Amira - a Highly Interactive System for Visual Data Analysis

Detlev Stalling, Malte Westerhoff, and Hans-Christian Hege
Zuse Institute Berlin (ZIB), Germany

1 Introduction

What characteristics should a good visualization system hold? What kinds of data should it support? What capabilities should it provide? Of course, the answers depend on the particular task and application. For some users a visualization system may be nothing more than a simple image viewer or plotting program. For others it is integrated software dedicated to their personal field of work, such as a computer algebra program or a finite-element simulation system. While in such integrated systems visualization is usually just an add-on, there are also many specialized systems whose primary focus is upon visualization itself.

On the one hand, there are many self-contained special-purpose programs written for particular applications. Examples include flow visualization systems, finite-element post-processors, and volume rendering software for medical images. On the other hand, several general-purpose visualization systems have been developed since scientific visualization became an independent field of research in the late 1980s. These systems are not targeted to a particular application area, but provide many different modules which can be combined in numerous ways, often adhering to the data-flow principle and providing means for 'visual programming' [23, 4, 17, 1, 5, 15, 7].

In these ways, custom pipelines can be built to solve specific visualization problems. Although these visualization environments are very flexible and powerful, they are usually more difficult to use than special-purpose software. In addition, a major drawback induced by the data-flow principle or pipelining approach is the lack of sophisticated user interaction. Any operation which requires manual interaction, such as segmenting a medical image into different regions, editing a polygonal surface model or molecular structures, or cropping and selecting different parts of a complex finite-element model, is difficult to incorporate into a pipeline of modules that is executed automatically. One may argue that these interactive tasks are not visualization problems per se. But it is a matter of fact that such operations require visual support and are essential for solving problems in many application areas.

In order to close the gap between the ease of use, power, and interactivity of monolithic special-purpose software, and the flexibility and extensibility of data-flow oriented visualization environments, the software system amira has been designed. Initially developed by the scientific visualization group at the Zuse Institute Berlin (ZIB), today amira is available as a commercial product together with several extensions and add-ons [2]. One major focus of the software is the visualization and analysis of volumetric data which is common in medicine, biology, and microscopy. These volumes can be displayed and segmented, 3D polygonal models can be reconstructed, and these models can be further processed and converted into tetrahedral volume

grids. Due to its flexible design, many other tasks can also be performed in amira, including finite-element post-processing, flow visualization, and visualization of molecules. In this chapter we discuss the fundamental concepts, techniques, and features of amira.

1.1 Design Goals

The development of amira was driven by the following design goals:

- *Ease-of-use.* Simple visualization tasks such as extracting an oblique slice from a 3D image or computing an isosurface should not require more than a few mouse clicks. Untrained users should be able to get results as quickly as possible.
- *Flexibility.* The system should be able to work with a large number of different data types and with multiple data sets at once. Complex operations requiring the combination of multiple modules should be possible, too.
- *Interactivity.* Techniques or components requiring heavy user interaction, both in 2D and in 3D, should be easy to integrate. Examples are image segmentation, surface editing, alignment operations, and many more.
- *Extensibility.* Users should be able to add new features to the system, e.g., new I/O routines, new modules, or even new data types or interactive editors. Existing components should be customizable to some extent.
- *Scripting interface.* The system should be programmable via a scripting language, enabling batch processing, in order to simplify user-specific routine tasks and presentations, and to facilitate customization of the software.
- *Multi-platform support.* Different hardware platforms and operating systems should be supported, in particular Windows, Linux, and other Unix variants. 64-bit code should be supported in order to process large data sets efficiently.
- *State-of-the-art algorithms.* Modern visualization techniques such as direct volume rendering and texture-based flow visualization should be implemented. All techniques should be optimized for both, performance and image quality.

In the following section we first describe the general concepts we have chosen to meet these goals. Next, we will illustrate how the system can be applied in different fields of work. We do this by identifying common tasks and show which methods are provided to solve these tasks. Finally, we are going to discuss some amira extensions, most prominently amiraVR, an extension which allows the software to operate upon large tiled displays and within immersive virtual reality environments.

2 General Concepts

In Fig. 1 a snapshot of the amira user interface is shown. When the software is started, three windows are invoked: the main window containing the “object pool” and the control area, a graphics window where visualization results will appear, and the console window where messages are printed and additional commands can be typed in. Data sets can be easily imported via the file browser or via drag-and-drop. After a data set has been imported, it is represented as a small icon in the object pool. The data set can be visualized by choosing an appropriate display module from a context-sensitive popup menu over the data icon. This popup menu lists only modules which can be connected to the particular data object. The output of a display module is immediately shown in the graphics window. Thus, a data set often can be visualized with a single

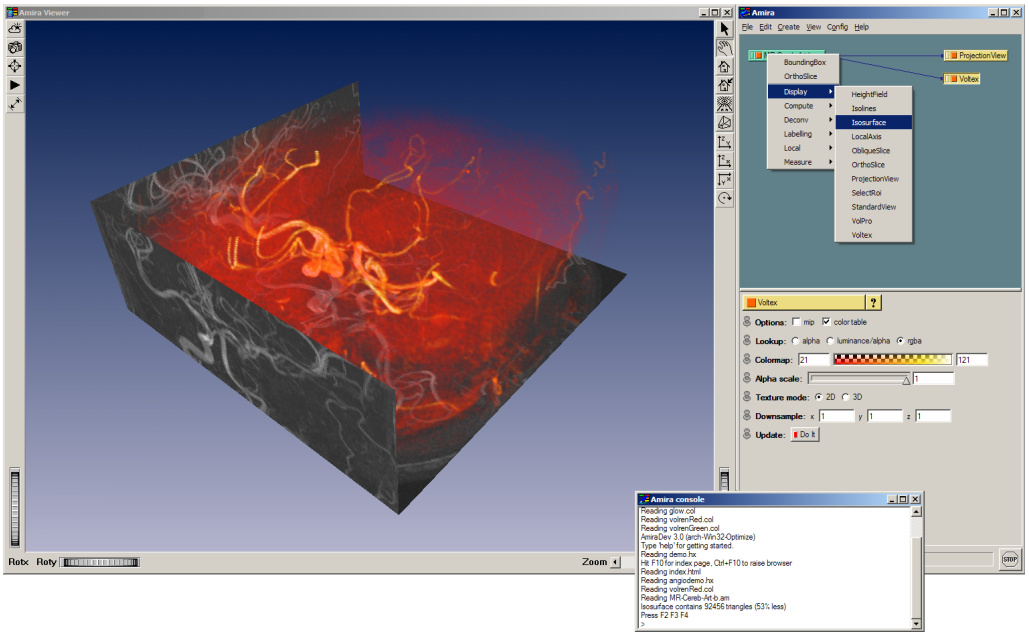

Figure 1: The amira user interface consists of the main window with the object pool and the control area, a large 3D graphics window, and a console window for messages and command input. The figure shows a medical image data set from MR angiography, visualized by a projection view modules and a volume rendering module.

mouse click once it has been imported. Also, different visualization techniques can be combined with each other without any limitations.

