.
- You can manually repair intersections using four basic operations: *Edge Flip*, *Edge Collapse*, *Edge Bisection*, and *Vertex Translation*. See the description of the *Surface Editor* for details.
- After repairing, invoke the intersection test again by selecting it from the *Surface/Tests* menu or by pressing the *Compute* button.
- When the intersection test has been successfully passed, select the *Orientation test* from the *Surface/Tests* menu. After surface simplification, the orientation of a small number of triangles may be inconsistent, resulting in a partial overlap of the materials bounded by the triangles. In case of such incorrect orientations, which should occur quite rarely, there is an automatic repair. If this fails, the detected triangles will be shown, and you can use the above mentioned manual operations for repair. **Note:** There are two prerequisites for the orientation test: the surface must be free of intersections, and the outer triangles of the surface must be assigned to material *Exterior*. If the surface does not contain such a material or if the assignment to *Exterior*

Medivis Exhibit 1023

is not correct, the test will falsely report orientation errors.

A successful pass of the intersection and orientation test is mandatory for tetrahedral grid generation. These tests are automatically performed at the beginning of grid generation. So you can directly enter the *TetraGen* module (see below) and try to create a grid. If one of the tests fails, an error message will be issued in the console window. You can then go back to the *Surface Editor* and start editing.
The remaining three tests analyze the surface mesh with respect to different quality measures. These tests only need to be performed if the tetrahedral quality of the volumetric grid plays an important role, e.g., if the grid will be used for a numerical simulation.

- Select *Aspect ratio* from the *Surface/Tests* menu. This computes the ratio of the radii of the circumcircle and the incircle for each triangle. The triangle with the worst (i.e. largest) value is shown first, and the actual value is printed in the console window. The largest aspect ratio should be below 20 (better below 10). Fortunately there is an automatic tool for improving the aspect ratio included in the *Surface Editor*.
- Select *Flip edges* from the *Edit* menu. A small dialog window appears. In the Radius Ratio area, set the value of the "Try to flip a triangle..." field to 10. Select mode *operate on whole surface*. Press button *Flip*. All triangles with an aspect ratio larger than 10 will be inspected; if the aspect ratio can be improved via an edge flip, this will be done automatically. The console window will tell you the total number of bad triangles and how many of them could be repaired. Press the *Close* button to leave the *Flip edges* tool.
- Select again *Aspect ratio* from the *Surface/Tests* menu. Only a small number of triangles with large aspect ratio should remain after applying the *Flip edges* tool.
- Select *Dihedral angle* from the *Surface/Tests* menu. For each pair of adjacent triangles, the angle between them at their common edge will be computed. The triangle pair including the worst (i.e. smallest) angle is shown in the viewer, and the actual value is printed in the console window. The smallest dihedral angle should be larger than 5 degrees (better larger than 10).
- For a manual repair of a small dihedral angle proceed as follows: select the third points of both triangles (i.e. the points opposite to the common edge) and move them away from each other. For moving vertices you must enter *Vertex Translation* mode by clicking on the first icon from the right on the

Medivis Exhibit 1023

top of the viewer window or by pressing the `"t"` key. If the viewer is in viewing mode, switch it into interaction mode by pressing the `ESC` key or by clicking on the arrow icon (the first icon from the top) on the right of the viewer window. Click on the vertex to be moved. At the picked vertex a point dragger will be shown. Pick and translate the dragger for moving the vertex.

- In some cases an edge flip might also improve the situation. Enter *Edge Flip* mode by clicking on the third icon from the right on the top of the viewer window or by pressing the `"f"` key. Switch the viewer into interaction mode. Click on the edge to be flipped.
- Select *Tetra quality* from the *Surface/Tests* menu. For each surface triangle the aspect ratio of the tetrahedron which would probably be created for that triangle will be calculated. The aspect ratio for a tetrahedron is defined as the ratio of the radii of the circumsphere and the inscribed sphere. The triangle with the worst (i.e. largest) value is shown in the viewer, and the actual value is printed in the console window. The largest tetrahedral aspect ratio should be below 50 (better below 25). If all small dihedral angles have already been repaired, the tetra quality test will mainly detect configurations where the normal distance between two triangles is small compared to their edge lengths. Again, the *vertex translation* and the *edge flip* operation are best suited for a manual repair of large tetrahedron aspect ratios.
- Leave the *Surface Editor* by again clicking on the Surface Editor button in the Properties Area.

Hint: In order to see the entire surface again, select the SurfaceView icon, then press its *Show/Hide* button, then press the *ViewAll* button in the viewer toolbar.

2.6.3 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 may be time consuming it can be performed as a *batch job*. You can then continue working with Amira while the job is running. However, for demonstration purposes we want to compute the grid right inside Amira.

- Connect a *TetraGen* module to the *lobus.surf* surface by choosing *Compute TetraGen* from the popup menu over the *lobus.surf* icon.

- Leave toggle *improve grid* switched on and toggle *save grid* switched off at the *Options* port. The *improve grid* option will invoke an automatic post-processing of the generated grid, which improves tetrahedral quality by some iterations that move inner vertices and flip inner edges and faces. See the description of the *Grid Editor* for details.

  If toggle *save grid* is selected, an additional port *Grid* appears, where you can enter a filename. The resulting tetrahedral grid will be stored automatically under that name. If you want to run grid generation as a batch job, you must select the *save grid* option.
- Press the *Meshsize* button of the *Action* port. An editor window will appear. It allows you to define a desired mesh size, i.e., mean length of the inner edges to be created, for each region. For this you must enter the bundle of that region, and select parameter *MeshSize*. Then you can change the value in the text field at the lower border of the editor. There are some predefined region names in Amira for which a default mesh size will be automatically set. Make sure that the default values are suitable for your application. If you are not sure about a suitable value, set the desired mesh size to 0. In this case the mean edge length of the surface triangles will be used.
- Press the *Run now* button at 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 tetrahedra which already have been created. In some situations grid generation may fail, for example, if the input surface intersects itself. Then an error message will occur at the *Console Window*. In this case go back to 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 Pool. You can select this icon in order to see how many tetrahedra the created grid contains. If grid generation takes too long, 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 press the *Add to* button of the *Buffer* port.

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 high-

Medivis Exhibit 1023

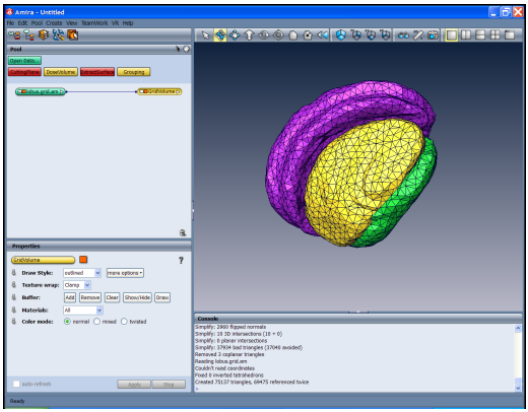


Figure 2.21: Volumetric representation of optical lobe as tetrahedral grid

lighted, 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.

Similar to the *Surface Editor*, there is a *Grid Editor* which can be invoked by selecting the green icon of the tetrahedral grid and clicking on the pencil icon (first from the right in the title bar) in the Properties Area. The editor allows for selecting tetrahedra with respect to different quality measures, e.g., aspect ratio, dihedral angles at tetrahedron edges, solid angles at tetrahedron vertices, and edge length. The editor contains several modifiers that can be applied for improving mesh quality.

2.7 Warping and Registration 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.

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:

Medivis Exhibit 1023

1. Displaying data sets in two viewers.
2. Creating a landmark set.
3. Alignment via a rigid transformation.
4. Warping two image volumes.

2.7.1 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 oriented differently. We want to look at each lobe in its own viewer.

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

You can see the two lobes in both viewers.

- Visualize the first lobe (blue) in the upper viewer and the second lobe (yellow) in the lower viewer. This is done by deactivating the lower viewer toggle (orange buttons in the icons) of the *Isosurface* module and by deactivating the upper viewer toggle in the *Isosurface2* module.

2.7.2 Creating a Landmark Set

Now, let us create a landmark set object.

- From the main window's *Create* menu select *Data/Landmarks (2 sets)*

in order to create an empty landmark object. A new green icon will show up in the Pool. Since we are going to match two objects by means of corresponding landmarks we had to select the landmark objects containing 2 sets of landmarks (*Landmarks (2 sets)*).

- Select object *Landmarks*.
- Launch the *Landmark Editor* by clicking on the Landmark Editor button in the Properties Area.

Medivis Exhibit 1023

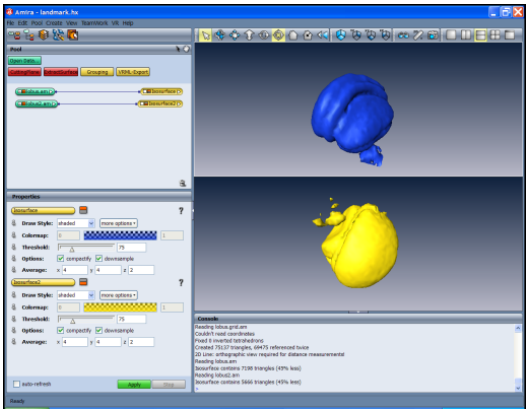


Figure 2.22: Two lobes visualized with isosurfaces in 2-viewer layout.

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 available now. We use the editor to define some markers in both objects. For the following, it is useful to push-pin the three ports on the landmark editor. In order to do so, select the gray push-pin toggles to the left of the port 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*.
- Set the viewer toggles of the two *LandmarkView* modules in the same manner as described for the *SurfaceView* modules previously. *LandmarkView* should be visible in the upper viewer and *LandmarkView2* in the lower 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.

Medivis Exhibit 1023

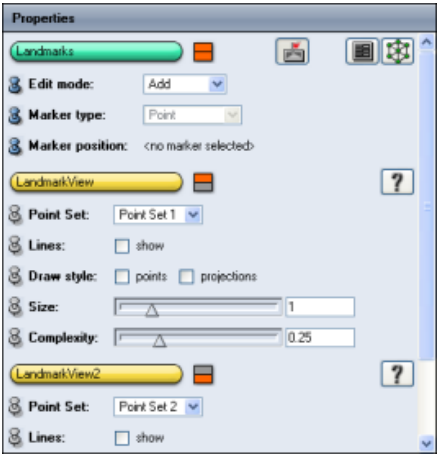


Figure 2.23: The image shows how the viewer toggles and *Point Set* ports should be set.

- Rotate the two lobes to align them roughly.

Now, we are ready to define and set corresponding landmarks. Select the *Landmarks* 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!!
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

Medivis Exhibit 1023

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 defining 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.

2.7.3 Registration via a Rigid Transformation

To register one object to the other, connect a *LandmarkWarp* module to the *Landmarks* object by clicking with the right mouse button on the *Landmarks* icon in the 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 white square 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 well as possible by performing only rotations and translations of the first object. To do that choose

Method: *Rigid*

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

Direction: $1 \rightarrow 2$.

Then press *Apply* 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 *Apply* 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.

2.7.4 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

Medivis Exhibit 1023

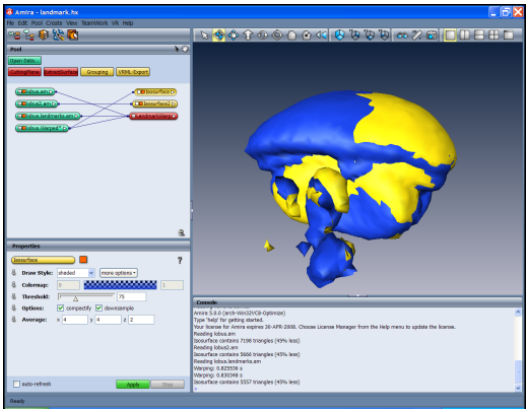


Figure 2.24: Result of landmark-based elastic warping using the Bookstein method.

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

To visualize the 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.

2.8 Registration of 3D image data sets

In medical imaging a frequent task has become the registration of images from a subject taken with different imaging modalities, where the term *modalities* here refers to imaging techniques such as Computed Tomography (CT), Magnetic Resonance Tomography (MRT) and Positron Emission Tomography (PET). The challenge in inter-modality registration lies in the fact that e.g in CT images "bright" regions are not necessarily bright regions in MRT images of the same subject.
In registration typically one of the data sets is taken as the *reference*, and the other one is transformed until both data sets match. Amira's *AffineRegistration* module

Medivis Exhibit 1023

provides an affine registration, i.e. it determines an optimal transformation with respect to translation, rotation, anisotrope scaling, and shearing.
Closely related to registration is the task of *image fusion*, i.e., the simultaneous visualization of two registered image data sets.
This tutorial shows how a registration can be performed and how to visualize the results. The following issues will be discussed:

1. Basic manual registration using the *Transform Editor*
2. Automatic registration
3. Image fusion

2.8.1 Basic Manual Registration

In this tutorial we want to register a CT and an MRT data set of a patient, showing the pelvic region. The images are located in the Amira data directory in the subdirectory *registration*.

- Load the files `data/registration/CT-data.am` and `data/registration/MRT-data.am` into Amira.
- Attach an *OrthoSlice* module to each of the data sets.
- Select *Coronal* (or *xz*) at the *Orientation* port of the *OrthoSlice* module connected to the MRT data.
- Select a camera position for the 3D viewer where you can see both the axial slices of the CT data and the coronal slices of the MRT data.
- Select slice 31 at the *Slice Number* port of the *OrthoSlice* module connected to the CT data.

Now one OrthoSlice module should show an axial slice through the hip joints. Move the coronal slice through the MRT data. You will observe that the two data sets are not correctly aligned.

- Select the green icon of the MRT data set. Invoke the *Transform Editor* by pressing the Transform Editor button in the Properties Area. The *Transform Editor* enables you to specify an affine transformation, including translation, rotation, and scaling. This transformation will be applied to the corresponding 3D data set. You can edit the transformation interactively in the 3D viewer using different Open Inventor draggers. You can also enter transformations numerically.

- Press the *Dialog* button. A dialog window will pop up.
- Enter `-2` at the third text field of the *Translation* port of the dialog window. This means a translation of -2 cm in the z direction.
- Enter `5` at the first text field at the *Rotation* port. This means a rotation of 5 degrees. The axis of rotation is defined at the next ports, here it is the z-axis.
- Press the *Close* button of the dialog window. Leave the Transform Editor by pressing again the Transform Editor button.

Inspect some coronal slices through the MRT data set. Now there is a better alignment of the CT and MRT data, but it's still not perfect.

2.8.2 Automatic Registration

The *Registration* module provides an automatic registration via optimization of a quality function. For registration of data sets from different imaging modalities, the *Normalized Mutual Information* is the best suited quality function. In short, it favors an alignment which "maps similar gray value structures to similar gray value structures". A hierarchical strategy is applied, starting at a coarse resampling of the data sets, and proceeding to finer resolutions later on.

- Attach an *Registration* module to the MRT data set by choosing *Compute/AffineRegistration* from the popup menu over the *MRT-data.am* icon.
- Connect the second input port *Reference* of module *Registration* to the CT data set. For this click with the right mouse button on the white square at the left hand side of the module's icon.
- Select toggle *Extended options*. More ports of the *Registration* module will become visible.

The first three ports of the *Registration* module define the optimization strategy. The default settings mean that an *ExtensiveDirection* optimizer is used for the coarse levels and a *QuasiNewton* optimizer for the finest two levels of the resampling hierarchy. At the *CoarsestResampling* port you can select the resampling rate for the coarsest resolution level. The default resampling rate is smaller in the z direction because the reference data set has a finer resolution in the x and y direction (0.17 cm) than in the z direction (0.5 cm). For the default settings (8,8,3), the resampling hierarchy will consist of four levels: (8,8,3), (4,4,2), (2,2,1), and the original resolution, (1,1,1).

Medivis Exhibit 1023

The *Normalized Mutual Information* is calculated from gray value histograms. The selected histogram ranges should enclose the essential information of each data set. Normally you can choose the same range as for visualization via an *OrthoSlice* module.

- Set `-200` and `200` at the two text fields of the *Histogram range reference* port.

At the *Transform* port you can specify the type of affine transformation. The default settings mean that only rigid body motions will be applied, i.e. translations and rotations.
Option *Ignore finest resolution* means that optimization is done on all but the finest level of the resampling hierarchy. This will slightly reduce the accuracy, but save a large amount of computing time.
Automatic registration may take some time depending on the resolution of the images and the quality of the pre-alignment. You can interrupt automatic registration at any time using the stop button. Interruption may take some seconds. The progress bar shows the current hierarchy level and the progress at that level.

- Start automatic registration by pressing the *Register* button at the *Action* port.

2.8.3 Image Fusion

The task of image fusion is the simultaneous visualization of two data sets. To that end Amira offers for all types of slicing modules (*Orthoslice* and *ObliqueSlice*) the *Colorwash* module. Using *Colorwash*, the images from one data set can be overlayed over that of another taking into account their orientation in space.

- Remove the *OrthoSlice* module connected to the MRT data set.
- Select the green icon of the MRT data set.
- Select a *Colorwash* module from the popup menu over the icon of the *OrthoSlice* module connected to the CT data set.
- Select the yellow icon of the *Colorwash* module.
- Select the *physics.icol* colormap at port *Colormap*.
- Set `70` as the upper bound for the colormap range.
- Select the icon of the *OrthoSlice* module.
- Inspect axial slices with slice numbers between 15 and 45.

Medivis Exhibit 1023

You will observe a good alignment of the pelvic bone from both data sets. The soft tissue contours are not perfectly aligned because there was some soft tissue deformation between both scans. This cannot be described by a rigid transformation.
In image fusion it is sometimes necessary to observe all three orthogonal directions simultaneously. For that the *StandardView* module can be used for image fusion. The *StandardView* module opens a separate window with four viewers, three of them showing the three orthogonal slices of the image data and a fourth being a new instance of the 3D viewer.

- Attach a *StandardView* module to the CT data set by choosing *Display StandardView* from the popup menu over the *CT-data.am* icon. Amira's *Viewer* window will now be split into four parts showing three orthogonal slices through the CT data, and the 3D Viewer in the upper left part.
- Connect the second input port *OverlayData* of the *StandardView* module to the MRT data set. For this, click with the right mouse button on the white square at the left hand side of the module's icon.
- Select slice numbers 179, 149, and 31 at ports *Slice x*, *Slice y*, and *Slice z*, respectively. The three orthogonal slices will show the hip joints now.
- Increase the zoom-factor by clicking twice on button > at the *Zoom* port.
- Select *checkerboard* at the *Overlay mode* port.
- Vary the size of the checkerboard tiles by moving the slider at the *Pattern size* port. In this way you can again check the alignment of the CT and MRT data sets.

The bone contours around the hip joints show a good match. Note that bone is represented by white (i.e high intensity) voxels in the CT data, but may occur as both white and black voxels in the MRT data. In the axial slice you can observe larger deviations of the outer body contour between the CT and MRT data.

2.9 Alignment of 2D Physical Cross-sections

Many microscopic techniques require the sample to be physically cut into slices. Then images are taken from each cross-section separately. Usually the images will be misaligned relative to each other. Before a 3-dimensional model of the sample can be reconstructed the images have to be aligned taking into account translation and rotation. This tutorial shows how this task can be performed using the *AlignSlices* module.

Medivis Exhibit 1023

The following issues will be discussed:

1. Basic manual alignment
2. Alignment via landmarks
3. Optimizing the quality function
4. Resampling the input data
5. Other alignment options

2.9.1 Basic Manual Alignment

In this tutorial we want to align 10 microscopic cross-sections of a leaf showing a stomatal pore. The images are located in the data directory in the subdirectory *align*. Each slice is stored as a separate JPEG image. The file *leaf.info* defines a 3D image stack consisting of the 10 individual slices. It is a simple ASCII file as described in the *stacked slices* file format section.

- Load the file `data/align/leaf.info`.
- Create an align module by choosing *Compute AlignSlices* from the popup menu over the *leaf.info* icon.
- Press the *Edit* button of *AlignSlices*.

A new graphics window is popped up allowing you to interactively align the slices of the 3D image stack. To facilitate this task, usually two consecutive slices are displayed simultaneously. One of the two slices is *editable*, i.e., it can be translated and rotated using the mouse. By default the upper slice is editable. This is indicated in the tool bar of the align window (the ”upper slice” button is selected).

- If necessary, press the zoom out button (”-” magnifying glass) to allow the entire slice to be visible in viewer.
- Translate the upper slice by moving the mouse with the left mouse button pressed down.
- Rotate the upper slice by moving the mouse with the left mouse button and the Ctrl key pressed down. Alternatively, slices can be rotated using the middle mouse button.
- Make the lower slice editable by selecting the ”lower slice” tool button. Translate and rotate the lower slice.
- Hold down key number 1. While this key is hold down only the lower slice is displayed.

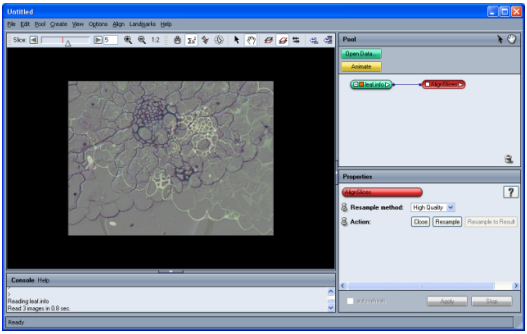


Figure 2.25: User-interface of the align tool.

- Hold down key number 2. While this key is hold down only the upper slice is displayed.
- Pressing key number 1 and 2 also changes the editable slice. Note how the slice tool buttons change their state.

Other pairs of slices can be selected using the slider in the upper left part of the align window. Note that the number displayed in the text field at the right side of the slider always refers to the editable slice. The next or the previous pair of slices can also be selected using the space bar or using the backspace key, respectively. The cursors keys are used to translate the current slice by one pixel in each direction.

- Browse through all slices using the space bar and the backspace key. Translate and rotate some slices in an arbitrary way.
- Translate all slices at once by moving the mouse with the left mouse button and the Shift key pressed down.
- Rotate all slices at once by moving the mouse with the left mouse button and the Shift and Ctrl key pressed down.

Transforming all slices at once can be useful in order to move the region of interest into the center of the image.

Medivis Exhibit 1023

2.9.2 Alignment Via Landmarks

Besides manual alignment, four automatic alignment options are supported, namely alignment using a principal axes transformation, automatic optimization of a quality function, edge detection-based alignment and alignment via user-defined landmarks. The principal axes method and the edge detection method are only suitable for images showing an object which clearly separates from the background. The optimization method requires that the images be already roughly aligned. Often such a pre-alignment can be achieved using the landmarks method.
Alignment via landmarks first requires you to interactively define the positions of the landmarks. This can be done in *landmark edit* mode.

- Activate *landmark edit* mode by pressing the arrow-shape tool button located between the hand-shape button and the edge detection button.

In *landmark edit* mode only one slice is displayed instead of two. Two default landmarks are defined in every slice.

- Click on one of the default landmarks. The landmark gets selected and is drawn with a red border.
- Click somewhere into the image in order to reposition the selected landmark.
- Click somewhere into the image while no landmark is selected. This causes the next landmark to be selected automatically.
- Click at the same position again in order to reposition the next landmark.

The double-click method makes it very easy to define landmark positions. Of course, additional landmarks can be defined as well. Landmarks can also be deleted, but the minimum number of landmarks is two.

- Choose *Add* from the *Landmarks* menu.
- Click anywhere into the image in order to define the position of the new landmark.
- Select the yellow landmark by clicking on it.
- Choose *Remove* from the *Landmarks* menu in order to delete the selected landmark again.

Two landmarks should be visible now, a red one, and a blue one. Next, let us move these landmarks to some reasonable positions so that we can perform an alignment.

- Select slice number 0.

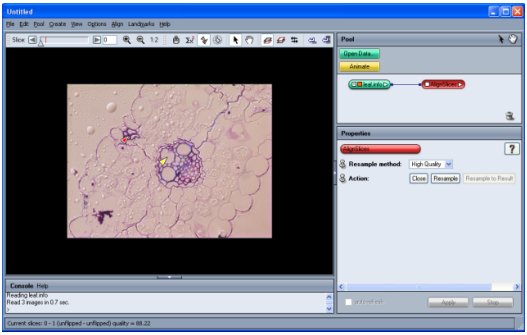


Figure 2.26: The figure shows how the landmarks should be set in the tutorial.

- Place the landmarks as shown in Figure 2.26. Make use of the double-click method.
- In all other slices place the landmarks at the same positions.

Once all landmarks have been set, we can align the slices. It is possible to align only the current pair of slices, or to align all slices at once. Note that all alignment actions as well as landmark movements can be undone by pressing Ctrl-Z.

- Switch back to *transform* mode by pressing the hand-shape tool button. Two slices should be displayed again.
- Align the current pair of slices by pressing the second tool button from the right (the one with only two lines).
- Align all slices by pressing the first tool button from the right (the one with many lines).
- Move and rotate the whole object into the center of the image using the mouse with the Shift key hold down.

In most slices the alignment now should be quite good. However, looking at the pairs 3-4 and 4-5 (displayed in the lower left corner of the align window) you'll notice that there is something wrong. In fact, slice number 4 has been accidently inverted when taking the microscopic images. Fortunately, we can compensate for this error as follows:

- Select slice pair 3-4 and make sure that the upper slice, i.e., slice number 4,

Medivis Exhibit 1023

is editable.

- Invert the upper slice by pressing the invert button (third one from the right).
- Realign the current pair of slices by pressing the second button from the right).
- Select slice pair 4-5 and realign this pair of slices as well.

Alternatively, you could have aligned all slices from scratch by pressing the first button from the right.

2.9.3 Optimizing the Quality Function

Once all slices are roughly aligned, we can further improve the alignment using the automatic optimization method. At the bottom of the align window the quality of the current alignment is displayed. This is a number between 0 and 100, where 100 indicates a perfect match. The quality function is computed from the squared gray value differences of the two slices. The optimization method tries to maximize the quality function. Since only local maxima are found, it is required that the slices be reasonably well aligned in advance.

- Click on the slice in the viewer. The quality of the alignment is displayed in the status bar at the bottom of the window. Remember the current quality measure.
- Activate the optimization mode by pressing the tool button with the sum x squared symbol.
- Align the current pair of slices by pressing the second button from the right. Observe how the quality is improved.

Automatic alignment is an iterative process. It may take quite a long time depending on the resolution of the images and of the quality of the pre-alignment. You can interrupt automatic alignment at any time using the *Stop* button.

- Automatically align all slices by pressing the first button from the right.

2.9.4 Resampling the Input Data

If you are satisfied with the alignment, you can resample the input data set in order to create a new aligned 3D image. This is done using the *Resample* button of the *AlignSlices* module.

Medivis Exhibit 1023

- Press the *Resample* button of the *AlignSlices* module.
- Attach an *OrthoSlice* module to the resulting object *leaf.align* and verify that the slices are aligned.

Sometimes you may want to improve an alignment later on. In this case it is a bad idea to align the resampled data set a second time, since this would require a second resampling operation. Instead, you could write the transformation data into the original image object and store this object in *AmiraMesh* format. After reloading the *AmiraMesh* file you can attach a new *AlignSlices* module and continue with the stored transformations.

- Choose *Save transformation* from the *Options* menu of the align tool. This will store the transformation data in the parameter section of the input object *leaf.info*.
- Delete the *AlignSlices* module.
- Save *leaf.info* in *AmiraMesh* format.
- Reload the saved object *leaf.am*.
- Attach a new *AlignSlices* module to *leaf.am* and click the *Edit* button. Note that the original alignment is restored.

2.9.5 Using a Reference Image

In some cases you might want to correct the alignment after image segmentation has been performed. In order to avoid segmenting the newly resampled image from scratch, you can apply the same transformations to a label field using a reference image.

- Delete any existing align tool.
- Load the file `data/align/leaf-unaligned.labels`.
- Attach a new *AlignSlices* module to the label field.

In the label field the guard cells of the stomatal pore are marked. Segmentation has been performed before the images were aligned. Now we want to apply the same transformation defined for the image data to the labels.

- Connect the *Reference* port of *AlignSlices* to *leaf.am* (this is done by activating the popup menu over the small white square at the left side of the *leaf.am* icon). Observe how the transformations are applied to the label field.
- Export an aligned label field by pressing the *Resample* button.

Medivis Exhibit 1023

The image volume used in this tutorial is an RGBA color field. However, the image segmentation editor only supports gray level images. Therefore you must convert the color field into a scalar field using *CastField* before you can invoke the image segmentation editor for the resampled labels.

2.10 Visualization of Vector Fields

This step-by-step-tutorial briefly explains some Mesh Option modules for vector field visualization. The use of these modules is explained by visualizing data representing the blood flow inside the carotid artery obtained with a simple numerical simulation. The image data used to segment the carotid are courtesey of Dr. Sell, University of Erlangen, Germany.
The following topics will be covered in this tutorial:

1. How to simply represent of a vector field
2. Use of Illuminated Stream Lines to visualize the flow
3. Extend the visualization with animated Particle Plot

2.10.1 Simple Vector Representation

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

- Load the surface model of the carotids *bloodFlow.surf.am* from the tutorial directory `data/tutorials/vector`.
- Attach an *SurfaceView* module to this data object (see the *Surface reconstruction* tutorial).
- Load the data set called `bloodFlow.vector.am` from the tutorial directory `data/tutorials/vector`.
- Connect a *Vectors* module to the vector field *bloodFlow.vector.am*.

The vector field itself is stored in Amira's native *AmiraMesh* file format. The data represents the simulated flow inside the carotids computed on a regular grid. By selecting the green data icon *bloodFlow.vector.am*, you can find out in the Properties that the number of grid nodes in x,y,z-direction is 116 x 81 x 227.
Besides *SurfaceView*, two new objects are available now in the Pool, *Vectors* and *EmptyPlane*. The module *Vectors* specifies how the vectors are visualized and the module *EmptyPlane* identifies on which the plane.

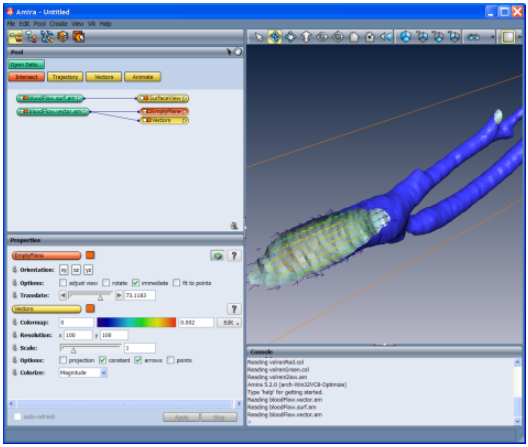


Figure 2.27: Surface of the carotis clipped and the inner vector field represented with simple vecotrs

- Set the *Resolution* of the *Vectors* module to 100 in X and Y
- Check in *Options* the toggles *constant* and keep *arrows* checked
- Right-click on the colorbar of the port *Colormap*, select the *physics* colormap and set the upper value to 0.002.
- In the module *EmptyPlane* click on the ”YZ” button and on the ”Clip” button on the upper-right corner of the Properties (beside to the question mark button).
- Rotate the object in the viewer and slide the slider of the port *Translate* of *EmptyPlane* to explore the vector field inside the carotid.

The vectors are represented as arrows equal in length, but varying in color. The color, taken from the colormap, indicates the vector’s magnitude. To use the length of the arrows to indicate the magnitude, deselect the toggle *constant* in the *Options* port.

Medivis Exhibit 1023

2.10.2 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 blood flow vector field.

- Remove the *Vectors* module.
- Connect a *DisplayISL* module to the vector field *bloodFlow.vector.am* by right-clicking on it and selecting *Display-¿Vectors*.
- Set *Num Lines* to 300.
- Press the *Apply* button.

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

- Switch the viewer into interact mode.
- Try out what happens if you click with the left mouse button on one of the green tabs on the corners or edges of the *TabBox* and drag them around.
- To move the whole *TabBox*, click with the left mouse button in the box and move it.
- Try to get the *TabBox* into a shape and position as shown in the image.
- Press *Apply* button again in order to recalculate the stream lines.
- As we did before, slide the slider of the port *Translate* of *EmptyPlane* to explore the vector field inside the carotid.

You can colorize of the stream lines according to a scalar data set of the same size and even animate the lines.

- Load *bloodFlow.density.am* from the tutorial directory `data/tutorials/vector`. This data set represents the 3D scalar field of the simulated fluid densitiy associated with the 3D vector field.
- Click on the white little square of the *DisplayISL* icon, select Color Field and attach *bloodFlow.density.am*.
- The port *Colormap* is now available in the Properties window of *DisplayISL*. Right-click on the color bar and select *temperature.icol*, set the colormap

Medivis Exhibit 1023

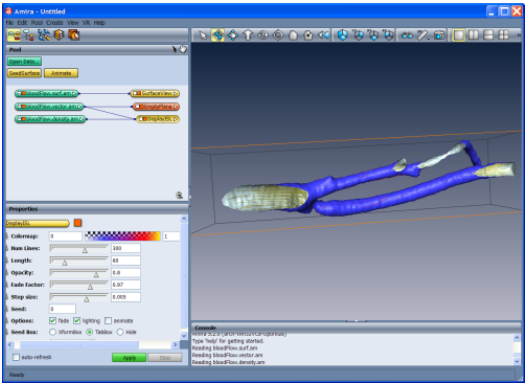


Figure 2.28: Illuminated blood flow stream lines with particle plot

interval to 0-1 and press *Apply*.

- To animate the stream lines just check the toggle *animate* in the port *Options*.

2.10.3 Animated Particle Plot

This module visualizes scalar vector fields by particles (cones) pointing in the direction of the local flow. The placement of particles can be done similar to the distribution modes of the DisplayISL module we have seen above. The particles are animated to generate a sequence of objects ”walking” through the vector field.

- Remove the *DisplayISL* module.
- Connect a *ParticlePlot* module to the vector field *bloodFlow.vector.am*.
- In the port *Option* and *Option2* set *NumCones* to 10, *Height* to 0.8 and *Radius* to 0.5.
- Right-click on the slider of the port Animate and select *Loop*. Click now the play button to start the animation.

Medivis Exhibit 1023

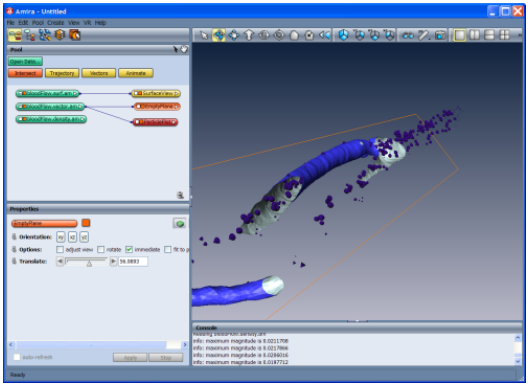


Figure 2.29: Blood flow visualized with particle plot and the reconstructed carotid surface

2.11 Creating animated demonstrations

In this tutorial you will learn how to use the *DemoMaker* module for creating an animated sequence of operations within Amira. In our example, we will visualize a polygon model using effects such as transparency, camera rotation, and clipping to make the visualization more meaningful and attractive.
The tutorial covers the following topics:

- creating an initial network for the demo
- animating an *OrthoSlice* module
- activating additional modules during the demo
- using a camera rotation or path
- editing or removing events that are already defined
- overlaying a bone model with a transparent skin model
- using clipping to make the skin appear gradually over the bone
- advanced clipping issues
- inserting breaks and defining demo segments
- using function keys for jumping between demo segments
- defining partial loops within the demo sequence

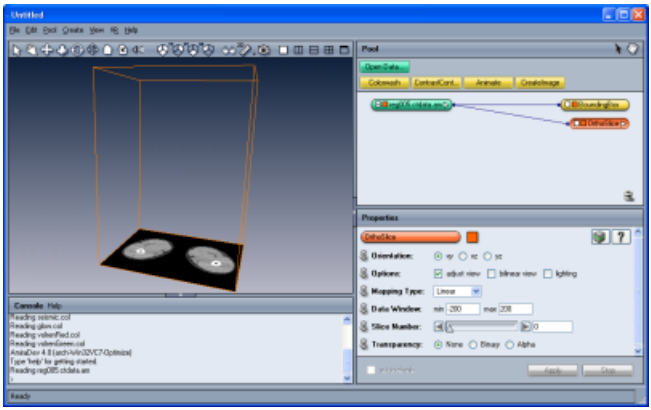
Medivis Exhibit 1023

- storing and replaying a demo sequence

Once you have learned how to define an animated demo sequence, you can further learn how to record the demonstration into a movie file in Section 2.12.

2.11.1 Creating a Network

First, we need an Amira network that contains all the data and modules for the visualization and animation we want to do. In our example, we pick the medical CT scan data set `reg005`. Start by loading `data/medical/reg005.ctdata.am` from the `AMIRA_ROOT` directory. By right-clicking on the green data icon and selecting from the data set's popup menu, attach a *BoundingBox* module as well as a *OrthoSlice* module to the data. If you use the mouse to navigate around the model in the 3D viewer, you should manage to get a result similar to this:

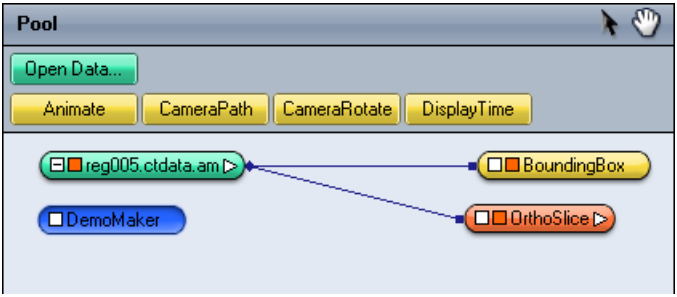


(load network)

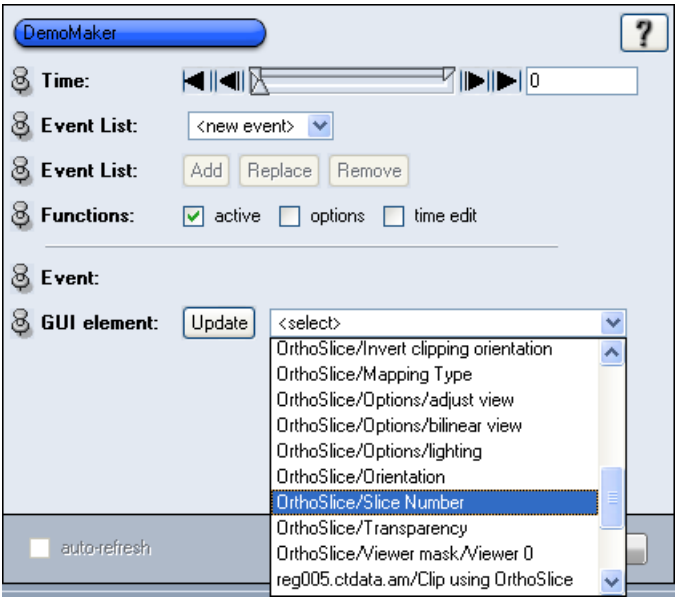
2.11.2 Animating an OrthoSlice Module

Let us move the *OrthoSlice* plane up and down to show what the data looks like. Note that the *OrthoSlice* module has a port called *Slice Number*. If you change the value of that slider, you see the plane move in the viewer.
Now let us animate this slider using the *DemoMaker* module as our first exercise. From the menu bar, select *Create/Animation/Demo/DemoMaker*. A blue *script object* appears in the Pool:

Medivis Exhibit 1023

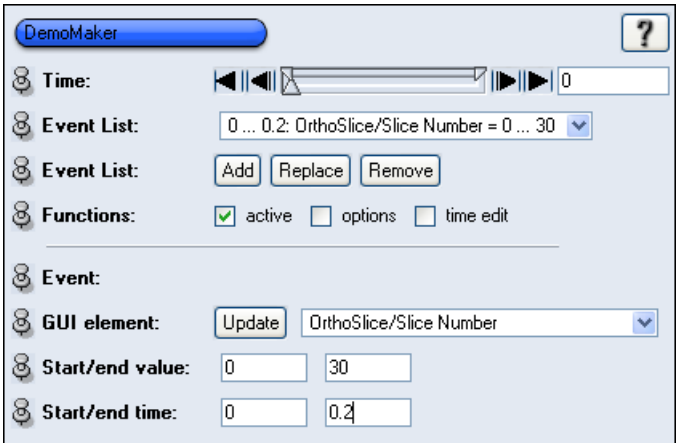


Click on the blue icon to see its user interface. Whenever you want to animate some port of the current network, you must select that port in the selection list called *GUI element*. Try to find the entry called *OrthoSlice/Slice Number*, which corresponds to the *Slice Number* port of the current *OrthoSlice* module. If you cannot find the entry, you may need to press the *Update* button to the left of the selection menu (see below for an explanation).



Once you have selected *OrthoSlice/Slice Number*, you see two more ports appear in the *DemoMaker* module: *Start/end value* as well as *Start/end time*. The start and

end value specify between which two values the OrthoSlice slider will be moved. Click the mouse into the start or end value fields, hold down the `Shift` key, and drag the mouse with `Shift` held down. With this feature called the *virtual slider* you can quickly set the desired value within the allowed range. Set the start value to 0 and the end value to 30. Then, set the start time to 0 and the end time to 0.2. These time values specify times on the *Time* slider of the *DemoMaker* module (first port in the module). You have now specified an *event* starting at time 0 and ending at time 0.2, varying the OrthoSlice slice number between 0 and 30 during that time.
Now press the *Add* button in the *Event List* port to add the newly defined event to the list of events. This list of events is represented in the selection menu above the *Add* button.



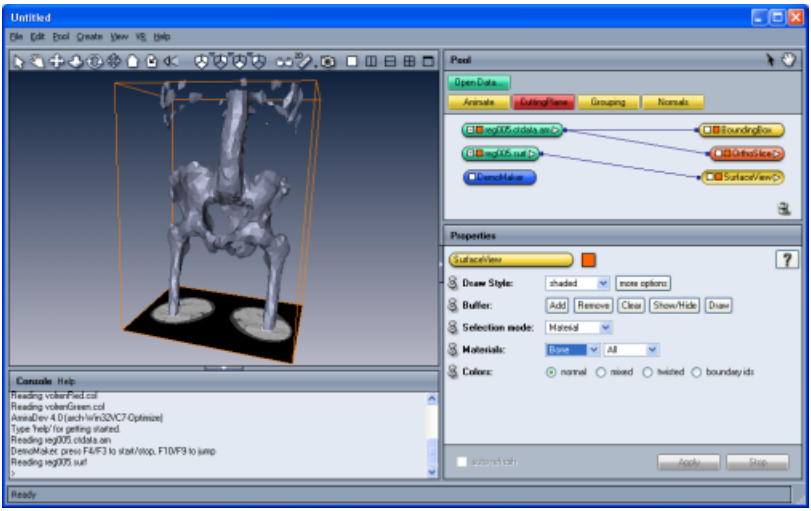
Test the result by pressing the play button of DemoMaker's time slider, represented by the triangle pointing to the right. When you press it, the time slider will start moving from the left to the right. When the time value is between 0 and 0.2, you should see that the OrthoSlice plane is moving between the specified start and end values in the viewer (load network). You can also play your demo backwards using the play button to the left of the time slider, or simply click somewhere on the time slider to jump to any point in time of the demonstration.
If the demo sequence runs too slow or too fast, you can adjust this by right-clicking anywhere on the time slider and selecting *Configure* from the popup menu. Change the *Increment* value in the dialog box that appears. A smaller increment will make the animation slower, whereas a larger increment makes it faster. If you choose an

Medivis Exhibit 1023

increment value too large, the animation might become ”jerky”.

2.11.3 Activating a Module in the Viewer Window

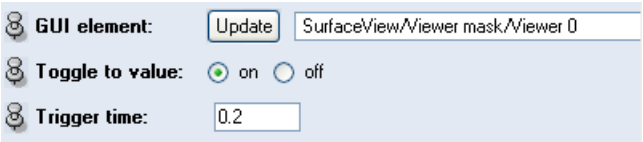
Next, let us add a visualization of the bone structure in the data set after we have moved the OrthoSlice. Load the data set `data/medical/reg005.surf` in addition to the current network. Attach a *SurfaceView* module to it. Click on the yellow *SurfaceView* module to see its user interface. Press the *Clear* button in the *Buffer* port, then select *Bone* and *All* in the *Materials* port and press the *Add* button in the buffer port. This will visualize the bone surface of the model.



If you want to switch the bone visualization on and off manually, you would use the *viewer toggle* (orange rectangle) of the *SurfaceView* module. If you want to include this action in your demo sequence, you need to do the following:

- Click on the *DemoMaker* module.
- Click on the *Update* button in the *GUI element* port. This is necessary whenever you add new modules to the Pool after you have created the *DemoMaker* module.
- Select *SurfaceView/Viewer Mask/Viewer 0* from the *GUI element* list.
- Enter *on* in the *Toggle to value* port.
- Enter *0.2* in the *Trigger time* port.

- Press the *Add* button in the *Event List* port to add this newly defined event to the event list.

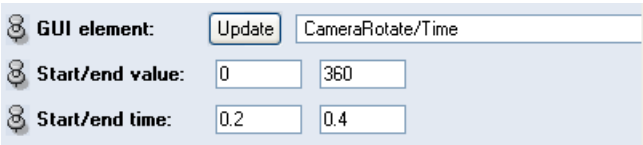


To test the newly added event, first toggle the *SurfaceView* module off using its orange viewer toggle icon. Now click to the very left of the *DemoMaker* time slider to jump to the start of the demo sequence. Click the play button. As before, the slice moves up. When it reaches the maximum value at time 0.2, the bone model is switched on (load network).

2.11.4 Using a Camera Rotation

To look at the 3D patient model from all sides, let us add a camera rotation to our demo sequence. Select *Create/CameraRotate* from the menu. Try the rotation by playing the time slider in the *CameraRotate* module. If you do not like the axis of rotation, reset the time slider to 0, navigate to a good starting view in the viewer window, and click on *recompute* in the *CameraRotate* module. Note that the values of the *CameraRotate* time slider range from 0 to 360.
Once you are satisfied with the camera rotation, add it to the event list:

- Click on the *DemoMaker* module.
- Click on the *Update* button.
- Select *CameraRotate / Time* from the *GUI element* list.
- Enter 0 and 360 as the start and end value.
- Enter 0.2 and 0.4 as the start and end time.
- Click on the *Add* button in the *Event List* port.



Now play the demo to see the result. After moving the slice and switching on the bone model, the view is rotated so that the bone can be seen from all sides (load network).

Medivis Exhibit 1023

2.11.5 Editing or Removing an Already Defined Event

When you look at the demo sequence so far, you may think that it would be nice to wait for a short time before rotating the bone model. This can be done by starting the rotation at a later time step. We can easily correct this in the *DemoMaker* module:

- Select the *0.2 ... 0.4: CameraRotate* event in the *EventList* port.
- You see the start/end value and start/end time of this event appear in the lower part of the *DemoMaker* module.
- Change the start/end time to 0.3 and 0.5.
- Click on the *Replace* button in the *Event List* port. This replaces the currently selected event in the list by the event as defined in the lower part of the module.

Now you have moved the camera rotation event from 0.2-0.4 to 0.3-0.5 on the time line. Check the results by playing the time slider (load network).
Please note that you can delete an event from the list by simply selecting it from the *Event List* menu and clicking the *Remove* button.

2.11.6 Overlaying the Bone with Skin

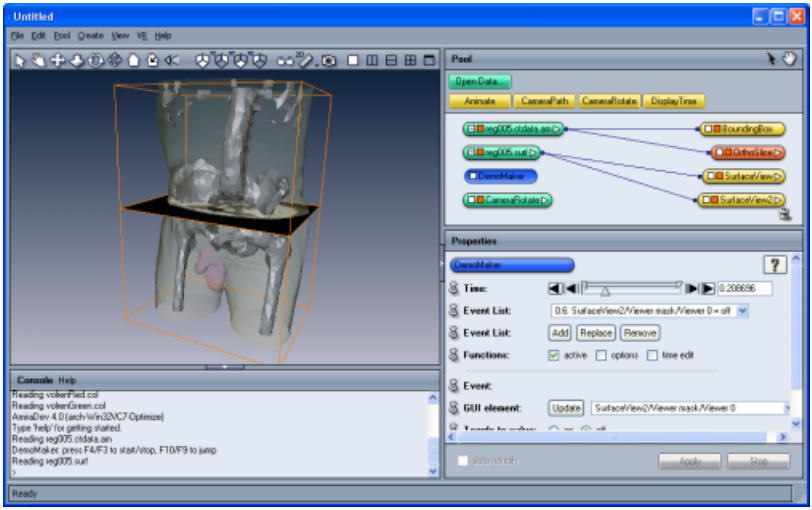
Now we want to show the patient's outer surface overlayed over the bone model.

- Attach a second *SurfaceView* module to the *reg005.surf* data set. Since *Exterior* and *All* are selected as the default materials, this brings up the patient's exterior surface.
- Click on the second *SurfaceView* module. It should be called *SurfaceView2*.
- Select *transparent* from the *Draw Style* port.
- It will be helpful to show the bone underneath the exterior surface, so jump to time step 0.2 or later in the DemoMaker module.
- Adjust the grade of transparency using the *BaseTrans* slider in *SurfaceView2*.
- Smooth out the outer surface by clicking on *more options* in the *Draw Style* port and selecting *Vertex normals*.

Like we did with the bone model, we can switch on the skin model at some point in the demo sequence:

- Click on the *DemoMaker* module.

- Click on the *Update* button in the *GUI element* port.
- Select *SurfaceView2/Viewer Mask/Viewer 0* from the *GUI element* list.
- Check *on* in the *Toggle to value* port.
- Enter *0.6* in the *Trigger time* port.
- Click on *Add* in the *Event List* port.



Again, check out the results by playing the demo sequence.

2.11.7 Using Clipping to Add the Skin Gradually

Instead of just switching the skin on at one point, we can make it appear gradually over the bone from bottom to top. In order to do so, we use the *OrthoSlice* plane to *clip* the skin model, and then move the *OrthoSlice* plane up.
First, we need to move the *OrthoSlice* plane down again to where we want to start the clipping:

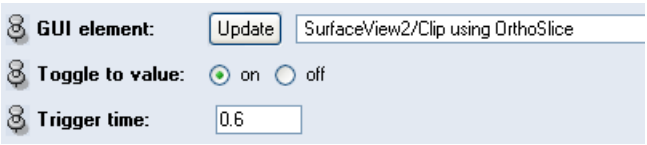
- Click on the *DemoMaker* module.
- Select *OrthoSlice/Slice Number* from the *GUI element* list.
- Enter 30 and 3 as the start/end values.
- Enter 0.3 and 0.5 as the start/end time.
- Click on *Add* in the *Event List* port.

Medivis Exhibit 1023

Now, when you play the demo, the *OrthoSlice* plane will move down again during the camera rotation (load network).

Now, we will clip the skin model using the OrthoSlice plane:

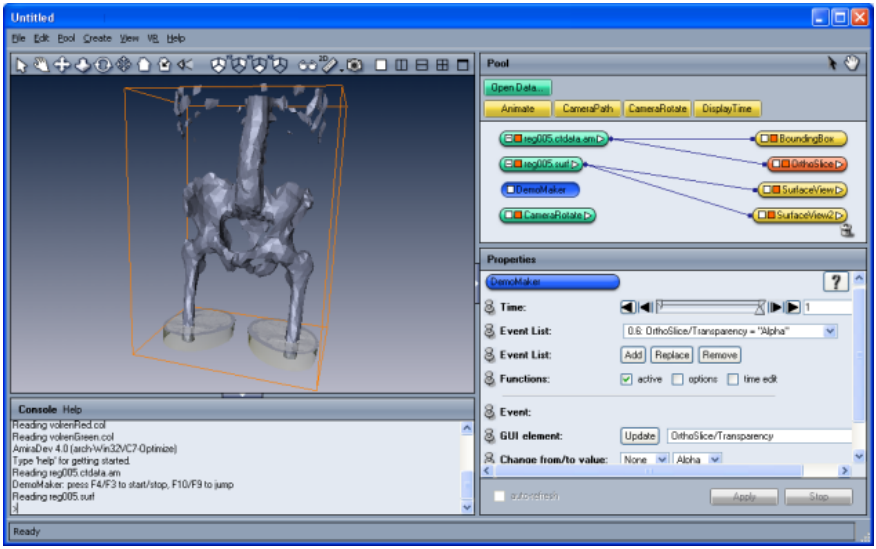
- Click on the *DemoMaker* module.
- Select *SurfaceView2 / Clip using OrthoSlice* from the *GUI element* list.
- Enter *on* as toggle value and *0.6* as trigger time.
- Click on *Add* in the *Event List* port.



When you run the animation now, you will not see the skin surface. This is because it is clipped above the *OrthoSlice* plane, and only visible below that plane. To see the partial surface below the plane, we must make the *Orthoslice* display transparent:

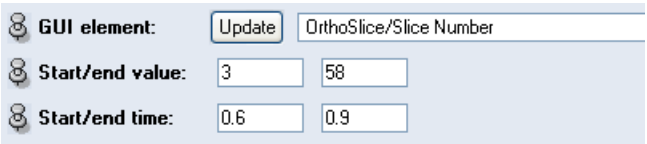
- Click on the *DemoMaker* module.
- Select *OrthoSlice / Transparency* from the *GUI element* list.
- Select *None* and *Alpha* as the from/to values.
- Enter *0.6* as the trigger time.
- Click on *Add* in the *Event List* port.

This way we have specified that at time 0.6, the *Transparency* port of the *OrthoSlice* module will be changed from value *None* to the new value *Alpha*. When running the demo sequence, the result should look like this:



As you see, part of the skin model is showing below the transparent *OrthoSlice* plane. To show all of the skin, we simply move the plane upwards pretty much the same way we did before:

- Click on the *DemoMaker* module.
- Select *OrthoSlice / Slice Number* from the *GUI element* list.
- Enter *3* and *58* as the start/end value.
- Enter *0.6* and *0.9* as the start/end time.
- Click on *Add* in the *Event List* port.

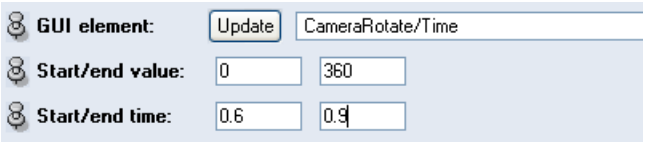


Now you see the skin slowly appearing over the bone as the clipping plane moves upwards.
As a last step, you might want to rotate the view again while the skin is appearing. You can simply reuse the old camera rotation during a second time range:

- Click on the *DemoMaker* module.

Medivis Exhibit 1023

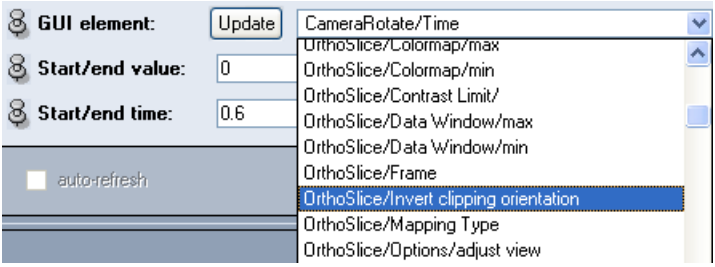
- Select *0.3 ... 0.5: CameraRotate* from the *Event List* menu.
- You will see start/end value and start/end time appear in the lower part.
- Change the start/end time to 0.6 and 0.9.
- Click on *Add* in the *Event List* port. This will leave the old event untouched and add a second camera rotation event to the list.



You can check out the final animation by loading a saved network script.

2.11.8 More Comments on Clipping

Clipping can sometimes be a little bit more complicated than in our example, because clipping can be applied to a plane in two different orientations. This means that you can either clip away everything *above* the plane, or *below* the plane. Unfortunately it is not always obvious which of the two cases you are in.
However, you can simply invert the orientation of the clipping in *DemoMaker*. In our example, you would simply select *OrthoSlice / Invert clipping orientation* from the *GUI element* port and add that event at the very beginning of your demo sequence (e.g., at some time before the clipping takes effect).

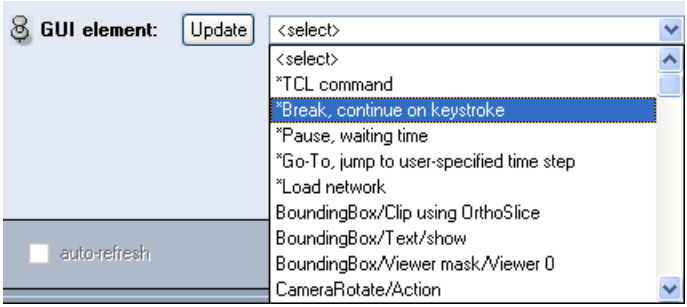


You do not need to use an *OrthoSlice* module to do clipping. As you have seen, the *OrthoSlice* might occlude parts of what you want to show. In that case, it is better to create an empty *ClippingPlane* module by selecting *Create/Clipping Plane* from the menu. Attach the module to the data set you want to clip (e.g., to *reg005.surf* in our example), and then use the *ClippingPlane* for clipping just as you used the *OrthoSlice* before.

2.11.9 Breaks and Function Keys

The demo sequence that we have created in this tutorial automatically runs through the complete time range that we defined. Sometimes it might be desirable to split the sequence into several segments, so that the demo will stop at some point and can be continued whenever the user desires to do so.
To take this into account, you can insert *breaks* in the *DemoMaker* event list. Let us insert one such break right after the bone model appears:

- Click on the *DemoMaker* module.
- Select **Break, continue on keystroke* from the *GUI element* list.
- Enter 0.21 as the trigger time.
- Click on *Add* in the *Event List* port.

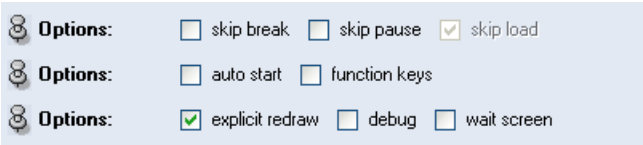


This way the demo will stop at time 0.21, which is right after the time when the bone model is switched on (0.2). When you play the demo from the start, you will notice that after the bone is switched on, the demo will stop.
Let us insert a second break at time step 0.51, which is right before the skin is starting to show. Proceed as above, using a trigger time of 0.51 instead of 0.21 (load network).
If you run the demo from the very beginning, it will stop after the bone is displayed, and you can read a message in the console window telling you that *DemoMaker* just stopped and you may press `F4` to continue. Try this by pressing the function key `F4`. The demo continues.
Likewise, the demo will stop just before showing the skin. Again, you can continue the demo by pressing `F4`. In general, at any point while the demo is running, you can press the `F3` key to stop it manually. Pressing `F4` will continue from the point where the demo stopped.

Medivis Exhibit 1023

If you have defined breaks as we did above, there are two more interesting function keys that in some sense allow you to *navigate* through the demo segments: pressing `F9` will jump back to the previous break or to the very beginning of the demo, and `F10` will jump to the next break, or to the very end of the demo. If you use `F9/F10` when the demo is stopped, it will just jump, and you need to press `F4` to start playing it from the new time step. If you press `F9/F10` while the demo is running, it will just jump to the new time step and continue running.

Please note that you can disable the breaks by checking the *skip break* toggle in the *Options* port of the *DemoMaker* module. You may even disable the definition of function keys by checking the *options* toggle in the *Functions* port, and then unchecking *function keys* in the second *Options* port. This is especially important if you want to use multiple *DemoMaker* modules, since only one of the modules can define the keys.

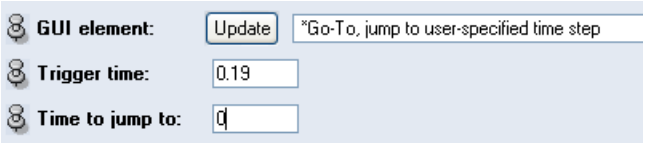


2.11.10 Loops and Go-to

One more feature that might be required for certain kinds of demos is the definition of loops. If you just want the whole demo to run in a loop, you can do this easily using the built-in features of the *time slider*: right-click on the slider and select *loop* or *swing*. Now if you play the time slider, it will start over from the beginning (loop mode), or play forwards, backwards, forwards... (swing mode).

However, you may want to define some part of the demo to run in a loop, and then stop the loop and continue with the demo upon key press. You can easily do this with the *go-to* feature of *DemoMaker*:

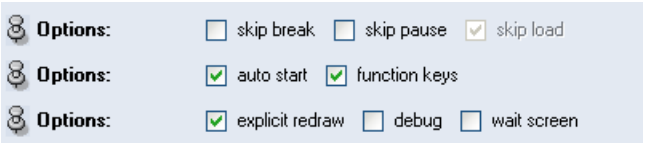
- Click on the *DemoMaker* module.
- Select **Go-to, jump to user-specified time step* from the *GUI element* list.
- Enter 0.19 as the *Trigger time*.
- Enter 0.0 as the *Time to jump to*.
- Click on *Add* in the *Event List* port.



When you run the demo sequence now, it will loop in the first segment, only showing the *OrthoSlice* move up, jump down, move up again... You can stop this by clicking on the stop button of the *DemoMaker* time slider or by pressing `F3`. To continue after the loop, you need to jump to the next segment by pressing `F10`, and then start playing again by pressing `F4`.

2.11.11 Storing and Replaying the Animation Sequence

As you may have noticed by now, storing a demo sequence once you have defined it is quite easy: simply save the whole Amira network by selecting *File / Save Network...* from the menu. The *DemoMaker* module will be saved along with the network, and so will the demo sequence you have defined.
When you load the network back into Amira, the state of the network will be the same as it was when you saved it. This means that you should be careful to reset the *DemoMaker* time slider to 0 before saving the network, if you want the demo to start from the beginning.
After loading the network, you can start the demo by clicking on the play button of the *DemoMaker* module, or by pressing `F4`. If you want to run the demo automatically right after the network is loaded, you can use the *auto start* feature that you find when you check *options* in the *Functions* port:



Just check the *auto start* toggle and save the network. When you load it again, the demo will start running automatically (load network).

2.12 Creating movie files

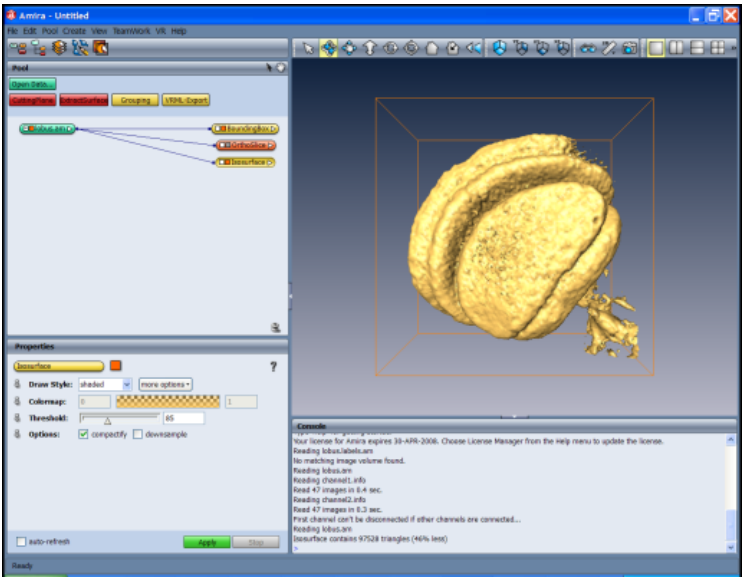
In this tutorial you will learn how to record a self-created animated sequence into a movie file using the *MovieMaker* module.

Medivis Exhibit 1023

In our first example we will just use a camera path to animate the scene, whereas in our second example we will rely on the demo sequence created in Section 2.11.

2.12.1 Attaching MovieMaker to a Camera Path

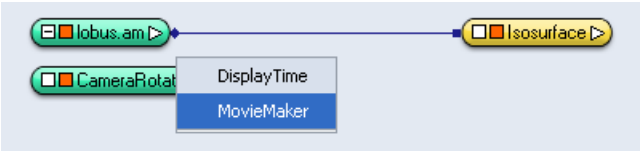
If you have created a visualization of your data and want to create a movie showing this visualization from all sides or from certain interesting viewpoints, you can create an appropriate camera path and record a movie by following the camera along that path.
Let us create a simple example. Load the `lobus.am` data set from the `tutorial` subdirectory and attach an *Isosurface* module to it. Choose an isosurface threshold of 70 and press the *Apply* button. The result should look similar to this:



The easiest way to create a simple camera path is to use the *CameraRotate* module. Select *Create/CameraRotate* from the menu, and press the play button of the newly created module. You can watch the scene rotate in the viewer while the time slider is playing (load network).
To record an animated scene into a movie file, you need to attach a *MovieMaker* module to a module that possesses a *time slider port*. The movie is recorded by

going through the individual time steps and taking snapshots of the viewer along the way.

In our example, the *CameraRotate* module has a time slider, so we can attach a *MovieMaker* module to it by right-clicking on the *CameraRotate* icon in the Pool and selecting *MovieMaker* from the popup menu:

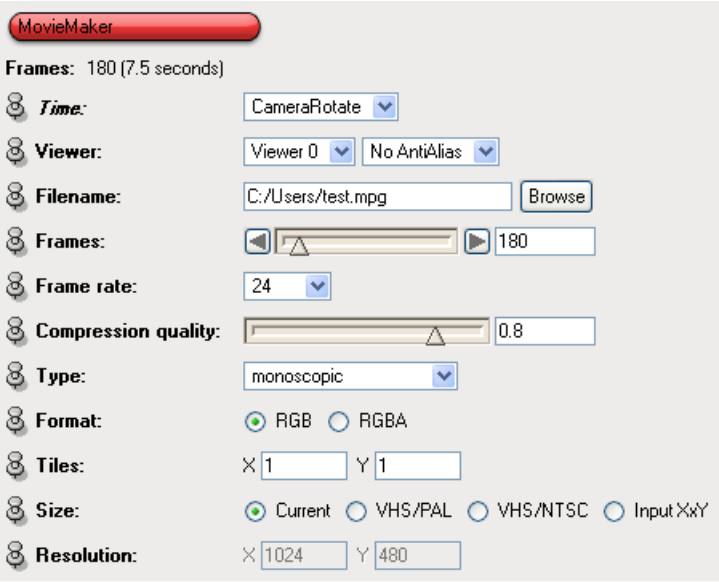


In the *MovieMaker* module, first click on the *Browse* button in the *Filename* port and enter a movie file name like `c:/tmp/test.mpg`. The `.mpg` suffix suggests that the movie file format will be MPEG, which is a widely accepted standard format for digital movies achieving a good compression ratio.

Next, adjust the parameters of the *MovieMaker* module to your liking, e.g., change the *number of frames*, the *image size*, or the *compression quality*. Please refer to the *MovieMaker documentation* for details.

In our example, let us choose 180 frames and leave all other parameters untouched. Since the *CameraRotate* module does a full rotation of 360 degrees, each of the 180 frames will represent a rotation of two degrees with respect to the previous frame. Press the *Create Movie* button to start recording.

Medivis Exhibit 1023



Wait for some time while the *MovieMaker* module drives the *CameraRotate* module and accumulates the snapshots. *Please note that the speed during the recording process is different than the playback speed of the movie.* Now view the resulting movie file `test.mpg` with a movie player of your choice (e.g., Windows Media Player or a similar tool). Experiment with the recording parameters until you get the desired result (e.g., control the file size and image quality by changing the *Compression quality* value, choose different image sizes to see up to which image size your computer is capable of smoothly displaying the movie, and change the number of frames to control the speed of the rotation).

2.12.2 Attaching MovieMaker to DemoMaker

Now we try to record a movie of a more complex animated scene. To this end, we load one of the networks that we have created in in Section 2.11: load network.

As you might remember, the basic idea of the *DemoMaker* module was that you define a set of events to be executed on a certain time line. Check this out by clicking the play button of the time slider in the *DemoMaker* module. You should see a nicely animated demonstration.

If you remember the previous section in this tutorial, you might already have an idea of how we can record this animated demonstration into a movie file. Like the *CameraRotate* module in the first example, the *DemoMaker* module is controlled via a *time slider* that we can attach to. So simply right-click on the *DemoMaker* icon in the Pool and attach a *MovieMaker* module. Like before, enter a movie file name and select the number of frames before you click on the *Create Movie* button to start recording.

2.13 Using MATLAB Scripts

In this tutorial you will learn how to integrate complex calculus into Amira using MATLAB (The MathWorks, Inc) by the means of the *CalculusMatlab* module.
In order to use the CalculusMatlab module, MATLAB 7 must be correctly installed on your computer. The CalculusMatlab module establishes a connection to the MATLAB computational engine that was registered during installation. If you did not register during installation, on the Windows command line you can enter the command:

- matlab /regserver

The limitations of the CalculusMatlab module are listed in its *documentation*.
This tutorial covers the following topics:

1. Loading and executing a MATLAB script.
2. Lowpass filtering on images.
3. Controlling the script with a time slider.
4. Thresholding on a volume.

2.13.1 Lowpass Filtering on Images

In this section we will learn how to apply a lowpass filter on an Amira image using the MATLAB Fourier transformation. This example shows how to pass data and control variables from Amira to MATLAB, execute a MATLAB script, and import the data back into Amira.

- Load the *lena.png* image file located in subdirectory *data/tutorials/matlab*.
- Choose *Luminance* in the *Channel Conversion field* as shown in Figure 2.30.
- Right click on the green icon and choose CalculusMatlab from the Compute section.

Medivis Exhibit 1023

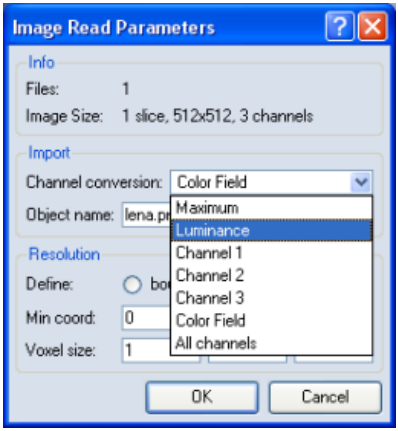


Figure 2.30: Loading the image

A new red icon appears, the CalculusMatlab module that will try to connect to the MATLAB engine. This may take a while.

- Load the script *lowpass.m* located in subdirectory *data/tutorials/matlab* by clicking the *Read* button of the *File* port.
- Execute the script by clicking on the *buffer* button of the *Execute* port.
- Connect an *OrthoSlice* to the filtered image *result*.

The module uses the MATLAB computation engine which has its own user interface.
You can easily show or hide the MATLAB console using the checkbox in the options field. The MATLAB console is very useful for debugging purposes because it allows you to access variables of the MATLAB workspace. Any variable not cleared by the MATLAB "clear" command in the script is accessible in the MATLAB workspace, even after finishing the current CalculusMatlab computation (see the *CalculusMatlab* documentation).
In addition you can control scalar parameters of the script using time sliders:

- Create a time slider (*File/Create/Data/Time*).
- Connect the CalculusMatlab module to the time slider.
- Change the line cutoff=0.05 to cutoff=t in the script (see the *CalculusMatlab*

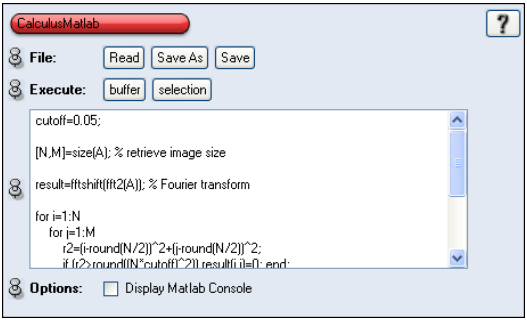


Figure 2.31: The CalculusMatlab module

Figure 2.32: Left: original Right: lowpass filtered

documentation for more information about the keyword t).

- Click on the time slider and adjust the value that will be assigned to w. Right mouse click in the text field of the time slider and select *Configure* to adjust the data range of the parameters.

Note: To handle RGBA image filtering, you must load the image with Color Field Channel Conversion and treat each channel separately in the script.

Medivis Exhibit 1023

2.13.2 Thresholding on a Volume

In this section we will learn how to apply a threshold to a volume. This is done by setting a value for a threshold. If the value for the voxel is less than the threshold, the voxel value is assigned the value of zero. If it is above the threshold, it is assigned a value of 255.

- Load the file *lobus.am* located in subdirectory *data/tutorials*.
- Right click on the green icon and choose *CalculusMatlab* from the *Compute* section.
- Load the script *threshold.m* located in subdirectory *data/tutorials/matlab* by clicking the *Read* button of the *File* port.
- Execute the script by clicking on the *buffer* button of the *Execute* port.

A new green icon appears, the Lattice that will hold the threshold result. Connect an *OrthoSlice* or a *Voltex* to see the result.

Note: Any variable accessed by MATLAB and pushed back into the Amira workspace will lose its voxel size information. You will need to correct the voxel size manually using the *Crop Editor*.

2.14 Introduction to the Filament Editor

This section provides a step-by-step introduction to the Filament Editor.

The Filament Editor is a special purpose sub-application in Amira that is designed to extract complex three-dimensional networks of filamentous structures from image data and post-process such data by editing and annotating the network with label scalar data. In this tutorial we want to demonstrate the principles of the Editor by extracting the dendritic fiber network of an invertebrate neuron. The data set we will be working with depicts a neuron from the honeybee brain imaged with a confocal microscope and was kindly provided by Prof. R. Menzel, Free University of Berlin.

In this section the following topics will be discussed:

1. Exploring the volume data.
2. Automatic extraction of the network.
3. Interactive tracing.
4. Labeling and visualizing the network.

Medivis Exhibit 1023

To access the Filament Editor, click the icon in the *sub-application bar*. We begin our work by loading the 3D image data set.

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

2.14.1 Exploration of the volume data

Once the image data has been loaded, the left-hand panel of the viewer shows a 2D slice of the volume, while the right-hand panel shows the 3D objects. In order to have a clear idea of how the neuron spreads out in space, we explore the volume by using slicing, windowing, and rendering the gray values.

- **Window Level** Select the *Window Level* icon, click the 2D viewer, hold the mouse button pressed, and drag the mouse cursor left/right to change the window width, or up/down to change the window center. Set the window level around 30-80. Click the 3D button to visualize the selected voxels using a shaded volume rendering.
- **Browse Slice** The browsing tool allows using the mouse to navigate through an image stack in the 2D viewer (or MPR viewer). Select the Browse Slice icon, click the 2D viewer, hold the mouse button pressed, and drag the mouse cursor up/down to scroll forward or backward through the image stack. If you are working with a wheel mouse you can use the wheel instead of this icon. Simply click the viewer and scroll the mouse wheel to scroll through the image stack.
- **Thick Slice** The thickness of the slice can be set by sliding the *Thickness* slider. When displaying a thick slice, data values are computed as maximum intensity of the values of the original slices. Set thickness to 25. Click the 3D slice button to visualize the thick slice in the 3D viewer.

2.14.2 Automatic extraction of the dendritic tree

In a first step you may want to automatically extract what will serve as a draft version of the neuron's skeleton. This can be achieved with the Filament Editor's *Auto Skeleton* tool.

- Set the *Window Level* to `37 137`.
- Click the 3D button on beside the *Window Level* slider. The 3D viewer shows a shaded volume rendering of the neuron's main branches and gives

Medivis Exhibit 1023

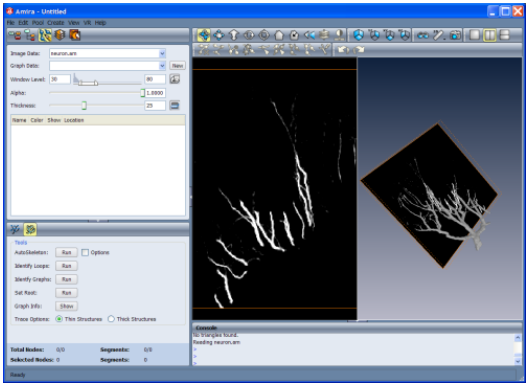


Figure 2.33: The Filament Editor immediately after the image data has been loaded. The left-hand panel of the viewer shows a 2D slice of the volume, while the right-hand panel shows the 3D objects. The 3D and slice rendering have been activated in the 3D viewer.

a rough estimate of what the automatic tracing procedure will consider as neuron.

- Select the *Tools* tab in the **Tool Box** and press the *Run* button of the Autoskeleton port. After a few seconds you will see green lines and blue points in the 2D viewer. In the 3D viewer gray spheres are displayed together with the preview rendering. If you do not see them, click the icon *View All* (or the space bar) to center the 3D viewer on them.
- Adjust the *Alpha* slider, which is appeared when you activated the 3D rendering. You can see the generated graph through the rendering.

The *Auto Skeleton* tool traces connected regions according to a user-defined window level and converts the centerline of those regions into graphs composed of points, segments, and nodes, which are the elements of the data class *SpatialGraph*. The result of the ”skeletonization” is, therefore, a *SpatialGraph* object, which is being visualized in the 3D viewer as balls and lines, and as blue points connected by green lines in the 2D viewer.

Depending on the quality of the data and on the selected window level the result of skeletonization will typically show several disconnected elements and as well

Medivis Exhibit 1023

as loops within the networks. Disconnected elements will break the single neuron into several sub-graphs. Loops, on the other hand, are parts of a graph where segments connect onto itself. In some biological objects like e.g. neurons loops must not occur. To identify graphs and loops the Filament Editor offers dedicated tools.

- Press the button *Identify Graphs* to automatically detect and label all graphs in the SpatialGraph object, that is all connected components. This creates a new label group in the Label Editor named *Identified_Graphs* under which all identified graphs are listed as *Graph1, Graph2, ..., GraphN*.
- To visually identify them just click on one of the items in the *Identified_Graphs* list. The selected item is highlighted in red in both viewers.
- Under *Identified_Graphs* select *Graph2* and shift-select *Graph25*, then remove them by clicking the *Delete Selection* icon.
- Press again the button *Identify Graphs*, now you should have only two graphs.
- Scale the nodes by moving the *Node Scaling Factor* slider in the *View Options* tab of the **Tool Box** (we will see the details in the next paragraph).

At this point we have only two graphs: Graph0 is the large tree, while Graph1 is just three segments and four nodes. Switch back to the *Tool* tab in the **Tool Box**.

- Select *Graph1* in the Label Editor under *Identified_Graphs*
- Activate the *Select Single* icon and Ctrl-click on the closest node of Graph0 in the 3D viewer.
- Select the *Connect Selected* icon to connect the graph.
- Press again *Identify Graphs*. At this point you should have only one graph.
- Press *Identify Loops* to get another label group *Identified_Loops*.
- Under *Identified_Loops* select *Loop2* and remove it by clicking the *Delete Selection* icon. Note that a segment is defined by two nodes. Therefore, when you remove a node the associated segment will be removed too. In this example this will not happen, but in general it is very likely that some connected segments are removed when a loop is deleted.

It should perhaps be noted that the *Auto Skeleton* tool is also available as a compute module in the Pool under the *Skeleton* category. Also, there is a rich set of tools available in the *Segmentation Editor* to perform the threshold segmentation necessary for the skeletonization process. Finally, in the case of a strict tree topology it

Medivis Exhibit 1023

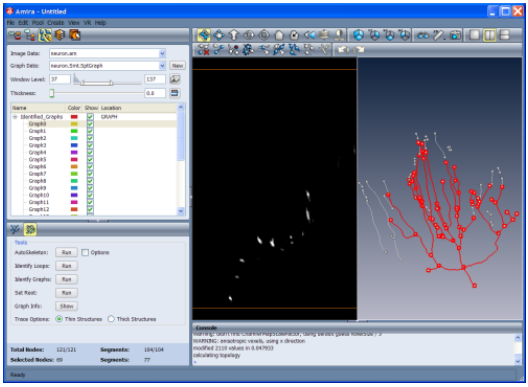


Figure 2.34: The SpatialGraph object obtained with the *Auto Skeleton* tool. Note the highlighted main graph.

may be advantageous to restrict the search to a tree. In this case you may want to use module *CenterlineTree* to extract a graph with guaranteed tree topology.
Before closing this subsection, we should save the *SpatialGraph* data object we have created, it will be useful in further step of this tutorial.

- Switch back to the Pool, save the object called *neuron.Smt.SptGraph* in a directory of your choice and then remove it from the Pool.
- Switch again to the Filament Editor to prepare the next step.

2.14.3 Filament Tracing

The *Interactive Tracer* tool enables you to trace the filaments directly on the gray values using an innovative tracing algorithm developed by Visage Imaging. The user sets the starting and ending points of a segment and the *Interactive Tracer* finds the shortest line connecting the two points with respect to the user-defined data window. Now we can use the *Interactive Tracer* to re-trace the neuron segments. In order to clearly understand how to use the tracer, we will try to re-trace a small part the of the neuron tree we extracted in the previous paragraph.

- Press the *New* button at the *Graph Data* port to crate a new *SpatialGraph* data object.

Medivis Exhibit 1023

- Click in the 2D viewer. Using the *Browse Slices* tool or the mouse wheel, slice through the volume until you reach the root of the neuron.
- Set the thickness of the *Thick Slice* at 10 for a better visualization.
- Activate the *Interactive Tracer* by clicking the *Trace Filament* icon in the toolbar and make sure that the *Thick structures* option is checked in the *Tools* tab.

Note that the *Interactive Tracer* is alway active while the *Trace Filament* icon is highlighted and it can be triggered only if the 2D viewer is active.
If you now move the mouse over the 2D slice, the cursor has a cross shape whenever it is over black regions and turns into a crosshair (cross within circle) when it is over voxels that are within the window level. The cross-hair shape indicates that it is possible to set tracing key points. As you will see later, when the pointer is over a traced segment (green line) a small "P" appears in the crosshair. This indicates that a click would select the nearest point as a key point. Likewise, an "N" is shown in the cursor whenever the cursor is over a node and a click will select this node as a key point. Furthermore, the cursor turns red when the trace tool is in **append mode**, meaning that the next click will trigger a tracing action between the previously selected and the current key point.

- Start the tracing by clicking on a gray value of the tree root.
- Slice ahead and set further points, until you reach the first bifurcation and click on it.
- Deactivate the *Interactive Tracer* e.g. by clicking the *Select Single* tool or by clicking again its icon.

If you press *Identify_Graphs* graphs, you will see only one graph. You can also access some statistical information by clicking on *Graph Info* button in the *Tools* tab. This opens a spreadsheet window containing a list of all segments in the graph. The spreadsheet is interactive, i.e. if you click on a line, the corresponding segment will be highlighted in the viewer.
Repeat the actions to trace the first branching sections from the root.

2.14.4 Visualize the network

The *SpatialGraph* data object is able not only to store the spatial structure, i.e., the geometry of the network, but also to associate it with scalar and label data.
Labels can also be used to tag or annotate sub-graphs according to their topology.

Medivis Exhibit 1023

In this simple example we want to tag the three main branches of the *Topology* as *Central*, *Left*, and *Right* od a neuron tree.
If you already extracted the graph as described in the second subsection you should now load the file you previously saved, otherwise you should go two steps back and read the subsection *"Automatic extraction of the dendritic tree"*.

- Right-click into the Label Editor and select *Add graph label group*.
- Right-click on the created label and rename it *Topology*.
- Right-click on the *Topology* label and select *Add label*. Repeat this action to add three labels.
- Select and right-click on each of the created labels and rename them *Central*, *Right*, and *Left*.
- Select the *Select Lasso* icon and select-lasso the right-hand branch. You can use the *Select Lasso* tool in combination with *Shift* key to deselect-lasso or with the *Ctrl* key to add elements.
- Right-click on label *Right* and select *Assign selection*.
- Repeat the last two items to label *Central* and *Left* branches. Note that this is just an exercise, it does not really matter to exactly select the "Center", "Right" or "Left" segments.

  Select now the *View Options* tab in the *Tool Box*
- Scale the nodes using the *Scaling Factor* slider.
- Colorize the nodes according to the label by selecting *Topology* in *Node Coloring*.
- Repeat the last item for the segments.
- Click on the color button of label *Right* and select a new color.

For advanced network visualization you can switch to the Pool or Tree View. If you do so, you may attach a *SpatialGraphView* module to the *SpatialGraph* object we created before.

2.14.5 Alternative way to extract a network using the Segmentation Editor

From the Filament Editor just switch to the *Segmentation Editor* by clicking the icon in the *Sub-application Tool Bar*. For an in-depth treatment of the Segmentation Editor we refer you to its documentation and tutorial. For this simple example you may try the following simple steps.

Medivis Exhibit 1023

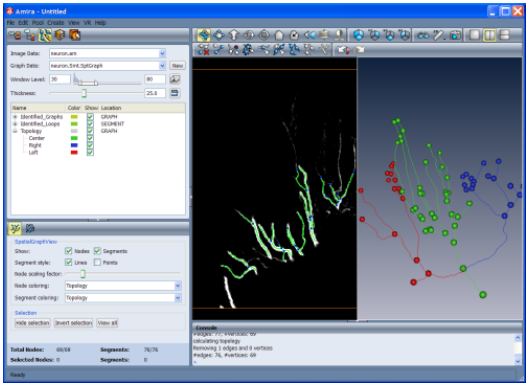


Figure 2.35: The visualization of the SpatialGraph obtained with the *AutoSkeleton* and Tracing tools.

- Create a new label field by pressing the *New* button in the *Label Data* port located in the upper part of the user-interface.
- Create a new material by pressing the *New* button in the *Materials* port.
- Right-click the new material in the *Material List*, select *Rename Material* and enter ”neuron”.
- In the area *Display and Masking* area set the threshold to 100-255
- In the tool box at the bottom part of the user-interface, select the *Threshold* tool and click the *Select* button.
- in the *Selection* box click the ”+” button to assign the selected region to material ”neuron”.
- Switch back to the Filament Editor and select *neuron.Labels* from the *Image Data* pull-down menu. Now skeletonize the label field by pressing the *Run* button in the *Auto Skeleton* tool.

2.15 Introduction to the Multi-planar Viewer

The Multi-planar Viewer can be used to visualize one or two data set at the same time using a Multi Planar Reconstruction (MPR) and a 3D viewer. Addition-

Medivis Exhibit 1023

ally, the image volumes can be registered automatically or manually using specific graphical interfaces.
In this tutorial we will try to illustrate how to:

1. Exploration of the volume data, including an associated label field
2. Explore two data sets in fusion mode
3. Register two data sets

Before we begin our work we need to define the terms *Primary* and *Overlay* in the context of the Multi-planar Viewer

1. **Primary** - Reference data set, this data set will be considered our principal data set to which a secondary data set could be eventually compared or registered. In the context of PET/CT fusion the CT data set would be used as reference data set as it provides the best anatomical reference.
2. **Overlay** - Secondary data set, which could be compared or registered to the Primary.

2.15.1 Exploration of the volume data

To activate the Multi-planar Viewer select its icon in the sub-application toolbar. On a first glance the Multi-planar Viewer looks similar to the Segmentation Editor: three slice views, a 3D viewer and a material list. In order to start exploring, load the Lobus data set located in the tutorial directory `data/tutorial/lobus.am`:

- **3D Volume Rendering** Select the 3D Rotate icon (it should have been selected by default as you start the Multi-planar Viewer) in the toolbar. Now rotate the object in the 3D viewer.
- **Volume Rendering Modes** Try to visualize the volume using VRT or MIP techniques and their optional rendering modes. To do so, switch between the radio buttons in the 3D Settings panel.
- **3D MPR** Click the icon in the 3D Settings panel to activate MPR in the 3D viewer. You should see now the volume rendered in 3D and the three MPR slices.
- **Window Level** Select the Window Level icon, click into one of the 2D viewers, hold the mouse button pressed, and drag the mouse cursor left/right to change the contrast (window width), or up/down to change the brightness (window center). Note how the bars in the graphical user interface (GUI)

Medivis Exhibit 1023

change according to the pointer movements. Click in the 3D viewer and do the same.

- **Browse Slice** The browsing tool allows using the mouse to navigate through an image stack in the 2D viewer. Select the Browse Slice icon, click in a 2D viewer, hold the mouse button pressed, and drag the mouse cursor up/down to scroll forward or backward through the image stack. If you are working with a wheel mouse you can use the wheel to access the browse slice functionality. Simply click once the viewer to activate it and roll the mouse wheel to scroll through the image stack.
- **Thick Slice** The thickness of the slice can be set by sliding the Thickness slider. When displaying a thick slice, data values are computed as the average, the minimum or the MIP of the values of the included slices, according to the selected option.
- **Label field** Load the associated label field (`lobus.label.am`) from the tutorial directory and select the checkbox of the Label Field port. Visualize the materials in 3D with constant color, hide or fade them. Note that the port Label field refers to the Primary data set; to be activated the loaded LabelField has to have the same size as the Primary.
- **Extend viewers** Double-click one viewer to extend it, double-click to set the configuration back to 4-window layout again.

Note that clicking the right button in the viewers activates the context menu, which provides a quicker way to access tools and settings.

2.15.2 Explore two data sets using fusion mode

The process of *image fusion* consists of combining relevant information from two or more images into a single image using different color mappings. The Image Fusion became very relevant especially in medical science, where different imaging modalities can be obtained from the same patient. Typically, the multimodal imaging includes magnetic resonance (MR), computed tomography (CT), and positron emission tomography (PET/SPECT). Image fusion can be very useful in microscopy imaging for alignment and registration of multi-channel, 3D, and 4D data sets.
When only one data set is available in the Pool, the Overlay Data port is not available. Load the Lobus2 data set (`data/tutorial/lobus2.am`) from the tutorial directory and the checkbox of the port will become active. Select the Overlay checkbox and Lobus2 will be shown in the available viewers. Change the min-

Medivis Exhibit 1023

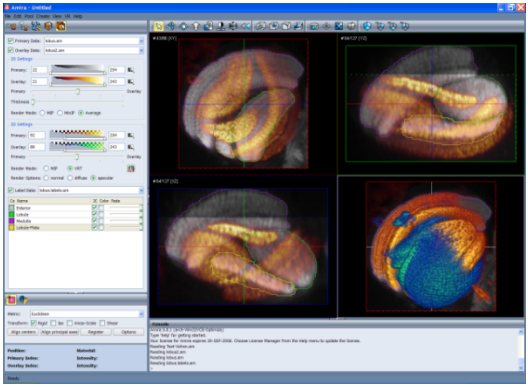


Figure 2.36: The Multi-planar Viewer immediately after the image data has been loaded.

imum and maximum values of the Overlay color map. To do so, select the 3D viewer to activate it, slide the colormap bar to the left and the right or change the upper and lower limits. You can do the same for the 2D viewers as well.
Select again the *Window Level* icon, keep the mouse button pressed and move the pointer: the window level changes only for the Primary data set. When Primary and Overlay images are loaded together, the Primary is the main object in the Multi-planar Viewer. The activated tools refer always to the Primary or to Primary and Overlay together, never to the Overlay individually.
You should be familiar now with the 2D and 3D visualization options. Therefore, experiment with changing the color maps or the blend of Primary and Overlay data sets.
To better explore internal structures of the two volumes you can use the 3D cropping and clipping tools offered by the Multi-planar Viewer.

- Set the 3D visualization mode to VRT with specular light effect.
- Click the Crop Corner icon. Browse the slides in 2D and rotate the volume in 3D to better see the cutting region.

The Crop Corner tool is very similar to the Corner Cut module available in the Pool. You can set the cutting boundaries directly by moving them in the 2D viewers. Experiment with the other cutting tools.

Medivis Exhibit 1023

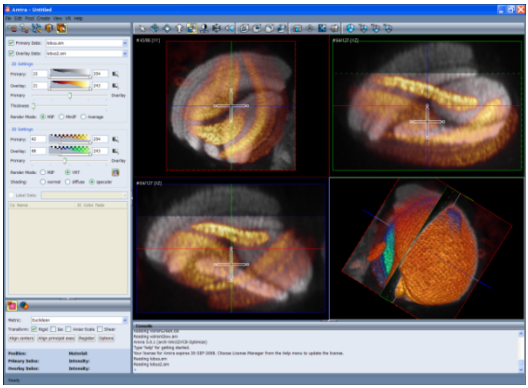


Figure 2.37: The Multi-planar Viewer with the Manual Registration tool activated.

2.15.3 Manually register two data sets

You might have noticed that as soon as Lobus2 was loaded, the icon Registration tool was activated. The registration tools work only when two data sets are loaded in the Multi-planar Viewer. Click now the Registration Tool icon and you will see an arrow-cross in the center of the 2D viewers of the Overlay (we assume that the Overlay is to be registered to the Primary). Move the pointer along the cross: the shape of the pointer changes: Arrow-cross translate Turning arrow - rotate Arrowhead - scale the image volume

You can also digitally transform the volume by using the Transform Editor options tab in the Registration Toolbox at the bottom of the user interface. This editor corresponds to the user interface of the "Absolute" tab of the *Transform Editor* dialog. More information is available in its help file.

- Activate the Registration tool by clicking its icon.
- Try to manually align Lobus2 to Lobus.
- Use the Transform Editor options in the Registration toolbox to digitally refine the registration.

Once the two volumes are roughly aligned, use the Automatic Registration tool in the Registration toolbox to refine the results. You can get more information about

Medivis Exhibit 1023

automatic registration by reading the Registration tutorial.

Medivis Exhibit 1023

3 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 2, particularly the very first one described in Section 2.1 (Getting Started).

3.1 Interface Components

In this section the following interface components are described:

- *File Menu*, *Edit Menu*, *Pool Menu*, *Explorer Menu*, *Create Menu*, *View Menu*, *Help Menu*
- *Main Window*, *Viewer Window*, *Console Window*
- *File Dialog*, *Job Dialog*, *Preferences Dialog*, *Snapshot Dialog*, *System Information*

3.1.1 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.

3.1.1.1 Open Data

The *Open Data* 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 4.1.
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.

3.1.1.2 Open Time Series Data

This button also activates the *file dialog*, but in contrast to the ordinary *Load* option it is assumed that all selected files represent different time steps of a single data object. When loading such a time series an instance of a *time series control* module is created. This module provides a time slider allowing you to adjust the current time step. Whenever a new time step is selected the corresponding data file is read, and data objects associated with a previous time step are replaced. The module also provides a cache so that the data files only need to be read once provided the cache is large enough.

3.1.1.3 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*.

3.1.1.4 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.

Medivis Exhibit 1023

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 Pool.

3.1.1.5 New Network

This button does a *Remove all objects* so that you can start building a new network in the Pool. It also sets the new network name to Untitled.hx.

3.1.1.6 Open Network

The *Open Network* button activates Amira's *file dialog* and lets you load a network stored in a file. Network files show up in the dialog as being of format "Amira Script". A *Remove all objects* will be done before the new network is loaded so that effectively the new network replaces the old network in the Pool. Currently only files with extension .hx can be opened.

3.1.1.7 Save Network

This button allows you to save the complete network of icons and connections shown in the Pool. If the network has not been previously saved, you will need to specify the name of an Amira network script in the file dialog. When executed, the network script restores all data objects and modules as well as the current object transformations and the camera settings. The feature is useful for resuming work at a point where it was left in a previous Amira run.
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 specify that all required data objects should be saved automatically in a separate subdirectory.
Instead of the option *Amira script* you can also choose *Amira script and data files (pack & go)* from the file dialog's format menu. In this case *all* data objects currently loaded will be saved in a separate directory. More options affecting the export of network scripts can be adjusted in the *Preferences* dialog.

Medivis Exhibit 1023

3.1.1.8 Save Network As

This button works like the *Save Network* button except that in this case you will always need to specify the name of an Amira network script in the file dialog.

3.1.1.9 Recent Files

This button can be used to load recently used files. When choosing this menu entry a submenu appears listing the five most recent files. If multiple 2D images have been loaded this is indicated with the name of the first file followed by three periods (...).

3.1.1.10 Recent Networks

This button can be used to load recently used network scripts. When choosing this menu entry a submenu appears listing the five most recent network scripts.

3.1.1.11 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.

3.1.1.12 Quit

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

3.1.2 Edit Menu

The *Edit* menu provides access to the standard cut/copy/paste/delete commands, as well as to an extended version of the Parameter Editor and the Amira preferences dialog.

3.1.2.1 Cut

In the *Console*, this command cuts selected text and copies it to the clipbord. In the *Pool*, this command removes the selected objects.

Medivis Exhibit 1023

3.1.2.2 Copy

In the *Console*, this command copies the selected text to the clipboard. In the *Pool*, this command has no effect.

3.1.2.3 Paste

In the *Console*, this command pastes the text in the clipboard to the current text insertion point. In the *Pool*, this command has no effect.

3.1.2.4 Delete

In the *Console*, this command deletes the selected text. In the *Pool*, this command deletes the selected objects.

3.1.2.5 Select All

In the *Pool*, this command select all objects.

3.1.2.6 Database

The *Database* button activates an extended version of the *Parameter Editor*, allowing you to manipulate Amira's global parameter database. Among others, the parameter database contains a set of predefined materials (to be used for image segmentation and surface reconstruction) and of predefined boundary ids (to be used for surface editing and FEM pre-processing). For example, for each material and for each boundary id a default color can be defined in the database.

Modification, insertion, and removal of parameters is performed in the same way as in the ordinary parameter editor. In addition, the extended parameter dialog provides a menu bar allowing you to load, import, save, or search the global parameter database. Amira's default database is stored in the file `share/materials/database.hm` located in the directory where Amira was installed. Use the *Database* menu option *Set default file* to specify that a different database file be used instead. This change is permanent, i.e., it takes effect also if Amira is restarted. To switch back to the system default, use the *Database* menu option *Use system default file*.

3.1.2.7 Preferences

This option opens the Amira preferences dialog described in Section 3.1.13. The dialog controls how network scripts are exported. It also lets you choose if warning dialogs should be popped up if you try to delete data objects which have not yet been saved to file, or if you try to exit Amira without having saved the current network before. Possibly most interesting, it allows you to configure the layout of your Amira window.