Having discussed the basic modes of operation, we now consider the concepts behind amira in more detail.

2.1 Object Orientation

In amira data sets as well as modules are considered to be objects. These objects are represented visually as icons in the object pool. Looking at the object pool, one can easily observe which objects exist and how they are related to each other. If an object is selected with the mouse, additional information and corresponding user interface elements are displayed in the work area. These interface elements, known as “ports” in amira, allow users to interact with a module. For example, the threshold of an isosurface module or the position of a slicing module can be adjusted with corresponding sliders. In addition, every object also provides a script interface. One can query an object for its properties, or one can interact with an object by calling it with certain commands.

A major advantage of the object concept is that objects can be easily modified or edited by other components. For this purpose powerful editors are provided, allowing modification of data sets in a highly interactive way. Examples are the amira segmentation editor, the landmark editor, and the surface editor. Objects can be modified not only by editors, but also by other components, mainly compute modules. This is not easily possible in a data-flow oriented system, as in such systems data is usually not represented to users as an independent object.

2.2 Inheritance and Interfaces

The object oriented approach in amira also makes use of class inheritance. For example, all data objects are derived from a common data class. This class provides methods to duplicate and save the object or to associate arbitrary parameters with it. A more specialized data class represents spatial data objects, i.e., data objects which are embedded in 3D space. This class provides methods to query the bounding box of the data object, and to set or get an optional affine transformation matrix. Another base class represents fields, i.e., data objects which can be evaluated at any point in a 3D domain. This class provides methods to query the component range of the field, or to evaluate it at an arbitrary point in a transparent way, i.e., without needing to know how the field is actually represented. The field may be defined on a regular grid, on an unstructured grid, or even procedurally by specifying an arithmetic expression. New fields defined in other ways may also be easily added. All display modules defined for the base class can be used automatically for these new fields. In amira the generic evaluation methods of the field class are used (for example) by data probing modules, which plot field values at a point or along a line; by flow visualization modules, which need to compute trajectories in a vector field; or by slicing modules, which need to resample a field on a 2D grid.

However, in some cases it is not possible to derive data classes with common properties from a common base class. For example, in amira there are separate base classes for scalar fields and for vector fields. At the same time, scalar fields and vector fields defined on the same type of grid, e.g., on a regular cartesian grid, have many things in common. For this reason, amira provides a mechanism called interfaces. These are classes which describe common properties of objects which do not necessarily have a common base class. For example, both regular scalar fields and regular vector fields provide a lattice interface. In this class, the number of data values per node is a variable. Thus a module or export routine using the lattice class interface can be automatically applied to both regular scalar fields and regular vector fields, or to any other object providing this interface.

2.3 User Interface Issues and 3D Interaction

A primary design goal of amira is ease-of-use. Of course, this is a somewhat subjective and loosely-defined requirement. We try to accomplish this goal using several different strategies. First, context-sensitive popup menus are provided, offering only those modules which actually can be connected to a given data object. Next, fewer but more powerful modules are preferred compared to a larger number of simpler entities. For example, in order to display one slice of a 3D image, in a data-flow oriented visualization system, users must often first extract a 2D subimage, then convert this into geometry data, and finally display the geometry using a render module. In amira all of this is done by a single OrthoSlice module.

By default, visualization modules show their results directly in the main graphics window. In many cases one simply chooses a module from a data object's popup menu and immediately receives a visual result. Modules which may need more time for preprocessing usually provide an additional *DoIt* button which must be pressed in order to generate a result. In this way it is possible, for example, to first adjust the threshold of an isosurface module or to first select the colormap of a volume rendering module before starting any computation. Optionally, *DoIt* buttons can be "snapped" to an "on" position, thus facilitating automatic updates. Another point improving the clarity of the user interface is to reduce the number of open windows and avoid overlapping windows so far as possible. The controls of amira objects are shown in a single scrolled list once an object has been selected. Although multiple objects can be selected at once, usually this is not the case. Typically, the user interface is organized as a list of so-called ports, where each port comprises a single line with a label, followed by some buttons, text fields, or sliders. If required, important ports can be "pinned," which makes them visible even if the corresponding object has been deselected.

In addition to the standard controls, many modules also provide a means for 3D interaction in the graphics window. For example, a slice may be picked and translated in 3D. In order to choose a different orientation, a trackball icon can be activated, which then in turn can be picked and rotated. The positions of landmarks and other points can defined by simply clicking on other objects. Similarly, information about points or faces of a grid or surface, as well as associated data values, can be obtained by clicking on them. Parts of a 3D model or individual triangles can be selected using a lasso tool, i.e. by drawing a contour in the graphics window.