3.1.3 Pool Menu

The Pool menu is displayed if "graph view" is selected. This selection is made using the the *Layout* tab of the *Edit/Preferences* menu.

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

3.1.3.1 Hide

The *Hide* button hides all currently selected objects. The object's icons are removed from the Pool but the objects themselves are retained. You get the same effect by pressing the `Ctrl-H` key. Hidden objects can be made visible again using *Show* or *Show All*.

3.1.3.2 Remove

The *Remove* button deletes all selected objects and removes the corresponding icons from the Pool. You can get the same effect by pressing `Ctrl-X`. If you want to reuse a data object later on, be sure to save it in a file before deleting it. If a data object has been modified but has not yet been saved to a file, it is marked by a little asterisk in the object icon. In the Preferences dialog you can choose whether a warning dialog should be printed if you try to delete unsaved data objects which cannot be recomputed by an up-stream compute module. If you delete a data object, all connected modules will be deleted as well. However, if you delete a module connected data objects (e.g., the results of a compute module) will be retained.

Medivis Exhibit 1023

3.1.3.3 Duplicate

The *Duplicate* button creates copies of all selected data objects. For each copy a new data icon is put in the Pool. The name of a duplicated data object differs from the original one by one or more appended digits. The duplicate option is not available if you have selected icons that do not represent data objects (e.g., display or compute modules).

3.1.3.4 Rename

This button allows you to change the name of a selected object in a small dialog box which is popped up when the button is pressed. If no object is selected or if multiple objects are selected the button is disabled. Note that no two objects in Amira can have the same name. Therefore, the name entered in the dialog may be modified by appending digits to it, if necessary. The dialog can also be raised by pressing the `F2` key on a selected object into the pool.

3.1.3.5 Show

The *Show* button allows you to make hidden objects visible, so that their icons are displayed in the 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.

3.1.3.6 Show All

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

3.1.3.7 Remove All

The *Remove All* button deletes all currently visible icons and the associated objects from the Pool. A pre-loaded colormap that is currently visible is also deleted, but all hidden objects are retained. If you select the option *check if data objects need to be saved* in the Preferences dialog, a warning dialog is popped up if there are data objects which have not yet been saved to a file.

Medivis Exhibit 1023

3.1.3.8 Duplicate mode

This mode is applicable when multiple modules of the same type will be attached to a single data object. If the toggle is on, the port values of the first module will be used to initialize the port values of subsequently attached modules. This can be convenient if you want the modules to have the same initial port values.
Example: Attach an Orthoslice to your data object and set the Mapping type to ”Colormap” and the colormap range to 0-3. If you attach another Orthoslice to the same data object, its Mapping type will be ”Colormap” and its colormap range will be 0-3.

3.1.3.9 Auto adjust range of colormaps

If this toggle is on, requests that Amira automatically adjust the range of a colormap to correspond to the data range of the data set it is connected to. Note that this feature only applies to the seismic modules: seismic slices Inline, Crossline, TimeSlice, etc., CroppedVolume, and SeismicSurfaceView.

3.1.4 Explorer Menu

The Explorer menu is displayed if ”tree view” is selected. This selection is made using the the *Layout* tab of the *Edit/Preferences* menu.
The Explorer menu is identical to the Pool menu except that the *Show*, *Show All*, and *Hide* items are not available.
See the *Pool menu* description for complete details.

3.1.5 Create Menu

The *Create* menu lets you create modules or data objects that cannot be accessed via the popup menu of any other object. The *Create* menu provides different categories like the popup menu in the Pool. For example, you can create a procedurally defined scalar field (where you can type in some arithmetic expression) by choosing *Scalarfield* from the *Data* sub-menu. The icon of a newly created object usually will not be connected to any other object in the Pool. In order to establish connections later on, use the popup menu over the small white square on the left side of the object’s icon. You can also put in links to scripts in the *Create* menu. Details are defined in Section 5.5 (Configuring popup menus).

Medivis Exhibit 1023

3.1.6 View Menu

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

3.1.6.1 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.

3.1.6.2 Background

The *Background* button opens the background dialog, allowing you to switch between the different background styles *uniform*, *gradient*, and *checkerboard*. In addition, the dialog allows you to adjust the two colors used by these styles.
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 5.3.3.1, viewer commands).

3.1.6.3 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 Delayed:* Additive alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Add Sorted:* Like *Add Delayed*, 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.

Medivis Exhibit 1023

- *Blend Delayed:* Multiplicative alpha blending with two rendering passes. Opaque objects come first and transparent objects come second.
- *Blend Sorted:* Like *Blend Delayed*, but transparent objects are sorted by distances of bounding box centers from the camera and are rendered in back-to-front order.
- *Sorted Layers:* Uses a fragment-level depth sorting technique, which gives better results for complex transparent objects. Multi-Texture, Texture Environment Combine, Depth texture, and Shadow OpenGL extensions must be supported by your graphics board. If the graphics board does not support these extensions, behaves as if *Blend Sorted* was set.
- *Sorted Layers Delayed:* Like *Sorted Layers*, but rendering all transparent objects after opaque ones.

3.1.6.4 Lights

The *Lights* menu lets you activate different light settings for the 3D viewer. By default, the viewer uses a single headlight, i.e., a directional light pointing in almost the same direction as the camera is looking. The headlight can be switched on or off in each viewer via the viewer's popup menu. Alternatively, the headlight can be switched on or off for all viewers using the headlight toggle in this *Lights* menu. This standard light settings can be restored using the *Standard* button. More light settings can be defined by creating an appropriate file in `$AMIRA_ROOT/share/lights`. By default, Amira provides one additional light setting including colored lights (*BlueRed*).
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 Pool, just like ordinary data objects or modules. In order to make all hidden light icons visible, use the *Show all icons* option. *Hide all icons* hides the icons of all light objects. For more information about *lights*, please refer to the Reference Section of this manual.

3.1.6.5 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.*

Medivis Exhibit 1023

- *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.

3.1.6.6 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.

3.1.6.7 Measuring

The *Measuring* button creates an instance of a *Measuring* module that lets you measure distances and angles on objects within the viewer.

3.1.6.8 Easy Fade

The *Easy Fade* 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.

3.1.6.9 Frame Counter

The *Frame Counter* toggle lets you switch on a frames-per-second counter that will be displayed in the first viewer (viewer 0).

3.1.6.10 Console

The *Console* toggle lets you switch on or off display of the *Console window*.

3.1.7 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 section 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 accessible from the home page of the online help browser. Amira modules also provide a question mark button in the Properties Area. Pressing this button directly pops up the help browser for the particular module.

By default, the help browser is displayed in its own top-level window, but you can easily integrate it into the console pane. In this case, a tab button allows you to switch between the Console and the Help window. To change the way the help browser is displayed, use the *Layout* tab of the *Edit/Preferences* menu.

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 content and full text search. If you want to search through the complete documentation, enter the desired text into the text field. Pressing CTRL-SHIFT-F moves the focus to this text field and marks all text in this field for quick access. The search will be performed upon pressing the *Enter* key.

The search is done by using the complete search phrase. For example, suppose you are looking for information about the *surface editor* the output shows the page title where the phrase is contained and a small text surrounding the search phrase.

Medivis Exhibit 1023

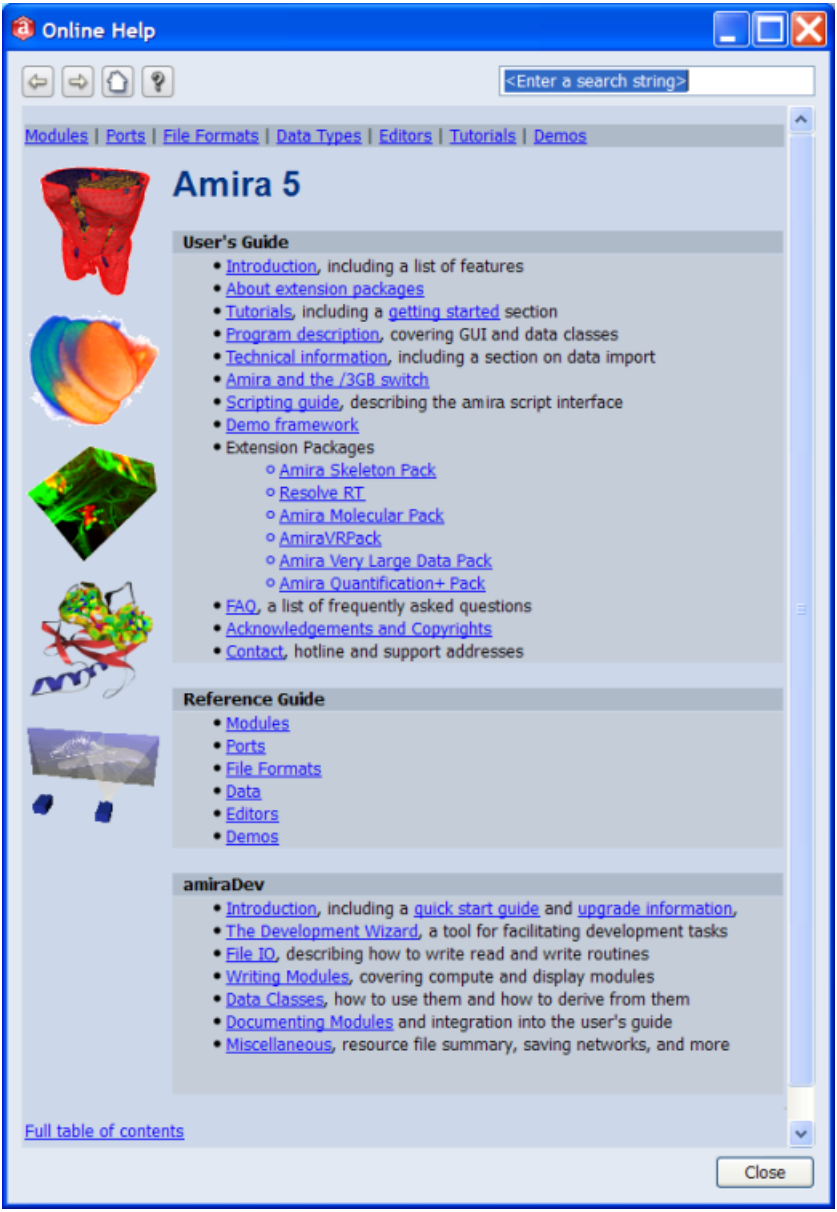


Figure 3.1: Amira's help window.

Figure 3.2: The *Find pane*.

Searching in a help document is done by entering text in the *Find pane*. You can open it by clicking on the *Show find pane* icon or by pressing *CTRL-F*. This function behaves like the content search and looks for the occurrence of the complete phrase you type into the text edit field. Pressing *Enter*, *F3* or clicking on *next* will mark the searched phrase in the text. Another click/keypress takes you to the next occurrence of this phrase. You can search backwards by pressing *SHIFT-F3* or clicking on *prev* to find the previous occurrence of the phrase. If the search reaches the upper or lower end of the document, it starts over depending on the search direction. Checking the *Highlight all* check box highlights the searched phrase with a yellow background wherever it is found in the text. Highlighting, if enabled, is maintained when clicking on links. The *Find pane* can be closed by clicking the red X on the pane or by unchecking the *Show find pane* icon.
All searches are done case insensitive!
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.

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 the browser.

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

Medivis Exhibit 1023

`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 4.4.

`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.

3.1.8 Main Window

Amira's Main window consists of two components: the *sub-application Pool* or *Explorer* in the upper part of the window, and the *Properties Area* in the lower part.

The Pool contains icons representing data objects and modules currently in use as well as lines connecting icons indicating dependencies between objects and modules. Like the Pool, the Explorer contains icons representing data objects and modules currently in use. However this information is displayed in a tree view. The user can choose to use either display type by selecting the icons in the *Sub-application Toolbar* In "graph view" mode, the Pool is displayed. In "tree view" mode, the Explorer is displayed.

Note: Other than when attention is being specifically called to highlight the differences between the Pool (Pool View) and the Explorer (Tree View), "Pool" will be used throughout the Amira documentation to refer generically to the upper pane of the main Amira window containing the collection of objects and modules that define the visualization network.

The Properties 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 through them. The Pool and the Properties Area are described below.

By default, the Main Window, the *Viewer window*, the *Console window*, and the *Help browser* are all panes of a single large Amira window. However, it is possible to designate that any of these last three components be displayed in its own top-level window using the *Layout* tab of the *Edit/Preferences* menu.

Medivis Exhibit 1023

3.1.8.1 Pool (Graph View)

Concepts

The Pool is displayed if ”graph view” is selected. This selection is made using the the *Layout* tab of the *Edit/Preferences* menu.

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

Icon colors are used indicate different types of objects. Data objects are shown in green and are the only objects which can be saved to disk using *File/Save*. Computational modules are shown in red, visualization modules are yellow, and visualization modules of *slicing* type are orange. 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. For display modules, these connections show the data used to generate the display. 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 a data object icon with the right mouse button, a menu pops up that shows all types of modules that can be connected to that object. (Alternatively, you can click on the small white triangle on the right side of the icon with the left, right, or middle button.) Selecting one of the modules 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 used to trigger them. Note that modules used will appear as shortcut (or macro) buttons in the upper part of the window.

If you look closer at an object’s icon you will notice two small squares on its left, one white and one orange. If you click on the white square with the left (or right, or middle) mouse button, a menu pops up that shows all connection ports of that object.

If the white square has a ”-” or a ”+” inside, you can click it with the left (or right, or middle) mouse button to hide (”collapse”) or unhide the objects connected to that object. The collapse feature is helpful if there are too many objects in the Pool to view easily at once. The hidden objects will be visible in the *Pool/Show Object*

Medivis Exhibit 1023

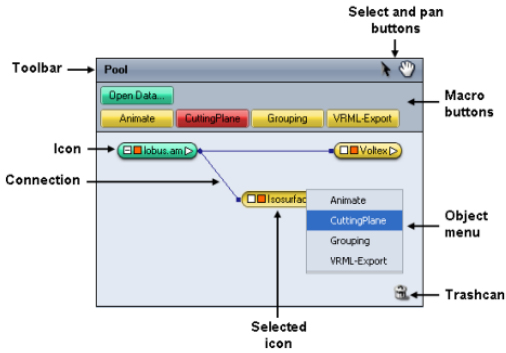


Figure 3.3: The Pool contains data objects and module icons.

menu.

The orange square controls the object's visibility in the viewers. It shows the current viewer layout (1 viewer, 2 viewers, etc.). Click inside the region of the square corresponding to a specific viewer to toggle the visibility of the object in that viewer. The region turns gray when the module is set to invisible. If you are using more than 4 viewers, each additional viewer will have its own orange square on the object's icon. You can also control an object's visibility by clicking on its visibility toggle in the Properties Area.

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's Colorfield port in order to color the surface using the values in the Colorfield.

Once you have selected an entry from the connection popup menu of the object icon, you can choose a new input object for that port. In order to do so, click on the input object's icon in the Pool. The blue connection line will become lighter blue if the connection port can be connected to the chosen object. Reasons for not being able to connect an input to a port can include the following: incompatible data type (use Castfield to change), incompatible dimensionality of the input (use ChannelWorks to change), incompatible grid type of the data (for example, use Arithmetic to sample on a regular grid), or simply that the connection does not make sense, for example, connecting a display module for surfaces to a CT data

Medivis Exhibit 1023

set.
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 left mouse button. In order to select multiple objects, you can shift-click the corresponding icons or select them by using the rectangular selection tool (press and hold down the left mouse button over the Pool background, then sweep out a rectangle). Clicking on the icon of a selected object deselects it again. Clicking somewhere on the background of the Pool causes all selected objects to be deselected. One or more selected icons may be dragged around in the Pool by clicking on them and moving the mouse pointer while holding down the mouse button.
Interface
At the right of the Pool banner are the *PoolSelect* (arrow) and *PoolPan* (hand) buttons. In PoolSelect mode, the mouse is used for selecting, positioning, connecting, and disconnecting objects in the Pool. In PoolPan mode, moving the mouse pans the Pool workspace. Press the Control key to switch temporarily from one mode to the other.
At the top of the Pool is a region containing shortcut (or macro) buttons. The *Open Data* button is a shortcut to the *File/Open Data* menu item. Up to 4 additional buttons are automatically displayed depending on the currently selected object(s). They provide easy access to the modules that are most commonly used and/or that have been recently used with the currently selected object.
The trashcan at the bottom right of the Pool provides a convenient shortcut for removing an object. Drag the object to the trashcan. When the cursor turns into an ”X”, release the mouse button and the object will be removed from the Pool. If you remove a data object, all modules downstream from that module will be deleted as well. Alternatively, you can use the *Remove object* item from the *Pool* menu, or you can use the Delete key or Ctrl-x to remove a selected object.

3.1.8.2 Explorer (Tree View)

The Explorer is displayed if ”Tree View” is selected in the *Sub-application Toolbar*.

Medivis Exhibit 1023

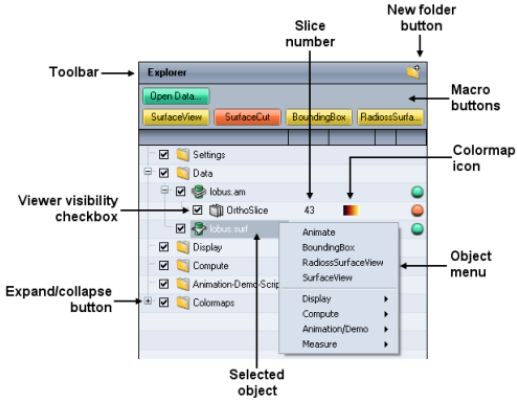


Figure 3.4: The Explorer contains data objects and module icons in a tree view.

Once a data object has been loaded or a module has been created, it will be represented by a new entry in the tree view.
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 in the *Properties Area* once the object has been selected, i.e., by clicking on its icon with the left mouse button. In order to select multiple objects, you can shift-click the corresponding icons. Clicking on the icon of a selected object deselects it again. Clicking somewhere off of the tree view, e.g., beyond the bottom of the tree view, causes all selected objects to be deselected.
At the top of the Explorer is a region containing shortcut (or macro) buttons. The *Open Data* button is a shortcut to the *File/Open Data* menu item. Up to 4 additional buttons are automatically displayed depending on the currently selected object(s). They provide easy access to the modules that are most commonly used and/or that have been recently used with the currently selected object.
The tree view display has 5 columns:
Column 1: Contains the actual tree view.

- **Organization**: The tree is organized into folders containing different kinds of objects: Data, Display, Compute, etc. Most of the folders are predefined and cannot be modified. A few, however, such as the Data folder, can have more folders added beneath.

Currently, there are two ”templates” that control the organization of the tree view: a default template for most Amira users, and a ”geoscience” template that includes predefined folders for 3D surveys, horizons, faults, etc. The user can choose the desired template using the *Layout* tab of the *Edit/Preferences* menu.

- **Connections**: If you click on a data object 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 the modules will automatically create an instance of that module type and connect it to the data object. A new item in the tree view will appear in response. In 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 used to trigger them. Note that modules used will appear as shortcut (also known as ”macro”) buttons in the upper part of the window.

  For most objects the required connections are automatically established on creation. The connection is shown in the connected object's input port in its control panel in the Properties Area. For example, when you attach a TimeSlice to `my_data.lda`, in the Data port of the TimeSlice control panel, you will see ”my_data.lda” in the pulldown list. If other objects in the Explorer can be connected to the TimeSlice, they will be in the list as well. To switch the connection, simply select a different item from the list. The tree view (and possibly the viewer display) will be updated accordingly. Selecting ”NO SOURCE” from the list disconnects the module from any object.

  Reasons for not being able to connect an input to a port can include the following: incompatible data type (use Castfield to change), incompatible dimensionality of the input (use ChannelWorks to change), incompatible grid type of the data (for example, use Arithmetic to sample on a regular grid), or simply that the connection does not make sense, for example, connecting a display module for surfaces to a volume data set.

  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.

Medivis Exhibit 1023

- **Expand/Collapse**: You can click on the + and − icons to expand or collapse respectively a portion of the tree.

- **Drag-and-Drop**: You can use drag-and-drop to move items in the tree view. For example, to connect an existing visualization module to a different data set, you can drag and drop it from its current location onto a compatible data set. Drag-and-drop does not work for disconnecting objects. You must use the connection ports for this.

- **Viewer Toggle**: The check boxes control the visibility of the object in the *currently active* viewer. If there are multiple viewers, e.g., a 4-viewer layout, you can select a viewer to be active, set the object's visibility in that viewer, then repeat as often as necessary to set the desired visibility in each viewer. However, it will probably be more convenient to control the visibility in individual viewers by using the viewer toggle button in the object's control panel in the *Properties Area*. The viewer toggle button is the orange square just to the right of the module name in its control panel. The button is divided into regions corresponding to the current viewer layout. Click on the region to toggle visibility of the object on/off in the corresponding viewer. If there are more than 4 viewers, each additional viewer will have it own toggle button.

- **Adding a Folder**: At the right of the Explorer banner is the *TVNewFolder* button used for adding a new folder directly below the currently selected folder. You can also add a folder by right clicking on an existing folder and selecting *New Folder* from the context menu. At some positions in the tree, it may not be possible to add a new folder.

- **Removing an Object**: To remove one or more objects from the Explorer, first select the object(s) to be removed. You can then use the *Remove object* item from the Explorer context menu, or you can press the Delete key or Ctrl-x to remove a selected object. If you remove a data object, all modules downstream from that module will be deleted as well.

- **Hovering**: Hovering the mouse over an item in the tree view displays information about item, usually its type, e.g., HxUniformScalarField3.

Column 2: Shows, for certain objects, the current value of a port of interest. Currently, the slice number is shown for slice modules (OrthoSlice, Inline, etc.) and the Time Scale Factor is shown for the SeismicSettings module.

Column 3: Shows the current colormap, if any, associated with the module or object. You can right click on it to bring the standard colormap context menu. Or you can left click on it to bring up the *Colormap Editor*.

Column 4: Displays the icon of the editor, if any, currently operating on the object. For example, the Crop Editor, the Parameter Editor, etc.

Column 5: Displays a colored-coded marker indicating the kind of object or module, as follows:

- Red – computational modules
- Orange – visualization modules of *slicing* type. These modules may be used to clip the graphical output of any other module.
- Yellow – visualization modules
- Green – data objects. These are the only objects which can be saved to disk using *File/Save*.
- Blue – script objects
- Purple – settings modules, e.g., SeismicSettings, LDAExpertSettings, etc.

Medivis Exhibit 1023

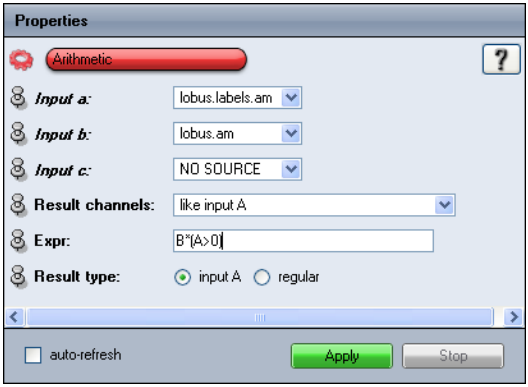


Figure 3.5: The Properties Area contains the input controls for objects in the Pool. This is the ”control panel” for the Arithmetic module.

3.1.8.3 Properties Area

Once an object has been selected, its input controls will be displayed in the Properties Area below the 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.
The *Apply* button is active (green) for modules that require you to explicitly initiate their action. Typically this is the case for modules whose action may take a significant amount of time, such as some of the compute modules. When the *Stop* button is red, you can press it to interrupt the action.
Check on the *auto-refresh* check box to automatically force an apply for any change in the network.
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

Medivis Exhibit 1023

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 Properties 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.

Ports whose labels are displayed in italic are special: they are input connection ports. They are used for connecting data objects and modules into a network that represents the computational steps required to carry out a specific visualization task. Each connection port contains the list of objects to which it can be possibly connected, plus a "NO SOURCE" item, which means not to connect to anything.
In graph view mode, display of these connection ports can be toggled on and off via the *Layout* tab of the *Edit/Preferences* menu. They are displayed by default to help beginners understanding the Pool architecture.
In tree view mode, these connection ports are always displayed because this is the only mechanism in this mode for defining networks. When an object is completely disconnected from all others, it will be moved in the tree to an appropriate folder. For example, to Display for a display module, to Compute for a compute module, and so on.
If the shadowing effect is switched on in the *Rendering* tab of the *Edit/Preferences* menu, some visualization module shows the "Shadow" button. Clicking the button will switch across the following modes:

- Cast Shadow – the object will cast shadows

- Receive Shadow – the object will only receive shadows

Medivis Exhibit 1023

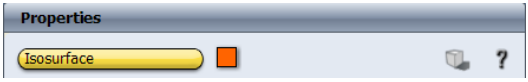


Figure 3.6: Some visualization module can show in the "Shadow" button in the Properties Area.

- Cast & Receive Shadow – the object will cast and receive shadows
- No Shadow – no shadows will be visualized

3.1.9 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.
The virtual trackball controlled by the left mouse button allows for free rotation of the camera. The camera trackball, displayed by default in the lower right corner of the viewer, is used for constrained rotation: rotation of the camera about the screen-aligned X, Y, or Z axes. Click on the vertical wheel (it becomes red when you select it) and move the mouse up/down to rotate about the X axis. Click on the horizontal wheel (it becomes green when you select it) and move the mouse left/right to rotate about the Y axis. Click on the third wheel (it becomes blue) and move the mouse up/down to rotate about the Z axis.
A 3D compass indicating the current camera viewing direction may be displayed in one of the corners of the display. It is an indicator only, you cannot use it to control the viewing direction.
The *Edit/Preferences/Layout* dialog can be used to control the visibility, auto-hide option, and position of the trackball and the compass.
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

Medivis Exhibit 1023

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 left 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.

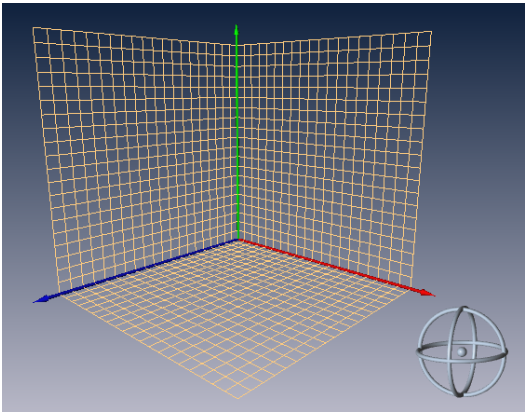


Figure 3.7: Amira's viewer window provides a virtual trackball for easy navigation, as well as a camera trackball (lower right) for constrained rotation.

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 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 in Section 5.3.3.1.

The toolbar of the main viewer window provides several buttons and controls, allowing you for example to switch between viewing mode and interaction mode, to choose certain orientations, or to take snapshots. The precise meaning of these controls is described below.

Medivis Exhibit 1023

Interact:

Switches the viewer into interaction mode. You can also use the `[ESC]` key to toggle between viewing mode and interaction mode.

Trackball:

Switches the viewer into viewing mode. You can also use the `[ESC]` key to toggle between interaction mode and viewing mode. The left mouse button is used to change the view direction by means of a virtual trackball.

Translate:

Same as *Trackball* except that the left mouse button is used for panning (translation).

Zoom:

Same as *Trackball* except that in this mode vertical motion of the left mouse button controls zooming.

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.

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.

Home:

Resets camera to the home position.

Set Home:

Sets the current position as the new home position.

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. **Note:** Only one of these buttons will be visible at a time, the button indicating the currently active camera type.

View All:

or

Repositions the camera so that all objects become visible. The orientation of the camera will not be changed. **Note:** The first button is seen if *technical* naming or the geoscience template has been selected in the *Edit/Preferences/Layout* dialog, the second button will be displayed if *medical* naming has been selected.

YZ-, XY- and XZ-Views:

or

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. **Note1:** The first set of buttons is seen if *technical* naming has been selected in the *Edit/Preferences/Layout dialog*, the second set of buttons will be displayed if *medical* naming has been selected. They correspond to axial, coronal, and sagittal views respectively.

Medivis Exhibit 1023

Stereo:

Allows you to enable or disable stereo viewing, as well as specify various stereo viewing parameters via the *Stereo Preferences* dialog.

Measuring:

Pressing this button creates an instance of a *Measuring* module that lets you measure distances and angles on objects within the viewer. Clicking on the down arrow will display a menu of measuring tools to chose from: *2D line*, *3D line*, *2D angle*, *3D angle*, and *annotation*. See the Measuring module's documentation for details. **Note:** Only one of these buttons will be visible on the toolbar at a time, the button of the measuring tool most recently accessed from the viewer toolbar.

Snapshot:

Takes a snapshot of the current rendering area and saves it in a file. The filename as well as the desired output format must be entered through the *Snapshot dialog*. Snapshots may also be taken using the *viewer command* `snapshot`.

Layout:

Selects the viewer layout: a single view, two viewers side-by-side, two viewers stacked, or four viewers.

Fullscreen:

Selects fullscreen mode. In this mode, the viewer occupies the entire screen and no other windows will be visible. To exit fullscreen mode, click the right mouse button and uncheck *Fullscreen* in the popup menu.

In addition to these buttons, the Amira viewers provide an extensive set of *Tcl commands*, which are listed in Section 5.3.3.1.

3.1.10 Console Window

The *console window* is a command shell allowing to access Amira's advanced control features. It serves two purposes. First, it gives you some feedback on what is currently going on. Such feedback messages include warnings, error indications and notes on problems as well as information on results. Second, it provides a command line interface where Amira commands can be entered.

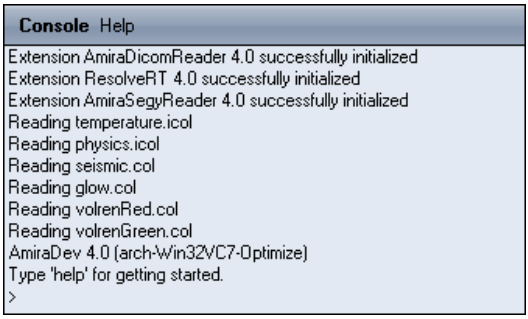


Figure 3.8: Amira's console window displays info messages and lets you enter Tcl commands.

Amira's console commands are based on the *Tcl* script language *(Tool Command Language)*. Examples are:

```
load C:/MyData/something.am
viewer 0 setSize 200 200
viewer 0 snapshot C:/snapshot.tif
```

The Amira scripting syntax and the specific commands are described in the Chapter 5 (Scripting). 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, then the name of the object will be automatically inserted.
You can also type the beginning of a command word and type the `[TAB]` key to complete the word. This only works if the beginning is unique. Pressing `[TAB]` a second time will show the possible completions. Often, this saves a lot of typing. Commands provided by data objects and modules are documented in the reference

Medivis Exhibit 1023

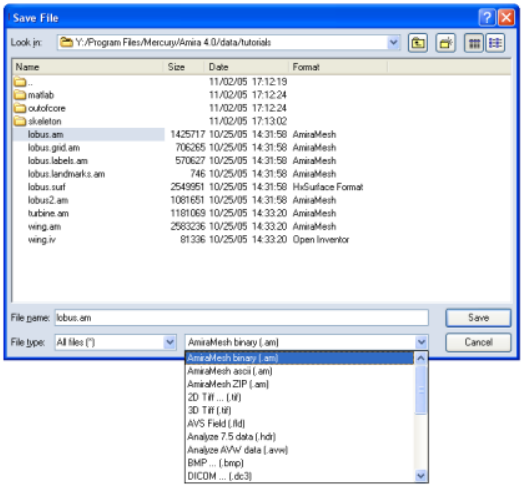


Figure 3.9: Amira's file dialog.

section of the users guide. Pressing the `[F1]` key for such a command without any arguments pops up the help text for this command. This is also true for commands provided by the *ports* of an object.
Additionally the *console window* provides 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 `source <filename>` or load the script file via Amira's file dialog from the file menu. Amira script files are usually identified by the extension `.hx`. For advanced script examples take a look at Amira's demo files located in `$AMIRA_ROOT/share/demo`.

3.1.11 File Dialog

The *File Dialog* is the user interface component for importing and exporting data into and out of 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 analyzing 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).

3.1.11.1 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 containing 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.

3.1.11.2 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.

3.1.11.3 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`.

Medivis Exhibit 1023

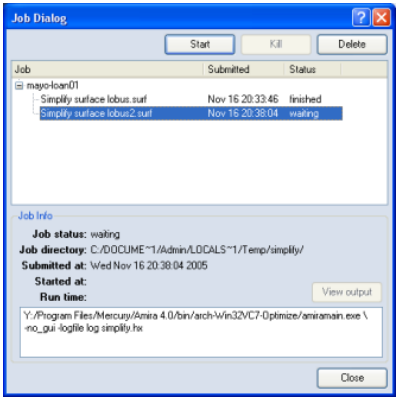


Figure 3.10: The job dialog lets you start, stop, examine, and delete batch jobs.

3.1.11.4 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.

3.1.12 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.

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"`

Medivis Exhibit 1023

`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 consecutively by `job run`.

3.1.13 Preferences Dialog

The *Preferences Dialog* allows you to adjust certain global settings of Amira. The preferences are stored in a permanent fashion on a per-user basis. The dialog contains a tab bar with several tabs. The first tab, *General*, is related to the Pool and saving the network. The second tab, *Layout*, affects the layout of the user interface. The third, *On Exit*, controls what conditions are checked for in the networks when Amira exits. The *Molecules* tab allows you to specify options for handling molecular data. The *LDA* tab allows you to specify options for out-of-core data. For advanced users, additional options can be set using the *LDAExpertSettings* module. The *Segmentation* tab allows you to specify options for the Segmentation Editor. See the *Segmentation Editor* documentation for details.

3.1.13.1 The General Tab

2-pass firing algorithm
If set, a slightly more complex firing algorithm is used which ensures that downstream modules connected to an up-stream object via multiple paths are only fired once if the up-stream object changes. The default is on.
Auto-select new modules
If set, a new module selected from the popup menu of its parent object is shown automatically in the Properties Area. The default is on.
Deselect previously selected modules
This option can only be set if auto-selection is turned on. If set, all objects are deselected before selecting the new module. Otherwise the new module will be appended at the end of the Properties Area. The default is on.
Draw viewer toggles on icons
If set, small viewer mask toggles are drawn on the icons of data objects and display modules. This allows you to show or hide a module in a viewer without selecting it first. The default is on.
Draw compute indicator
If set, a small red rectangle is drawn inside the icon of a module to indicate that the module is currently working. The default is on.

Medivis Exhibit 1023

Include unused data objects
If set, all data objects including hidden colormaps are stored in network scripts. When executing such a script all existing objects are removed first. If not set only visible data objects and objects which are referenced by others are stored in a network script. When executing the script hidden data objects are not removed. The default is on.
Include window sizes and positions
If set, window sizes and positions are stored in network scripts. Be careful using this option if you want to send your script to a machine with a different screen resolution.
Overwrite existing files in auto-save
If set, no overwrite check is performed for data objects which need to be saved automatically in order to create a network script. Otherwise a unique file name will be chosen. The default is on. Details about the auto-save feature are described in Section 3.1.1.7.

3.1.13.2 The Layout Tab

Naming convention
These toggles allow you chose between two naming conventions: *medical* and *technical*. Your choice will affect the labels of some ports, for example, Orientation ports, as well as the appearance of the viewer buttons controlling the view orientation.
Windows
These items allow you to configure the layout of the Amira windows. By default, the main Amira interface components (Pool, Properties Area, main Viewer window, Console, and Help Browser) share one large window. You can use the check boxes to display some of these components in separate top-level windows. This gives you additional flexibility in managing the ”real-estate” of your graphics display. For example, on a dual-head display it can be interesting to display the main Viewer window on one display and the rest of the Amira interface on the other.
Show viewer window in top-level window
The main Viewer window will be displayed in a top-level window.
Show viewer on left-hand side
The main Viewer window will be displayed on the left-hand side.
Show console in top-level window
The console will be displayed in a top-level window.

Medivis Exhibit 1023

Show help browser in top-level window
The help browser will be displayed in a top-level window.
Show "DoIt" buttons
Some modules have a button, usually labeled "DoIt", to initiate the action of the module. Since Amira 4.0, by default, this button is not displayed in the Properties Area. Rather, the green *Apply* button at the bottom of the Properties Area is used to initiate the action. If this box is checked, the button will be displayed in the Properties Area. The green *Apply* button will still be available for use as well.

Finally, you have the choice to *Save current layout (now)*, and *Restore current layout (now)*.

Viewer gadgets

There are two viewer gadgets, a camera trackball and a compass.

The camera trackball, used for constrained rotation of the camera about the screen-aligned X, Y, or Z axes, is described in Section 3.1.9.

The 3D compass indicates the direction from which the camera is viewing the scene. See Section 3.1.9 for more details on the compass.

Click on the tab of the gadget whose attributes you wish to control, then set its attributes as described below.

Show the camera trackball
Show the compass
This toggle controls the visibility of the trackball (compass). The default is on.
Auto-hide the camera trackball
Auto-hide the compass
When this box is checked, the trackball (compass) is only displayed while the mouse is within the trackball (compass) display area. It is hidden as soon as the mouse moves outside the trackball (compass) display area. The default is off. This item is not active if the *Show the trackball (or compass)* item is off.
Camera trackball position
Compass position
This menu allows you to specify which corner of the viewer window to display the trackball (compass) in. The default is *Lower right* for the trackball, and *Upper right* for the compass. This item is not active if the *Show the trackball (or compass)* item is off.

Pool

You can choose between a graph view and a tree view display of the contents of the Pool. The default display modus is the *Graph view*, which is described in Section 3.1.8.1. A description of the *Tree view* is available in Section 3.1.8.2.

If *Graph view* is selected, you can choose to *Show connection ports in the Properties Area*. If you check this box on, an object's/module's connection ports are shown at the top of its control panel in the Properties Area. By default this is off because the connections (shown as blue lines) between data objects and modules can be conveniently managed directly in the Pool. Changes to the connections made in the Pool are reflected in the connection ports in the Properties Area, and vice versa.
If *Tree view* is selected, this box is *always* checked on because in tree view, using the connection ports in the Properties Area is the only convenient way to manage the connections between data objects and modules.

3.1.13.3 The On Exit Tab

The page allows you to control what conditions are checked for in the networks when Amira exits.

3.1.13.4 The Molecules Tab

Color Schemes
These check boxes allow you to chose alternate color schemes: CPK for atoms, and RasMol for amino acids.
Selection Info
These items control how much information is printed into the console when parts of a molecule are selected. Activate *Molecule name* if the name of the molecule should be printed. Activate *Group name* if the name of the selected group should be printed. If you activate *Group attributes*, all attributes of the selected group are printed. If *Explicit attributes* is activated, the printed attributes are restricted to those explicitly named in the corresponding text field.
Atom Expressions *ID case sensitive*
Specifies if atom identifiers are case sensitive or note.

3.1.13.5 The LDA Tab

Out-of-core threshold
Specifies the size above which data sets will be treated as out-of-core data.
Compression
Specifies the type of data compression.

Medivis Exhibit 1023

Tile Size
The size of the tiles to be compressed.
Border Size
The size of border of the tiles.
Main memory amount
Sets the maximum allowed main memory in MB (megabytes) for all the out-of-core data sets.
Video memory amount
Sets the maximum allowed texture memory in MB (megabytes) for all the out-of-core data sets.
Viewpoint refinement
If set, refinement depends on the viewpoint.
View culling
If set, refinement takes place only in the view frustum.
Move low resolution
If set, ortho slices and oblique slices are computed in half resolution when moving.
Loading policy
Sets loading behavior. If *No Interaction* is selected, the asynchronous loading thread will only load when the user does not interact with the scene. If *Always* is selected, loading occurs as long as there is something to load. If *Never* is selected, no loading occurs. The default is *No Interaction*.

3.1.13.6 The Segmentation Tab

3D Draw style
This option lets you choose the material draw style used in the 3D viewer.
Selection Draw style
This option lets you choose the material draw style used to highlight the voxels selection.

3.1.13.7 The Rendering Tab

Shadowing
If set, shadows will be generated. You can select also the default behaviour, the intensity and the quality of the shadows. If this option is switched on, some visualization module shows the ”Shadow” button, through which you can directly specify the object behaviour.

Medivis Exhibit 1023

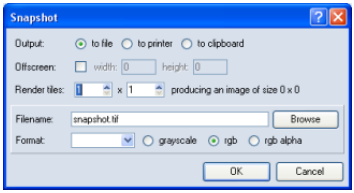


Figure 3.11: The snapshot dialog allows you to save or print the contents of a viewer window.

Surface Rendering
The two options can be used to optimize the rendering performance of surface objects according to your system requirements.

3.1.14 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 *viewer* toolbar.

- **Output:** Specifies the output device. With *to file* the grabbed image is saved to a file, with *to printer* the image is sent directly to the printer, and with *to clipboard* it is sent to the clipboard. In the *to printer* mode you first must select and configure a printer by pressing the *Configure* button. In addition, you may enter an arbitrary text string which is printed as an annotation text below the snapshot image.
- **Offscreen:** Lets you grab images larger than the actual screen size. When this option is checked, the output dimensions can be specified in the *width* and *height* text fields up to a maximum of 2048 x 2048 pixels.
- **Render tiles:** Use this option to render snapshots of virtually unlimited resolution (e.g. for high quality printouts). In this mode the scene is divided into n x m tiles where n and m can be entered into the adjacent text fields. Then the camera position is set such that each tile fills the current viewer and a snapshot is taken. Finally the tiles are internally merged to a single image and sent to the device specified in the *Output* port.
- **Filename:** Lets you specify the filename if the *to file* option is set. The *Browse* button allows you to browse to a desired location within the filesys-

Medivis Exhibit 1023

tem.

- **Format:** The format option lets you select the file format to be produced for file output. The following formats are supported: TIFF (`.tif, .tiff`), SGI-RGB (`.rgb, .sgi, .bw`), JPEG (`.jpg, .jpeg`), PPM (`.pgm, .ppm`), BMP (`.bmp`), PNG (`.png`), DCM (`.dcm`), and Encapsulated PostScript (`.eps`). In addition, this port offers three radio buttons to choose between *grayscale*, *rgb*, and *rgb alpha* type of raster images. If *rgb alpha* option is set, images are produced such that the viewer background is assigned to the alpha channel. This option is not available for file formats that do not support an alpha channel.

3.1.15 System Information Dialog

The system information dialog provides diagnostics information allowing the user or the Amira support team to better analyze software problems. The dialog contains a tab bar with three pages. The first page lists information about the current CPU. The second page lists information about the current OpenGL graphics driver. The third page lists version information about the currently installed Amira components.

In the lower left part of the dialog you will find a button *Save Report*. With this button all information can be written into a text file. In case of a support call, you may be asked to send this text file to the hotline.

3.1.15.1 The CPU Tab

This page displays information about the system on which you are running Amira. This information can be useful when reporting problems to technical support.

3.1.15.2 The OpenGL Tab

This page displays information about the current OpenGL graphics driver. In particular, a list of available OpenGL extensions is printed. This list allows it to check if certain rendering techniques like direct volume rendering via 3D textures are supported on a particular hardware platform or not.

Medivis Exhibit 1023

3.1.15.3 The Libraries Tab

This page displays a list of all DLLs or shared libraries contained in the Amira lib directory. For each library certain version information as well as an MD5 checksum are printed. In this way it is possible to check whether a certain patch has already been installed or not. For most libraries the version information is compiled in. For other libraries this information is read from the version information files found in `share/versions` in the `AMIRA_ROOT` directory.

3.2 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*
- *Shadowing*
- *Subapplication Concept*

3.2.1 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., gray level image data or vector field sets, appear as separate icons in the *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 *OrthoSlice* 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

Medivis Exhibit 1023 166

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 occurring 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 tetrahedral 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 *Properties 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.

Medivis Exhibit 1023

For example, a bounding box is usually created by choosing the *BoundingBox* item from the pop-up menu of a data object that is to be visualized, but you may also create it by typing `create HxBoundingBox` in the command window.

3.2.2 Scalar Field and Vector Fields

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

3.2.2.1 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 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

Medivis Exhibit 1023

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 *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.

3.2.2.2 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.

3.2.3 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.

3.2.3.1 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 neighboring 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.

Medivis Exhibit 1023

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.

3.2.3.2 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.

3.2.4 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.

Medivis Exhibit 1023

3.2.5 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.

3.2.6 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.

Medivis Exhibit 1023

3.2.7 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 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.

3.2.8 Shadowing

Real time shadow casting is enabled through the Edit/Preferences/Rendering menu.
Once enabled, most of display modules can cast or receive shadows.
Different modes are available, and switching from one to another is possible by clicking on the shadow icon of a display module

- `No Shadows` the displayed shape neither cast or receive shadows
- `Cast Shadows` the displayed shape only cast shadows
- `Receive shadows` the displayed shape only receive shadows

Medivis Exhibit 1023

- `Cast & Receive shadows` the displayed shape both cast and receive shadows

Restrictions:

At least the Multi-Texture and Texture Environment Combine OpenGL extensions must be supported by your graphics board. Otherwise no shadows will be computed. These extensions are now standard in OpenGL 1.3 and later.

The Shadow and Depth Texture OpenGL extensions (standard in OpenGL 1.4) are used if they are available and generally improve performance.

Some aliasing artifacts can appear if you zoom in very close to the scene. You can then increase the quality in the preferences dialog.

Transparent objects are treated as follows, depending on the transparency type:

- `Screen door`: fully compatible
- `Add, Blend` : transparent objects cast a shadow and are shadowed but the shadow intensity doesn't depend on the transparency value (the same shadow is displayed for full transparent shapes and opaque shapes).
- `All other modes` : incompatible with shadowing.

3.2.9 Sub-application Concept

Beside visualization modules, the user interface enable an easier and faster swap between application areas. With the user-friendly "sub-application" concept several dedicated components are accessible as independent applications assembled together into the general Amira platform. Each component provides its own specific interface, its dedicated tools and visualization options. Since switching between different components takes place through a simple toolbar, the user can explore and approach complex data sets using dedicated tools in the sub-applications, and keep a clear overview in the general purpose framework. To access the sub-application a toolbar is available in the upper-left corner of the main window.
The following components are currently sub-applications:

- Object Pool

Medivis Exhibit 1023

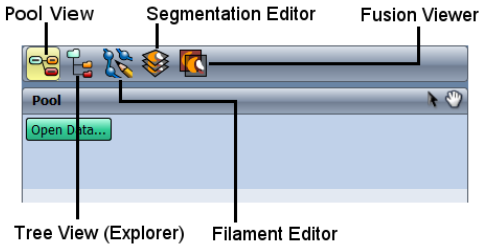


Figure 3.12: The Sub-application Toolbar.

- Tree View
- Segmentation Editor
- Filament Editor (requires Microscopy Option)
- Multi-planar Viewer

Technically, the sub-applications shall be registered dynamically in Amira .rc files like standard modules. The DLL providing a sub-application are loaded at run-time when the sub-application is opened.

Medivis Exhibit 1023

4 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*
- *System Requirements*
- *Amira License Manager*
- *Notes for Mac users*

4.1 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

Medivis Exhibit 1023

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.

Alternatively, with Developer Option a C++ programmer can extend Amira in order to integrate a custom reader or writer.

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.

4.2 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-Win32VC8-Optimize/`Amira`main.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.

Medivis Exhibit 1023 176

- `-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.
- `-clusterdaemon`
  Start as VR daemon, on a cluster slave node (Virtual Reality Option). This may be replaced by a service. See Section 8.3.

4.3 Environment Variables

In order to execute Amira no special environment settings are required. On Unix systems some environment variables like the shared library path or the `AMIRA_ROOT` directory are set automatically by the Amira start script. Other en-

vironment variables may be set by the user in order to control certain features. These variables are listed below. On Unix systems environment variables can be set using the shell commands `setenv` (csh or tcsh) or `export` (sh, bash, or ksh). On Windows environment variables can be defined in the system properties dialog (Windows 2000/XP/Vista).

- `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_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. In such cases the performance of the *Voltex* module might be improved using this variable.
- `AMIRA_MULTISAMPLE`
  On high-end graphics systems, a multi-sample visual is used 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. Note that on other systems, especially on PCs, anti-aliasing cannot be controlled by the application but has to be activated directly in the graphics driver.
- `AMIRA_NO_LICENSE_MESSAGE`
  By default, Amira issues warning messages to the console when your Amira license is about to expire. This allows you to take timely action so that your use of Amira is not interrupted unexpectedly when the license expires. To disable these messages, set this variable to 1.
- `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, or if your X server does not support overlay visuals.
- `AMIRA_NO_SPLASH_SCREEN`
  If this variable is set, Amira will not display a splash screen while it is initializing.