2.4 Software Technology

Flexibility and extensibility of amira is ensured by a specific modular software architecture where multiple related modules are organized into different packages. Each package exists in the form of a shared library (or DLL under Windows) which is linked to the main program at run-time. For each package there is a resource file specifying which data classes, modules, editors, or I/O routines are defined in that package. This way, only shared libraries providing code which is actually being used need to be loaded. This keeps the executable size small, but still makes it possible to have an almost unlimited number of different components in the system. In order to extend the functionality of the system, developers can add custom packages or even replace existing packages as needed.

amira is written in C++, and requires only a few external standard libraries. For 3D graphics support the TGS Open Inventor toolkit is used, which is a well established and proven scene-graph layer [13]. Open Inventor provides multi-platform support, multi-threaded rendering, and many advanced display nodes. In addition, several Open Inventor custom nodes have been added to amira for improved performance and image quality. All of these nodes have been directly implemented using OpenGL. The graphical user interface of amira is built using Qt, which is a multi-platform widget library [16]. With Qt it is possible to use the same code base for all supported platforms. Currently support platforms are Windows, Linux, IRIX, HP-UX, and Solaris, all with both 32-bit and 64-bit code. A Mac OS X version is planned for the near future. The ability to run in 64-bit mode is important because it allows to process very large data sets as they become more and more frequent in many areas of science and engineering.

2.5 Scripts and Script Objects

The script interface of amira is based on the scripting language *Tcl*, which is also an established industry standard [14]. The standard set of Tcl commands has been extended by many amira specific commands. In particular, the name of each object in the amira object pool can be used as a command name allowing to interact with that particular object. Furthermore, the name of each port of a data object or module can be used as an argument or subcommand for that object. All ports provide Tcl methods to set or get their respective values. For example, to set the threshold of an isosurface module in a script (i.e., the value of the port representing the threshold), one would use the command

```
Isosurface threshold setValue 100
```

.

Scripts allow one to simplify routine tasks or to run complex presentations. When an amira network is saved, a Tcl script is generated, which, when executed, restores the current state. Tcl code can also be bound to certain function keys or to entries in the context menu of a data object. This menu lists all modules which can be connected to an object. It is even possible to modify the default settings of any amira module using Tcl code. Finally, Tcl expressions can be used to decide at run-time whether a particular module can be connected to some data object or whether a particular export routine can be called for that object. In this way, the default rules for matching object types and interface names can be overwritten.

Besides standard scripts, amira also supports *script objects*. Script objects are similar to ordinary modules but are implemented completely in Tcl. Usually they provide at least three Tcl

procedures: a constructor, a compute routine, and a destructor. The constructor is called when the object is initialized. Any number of standard GUI components, i.e., ports, can be created and initialized here. The compute routine is invoked whenever one of the ports is changed. Finally, the destructor is called when the script object is deleted. Script objects are well suited to solve specific problems using high level commands. Often multiple standard modules are combined in a script object in order to generate a result.

2.6 Affine Transformations

In many applications, alignment or registration of multiple data sets is important. Therefore visualization environments should allow the user to easily transform individual data sets spatially with respect to others. Such transformations should be applicable to any spatial data object, i.e., any data object embedded in 3D space. For this reason, in amira an optional transformation matrix can be set for all data objects derived from the spatial data's base class. This transformation matrix is automatically applied to any display module visualizing the data object.

The transformations can be defined interactively using the "transformation editor." This editor allows easy transformation in 3D using so-called Open Inventor draggers, such as a 3D tab box or a 3D virtual trackball. In addition, absolute or relative translations, rotations, and scaling operations can be applied using text input. It is important to note that the data itself is not modified by such a transformation. In order to actually apply the transformation, a separate module is provided. This module transforms the point coordinates of a vertex-based data object (such as a surface or a tetrahedral grid) and resets the transformation matrix to the identity matrix. Voxel-based data objects such as 3D images must be resampled onto a new grid. This can be done using several different interpolation filters, either taking the original bounding box or using an enlarged one which encloses the complete transformed data set.

2.7 Parameters

Another concept which amira users have found very helpful across many different applications is the ability to add arbitrary parameters to a data object. Parameters are identified by a unique name and are associated with some value. The value may be a simple number or a tuple of numbers, a string, some binary data, or a subfolder containing an additional list of parameters. In this way a hierarchy of parameters can be defined. Parameters are used to store additional information for a data set, as may be contained in specific file formats. For example, in the case of confocal images, the wavelength of the emitted light and a description of the particular optics is often encoded. For medical data, the patient name or a patient id is usually stored together with many additional parameters. Some parameters are interpreted by certain amira modules. For example, a parameter called *DataWindow* is used to indicate the default greylevel window of an image data set. Similarly, a parameter called *Colormap* specifies the name of the default colormap used to visualize the data set. This way it is easy to associate additional information with existing data types, which then may be interpreted by custom modules. Parameters can be edited interactively using the amira parameter editor. In addition, they can also be set and evaluated via the Tcl command interface.

3 Features and Applications

In this section we wish to illustrate how amira can be applied within different areas of science and engineering. Though the spectrum of applications is rather wide, there are often similar tasks to be solved. Therefore, we will identify common requirements and show how these are addressed within amira.

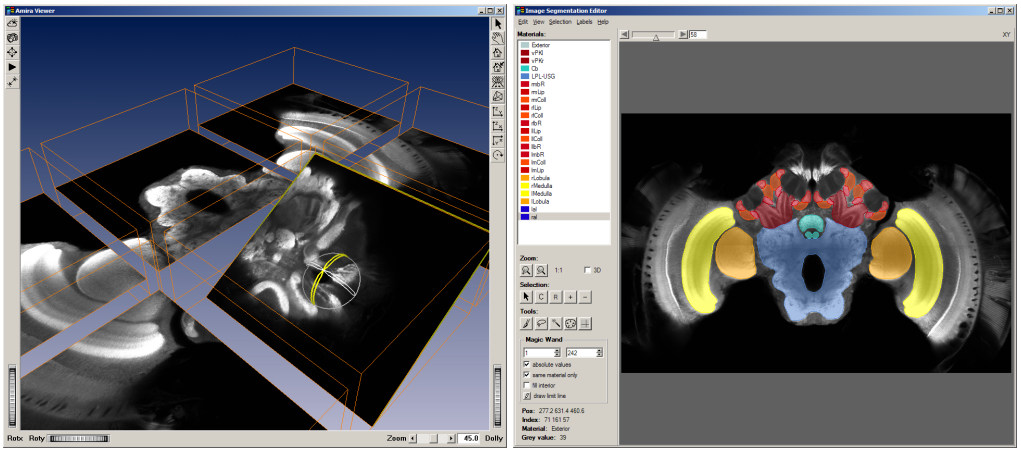

Figure 2: Left: Multi-part 3D confocal image stack of a bee brain visualized by slices. The orientation of an oblique slice can be easily adjusted using a trackball dragger. Right: The amira segmentation editor. This component allows the user to identify and separate different objects in a 3D image stack. Here different parts of the bee brain have been segmented.

3.1 Visualization of 3D Image Data

3D image data are important in medicine, biology, as well as in many other areas. Sources of 3D images include CT or MRI scanners, ultrasound devices, 3D confocal microscopes, and even conventional microscopes (which usually require the specimen to be physically cut into sections). The main characteristics of a 3D image is its regular structure, i.e., voxels arranged in a 3D array. Many modules in amira require a 3D image to have uniform or stacked coordinates, although rectilinear and curvilinear coordinates can be represented as well. In the case of uniform coordinates, all voxels have the same size. In the case of stacked coordinates, the distance between subsequent slices in z-direction may vary.

The most basic approach for investigating 3D images is to extract individual 2D slices. In amira two modules are provided for this, *OrthoSlice* and *ObliqueSlice* (see Fig. 2 left). The first module extracts axis-aligned slices, while the second displays arbitrarily oriented slices. In the latter case, the data must be resampled onto a 2D plane. This can be done using different interpolation kernels. Another useful module is *ProjectionView*, which computes a maximum intensity projection on the xy-, xz-, and yz-plane. In order to grasp the 3D structure of an image data set, isosurfaces can be computed or direct volume rendering can be applied. For the latter two different modules are provided. One utilizes the standard texture capabilities of modern graphics cards, while the other one makes use of special purpose hardware (VolumePro 1000 from TeraRecon Inc.). In any case a suitable colormap must be chosen to define how the image data are mapped to color and opacity. With the exception of isosurfaces, all methods can be applied not only to greyscale images, but also to RGBA color images and multi-channel images.

3.2 Image Segmentation

Image segmentation denotes the process of identifying and separating different objects in a 3D image. What constitutes an 'object' depends upon the application. Image segmentation is a prerequisite for geometry reconstruction from image data and for more advanced analysis of image data. Consequently, it is an important feature in an image oriented 3D visualization system such as amira.

In amira, segmentation results are represented by labels. For each voxel, a label is stored specifying which object or material to which this voxel belongs. In general, image segmentation

can not be performed fully automatically, and human intervention is necessary. For this reason in amira a special purpose component, the *segmentation editor*, is provided (see Fig. 2 right). The editor offers a variety of different tools for manual and semi-automatic segmentation, in both 2D and 3D. In the simplest case, regions can be selected using a lasso, a brush or thresholding. More advanced tools such as 2D or 3D region growing or a live-wire method are also provided. In region growing the user selects a seed point and adjusts the lower and upper bound of a greylevel interval. All connected voxels within this interval are then selected. In the live-wire tool, the user selects a starting point on a boundary and then drags the cursor roughly around the outline [3]. The minimum cost contour from the seed point to the current cursor position is displayed in real time. The cost is based on the image gradient and Laplacian, such that computed paths cleanly follow region boundaries.

Although segmentation is primarily performed in 2D, a 3D view of the currently selected regions is available at any time. For this purpose a fast point-based rendering technique is applied. Noisy regions or regions which have been falsely selected by a 3D threshold or region growing operation can be easily cleared by marking them in the 3D view using the lasso tool. Another approach for reducing the amount of work needed for image segmentation is to interpolate segmentation results between subsequent slices. Optionally, the interpolated results can be automatically adapted to the image data using a "snakes" technique [9]. Furthermore, shape interpolation from a few segmented orthogonal slices is provided by a 3D wrapping tool. The segmentation editor also provides a number of different filters, e.g., denoising and smoothing filters,and or morphological filters for erosion, dilation, opening, and closing operations. Various other experimental (research stage) amira modules exist, providing additional image segmentation methods, e.g. based on statistical shape models [10].

3.3 Geometry Reconstruction

After a 3D image has been segmented, i.e., after every voxel has been assigned to some material, a polygonal surface model can be created. Several algorithms have been described which attempt to construct a surface model by connecting contours in neighbouring slices in the appropriate way. However, these algorithms are not fail-safe, especially if multiple different materials are involved. In this case non-manifold surfaces must be created, i.e., surfaces with edges where more than two triangles join. In amira a robust and fast surface reconstruction algorithm is applied which triangulates all grid cells individually, similar to the marching cubes algorithm for computing isosurfaces [12]. This algorithm guarantees that the resulting surfaces are free from cracks and holes, that no triangles intersect each other, and that all regions assigned to different materials are well separated from each other. If additional weights are defined (by prior calculations) for each voxel, a smooth surface can be reconstructed. The weights are computed by applying a Gauss filter to the binary labels, so that a non-binary smooth result is obtained. A disadvantage of this technique is that small details of the segmented data set may be lost. Therefore a constrained smoothing method is also provided which ensures that the final surface is still consistent with the original labelling. A similar but more computationally expensive method has been described in [25]. An example of a smooth 3D model reconstructed by amira is shown in Fig. 3 (left).

3.4 Surface Simplification and Editing

Surfaces reconstructed from a segmented 3D image usually have a large number of triangles. In fact, the polygon count of the triangular surface is in the order of the voxel size. For many purposes the number of triangles needs to be reduced, i.e., the surface needs to be simplified. In amira this can be done using an advanced simplification algorithm based on edge contraction. The method tries to reduce the error induced by the simplification process as far as possible while simultaneously optimizing triangle quality. In order to control the maximal deviation, a quadric