Medivis Exhibit 1023

- AMIRA_LOCAL
  Specifies the location of the local Amira directory containing user-defined modules. IO routines or modules defined in this directory replace the ones defined in the main Amira directory. This environment variable overwrites the local Amira directory set in the development wizard (see Amira programmer's guide for details).
- AMIRA_SMALLFONT
  Unix systems only. If this variable is set a small font will be used in all ports being displayed in the *Properties 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*.
- AMIRA_SPACEMOUSE
  This variable has to be set in order to support a spaceball or spacemouse device (see *www.spacemouse.com*). With the spacemouse you can navigate in the 3D viewer window. Two modes are supported, a rotate mode and a fly mode. You can switch between the two modes by pressing the spacemouse buttons 1 or 2.
- AMIRA_STEREO_ON_DEFAULT
  If this variable is set the 3D viewer will be opened in OpenGL raw stereo mode by default. In this way some screen flicker can be avoided which otherwise occurs when switching from mono to stereo mode. Currently the variable is supported on Unix systems only.
- 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.

4.4 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/Amira of the Amira installation directory. This script is read each time the program is started. Among other things, the start-up script

Medivis Exhibit 1023

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/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.

4.5 System Requirements

Amira runs on:

- Windows 2000/XP/Vista, 32-bit
- Windows XP/Vista (Intel 64/AMD64 architecture), 64-bit

Medivis Exhibit 1023

- Mac OS X 10.5 (Leopard) with Intel CPU
- Red Hat Enterprise Linux 5.0 for x86_64 or compatible

Amira relies on hardware-accelerated OpenGL 3D graphics. Please note that if software rendering is used, rendering performance may drop significantly, especially for visualization techniques like volume rendering. Some visualization techniques may also require 3D texturing or programmable shaders, available on recent graphics boards. For platform-specific details on hardware acceleration, see below. Amira requires a display resolution of at least 1024x768 and at least 15 bits of color depth. We strongly recommend 1280x1024 with 24-bit color depth at least.

Apart from 3D graphics hardware, probably the most important system parameter is main memory. You should have at least 512 MB, preferably 2 GB or more.
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 2 GHz processor.

4.5.1 Troubleshooting

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 1 GB. 2 GB would be even better.
Especially on PC platforms, OpenGL drivers today are not always 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 24 or 32 bit.

Medivis Exhibit 1023

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 overnight (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 or a patch can be prepared.

4.5.2 Microsoft Windows

Amira runs on Intel or AMD-based systems with Microsoft Windows 2000, Microsoft Windows XP and Microsoft Windows Vista, 32 bits and 64 bits.

Graphics Hardware: You should use a graphics board with OpenGL support and texture mapping capabilities. Some visualization techniques may also require 3D texturing and programmable shaders, available on recent graphics boards.

Note: On Windows Vista, you may want to disable Windows Aero for saving hardware resources and achieving best 3D performance with Amira. In Control Panel, open *Appearance and Personalization*, then click *Personalization*. You can alternatively right-click an empty spot on your desktop and click *Personalize* in pop-up. Click *Window Color and Appearance*, and then click *Open Classic Appearance Properties for More Color Options*. Now you can select a *Color scheme* different from Windows Aero, such as the Windows classic theme.

In order to add custom extensions to Amira with Developer Option you will need Visual Studio 2005, while Visual Studio 2008 is not officially supported for development. Even though not supported, you may compile new modules with Visual Studio 2008, but you will need to install the Visual Studio 2008 C runtimes additionally to Amira on any Windows PC that uses your custom modules.

4.5.3 Linux

The Linux version of Amira as been developed and tested on Red Hat Enterprise Linux 4.0 and 5.0.

On other Linux distributions this version might not run because certain required system libraries are missing or because different versions of these libraries are installed. In particular you may need to install `libstdc++.so.5` included in the

Medivis Exhibit 1023

compat-libstdc++ package (or you may copy it into the Amira installation directory under `lib/arch-Linux...-Optimize/`).

On Linux, Amira is only available for AMD64 / Intel 64 systems.

Graphics Hardware: Amira works with the current 3D graphics drivers from nVidia and ATI under XFree86 4/Xorg.

Notes:

- After a standard installation of Linux, hardware acceleration is not necessarily activated, although X-Windows and Amira may work fine. To enable OpenGL hardware acceleration specific drivers may have to be installed. This can dramatically increase rendering performance. Sometimes it is necessary to disable the stencil buffers (by starting Amira with the option -no_stencils) to get acceleration.

- On some distribution, you may see incorrect display of some parts of the user interface, such as the segmentation editor. This is a known Qt issue. You can work around this by disabling the composite option in the extension section of your XFree86.conf or Xorg.conf configuration file:

```
Section "Extensions"
     Option      "Composite"     "disable"
EndSection
```

In order to add custom extensions to Amira with Developer Option you will need gcc 4.1.x on RHEL 5.

4.5.4 Mac OS

The Mac version of Amira has been developed and tested on Mac OS X 10.5 (Leopard) running on an Intel CPU. This version might not work properly on other Mac OS releases or non-Intel CPU Apple platforms.

In order to add custom extensions to Amira with Developer Option you will need gcc 4.0.x provided by the standard XCode development environment.

Graphics Hardware: Amira works with the current Mac 3D graphics drivers from nVidia and ATI.

4.6 Amira License Manager

4.6.1 Contents

- *About Password Protection*
- *About the Amira License Manager*
- *Troubleshooting*
- *Using TGS_LICENSE_DEBUG*
- *Contacting the License Administrator*
- *Contacting Technical Support*

4.6.2 About Password Protection

Your Amira software is ”password enabled”. Each time the software runs, it checks for a valid password. If a valid password is found, the application runs. Otherwise, a license dialog is displayed, allowing you to enter a valid Amira license.

Generally, your software is also ”computer ID locked”. Computer ID locking is a form of software license control that allows the software to run only on specifically licensed systems. If your license is for a specific system, your password string will include a computer ID.

If your site has purchased Amira floating licenses for use with FLEXNet, you will find information in a specific section about *Configuring FLEXNet Licensing*, including instructions for *installing the Client License Files*. This documentation may be found in

```
<install directory>/share/license/FLEXNet
```

4.6.3 About the Amira License Manager

The Amira License Manager dialog is displayed when you attempt to start Amira on a system for which no valid Amira license is found:

The Amira License Manager has an editable text field displaying the contents of the active license file. The path of the active license file is given in the box above.

Medivis Exhibit 1023

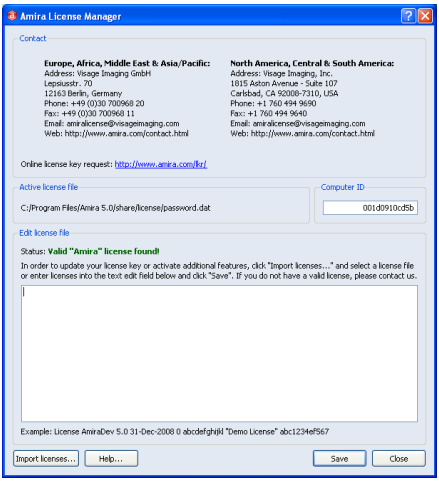


Figure 4.1: Amira license manager

You may enter new license keys by directly typing into the text field or by pasting the contents of the clipboard. Alternatively, you may import licenses by clicking the *Import license file ...* button and pointing the file browser to the file containing the license keys, or use Drag & Drop onto the text field to load the license file. The contents of the file will be appended to the active license file. License keys that are invalid are printed in red, valid licenses are printed green. To run the application there has to be a valid license for the product ”Amira ”. Thus, the file may contain valid (i.e. green) license keys for product options other than ”Amira ”, and still Amira will not start.
Once Amira finds a valid ”Amira ” license it will start. If you want to update the license key or enter additional license keys for options you will need to select **Help** > **License Manager** from the Amira menu bar and apply one of the methods described above to update the license.

4.6.4 Troubleshooting

Having trouble entering your Amira license? License is not valid? When you try to run Amira, the License Manager dialog is displayed again?

Here are steps for remedying the situation.

1. If you are using an evaluation or demo license,

   ```
   License Amira 5.0 31-Oct-2006 0 7f3a8i5u4j4j
   "Demo License"
   ```

   make sure that expiration date is in the future.

   Action: Ask your Amira reseller for an extension of the trial period or purchase a license.

2. If your license key has a computer ID, for example,

   ```
   License Amira 5.0 31-Oct-2006 0 7f3a8i5u4j4j
   "JustTesting" 001422505396
   ```

   make sure that the computer ID of your license key matches the computer ID shown in the computer ID field of the License Manager.

   Action: If the computer IDs do not match, you will need to **request a new license from the** *License Administrator* (see below). Please supply the host ID from the License Manager dialog as well as your existing license key for reference.

3. If the above suggestions do not resolve the license problem, it will be necessary to debug the problem using environment variable TGS_LICENSE_DEBUG. See below for instructions.

4.6.5 Using TGS_LICENSE_DEBUG

If the TGS_LICENSE_DEBUG environment variable is set to a file name, when Amira is run, licensing debug output is written to the specified file. You can open

Medivis Exhibit 1023

and read this debug file yourself. However, most likely you will need to send it to the *technical support team* (see below) for analysis.
Below are instructions for setting this environment variable on *Windows* and on *Linux/UNIX* systems.

4.6.5.1 Windows

On Windows, you can set the environment variable via the Control Panel or you can set it in a command prompt.
Windows - Control Panel

1. You can get to the Control Panel via the Windows Start menu.
2. In the Control Panel, select the System application.
3. Click on the Advanced tab.
4. Press the Environment variables button.
5. In either the User or System variables, press the New button.
6. For the Variable name enter TGS_LICENSE_DEBUG.
7. For the Variable value enter ”debug.txt” (without the quotes).
8. Click all of the OK buttons to exit the dialog.
9. Run Amira from the Start menu. When the License Manager dialog comes up, dismiss it.
10. Then look for a file named ”debug.txt” in the top level of the Amira installation directory. Send that file to *tech support* for analysis.

Windows - Command Prompt

1. On Windows 2000 and Windows XP you can get to a command prompt via the Windows Start menu as follows: Start > Programs > Accessories > Command Prompt
2. In the command prompt window, type:
   `set TGS_LICENSE_DEBUG=debug.txt`
3. Change the current directory to where your Amira executable is. For example,
   `cd /Program Files/Amira5/bin/arch-Win32VC8-Optimize`
4. Run Amira as follows:
   Amira`main.exe`
5. When the License Manager dialog comes up, dismiss it.

Medivis Exhibit 1023

6. Look for the file debug.txt in the current directory. Send that file to *tech support* for analysis.

4.6.5.2 On Linux/UNIX

1. On Linux/UNIX systems, the exact command to use for setting environment variables depends on the kind of shell you are using. Here are examples for a few commonly used shells. Bash shell:
   `export TGS_LICENSE_DEBUG=debug.txt` C shell:
   `setenv TGS_LICENSE_DEBUG debug.txt`
2. In the window in which you set the environment variable, cd to the directory containing the Amira executable. For example,

   ```
   cd /opt/Amira5
   ```
3. Run Amira as follows:

   ```
   bin/start
   ```
4. When the License Manager dialog comes up, dismiss it.
5. Look for the file debug.txt in the current directory. Send that file to *tech support* for analysis.

   NOTE: If you don't see the debug.txt file there, possibly you do not have privilege to write into the Amira installation directory. In this case, set TGS_LICENSE_DEBUG to a path where you can write. For example, /home/your_login_name/debug.txt

4.6.6 Contacting the License Administrator

From North, South, and Central America
amiralicense@visageimaging.com
From Europe, Middle East, Africa, Asia, and Australasia
amiralicense@visageimaging.com

4.6.7 Contacting Technical Support

From North, South, and Central America
amirasupport@visageimaging.com

Medivis Exhibit 1023

From Europe, Middle East, Africa, Asia, and Australasia
amirasupport@visageimaging.com

4.7 Notes for Mac users

The user will find several important differences when comparing the Mac to the Windows version. Due to the platform architecture, the Mac version requires specific implementation not only for the interface components, but also for the modules functionalities. Additionally, the tutorials have been developed on Windows platforms. Therefore, the user has to keep in mind the following issues when reading the examples of the tutorials and whenever using Amira:

- Ctrl Key - The *Ctrl key* of Windows and Linux corresponds to the *Apple/Command* key on Mac platforms.
- Stereo - Stereo visualization is not fully supported on Mac, due to Open Inventor compatibilities. Only few stereo modes are, therefore, available.
- Main Menu - The main menu on Mac has been redesigned according to the native style. The first item in the main menu is *application item* "Amira", where application-specific information are available as *About Amira* or *Preferences...*.
- Graphic performances - High-end graphic boards are required to use the *Volren* module and the editors using this visualization technique. Some iMac and older Mac platforms may not have adequate graphic cards.
- amiraDev - After developing new modules using the amiraDev Package, start Amira normally to use or test them. Starting Amira in "debug mode" is not required for Mac platforms. See the amiraDev documentation for details.

4.8 Amira and the /3GB switch

This page describes problems and possible solutions related to the usage of large data sets in Amira 5 on computers running Windows XP. If you frequently encounter dialogs in Amira such as "cannot allocate xxxxxx bytes of memory" although you think you have enough physical memory installed, then the information provided on this page might be useful for you.

Medivis Exhibit 1023

4.8.1 The Problem

Any 32-bit operating system such as Windows (NT4.0, 2000, XP) or Linux can manage at most 4 gigabytes (GB) of memory. Windows divides this addressable space into 2 GB reserved for usage by the system only and 2 GB for any user application. Therefore, a single application can use at maximum 2 GB of memory. Note that this is independent of the actual physical memory, i.e., it does not matter whether the system has 1, 2, 3 or 4 GB of memory installed. Within the 2 GB reserved for the user, the software itself (executable and all of its DLLs) and all of the data has to fit. This makes clear that it is impossible for a Windows program to load 2 GB of data at a time.
To overcome this, Microsoft has provided a boot option for Windows that alters partitioning of address space so that 3 GB are reserved for the user and 1 GB for the system. This boot option is available starting with Windows XP Service Pack 2.

4.8.2 How to activate the /3GB switch

It should be noted that we cannot guarantee that the following changes to be made in your system will not affect the execution of other programs although we have not found any incompatibilities so far. We also wish to stress that we assume no responsibility for any loss of data as a consequence of those changes. *We therefore strongly recommend that you back up your system before making these changes.*
All you need to do is to edit the file C:/boot.ini. To do so, make sure that you are logged on your machine with administrative privileges. Then select from Start->Settings->Control Panel->System->Advanced->(Startup and Recovery) Settings and click the Edit button. Duplicate the line under [operating systems]. Change the name between the quotes and add the ”/3GB” switch right after the ”/fastdetect” switch. The new entry might look then similar to this:

```
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP \
                                Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP \
                                Professional 3GB" /fastdetect /3GB
```

Note that for the usage of the /3GB boot option the application must be aware of the altered memory assignment. For Amira 5 there is already native support for the /3GB boot option. If you are using an older version of Amira, we strongly recommend updating to 5.

Medivis Exhibit 1023

Now, as you boot up your computer, a screen is displayed that gives you the choice between two boot options. Use the arrow keys to select the entry with the ”3GB” in the line and press return. Amira should now be able to address 3 GB of memory instead of the default 2 GB. You can appreciate this by comparing the result of ”app maxmalloc” in the Amira console window with and without the 3 GB option. On a typical installation with 1 GB of memory installed and without the /3GB switch, the command ”app maxmalloc” may result in something like this:

```
Allocated chunk of 942 MB.
Allocated chunk of 225 MB.
Allocated chunk of 129 MB.
Allocated chunk of 121 MB.
Allocated chunk of 72 MB.
Allocated a total of 1723 MB in 25 chunks
```

With the /3GB switch on the same installation, the command results in the following:

```
Allocated chunk of 1023 MB.
Allocated chunk of 919 MB.
Allocated chunk of 225 MB.
Allocated chunk of 111 MB.
Allocated a total of 2278 MB in 4 chunks
```

What you can see is that the total amount of allocatable memory has not changed dramatically. However, with the /3GB switch, the second largest chunk of memory is nearly as large as the first one.

Medivis Exhibit 1023

5 Scripting

5.1 Introduction

This chapter is intended for advanced Amira users only. If you do not know what scripting is, it is very likely that you will not need the features described in this chapter.

Beside the interactive control via the graphical user interface, most of the Amira functionality can also be accessed using specific commands. This allows you to automate certain processes and to create scripts for managing routine tasks or for presenting demos. Amira's scripting commands are based on Tcl, the *Tool Command Language*. This means you can write command scripts using Tcl with Amira-specific extensions.

Amira commands can be typed into the Amira console window, as described in Section 3.1.10. Commands typed directly into the console window will be executed immediately. Alternatively, commands can be written into a text file, which can then be executed as a whole.

This chapter is organized as follows:

Section 5.2 (Introduction to Tcl) gives a short introduction to the Tcl scripting language. This section is not very Amira specific.

Section 5.3 (Amira Script Interface) explains Amira-specific commands and concepts related to scripting. This includes a *reference of global commands*.

Section 5.4 (Amira Script Files) explains the different ways of writing and executing script files including references to script objects, resource files, and function-key bound Tcl procedures.

Section 5.5 (Configuring Popup Menus) describes how the popup menu of an object can be configured using script commands, and how new entries causing a script to be executed can be created.

Section 5.6 (Registering pick callbacks) describes how script callbacks can be attached to objects or viewers and be invoked on user pick events.

Data Type: Script Object describes how to use Tcl scripts for defining custom modules that have their own graphical user interface and can be used like the built-

Medivis Exhibit 1023

in Amira objects.

5.2 Introduction to Tcl

This chapter gives a brief introduction to the Tcl scripting language. If you are familiar with Tcl you can skip this section. However, please notice that instead of the `puts` command, you should use `echo` for output to the Amira console.
This chapter is not intended to cover all details of the language. For a complete documentation or reference manual of the Tcl language, refer to a text book like *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.
Alternatively you can easily find Tcl documentation and reference manuals on the internet e.g., at `www.scriptics.com`, or looking up keywords like `Tcl tutorial` or `Tcl documentation` with a search engine like `www.google.com`.
When you type Tcl commands into the Amira console, they will be executed as soon as the return key is pressed. Use the completion and history functions provided by the Amira console, as described in Section 3.1.10 (console window).

5.2.1 Tcl Lists, Commands, Comments

First, please note that Tcl is case sensitive: `set` and `Set` are not the same.
A Tcl command is a space-separated list of words. The first word represents the command name, all further words are treated as arguments to that command. As an example, try the Amira-specific command `echo`, which will print all its arguments to the Amira console. Try typing

```
echo Hello World
```

This will output the string *Hello World*. Note that Tcl commands can be separated by a semi-colon (;) or a newline character. If you want to execute two successive `echo` commands, you can do it this way:

```
echo Hello World ; echo Hello World2
```

or like this:

Medivis Exhibit 1023

```
echo Hello World
echo Hello World2
```

Instead of a command, you can also place a comment in Tcl code. A comment starts with a hash character (#) and is ended by the next line break:

```
# this is a comment
echo Hello World
```

5.2.2 Tcl Variables

In Tcl, variables can be used. A variable represents a certain state or value. Using Tcl code, the value of the placeholder can be queried, defined, and modified. To define a variable use the command

```
set name value
```

e.g.

```
set i 1
set myVar foobar
```

Note that in Tcl internally all variables are of string type. Since the set command requires exactly one argument as the variable value, you have to quote values that contain spaces:

```
set Output "Hello World"
```

or

```
set Output {Hello World}
```

In order to substitute the value of a variable with name *varname*, a $ sign has to be put in front of that name. The expression $varname will be replaced by the value of the variable. After the above definitions,

```
echo $Output
```

would print

```
Hello World
```


Medivis Exhibit 1023

in the console window, and

```
echo "$i.) $Output"
```

would yield the output *1.) Hello World*. Note that variable substitution is performed for strings quoted in `"`, while it is not done for strings enclosed in braces {}. Even newline characters are allowed in a { } enclosed string. Note however that it is not possible to type in multi-line commands into the Amira console.

5.2.3 Tcl Command Substitution

To do mathematical computations in Tcl, you can use the command `expr` which will evaluate its arguments and return the value of the expression. Examples are:

```
expr 5 / ( 7 + 3)
expr $i + 1
```

In order to use the result of a command like `expr` for further commands, an important Tcl mechanism has to be used: command substitution, denoted by brackets []. Any list enclosed in brackets [] will be executed as a separate command first, and the [...] construct will be replaced with the *result* of the command. This is similar to the '...' construct in Unix command shells. For example, in order to increase the value of the variable i by one you can use:

```
set i [expr $i + 1]
```

Of course, command expressions can be arbitrarily nested. The order of execution is always from the innermost bracket pair to the outermost one:

```
echo [expr 5 * [expr 7 + [expr 3+3]]]
```

5.2.4 Tcl Control Structures

Further important language elements are `if-else` constructs, `for` loops and `while` loops. These constructs typically are multi-line constructs and therefore can not be typed conveniently into the Amira console. If you want to try the examples shown below, write them into a file like `C:\test.txt` by using a text editor of your choice, and execute the file by typing

```
source C:\test.txt
```


Medivis Exhibit 1023

We start with the `if-then` mechanism. It is used to execute some code *conditionally*, only if a certain *expression* evaluates to ”true” (meaning a value different from 0):

```
set a 7
set b 8
if {$a < $b} {
    echo "$a is smaller than $b"
} elseif {$a == $b} {
    echo "$a equals $b"
} else {
    echo "$a is greater than $b"
}
```

The `elseif` and `else` parts are optional. Multiple `elseif` parts can be used, but only a single `if` and `else` part.
Another important construct is the conditional loop. Like the `if` command, it is based on checking a conditional expression. In contrast to `if`, the conditional code is executed multiple times, as long as the expression evaluates to true:

```
for {set i 1} {$i < 100} {set i [expr $i*2]} {
    echo $i
}
```

In fact this code is identical to:

```
set i 1
while {$i < $100} {
    echo $i
    set $i [expr $i * 2]
}
```

Both loops would produce the output *1, 2, 4, 8, 16, 32, 64*.
If you want to execute a loop for all elements of a list, there is another very convenient command for that:

```
foreach x {1 2 4 8 16 32 64} {
  echo $x
}
```

This will generate the same output as the previous example. Note that the expression enclosed in braces is a space-separated list of words.

Medivis Exhibit 1023

5.2.5 User-Defined Tcl Procedures

A new function or procedure is defined in Tcl using the `proc` command. Proc takes two arguments: a list of argument names and the Tcl code to be executed. Once a procedure is defined, it can be used just like any other Tcl command:

```
proc computeAverageA {a b} {
    return [expr ($a+$b)/2.0]
}
proc computeAverageB {a b c} {
    return [expr ($a+$b+$c)/3.0]
}
echo "average of 2 and 3: [computeAverageA 2 3]"
echo "average of 2,3,4: [computeAverageB 2 3 4]"
```

As you can see in the example, the argument list defines the names for local variables that can be used in the body of the procedure (e.g. `$a`). The `return` command is used to define the result of the procedure. This result is the value that is used in the command bracket substitution `[]`.
If you want to define a procedure with a flexible number of arguments, you must use the special argument name `args`. If the argument list contains just this word, the newly defined command will accept an arbitrary number of arguments, and these arguments are passed as a *list* called `args`:

```
proc computeAverage args {
    set result 0
    foreach x $args {
        set result [expr $result + $x]
    }
    return [expr $result / [llength $args]]
}
```

In this example, the `llength` command returns the number of elements contained in the `args` list.
Note that the variable `result` defined in the procedure has *local* scope, meaning that it will not be known outside the body of the procedure. Also, the value of globally defined variables is not known within a procedure unless that global variable is declared using the keyword `global`:

Medivis Exhibit 1023

```
set x 3
proc printX {} {
    global x
    echo "the value of x is $x"
}
```

There is much more to be said about procedures, e.g., concerning argument passing, evaluation of commands in the context outside of the procedure, and so on. Please refer to a Tcl reference book for these advanced topics.

5.2.6 List and String Manipulation

Finally, at the end of this brief Tcl introduction, we come back to the concept of lists. Basically everything in Tcl is constructed using lists, so it is very important to know the most important list manipulation commands as well as to understand some subtle details.
Here is an example of how to take an input list of numbers and construct an output list in which each element is twice as big as the corresponding element in the input list:

```
set input [list 1 2 3 4 5]
set output [list]
foreach element $input {
  lappend output [expr $element * 2]
}
```

You can think of lists as simple strings in which the list elements are separated by spaces. This means that you can achieve the same result as in the previous example without using the list commands:

```
set input "1 2 3 4 5"
set output ""
foreach element $input {
  append output "[expr $element * 2] "
}
```

The *append* command is similar to *lappend*, but it just adds a string at the end of an existing string. List manipulation becomes much more involved when you start nesting lists. Nested lists are represented using nested pairs of braces, e.g.

Medivis Exhibit 1023

```
set input {1 2 {3 4 5 {6 7} 8 } 9}
foreach x $input {
  echo $x
}
```

The result of this command will be

```
1
2
3 4 5 {6 7} 8
9
```

Please note that Tcl will automatically *quote* strings that are not single words when constructing a list. Here is an example:

```
set i [list 1 2 3]
lappend i "4 5 6"
echo $i
```

will yield the output

```
1 2 3 {4 5 6}
```

You can use the *lindex* command to access a single element of a list. *lindex* takes two arguments: the list and the index number of the desired element, starting with 0:

```
set i [list a b c d e]
echo [lindex $i 2]
```

will yield the result `c`.

5.3 Amira Script Interface

Although the Tcl language is not intrinsically object oriented, the Amira script interface is. There is one command for each object in the Amira Pool. In addition there are several *global commands* associated with global objects in Amira such as the *viewer* or the Amira.
A command associated with an object in the Pool (e.g., an OrthoSlice module or an Isosurface module) only exists while the object exists. These commands are

Medivis Exhibit 1023

identical to the name of the object as displayed in the Pool. Typically the script interface of a specific object contains many different functions. The general syntax for an Amira object-related command is

```
<object-name> <command-word> <optional-arguments> ...
```

For example, if an object called GlobalAxis exists (choose View/Axis from the Amira menu) then you can use commands like

```
GlobalAxis deselect
GlobalAxis select
GlobalAxis setIconPosition 100 100
```

Remember to use the completion and history functions provided by the Amira console, as described in Section 3.1.10 (console window) to save typing.
If you have already used Amira you have noticed that the parameters and the behavior of an Amira module are controlled via its *ports*. The ports provide a user interface to change their values when the module is selected. All ports can also be controlled via the command interface. The general syntax for that is

```
<object-name> <port-name> <port-command> <optional-arguments> ...
```

For example for the GlobalAxis you can type

```
GlobalAxis options setValue 1 1
GlobalAxis thickness setValue 1.5
GlobalAxis fire
```

When you type in these commands you will notice that the values in the user interface change immediately. However the module's compute method is not called until explicitly firing the module using the `fire` command. This allows you to first set values for multiple ports without a recomputation after each command. However, note that some modules automatically reset some of their ports for example when a new input object is connected. In such cases you may need to call `fire` after setting the value of every single port.
Usually the name of a port is identical to the text label displayed in the graphical user interface, except that white spaces are removed and command names start with a lower case letter. To find out the names of all ports of a specific module use the command

Medivis Exhibit 1023

```
<object-name> allPorts
```

Almost all ports provide a `setValue` and a `getValue` command. The number of parameters and the syntax, of course, depend on the ports.

Commands of the type `<object-name> <port-name> setValue ...` make up more than 90% of a typical Amira script. However, besides the port commands, many Amira objects provide additional specific commands. The command interface for a particular module is described in the User's Reference Guide. You can quickly find the corresponding page by clicking the ? button in the Properties Area when the module has been selected.

As a quick help, entering an object's name without further options will display all commands available for that object. Note that this will also show undocumented, unreleased, and experimental commands. In order to get more information about a particular module or port command, you can type it into the console window without any arguments and then press the F1 key. This opens the help browser with a command description.

Amira objects are part of a class hierarchy. Similar to the C++ programming interface, also script commands are inherited by derived classes from its base classes. This means that a particular object like the axis object will beside its own specific commands also provide all the commands available in its base classes. Links to the base class commands are given in a module's documentation.

5.3.1 Predefined Variables

There exist some variables in Amira Tcl, which are predefined and have a special meaning. These are

- `AMIRA_ROOT`: Amira installation directory.
- `AMIRA_LOCAL`: Personal Amira development directory (Developer Option only).
- `SCRIPTFILE`: Tcl script file currently executed.
- `SCRIPTDIR`: Directory in which currently executed script resides.
- `hideNewModules`: If set to 1, icons of newly created modules will initially be hidden.

Medivis Exhibit 1023

5.3.2 Object commands

The basic command interface of Amira modules and data objects is described in the data type chapter of the reference part of the user's guide in the *Object* section. The basic syntax of object commands is

```
<object> <command> <arguments> ...
```

where `<object>` refers to the name of the object and `<command>` denotes the command to be executed. Each module or data object may define its own set of commands in addition to the commands defined by its base classes. The commands described in the *Object* section are provided by all modules and data objects.
In the following section *Global commands* are described.

5.3.3 Global Commands

This section lists Amira-specific global Tcl commands. Some of these commands are associated with certain global objects in Amira, such as the console window, the main window, or the viewer window. Other commands are such as `load` or `echo` are not. These commands are described in one common subsection. In summary, the following command sections are provided:

- *viewer command options* (`viewer`)
- *main window command options* (`theMain`)
- *console command options* (`theMsg`)
- *common commands for top-level windows*
- *progress bar command options* (`workArea`)
- *application command options* (`app`)
- *other global commands*

5.3.3.1 Viewer command options

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.

Medivis Exhibit 1023

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>`

(in top-level mode only) Sets the position of the viewer window relative to the upper left corner of the screen. If more than one viewer is shown in the same window the position of the toplevel window is set.

`viewer [<number>] getPosition`

Returns the position of the viewer window. If more than one viewer is shown in the same window the position of the toplevel window is returned.

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

(in top-level mode only) Sets the size of the viewer window. Width and height specify the size of the actual graphics area. The window size might be a little bit larger because of the viewer decoration and the window frame.

`viewer [<number>] getSize`

Returns the size of the viewer window without decoration and window frame.

Medivis Exhibit 1023

`viewer [<number>] setCamera <camera-string>`
Restores all camera settings. The camera string should be the output of a `getCamera` command.

`viewer [<number>] getCamera`
This command returns the current camera settings, i.e., position, orientation, focal distance, type, and height angle (for perspective cameras) or height (for orthographic cameras). The values are returned as Amira commands, which can be executed in order to restore the camera settings. The complete command string may also be passed to `setCamera` at once.

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

`viewer [<number>] setCameraPosition <x> <y> <z>`
Returns 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>] getCameraOrientation`
Returns the current orientation of the camera in the same format used by `setCameraOrientation`.

`viewer [<number>] setCameraFocalDistance <value>`
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>] getCameraFocalDistance`
Returns the current focal distance of the camera.

`viewer [<number>] setCameraHeightAngle <degrees>`
Sets the height angle of a perspective camera in degrees. Making the angle smaller makes the field of view smaller, effectively "zooming in", as with a telephoto lens. Unless you specifically want to change the camera field of view, it is normally better to move the camera closer to an object (sometimes called

”dolly in”) to make the object appear larger. The command has no effect if the current camera is an orthographic one.

`viewer [<number>] getCameraHeightAngle`
Returns the height angle of a perspective camera.

`viewer [<number>] setCameraHeight <height>`
Sets the height of the view volume of an orthographic camera. The command has no effect if the camera is an perspective one.

`viewer [<number>] getCameraHeight`
Returns the height of an orthographic camera.

`viewer [<number>] setCameraType <perspective|orthographic>`
Sets the camera type.

`viewer [<number>] getCameraType`
Returns the camera type.

`viewer [<number>] setTransparencyType <type>`
This command defines the strategy used for rendering transparent objects. The argument *type* may be a number between 0 and 8, corresponding to the entries *Screen Door*, *Add*, *Add Delay*, *Add Sorted*, *Blend*, *Blend Delay*, *Blend Sorted*, *Sorted Layers* and *Sorted Layers Delayed* as described for the *View menu.*

Most accurate results are obtained using mode 8. Default is mode 6. In the default mode, 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>] setSortedLayersNumPasses <value>`
Sets the number of rendering passes used when transparency type is *Sorted Layers* or *Sorted Layers Delayed.* Use more passes for more correct transparency. Usually four passes (which is the default value) gives good results.

Medivis Exhibit 1023 206

`viewer [<number>] getSortedLayersNumPasses`
Returns the number of rendering passes used when transparency type is *Sorted Layers* or *Sorted Layers Delayed*.

`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>] getBackgroundColor`
Returns the primary background color as an RGB triple with values between 0 and 1.

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

`viewer [<number>] getBackgroundColor2`
Returns the secondary background color as an RGB triple with values between 0 and 1.

`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. Mode 2 causes a checkerboard pattern to be displayed. This might help to understand the shape of transparent objects. Finally, mode 3 draws an image previously defined with `setBackgroundImage` on the background.

`viewer getBackgroundMode`
Returns the current background mode.

`viewer setBackgroundImage <imagefile> [<imagefile2>] [-stereo]`
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. Otherwise it will be clipped. 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

Medivis Exhibit 1023

PostScript. If a second image file is specified, this file will be used as the right eye image in case of active stereo rendering. If the options `-stereo` is specified and only one image file is given, it it assumed that this file contains a left eye view and a right eye view composited side by side. These views then will be separated automatically.

`viewer getBackgroundImage`
This command returns the file name of the last background image file defined with `setBackgroundImage`. If a pair of stereo images was specified, two file names are returned. If the option `-stereo` was used in `setBackgroundImage`, this option will be returned too.

`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>] isAutoRedraw`
Returns true is auto redraw mode is on.

`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> [x|y|z|m|u|v]`
Rotates the camera around an axis. The axis to be taken is specified by the second argument. The following choices are available:

- x: the x-axis (1,0,0)
- y: the y-axis (0,1,0
- z: the z-axis (0,0,1)
- m: the most vertical axis of x, y, or z
- u: the viewer's up direction
- v: the view direction

The last option does the same as the rotate button of the user interface. In most cases the *m* option is most adequate. For backward-compatibility the default is *u*.

Medivis Exhibit 1023

`viewer [<number>] setDecoration <state>`
Deprecated.

`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 Open 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>] 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.

`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).

Medivis Exhibit 1023

5.3.3.2 Main window command options

The command `theMain` allows you to access and control the Amira main window. Besides the specific command options listed below also all sub-commands listed in Section 5.3.3.4 (Common commands for top-level windows) can be used.

Commands

`theMain snapshot` *filename*
Creates and saves a snapshot image of the main window. The format of the image file is determined from the file name extension. Any standard image file format supported by Amira can be used, e.g., `.jpg`, `.tif`, `.png`, or `.bmp`.

`theMain setViewerTogglesOnIcons {0|1}`
Enables or disables the display of the orange viewer toggles on object icons in the Amira Pool.

`theMain ignoreShow [0|1]`
Enables or disables the special purpose *no show flag*. If this flag is set, subsequent `mainWindow show` commands are ignored. This can be useful to run standard Amira scripts in a Virtual Reality Option environment. Calling the command without an argument just returns the current value of the flag.

5.3.3.3 Console command options

The command `theMsg` allows you to access and control the Amira console window. Besides the specific command options listed below also all sub-commands listed in Section 5.3.3.4 (Common commands for top-level windows) can be used.

Commands

`theMsg error <message> [<btn0-text>] [<btn1-text>] [<btn2-text>]`
Pops up an error dialog with the specified message. The dialog can be configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg warning <message> [<btn0-text>] [<btn1-text>] [<btn2-text>]`
Pops up a warning dialog with the specified message. The dialog can be

Medivis Exhibit 1023

configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg question <message> [<btn0-text>] [<btn1-text>] [<btn2-text>]`
Pops up a question dialog with the specified message. The dialog can be configured with up to three different buttons. The command blocks until the user presses a button. The id of the pressed button is returned.

`theMsg overwrite <filename>`
Pops up a dialog asking the user if it is ok to overwrite the specified file. If the user clicks *Ok*, 1 is returned, otherwise 0.

5.3.3.4 Common commands for top-level windows

These commands are available for all Amira objects which open a separate top-level window. In particular, these are the Amira main window (`theMain`), the console window (`theMsg`), and the viewer window (`viewer 0`). For example, you can set or get the position of these windows using the corresponding global command followed by `setPosition` or `getPosition`.

Commands

`getFrameGeometry`
Returns the position and size of the window including the window frame. In total four numbers are returned. The first two numbers indicate the position of the upper left corner of the window frame relative to the upper left corner of the desktop. The last two numbers indicate the window size in pixels.

`getGeometry`
Returns the position and size of the window without the window frame. In total four numbers are returned. The first two numbers indicate the position of the upper left corner of the window relative to the upper left corner of the desktop. The last two numbers indicate the window size in pixels.

`getPosition`
Returns the position of the upper left corner of the window including the window frame. This is the same as the first two numbers returned by `getFrameGeometry`.

Medivis Exhibit 1023

`getRelativeGeometry`
Returns the position and size of the window including the window frame in relative coordinates. The size of the desktop is (1,1). The position and size of the window is specified by fractional numbers between 0 and 1.

`getSize`
Returns the size of the window without the window frame. This is the same as the last two numbers returned by `getGeoemtry`.

`hide`
Hides the window.

`setCaption <text>`
Sets the window title displayed in the window frame.

`setFrameGeometry <x y width height>`
Sets the position and size of the window including the window frame. Four numbers need to be specified, the x- and y-positions, the window width and the window height.

`setGeometry <x y width height>`
Sets the position and size of the window without the window frame. Four numbers need to be specified, the x- and y-positions, the window width and the window height.

`setPosition <x y>`
Sets the position of the upper left corner of the window frame.

`setRelativeGeometry <x y width height>`
Sets the position and size of the window including the window frame in relative coordinates. The size of the desktop is (1,1). The position and size of the window is specified by fractional numbers between 0 and 1.

`setSize <width height>`
Sets the size of the window without the window frame.

`show`
Makes the window visible in normal state. Also raises the window.

`showMinimized`
Makes the window visible in iconified state.

Medivis Exhibit 1023 212

`showMaximized`
Makes the window visible in maximized state.

5.3.3.5 Progress bar command options

The command `workArea` allows you to access the progress bar located in the lower part of the Amira main window. You can print messages or check if the stop button was pressed.

Commands

`workArea setProgressInfo <text>`
Sets an info text to be displayed in the progress bar. The text can be used to describe the status during some computation.

`workArea setProgressValue <value>`
Sets the value of the progress bar. The argument must be a floating point number between 0 and 1. For example, a value of 0.8 indicates that 80% of the current task has been done.

`workArea startWorking [<message>]`
Activates the stop button. After calling this command the Amira stop button becomes active. In your script you can check if the stop button was hit by calling `workArea wasInterrupted`. When the stop button is active you can't interact with any other widget unless you call `workArea stopWorking` in your script. Therefore you must not enter this command directly in the console window, but you should only use it in a script file or in a Tcl procedure.

`workArea stopWorking`
Deactivates the stop button. Call this command when the compute task started with `workArea startWorking` is done or if the user pressed the stop button. This command also restores the progress info text which was shown before calling `startWorking`.

`workArea wasInterrupted`
Checks if the user pressed the stop button. You should only use this command between `workArea startWorking` and `workArea stopWorking`. If there are multiple nested compute tasks and the user presses the stop button, all subsequent calls to `wasInterrupted` return true until the first level is reached.

Medivis Exhibit 1023

5.3.3.6 Application command options

The `app` command provides several options not related to a particular object or component in Amira, but related to Amira itself.

Commands

`app version`
Returns the current Amira version.

`app uname`
Returns simplified name of operating system.

`app arch`
Returns Amira architecture string, e.g., arch-Win32VC8-Optimize, arch-LinuxAMD64-Optimize...

`app hostid`
Returns host id needed to create an Amira license key.

`app listen [port]`
Opens a socket to which Tcl commands can be sent. The TCP/IP port can be specified optionally. WARNING: This can create security holes. Do not use this unless behind a firewall and if you know what you are doing.

`app close`
Closes the Amira Tcl port.

`app port`
Returns port number of Amira Tcl port. Returns -1 if socket has not been opened.

`app send <command> [<host> [<port>]]`
Sends a Tcl command to a listening Amira. If no host or port are specified, the Amira instance will send the command to itself.

`app opengl`
Retrieve information about the used OpenGL driver including version number and supported extensions. This is useful information to send to the hotline if reporting rendering problems.

Medivis Exhibit 1023

`app cluster`
Returns the current node status which can be ”master” or ”slave” if some cluster mode is active or simply ”single” if is not the case.

5.3.3.7 Other global commands

Commands

`addTimeout` *msec procedure* [*arg*]
Schedules a Tcl procedure for being called after *msec* milliseconds. If *arg* is specified, it will be passed to the procedure. The specified procedure will be called only once. If necessary, you can schedule it again in the time-out procedure. Example: `addTimeout 10000 echo {10 seconds are over.}`

`all [-selected | -visible | -hidden]` [*type*]
Returns a list of all Amira objects currently in the Pool. If *type* is specified, only objects with that C++ class type (or derived objects) are returned. Search can be limited to selected, visible, or hidden objects, respectively. Example: `all -hidden HxColormap`.

`aminfo [-a` *outfile*`|-b` *outfile*`]` *AmiraMesh-File*
If used with only a file name as argument, this command will open the file which has to be in AmiraMesh format and print header information. If used with the -a or -b option, the outputfile specified as argument *outfile* will be written in ASCII (-a) or binary (-b) format, respectively. Thus, *aminfo* can be used to convert binary AmiraMesh into ASCII and vice versa.

`clear`
Clears console window.

`create` *class name* [*instance name*]
Creates an instance of an Amira object like a module or data object. Returns the instance name. Note that data objects are normally not created this way but by loading them from a file. Example: `create HxOrthoSlice MySlice`.

`dso` *options*
Controls loading of dynamic libraries. The following options are provided:

- `addPath path ...` Adds a path to the list of directories to be searched when loading a dynamic library.

- `verbose {0|1}` Switches on and off debug information related to dynamic shared object loading.
- `open <package>` Trys to load the specified dynamic library. It is enough to specify the package name, e.g., `hxfield`. This name will be automatically converted into the platform dependent name, e.g., `libhxfield.so` on Linux or `hxfield.dll` on Windows.

`echo` *args*
Prints its arguments to the Amira console. Use this rather than the native Tcl command `puts` which prints to stdout.

`help` *arguments*
Without arguments this opens the Amira help browser.

`httpd [port]`
Start a built-in httpd server. The http server will deliver any document requested. If a requested document ends with `.hx`, Amira will instead of delivering it execute the file as a Tcl script. This can be used to control Amira from a web browser. WARNING: This command can create security holes. Do not use this unless behind a firewall and if you know what you are doing.

`limit {datasize | stacksize | coredumpsize}` *size*
Change process limits. Available on Unix platforms only. Use "unlimited" as size for no limit. The size has to be specified in bytes. Alternatively you can use for example 1000k for 1000 kilobytes or 1m for one megabyte.

`load [fileformat]` *files*
Load data from one or more files. Optionally a file format can be specified to override Amira's automatic file format recognition. The file format is specified by the same label which is displayed in the file format combo box in the Amira file dialog.

`mem`
Prints out some memory statistics (available on Windows and Irix).

`quit`
Immediately quits Amira.

`remove {`*objectname — -all — -selected*`}`

Medivis Exhibit 1023 216

- `objectname` removes the specified Amira object.
- `-all` remove all objects.
- `-selected` remove selected objects.

`removeTimeout` *procedure* `[`*arg*`]`
Unschedules a Tcl procedure previously scheduled with addTimeout.

`rename` *objectname newname*
Changes instance name of an object. Identical to *objectname* `setLabel` *newname*, except that it returns 1 if successful, and nothing if unsuccessful.

`sleep` *sec*
Wait for *sec* seconds. Amira will not process events in that time.

`source` *filename*
Loads and executes Tcl commands from the specified file. If the script file contains the extension `.hx` the `load` command may be used as well.

`system` *command*
Execute an external program. Do not use this unless you know what you are doing.

5.4 Amira Script File

It is worth noticing that an Amira network is simply a Tcl script that will regenerate the current Amira state. Therefore it is often efficient to interactively create an Amira network, save it with "Save Network", and use this as a starting point for scripting.
The simplest way to execute Tcl commands in Amira is to type them into the Amira console window. This, however, is not practical for multi-line constructs, like loops or procedures. In this case, it is recommended to write the Tcl code into a file and execute the file with the command `source` *filename*. You can also use the `source` command inside a file in order to include the contents of a file into another file.
Alternatively one can also use the command `load` *filename* or the *Load* menu entry from the *File* menu and the file browser. Then, however, in order to let Amira recognize the file format, either the file name must end with `.hx`, or the file contents must start with the header line

Medivis Exhibit 1023

```
# Amira Script
```

There are some Tcl files that are loaded automatically when Amira starts. At startup, the program looks for a file called .Amira in the current directory or in the home directory (see Section 4.4 (Start-up script) for details). If no such user-defined start-up script is found, the default initialization script Amira.init is loaded from the directory $AMIRA_LOCAL/share/resources/Amira or $AMIRA_ROOT/share/resources/Amira. This script then reads in all files ending with .rc from the share/resources subdirectory. The .rc files are needed to register modules and data types. Therefore one can customize the startup behavior of Amira by simply adding a new .rc file to that directory or by modifying the Amira.init file.
Another way of executing Tcl code is to define procedures that are associated with function keys. If predefined procedures with the names onKeyF2, onKeyF3, ..., onKeyShiftF2, ..., onKeyCtrlF2, ..., onKeyCtrlShiftF2, ... exist, these procedures will be automatically called when the respective key is pressed in the Amira main window, console window, or viewer window. Note that F1 is reserved for help. To define these procedures, write them into a file and source it or write them into Amira.init or in one of the .rc files. An example is

```
proc onKeyF2 { } {
    echo "Key F2 was hit"
    viewer 0 viewAll
}
```

Finally, Tcl scripts can also be represented in the GUI and be combined with a user interface. In Amira this is called a *script object*. There is *separate documentation* for that.

5.5 Configuring Popup Menus

In Amira all of the modules that can be attached to a data object are listed in the object's popup menu which is activated by clicking on the object's icon with the right mouse button. For some applications it makes sense to customize new modules using Tcl commands after they have been created. Sometimes it also makes sense to add new entries to an object's popup menu, causing a particular script

Medivis Exhibit 1023

to be executed. This sections describes how to achieve these goals by modifying Amira resource files or creating new ones.

Amira resource files are located in the directory `$AMIRA_ROOT/share/resources`, where `$AMIRA_ROOT` denotes the directory where Amira has been installed. Resource files are just ordinary script files, although they are identified by the suffix `.rc`. When Amira is started all resource files in the resources directory are read. In a resource file, modules, editors, and IO routines are registered using special Tcl commands. Registering a module means to specify its name as it should appear in the popup menu, the type of objects it can be attached to, the name of the shared library or DLL the module is defined in, and so on. For example, the *LabelVoxel* module is registered by the following command in the file `hxlattice.rc`:

```
module -name "LabelVoxel" \
    -primary "HxUniformScalarField3 HxStackedScalarField3" \
    -check { ![$PRIMARY hasInterface HxLabelLattice3] } \
    -category "Labelling Compute" \
    -class "HxLabelVoxel" \
    -package "hxlattice"
```

The different options of this command have the following meaning:

- The option `-name` specifies the name or label of the module as it will be printed in the popup menu.
- The option `-primary` says that this module can be attached to data objects of type `HxUniformScalarField3` or `HxStackedScalarField3`. This means that *LabelVoxel* will be included in the popup menu of such objects only.
- With `-check` an additional Tcl expression is specified which is evaluated at run-time just before the menu is popped up. If the expression fails, the module is removed from the menu. In the case of the *LabelVoxel* module, it is checked if the input object provides a `HxLabelLattice3` interface, i.e., if the input itself is a label field. Although a label field can be regarded as a 3D image, it makes no sense to perform a threshold segmentation on it. Therefore *LabelVoxel* is only provided for raw 3D images, but not for label fields.
- The option `-category` says that *LabelVoxel* should appear in the *Compute* and *Labelling* submenus of the main popup menu. If a module should appear

Medivis Exhibit 1023

not in a submenu but in the popup menu itself, the category `Main` must be used.

- The option `-class` specifies the internal class name of the module. The internal class name of an object can be retrieved using the command `getTypeId`. It is this class name which has to be used for the `-primary` option described above, not the object's label defined by `-name`.
- Finally, the option `-package` specifies in which package (shared library or DLL) the module is defined in.