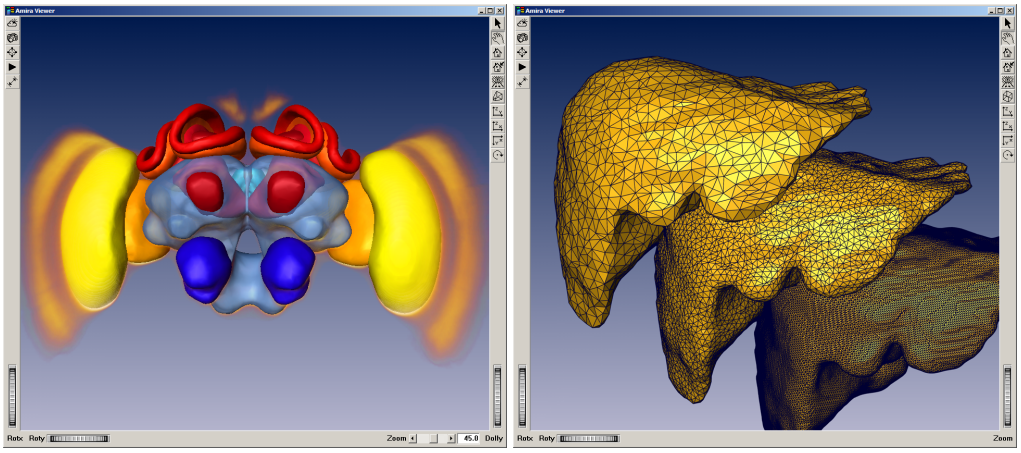

Figure 3: Left: Reconstructed polygonal surface model of a bee brain; a volume rendering of the image data has been superimposed, showing how well the model matches the image data. Right: Reconstructed model of a human liver, displayed at three different resolutions (1 cm maximum edge length, 0.5 cm maximum edge length, and surface at original resolution with 0.125 cm voxel size).

error metric is used (as proposed in [6]). In all cases, intersecting triangles are strictly avoided. The result of a simplified surface is shown in Fig. 3 right.

Simplification is not the only surface editing operation which can be performed in amira. For other operations an interactive *surface editor* is provided. Among other tasks, this editor allows the user to iteratively smooth or refine the surface (in whole or part); to cut parts out of a surface and to copy them into other surfaces; or to define boundary conditions on the surface. The latter is important when performing numerical simulations on the surface or on a tetrahedral finite-element grid derived from it. The surface editor also provides several tools for modifying the surface at a fine-grained level. In particular, individual edges can be flipped, subdivided, or contracted, and points can be moved. Also, tests can be performed to check whether the surface has intersections, holes, or inconsistently oriented triangles. Finally, triangles with a bad aspect ratio or with small dihedral angles can be found.

All these operations allow the user to interactively modify an arbitrary surface in such a way that a good tetrahedral grid can be generated afterwards. Grid generation itself is implemented as a separate module in amira using an advancing front algorithm [11, 8]. The grid quality can be improved by a subsequent smoothing or relaxation step. An example of a tetrahedral grid generated by amira is shown in Fig. 4 (left).

3.5 Alignment of Physical Cross-Sections

With the previously described techniques, polygonal surface models and tetrahedral volume grids can be reconstructed from 3D image stacks recorded by CT scanners, MR scanners, or confocal microscopes. Another common approach in microscopy is to physically cut an object into slices and to image each slice separately. In order to reconstruct geometries from such data, the individual slices usually need to be aligned with respect to each other. For this purpose another tool is provided in amira, the *slice aligner* (compare Fig. 4 right). The slice aligner supports interactive, semi-automatic and automatic alignment. The tool displays two slices of a 3D image stack at once. Different view modes can be selected that help to distinguish visually the slices. For example, one image can be displayed in green and the other one in red, or the colors of one image can be inverted. The image slice then can be manually translated or rotated. Semi-automatic

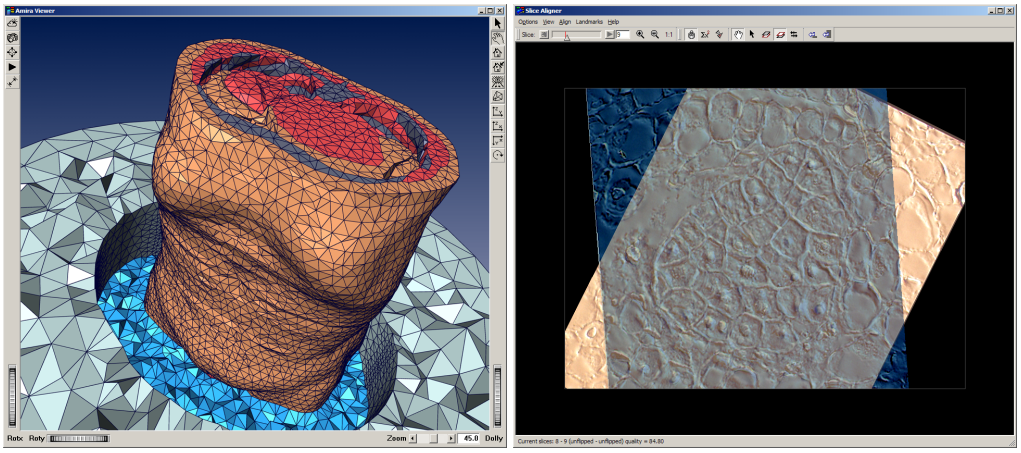

Figure 4: Left: Tetrahedral finite-element grid of a human body embedded in a device for hyperthermia treatment. Only parts of the grid are shown in order to reveal interior structures. Right: The amira slice aligner, an interactive tool for aligning 2D physical cross-sections.

alignment via landmarks is possible, too. Fully automatic pre-alignment can be achieved by matching the centers of gravity as well as the principal axes of the two images. Once this has been done, a multi-level optimization algorithm can be called which attempts to minimize the pixelwise difference of the two images.

3.6 Multiple Data Sets and 3D Registration

In biomedical applications users often work with multiple data sets. For example, one wants to compare images of multiple individuals, or images of the same individual recorded at different times, or images of the same object taken with different imaging modalities, such as CT and MR. In all of these cases it is crucial that multiple data sets can be visualized simultaneously. This requirement is met by amira in a natural way. In order to compare multiple data sets one can use (for example) semi-transparent displays. amira supports high-quality transparency with depth sorting and opacity enhancement at silhouettes. Other techniques are colorwash displays, where images of multiple data sets are superimposed on 2D slices, or computation and visualization of difference images. For surfaces it is also possible to compute the distance between the vertices of one surface and the nearest point on some other surface. The result can be visualized using conventional pseudo-coloring.

When multiple corresponding data sets are to be used, the problem of registering or aligning these data sets with each other becomes relevant. Here, the two major techniques are rigid and elastic registration. In the case of rigid transformation, the data set will be only translated, rotated and possibly scaled. Such transformations can be easily encoded in an affine transformation matrix, which are supported for all amira data objects. Thus, a manual rigid registration can be performed using amira's *transformation editor*. Another possibility is to make use of landmarks. Corresponding landmarks can be defined in both data sets with amira's *landmark editor*. Afterwards a rigid transformation can be computed which minimizes the squared distance between each pair of landmarks. Finally, a voxel-based automatic registration can also be computed. This method attempts to optimize a quality measure indicating the difference between both images. Several different quality measures are implemented, including the sum over squared pixel differences and a mutual information measure [24]. The latter is suitable for registration of multi-modal images, e.g., CT or MR, when there is no one-to-one correspondence between the grey values in the two images. Sometimes it is more appropriate to align reconstructed surfaces

instead of image data. In amira this is supported by an iterative method which automatically tries to find corresponding vertices and then minimizes the squared distance between these points.

In contrast to rigid registration, elastic registration is usually much more difficult to define. In addition, it requires image data to be resampled on a new axis-aligned grid. Currently, amira supports only an elastic registration method based upon landmarks. This method computes a Bookstein spline which exactly matches corresponding landmarks and smoothly interpolates in-between. This approach can be applied to both 3D images and triangular surfaces. An automatic voxel-based elastic registration method is currently under development.

3.7 General Data Processing and Data Analysis

In addition to the specific tools we have described, amira also provides other more general utilities for data processing. Probably one of the most important is a resampling module for reducing or enlarging the resolution of a 3D image or other data set defined on a regular grid. Some care must be taken when choosing a filter kernel for resampling. In amira several different kernels are supported, ranging from fast box and hat filters to a high-quality Lanzcos filter (which approximates a sinc function, the optimal filter from sampling theory), for finite images. Other tools are provided for cropping a data set, for enlarging it by replicating boundary slices, and for changing the primitive data type of a data set. For images, the most common primitive data types are bytes and 16-bit shorts, either signed or unsigned. Simulation data is usually encoded using 32-bit floating point numbers. In addition, in amira 32-bit signed integers and 64-bit floating point numbers are supported. While a scalar field has only one such component, any number of other components is also possible. For example, a vector field usually has three components. In amira a module is provided to extract one component from such a field, and to combine multiple components from different sources into a new field. Another valuable tool is the *arithmetic module*, used for combining multiple data sets by evaluating a user-defined arithmetic expression per voxel or per data value. In this way it is possible (for example) to subtract two data sets, compute the average, scale the data values, or mask out certain regions using boolean operations.

Another class of utility modules is related to statistical data analysis. This includes simple probing modules which evaluate a data set at some discrete points or plot it along a user-defined line segment. Moreover, a histogram of the data values can be computed, possibly restricted to some region of interest. Other modules are provided to compute statistical quantities such as volume, mean grey value, standard deviation, and so on for different regions encoded in a segmented label field; and also to compute volume-dose diagrams, or to count and statistically analyse the connected components in a binary labeled 3D image.

3.8 Finite-Element Post-Processing

Most of the tools described in the previous sections were related to the processing of 3D image data, or more generally, to data defined on regular 3D grids. However, other data types are also important; in particular, finite-element data defined on unstructured grids. amira supports the generation of triangular surfaces and tetrahedral volume grids suitable for numerical simulations. Such simulations are typically performed using some external code, but the results can again be visualized in amira. This task is known as finite-element post-processing. Besides tetrahedral grids, amira also supports hexahedral grids. Most of the general-purpose visualization techniques and analysis tools can be also applied to data on unstructured grids; for example, slice extraction, computation of isolines or isosurfaces, direct volume rendering (implemented via a cell projection algorithm), data probing, or computation of histograms. In addition, scalar quantities can be visualized using color-coding of the grid itself. In case of mechanical simulations, deformations are often computed. Such data can be visualized with displacement vectors, or by applying the displacement vectors to the initial grid sequentially such that an animation sequence

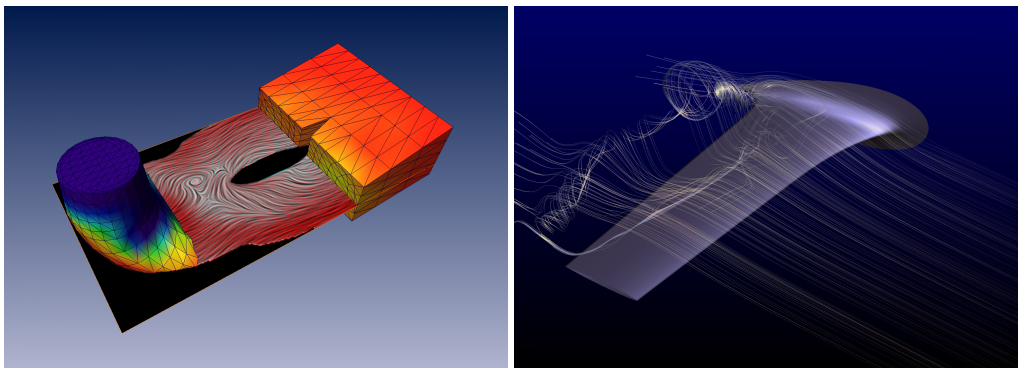

Figure 5: Left: Visualization of a turbine flow using the fast line integral convolution method in a user-selected plane. Right: Visualization of the 3D flow around an airfoil using the illuminated stream line technique.

is obtained. All of these methods can also be applied to visualize results from numerical simulations in biomedicine – e.g., simulations of mechanical loads in bones or of heat transport in tissue – or to visualize results from numerical simulations in engineering and related disciplines.

3.9 Flow Visualization

Flow visualization has evolved into an independent field of research in scientific visualization. Since flow fields are often generated by numerical computations, it can also be considered as a special form of finite-element post-processing. Beyond engineering domains such as computational fluid dynamics, where (e.g.) virtual windtunnel experiments are performed, flow visualization techniques are important also in in biomedicine – e.g., for analysing a simulated flow in blood vessels or an air flow in a nasal pathway.

Flow visualization techniques have been reviewed in depth in other chapters of this book. Therefore, here we list the different methods supported in amira without presenting algorithmic details.