Besides these standard options, additional Tcl commands to be executed after the module has been created can be specified using the additional option `-proc`. For example, imagine you are working in a medical project where you have to identify stereotactic markers in CT images of the head. Then it might be a good idea to add a customized version of the *LabelVoxel* module to the popup menu, which already defines appropriate material names and thresholds. This could be done by adding the following command either in a new resource file in `$AMIRA_ROOT/share/resources` or directly in `hxlattice.rc`:

```
module -name "Stereotaxy" \
    -primary "HxUniformScalarField3 HxStackedScalarField3" \
    -check { ![$PRIMARY hasInterface HxLabelLattice3] } \
    -category "Labelling" \
    -class "HxLabelVoxel" \
    -package "hxlattice" \
    -proc { $this regions setValue "Exterior Bone Markers";
            $this fire;
            $this boundary01 setValue 150;
            $this boundary12 setvalue 300 }
```

The variable `$this` used in the Tcl code above refers to the newly created module, i.e., to the *LabelVoxel* module. Note that the commands are executed *before* the module is connected to the source object for which the popup menu was invoked. Some modules do some special initialization when they are connected to a new input object. These initializations may overwrite values set using Tcl commands defined by a custom `-proc` option. In such a case you can explicitly connect the module to the input object via the command sequence

```
$this data connect $PRIMARY;
$this fire;
```


Medivis Exhibit 1023

Here the Tcl variable `$PRIMARY` refers to the input object. The same variable is also used in Tcl expressions defined by a `-check` option, as described above.
Besides creating custom popup menu entries based on existing modules, it is also possible to define completely new entries which do nothing but execute Tcl commands. For example, we could add a new submenu *Edit* to the popup menu of every Amira object and put in the *Hide*, *Remove*, and *Duplicate* commands here which are normally contained in the *Edit* menu of the Amira main window. This can be achieved in the following way:

```
module -name "Remove" \
    -primary "HxObject" \
    -proc { remove $source } \
    -category "Edit"

module -name "Hide" \
    -primary "HxObject" \
    -proc { $source hideIcon } \
    -category "Edit"

module -name "Duplicate" \
    -primary "HxData" \
    -proc { $source duplicate } \
    -category "Edit"
```

Of course, it is also possible to execute an ordinary Amira script or even an Amira script object with a `-proc` command.

5.6 Registering pick callbacks

A pick callback is a Tcl procedure attached to a module or a viewer. When a pick event occurs on this target, the callback is invoked. Such a callback can be registered by using the Tcl command `setPickCallback` on modules and viewers:

```
<module> setPickCallback <proc> [ <EventType> ]
viewer <n> setPickCallback <proc> [ <EventType> ]
```

Only one callback can be attached to a given module or viewer. In order to detach the callback, just call the register command with no parameter:

Medivis Exhibit 1023

```
<module> setPickCallback
viewer <n> setPickCallback
```

The optional argument `<EventType>` refers to the kind of event that will invoke the callback. Other events will be ignored. This argument can take the following values:

- `MouseButtonPress`, `MouseButtonRelease` (any mouse button),
- `VRButtonPress`, `VRButtonRelease` (any 3D button),
- `MouseButton1Press`, `MouseButton1Release`, etc. (a specific mouse button),
- `VRButton0Press`, `VRButton0Release`, etc. (a specific 3D button).

The default value is `MouseButton1Press`.
The actual callback procedure `<proc>` is expected to take one argument, which is to be interpreted as an associative array and which encodes all the picking information. The following elements are defined in the argument array:

- *object*, the name of Amira object the picked geometry belongs to,
- *x*, the x coordinate of picked point,
- *y*, the y coordinate of picked point,
- *z*, the z coordinate of picked point,
- *idx*, the index of picked element,
- *stateBefore*, the modifier state just before event occurs,
- *stateAfter*, the modifier state just after event occurs.

The procedure should return 0 if the picking event was not handled, in which case other callback procedures could be invoked. Here is an example:

```
proc pickCallback arg {
    array set a $arg
    echo "$a(object) : picked element $a(idx)"
    return 1
}
```

Note that any module is free to add specific information to this argument array. All elements can be displayed with:

```
proc pickCallback arg {
```


Medivis Exhibit 1023

```
        echo "arg = { $arg }"
        return 1
    }
```

Thus, some Amira modules append additional data:

- *VertexView: idx* is the picked point index.
- *ClusterView: idx* is the picked point index.
- *LineSetView: idx* is the picked line index, *pt0* and *pt1* the two points of the picked segment.
- *SurfaceView: idx* is the picked triangle index.
- *GridVolume: idx* is the picked triangle index, *tetra0* and *tetra1* the adjacent tetrahedra.
- *GridBoundary: idx* is the picked triangle index, *originalIdx* the index in the grid, *tetra0* and *tetra1* the adjacent tetrahedra.

Medivis Exhibit 1023

6 Demo Framework

Demo Framework adds a generalized framework for creating and steering demonstrations with Amira.
Demonstrations are often prepared to present results of ones work to others. Speaking in terms of Amira, this means that scripts and data sets, e.g. .hx files and AmiraMesh files have been compiled and stored for presentation. After a while lots of scripts and their corresponding data files may have been spread over many disks and computers by many people. Now it's time to rearrange the demos and make them useable by others.
Demo Framework enables you to put your demo collection in order. It incorporates four major parts:

- A *directory and file structure* containing the Amira scripts to execute the demos, their description and their data. The directory structure may include a grouping and can also resemble a project structure of the demos.

- Several scripts to *select demos* from the demo collection, utilities to download data from remote servers, helper scripts for demo steering, etc.

- A *GUI* for conveniently selecting and changing demo collections.

- *DemoSequence*, an Amira script object for steering the selected demos.

A nice and easy way to create demos is the *DemoMaker*, which is included in the base version of Amira. With the DemoMaker you can interactively create an animated sequence of actions of Amira modules.

6.1 Directory Structure and Files

In order to use Demo Framework, you must set environment variable `AMIRA_DEMOS` to the directory that contains the demos. This demo directory must

Medivis Exhibit 1023

have at least three subdirectories: Environment variable `AMIRA_ROOT` pointing to the directory where Amira has been installed, has to be set as well.

src/ The demo description files, Amira scripts, and thumbnail images must be stored here. Use subdirectories to organize the demos, for example, to reflect different projects or overall themes.

data/ This directory contains the data files in a subdirectory structure similar to src/. The Mesh Option files used in your demos must be stored here.

It may also have these subdirectories:

cacheddata/ This optional directory contains tar archives with the data sets for the demos in src/. It is filled when you download the data from a server.

output/ This is the default name of the folder where you find the selected demos if the selection has been done with the command-line tool.

selection/ Here you will find the selected demos if the selection has been done using the *Demo GUI*.

6.1.1 Directories and Files in src/

The directory structure is laid out recursively in order to group demos by themes or topics. Folders which group subfolders are called *container* or *project folders*, while folders containing the demos, i.e., the Amira scripts, are called *demo folders*. Due to their recursive structure, *project folders* can in turn contain *project folders*, but *demo folders* cannot contain any subfolders.
Folders are regarded either as a project or demo folder if they have a file named *description.xml*. Otherwise the directories are not considered to be part of the demo structure.
In order to set up a demo collection, create a directory below `$AMIRA_DEMOS/src` which reflects the structure of the subjects or projects you wish to prepare for demonstration. See figure 6.1 for example.
The above mentioned project folders are those folders which do not contain Amira scripts (.hx files). However, they must contain a `description.xml` file. Like

Medivis Exhibit 1023

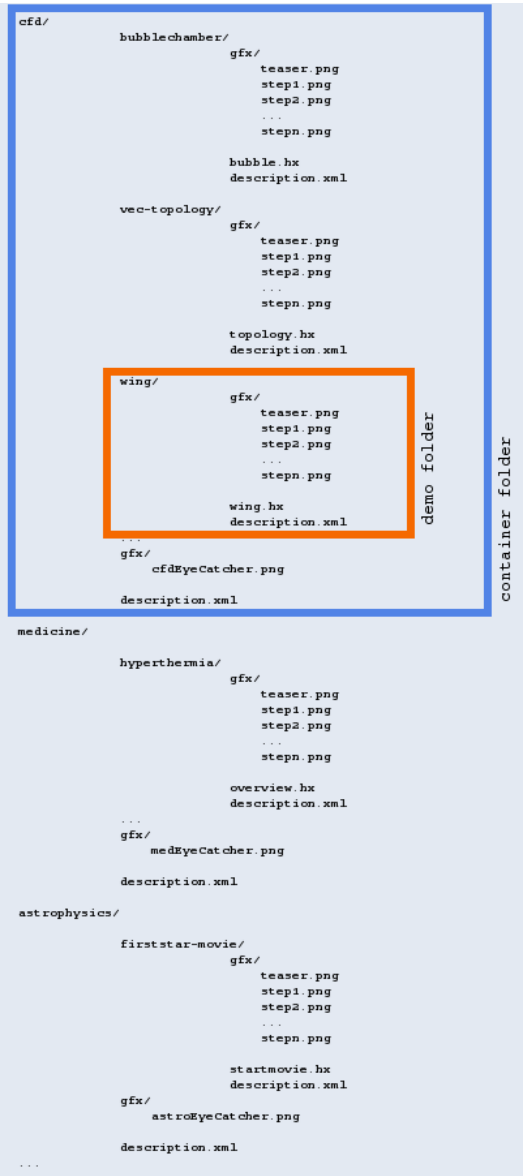


Figure 6.1: File structure below `$AMIRA_DEMOS/src`

every other directory in the demo structure, such folders have a `gfx` directory which in turn contains images for representing the project as a whole.
The associated description file for a project folder could look like the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<description style="project">
            <!-- Use style="projectlist" in the very first
                 description.xml file directly in the product
                 root directory! -->
   <title>
      Your Project Title
      <!-- (Keep this title _short_ and _meaningful_)!
           Don't start with "Visualization of" since this is certainly
           the case in all demos. Consider that finding your demo
           within a huge list of titles might become difficult. -->
   </title>
   <general>
      Your Short Description.
      <!-- Please, don't write a long story here - just two or three
           sentences. The text should guide an untrained presenter
           to explain your demo. -->
   </general>
   <!-- you might want to register a staff tag here -->
     <staff>
       <person>
         <name>Your Name</name>
         <userid>yourname</userid>
         <email>yourname@yourdomain</email>
         <tel>xxx</tel>
       </person>
       <!-- This information will give the presenter the chance to
            contact you in case your presence is desired or
            necessary. -->
       <!-- register more persons here -->
     </staff>
   <!-- Please provide a thumbnail for this demo
        (e.g., the teaser of your project page) -->
   <thumbnail src="gfx/eyecatcher.png"/>
</description>
```

Note: Use `projectlist` as the value to the `style` attribute for the description .xml file in the directory `$AMIRA_DEMOS`.
The final demo folders contain the Amira scripts, the `gfx` directory with one .png image for every demo step, and a description .xml file which may look similar to this:

Medivis Exhibit 1023

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <description style="demo">
    <attributes>
      <attribute name="display" values="normal" />
      <attribute name="version" values="zib-2006-2" />
      <attribute name="topic" values="flow" />
      <attribute name="state" values="released" />
    </attributes>
    <title>
        Vector Field in a Bubble Chamber
    </title>
    <general>
       In this demo several visualization methods
       for vector fields are shown.

       The data set comes from ...
    </general>
    <thumbnail src="gfx/teaser.png" />
    <staff>
      <person status="responsible">
         <name>Your Name</name>
         <userid>yourusername</userid>
         <email>youremailaddress</email>
         <tel>xxx</tel>
      </person>
      <!-- register more persons here -->
    </staff>
      <prepacked href="https://servername/prepackeddemodata">
        <file src="bubble\_2006.06.22-1_data.tar.gz"/>
      </prepacked>
      <demo>
        <script file="bubble.hx"/>
          <steps>
           <step name="Vectors" playcommand="vectors">
            Vector Arrows (For interactivity, pick plane
            and shift up or down)
           </step>
           <step name="LIC" playcommand="lic">
            Line Integral Convolution (LIC) (For interactivity
            pick plane and shift forward or backward)
           </step>
           <!-- add more steps here -->
          </steps>
      </demo>
  </description>
```

Note: You can use HTML tags to enhance titles and general descriptions within

the description file. The XSL parser is very strict, so always use a /> at the end for single tags like br or p. Also be aware of the quite cumbersome encoding details. If you are writing German with umlauts, be sure that your editor also stores these characters as ISO-8859-1 encoding, also known as Latin1. Otherwise you will get surprising results in your HTML output.
Each description file has a <title> and a <general> tag. Inside a <demo> tag there must be a <script> tag and a <steps> tag filled with <step> tags. The <demo> tag can have an additional attribute, stepmode, which can be set to random or static. The latter should only be used for demos, where the sequence of demo steps is strictly required (e.g., later steps rely on data loaded by earlier steps etc.). If no stepmode is specified, random is assumed.
The <script> tag has a filename of an Amira-script as an attribute. This script is executed whenever the corresponding demo is started. Use it to define Tcl procedures for subsequent steps and to load necessary data sets and modules.
The <step> tag takes an optional name and a mandatory playcommand attribute. An option is to register a jumpcommand attribute. Please write a short but meaningful description for each step. That way untrained users have a chance of presenting your demo as well.
The content of playcommand is executed as a Tcl command to display the corresponding step. The content of jumpcommand is executed as a Tcl command to jump to the beginning of a step and display the first image. This is useful if one wants to explain something before the step is executed.
In demo description files only, the <description> tag may contain an optional <prepacked> tag that specifies the files to download. .tar files are untarred automatically by downloadData.tcl, and .tar.gz files are gunzipped before. You can register any number of <file> tags inside <prepacked>. The data to be tarred in order to use it as a prepacked file should be stored relative to $AMIRA_DEMOS in a directory called $AMIRA_DEMOS/data.
Also only in demo description files a <attributes> tag could be used to allow a demo categorization by the search algorithm of the Demo GUI.
A <staff> tag filled with <person> tags of which one may have the parameter status="responsible" can be registered below <description> also.
If you compiled your demo using *DemoMaker*, you are in good shape. Typing <DemoMakerName> writeDescriptionFile in the Amira console window pops up a file browser. An appropriate description.xml file is written with jumpcommand and playcommand supplied correctly. All you have to do is fill in the descriptions for the demo as a whole and for each step. Also, replace the step names with more descriptive ones.

Medivis Exhibit 1023

Also a title must be given before the `description.xml` file will work.
A description file that does not follow the conventions mentioned above will be skipped when parsing of the demo file tree takes place.

Creating thumbnail pictures

Besides the explanation of the demo in the description file, a thumbnail image can be helpful to show what kind of view one can expect from a particular presentation. Thumbnails are shown in the *Demo GUI* and in the Amira help window when it is used for demo steering. Furthermore demos can be started and steered with a web browser. See *below*.

If you have prepared your demos and set up selection as described below, start Amira and type: `source $AMIRA_DEMOS/<outputfolder>/<demodirectory>/makeSnapshots.hx` in the Amira console. Here `<outputfolder>` is 'selection' by default or, when using the command line tools, 'output' resp. the name specified in the -o option of `prepareDemos.tcl`. After that there are as many step<N>.png files as there are steps defined in the description file in a directory `$AMIRA_DEMOS/src/<demodirectory>/gfx/`.

If for some reason you need manually created thumbnails for a particular demo, proceed as follows: Start that demo, and invoke source `$AMIRA_DEMOS/setup/makeIcon.hx`. After that a Tcl procedure `makeIcon` has been defined. Call it in the Amira console with `makeIcon "<thumbnail filename>" <width> <height>`. Do not give a filename extension, `.png` will be added automatically by `makeIcon`. If both values for width and height are given, the viewer will be resized accordingly. If one of these values set to 0, the viewer image will be scaled using Lanczos interpolation.

Note: For better image quality, set your viewer size to a multiple of the thumbnail size (in the default case 150x150 for thumbnails => 900x900 viewer size). Also consider the contents of the viewer –if there are many small features, they won't be visible in the thumbnail. As is often true, less is more.

Final remark: Consider the description together with the thumbnail images as a storyboard. It should describe the story you want to tell the audience while presenting the demonstration. Indeed you find a file `storyboard.html` in your output folder after you have setup a selection. It contains all descriptions and thumbnails of the whole selection together in one file. Useful for browsing and printing with your favourite webbrowser.

Medivis Exhibit 1023

6.1.2 Data Storage

All data being used in the demos should be stored in the `$AMIRA_DEMOS/data/` directory. You should create a directory similar in structure to `$AMIRA_DEMOS/src/` for storing your data.
The advantages of storing the data this way are:

- the data is separated from the script files.
- a data file can easily be identified with the demo to which it belongs.
- the data can be copied for *pack-and-go* automatically (see *below*).
- the data can be used by several demos without needing to be duplicated.

Using a File Server to Store the Demo Data

If you want to use the same demo data sets on several computers (perhaps at different locations), it may be helpful to store the data files in a tar archive on a remote http server.
It is beyond the scope of this documentation to explain how to set up such a server.
If you decided to use a data server, define a naming scheme for your data sets and save the data sets on the server according to that scheme. Now use the URL for the data in your `description.xml` files. Be sure to create the tar files relative to `$AMIRA_DEMOS`.
Example:
The following lines of `description.xml` contain the data to be downloaded:

```
<prepacked href="https://servername/prepackeddemodata">
  <file src="bubble_2006.06.22-1_data.tar.gz"/>
</prepacked>
```

and `bubble_2006.06.22-1_data.tar.gz` contains:

```
tar tvf bubble_2006.06.22-1_data.tar
... 2006-06-22 12:10:10 data/cfd/bubblechamber/
... 2006-06-22 12:29:20 data/cfd/bubblechamber/physics.am
... 2006-06-22 12:29:01 data/cfd/bubblechamber/Blasensaeule.surf
... 2006-06-22 12:10:10 data/cfd/bubblechamber/streamsurface.surf
... 2006-06-22 12:29:20 data/cfd/bubblechamber/Vektorfeld_GW_01.am
```

In order to fetch all data sets, you must download them as follows prior to starting the demos:

Medivis Exhibit 1023

```
cd $AMIRA_DEMOS/hxdemos/src
tclsh $AMIRA_DEMOS/hxroot/share/hxdemos-scripts/downloadData.tcl
```

`downloadData.tcl` has no parameters or options!
Note: `downloadData.tcl` downloads ALL data files mentioned in the `prepacked` tag of the description.xml files found in the recursively traversed directory tree! All downloaded files are stored in the directory `$AMIRA_DEMOS/cacheddata`. Before `downloadData.tcl` downloads the files from the server, it checks to see if the files scheduled for downloading are already in the *cacheddata* directory. This ensures that a possibly time-consuming download is done only once. Then all files with the extensions `.tar`, `.tar.gz` or `.tgz` are unpacked to `$AMIRA_DEMOS/data` while all other files are copied to that folder. Depending on the amount of data you stored on your data server, this may take a quite a while and use a lot of disk space.
See *below* for downloading only those data files which are needed for a certain demo selection.

6.2 Selecting Demos

Demos can be selected in two ways:

1. The most preferable way is to use the *Demo GUI*, or
2. to use the command-line tool `prepareDemos.tcl`.

Demo GUI

The Demo GUI provides a very convenient way for selecting demos from a pool of available demos. It is also possible to change a demo selection afterwards. A demo selection can be saved in an Amira script file for running it later. For more information, see the *Demo GUI documentation*.

Command-line Tools

If for any reason the *Demo GUI* couldn't be used to select demos, you can use a command-line tool `prepareDemos.tcl` to do the selection.
The command-line tools are stored under `$AMIRA_ROOT/share/hxdemos-scripts`.

Medivis Exhibit 1023

You can specify a selection of folders and steps as parameters to create adapted output in a subdirectory `$AMIRA_DEMOS/output`.
The subdirectory then contains a file named `demosequence.hx` which is an Amira script for executing the selected demos. `demosequence.demo` contains the demos and their steps and is used by the *DemoSequence* module for steering. Load the script into Amira to start the demos.
Additionally an `index.html` file which can be used to control the demos from a web browser. To do so, start Amira and execute the command `httpd start` in the console and afterwards direct your favorite web browser to the URL `http://localhost:7000/<$AMIRA_DEMOS>/<outputfolder>/index.html` where `<outputfolder>` is `output` or the directory specified with the `-o` option.
It is also possible to load the `index.html` file into Amira and start the demos from Amira's help window.
The full syntax of `prepareDemos.tcl` is

```
prepareDemos.tcl [--help | {[--include] {folder [--steps
   <steplist>]}* | --exclude <folderlist> }* |
   [--datainclude <folderlist>*] |
   [--output <outputfolder>] |
   [--force | --download | --[all]packandgo <folder>]

   --include,   -i: given folders are included recursively to
                    the output
   --steps,     -s: string of type "5 8 2 1" defining selection
                    and order of steps, steps starting from 1
   --exclude,   -e: following folders are omitted in the
                    recursion
   --output,    -o: specifies the outputfolder relative to
                    the demos directory
   --force,     -f: overwrites content in outputfolder or
                    packandgo-folder;
                    if not given and folder exists,
                    prepareDemos issues a warning and exits

   --download,  -d: downloads data needed by the selected demos
   --packandgo, -p: copies output and needed data to the
                    specified folder
   --allpackandgo : copies output, all from src and needed data
                    to the specified folder
   --datainclude  : given folders are included recursively to
                    the data folder, if --packandgo has been
                    specified, too
   --help,      -h: prints this helpscreen and exits
```

The `-o` option allows you to specify an output folder different from the default

Medivis Exhibit 1023

folder named *output*. If the `packandgo` option is given at the same time this folder will be created under the pack-and-go folder.
The `--packandgo` option copies all files recursively from the `$AMIRA_DEMOS/src` folder to a directory `src` under the folder specified for this option. The needed data sets are then copied from the directory `$AMIRA_DEMOS/cacheddata` into that folder, too. After that, you must unpack these tar files relative to the pack-and-go folder.
In order to have the data ready in *cacheddata*, specify the `--download` option.
As an alternative way to provide the needed data it is possible to supply a `--datainclude` option in conjunction with the `--packandgo` option. This option must have a selection of pathnames relative to `$AMIRA_DEMOS/data` which are copied to a data folder under the pack-and-go folder.
It is now possible to set the `$AMIRA_DEMOS` environment variable to the pack and go folder.
Note: Be sure not to have any "/" (slash) at the end of a path since this leads to wrong results. Please, don't use the options `--datainclude` and `--include` together in conjunction with `--packandgo`, cause it copies way too much into the packandgo folder.
Example:

```
cd $AMIRA_DEMOS/src
tclsh $AMIRA_ROOT/share/hxdemos-scripts/prepareDemos.tcl
cfd/bubblechamber medicine/hyperthermia -s "1 2 5"
--download --output mydemo
```

This selects the entire bubble chamber demo and three steps from the hyperthermia demo and stores the necessary files in `$AMIRA_DEMOS/mydemo`. The requested data sets for the two demos are stored under `$AMIRA_DEMOS/data` if there is a `<prepacked>` tag in the appropriate description files.
On Unix/Linux systems start Amira with `$AMIRA_ROOT/bin/start $AMIRA_DEMOS/mydemo/demosequence.hx` and on Windows systems load `$AMIRA_DEMOS/mydemo/demosequence.hx` just after Amira has been started.
Pack-and-Go Example:

```
tclsh $AMIRA_ROOT/share/hxdemos-scripts/prepareDemos.tcl
--packandgo /media/disk/hxdemos brusselator
microvisu/gigamosaic microvisu/synchrotron
microvisu/smallnetwork
```

This creates a folder named `hxDemos` under `/media/disk/`, selects four demos and puts the output into that folder. You also find under that folder the needed

Medivis Exhibit 1023

data as tar files, named as described in the appropriate description .xml file.

6.3 Prerequisites

The following system tools and programs are used by Demo Framework:

- A recent version of `tclsh` and `xsltproc` is required for the scripts `prepareDemos.tcl` and `downloadData.tcl`.

- `wget` and either `gtar` or `gzip` and `tar` is required for `downloadData.tcl`.

On Unix/Linux systems all tools should already be installed.
On Windows systems all tools should already be installed in *cygwin* as well. If `xsltproc` is missing, it is part of the package `libxslt`. Make sure that the `PATH` environment variable includes `cygwin/bin` to be able to use it via `exec` in Tcl scripts.

Medivis Exhibit 1023

Part II

Molecular Option User's Guide

7 Molecular Option Introduction

Molecular Option is an Amira extension providing support for the visualization as well as the analysis of molecules and dynamic molecular data.
The Molecular Option documentation is organized into the following sections:

- *Getting started with molecular visualization*
- *Structure and interdependence of the molecular data structures*
- *Essentials for displaying a molecule*
- *Alignment facilities in Molecular Option*
- *Visualizing dynamic data*
- *Atom expressions*

7.1 First Steps with Molecular Visualization in Amira

This chapter gives you an overview of the visualization of molecular data sets. Amira is not just able to display a 3D image of the molecule but also provides tools to investigate its distinct parts and properties. After reading the ”getting started” introduction you may continue with any of the following tutorials.

- *Getting Started* - first steps
- *Selection, Labeling, and Masking* - exploring a molecule
- *Alignment of Molecules* - visualizing dynamical data
- *Molecular surfaces* - wrapping a molecule
- *Sequential and Structural Alignment* - comparing molecules
- *Molecule Editor* - interactive manipulation of the molecule
- *Molecular Interface* - computing intersection surfaces
- *Measurement* - measuring distances and angles within a molecule

Note: If you want to visualize your own data, please refer to the section about data import in the Amira User's Guide. That section contains some general hints on how to import data sets into Amira.

7.1.1 Getting Started with Molecular Visualization

In this section you will learn how to

1. load a molecular demo data set into Amira,
2. view the molecule with the *MoleculeView* display module,
3. try out various color schemes,
4. select atoms in the viewer using the draw tool,
5. overwrite color scheme colors for selected atoms.

7.1.1.1 Loading Data into the System

In the following introduction we will consider a simple example to explain the basic functions of the *MoleculeView* module. Our example will be a small PDB (Protein Data Bank) structure consisting of 932 atoms in 213 residues.

- Load the file `2RNT.pdb` into the Pool from the directory `data/molecules/pdb`.

A green icon labeled `2RNT.pdb` will appear in the Pool. The green icon represents an object of type *Molecule*. Click on the green data icon with the left mouse button. In the Properties Area below the Pool information about the molecule, such as the number of atoms etc., will be shown. In addition, other ports named *Transformation*, *Selection Browser*, and *Transform*, can be seen; they will be explained in later tutorials.

7.1.1.2 Displaying the Molecule with the MoleculeView Module

The *MoleculeView* module is the basic display module for visualizing molecules. It allows you to display atoms as plates or balls and bonds as lines or cylinders.

- Click again on the green data icon in the Pool, this time using the right mouse button.

A menu, containing several entries and submenus, will appear.

Medivis Exhibit 1023

- Select the entry *MoleculeView*.

A new yellow icon labeled *MoleculeView* appears in the Pool. Yellow icons, in general, represent display modules, i.e., modules that visualize objects in the viewer. The blue line between the icons indicates a connection between the objects. In this case the MoleculeView module reads data from the object represented by the green icon and visualizes it.

The molecule is now displayed in the viewer as a wireframe. Using the left mouse button you can rotate the object; using the left and middle mouse buttons simultaneously you can zoom in and out. For these mouse operations to work, the viewer must be in *viewing mode*, in which case the mouse cursor is displayed as a hand.

Whenever the MoleculeView module is active, the little square on the yellow `MoleculeView` icon is orange. You can deactivate the module by clicking on the square with the left mouse button.

We will explore some basic ports of the MoleculeView module.

Mode port: Choose another mode to see both atoms and bonds of the molecule, or just atoms. If atoms are shown, use the *Atom Radius* port to adjust the size of the atoms as desired.

Quality port: If you choose the option *correct*, you can display a correct, i.e. three-dimensional, image of the balls and sticks. Consider the trade-off between correct representation and slower rendering performance. Use the *fast* mode if you want to display a large molecule. If your graphics hardware is fast enough, you can use the *correct* mode even for large molecules.

Complexity port: In order to allow interactive rendering, the default complexity of the scene is rather low. In most cases this will be sufficient. However, if you want to make screen shots or if you have small molecules, you might want to increase the complexity. The *Complexity* port is displayed only if *correct* quality is selected.

7.1.1.3 Changing the Color Scheme

The MoleculeView module allows the user to color the molecule according to levels, for example, atom level, residue level, secondary structure level, chain level, or user-defined levels. Each level has a number of attributes, the simplest of which is the *index* attribute. The default color scheme is to color the molecule according to the atom level's attribute *atomic_number*, i.e., the atom's type.

- Click the *Legend* button of the *Color* port to display a small window decoding the colors you see in the viewer window.

Medivis Exhibit 1023

- Choose *residues* from the first pop-up menu and *type* from the second menu to color the molecule according to the residue type, here the amino acid type.

The default colormap contains a constant color. Thus, currently all residues of the molecule have the same color.

- To change the colormap, right-click on the color bar of the *Discrete CM* port which has appeared below the *Color* port and select any other colormap.
- Try out different colormaps to find a map that suits your purposes.

Some people prefer the *CPK* color scheme for atoms and the amino acid colors as they are used in the *RasMol* program.

- You can set your preferences in the Amira*Edit* menu by selecting the entry *Preferences*.
- Press the *Molecules* tab and set your preferences. Pressing the *OK* button saves your preferences permanently.

Currently only amino acid colors are predefined. Thus, if the molecule contains residues other than the standard amino acids, those residues will be colored with the default color (black).

7.1.1.4 Using the Draw Tool

The draw tool appears in Amira in several modules. It enables you to select objects or parts of an object by drawing a line in the viewer.

- Press the *Draw* button of the *Highlighting* port and draw a line in the viewer window around the group of atoms you want to select.

The atoms that were enclosed by the line will be highlighted. If the viewer is the active window, pressing the `d` key is equivalent to pressing the *Draw* button of the *Highlighting* port.

- To unhighlight all atoms, press the *Clear* button of the *Highlighting* port.
- Also try using the keys `Ctrl` and `Shift` while drawing a line.

7.1.1.5 Setting Colors for Selected Parts

The *Define Colors* port allows you to overwrite colors of the color scheme.

Medivis Exhibit 1023

- Select some atoms using the draw tool.
- Press the *Set* button of the *Define Color* port.
- Change the current color of the *Color Editor* and press *OK*.

The previously highlighted atoms should now have the color selected in the color editor. This setting will be preserved even if the color scheme is changed. Unfortunately, the new setting will currently not appear in the color legend.

7.1.2 Selection, Labeling, and Masking

You already know how to load files and how to display molecules with the *MoleculeView* module. This tutorial will focus on exploring the structure of a loaded molecule. The tutorial consists of three subsections, in which you will:

1. Explore the possibilities for selecting within a molecule using the *MoleculeView* module.
2. Learn how to use the *MoleculeLabel* module.
3. Get to know the *Molecule Selection Browser*.

For this tutorial we will use the molecule `1IGM.pdb`.

- Load the file `1IGM.pdb` into the Pool from the directory `data/molecules/pdb`.
- Attach a *MoleculeView* module to the green object that has appeared,
- and change the *Mode* port to *balls and sticks*.

7.1.2.1 Interactive Selection with the MoleculeView Module

In the first tutorial you learned how to use the draw tool to select atoms within a molecule. The simplest way to select atoms, however, is to click on the atom of interest. In order to do so, the viewer must be in *interaction* mode. If the mouse cursor in the viewing window is depicted as a hand, you are in viewing mode. To change to interaction mode, you can either

- click on the arrow button in the upper right corner of the viewing window or press the `Esc` button while the viewing window is active.

The mouse cursor will change to an arrow.

- Now click one of the atoms.

Medivis Exhibit 1023

The atom you selected should now be highlighted by a red frame around it. If you select another atom, the first atom will be unhighlighted. In order to select more than one atom

- press the `Ctrl` key and keep it pressed while clicking additional atoms.

`Ctrl`-clicking a highlighted atom a second time unhighlights it.

- Now change the *Mode* port of the *MoleculeView* back to *sticks*.
- Select *residues* from the first menu of the *MoleculeView's Color* port.
- Choose a suitable colormap from the list of pre-loaded colormaps as described in the first tutorial.

The residues should now be colored according to their type.

- Now select an atom.

As result the whole residue should now be highlighted. The reason for this is that picking is bound to the coloring, by default. For example, if the color scheme is *atoms*, a click on an atom will only influence the selection for this atom; if the scheme is *residues*, the atom's residue will be selected; if it is *chains*, the atom's chain will be selected, and so on. However, since this is very restrictive you can easily change the selection mode to the most common levels, i.e., atoms, residues, secondary structures, and chains. In order to choose a certain level for the selection you need to press certain keys in advance. `Ctrl-Alt-Shift-a` chooses the *atoms* level, a click on an atom will only influence the selection for this atom. `Ctrl-Alt-Shift-r`, `Ctrl-Alt-Shift-c` and `Ctrl-Alt-Shift-s` choose *residues, chains* and *secondary_structure* level respectively. To switch back to the default behavior, where coloring determines selection, press `Ctrl-Alt-Shift-d`.
If you select some group and afterwards select a second one from the same level holding down the `Shift`-key all groups between the two will also be selected. Holding the `Ctrl`-key pressed while selecting groups allows you to select multiple groups and also to toggle a group when selecting it a second time.
If you do any selection by clicking in the viewer there will be some output in the console window informing you about what you have selected, the amount of output can be customized via the Preferences dialog:

- You can set your preferences in the Amira*Edit* menu by selecting the entry *Preferences*.

Medivis Exhibit 1023

- Press the *Molecules* tab and take a look on the options in chapter *Selection Info*:
- **Molecule name** determines that the name of the molecule, to which a selected group belongs, will be printed.
- **Group name** determines that the name of the selected group will be printed.
- **Group attributes** determines that not only the selected group's name but also all attribute values of the group will be printed in the console window.
- **Explicit attributes** restricts the output of attribute values to those attributes that are explicitly named in the text field.
- Pressing the *OK* button saves your preferences permanently.

7.1.2.2 Using the MoleculeLabel Module

So far you have seen that you can select atoms and groups of atoms by clicking on the molecule. The result of the picking is always printed in the console window. This might suffice in some cases. In other cases, however, it is necessary to label the molecule in the viewer so that you can easily track certain groups you are interested in. If you want to use this tool, here is how you do it.

- Click again on the icon *1IGM.pdb* with the right mouse button.
- Select *MoleculeLabel* from the *Display* submenu.

A second yellow icon labeled *MoleculeLabel* should have appeared in the Pool. By default, all clicks will now be handled by the *MoleculeLabel* module. This means that instead of highlighting groups when clicking on them they will now get labeled.

- Type `Ctrl-Alt-Shift-r` while the viewing window is active.
- Click on the molecule in the 3D viewer.

This action should cause the residue of the selected atom to get labeled. Pressing `Ctrl-Alt-Shift-a` and clicking the same atom will result in the selected atom being labeled. The `Ctrl-` and `Shift-` keys have the same effect as when selecting groups.

- Label a few residues and atoms.
- Click on the *MoleculeLabel* icon to view its ports in the Properties Area.
- Change the color of the atom labels by pressing the color button of the *Color* port and selecting a different color in the color editor.

- Now select *residues* in the *Levels* port. The *Attributes*, *Level Option*, *Buttons*, *Font Size*, and *Color* ports now affect the residue labels.
- Increase the font size (which only increases the font size of the current level, i.e., residues).
- Select the *name* entry in the second menu of the *Attributes* port.

The selected residues will now be labeled by their types and names. In order to understand the *Options* port of the *MoleculeLabel* module

- deselect the last option, *replace attributes*,
- and set the entry of the second *Attributes* menu back to *disabled*.
- Now select a new residue, holding the `Ctrl-` key down.

The new residue will only be labeled by its type. In contrast, the old residues are still labeled with type and name.
If the first option of the last port, *handle clicks*, is deselected, mouse clicks will be handled just as if the module *MoleculeLabel* did not exist.

- Deselect the *handle clicks* option.
- Select any residue.

The residue is selected, not labeled.

7.1.2.3 Molecule Selection Browser

In this section you will learn the basics about the *molecule selection browser*, which is a very important feature for the investigation of a molecule. The selection browser is a flexible tool for selecting those parts of the molecule that you are interested in. Moreover, it allows you to easily combine different viewing modules into one viewer.

- Select the green icon *1IGM.pdb* by clicking on it.
- Press the *Show* button of the *Selection Browser* port to open the browser.

The scroll view of the browser currently contains three columns, one with the heading *level/name*, the second with the heading *type*, and the third with the heading *MV* which stands for *MoleculeView*. In the first column, all residues of the molecule are displayed. The second column shows the residue type.

- Connect a *BondAngleView* to `1IGM.pdb` by right-clicking the icon and choosing *BondAngleView* from the *Display* submenu.

Medivis Exhibit 1023

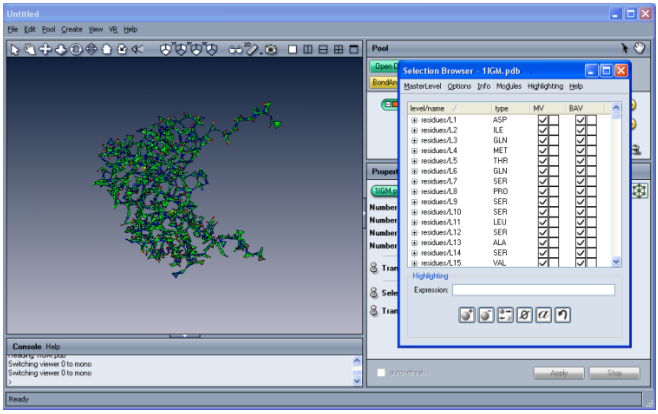


Figure 7.1: On the left, MoleculeView and BondAngleView displaying the molecule simultaneously. On the right, the Selection browser after connecting two viewing modules to the molecule.

You will observe a new column in the selection browser, representing the *BondAngleView (BAV)* (see Figure 7.1).

Changing the Appearance of MoleculeView and BondAngleView

In order to get an image similar to Figure 7.2, we first need to set the ports in the *MoleculeView* and the *BondAngleView* modules correctly. We begin with the *MoleculeView* module.

- Set the *Mode* port to *balls*.
- Set the *Quality* port to *correct*.
- Set the *Atom Radius* port to 1.0.

For the *BondAngleView* continue as follows:

- Select the *residues* entry from the first menu of the *Color* port.
- From the second menu of *Color* port, select the *index* entry.
- Right-click on the color bar of the *Discrete CM* port and select the colormap *physics.icol*, if this colormap is pre-loaded. Otherwise load it from the directory `data/colormaps`.

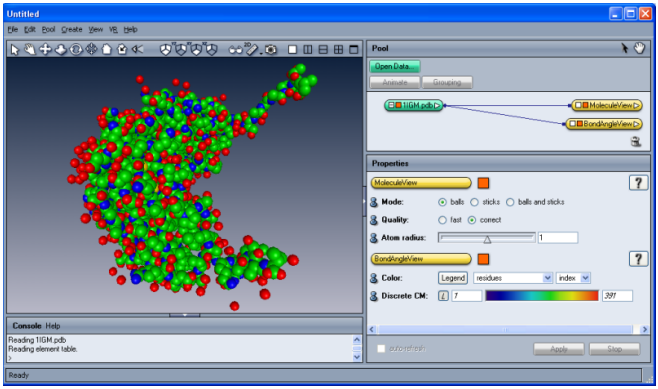


Figure 7.2: Setting the ports in the viewing modules.

We now only see the *MoleculeView*, since the triangles of the *BondAngleView* are hidden by the van der Waals spheres. In order to combine the two viewing modules into one image, continue with the next section.

Highlighting and Masking with the Selection Browser

We now want to use the selection browser to display parts of the molecule with the *MoleculeView* module and the rest with the *BondAngleView* module. In this section we will only use the selection browser, so all menu names etc. refer to this window.

- Select *chains* as new master level from the *MasterLevel* menu.

You will now see three entries in the first column: *chains/L*, *chains/H*, and *chains/W*. This means that the level *chains* contains three groups, named *L*, *H*, and *W*. In order to view the group *chains/L* with the *MoleculeView* and the other two groups with the *BondAngleView*,

- remove the check marks for *chains/H* and *chains/W* from the column with heading *MV* by clicking on them.

To deselect the group *chains/L* for the *BondAngleView*,

- click on the respective box in the column *BAV*.

Medivis Exhibit 1023

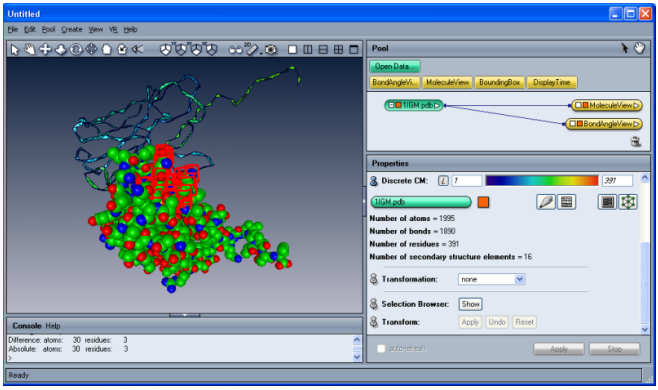


Figure 7.3: Displaying parts of the molecule with the *MoleculeView* module and the rest with the *BondAngleView* module.

Now you should see an image that is pretty close to that in Figure 7.3. However, the *BondAngleView* displays more triangles than in the figure. The *BondAngleView* in Figure 7.3 only displays *backbone* atoms. In order to achieve this

- right-click on the heading labeled *BAV*.

A pop-up menu as in Figure 7.3 should appear.

- Select *Backbone* from the *Restrict to* submenu.

The side-chain atoms should not be visible anymore. Next, you should explore the group *chains/L* a bit further.

- Click on the little icon left to it.

After this action the residues contained in chain *L* will have appeared.

- Click on the little icon left to the residue *L1*.
- Type `atoms/L4-L33` in the text field labeled *Expression*, below the scroll window, and press the *Replace* button.

Atoms *L4* to *L8* and residues *L2*, *L3*, and *L4* are highlighted in red. The residue *L1* and the chain *L* are highlighted in green. The color *red* means that the whole

Medivis Exhibit 1023

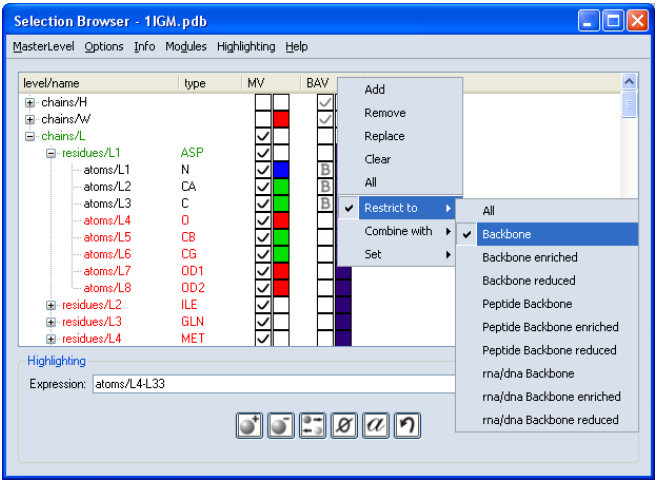


Figure 7.4: Molecular browser dialog.

group is selected, i.e., all its elements. *Green* denotes that the group is partially selected. See Figure 7.4.

To get familiar with the browser, play around with it. If you need further information, just press the `F1` button in the browser window or see the description of the *Selection Browser*.

7.1.3 Alignment of Molecules

In this section you will learn how to

1. align molecules by considering selected atoms,
2. compute a *mean molecule*,
3. compute a *configurational density*,
4. compare metastable molecular conformations.

Medivis Exhibit 1023

7.1.3.1 Comparing two Molecules

In this section we will compare two different three-dimensional structures of the same molecule.

- Load the data file `alkane.zmf` from the directory `data/molecules/alkane`.
- Select *MolTrajectory* from the pop-up menu of the *alkane.zmf* icon.
- Select *Molecule* from the pop-up menu of the `MolTrajectory` icon.
- Attach a *MoleculeView* module to the molecule.
- Repeat the last two steps, creating two new objects, *Molecule2* and *MoleculeView2*

The *alkane.zmf* icon represents a bundle of molecular trajectories, in this case butane, pentane, and hexane. By attaching a *MolTrajectory* to it, we extract a single trajectory. By default, the first trajectory, which is butane, is selected. We can now extract single time steps from the trajectory by attaching a *Molecule* object to the trajectory. We have done this twice since we want to compare two time steps with each other. Currently, both molecules extract the same time step. This is the reason for only seeing a single molecule. In order to get more information about the data structures, go to the section on *molecular data structures*.

- Select the *MoleculeView* icon and change the *Mode* port to *balls and sticks* and the *Quality* port to correct.
- Repeat this action for the *MoleculeView2*.
- Select the *Molecule2* icon and change the value of the *Time* port to 2.

You should now see two butane molecules in the viewer, slightly displaced.

- Right-click on the white square at the far left side of the *Molecule2* icon and select *AlignMaster* from the pop-up menu.
- Connect the blue line to the *Molecule* icon.
- Select the *Molecule2* icon.
- Press the *all* button of *Molecule2's Select* port.

You have now aligned the molecules *Molecule* and *Molecule2* using a least squares fitting of all atoms. This only works for molecules with the same number of atoms. In the following we will align the two molecules by considering only three carbon atoms.

Medivis Exhibit 1023

- Deactivate the *MoleculeView* by clicking on the orange square of the *MoleculeView* icon.
- `Ctrl`-click on three consecutive carbon atoms.

The frame of a red cube should appear around each of them.

- Activate the *MoleculeView* by clicking on the gray square on the *MoleculeView* icon.
- Select the *Molecule2* icon and press the *in slave* button of the *Select* port.

The three selected atoms should now fit better onto the corresponding atoms of the object labeled *Molecule*.

7.1.3.2 Computing a Mean Molecule

In this section we will compute a mean molecule of a metastable molecular conformation of butane.

- Remove the objects *Molecule* and *Molecule2*.
- Select the *MolTrajectory* icon.
- Load the data file `but_cluster_3_1.idx` from the directory `data/molecules/alkane`.

but_cluster_3_1.idx is a subtrajectory of *MolTrajectory*, i.e., a subset of all configurations in the trajectory.

- Attach a molecule to the icon *but_cluster_3_1.idx*.
- Select the *MeanMolecule* entry from the *Compute* submenu of *but_cluster_3_1.idx*'s pop-up menu.
- Right-click with the mouse on the white square at the far left side of the *MeanMolecule* icon and select *AlignMaster*.
- Connect the *AlignMaster* port to the *Molecule* icon by attaching the blue line to it.
- Select the *MeanMolecule* icon and press the *all* button of the *Select* port of the *MeanMolecule* module.
- Press the *Apply* button to compute the mean molecule.
- Visualize the mean molecule *but_cluster_3_1.mean* by attaching a *MoleculeView* to it.

Medivis Exhibit 1023

- Connect the *AlignMaster* port of the module *MeanMolecule* to the *but_cluster_3_1.mean* icon.
- Press the *Apply* button of the *MeanMolecule* module two or three times.

You have now computed a mean molecule of the subtrajectory *but_cluster_3_1.idx*. The *AlignMaster* is used to align all time steps before accumulating the atom positions. Changing the *AlignMaster* to the previously computed mean molecule and repeating the computation of the mean molecule will improve it. This procedure should converge.

A better way to compute a mean molecule is to use the module *PrecomputeAlignment*, using the *Multiple Alignment* option of the *Mode* port. This module creates an object containing a transformation for each structure in the trajectory. This object can then get connected to the *PrecomputeAlignment* connection port of the *MeanMolecule* module.

7.1.3.3 Computing and Visualizing a Configuration Density

For the subset of configurations contained in the subtrajectory *but_cluster_3_1.idx*, we will now compute a *configuration density* and visualize it with the *Isosurface* display module.

- Select the entry *ConfigurationDensity* from the *Compute* submenu of *but_cluster_3_1.idx*'s pop-up menu.
- Connect the *AlignMaster* port of the *ConfigurationDensity* icon to the *but_cluster_3_1.mean* icon.
- Select the *ConfigurationDensity* icon.
- Press the *all* button of the *Select* port.
- Press the *Field* button of the *Compute* port to compute the density.
- Visualize the created scalar field *but_cluster_3_1.scalar* with an isosurface by first selecting the *Isosurface* entry from the *Display* submenu of the icon's pop-up menu,
- second, setting the *Isosurface*'s *Threshold* value to 0.1,
- and third, pressing the *Apply* button.
- Try different *Threshold* values.

Medivis Exhibit 1023

7.1.3.4 Comparing Metastable Molecular Conformations

The set of configurations in the subtrajectory *but_cluster_3_1.idx* belongs to a metastable conformation of butane. In this section we will compute the density of a second metastable conformation and compare the two with each other.

- Select the *MolTrajectory* icon.
- Load the data file `but_cluster_3_2.idx` from the directory `data/molecules/alkane`.
- Compute the mean molecule of the subtrajectory *but_cluster_3_2.idx* by repeating the steps described *above* for the subtrajectory *but_cluster_3_1.idx*.
- Visualize the second mean molecule with the *MoleculeView* module.
- Select three consecutive carbon atoms in the second mean molecule as was done in the first tutorial.
- Attach the *AlignMaster* of the object *but_cluster_3_2.mean* to the first mean molecule *but_cluster_3_1.mean*.
- Select the *but_cluster_3_2.mean* icon and press the *in slave* button of the mean molecule's *Select* port.

The two mean molecules should now be aligned to each other, i.e., the three selected carbon atoms should superimpose.

- Compute the density for the second subtrajectory using the *but_cluster_3_2.mean* molecule as *AlignMaster*.
- Visualize the computed density with an *Isosurface* module.
- Double-click on the colormap of the *Isosurface2* module and select a different color to better distinguish the two isosurfaces from each other.