Likely the simplest method for visualizing a vector field is to draw small arrows attached to discrete points. Arrows can be drawn on a slice, within the volume, or upon a surface in amira. More highly resolved and comprehensible visual representations can be obtained using texture based methods. amira supports fast line integral convolution, both on slices and on surfaces with arbitrary topology [21, 20] (see Fig. 5, left). Probably the most popular approach to reveal the structure of a flow field in 3D is to draw stream lines. amira includes support for illuminated streamlines (c.f. Fig. 5, right) – i.e., streamlines that are rendered as line primitives with a lighting applied to them [22]. This allows rapid rendering of many streamlines, while at the same time highlighting the 3D structure of the field. Another method based on streamline computation is the display of stream ribbons (c.f. Fig. 6, left). In addition to streamline, stream ribbons also show the swirl and torsion of flow fields. A further extension supported by amira is the stream surface (see Fig. 6, right). A stream surface is spanned by multiple stream lines starting from some user-defined seed shape or rake. Stream surfaces are commonly started from a straight line, or from a line traced along the normal or binormal direction of the vector field. All of these stream visualization techniques are highly interactive. While seedpoint distributions can be automatically calculated, users can also select and interactively manipulate seed points and structures, thus supporting the investigation the flow field and highlighting of different features. Each of these techniques again support 3D interaction, allowing the user to pick and move the seed volume or seed shape directly within the 3D viewer.

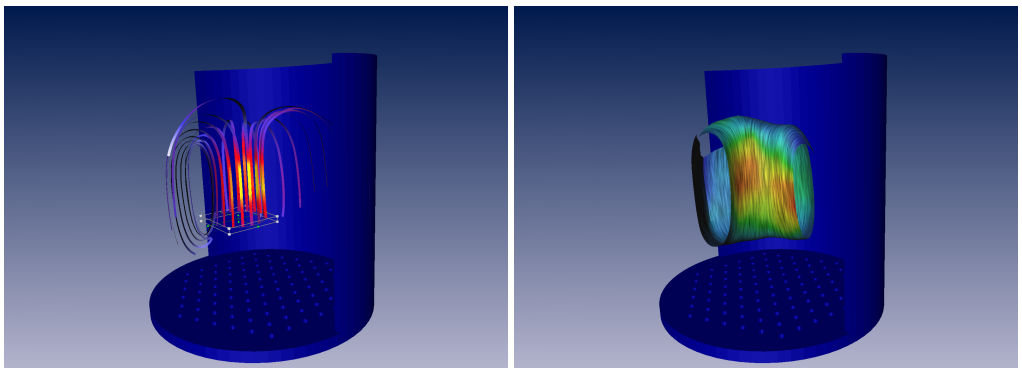

Figure 6: Visualization of fluid flow within a bioreactor. Left: Stream ribbons starting in the interactively positioned seed box. Right: Stream surface with the tangential flow depicted by line integral convolution.

4 amiraVR and Other Extensions

The modular structure of amira makes it possible to extend the system in various ways and to provide extensions addressing more specific application areas. Some major extensions are directly available as optional products. Among these, the most prominent is amiraVR, which allows amira to operate on a large tiled display or in a multi-wall virtual environment.

4.1 amiraVR

High-resolution multi-wall displays have received considerable attention in scientific visualization over the last few years. Two major approaches have held special interest. The first is flat multi-tile displays, often called “power walls.” Here, the goal is to create a very high resolution display, usually by combining several projectors in one rear projection system. With such a display, fine details in high resolution data sets can be visualized and shown to a small or medium sized group of observers. The other approach is to construct an immersive environment for virtual reality applications. Usually, such environments incorporate multiple screens in a non-planar configuration. In order to compute correct views the actual position of the observer needs to be tracked. Non-tracked observers usually see somewhat distorted images and artifacts at the boundary between neighbouring screens.

For performance reasons, the images for the different parts of a tiled display or for the different screens of a VR environment should in rendered in parallel, if possible. The simplest approach from a software perspective is to use a multi-processor shared-memory machine with multiple graphics pipes. This architecture is implemented by SGI Onyx systems, and also by other workstations from vendors such as Sun or HP. To support such an architecture, it must be possible to perform the actual rendering in parallel using multiple threads. This is supported by amira. amira’s rendering process involves the traversal of an Open Inventor scene graph and the calling of render methods for each node in this graph. Although early versions of Open Inventor were not thread-safe originally, this is currently the case (since v3.1 release by TGS).

The use of amiraVR requires specification of the display configuration – i.e., the actual setup of the display system and (optionally) the tracking system. When the configuration file is read, additional graphics windows are opened on the graphics pipes as specified. In case of a tiled display, the modules can be controlled via their usual interface with the 2D mouse. For user interaction within an immersive environment, 3D versions of all the standard GUI elements of amira data objects and modules are provided. In addition, a user-defined 3D menu can be displayed. Interactive elements such as slices or draggers, which can be picked in the viewer window using

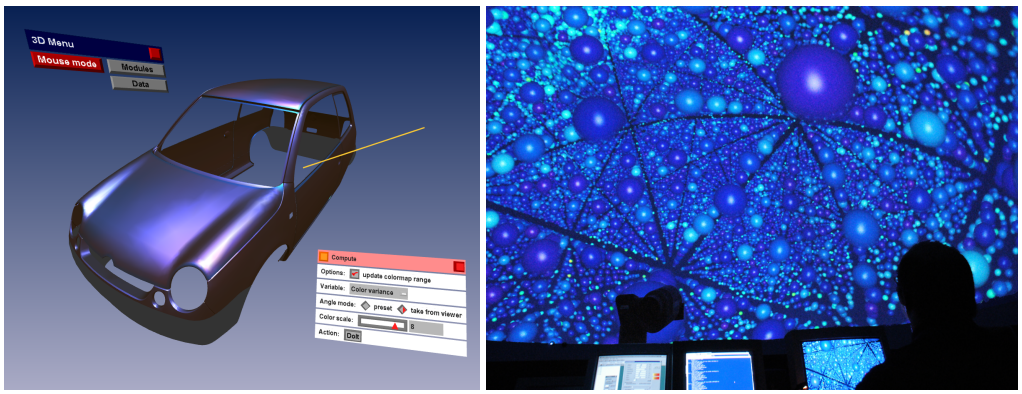

Figure 7: Left: Coating on a car body investigated with amiraVR. For every module a 3D version of its respective user interface can be used. This allows full control of amira from within an immersive environment. Right: Atoms in a crystal lattice shown in a dome using amiraVR. The dome was illuminated by six laser projectors with partially overlapping images, with five arranged in a circle and one at the top (courtesy Carl Zeiss Jena).