You should now clearly see how the two conformations of butane differ. For larger molecules it might be interesting to color the isosurface according to the atom's colors. This can be done using the same *ConfigurationDensity* modules by selecting the *Color Field* option of the *Field* port. Attach the computed color field to the *ColorField* connection port of the *Isosurface* module.

Notice that you can also visualize the densities with the *Voltex* module of the *Display* submenu.

Medivis Exhibit 1023

7.1.4 Molecular Surfaces

In section 7.1.1 of this tutorial you have seen how to visualize molecules with the *MoleculeView* module. Section 7.1.2 , on selection, labeling and masking, showed you how to use Amira's facilities to select, mask, and label certain parts of the molecule. In this tutorial we will

1. compute a molecular surface with the *CompMolSurface* module,
2. compute the molecular surface for a restricted set of atoms,
3. compute partial surfaces,
4. explore the *MolSurfaceView* module,
5. get to know the picking facilities of the *MolSurfaceView*.

7.1.4.1 Compute Molecular Surfaces

In this section you will learn how to use the *CompMolSurface* module to compute molecular surfaces with different resolutions. You will also become familiar with some of the module's basic ports.

- Load the data file `2RNT.pdb` from the directory `data/molecules/pdb`.
- Attach the *CompMolSurface* module to the green icon by selecting the corresponding entry from the *Compute* submenu of `2RNT.pdb`'s pop-up menu.
- Select the `CompMolSurface` icon.
- Press the *Apply* button.
- Attach the *MolSurfaceView* module to the newly created green icon, `2RNT-surf`.

You should now see a gray solvent excluded surface which is still pretty coarse. If you want a surface with better resolution,

- increase the number of points per $Å^2$ in the CompMolSurface module.
- Press the *Apply* button again.
- Try different resolutions.
- Now select the *no duplicate points* option in the *Options* port.
- Press the *Apply* button.

Due to the last action, some sharp edges will have disappeared. If there are no duplicate points at sharp edges, the surface normals of the adjacent triangles will

Medivis Exhibit 1023

be interpolated, thus leading to a smoothing, which in this case might be undesired. However, this option is important if it is necessary to have a completely closed surface.

- Change the *Quality* port to *faster*.
- Press the *Apply* button.

You should observe that some atoms disappear. This is due to the fact that now only one surface component will be computed. If the molecular surface consists of only one component, the whole surface will be computed. Since the underlying algorithm does not need to touch every single atom, but approximately only every second atom, the algorithm is much faster. If performance is an issue, you might consider using this option.

7.1.4.2 Compute Partial Surfaces

In order to compute partial surfaces we need to select the atoms for which we want to compute their surface contribution.

- Open the selection browser. If you do not know how to do this, take a look at the *second tutorial*.
- Type `within(residues/HET105, 5)` in the browser's *Expression* command line.
- Press the selection browser's *Replace* button.

All atoms within a distance of 5 Å of the *HET105* residue will now be selected.

- Reset the *CompMolSurface's Quality* port back to *correct*.
- Select *partial surface* from the *Options* port of *CompMolSurface*.
- Press the *Apply* button.
- Select *adjacent patches* from the *Options* port of *CompMolSurface*.
- Press the *Apply* button.

The last action causes the partial surface to be expanded by the toroidal patches adjacent to the partial surface.

7.1.4.3 Molecular Surface of a Restricted Set of Atoms

The molecule *2RNT.pdb* contains some water molecules, which are in most cases not desired when computing the molecular surface. In order to exclude the water

Medivis Exhibit 1023 256

molecule from the surface computation

- open the selection browser again.
- In the selection browser scroll to the end of the list of residues.
- Click on one of the strings *HOH* in the *type* column.

All residues of type *HOH* should now be highlighted.

- In the browser, right-click on the heading *CMS* which stands for *CompMolSurface* and select the entry *Remove*.
- Deselect *partial surface* in the *CompMolSurface* module.
- Press the *Apply* button.

For the new surface only those residues that have a check mark in the selection browser were considered. You may combine this procedure with the partial surface computation described above.

7.1.4.4 Exploring the MolSurfaceView

The *MolSurfaceView* module is already attached to the molecular surface. A second connection exists to the molecule `2RNT.pdb`, from which data is read to enhance the visualization.

- Compute the whole molecular surface with a resolution of 2 points per $Å^2$.
- Click on the `MolSurfaceView` icon to see its user interface in the Properties Area.
- Select *molecule* for the *Color Mode*.
- Change the *Color* port's first menu entry to *residues*.
- Select an appropriate colormap for the *Discrete CM* port by right-clicking on the color bar.
- Switch to interaction mode by clicking on the arrow button in the upper right corner of the viewing window.
- Click on the surface in the viewing window.

All triangles belonging to the picked triangle's residue will be highlighted. If the triangle belongs to two or even three (maximum) residues, all of those residues will be highlighted.
Clicking on the surface with the middle mouse button displays information about the atom you clicked on in the upper left corner of the viewing window as long

Medivis Exhibit 1023

as you keep the mouse button pressed. If you `Ctrl`-click, the information will remain displayed even after releasing the mouse button until the next mouse click on the surface.

- Press the *Highlighting* port's *Clear* button to remove the selection.
- Change the *Pick Action* port to *clipping*.
- Pick any triangle of the surface.

All triangles further away from the picked triangle than the distance given by the *Selection Distance* port will be cut off. All triangles within this distance will remain, however, only if they are connected to the picked triangle without leaving the sphere around the picked point.

- Press the *All* button of the *Buffer* port to display the whole surface.
- Change the *Pick Action* port to *surface*.
- `Shift`-click on the surface in the viewing window.

All clicks on the surface will now be handled as you might be familiar with from the *SurfaceView* module.

7.1.5 Sequential and Structural Alignment

In this tutorial you will become familiar with the *AlignSequences* and *AlignMolecules* modules.
The *AlignSequences* tool facilitates the comparison of two sequences. It can be applied to both proteins and nucleic acids except for t-rna molecules containing modified bases. The *AlignSequences* module is *not* highly advanced, but it might suffice for some purposes.
Having done the sequential alignment, you can use the correspondence produced by the sequence alignment to do a structural alignment of the molecules.

- Load the files `1IGM.pdb` and `2JEL.pdb` from the directory `data/molecules/pdb/` and connect a *MoleculeView* to each of them.

You should know how to do this from the *first tutorial*. For both *MoleculeViews*

- select *residues* from the first menu of the *Color* port.

A colormap (*Discrete CM*) with constant color will appear. To distinguish the molecules, choose a different color for one of them. In order to do so

Medivis Exhibit 1023

- double-click on the colorbar (the *Color Dialog* should appear) and
- change the current color by dragging and dropping any of the custom colors.
- Press the *OK* button of the color editor.

7.1.5.1 Sequence Alignment

We will now use the *AlignSequences* module to align the sequences of the two molecules. In the next section we will use the associated amino acid pairs for a structural alignment.

- Right-click on the *1IGM.pdb* icon and select *AlignSequences* from the *Compute* submenu.
- Right-click the white square on the far left side of the *AlignSequences* icon.

A pop-up menu displaying all connection ports opens.

- Select *MoleculeB* from it and connect the port to the *2JEL.pdb* icon by clicking on it.

There should now be two blue lines connecting the *AlignSequences* icon with the *1IGM.pdb* and *2JEL.pdb* icons, respectively.

- Select the module *AlignSequences* by clicking on its icon in the Pool.
- Choose *semiglobal* from the second menu of the *Align Type* port.
- Press the *Apply* button.

If the alignment has been successful, a window displaying several slightly different alignments will appear.

- Press the *AcceptAll* button. Then press the *Close* button.

This action results in the alignments being written to the molecules as new levels. In order to check this,

- type `1IGM.pdb list` in the Amira console window.

In addition to levels such as *atoms, bonds, residues, etc.* you will also see levels named *semiGlobalSeqAlign1* to *semiGlobalSeqAlign10*. *2JEL.pdb* has the same levels with an equal number of groups.

Medivis Exhibit 1023

7.1.5.2 Align Molecules by using the Mean Distance Criteria

We can now use the levels *semiGlobalSeqAlign** for a structural alignment. In order to do so,

- right-click the icon *1IGM.pdb* and select *AlignMolecules* from the *Alignment* submenu.
- Right-click the white square at the far left side of the *AlignMolecules* icon, select the *MoleculeB* entry, and connect the module to *2JEL.pdb*.
- Select the *AlignMolecules* icon to make it appear in the Properties Area.
- Select any of the levels named *semiGlobalSeqAlign** in the *AlignLevel* port and press the *Apply* button.

If you want to follow the alignment process,

- click the *show alignment* option before pressing the *Apply* button.

Compare your result to the online demo.

7.1.6 Editing of molecules

An essential tool for manipulating the molecular data structure from Molecular Option is the *Molecule Editor*. It allows adding new bonds or changing the overall topology of the molecule as well as manipulating Cartesian or internal coordinates. For each of these tasks, there is an example in the following section to give you an easy "learn by doing" introduction to the available features. Each action performed with the editor affects all currently selected atoms. Thus, it is necessary for you to be familiar with the functions of the *SelectionBrowser* beforehand (see section 7.1.2 on selection, labeling, and masking).

- Load the file `1HVRm.pdb` from the directory `data/molecules/pdb/` and connect a *MoleculeView* module to it.
- Select the *balls and sticks* representation for the *MoleculeView*.

The data structure loaded is an enzyme of HIV in complex with the inhibitor XK263.

7.1.6.1 Invoking the Molecule Editor

To start the editor

Medivis Exhibit 1023

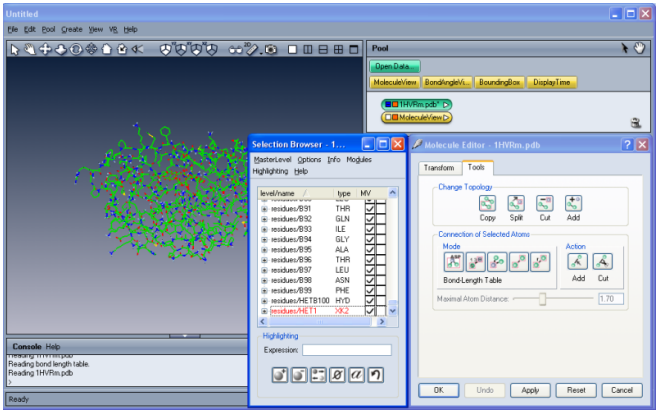


Figure 7.5: Adding bonds to the ligand

- select the `1HVRm.pdb` object in the Pool and press the Molecule Editor button (pencil icon) in its user interface.

The molecule editor interface is now visible. However, for the tasks we want to perform, we also need the selection browser, which is opened by

- pressing the *Show* button of the *Selection Browser* port of *1HVRm.pdb*.

7.1.6.2 Adding Bonds to a Part of a Molecule

The inhibitor XK263 is currently without bonds. To add bonds, carry out the following steps:

- Open the selection browser and select the residue with the type *XK2* (it is at the end of the list).
- Switch to the *Tools* tab in the molecule editor and press the *bond-length table* button in the *Mode* section. Then, press the *add* button in the *Action* section.

Adding bonds in the *bond length table* mode will create new bonds in the selected part of the molecule by comparing the distances between the atoms with a table of average bond lengths. The result should now look like Figure 7.5.

7.1.6.3 Splitting the Molecule

We now want to split the molecule into inhibitor and protein.

- If it is not currently selected, reselect the *XK2* residue in the selection browser.
- Press the *Split* button on the *Tools* tab of the molecule editor.

You will notice that the atoms of the inhibitor are no longer displayed by the *MoleculeView* and that a new object, *1HVRm2.pdb*, has appeared in the Pool. This object contains the previously selected atoms of the ligand.

7.1.6.4 Adding another molecule

The protein of the 1HVR entry is a protease which uses up one water molecule to split a polypeptide. We now want to add the water to the active site of the enzyme.

- Load the file `h2o.pdb` from the directory `data/molecules/pdb`.
- Go to the molecule editor and press the *Add* button in the *Change Topology* section.
- In the window that opens, all other molecules in the Pool will be shown. Select the *h2o.pdb* molecule and press *OK*.

The atoms of the water molecule have now been copied to the *1HVRm.pdb* object.

7.1.6.5 Moving Parts of the Molecule

To concentrate our view on the region of interest we will reduce the molecular view to amino acids *A25* and *B25* between which the water molecule should be placed.

- Type `r/name=?25 OR r/type=H2O` into the selection browser and press the *Add* button.
- Now move your mouse on the *MV* heading, press the right mouse button, and activate the *Replace* option.

With only the residues A25 and B25 and the water molecule displayed, all that is left to do is to drag the water molecule to its correct location.

- Select the water molecule in the selection browser (the residue at the end of the list).

Medivis Exhibit 1023

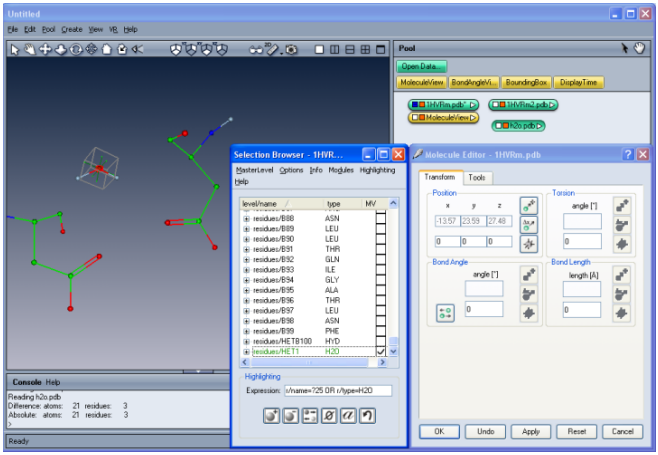


Figure 7.6: Moving the water molecule to the desired location

- Switch to the *Transform* tab of the molecule editor.
- Show the position of the transform dragger by pressing the button in the *Position* section of the *Transform* tab.
- Left-click on the dragger and hold the mouse button down. You can now translate the dragger in different planes depending on the side you clicked on. Move the water molecule between the two amino acids as shown in Figure 7.6.
- To reorient the water molecule, click on the green knobs of the dragger. They allow the dragger to rotate in different planes.

Repeat the steps described above until you are satisfied with the result.
To apply the changes that you made to the edited molecule, you must press the *Apply* button or end the editing by using the *OK* button.

7.1.7 Molecular Interfaces

This tool is particularly interesting for visualizing contact areas between parts of a single molecule or between different molecules, e.g., enzymes and their ligands. In this example we will consider the second case.

Medivis Exhibit 1023

- Please load the file `2RNT.pdb` from the directory `data/molecules/pdb`.

7.1.7.1 Defining groups

In this section we will create a new level, called *interface*, for the molecule and define two groups of the level, *substrate* and *receptor*, respectively.

- Select the `2RNT.pdb` icon in the Pool by clicking on it.
- Now click in the console window to activate it, and press the `TAB` key.

The name of the selected icon, `2RNT.pdb`, should appear in the console window. If it does not,

- please type `2RNT.pdb`.
- On the same line, type `define interface/substrate residues/HET105`

The following text should now be written in the console window
`2RNT.pdb define interface/substrate residues/HET105`

- Press the `ENTER` key.

You have now defined the group *substrate* in the level *interface*. The group consists of the residue HET105. We will now define the group *receptor*.

- Press the `TAB` key again, then type `defregexp interface/receptor NOT residues/HET105 AND NOT residues/type=HOH`

With `NOT residues/HET105` we exclude the substrate and with `NOT residues/type=HOH` we exclude all water molecules. Thus, the receptor is defined as consisting of all atoms not belonging to either the substrate or any water molecule.
If you now list all levels by typing

- `2RNT.pdb list`

you will see an *interface* level containing two groups.

- Type `2RNT.pdb list interface`

and all groups of the *interface* level will be printed in the console window.

Medivis Exhibit 1023 264

7.1.7.2 Computing the Interface

We will now compute the interface between the receptor and the substrate. The interface between two molecules is defined as the surface equidistant to both molecules. For the approximation we compute, this might not be exactly true for all points on the surface. The module to compute the interface is called *CompMolInterface*.
Now, to compute the interface,

- right-click the icon `2RNT.pdb` and select *CompMolInterface* from the *Compute* submenu.
- Select the icon labeled `CompMolInterface` in the Pool.
- Choose *interface* from the *Levels* menu of the *CompMolInterface* module.
- Press the *Apply* button.

Two objects result from this action, an interface object of type *MolSurface*, *2RNT-interface.surf*, and a distance field of type *UniformScalarField3*, *2RNT-distance.field*.

7.1.7.3 Visualizing the interface

Finally, we will view the computed interface in the viewing window with the *MolSurfaceView* module.

- Right-click on the `2RNT-interface.surf` icon and select *MolSurfaceView*.
- Right-click on the white square at the far left side of the *MolSurfaceView* icon and connect the *ColorField* port with the distance field *2RNT-distance.field*.

The user interface of the *MolSurfaceView* module should be visible in the Properties Area.

- Choose the *field* option from the *Color Mode* port, resulting in a new port, *Colormap*, appearing in the user interface.
- Right-click on the colormap bar and choose any colormap.
- Play around with the coloring by changing the colormap range. The full range of values can be seen when you select the *2RNT-distance.field*, i.e, click on it.

Medivis Exhibit 1023

- Also, change the *Cutoff distance* in the *CompMolInterface* module, e.g., to 1.0, and the *Voxel size*, e.g., to 0.5.
- See what happens when you press the *Apply* button of the *CompMolInterface* module.

Warning: If you choose a very small voxel size, the computation might take very long. Also, there might not be enough memory to store such a large distance field. Usually, 1.0 or 0.5 are good values.
After having done the steps described above, what you see should be similar to the Nuclease demo on the demo pages. The interface here, however, has been generated from two separate files.

7.1.8 Measurement

In this tutorial you will learn how to measure distances and angles between atoms in a molecule. For this tutorial it is necessary for you to have already done the tutorial 7.1.1.

- Load any molecule from the directory `data/molecules/*` and connect a *MoleculeView* to it.
- Select the *MoleculeView* by clicking on its icon in the Pool.
- Select *balls and sticks Mode* and *fast Quality*.
- Next, right-click on the `MoleculeView` icon, and select *Measurement* from the pop-up menu.

The user interface of the *Measurement* tool should now be visible in the Properties Area. An *Info* port tells you that you need to select 2, 3, or 4 atoms. In order to do so, switch to interaction mode by

- pressing the `ESC` key or by pressing the arrow button in the upper right corner of the viewing window.

You can now select single atoms in the viewer by clicking on them. If you click on one atom, the previously selected atom will be deselected. By using the `Ctrl`-key, you can select more than one atom.

- Select 2, 3, or 4 atoms and observe what is being displayed in the user interface of the *Measurement* module.

The `Ctrl`-key is also used to deselect atoms.

Medivis Exhibit 1023 266

7.2 Molecular Data Structures

Several molecular file formats including, for example, PDB, Tripos, and UniChem, can be read and written by Molecular Option, other file formats, such as CHARMM, can only be read. The information read from the data files will be stored in different types of objects: *Molecule*, *MolTrajectory*, and *MolTrajectory-Bundle*.

Molecule. Single molecular configurations are represented as data objects of type *Molecule*. This kind of object can either be created by loading a file containing a single molecular structure or by attaching it to an object of type *MolTrajectory*, in which case it will act as a projector, extracting one of the trajectory's 'time steps', i.e., a single structure.

Trajectory. The *MolTrajectory* data structure represents a series of molecular configurations of the same molecule. Again two cases are possible. Either the trajectory is loaded directly from a file. Or it can be attached to an object of type *MolTrajectoryBundle*, extracting one of the trajectories contained in that bundle.

Bundle of trajectories. A trajectory bundle reflects the case of a file containing more than one molecular trajectory. If such a file is loaded into Amira, an object of type *MolTrajectoryBundle* will be created. A single trajectory can be accessed by attaching a *MolTrajectory* object.

7.2.1 Internal Structure of Molecules

An object of data type *Molecule* contains information about the structure of a molecule and the atomic coordinates of one of its configurations. The mandatory part of structural information concerns the number and types of all atoms contained in the molecule. All the topological information is organized in levels which are cliques of groups. Each group contains other groups or atoms. The simplest example is the level *bonds*, which consists of groups of two atoms.

Some levels can build hierarchies. For example, a residue consists of a number of atoms, and a couple of residues may form a secondary structure. These levels and groups can be defined by file readers or interactively via *Tcl commands*.

7.3 Displaying Molecules

Molecules can be visualized with the *MoleculeView*, *BondAngleView*, *SecStructureView*, or *TubeView* modules. A sample output of the first two modules is

Medivis Exhibit 1023

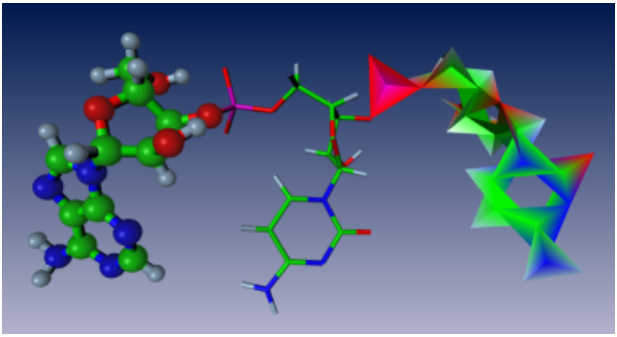


Figure 7.7: MoleculeView (left and middle) and BondAngleView (right) display the molecule simultaneously.

shown in Figure 7.7.
In addition, there are two modules for generating molecular surfaces. The *CompMolSurface* module enables you to generate the *solvent accessible, solvent excluded*, and *van der Waals surfaces* of a molecule. The *CompMolInterface* module can be used to generate intra- and intermolecular interfaces, such as between single atoms or residues, or between two molecules, respectively.

7.3.1 Coloring Molecules

The atom-oriented modules *MoleculeView*, *BondAngleView*, *ConfigurationDensity*, and *MolSurfaceView* allow a coloring on a per-atom basis, i.e., each atom is assigned a color. Balls in the *MoleculeView* will have the corresponding colors. Sticks will be split into halves colored according to their attached atoms. In the *BondAngleView*, atom colors are assigned to the vertices and color is interpolated over the triangles.
To determine the coloring you can choose a level, e.g., atoms, residues, secondary structures, or chains. Furthermore, you can choose one of the level's attributes. The coloring will then be done according to the group's attributes such that all atoms of the same group will have the same color.

Color

Medivis Exhibit 1023

A molecule may be colored according to various schemes. For example, each atom of a specific type may have the same color. Another possibility is to give all atoms belonging to a certain group the same color. The first menu of the *Color* port specifies the group level, e.g., atoms, residues, secondary structures, or chains. The second menu allows you to decide which attribute of the specified level should be considered to color its groups.
If you press the *Legend* button, a separate window will be opened, which displays a legend of the coloring, i.e. a table associating colors with attribute values according to the current coloring. Clicking on the text to the right of a color will select all atoms with this color.

Continuous CM

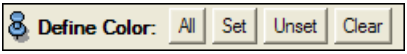
This colormap is used to map values of float attributes to colors. For optimal mapping, the colormap range must be set correctly. By default, the range is set to the minimum and maximum values that occur using the chosen color scheme, if the button in front of the first text field displays a capital ”L”. The ”L” means that the colormap uses a *local* range which allows you to modify the range without affecting the range of the same colormap used in other modules. By pressing the button you can toggle between *local* and *global* range.

The default colormap has a constant color over the whole range. By leaving it constant you give all atoms the same color, which may be useful if you want to compare different molecules. For more information, see the section on *colormaps*.

Discrete CM

This colormap is used to map values of discrete attributes, like integers and strings, to colors.

Define Color



This port can be used to overwrite the standard colors of the selected color scheme. With the *All* button you can set a new color for all atoms. The *Clear* button unsets the user-defined colors for all atoms. The *Set* and *Unset* buttons operate on the set of highlighted atoms. These buttons can be used to set and unset the colors of those atoms, respectively.

7.3.2 Selecting and Filtering atoms

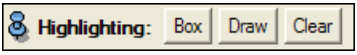
Various tasks require the selection of atoms in a molecule. The most prominent are the alignment of molecules and the selective display of molecule parts. Amira offers two selection methods. You can select atoms visually via any of the viewing modules. Alternatively, selection can be done via the molecule's *selection browser*, which can be opened by pressing the *Show* button of the molecule.

7.3.2.1 Selection of Atoms with a Viewing Module

The selection of atoms with a viewing module can be done in two ways. The first way is by clicking on certain displayed parts of the molecule. In general, this will highlight the selected parts. By default, the number of atoms that will be affected by a click depends on the selected group level of the *color port*. For example, if the atoms in the viewing module are colored according to the *residues* level, all atoms belonging to the same residue as the picked atom will get selected. However, since this is very restrictive, you can easily change the selection mode to the most common levels, i.e., atoms, residues, secondary structures, and chains. In order to choose a certain level for the selection you need to press certain keys in advance. `Ctrl-a` chooses the *atoms* level, a click on an atom will only influence the selection for this atom. `Ctrl-r`, `Ctrl-c` and `Ctrl-s` choose *residues, chains* and *secondary_structure* levels, respectively. To switch back to the default behavior, where coloring determines selection, press `Ctrl-d`.
If you pick another part, the previously highlighted atoms will be unhighlighted and the newly selected atoms are highlighted. `Ctrl`-clicking a part of the molecule leaves the existing selection state of all atoms unchanged except for the newly selected atoms. If the selected atoms had been selected before, they will get deselected, otherwise they will get selected. On the one hand, this allows you to add atoms to the selection, on the other hand it also allows you to deselect atoms.
If you select some group and afterwards select a second one from the same level holding down the `Shift`-key all groups between the two will also be selected. The `Shift`-key can also be used in conjunction with the `Ctrl`-key.
If you do any selection by clicking in the viewer there will be some output in the console window informing you about what you have selected, the amount of output can be customized via the *Preferences* dialog, which is opened by choosing the *Preferences* entry from the *Edit* menu. The preferences concerning Molecular Option are found on the *Molecules* tab.

Highlighting

Medivis Exhibit 1023



The second way to select parts of the molecule is by using the functionality of the *Highlighting* port.

- If you press the *Box* button, the molecule's bounding box will be displayed. All atoms contained in the box will be highlighted. You can change the size of the box in *interaction mode* by clicking on the highlighted handles (usually in a light green) of the box. Keep the mouse button pressed and drag in the desired direction. Notice that the box will not exceed the molecule's bounding box. You can also move the box within the bounding box by clicking on one of the box's sides. Pressing the *box* button a second time hides the box.
- After pressing the *Draw* button, you can draw a line in the viewer window which selects all atoms that are inside the line. Pressing the `Ctrl`-key while drawing inverts the selection. With the `Shift`-key you can remove atoms from the selection.
- The *Clear*-key deselects all atoms and hides the box if it is visible.

7.3.2.2 Filtering atoms

Filtering atoms gives us the ability to hide parts of the molecule for a certain module. All viewing modules and some computational modules, such as the *CompMolSurface* module, contain a *filter* that keeps track of all atoms of a molecule which are currently *in use*. The term *in use* means that the module will only see the *in use* atoms and regard all other atoms as non-existing. Thus, a viewing module will only display the *in use* atoms, and the computational module *CompMolSurface* will compute the molecular surface ignoring the *not in use* atoms. Each filter will register itself at the *molecule's selection browser*, so that you can easily determine the *in use* atoms of a module. For more information about how to do this, see the description of the *selection browser*.
In most modules, the filter's *Buffer* port will be visible, which adds to the convenience. The functionality of the *Buffer* port is described below.

Buffer

Pressing *Add* will append the selected atoms to the *in use* atoms. The *Remove*

Medivis Exhibit 1023

button will delete the selected atoms from the *in use* list. *Replace* will first clear the *in use* list and then add the selected atoms to it. Finally, the *All* and *Clear* buttons are shortcuts to delete or add all atoms from or to the *in use* list, respectively.
The buffer can also be accessed via tcl commands. Each visualization module using a buffer offers the following set of commands:

`showAll`
All atoms are shown.

`hideAll`
All atoms are hidden.

`setVisible`
All selected atoms will be hidden.

`addVisible`
All selected atoms will be hidden.

`removeVisible`
Only selected atoms are shown.

7.4 Aligning Molecules

To compare the structures of several molecules it is necessary to bring them into a suitable geometric arrangement relative to each other, because their absolute positions in the common coordinate system are generally meaningless. By arranging the molecules in various different ways, you can investigate various aspects.

7.4.1 Alignment of Trajectories

Molecular Option offers a reusable component for the alignment of molecules. It appears in several modules, e.g., *Molecule*, *ConfigurationDensity* and *MeanMolecule*. Here, we will give an overview of the available functionality.
Two connection ports are supplied:

AlignMaster `[optional]`

To compute an alignment relative to a reference molecule, this port must be connected to the reference.

PrecomputedAlignment `[optional]`

This port can be connected to an alignment precomputed by using the module *PrecomputeAlignment*.

Medivis Exhibit 1023

The control of the alignment is done via three ports, as described below:

Transformation

This port allows you to specify how the molecule should be aligned. The possible options are:

none: Atom coordinates are left as they are.

center of gravity: The molecule is positioned to the coordinate center (center of gravity) by applying a translation.

align to master: This mode requires a master molecule to be connected to the *AlignMaster* port which is used as a reference to which other molecules, the *slaves*, are aligned. We consider correspondences between atoms from the reference molecule, i.e. *master*, and atoms from the molecule to be aligned, i.e. *slave*. The alignment is done by minimizing the sum of squared distances between all corresponding atoms.

precomputed: This option is only enabled if the *PrecomputedAlignment* connection port is connected to an alignment object that has been previously computed. If *precomputed* is chosen, the transformation will be taken directly from this object.

external: This option indicates that the molecule's global transform has been explicitly set, e.g., via the Tcl command *setTransform* or by using the *Transform Editor*.

Select

This port becomes important if *align to master* is selected. It gives control over the atoms in the slave and the master that are used for the alignment. In the general case master and slave molecules are different, i.e. their numbers of atoms differ, and an alignment requires an explicit specification of pairs of atoms. This can be done by highlighting atoms in both molecules via the *MoleculeView* module and

Medivis Exhibit 1023

then pressing the *in both* button. Depending on whether the option menu is set to *Set* or *Add*, the highlighted atoms will either replace an existing list of selected atoms or otherwise will be appended to it. The matching between atoms in the two molecules is defined by their order of selection.
The buttons *all*, *in slave*, and *in master* offer shortcuts for the frequent case of slave and master having the same number of atoms including the assumption that their order defines a natural matching. The easiest way is to use all atoms by pressing *all*. Selection of a subset can be done by highlighting in either the slave or the master and then pressing *in slave*, or *in master*, which will use the selected atoms in the respective molecule for the other molecule, too.

Selection

If *align to master* is selected, this port displays the existing state of selection that is used to compute the alignment. Possible forms are:

empty: No atoms are selected. At least three are necessary for an alignment.

all atoms: Master and slave have the same number of atoms and all are used for the alignment.

0, 5, 6: A list of indices implies that master and slave have the same number of atoms and identical subsets have been selected.

0 →7, 5 →6, 6 →5: A list of index pairs (master index →slave index) indicates a matching resulting from individual selections in master and slave.

7.4.2 Mean Distance Alignment

Apart from the above mentioned alignment procedure, there exists a module that allows you to align two molecules by using the *mean distance* criterion instead of the *mean squared distance*. See *AlignMolecules* for more information.

7.4.3 Sequence alignment

Molecular Option allows you to *align the sequences* of two molecules. Three algorithms are available for use depending on the kind of data and your needs: local,

Medivis Exhibit 1023

semi-global, and global alignment. The algorithms work both for proteins and ribonucleic acids, whereas t-RNA molecules are currently not supported because they contain modified bases. This module can be very helpful when used in conjunction with the *AlignMolecules* module.

7.5 Visualizing Molecular Trajectories and Metastable Conformations

A *MolTrajectory* can be visualized via animation of single time steps by attaching a *Molecule* module to the *MolTrajectory* and then attaching a *MoleculeView* or *BondAngleView* to the *Molecule*. The animation is controlled via the *Molecule's Time* port.
For *MolTrajectories* that represent metastable conformations, the modules *MeanMolecule*, *PrecomputeAlignment*, and *ConfigurationDensity* can be used. The *MeanMolecule* module aligns all steps of a trajectory and computes a mean molecule by averaging every atomic coordinate over all time steps. Instead of computing the mean molecule to a reference, as with the *MeanMolecule* module, the *PrecomputeAlignment* module allows you to find the optimal transformation of each time step to minimize the overall sum of squared distances. With the *RankTimeStep* module you can search in a trajectory for a desired time step, using different criteria, such as the *rmsd* value to a given reference.
The *ConfigurationDensity* module gives an impression of the fuzziness of the conformation by computing a probability density for the positions of atoms and bonds within a molecular trajectory. This density can then be visualized with the *Isosurface* and *Voltex* modules.

7.6 Atom Expressions

7.6.1 Overview

Atom expressions are a query language to find and select atoms of certain properties in the molecule for further action. The most important application of atom expressions in Amira is the highlighting section of the *Selection Browser*.
In Amira, a molecule is separated into groups of different levels. Each group contains a set of attributes. (More details of this concept can be found in the description of the *Attribute Editor*). Atom expressions are a simple form of a relational

Medivis Exhibit 1023

query language which accesses these attributes.
The simplest form of an atom expression is an *atom specifier*. This is a literal defining a level, one of its attributes, and a condition for this attribute. For example, `atoms/atomic_number=8` defines all atoms whose atomic number attribute equals 8 (i.e., all oxygens).
Such atom specifiers can be combined with logical operators like `AND`, `OR` and `NOT`. For example, `atoms/atomic_number=8 AND NOT atoms/charge=0` will select all charged oxygen atoms.
There are additional operators like `WITHIN`, `BONDED` and `GROUP` which apply certain conditions to coordinates or bond structure. These are explained in section on *operators*.

7.6.2 Grammar

All possible syntaxes of atom expressions are shown in the following grammar. The different literals and operators are further explained in the following sections.

```
atomExpr →          ( atomExpr )
                    | NOT atomExpr
                    | atomExpr AND atomExpr
                    | atomExpr OR atomExpr
                    | WITHIN (atomExpr,float)
                    | BONDED (atomExpr[,int])
                    | GROUP (groupParts)
                    | CS
                    | atomSpecifier
atomSpecifier →     hierName/[attrName=]ID
groupParts →        groupPart
                    | groupPart,groupParts
groupPart →         groupSpecifier
                    | groupSpecifier [bondOrderSymbolChar] groupSpecifier
groupSpecifier →    [!] elementSymbol index
```

7.6.3 Literals

As mentioned in the overview, the simplest form of an atom expression is an atom specifier. An atom specifier consists of three literals: *hierName*, the optional *attrName*, and *ID*.

Medivis Exhibit 1023

hierName stands for a name of a hierarchy level (e.g., `residues`). The following abbreviations can be used for the most common levels:
a=atoms, r=residues, b=bonds, s=secondary_structure, c=chains
attrName is optional and specifies the name of an attribute (e.g., `temperature, occupancy, type, ...`) of the given level. If it is omitted, the ID is assumed to specify the attribute name or index as shown by the list command. If an attribute name is given, the ID is assumed to stand for values of the attribute.
To see which hierarchy levels and respective attributes are defined for a given molecule, take a look at the *Color port* which is used in several modules. The right pull-down menu will show all available attributes for the level chosen in the left pull-down menu.
ID specifies an identifier of a member of the given level. If an attribute name is given, ID specifies a value of this attribute. It may contain wildcards such as * (any substring will match) and ? (any single charackter will match). For integer and float attributes ranges can be used by delimiting the boundaries with a colon (i.e. atoms/index=5:10 selects all atoms with an index within this range). If no attribute name is given, the name attribute is used as a default. In this case, specifying a range will select all atoms whose index is between the index of the selected boundaries. (as an example, for a moelcule imported form PDB the residue name is the chain identifier plus the residue index, thus r/L1:L5 selects residues 1 to 5 on the L chain). An exception to this is the atoms level. Molecules in Amira contain an atomic number attribute instead of an element symbol attribute. To select atoms via their element symbol you can simply type *a/element*. Thus the atom specifier for all oxygen atoms *a/O* is equivalent to the atom specifier *a/atomic_number=8*.
Instead of the '=' comparison you can also use the comparison operators $'<', '>'$ $, '>=' \; and \; '<='$. These, however, are only available for index, integer and float attributes, not for string attributes.
Another atom expression form involving several literals is the *groupParts* expression which is used with the `GROUP` operator. Operators will be explained in the next section.

7.6.4 Operators

Logical Operators
Several *atomSpecifier* combinations can be used in one expression by linking them logically via the operators `AND`, `OR`, and `NOT` (&, |, and !). Priorities can be specified using usual parentheses `(` and `)`.

`WITHIN(atomExpr,float)`
This operator selects all atoms which are nearer than *float* Å to any of the atoms specified by *atomExpr*.
`BONDED(atomExpr[,int])`
With this operator, all atoms that are recursively connected to any atoms specified by *atomExpr* will be chosen. You can optionally specify an integer value defining the maximal bond steps. If this is omitted, there will be no limit.
`CS`
`CS` specifies all currently selected (highlighted) atoms.
`GROUP(groupParts)`
The `GROUP` operator is a powerful tool to find functional groups by searching for a certain atom and bond pattern in the molecular graph. To define a graph as a search pattern (*groupParts)*, you must divide it into linear pieces of sequential atoms (a *groupPart*). Each atom must be defined by its element symbol and an index which distinguishes it from other atoms with the same symbol but other indices (*groupSpecifier*). Thus, `C1C2C3` would be a *groupPart* consisting of three different *groupSpecifiers* representing a chain of three carbons. This group part can now be combined with another group part by using one of its *groupSpecifiers* as branch point (e.g., `C2O1H1` for a hydroxyl group branching from the second carbon of the chain). At the beginning of each group specifier, there can be an optional '!'. If it is given, the groupSpecifier is only used for matching, but the corresponding atoms will not be selected. (Thus `!C1O1H1` would find the hydroxyl groups without selecting the flanking carbon, which must be given, however, to avoid selecting structures, such as OH groups in H2O). If the atomSpecifier in question appears several times (to define branch points), it is sufficient to mark it with the exclamation mark once.
Consecutive *atomSpecifiers* in a *groupPart* are usually not divided from each other. This means that there must be a bond of any type between the consecutive atoms. If you want to define the bond order further, you can give an optional *bondOrderSymbol*. ('-' for a single, '=' for a double, '#' for a triple and '~' for an aromatic bond).
Take a carboxyl group as an example. To define the group pattern, the central carbon atom (C2) needs to be connected to three atoms: two oxygens (O1, O2), and one other carbon (C1). This can be done in the following way: `GROUP(!C1C2O1,C2O2H1)`. Thus, the hydroxyl group (O2H1) is given as a branch of the CCO chain. There are, of course, several other ways to split this branch into linear pieces which you can easily find yourself. If your molecule contains the bond type attribute, you can also make use of the double bond. Thus the

Medivis Exhibit 1023

expression becomes `GROUP(!C1-C2=O1,C2-O2-H1)`.
Notice: The keywords do not need to be in capital letters. Lower case letters, even a combination of lower and upper case letters, works as well.

7.6.5 Shortcuts

Molecular Option also provides pre-defined shortcuts that have been assembled using the previously mentioned syntactical elements. The shortcuts can be found in your local Amira directory in `share/molecules/atomExpr.cfg`, and can be edited and supplemented.
The standard aliases included in the current Amira release are listed in Table 7.1.

7.6.6 Further Examples

- `all`
  selects all atoms.
- `atoms/5-8`
  all atoms whose index is in the range 5 to 8, inclusive.
- `atoms/atomic_number>1`
  all atoms, except hydrogens
- `s/type=helix AND NOT (a/C OR a/N)`
  all atoms which belong to helices, except C and N atoms.
- `r/type=A*`
  all atoms which belong to residues whose type name begins with the letter A.
- `BONDED(a/4 OR a/100,6)`
  all atoms which are connected via at most 6 steps to the two specified atoms
- `WITHIN(r/A11,3.1) AND C`
  all carbon atoms which are not away further than 3.1 angtroms from atoms of residue 11 on chain A
- `GROUP(C1C2C3C4C5C6C1)`
  all cycles consisting of 6 carbons (e.g., cyclohexane).
- `acidic AND helix`
  all atoms of acidic amino acids which belong to helices

Medivis Exhibit 1023

acidic	acidic amino acids
acyclic	acyclic amino acids
aliphatic	aliphatic amino acids
alkali	atoms which are alkali metals
alkaliearth	atoms which are alkali earth metals
all	selects everything
amino	amino acids
aromatic	aromatic amino acids
at	adenine or thymine
backbone	atoms of protein or DNA/RNA
basic	basic amino acids
buried	amino acids usually found inside the protein
cg	cytosine or guanine
charged	charged amino acids
cyclic	cyclic amino acids
h2o	water molecules
helix	helices
halfmetallic	atoms which have half metallic properties
halogens	halogenic atoms
hetero	heterogenic atoms
hydrophobic	hydrophobic amino acids
ions	charged heterogenic atoms
metallic	atoms which have metallic properties
neutral	neutral amino acids
noblegas	atoms which have noble gas properties
nonmetallic	atoms which have nonmetallic properties
nucleic	nucleic acids
nucleicbackbone	backbone atoms of RNA/DNA
polar	polar amino acids
proteinbackbone	backbone of a polypeptide
purine	adenine or guanine
pyrimidine	cytosine or thymine
sheet	sheets
sidechain	atoms not belonging to the backbone
site	
surface	amino acids usually to be found on the surface
turn	

Table 7.1: Predefined expressions for selecting parts of molecules in Amira.

Medivis Exhibit 1023

Part III

Virtual Reality Option User's Guide

8 Virtual Reality Option User's Guide

The Virtual Reality Option is an Amira extension providing support for large tiled displays like screen arrays as well as immersive multi-wall displays like *CAVEs* and *Holobenches*, with applications ranging from extended resolution displays to true immersive virtual reality. For performance use of multiple screens or projectors, Virtual Reality Option supports either:

- parallel multi-threaded rendering on *multi-pipe* machines, which are hosting multiple graphics boards,
- parallel distributed rendering on *graphics clusters*, which are interconnected PCs equipped with graphics board (not to be confused with compute clusters).

For enhanced interaction and immersion, VR configurations can support active and passive stereo modes, soft edge blending, head tracking and advanced 3D user interaction. Any *VRCO trackd*-compatible tracking system and control devices can be used together with Virtual Reality Option. Existing Amira modules can be directly used in an immersive environment by means of 3D menus. Scripts can be used for customization. In addition, a simple API is provided, allowing an Developer Option programmer to add display modules with a specific interaction behaviour.

The documentation of Virtual Reality Option configurations is separated into the following parts:

Configuration and startup

- *Virtual Reality Option essentials*
- *Using a multi-pipe system*
- *Using a cluster system*
- *Flat screen configurations*
- *Immersive configurations*

- *Calibrating the tracking system*

Working with Virtual Reality Option:

- *3D user interaction*, including the *3D menu*
- *Writing Virtual Reality Option custom modules*

Reference sections

- *Config file reference*
- *The AmiraVR control module*
- *The ShowConfig module*
- *The tracker emulator*

8.1 Virtual Reality Option Essentials

8.1.1 Virtual Reality Option configurations

Virtual Reality Option can be configured in many different ways. A particular configuration is described in a config file under `$AMIRA_ROOT/share/config/vr` or `$AMIRA_LOCAL/share/config/vr`. The config file contains things like the layout or physical extent of the screens of the display system, information about the display sources such as the X display or the parts of the desktop mapped onto the screens, as well as optional calibration data for the tracking system.

When Virtual Reality Option is installed the Amira main window provides an additional menu labelled *VR*. This menu lists all config files found in the config directory. Once a particular configuration is selected, the *AmiraVR* module is created - if one does not already exists. Depending on the particular configuration the AmiraVR module allows the user to connect to the tracking system, to calibrate the tracking system, or to activate additional options.

Configurations are detailed in further sections.

8.1.2 Immersive interaction and the trackd daemon

For immersive displays with head tracking or interaction with 3D input devices, Virtual Reality Option makes use of the VRCO *trackd* software in order to access

Medivis Exhibit 1023

the tracking system and controller devices. Trackd itself is not part of Virtual Reality Option. For more information about purchasing and installing trackd, please refer to `www.vrco.com` or `3dviz.mc.com`.
Before a tracking system can be used in Virtual Reality Option the trackd daemon (and possibly a trackd server) has to be started. The trackd daemon connects to the tracking system and controller device and provides the actual tracker and controller data in two shared memory segments, which are read by Amira.

8.1.3 Multiple displays and parallel rendering

Virtual Reality Option can use simple configurations driven by a single graphics board, possibly using several video outputs (e.g. dual-head). It also supports configurations requiring more outputs than available from a single board (tiled or arbitrary multi-displays). Having a single screen or projector per graphics board may also be preferred for performance. This can be achieved either by using a *multi-pipe* computer hosting multiple graphics boards (section 8.2), or using a *cluster* of computers (section 8.3).
Virtual Reality Option manages *application synchronization* and *OpenGL frame buffer swap synchronization* for all displays, so that the same content can rendered simultaneously on each display, yet possibly with a different viewpoint.
A type of stereoscopic displays relies on active shutter glasses synchronized with the display, where the left and right eye images are rendered alternatively (OpenGL quadbuffered stereo). Active stereo requires synchronizing the video signals of the whole graphics boards. This can be achieved for instance with hardware frame-lock/genlock as provided by NVidia 'G' cards. Passive stereoscopy, for instance using polarizing filter glasses, does not require video frame synchronization.

8.2 Using Virtual Reality Option on a multi-pipe system

8.2.1 Configuration

Effective rendering on a multi-pipe system requires parallel rendering, i.e. sending data to the graphics engines from several threads running in parallel. Note that for all common current graphics architectures it makes no sense to render multiple windows on the same pipe in parallel. Typically, this even implies a significant performance decrease. Virtual Reality Option configurations allows to define display

Medivis Exhibit 1023

thread groups for multi-pipe rendering (more about configuration with examples in next sections).

8.2.2 Starting Amira

After Virtual Reality Option has been installed Amira can be started as usual. However, in order to activate parallel rendering of screens assigned to different thread groups, Amira should be started with the `-mt` command line options. This option enables multi-threading. In order to permanently activate multi-threading, the environment variable `AMIRA_MULTITHREAD` can be set. In order to disable multi-threading when `AMIRA_MULTITHREAD` is set, Amira can also be started with the `-st` command line option (single-threaded mode).
Note: On some graphics engines such as SGI Onyx systems there are known problems with the OpenGL driver related to the use of texture objects in different OpenGL contexts. If you are rendering more than one screen on a single pipe, you may define the environment variable `AMIRA_NO_CONTEXT_SHARING` as a workaround. This may be fixed by some OS or driver update.

8.3 Using Virtual Reality Option on a cluster

8.3.1 Overview

A graphics cluster consists of multiple computers *cluster nodes*) connected via a network. Each computer controls one or more screens of a tiled or multi-wall display system. Virtual Reality Option synchronizes the different cluster nodes, ensuring that the same scene is rendered simultaneously from different viewpoints. This cluster support enables distributed parallel rendering. It does not speed up ordinary computations by distributing them across multiple nodes of the cluster. Instead, on each cluster node a separate instance of Amira is running, with the complete modules network and all data being replicated in cluster's node memory. Changes of Amira modules made on a *master node* will be propagated to the *slave nodes* as lightweight command messages and executed synchronously: therefore the master and the slaves nodes execute the same processing steps. A special *daemon* running on slave nodes is responsible for starting the Amira instance upon requests of the master, once a cluster configuration is selected. The 2D mouse or a 3D input device can be used to interact with the application on the master node.

Medivis Exhibit 1023 286

8.3.2 Requirements

- All nodes of the cluster should be of the same type. If different hardware is used, it is possible that the cluster will run out-of-sync for instance because of round-off errors.
- Rendering speed is limited by the slowest node of the cluster, including the master node. Therefore it is recommended that all cluster nodes including the master node have the same type of graphics board.
- All nodes of the graphics cluster must be able to communicate with each other via a TCP/IP connection. A standard 10 Mbit ethernet connection may be fast enough, since only synchronization commands are exchanged, rather than images or raw data blocks.
- For active stereoscopy, the graphics cards of the different cluster nodes must be genlocked with a genlock cable. Currently only a small number of graphics cards (such as WildCat, SUN Zulu, NVidia 'G' boards) provide genlocking. Genlocking ensures that the video refresh cycles are synchronized. Virtual Reality Option itself only synchronizes OpenGL buffer swaps. Passive stereoscopy does not require genlocking.
- Installation path and data location.
  When Amira loads scripts or data below its installation directory, the relative path on the master is 'rooted' to Amira installation directory on slaves. For instance, `/master-path/Amira/data/xxx` is translated into `/slave-path/Amira/data/xxx`. However it is highly recommended that all data files and scripts have the same absolute file path on all nodes of the cluster. Likewise, Amira should be installed at the same location on all nodes. For convenience, it is recommended to run Amira from a shared mounted file system. This ensures that the same scripts and config files will be used on all cluster nodes.
- On a Linux or UNIX cluster, it is recommended to have a shared home directory. This ensures that some modules have consistent access to preferences settings stored in .AmiraRegistry.

8.3.3 Preparing cluster slave nodes

Amira requires a daemon process to run on each slave node. This sections explains how this can be achieved automatically or manually.

Medivis Exhibit 1023

Using the Amira service

Amira can manage services for automatic start-up of daemon processes such as the Virtual Reality Option daemon. You may request installation of services during Amira installation, or you can control the services with a dedicated utility program provided with Amira.
Note: To manage the Amira services, you must have administrator privileges (Windows) or root privileges (UNIX/Linux).

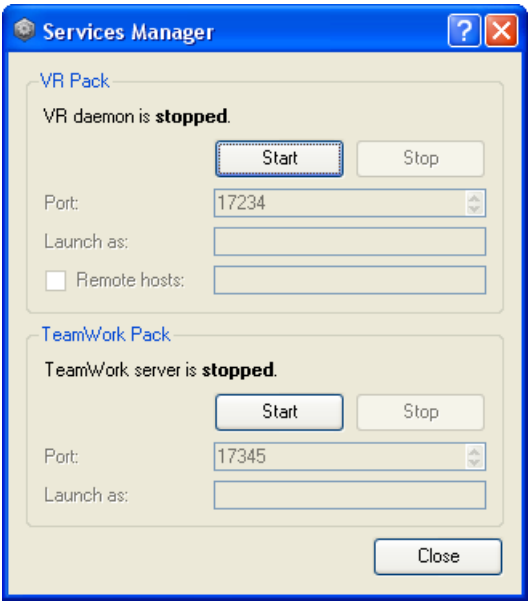


Figure 8.1: The Amira service configurator.

Just launch "`bin\arch-...-Optimize\hxservicemanager.exe`" on Windows, and "`bin/start -servicemanager`" on Linux. This tool manages the execution of Virtual Reality Option services. Indeed, instead of the user launching the Virtual Reality Option daemon manually, a service can be installed on the system so that it automatically starts these processes at system start-up.

Medivis Exhibit 1023

In order to install and start a service, click on the Start button. Then, clicking on the Stop button will stop and uninstall the service.

Platform-dependent notes

- On Windows platforms:

  Once installed, services can be started or stopped from the services manager of the system that can be launched via the Control Panel (select Administrative tools, then Services) or with the command: `services.msc /s`.

  - To start an Amira service, select it in the list ("hxvrdaemon"), then select Start from the Action menu.
  - To restart the service, select it in the list, then select Restart from the Action menu.
  - To stop the service, select it in the list, then select Stop from the Action menu.

  Warning about virtual drives: Amira services do not support logical or network volumes. Indeed, such virtual drives do not exist in the service execution context. You must specify full paths constructed according to the universal naming convention (UNC), such as "`\\remotemachine\directory`".

  For example, to install the service from a virtual drive `S:` targeting a distant mount point "`\\remotemachine\directory`", do not launch the installer from `S:`, but from "`\\remotemachine\directory`". Then, for the same reason, the user should not load data from virtual drives because Amira slaves cannot resolve virtual paths unless they are below `%AMIRA_ROOT%`, in which case data paths are re-interpreted on slaves with the local `%AMIRA_ROOT%`.

- On Unix/Linux platforms:

  Amira services are implemented as daemons, started from the "`/etc/init.d`" directory. Once installed, services can be manually started or stopped using the appropriate script. For example, type "`/etc/init.d/hxvrdaemon stop`" to stop the Virtual Reality Option daemon. Note that this will not uninstall the daemon, that will be launched on next system start-up (or next "`init 5`" mode switch in

particular).

One specific feature of Amira services on Linux platforms is that they can be executed as a given user, other than root. To do so, just type a valid login in the user field, then restart the service to take the change into account.

One other specific feature is that Virtual Reality Option daemons can be installed on remote nodes of a cluster from the master node, provided the fact that `%AMIRA_ROOT%` is the same on every machine. Just type the list of the nodes hostnames separated by space chars. Make sure that ssh is installed on all machines and configured to use RSA authentication protocol (public and private key files) so that you will not be prompted for password on any operation.

Running the daemon

If the Amira service cannot be used for some reason, one can use manual start-up. On Linux start Amira with command line option `-clusterdaemon`, on Windows start hxvrdaemon.exe
On Windows, to start the daemon automatically after the computer is booted, a shortcut to the daemon executable can be added to the Start/Programs/Startup folder.

8.3.4 Running Virtual Reality Option on cluster

The following steps are required to use cluster configurations:

1. Install Virtual Reality Option on every node of the graphics cluster under the same absolute path name, or better yet, create a mounted file system with the same absolute path name on every node.
2. Create a standard Virtual Reality Option config file describing the geometry of your display system (see the next sections about configurations). Then add a field *hostname* in each screen section. This field indicates on which node the screen will be rendered. The config file must be stored on all nodes in the directory `$AMIRA_ROOT/share/config/vr`.
3. Choose one node as the master node. After Amira installation, a daemon is started on the slave nodes, by a service on Windows systems or by inetd on Unix/Linux systems. Start Virtual Reality Option on the master and select your cluster configuration from the config menu. Slave instances of Virtual Reality Option should now be started on all nodes specified in the config file.

Medivis Exhibit 1023

To use the `-clusterdaemon` option, stop the daemon (see 8.3.3 service management) and replace step 3 by the following steps:

1. Choose one node as the master node. On all other nodes start Amira with the command line option `-clusterdaemon`. With this option a little daemon is started, to which the master process can talk. The daemon automatically starts a slave instance of the real Amira if necessary.
2. Start Virtual Reality Option on the master node. Then select your cluster configuration from the config menu. Slave instances of Virtual Reality Option should now be started on all nodes specified in the config file.
3. Next you can load any standard Amira network script. For example, open the online help browser on the master node and choose one of the standard Virtual Reality Option tracking demos. The particular script will be loaded on all slaves as well. Once a script has been loaded, all standard Virtual Reality Option interaction modes are available in cluster mode too.

8.3.5 Limitations

- The primary focus of the Virtual Reality Option cluster version is on working in a VR environment. Consequently all manipulations performed in VR mode, e.g., via the 3D menu, will be properly synchronized. On the other hand, currently not all manipulations made on the master node via the conventional 2D user interface will be synchronized.
- Any kind of 2D mouse interaction in a 3D viewer window also will not be synchronized in cluster mode. Thus you cannot draw a lasso area with the 2D mouse. Mouse manipulation and interaction are entirely disabled on slaves nodes.
- Transform editors and clip planes are synchronized in cluster mode, while other editors may be grayed out and not accessible.
- When loading a new cluster configuration file, Amira attempts to send the current network to slaves nodes for reloading after they are restarted. Data should be duplicated or shared in order to have the same path relative to script. This may also fail if default directory for temporary files are not the same path on master and slaves. You may check TMPDIR, TEMP or TMP environment variable depending on your system.

8.3.6 Troubleshooting cluster configurations

- Check that cluster slaves can be reached from master. `ping <slave-hostname>` or `ping <slave-IP-adress>` You can change hostnames with IP adresses in the configuration file.
- Check that cluster master can be reached from slave with master hostname with `ping <master-hostname>`. Otherwise, on the master node you can set the environment variable `AMIRA_HOSTNAME` to the master hostname or IP address that can be used from slaves to reach the master.
- Check that Amira runs properly on each slaves. You may need to switch keyboard and mouse to each slave nodes for checking. You may first try hint below.
- In order to show and check console output on slaves, you can reduce the channel size in SoScreen, and add in the configuration file:

```
SoVRProperty {
   showSlaveConsole TRUE
   showSlaveGUI TRUE
   showSlaveCursor TRUE
}
```

  showSlaveCursor should be set if you wish to use a mouse connected to a slave node.
- It is recommended to have one and only one screen defined for the master in the configuration file (SoScreen without specified hostname). Use SoVRProperty with keepMasterViewerInsideGUI field for convenience.
- Do not mix flat and immersive screen definitions.
- Make sure that no firewall limits communication within the cluster (port 17234 and dynamic port are used).

8.4 Flat Screen Configurations

A flat screen configuration consists of usually two or more screens forming a bigger 2D virtual graphics window commonly called a *tiled display* or *power wall*. Users can interact with the ordinary 2D mouse, i.e., mouse events in the different windows are translated and interpreted in the 2D virtual window. There is

Medivis Exhibit 1023

no need for a tracking system in case of a flat screen configuration. For such a configuration the only options provided by the AmiraVR control module are for controlling stereo viewing. Besides standard OpenGL active stereo modes, passive stereo modes can also be configured. In the most simple case this is done by defining two full screen windows on two different channels, one for the left eye view and one for the right eye view. This particular configuration is illustrated in Figure 8.2. Other passive stereo configurations are possible too, e.g., a super-wide tiled passive stereo configuration with four channels (left side left eye, left side right eye, right side left eye, right side right eye), possibly with an overlap region for soft-edge blending.

The main advantage of a flat screen configuration is its ease of use. There is no need for a tracking system. Flat screen configurations are well suited for presentations targeted to a larger audience, e.g. presentations in a seminar room or in a lecture hall.

In the following we describe some common flat screen configurations in more detail, namely a standard-resolution two-projector passive stereo configuration, a super-wide two-projector mono configuration with soft-edge blending, and a tiled 2x2 four-channel monitor configuration. For some cases also hardware solutions are available, namely special-purpose video splitters converting an interlaced active stereo signal into separate non-interlaced left eye and right eye signals for the passive stereo case, or hardware edge-blending units for blended super-wide configurations. However, these hardware solutions are usually expensive and less flexible. Therefore they are often not suitable for temporary demonstrations or experiments. We also illustrate below use of multi-pipe or cluster systems.

- *A two-channel passive stereo configuration*
- *A super-wide configuration with soft-edge blending*
- *A tiled four-channel 2x2 monitor configuration*

8.4.1 Example: A 2-channel Passive Stereo Configuration

For a single-screen passive stereo projection system two video projectors emitting orthogonally polarized light are required. One projector displays the left eye image, the other one the right eye image. This is illustrated in Figure 8.2. Both images are projected onto a non-depolarizing screen either using front projection or rear projection. Observers wear suitable light-weight polarized glasses so that each eye only sees its own image, but not the image determined for the other eye.

Medivis Exhibit 1023

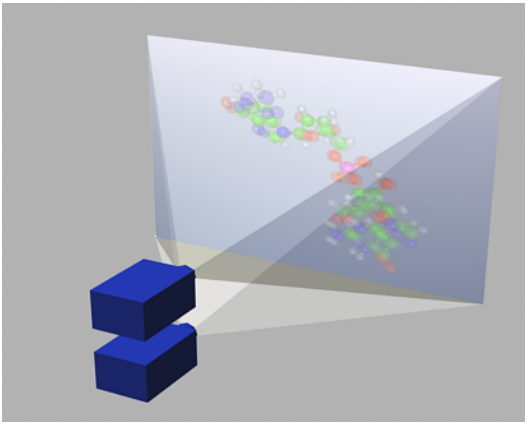


Figure 8.2: Sketch of a single-screen passive stereo projection system.

Simple dual-head wide desktop configuration

Without special-purpose hardware such a passive stereo projection system can be driven in full screen mode even using an inexpensive low-end dual-head graphics adapter. We assume that the dual-head graphics computer is configured so that the left half of the desktop is output on one channel and the right half is output on the other channel. Here is the Virtual Reality Option configuration file:

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Left eye view"
        channelOrigin   0 0
        channelSize     0.5 1
        tileOrigin      0 0
        tileSize        1 1
        cameraMode      LEFT_VIEW
    }
    SoScreen {
        name            "Right eye view"
```


Medivis Exhibit 1023

```
        channelOrigin   0.5 0
        channelSize     0.5 1
        tileOrigin      0 0
        tileSize        1 1
        cameraMode      RIGHT_VIEW
    }
}
```

The fields *channelOrigin* and *channelSize* indicate that the two windows exactly cover the left half and the right half of the desktop. The fields *tileOrigin* and *tileSize* indicate that both screens should display the full viewer window, i.e., there is no tiling at all. Finally, the field *cameraMode* indicates that one screen should display the left eye view and the other the right eye view. Note that the graphics computer need not to support active stereo for this configuration.

Controlling stereoscopy

The *AmiraVR* module provides ports in order to change the default stereo parameters *eye offset* and *stereo balance*.

Here `<offset>` denotes the eye offset and `<balance>` denotes the stereo balance, i.e., the location of the zero parallax plane. Depending on the balance value objects appear to be in front of the projection screen or behind it.
The following standard Amira Tcl command can also be used:

```
viewer 0 setStereo [-b <balance>] <offset>
```

Active-to-passive configuration

The simple configuration above may not be convenient if you often need to manipulate 2D user interface, windows and mouse cursor. Because these are displayed only once on the desktop, they are visible only on one eye and 'ghosted' because or superimposed other image. Some graphics boards or special hardware support active stereo with conversion to passive stereo, automatically replicating desktop 2D users interface on both outputs.
On Windows, you may check in the display properties panel the available settings of you graphics board for stereoscopy. On Linux, you may add active-to-passive

Medivis Exhibit 1023

settings in your X11 configuration, for instance in /etc/X11/XFree86Config or /etc/X11/Xorg.conf, depending on your specific system and graphics board. It may look like this:

```
Option "TwinView" "True"
Option "TwinViewOrientation" "Clone"
Option "Stereo" "4"
```

Cluster configurations

Here is a configuration for cluster, with 2 nodes. The graphics boards can be configured for 'single head' way with full desktop used for each eye channel. The rendering performance can be dramatically improved as each graphics board has to render only half of the frames as compared to the dual-head configuration above. One of the nodes is used as master: the master Amira is started on the master node, where the used mouse and keyboard are attached. You can notice that the hostname doesn't need to be specified for the master's screen. Also the default channel and tile origin and size (0 0 and 1 1) are used for full scene display on full screen.

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Left eye view"
        cameraMode      LEFT_VIEW
    }
    SoScreen {
        name            "Right eye view"
        hostname "slave.cluster"
        cameraMode      RIGHT_VIEW
    }
}
```

Here is a configuration for cluster with 3 nodes. A third node is used as separate master, with default MONOSCOPIC cameraMode. The manipulations of 2D user interface will then be invisible in the left/right projected screen. The *keepMasterViewerInsideGUI* field is used for convenience to force the master screen within a standard viewer window (then specifying channel size and origin would have no effect on the master).

Medivis Exhibit 1023

```
#Inventor V2.1 ascii

Separator {
    SoVRProperty {
        keepMasterViewerInsideGUI TRUE
    }
    SoScreen {
        name            "Master view"
    }
    SoScreen {
        name            "Left eye view"
        hostname "slave1.cluster"
        cameraMode      LEFT_VIEW
    }
    SoScreen {
        name            "Right eye view"
        hostname "slave2.cluster"
        cameraMode      RIGHT_VIEW
    }
}
```

8.4.2 Example: A Super-wide Configuration with Soft-edge Blending

Super-wide images with two times a standard monitor's resolution can be displayed using two projectors and a low-end dual-head graphics adapter. However, often it is very desirable to have an overlap between the two projected images in order to hide at best the transition between the projected images. In the overlap region one image softly fades out from full intensity to black, while the other image fades in from black to full intensity. This technique is called *soft-edge blending* (compare Figure 8.3). With soft-edge blending the border between the two projected images becomes almost invisible. Virtual Reality Option is able to generate two partially overlapping images. In addition, the soft-edge can be computed in software, yet some projectors or external blending units support this directly in hardware. With these features perfect full-screen demos can be presented on a super-wide projection system.

Medivis Exhibit 1023

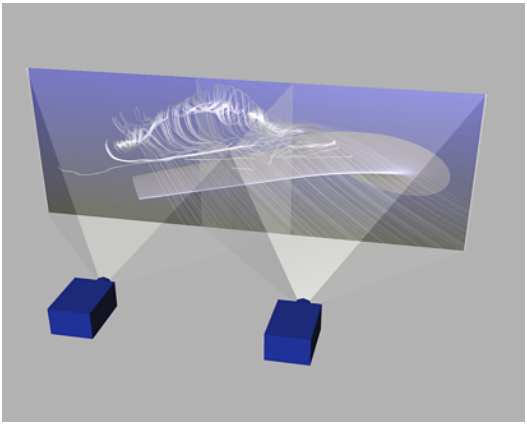


Figure 8.3: Sketch of a super-wide projection system with soft-edge blending.

Dual-head wide desktop configuration

We assume that the dual-head graphics computer is configured so that the left half of the desktop is output on one channel and the right half is output on the other channel. We further assume that there is a 20 percent overlap between the projected images of the two projectors. Here is the Virtual Reality Option configuration file:

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Left half"
        channelOrigin   0 0
        channelSize     0.5 1
        tileOrigin      0 0
        tileSize        0.6 1
        softEdgeOverlap [ 0, 0.2, 0, 0 ]
        softEdgeGamma   [ 0, 1.2, 0, 0 ]
    }
    SoScreen {
        name            "Right half"
```


Medivis Exhibit 1023

```
        channelOrigin   0.5 0
        channelSize     0.5 1
        tileOrigin      0.4 0
        tileSize        0.6 1
        softEdgeOverlap [ 0.2, 0, 0, 0 ]
        softEdgeGamma   [ 0, 1.2, 0, 0 ]
    }
}
```

The fields *channelOrigin* and *channelSize* indicate that the two windows exactly cover the left half and the right half of the desktop. The fields *tileOrigin* and *tileSize* indicate that both screens display an area of 0.6 times the width of the full tiled window. The first screen displays the left half of that window, the second screen displays the right half.

How to compute tileOrigin and tileSize fields ?

The fields tileOrigin and tileSize define to the portion of the full view volume, i.e. of the total virtual big screen, to be mapped onto a specific screen. Therefore, if *width* is the pixel width of one projector, *overlap* the amount of overlapping pixels, and *n* the number of projectors, the total pixel width of the screen is:
total = (*width* - *overlap*)*(*n*-1) + *width*
Then:
tileSize x = *width* / *total*
tileOrigin x = *screen-index* * (*width* - *overlap*) / *total*

Soft-edge blending fields

The field *softEdgeOverlap* specifies the relative width of the soft-edge region at the left, right, bottom, and top border of the screen. For the first screen there is a 20 percent soft-edge region at the right border, for the second screen there is a 20 percent soft-edge region at the left border.
Finally, the field *softEdgeGamma* specifies the gamma factor for the soft-edge region. The gamma factor determines how the fading from full intensity to black is done. A gamma factor of one means a linear transition in terms of RGB values. However, since RGB values are usually not mapped linearly to light intensity by the video projector often a decrease of overall light intensity is observed in the

Medivis Exhibit 1023

overlap region. In order to compensate for this a gamma factor larger than one can be used.
In order to facilitate the initial calibration of a super-wide projection system with soft-edge blending the AmiraVR module provides a special-purpose Tcl command *setSoftEdge*. This command is used in the following way:

```
VRSettings setSoftEdge -g <gamma> <overlap> [<overlap>...]
```

This command automatically creates a two-screen configuration similar to the one above. However, the correct values for *tileSize*, *tileOrigin*, and *softEdgeOverlap* are computed automatically from the relative overlap value. By gradually adapting this value the correct settings for a given but unknown setup of the projection system can be easily found.

Curved screen displays

Tiled display configurations are most often fine for curved displays, which are then handled like flat displays. Immersive display configuration fitting the actual screen geometry is required for using stereo with head tracking. Projection onto cylindrical or semi-spherical surfaces may also introduce some barrel-shaped distortions. Some special warping hardware can compensate such distortion.

8.4.3 Example: A Tiled 4-channel 2x2 Monitor Configuration

There are some interesting graphics workstations with two two-channel graphics pipes. One example is the SGI Octane Fuel. With such a computer four different monitors or projectors can be used. For some purposes it is useful to have a single viewer subdivided into 2x2 parts and each part being displayed by a different monitor. Such a configuration can be easily created using Virtual Reality Option.
We assume that the graphics computer has two pipes with two channels each. The two pipes are configured as two separate X11 screens, `:0.0` and `:0.1`. The left and right part of each screen is output on the two different channels of the particular pipe. Then the configuration files looks as follows:

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Upper left monitor"
```


Medivis Exhibit 1023

```
        display         ":0.0"
        channelOrigin   0 0
        channelSize     0.5 1
        tileOrigin      0 0.5
        tileSize        0.5 0.5
        threadGroup     0
    }
    SoScreen {
        name            "Upper right monitor"
        display         ":0.0"
        channelOrigin 0.5 0
        channelSize 0.5 1
        tileOrigin      0.5 0.5
        tileSize        0.5 0.5
        threadGroup     0
    }
    SoScreen {
        name            "Lower left monitor"
        display         ":0.1"
        channelOrigin   0 0
        channelSize     0.5 1
        tileOrigin      0 0
        tileSize        0.5 0.5
        threadGroup     1
    }
    SoScreen {
        name            "Lower right monitor"
        display         ":0.1"
        channelOrigin 0.5 1
        channelSize 0.5 1
        tileOrigin      0.5 0
        tileSize        0.5 0.5
        threadGroup     1
    }
}
```


Medivis Exhibit 1023

Multi-pipe rendering

Since the two X11 screens `:0.0` and `:0.1` are driven by two independent graphics pipes it makes sense to perform the rendering on these pipes in parallel. This is done by assigning the corresponding screens two different thread groups.

- Note that for all common current graphics architectures it makes no sense to render multiple windows on the same pipe in parallel. Typically, this even implies a significant performance decrease. Therefore, here we use only two thread groups instead of four.
- In order to actually activate parallel rendering Amira must be started with the command line option `-mt`. Alternatively, the environment variable `AMIRA_MULTITHREAD` can be defined.

8.5 Immersive Configurations

In Virtual Reality Option an immersive configuration differs from a flat screen configuration mainly in the way the screens are described in the config file. While for a flat screen configuration it was sufficient to specify which part of a big 2D virtual screen was covered by each screen, for an immersive configuration the true physical coordinates of the screens have to be specified. Knowing the exact spatial arrangements of the screens has the advantage that correct perspective views can be computed for all screens, provided the 3D position of the observer is also known. In particular, the different screens no longer need to be arranged in a plane. Instead, the screens might be arranged perpendicular to each other like in a *CAVE* or on a *Holobench*. In fact, Virtual Reality Option supports any other oblique arrangement as well.
Since the 3D position of the observer need to be known in order to compute correct perspective views, usually the observer's eye position is tracked in an immersive environment. In Virtual Reality Option this so-called *head tracking* can be achieved using a variety of different tracking systems. Instead of accessing the tracking system directly an intermediate software layer is used, namely the *trackd* software of VRCO. *trackd* runs as a server, communicates with the tracking system, and stores the tracking data in a shared-memory areas which then is read by Virtual Reality Option. Using a second sensor not only the observer's eye position can be tracked, but also the position of a virtual wand, i.e., a kind of pointing or interaction device to be held in a hand. For large planar screen configurations it sometimes also makes sense to use a virtual wand without head tracking. The images then

Medivis Exhibit 1023 302

will always be computed for a fixed observer's eye position and possible image flicker due to noise in the tracking data is avoided, which is an advantage for 3D demonstrations in front of a larger audience.
Obviously, immersive configurations are more difficult to setup than flat screen configurations. Besides defining a suitable Virtual Reality Option configuration file, also the tracking system and the *trackd* daemon have to be properly initialized. Finally, the tracking system has to be calibrated for use with Virtual Reality Option. In the following sections we first want to present some example config files for common immersive environments. In particular config files for a single-wall workbench, for a double-wall Holobench, and for a four-sided CAVE shall be discussed. The process of calibrating the tracking system and customizing 3D user interaction is described in subsequent sections.

- *A Workbench configuration*
- *A Holobench configuration*
- *A CAVE configuration*

8.5.1 Example: A Workbench Configuration

A single-screen 3D projection system is often called an immersive workbench. The actual projection screen can either be in up-right position, or it can be oriented like the surface of a table. Of course, any other orientation is possible too. There are even devices like the Barco Baron with can be arbitrarily tilted between fully horizontal and fully vertical position. For Virtual Reality Option there is no difference between these configurations, as long as the tracking system is calibrated in the right way. Traditionally a workbench is driven by a CRT projector using active stereo. However, today also passive stereo systems based on two LCD or DLP projectors become more common. In the config file below we assume that an active stereo system is used, or that a passive stereo system is connected via an appropriate signal splitter. This means that the workbench can be used with any standard stereo-capable graphics computer. No dual-head or multi-pipe computer is necessary. An example of a single-screen workbench is shown in Figure 8.4.
An Virtual Reality Option config file for driving a single-screen immersive workbench with a tracked 3D input device but without head tracking is listed below. Instead of actually tracking the observer's eye position in this case a fixed default camera position is used. This can be useful for demonstrations in front of small or medium-size groups, since it produces less fidget images. In addition, it saves a second sensor for the tracking system.

Medivis Exhibit 1023

Figure 8.4: A single-screen 3D projection system.

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Workbench"
        lowerLeft       0 0 0
        lowerRight      170 0 0
        upperRight      170 130 0
        upperLeft       0 130 0
        cameraMode      ACTIVE_STEREO
    }
    SoTracker {
        server          "4147:4148"
        autoConnect     TRUE
        wandTrackerId   0
        headTrackerId   -1
        defaultCameraPosition 85 65 140
        defaultObjectPosition 85 65 0
        referencePoints [0 0 0,170 0 0,170 130 0,0 130 0]
    }
}
```


Medivis Exhibit 1023

Single screen configuration

In the *SoScreen* section of the config file the physical coordinates of the four corners of the workbench are defined. This can be done using an arbitrary right-handed coordinate system with arbitrary units. In this case the coordinates might be specified in centimeters. The origin of the coordinate system was chosen in the lower left corner of the screen. By default, a full-screen graphics window is opened when activating this configuration. The field *cameraMode* indicates that the graphics window should be opened in active stereo mode by default.

Tracking system configuration

In the *SoTracker* section of the config file the tracking system is described. First the shared memory ids of the trackd daemon as specified in the *trackd.conf* file are listed.
The syntax is `Id-of-controller-reader:Id-of-tracker-reader`. For more information about setting up trackd please refer to section 8.6. The *autoConnect* field indicates that a connection to the trackd server should be established automatically as soon as the configuration is activated. *wandTrackerId* denotes the trackd sensor id of the 3D input device. Head tracking is disabled by setting *headTrackerId* to -1. Instead the default camera position set in the line below is used. This position must be specified using the same coordinate system as the screen. In this case the camera is located 140 cm in front of the screen's center. The default object position is the position where the scene is placed by default (or whenever a *view all* request comes from the viewer).
At the end of the config file four reference points are specified, namely the four corners of the screen. Before the configuration can be actually used, the tracking system has to be calibrated (see section 8.6). This is done by placing the input device at the reference points and clicking an input button. Once the tracking system is calibrated, the config file should be written by clicking the *write config* button of the AmiraVR control module. The new config file will contain the same information listed above, but in addition it will also contain some calibration data, e.g., the transformation between raw tracker coordinates and screen coordinates.

8.5.2 Example: A Holobench Configuration

A Holobench (TM) - or dual-screen workbench - is a special display system consisting of two screens oriented perpendicular to each other. One screen is oriented

Medivis Exhibit 1023

vertically, the other one is oriented horizontally like a table. This provides a wide field display with more space free for hand interaction and manipulation as compared to a single-screen display. On a good Holobench there is almost no visible border between the two screens. Provided the observer's eye position is known, correct perspective views can be computed so that the displayed scene finally does not appear to have any break. However, in contrast to a single-screen workbench this is only the case for the tracked observer. Other spectators will more or less clearly notice a break.
In order to drive a workbench at least a stereo-capable dual-head graphics computer is required. In principle, two different settings are possible. Either, there is a big desktop and one half of it is output on a first channel and the other one is output on a second channel. Or, there are two independent graphics adapters (pipes) which are configured as two different X11 displays or as one display with two X11 screens. In the config file below we assume, that the desktop is split into two halves (left and right). An example of how to use a multi-pipe graphics computer is presented in the next section where a 4-sided CAVE configuration is described.

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Vertical Screen"
        lowerLeft       0 0 0
        lowerRight      180 0 0
        upperRight      180 110 0
        upperLeft       0 110 0
        channelOrigin   0 0
        channelSize     0.5 1
        cameraMode      ACTIVE_STEREO
    }
    SoScreen {
        name            "Horizontal Screen (rotated)"
        lowerLeft       180 0 0
        lowerRight      0 0 0
        upperRight      0 0 110
        upperLeft       180 0 110
        channelOrigin   0.5 0
        channelSize     0.5 1
        cameraMode      ACTIVE_STEREO
    }
```


Medivis Exhibit 1023 306

```
    SoTracker {
        server          "4147:4148"
        autoConnect     TRUE
        wandTrackerId   1
        headTrackerId   0
        leftEyeOffset   6 0 0
        rightEyeOffset  13 0 0
        defaultCameraPosition 90 55 110
        defaultObjectPosition 90 20 20
        referencePoints [60 0 110,120 0 110,120 0 55,60 0 55]
    }
}
```

Display configuration

In the two *SoScreen* sections of the config file the geometry of the Holobench is described by specifying the physical coordinates of the four corners of each screen. An arbitrary right-handed coordinate system with arbitrary units can be chosen. Here the coordinates are specified in centimetres and the origin was put in the lower left corner of the vertical screen. Notice, that the horizontal screen was rotated by 180 degrees with respect to the vertical screen. I.e., instead of the upper left corner the lower right corner of the horizontal screen is located at the origin. This is because the horizontal image of a Holobench is usually projected that way. If the corresponding lower scan-lines of the two images meet at the border between the two screens artifacts due to delayed response of the active shutter glasses are avoided. The fields *channelOrigin* and *channelSize* indicate that the graphics window for the vertical screen should be opened on the left half of the desktop, while the graphics window for the horizontal screen should be opened on the right half. Both windows are opened in stereo mode by default as specified by *cameraMode*.

Tracking system configuration

In the *SoTracker* section of the config file the tracking system is described. First the shared memory ids of the trackd daemon as specified in the *trackd.conf* file are listed.
The syntax is `Id-of-controller-reader:Id-of-tracker-reader`. For more information about setting up trackd please refer to section 8.6. The *autoConnect* field indicates that a connection to the trackd daemon should be es-

Medivis Exhibit 1023

tablished automatically as soon as the configuration is activated. *wandTrackerId* denotes the trackd sensor id of the 3D input device. *headTrackerId* denotes the trackd sensor id of the head sensor which is usually mounted at the shutter glasses. The fields *leftEyeOffset* and *rightEyeOffset* specify the actual position of the eyes with respect to the head sensor. Standing in front of the Holobench the x-axis points horizontally to the right, the y-axis points upwards, and the z-axis points towards the observer. In this case we assume the head sensor to be mounted on the left side of the glasses since both left and right eye have a positive offset in x-direction.
The *AmiraVR* control module allows for interactive adjustment of the eye offset.

The default camera position and the default object position both are specified in the same coordinate system as the screens. The default camera position is only used if the tracking system is disconnected. The default object position is the position where the scene is placed by default (or whenever a *view all* request comes from the viewer).
At the end of the config file four reference points are specified, namely four points on the horizontal screen. Before the configuration can be actually used, the tracking system has to be calibrated (see section 8.6). This is done by placing the input device at the reference points and clicking an input button. The reference points were chosen so that they can be easily accessed with the hand. In addition, it is important that the points are well inside the operating range of the tracking system. During calibration the raw coordinates of the wand sensor are displayed, so you can check if these values are reasonable. Once the tracking system is calibrated, the config file should be written by clicking the *write config* button of the AmiraVR control module. The new config file will contain the same information listed above, but in addition it will also contain some calibration data, e.g., the transformation between raw tracker coordinates and screen coordinates.

8.5.3 Example: A 4-side CAVE Configuration

A CAVE (TM) is a more or less fully immersive VR display system with the shape of a cubical box. Typically the size of the box is something like 3 x 3 x 3 meters. Three, four, five, or even six sides of the box are implemented as projection screens. In this way an observer inside the box can be completely immersed in a

Medivis Exhibit 1023

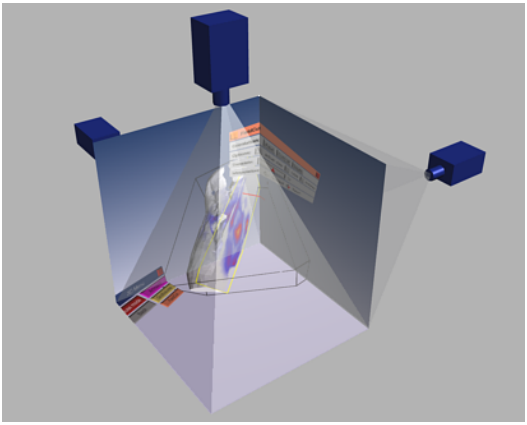


Figure 8.5: Schematic view of a 3-side CAVE system.

virtual world. In order to compute the correct perspective views, the eye position of the observer need to be tracked. Other non-tracked observers will perceive distorted images, especially at the edges between the individual walls. A schematic view of a 3-side CAVE is shown in Figure 8.5.
The Virtual Reality Option config file listed below was designed for a 4-side CAVE. In order to drive such a system four different images need to be generated, one for the front, bottom, left, and right wall respectively. This can be done for example using a two-pipe SGI Onyx system. Each pipe has two channels. Thus it is able to output two different images. We assume that there is one X server running on the machine. The two pipes are configured as two independent X11 screens, denoted `:0.0` and `:0.1`. On each X11 screen there is a double-wide desktop. The left half and the right half of the two desktops are output by the two different channels of each pipe. For such a setting the Virtual Reality Option config file looks as follows:

```
#Inventor V2.1 ascii

Separator {
    SoScreen {
        name            "Front"
        display         ":0.0"
        lowerLeft       0 0 0
        lowerRight      300 0 0
```


Medivis Exhibit 1023

```
        upperRight      300 300 0
        upperLeft       0 300 0
        channelOrigin   0 0
        channelSize     0.5 1
        cameraMode      ACTIVE_STEREO
        threadGroup     0
    }
    SoScreen {
        name            "Bottom"
        display         ":0.0"
        lowerLeft       0 0 300
        lowerRight      300 0 300
        upperRight      300 0 0
        upperLeft       0 0 0
        channelOrigin   0.5 0
        channelSize     0.5 1

        # If the bottom image is rotated try this:
        # lowerLeft       300 0 0
        # lowerRight      0 0 0
        # upperRight      0 0 300
        # upperLeft       300 0 300

        # This modifies the gradient background
        backgroundMode  BG_LOWER
        cameraMode      ACTIVE_STEREO
        threadGroup     0
    }
    SoScreen {
        name            "Left"
        display         ":0.1"
        lowerLeft       0 0 300
        lowerRight      0 0 0
        upperRight      0 300 0
        upperLeft       0 300 300
        channelOrigin   0 0
        channelSize     0.5 1
        cameraMode      ACTIVE_STEREO
        threadGroup     1
    }
    SoScreen {
        name            "Right"
        display         ":0.1"
        lowerLeft       300 0 0
        lowerRight      300 0 300
        upperRight      300 300 300
        upperLeft       300 300 0
```


Medivis Exhibit 1023

```
        channelOrigin   0.5 0
        channelSize     0.5 1
        cameraMode      ACTIVE_STEREO
        threadGroup     1
    }
    SoTracker {
        server          "4147:4148"
        autoConnect     TRUE
        wandTrackerId   1
        headTrackerId   0
        leftEyeOffset   6 0 0
        rightEyeOffset  13 0 0
        defaultCameraPosition 50 50 100
        defaultObjectPosition 50 50 0
        referencePoints [
        0 150 100, 0 150 100, 100 150 0, 200 150 0, 300 150 100 ]
    }
}
```

8.6 Calibrating the Tracking System

8.6.1 Overview

This section describes how to calibrate a tracking system for use in Virtual Reality Option. Here, calibration essentially means finding the transformation between the raw tracker coordinates and the coordinates in which the geometry of the display system, i.e., the corners of the screens, has been defined. Amira allows users to do this interactively by moving a tracked device sequentially to specified *reference points*. Calibrating the tracking system does not involve changing the trackd config file *trackd.conf* in any way. Instead the calibration data is completely stored in the Virtual Reality Option config file.

Usually the tracking system need to be calibrated only once. A new calibration is necessary for example if the antenna of an electromagnetic tracking system is moved with respect to the display system, if the screen of a single-wall immersive workbench is tilted, or if a new 3D input device is used where the 3D sensor is oriented in a different way.

The only part of the calibration process which one might repeat many a time is picking the wand. Stereo eye offset for may also need to be adjusted sometimes depending on the user.

Medivis Exhibit 1023

Wand offset

Picking the wand means to specify the offset between the actual position of the 3D input device and the visual representation of the wand in Amira. Sometimes it is useful not to have any offset, while sometimes an offset provides more pleasant and less exhausting working conditions. The wand offset can be quickly adapted even multiple times within a single Virtual Reality Option session.

Eye offset

Eye offset for stereoscopy can be defined in the configuration file. It is not a mandatory step of the interactive calibration process described below. However the AmiraVR control module allows for eye offset adjustment at any time (if stereo is enabled), and you can then save the configuration.

8.6.2 Tracking system essentials

Before starting calibration make sure that you have a valid Virtual Reality Option config file for your particular display system (you may test it independently from tracking system). Also make sure that the trackd software and the tracking system itself are setup in the right way.

Troubleshooting the tracker

A number of factors can affect the tracking system and the returned sensors coordinates, and should be kept in mind when calibrating and using Virtual Reality Option, in particular for choosing the reference points used for calibration. Here is summary of the most common possible issues:

- Range. Symptom: instable or wrong tracker coordinates.
  The calibration reference points or working area may be too far away or hidden from the tracking system. You may check the physical range of your device. You may need to choose reference points closer to tracker, or off the screen(s), for instance using ruler to offset in front of the screen.
- Hemisphere. Symptom: mirrored coordinates along one axis, reversed rotations after calibration
  You should make sure that the tracker coordinates are reported in a right-handed coordinate system. Many electromagnetic tracking systems cannot distinguish between two opposite hemispheres. In the *trackd.conf* file you

Medivis Exhibit 1023

have to specify in which hemisphere you want to operate (e.g. +x, -z..., relative to emitter antenna orientation). If the wrong hemisphere is specified a left-handed coordinate system is used, which cannot be correctly calibrated in Amira.

- Magnetic perturbation. Symptom: instable/wrong coordinates.
  With electro-magnetic tracking systems you should avoid calibrating and working too close to metallic parts such as ferro-magnetic screen frames or sources of magnetic fields such as CRT display monitors. You may need to offset reference points off the screen.
- Acoustic perturbation. Symptom: instable/wrong coordinates.
  With acoustic or hybrid tracking systems, you may need to avoid calibrating and working too close from emitters/receivers. You may need to offset reference points off the screen.
- Optical perturbation. Symptom: inaccurate coordinates, intermittent or no detection.
  With optical trackers, some possible issues are occlusion, loss of accuracy at the limits of view field, insufficient or excessive lighting. Depending on cameras location, you may need to offset reference points off the screen.
- Sensor connection. Symptom: no coordinate change, or exchanged head/wand sensors.
  The serial port can be disconnected or busy (check system configuration). Wrong head or wand sensor identifier may be specified in configuration file (headTrackerId, wandTrackerId). Amira can display on the screen coordinates and button status for checking.
- Heat. Symptom: intermittent failures.
  Some hardware may become unstable with excessive heat. Make sure cooling is sufficient.
- Report rate. Symptom: irregular moves.
  Report rate from trackd may need to be lowered with slow computers or increased for better accuracy. The symptom can also relate to graphics performance, with large data and low end hardware. Remember that using a multi-pipe or cluster rendering system can accelerate performance.

Tracking configuration examples

The *VRCO's trackd* daemon is used to collect device input for Amira. It is typically started with:

Medivis Exhibit 1023

`trackd [-file <trackd-configuration-file>]` (trackd.conf by default).
Please refer to the trackd documentation for details on how to use and configure trackd.
Here is an example of trackd.conf configuration file:

```
# Device: e.g. Ascension's Flock-of-Birds 2 sensors
DefineDevice FOB fobirds 2

# standalone box (device-specific option)
DeviceOption FOB srt 1

# device attached to serial port com1
DeviceOption FOB port com1

# device communication baud rate
DeviceOption FOB baud 38400

# reset device at start (optional, delays startup)
DeviceOption FOB reset yes

# use 3D mouse controller (device-specific)
DeviceOption FOB mouse 1

# working hemisphere
DeviceOption FOB hemisphere +x

# sampling factor (device-specific, optional)
DeviceOption FOB reportrate 2

#Defines output for tracker info (head and wand position+orientation):
# report 2 sensors to application
DefineConnector Shm1 shm out 2

# tracker shared memory key
ConnectorOption Shm1 key 4148

# position+orientation for each sensor device
ConnectorOption Shm1 data tracker

#Defines output for controller info (3D mouse/wand buttons...):
# report 1 controller to application
DefineConnector Shm2 shm out 1

# controller shared memory key
ConnectorOption Shm2 key 4147
```


Medivis Exhibit 1023

```
# buttons+valuators state for each controller device
ConnectorOption Shm2 data controller
```

Here is an example of Amira VR configuration with corresponding tracking information:

```
Separator {
    SoScreen {
        name            "desk"
        lowerLeft        0 0 0
        lowerRight       170 0 0
        upperRight       170 130 0
        upperLeft        0 130 0
        cameraMode       ACTIVE_STEREO
    }
    SoTracker {
        server           "4127:4126"
        wandTrackerId    0
        headTrackerId    1
        defaultCameraPosition 85 65 140
        defaultObjectPosition 85 65 0
        referencePoints [ 0 0 0, 170 0 0, 170 130 0, 0 130 0 ]
    }
}
```

- Notice the shared memory keys specified in *server* field, corresponding to those specified in the trackd.conf file above
- The wandTrackerId and headTrackerId must correspond to the tracker's sensors actually used for head and wand.
- Reference points for calibration are located at the corners of the screen. One should make sure that these points can be properly reach by the tracking system.

Some trackd tips

- *trackd -status* displays configuration information, then once per second the device inputs such as coordinates, orientation, buttons state.... Notice that Amira can also report tracker coordinates on the screen (see below).
- Input devices can be attached to remote machines. A trackd server can then fowards inputs to the trackd daemon running on the application machine, for which the trackd license must be set.
- trackd supports DirectX devices (mouses, 3D joysticks, etc...). Notice that

Medivis Exhibit 1023

device name can be dependent on international language used on your computer.

- A controller device can be used to emulate a tracker.

8.6.3 Calibration step by step

Here is a complete step-by-step description of the calibration process.

1. Make sure that the trackd software is running and that the tracking system is operating.
2. Start Amira and choose the appropriate configuration from the main window's *VR* menu.
3. In the AmiraVR control module, *connect* to the tracking system if necessary. Activate the *values* toggle in order to display the current 3D coordinates in the upper left corner of the main screen. Make sure that the coordinates are changing if you move the 3D input device and the head sensor (unless head tracking is disabled in the Virtual Reality Option config file). Also make sure that the button status changes if you press a button of the 3D input device.

   Note: Unless you are in calibration mode the coordinates reported by the *values* option are transformed coordinates, i.e., they are in the same coordinate system in which the screens in the config file have been defined.

   Hint: If the wand tracker and the head tracker seem to be exchanged modify the fields *wandTrackerId* and *headTrackerId* in the Virtual Reality Option config file.

4. Click the *Calibrate* button of the AmiraVR control module. You are now in calibration mode. A fixed 2D control grid is displayed on all screens. You are requested to click at the first reference point (highlighted in red). In the current version of Virtual Reality Option, the control grid is not guaranteed to match the actual reference points. However, you could choose the reference points in the config file to be at the corners of the control grid (one third and two thirds of the screen width in horizontal direction, one half of the screen height in vertical direction). Besides the number of the reference point its coordinates as specified in the config file are displayed in the upper left corner of the main screen.

   Now move the 3D input device at the first reference point and click any button of the device.

Medivis Exhibit 1023 316

Note: Make sure that the raw tracker coordinates which are displayed in the upper left corner of the main screen are valid at the reference point. Some devices just report zero values if the sensor is outside the operating range of the tracking system. Some other devices report non-sense values, typically with large oscillations. If you don't get stable values at a reference point, choose some other point in the Virtual Reality Option config file.

5. Next you are asked to click at second reference point using the 3D input device. Repeat the above procedure until all reference points have been located.

   Note: If less than three different reference points are specified in the config file, then by default the upper left, lower left, and lower right corners of the first screen are used as reference points. You may want to specify other reference points or a larger number of reference points in order to improve accuracy. *The reference points must not lie all on one line*. However, they may well be located in a common plane, e.g., in the plane of the main screen.

6. Next, you need to align the glasses for head tracking. However, if head tracking is disabled in the Virtual Reality Option config file, this step is omitted.

   Align the glasses parallel to the x-axis of the screen coordinate system. Usually the x-axis will oriented horizontally with respect to the front screen of the display system. The up-direction of the glasses should be aligned parallel to the y-axis of the screen coordinate system. Usually the y-axis will be oriented vertically with respect to the front screen. Once you have aligned the glasses click any button of the 3D input device.

7. Now camera calibration is done and correct stereoscopic images should be displayed. As a final step you are requested to pick the wand in order to define the wand offset. The wand avatar is displayed half-way between the current eye position (or the default camera position, if head tracking is disabled) and the default object position. Now move the 3D input device near the origin of the wand (or at any other position) and click any button. The virtual wand remains stuck to the 3D input device in exactly that position. You can repeat this last calibration step any time by clicking on the *pick wand* button of the AmiraVR module.

Medivis Exhibit 1023

Hint: If you don't get a clear 3D image of the wand make sure left and right eye images are not exchanged. If the eyes are exchanged you get a result which somehow also looks 3D but which isn't comfortable to work with at all. Also make sure that the fields *leftEyeOffset* and *rightEyeOffset* are set correctly in the config file. You can optionally tune stereo eye offset with AmiraVR control module.

8. At this point, the calibration is finished. You may now want to write a new Virtual Reality Option config file (or to overwrite the original one) in order to save the calibration data permanently. This can be achieved most easily by pressing the *write config* button of the AmiraVR control module. Pressing this button causes the Amira file browser to be activated. You can accept the name of the previous config file or choose a new one.

   When reloading the new configuration calibrated tracker data will be generated automatically. You can verify the correctness of the calibration process by turning on the *values* toggle and moving the 3D input device to one of the reference points. The reported values should be almost the same as the coordinates of the reference point specified in the config file.

Manual calibration

In some case you may want to enter directly in the configuration file the calibration information corresponding to the defined reference points. The *calibration* field contains the tracker coordinates for 3 points with following coordinates relative to the screen's reference defined by:
origin x=0 y=0 z=0
point1 x=X y=0 z=0, where X is the physical width of the first screen in config file
point2 x=0 y=1 z=0
For instance considering a screen of 170x130 cm, with tracker origin located in the center/botton of the screen and tracker coordinates reported in meters, with same axis x, y, z.
- ReferencePoints: [0 0 0, 170 0 0, 170 130 0, 0 130 0]
- Corresponding tracker coordinates: [0 0 0, 17 0 0, 17 13 0, 0 13 0]
- Resulting calibration field: calibration [0 0 0, 17 0 0, 0 0.1 0]

Medivis Exhibit 1023

Checking immersive displays

Here are some further hints for checking whether the immersive system is configured and working properly when used with stereo and head tracking:

- Does the virtual wand avatar follow the actual wand rotations ?
  With electromagnetic tracker, if rotations look wrong or reversed along some axis, the wand has probably crossed the tracker's working hemisphere when calibrating. Change the working hemisphere in trackd.conf and calibrate again.
- Stereo disparity, reversed eyes ?
  If stereo effect looks too strong, uncomfortable or inconsistent, check screen coordinates in the configuration file and eye offset in configuration file or with the AmiraVR module. For reversed eyes you may either change eye offsets, or change your system's display properties, e.g. 'stereo reverse eyes' in display control panel.
- Is the plane of projection located on the physical screen ? The screen projection of virtual objects located behind physical screen should look smaller as eyes get closer to the screen. Projection of virtual points located on the physical screen shouldn't move as eyes are moving. Otherwise check screen coordinates and tracker calibration.
- With multiple screens, do straight lines keep straight across screen edges ? Otherwise check screen coordinates, calibration, tracking, eye offset.
- Does wand avatar follow the sensor ? Otherwise check calibration and screen coordinates.

8.7 3D User Interaction

Virtual Reality Option flat screen configurations, i.e., tiled displays or power walls, can be completely controlled using an ordinary 2D mouse. However, in case of true immersive configurations this isn't feasible. Therefore, several ways of 3D user interaction are provided in Virtual Reality Option. The view can be changed using different navigation modes, a 3D menu is provided in order to control modules in 3D, and special objects like slices, selection boxes, or 3D point probes can be directly picked and manipulated using a tracked input device. In addition, a 2D mouse mode is provided allowing the user to control the conventional 2D mouse pointer using the tracked 3D mouse. In the following these features will be

Medivis Exhibit 1023

described in more detail.
In order to utilize 3D user interaction a tracked 3D input device with at least one button is required. By default Amira only interprets a single button. The easiest way to make use of additional buttons is to associate Tcl procedures with these buttons. This is described below in a section about *Tcl event procedures*.

- *The 3D menu*
- *User-defined 3D menu items*
- *3D module controls*
- *Navigation modes*
- *Tcl event procedures*
- *The 2D mouse mode*

8.7.1 The 3D Menu

The Virtual Reality Option 3D menu provides buttons allowing the user to bring up a 3D version of the user interface of every object in the Pool. In addition it can contain any number of user-defined buttons. These buttons can be configured most easily using the Tcl interface described in Section 8.7.2.
By default, the menu also contains a button for activating the *2D mouse mode*. This mode lets you control the 2D mouse using a 3D input device. In this way also the standard 2D user interface of Amira can be used in a VR environment.
The basic shape of the Virtual Reality Option 3D menu is shown in Figure 8.6 on the left hand side. The menu can be manipulated in the following way:

- Initially, after activating or reloading an Virtual Reality Option configuration, the 3D menu is hidden. In order to bring up the menu double-click the button of the 3D input device. Alternatively, you may select the 3D menu toggle button of the AmiraVR control module. By default the menu will be positioned in the upper left corner of the first screen defined in the Virtual Reality Option config file. You may change the default position by setting the field *menuPosition* in the tracker section of the Virtual Reality Option*config file*. There is also a field for setting the default orientation of the menu.
- You can move the menu by pointing the wand on the blue header labeled *3D menu*, clicking the main button of the 3D input device, and then moving the device while keeping the button pressed. When the wand is on the blue header a yellow frame is displayed, indicating that this element has focus.

Medivis Exhibit 1023

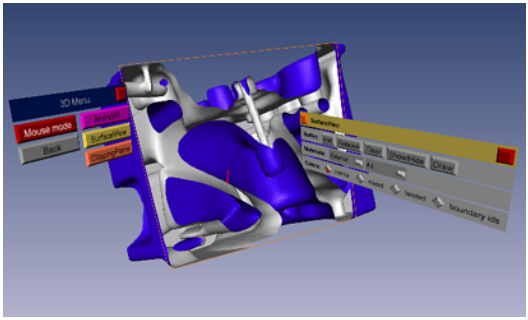


Figure 8.6: Virtual Reality Option scene with 3D menu on the left and 3D user interface of the *SurfaceView* module on the right.

If you pop up the menu using the 3D menu toggle, the menu position will be reset to its default. In contrast, if you pop up the menu by double-clicking the 3D mouse button, the menu will be shown at its current position.

- In order to hide the menu again, click the red *close button* at the right side of the blue header element. Double-clicking the 3D input device again pops up the menu at its previous position.

The 3D menu will also be available as an ordinary 2D sub-menu under the VR menu of the Amira main window. The 2D menu has the same structure as the 3D menu, and choosing an item from the 2D menu triggers the same actions as choosing an item from the 3D menu. This feature allows you to prepare and test a 3D menu on an ordinary desktop computer without an attached tracking system.

8.7.2 User-defined 3D Menu Items

Besides the default buttons, any number of user-defined buttons can be added to the Virtual Reality Option 3D menu using a Tcl script interface. This is useful for executing certain demos or for invoking other special actions. The script interface essentially consists of a single global Tcl method called *menu* which is provided by Virtual Reality Option. Calling the *menu* method with special parameters allows the user to add individual buttons or submenus to the main 3D menu. Each button or submenu has an id. This id can be used to modify the particular element afterwards. Whenever a button is pressed, a user-defined Tcl procedure is invoked.

Medivis Exhibit 1023

An example of how to create a two-level menu hierarchy is shown below. This example is taken from the file *$AMIRA_ROOT/share/demo/tracking/demomenu.hx* contained in the Virtual Reality Option demo section. You can execute that script in order to see how the user-defined buttons work.

```
menu reset

menu insertMenu -id 0 -text "Medical"
menu insertMenu -id 1 -text "Flow Dynamics"
menu insertMenu -id 2 -text "Reconstruction"
menu insertMenu -id 3 -text "Multi-Channel"

menu 0 insertItem -id 0 -text "CT Slices" -proc "Menu0"
menu 0 insertItem -id 1 -text "Isosurface" -proc "Menu0"
menu 0 insertItem -id 2 -text "Surface model" -proc "Menu0"
menu 0 insertItem -id 4 -text "Oblique slice" -proc "Menu0"
menu 0 insertItem -id 5 -text "Pseudo-color" -proc "Menu0"

menu 1 insertItem -id 0 -text "Wing" -proc "Menu1"
menu 1 insertItem -id 1 -text "Turbine ISL" -proc "Menu1"
menu 1 insertItem -id 2 -text "Turbine LIC" -proc "Menu1"

menu 2 insertItem -id 0 -text "Slice & Isosurface" -proc "Menu2"
menu 2 insertItem -id 1 -text "Volume Rendering" -proc "Menu2"
menu 2 insertItem -id 2 -text "Simplified Surface" -proc "Menu2"

menu 3 insertItem -id 0 -text "Projection view" -proc "Menu3"
menu 3 insertItem -id 1 -text "Slicing" -proc "Menu3"
menu 3 insertItem -id 2 -text "Isosurface" -proc "Menu3"
menu 3 insertItem -id 3 -text "Volume Rendering" -proc "Menu3"

proc Menu0 { id } {
    global AMIRA_ROOT
    switch $id \
        0 { load $AMIRA_ROOT/share/demo/medical/ctstack.hx } \
        1 { load $AMIRA_ROOT/share/demo/medical/isosurface.hx } \
        2 { load $AMIRA_ROOT/share/demo/medical/surf.hx } \
        3 { load $AMIRA_ROOT/share/demo/medical/tetra.hx } \
        4 { load $AMIRA_ROOT/share/demo/medical/gridcut.hx } \
        5 { load $AMIRA_ROOT/share/demo/medical/pseudocolor.hx } \
        6 { load $AMIRA_ROOT/share/demo/medical/splats.hx }
}

proc Menu1 { id } {
    global AMIRA_ROOT
    switch $id \
        0 { load $AMIRA_ROOT/share/demo/cfd/wing.hx } \
        1 { load $AMIRA_ROOT/share/demo/cfd/turbine-isl.hx } \
        2 { load $AMIRA_ROOT/share/demo/cfd/turbine-lic.hx }
}

proc Menu2 { id } {
    global AMIRA_ROOT
    switch $id \
        0 { load $AMIRA_ROOT/share/demo/recon/recon01.hx } \
        1 { load $AMIRA_ROOT/share/demo/recon/recon05.hx } \
        2 { load $AMIRA_ROOT/share/demo/recon/recon04.hx }
}

proc Menu3 { id } {
    global AMIRA_ROOT
    switch $id \
        0 { load $AMIRA_ROOT/share/demo/multichannel/projectionview.hx } \
        1 { load $AMIRA_ROOT/share/demo/multichannel/slicing.hx } \
        2 { load $AMIRA_ROOT/share/demo/multichannel/isosurfaces.hx }
        3 { load $AMIRA_ROOT/share/demo/multichannel/voltex.hx }
}
```

The example above give a general idea of how to create a user-defined button

Medivis Exhibit 1023

hierarchy. The *menu* command provides some additional parameters. The general form of the *menu* command is as follows:

```
menu [submenu-id [...]] cmd [options]
```

Here `submenu-id` is the id of any submenu. The root level has no id at all. The first level has one id, the second level has two ids, and so on.

- a button for activating the 2D mouse mode (id 1000)
- a button for getting a list of all modules (id 1001)
- a button for getting a list of all data objects (id 1002)

You can remove these default buttons using the command `menu clear`. In order to clear the menu and then reset the default buttons, use `menu reset`. The complete list of commands is this:

menu reset
: Removes all buttons and sub-menus from the menu and restores the default layout with entries for mouse mode, modules, and data objects. This command can only be applied at the main level.

menu [...] clear
: Removes all buttons and sub-menus from the specified menu. If applied at the main level also the default buttons (mouse mode, modules, and data) will be removed.

menu [...] count
: Returns the number of entries (action buttons and sub-menu buttons) in the specified menu.

menu [...] idAt *index* Returns the id of the item at position *index*.

menu [...] insertItem [options]
: This command creates a new button and inserts it into the specified menu. The following options are available:

  -id *id*
  : Specifies the id of the new button.

  -index *index*
  : Specifies the position of the new button. If *index* is -1 (which is the default) the new button will be appended at the bottom of the menu.

-text *text*
Specifies the text to be displayed by the new button.

-fg *"r g b"*
Specifies the text or foreground color of the new button.

-bg *"r g b"*
Specifies the background color of the new button.

-proc *proc*
The name of the Tcl procedure to be called when the button is pressed. Either an own procedure without arguments can be used for a button. Alternatively, a procedure with one argument can be used for all buttons in a menu. The argument then is set to the id of the pressed button (see example above).

menu [...] insertMenu [options]
This command creates a new sub-menu button and inserts it into the specified menu. The id of the new button can be used as a sub-menu id for the *menu* command itself. The same options as for *insertItem* can be used. In addition the following option is available:

-type *classtype*
If this option is specified a special sub-menu will be inserted which lists all objects of type *classtype* in the Amira Pool. For example, in order to get a list of all modules (like in the default menu) you should use insertMenu with *-type HxModule*.

menu [...] changeItem id -text *text*
Changes the text of an existing button.

menu [...] removeItem id
Removes an action button or sub-menu button from the menu.

menu [...] connectItem id *proc*
Connects a button to a Tcl procedure. The procedure will be called when the button is pressed.

menu [...] disconnectItem id *proc*
Disconnects a button from a Tcl procedure.

Medivis Exhibit 1023

8.7.3 3D Module Controls

The 3D menu provides a button called *Modules*. Clicking on this button brings up a list of all visible modules which currently exist in the Amira Pool. Clicking on a module button brings up a 3D version of the modules user interface. This 3D user interface looks very similar to the 2D user interface of the module which is normally shown in the Properties Area of the Amira main window. An example of the 3D user interface of the *SurfaceView* module is shown in Figure 8.6.
The 3D user interface of every module can be moved independently from each other like the main 3D menu itself. Again, this is done by picking the header bar of the module's GUI. The GUI can be removed by clicking on the red *close button* at the right side of the header bar. On the left side an orange *viewer toggle* is displayed. As in the 2D interface a display module can be switched on or off by toggling this button.
In 3D the ports of a module can be manipulated almost in the same way as in 2D. The only remarkable difference is that text input is not possible in 3D. Text fields with numerical values like the range of a colormap port or the data window of an OrthoSlice module still can be changed using a virtual slider. This is done by clicking into the text field with the wand, and then moving the wand upwards or downwards while keeping the button pressed. As a general rule, every active element shows a yellow frame when the wand is pointing on it, i.e., when it has input focus.

8.7.4 Navigation Modes

Virtual Reality Option supports two different navigation modes. The default mode is *scene manipulation*, i.e., the whole scene can be rotated and translated using the 3D input device. This is done by clicking with the wand into some empty or insensitive region in space, and then moving the wand while keeping the button pressed. In this mode the whole scene remains stick relative to the input device.
The second mode is *fly mode*. This mode can be activated by selecting the second entry from the *Mode* port of the AmiraVR control module (either in the 2D or in the 3D user interface). In fly mode you can click the 3D input device at some insensitive point, and then move it in any direction. The vector from the point where the wand was clicked to the current point defines a velocity vector which is used to modify the position of the camera. The longer the vector, i.e., the more the wand is shifted, the faster the camera is moving. In the same way the orientation of the camera is modified by rotating the 3D wand. An incremental rotational change

Medivis Exhibit 1023

is computed by comparing the current orientation which the orientation at the point where the wand was clicked initially.

8.7.5 Tcl Event Procedures

By default, Virtual Reality Option treats all buttons of a 3D mouse in the same way. This allows you to operate the program even with a one-button mouse. In order to make better use of a 3D input device with multiple buttons, you can define special Tcl procedures which are called when a button is pressed or released. If these procedures return the value 1, this indicates that the event has been handled by the Tcl procedure and that it should not be further processed by Amira. In particular, it then will not be send to the Open Inventor scene graph. The following procedures can be defined:

- `vrButton0Press, vrButton1Press, ...`
  These procedures are called when the button with the specified index is pressed. By redefining `vrButton0Press` you can overwrite the default interaction routine.
- `vrButton0Release, vrButton1Release, ...`
  These procedures are called when the button with the specified index is released.
- `vrButton0DblClick, vrButton1DblClick, ...`
  These procedures are called when the button with the specified index is double-clicked. Note that for the first click of a double-click event an ordinary button press event is generated.

When a 3D mouse button event occurs and no appropriate Tcl event procedure is defined or if this procedure returns 0, the event will be passed to the current Virtual Reality Option event handler. The default event handler checks if the 3D menu or some other 3D widget has been clicked. If not, it sends the event to the Open Inventor scene graph. Besides this default event handler there are also other event handlers, namely handlers for managing the different navigation modes (*examine* and *fly*).
If a 3D mouse button event was not handled by the current event handler, finally the following Tcl procedures will be called, provided they are defined:

```
vrButton0PressFallback, vrButton1PressFallback, ...
vrButton0ReleaseFallback, vrButton1ReleaseFallback, ...
vrButton0DblClickFallback, vrButton1DblClickFallback, ...
```


Medivis Exhibit 1023

Assuming that the default event handler is active (the one which checks the 3D menu and the Open Inventor scene graph), you can trigger certain actions when the user clicks into empty space. By default, when this happens one of the navigation mode handlers is activated. If you want to disable this feature you can define a dummy `vrButton0PressFallback` procedure which always returns 1.

8.7.6 The 2D Mouse mode

The 2D mouse mode is useful for accessing GUI elements which have no direct analogy in 3D. In this mode the position of the ordinary 2D mouse pointer is controlled using the 3D input device.

- In order to activate 2D mouse mode press the red *mouse mode* button of the 3D menu. Once mouse mode is activated the Amira main window is popped up automatically.
- You can control the 2D mouse by translating or rotating the 3D input device. Usually, rotating it is more convenient since less movement is required. The position of the 2D mouse is computed by determining the intersection of the virtual wand vector with the screen.
- In order to exit 2D mouse mode, double-click the 3D input device over some insensitive region of the 2D user interface. When leaving 2D mouse mode the main graphics window is raised automatically.

  If the 2D mouse stays in the middle of the screen, of course 3D impression is disturbed. Therefore, usually it is a good idea to move the 2D mouse in a corner before returning to 3D mode.

The two Tcl procedures *VRSettings_StartMouseMode* and *VRSettings_StopMouseMode* are called when entering and exiting 2D mouse mode. You can use these methods for example to pop up the Amira help browser with a page from which additional demos can be launched. For this, the following definition could be included for example in the file *$AMIRA_ROOT/share/resources/Amira/Amira.init*:

```
proc VRSettings\_StartMouseMode { } {
  global AMIRA_ROOT
  load $AMIRA_ROOT/share/usersguide/tracking.html
}
```

In order to activate or deactivate the 2D mouse mode from Tcl, the Virtual Reality Option control module provides the command `VRSettings`

Medivis Exhibit 1023

`enableMouseMode <value>`, where `<value>` can be either 0 or 1. In order to check if 2D mouse mode is currently active, you can use `VRSettings isMouseModeEnabled`.

8.8 Writing Virtual Reality Option Custom Modules

Amira can be easily extended by writing new I/O routines, data types, modules, and other components. Details about this are described in the Developer Option User's Guide.
In order to write custom modules which provide specific interaction features in a VR environment, Open Inventor nodes interpreting events generated by the 3D input device have to be inserted into the scene graph. In particular, the 3D input device generates events of type *SoTrackerEvent* and *SoControllerButtonEvent*. These two event classes have been introduced in Open Inventor 3.0. For more information about these classes please refer to the Open Inventor documentation. Amira itself provides an additional class *Hx3DWandBase* which provides some additional information about the virtual 3D wand. Among others, this class also allows to user to highlight the wand in order to provide some visual feedback during interaction.
Below we present the source code of a sample AmiraVR module which just displays a number of cubes. The cubes then can be picked and transformed using the 3D wand. The source code of the module is contained in the Developer Option demo package. The particular files are called *MyVRDemo.h* and *MyVRDemo.cpp*. The module can also be directly created from the popup menu of the AmiraVR control module.
Here is the listing of the header file:

```
/////////////////////////////////////////////////////////////////////
//
// Illustrates 3D interaction in a VR environment
//
/////////////////////////////////////////////////////////////////////
#ifndef MY_VR_DEMO_H
#define MY_VR_DEMO_H

#include <Inventor/nodes/SoSeparator.h>

#include <McHandle.h>
#include <Amira/HxModule.h>
#include <Amira/HxPortButtonList.h>
#include <mypackage/mypackageAPI.h>

class MYPACKAGE_API MyVRDemo : public HxModule
```


Medivis Exhibit 1023

```
{
    HX_HEADER(MyVRDemo);

  public:
    MyVRDemo();
    virtual void compute();
    HxPortButtonList portAction;

  protected:
    ~MyVRDemo();

    McHandle<SoSeparator> scene;
    McHandle<SoSeparator> activeCube;

    bool isMoving;
    SbVec3f refPos;
    SbMatrix refMatrix;
    SbRotation refRotInverse;

    void createScene(SoSeparator* scene);
    SoSeparator* checkCube(const SbVec3f& pos);
    void trackerEvent(SoEventCallback* node);
    void controllerEvent(SoEventCallback* node);

    static void trackerEventCB(void* userData, SoEventCallback* node);
    static void controllerEventCB(void* userData, SoEventCallback* node);
};

#endif
```

Here is the listing of the source file:

```
//////////////////////////////////////////////////////////////////
//
// Illustrates 3D interaction in a VR environment
//
//////////////////////////////////////////////////////////////////
#include <stdlib.h>

#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/nodes/SoMatrixTransform.h>
#include <Inventor/events/SoTrackerEvent.h>
#include <Inventor/events/SoControllerButtonEvent.h>

#include <Amira/Hx3DWandBase.h>
#include <mypackage/MyVRDemo.h>

HX_INIT_CLASS(MyVRDemo,HxModule)

MyVRDemo::MyVRDemo() :
    HxModule(HxObject::getClassTypeId()),
    portAction(this,"action",1)
{
    isMoving = 0;
    portAction.setLabel(0,"Reset");
    scene = new SoSeparator;
    createScene(scene);
    showGeom(scene);
}

MyVRDemo::~MyVRDemo()
{
```

```
    hideGeom(scene);
}

void MyVRDemo::compute()
{
    if (portAction.isNew() && portAction.getIndex()==0)
        createScene(scene);
}

void MyVRDemo::createScene(SoSeparator* scene)
{
    scene->removeAllChildren();

    SoEventCallback* cb = new SoEventCallback;
    cb->addEventCallback(SoTrackerEvent::getClassTypeId(),
        trackerEventCB, this);
    cb->addEventCallback(SoControllerButtonEvent::getClassTypeId(),
        controllerEventCB, this);
    scene->addChild(cb);

    for (int j=0; j<4; j++) {
        for (int i=0; i<4; i++) {
            SoSeparator* child = new SoSeparator;

            SoMatrixTransform* xform = new SoMatrixTransform;
            SbMatrix M;
            M.setTranslate(SbVec3f(i*3.f,j*3.f,0));
            xform->matrix.setValue(M);

            SoMaterial* mat = new SoMaterial;
            float hue = (rand()%1000)/1000.;
            mat->diffuseColor.setHSVValue(hue,1.f,.8f);

            SoCube* cube = new SoCube;

            child->addChild(xform);
            child->addChild(mat);
            child->addChild(cube);

            scene->addChild(child);
        }
    }
}

SoSeparator* MyVRDemo::checkCube(const SbVec3f& p)
{
    for (int iChild=1; iChild<scene->getNumChildren(); iChild++) {
        SoSeparator* child = (SoSeparator*) scene->getChild(iChild);
        SoMatrixTransform* xform = (SoMatrixTransform*) child->getChild(0);
        const SbMatrix& M = xform->matrix.getValue();

        SbVec3f q;
        M.inverse().multVecMatrix(p,q);
        if (fabs(q[0])<1 && fabs(q[1])<1 && fabs(q[2])<1) {
            if (activeCube.ptr()!=child) {
                if (activeCube) {
                    SoMaterial* mat = (SoMaterial*) activeCube->getChild(1);
                    mat->emissiveColor = SbColor(0.f,0.f,0.f);
                }
                SoMaterial* mat = (SoMaterial*) child->getChild(1);
                mat->emissiveColor = mat->diffuseColor[0];
                activeCube = child;
            }
            return activeCube;
        }
    }

    if (activeCube) {
        SoMaterial* mat = (SoMaterial*) activeCube->getChild(1);
        mat->emissiveColor = SbColor(0.f,0.f,0.f);
        activeCube = 0;
    }

    return 0;
}
```


Medivis Exhibit 1023

```
void MyVRDemo::trackerEventCB(void* userData, SoEventCallback* node)
{
    MyVRDemo* me = (MyVRDemo*) userData;
    me->trackerEvent(node);
}

void MyVRDemo::trackerEvent(SoEventCallback* node)
{
    SoTrackerEvent* e = (SoTrackerEvent*) node->getEvent();
    Hx3DWandBase* wand = GET_WAND(e);

    if (activeCube && isMoving) {
        if (!wand->getButton(0)) {
            node->setHandled();
            isMoving = 0;
            return;
        }

        SbMatrix T1; T1.setTranslate(-refPos);
        SbMatrix R; R.setRotate(refRotInverse*wand->orientation());
        SbMatrix T2; T2.setTranslate(wand->origin());

        SoMatrixTransform* xform = (SoMatrixTransform*) activeCube->getChild(0);
        xform->matrix = SbMatrix(refMatrix*T1*R*T2);

        wand->setHighlight(true);
        node->setHandled();
        return;
    }

    if (checkCube(wand->top()))
        node->setHandled();
}

void MyVRDemo::controllerEventCB(void* userData, SoEventCallback* node)
{
    MyVRDemo* me = (MyVRDemo*) userData;
    me->controllerEvent(node);
}

void MyVRDemo::controllerEvent(SoEventCallback* node)
{
    if (activeCube) {
        SoTrackerEvent* e = (SoTrackerEvent*) node->getEvent();
        Hx3DWandBase* wand = GET_WAND(e);

        if (wand->wasButtonPressed(0)) {
            SoMatrixTransform* xform = (SoMatrixTransform*) activeCube->getChild(0);
            refRotInverse = wand->orientation().inverse();
            refPos = wand->origin();
            refMatrix = xform->matrix.getValue();

            wand->setHighlight(true);
            isMoving = 1;
        }

        if (wand->wasButtonReleased(0))
            isMoving = 0;

        node->setHandled();
    }
}
```

8.9 Config File Reference

An Virtual Reality Option config file is an Open Inventor file with a .cfg extension containing one or more SoScreen *nodes* (one for each screen of the VR system),

Medivis Exhibit 1023

optional nodes SoTracker (containing information about the tracking system) and SoVRProperty (containing general information), as well as optional Open Inventor nodes (containing possibly a visual representation of the room).
In principle two different configurations are distinguished. A "flat screen" configuration defines a virtual big 2D screen, while a true "immersive" configuration defines multiple screens in a non-planar configuration. For a flat screen configuration, the fields *tileOrigin* and *tileSize* (see below) are used, while for an immersive configuration the fields *lowerLeft*, *lowerRight*, *upperRight*, and *upperLeft* are used. For a tiled configuration, ordinary 2D mouse interaction works, while an immersive configuration usually requires a tracking system.
By default, the list of available config files (*.cfg) is generated by looking in the `AMIRA_ROOT`/share/config/vr and `AMIRA_LOCAL`/share/config/vr directories. The user can define an `AMIRA_VR_CONFIGS_PATH` environment variable to use a different location at start-up. Then, at runtime, the list can be further changed or updated by using the *Change Configs Path...* item in the VR menu.

SoScreen

An SoScreen node can contain the fields detailed below. The field content type is given inside brackets using Open Inventor conventions, ie. SoSFVec3f stands for 'a vector of 3 float values', SoSFRotation is a rotation defined with 4 values represents an axis of rotation followed by the amount of right-handled rotation about that axis, in radians. For example, a 180 degree rotation about the Y axis is : 0 1 0 3.12159265

SoSFString name "Vertical screen"
The name of the screen (not used internally)

SoSFVec3f lowerLeft 0 0 0
The coordinates of the lower left corner of the screen. Any right-handed coordinate system can be used. It does not matter in which units the screen corners are defined because the transformation between the screen coordinates and the tracker coordinates is computed automatically when calibrating the tracking system.

SoSFVec3f lowerRight 180 0 0
The coordinates of the lower right corner of the screen.

Medivis Exhibit 1023

SoSFVec3f upperRight 180 110 0
The coordinates of the upper right corner of the screen.

SoSFVec3f upperLeft 0 110 0
The coordinates of the upper left corner of the screen.

SoSFString display ":0.1"
Specifies on which X display the screen should be rendered. On a multi-pipe machine such as an SGI Onyx, either different X servers may run on the different pipes, or there may be one X server with multiple screens. Depending on the actual configuration either ":1.0" or ":0.1" should be specified in order to enable multi-pipe rendering for the second screen. If this field is omitted, the window will be opened on the same display as the Amira user interface (determined by the value of the DISPLAY environment variable or by the -display command line option).

SoSFString hostname
This field is required if Virtual Reality Option will be run in cluster mode. It specifies on which node of a graphics cluster the particular screen should be rendered. The hostname can either be specified as a name or as a numeric IP address. See section 8.3 (Using Virtual Reality Option on a cluster).

SoSFVec2f channelOrigin 0.0 0.0
Specifies the position of the upper left corner of the graphics window in relative coordinates. (0,0) is the upper left corner of the desktop, (1,1) is the lower right corner of the desktop.

SoSFVec2f channelSize 0.5 1.0
Specifies the size of graphics window in relative coordinates. The size of the whole desktop is (1,1). In order to create a window covering the left half of the desktop, *channelOrigin* should be (0,0) and *channelSize* should be (0.5,1).

SoSFVec2f tileOrigin 0.0 0.0
This field is used in the case of a 2D tiled display instead of the fields *lowerLeft*, *lowerRight*, *upperRight*, and *upperLeft*. It specifies the origin of a rectangular part of a virtual big screen which should be rendered in the graphics window. The origin can be any point between (0,0) and (1,1). Here (0,0) denotes the LOWER left corner of the virtual big screen, and (1,1) denotes the UPPER right corner.

Medivis Exhibit 1023

SoSFVec2f tileSize 1.0 1.0

This field is used in the case of a 2D tiled display instead of the fields *lowerLeft*, *lowerRight*, *upperRight*, and *upperLeft*. It specifies the size of a rectangular part of a virtual big screen which should be displayed in the graphics window. The size can be any value between (0,0) and (1,1).

SoSFEnum cameraMode [MONOSCOPIC — LEFT_VIEW — RIGHT_VIEW — ACTIVE_STEREO]

This field specifies the camera mode used for the screen. The default is MONOSCOPIC. The values LEFT_VIEW and RIGHT_VIEW are used for passive stereo applications. If ACTIVE_STEREO is specified, the window is opened in active stereo mode by default.

SoSFEnum backgroundMode [BG_AS_IS — BG_LOWER — BG_UPPER — BG_REVERSE]

This field affects how the default Amira gradient background is rendered on the screen. If BG_LOWER or BG_UPPER is specified, a uniform background with either the lower or upper color of the standard gradient background is used. This is useful for the floor or ceiling of a CAVE.

SoSFInt32 threadGroup

Screens with different thread groups are rendered in parallel using multiple threads, provided Amira has been started with the -mt command line option or the environment variable `AMIRA_MULTITHREAD` has been set. If the same thread group is used for two different screens, these screens will always be rendered one after the other. Note that for all common current graphics architectures it makes no sense to render multiple windows on the same pipe in parallel. Typically, this even implies a significant performance decrease. Usually, screens being rendered on the same pipe (same display but different *channelOrigin* or *channelSize*) should be assigned the same thread group. Thread group and -mt command line option are only relevant on multipipe systems and do not apply to cluster systems.

SoMFFloat softEdgeOverlap

This field is used for soft-edge blending. It should contain exactly four floating-point values, indicating the size of the soft-edge region on the left, right, bottom, and top border of the screen relative to the total width or height of the screen. For example, in order to specify a 25% soft-edge at the right border, *softEdgeOverlap* should be [0.0, 0.25, 0.0, 0.0].

Medivis Exhibit 1023

SoMFFloat softEdgeGamma
This field is used for soft-edge gamma correction. It should contain exactly four floating-point values, indicating the gamma value of the soft-edge region at the left, right, bottom, and top border of the screen. A gamma value of 1 means, that image intensity is linearly faded out to black. For most projectors, gamma values bigger than 1 are required to achieve good results. The default gamma value for all borders is 1.2.

SoTracker

An SoTracker node contains information about the tracking system. Usually it is not necessary to define an SoTracker node for a "tiled" configuration, but only for true VR configurations which make use of the fields *lowerLeft*, *lowerRight*, *upperRight*, and *upperLeft* of SoScreen (see above).

SoSFString server 4127:4126
This field specifies a string used to initialize the connection to the tracking system. In order to connect to a running VRCO trackd daemon, the string should contain the shared memory key of the trackd controller data, and the shared memory key of the trackd tracker data separated by a colon ("¡controllerKey¿:¡trackerKey¿"). The shared memory keys are defined in the trackd.conf file.

SoSFBool autoConnect
If this field is set to TRUE, the connection to the tracking system will be established automatically when the configuration is loaded. The default value is FALSE.

SoSFVec3f defaultCameraPosition 90 50 150
Defines the camera position to be used as long as there is no connection to the tracking system. This position is also used if the head tracker id (below) is -1.

SoSFVec3f defaultObjectPosition 90 50 0
Defines the object position to be used as long as there is no connection to the tracking system. If there is a "view all" request in Amira (e.g., because the space bar of the keyboard was hit), the scene is centered at the default object position. In a CAVE it makes sense to choose the default object position in the center of the front wall.

Medivis Exhibit 1023

SoMFVec3f calibration
The field is used to store calibration data, i.e., data allowing the system to transform raw tracker data into the coordinate system in which the screens have been defined. Usually there is no need to set this field manually. Instead, load a configuration file without calibration data, perform calibration in Virtual Reality Option, and then export a new config file.

SoSFRotation rotGlasses
This field is used for calibrating the orientation of the tracked stereo glasses. Usually there is no need to set this field manually.

SoSFRotation rotWand
This field is used for calibrating the orientation of the wand tracker. Usually there is no need to set this field manually.

SoSFVec3f leftEyeOffset
The center of the left eye with respect to the head tracker. For calibration, the glasses must be align parallel to the front screen. In this position the horizontal axis is the x-axis, the vertical axis is the y-axis, and the axis pointing towards the observer is the z-axis. The eye offsets are defined with respect to this coordinate system. The difference between the left eye's x-offset and the right eye's x-offset should be around 6.5 cm (average eye separation).

SoSFVec3f rightEyeOffset
The center of the right eye with respect to the head tracker. The same remarks as for *leftEyeOffset* apply.

SoMFVec3f referencePoints
Reference points used for calibrating the tracking system. If no reference points are specified, the user is asked to click at the upper left, lower left, and lower right corner of the first screen defined in the config file. Note that at least three different reference points are required and that these points must not lie on a common line. However, the reference points may well lie in a common plane, e.g., in the screen plane itself.

SoSFInt32 wandTrackerId
Specifies the id of the trackd sensor which should be used to control the wand. By default the wand sensor is assumed to have the id 0.

SoSFInt32 headTrackerId
Specifies the id of the trackd sensor which should be used to control the

Medivis Exhibit 1023

camera (head tracking). By default the head tracker is assumed to have the id 1. If -1 is specified for a head tracker id, the default camera position is used instead of actually querying a head sensor.

SoSFInt32 headTrackingEnabled
This field determines if head tracking should be enabled or disabled by default if the configuration file is loaded. The default is TRUE. If the value is set to FALSE, head tracking is initially disabled, but can be activated later by pressing the head tracking toggle of the AmiraVR control module. However, *headTrackerId* must not be set to -1 in this case.

SoSFString wandFile
This is the name of an Open Inventor file which is used as the visual representation of the wand. The origin of the wand should be at (0,0,0). The wand itself should point into the negative z direction. The wand is scaled so that the length 1 corresponds to 0.16 times the width of the first screen in the config file. The hot spot of the wand should be indicated by an SoInfo node containing a string *"x y z"*. Usually the hot spot will be something like *"0 0 -1"*. The filename can be an absolute or relative path. If it is relative, the file is assumed to be in the same directory as the config file itself (namely in *share/config/vr*).

SoSFString highlightWandFile
This is the name of an Open Inventor file which is used as the visual representation of the wand in highlight state. The same remarks as for *wandFile* apply to this field too.

SoSFFloat wandScale
This field specifies a scaling factor applied to the wand geometry. By default, a value of 0.16 times the width of the first screen is assumed. In order to change the length of the wand you can either specify a custom wand file with a hot spot different from *"0 0 -1"*, or you can change the wand scale value.

SoSFVec3f menuPosition
Specifies the default position of the upper left corner of the 3D menu. By default it is placed in the upper left corner of the first screen in the config file. The values must be defined in the same coordinate system as the corners of the screens.

Medivis Exhibit 1023

SoSFRotation menuOrientation
Specifies the orientation of the 3D menu. By default the horizontal axis of the menu, i.e., the text direction, is aligned to the x-axis of the VR coordinate system, while the vertical direction is aligned to the y-axis. This field allows you to change the orientation by applying a rotation to the default orientation. The rotation is specified by four numbers. The first three numbers define the axis of rotation, and the fourth numbers defines the rotation angle in radians.

SoSFFloat menuSize
This field specifies the default width of the 3D menu in the same coordinates in which the corners of the screens were defined. You can adjust this value to make the menu smaller or bigger.

SoSFString onLoad
This field defines a Tcl command which is executed after the configuration has been loaded. You can put additional initialization code here. For example, you may always want to load a particular script *MenuInit.hx* for initializing the *3D menu* when a certain configuration is loaded. You can do this by defining *onLoad "load $*`AMIRA_ROOT`*/MenuInit.hx"*.
As another example you can show 3D menu with:
onLoad "AmiraVR options setValue 2 1"

SoSFString onUnload
This field defines a Tcl command which is executed before a configuration is unloaded.

SoSFString onConnect
This field defines a Tcl command which is executed after a connection to the tracking system has been established.

SoSFString onDisconnect
This field defines a Tcl command which is executed after the connection to the tracking system has been disconnected.

SoVRProperty

An SoVRProperty node contains general information about the Amira user interface.

Medivis Exhibit 1023

SoSFBool showSlaveGUI FALSE
Set TRUE to force the Amira GUI to be displayed on the slaves in cluster mode. Note that this interface is not synchronized from slaves, which means that anything done on one slave will not be forwarded to the others.

SoSFBool showSlaveConsole FALSE
Set TRUE to force the Amira Console to be displayed on the slaves in cluster mode. Note that this interface is not synchronized from slaves, which means that anything done on one slave will not be forwarded to the others.

SoSFBool showSlaveCursor FALSE
Set TRUE to display the mouse cursor on the slaves' render areas in cluster mode. However, this is only a "forbidden" type cursor that cannot interact with the Amira render area.

SoSFBool keepMasterViewerInsideGUI FALSE
In a cluster configuration, a screen must be defined on master node for correct synchronization, but this may disturb the user to have its main viewer in a floating window. Set the field keepMasterViewerInsideGUI to TRUE to force the main viewer to be kept inside the standard GUI. This flag only has effect on the primary screen of a VR node.

Configuration tips

Here is a summary of some configuration settings that may be helpful to polish your configuration:

```
#Inventor V2.1 ascii
SoVRProperty {
      keepMasterViewerInsideGUI TRUE  # If you want to use a standard
                                      # viewer on master
}
SoScreen {
      ...                             # Display configuration
      backgroundMode  ...             # For consistent backround across screens
      softEdgeOverlap ...             # Soft-edge region
      softEdgeGamma ...               # Soft-edge blending rate
}
SoTracker {
      ...
      autoConnect TRUE                # Note that tracking systems must be
                                      # running
      headTrackingEnabled FALSE       # Starts without head tracking (can
                                      # be activated later)
      wandFile "myWandFile"           # Open Inventor/VRML file for custom
                                      # wand appearance
```

```
      highlightWandFile "myHWandFile" # Highlighted version
      menuPosition 100 100 0          #
      menuSize 50                     #
      onLoad "..."                    # Custom TCL commands
      onConnect "..."
      onDisconnect "..."
}
Separator {                           # Separator containing "environment
                                      # scene"
      PointLight {location 0 0 0}     # Useful e.g. in CAVEs
      Separator {...}                 # Fixed geometry relative to screens
}
```

8.9.1 Tracker Emulator

The *Tracker Emulator* provides emulation of a 3D positioning device with two spatial sensors and two buttons. To activate it, type the word *test* into the tracker port of the *AmiraVR control module* and press the *connect* button. After that the user interface dialog of the tracker emulator appears.

As mentioned before, the emulator emulates two spatial sensors designated as *sensor0* and *sensor1*. For each of these sensors the position is changeable by three sliders, one for each direction. Alternatively, as well as to reach positions of greater distance than the sliders allow, one can type the values directly into the text entry fields to the right of the slider.

The sensor's rotation can be changed by selecting one of the main axes as the rotation axis and then further adjusting the rotation angle around this axis using the slider or the entry field. To perform an additional rotation, simply select a different rotation axis and angle. The rotations are performed cumulatively. The current rotation doesn't affect the orientation of the main rotation axes. The rotation center is always the current sensor position. To set the sensor's rotation to the initial unrotated state, press the *Reset* button.

In the buttons section one can toggle the emulated button's state. As long as the checkbox is checked, the emulated button is "down". If it's unchecked, the emulated button is "up".

Medivis Exhibit 1023

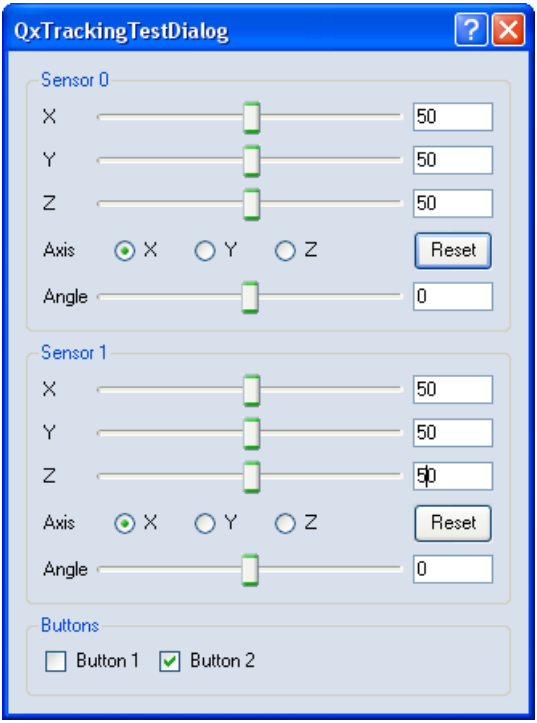


Figure 8.7: The tracker emulators user interface dialog.

Medivis Exhibit 1023

Part IV

Very Large Data Option User's Guide

Medivis Exhibit 1023

9 Very Large Data Option User's Guide

9.1 Working with out-of-core data files (LDA)

The out-of-core management tools allow you load and visualize data sets larger than the amount of RAM installed on your system, as well as convert these data sets into LDA (Large Data Access) files. These LDA files can be used to visualize very large data (hundreds of gigabytes), such as seismic or microscopy data, using a limited amount of memory. It is possible to convert original data of the following types: AmiraMesh, RawData, and StackedSlices (stacks of SGI, TIFF, GIF, JPEG, BMP, PNG, JPEG2000, PGX, PNM, and RAS raster files). LDA data allows subvolume extraction to display parts of the volume, or multi-resolution access to have a full subsampled view or accurate refined local views.
In particular, the following topics will be discussed in this tutorial:

1. Adjusting the size threshold to allow conversion
2. Loading the out-of-core data
3. Raw data parameters
4. Out-of-core conversion
5. Displaying an ortho slice, an oblique slice, and a 3D volume

Please follow the instructions below. Each step builds on the step before.

9.1.1 Tune the Size Threshold to Allow Conversion

In the *Edit* menu, select the *Preferences* item. This opens the Amira*Preferences* dialog. Please select the *LDA* tab (see Figure 9.1). Using the slider or text field, set the threshold to 32MB. When you load a file of file size greater than this threshold, the out-of-core data dialog will be displayed.
Note: To see the images as laid out in this tutorial, you should also use the *Layout* tab of the *Edit/Preferences* menu, and toggle on *show viewer in top-level window*.

Medivis Exhibit 1023

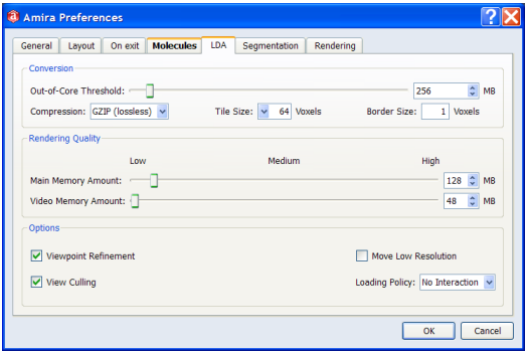


Figure 9.1: Amira Preferences, LDA settings

9.1.2 Load the Out-of-core Data

Please open the file 3DHEAD.raw using the *File/Open Data...* menu (see Figure 9.2). The file is located in data/tutorials/outofcore in the Amira install directory. Its size is slightly larger than 32MB, right above the defined threshold.
The Out-Of-Core data dialog is displayed. Three loading options are displayed (see Figure 9.3):

- *Convert to LDA*: convert the file to an LDA file, and then load it.
    - *PRO*: This builds a multiresolution file allowing full interactive view or local full resolution viewing.
    - *CON*: This can be time consuming, with an initial pass and then the true conversion pass.
- *Read from disk*: read data blocks from disk, allowing almost continuous disk access.
    - *PRO*: No need to generate an extra file.
    - *CON*: Continuous access to disk. Slow with data sets larger than 4GB.
- *Read in memory*: load full data into memory and then access to memory only.
    - *PRO*: Adapted for average sized data.

Medivis Exhibit 1023

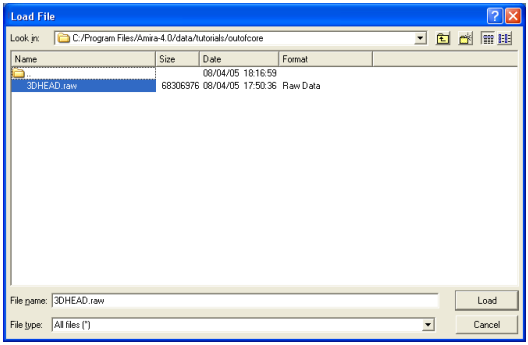


Figure 9.2: Open the out-of-core data file

- *CON*: Requires as much RAM as your data set size. Usually not applicable for data sets greater than 500MB or 1GB.

Please select ”Convert to LDA”. Then, on the next dialog (Destination file), specify the LDA destination file. 3DHEAD.lda for instance (see Figure 9.4).
Note: An .lda file can be loaded then, without any conversion required.
Another option allows you to perform conversion in batch mode so you can run other processes while the conversion is done in the background.

9.1.3 Raw Data Parameters

As the input data is raw, please fill in the raw data parameters dialog with information as on the following figure (see Figure 9.5):
Data type is byte, dimension 431*431*184.

9.1.4 Out-of-core Conversion

During conversion, the out-of-core conversion progress dialog is displayed (see Figure 9.6). This process is done in two steps. First of all, an initial step, and then the conversion step at about 4MB/s (on a P4 2.6GHz, no SATA disk). You can cancel the process if you wish.
The converted file is now in the Pool ready to be used and connected to other modules.

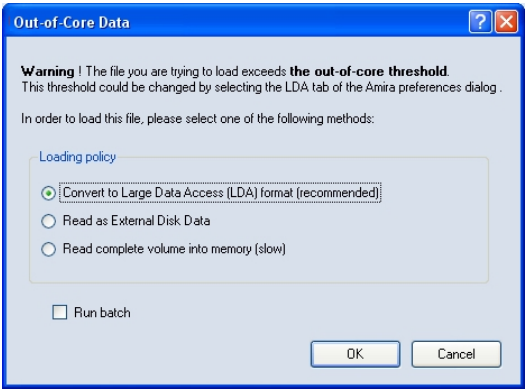


Figure 9.3: Out-of-Core data dialog

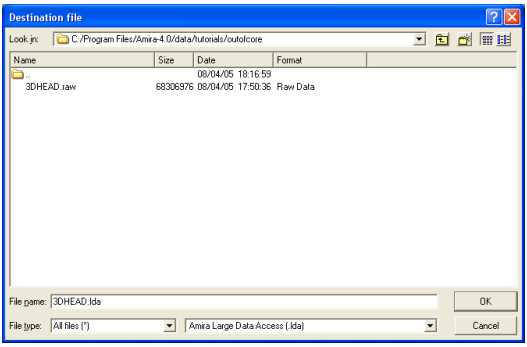


Figure 9.4: Choose LDA destination file

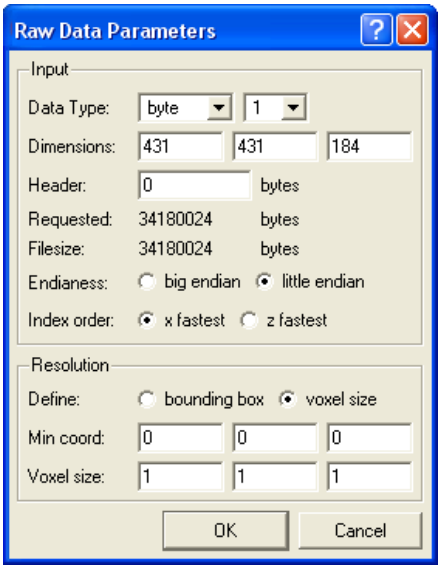


Figure 9.5: Raw data parameters panel

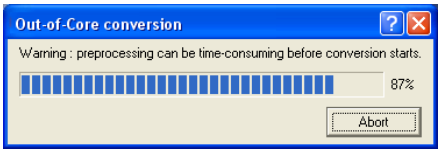


Figure 9.6: Out-of-core conversion progress dialog

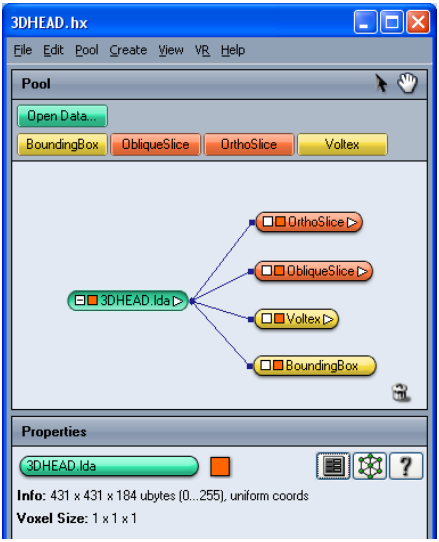


Figure 9.7: Display modules added

9.1.5 Display an Orthoslice, an Oblique Slice, and a 3D Volume

Right-click on the data icon in the Pool. In the Display submenu choose the OrthoSlice module. Repeat these steps for an ObliqueSlice and a Voltex (3D volume). You can also display the bounding box of the full volume.
In order to view this scene with the same settings, after converting 3DHEAD.raw into 3DHEAD.lda (lda file required, with the right name) please load the network 3DHEAD.hx (in the same directory data/tutorials/outofcore).

Medivis Exhibit 1023

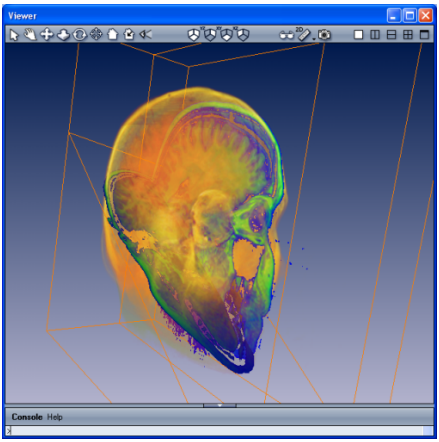


Figure 9.8: Network display

Part V

Quantification+ Option User's Guide

Medivis Exhibit 1023

This extension integrates the Visilog product by Noesis into Amira. The Quantification+ Option, with its wide range of image processing tools, can be used together with the 3D visualization and geometry reconstruction capabilities of Amira.
A set of demos is provided. For details, see the documentation of the *Visilog* module.

Medivis Exhibit 1023