the 2D mouse, also react on events generated by a tracked 3D "mouse." Thus slices can be easily translated and rotated in 3D, or seed volumes for flow visualization can be easily adjusted. All objects which can be picked with the 3D mouse, including menus, provide some visual feedback. This is important to make interaction in VR feasible. With these mechanisms, data can be visualized in a VR environemnt in a similar way to that of the desktop GUI. All modules and networks can be loaded without modification. This allows users (for instance) to prepare visual demonstrations for large display systems or VR environments on a PC or notebook computer.

4.2 Developer Version

For a modern visualization system it is crucial that new functionality can be added by the user. We have already stated that this is possible within amira, and simplified by amira's modular and object-oriented design. The amira developer version provides all of the header files and documentation required to derive new modules from existing classes. It also provides a unified make system, which creates either makefiles or project files for integrated development environments such as Microsoft Visual Studio. New modules based on Tcl code, *script objects*, do not require the developer version, and can be implemented from within amira's base version.

4.3 Molecular Visualization

For the application domains of chemistry, biochemistry and molecular biology, the amira extension amiraMol has been developed. This provides tools for the analysis of complex molecules, molecular trajectories, and molecular conformations. The extension is useful for inorganic and organic chemistry, but its emphasis is upon the analysis of biomolecules.

The central goal of molecular biology is to elucidate the relationship between sequence, structure, properties, and function of biomolecules. Such knowledge allows one to understand biological processes and pharmaceutical effects, as well as to identify and optimize drug candidates. Since bioactivity is guided by molecular shape and molecular fields, amiraMol provides special means for analysing the dynamic shapes of molecules as well the corresponding molecular fields. Standard and novel representations are available for visualizing biomolecules. Arbitrary grouping hierarchies on the molecule's topology can be defined for coloring, masking, and selection.

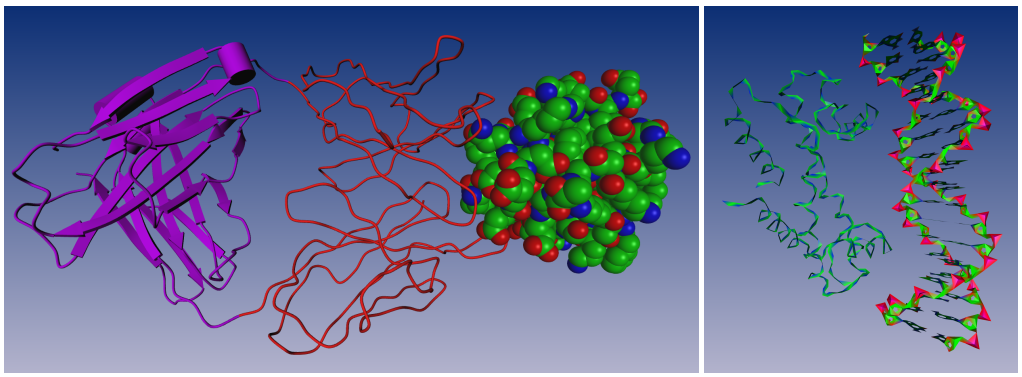

Figure 8: Left: Complex consisting of a mouse antibody and an antigen of Escherichia Coli. Secondary structure representation on the left side, pure backbone representation in the middle. Van-der-Waals balls depict the antigen. Right: Bond angle representation of a complex of the FIS protein (Factor for Inversion Simulation) and a DNA fragment (coloured according to atom types).

amiraMol supports common molecular file formats, such as PDB, Tripos, Unichem and MDL. For trajectories from molecular dynamics simulation, amira provides its own native data format, but also supports CHARMM's dcd format. Standard techniques for molecular visualization available in amiraMol are wire frame, ball-and-stick, van-der-Waals spheres, and secondary structure representations, see Fig. 8 (left). Beyond these, a novel technique called bond-angle representation has been developed, c.f. Fig. 8 (right), which displays a triangle for every group of three atoms connected by two bonds. This representation requires only few geometric primitives and provides a comprehensible view of the 3D structure.

Several colour schemes can be used to enhance the molecular representations. They permit the user to colour the atoms according to a number of attributes such as atomic number, charge, hydrophobicity, radius, or the atom's index.

To take into account structural information beyond atoms and bonds, groups can be defined. A group is a combination of atoms and other groups which can (for example) represent a residue, a secondary structure, or an α-chain. The groups are organized into levels, such as the level of residues or the level of chains. The user can define arbitrary new groups and levels by using expressions. Groups may contain not only atoms but groups of arbitrary and possibly different levels. Atoms can be coloured according to their membership in a group of a chosen level.

In order to support easy investigation of molecular structures, amiraMol offers a *selection browser*. This displays all groups in a chosen level of the hierarchy. Furthermore, additional information such as type and membership of the groups can be displayed. Groups can be selected by clicking on them in the browser, by using expressions, or through interaction within the viewer. Groups that were selected in the browser will also be highlighted in the viewer and vice versa. Apart from selecting, the selection browser offers the possibility to hide arbitrary parts in all of the above-mentioned representations, so that the user can concentrate on particular regions of interest.

Shape complementarity is an important aspect in molecular interactions. Shape properties are relevant for manual docking of ligands to proteins and for automated docking procedures. The characterization of molecular shapes is therefore very useful for molecular modeling. In addition to the above techniques, amiraMol offers algorithms for generating triangular approximations of solvent excluded and solvent accessible surfaces [18]. Additionally, van-der-Waals surfaces and interfaces between arbitrary parts of a molecule or between different molecules can be computed. All triangular surfaces can be colour-coded by arbitrary scalar fields.

Of course, the above molecular representations can be combined with all other visualization

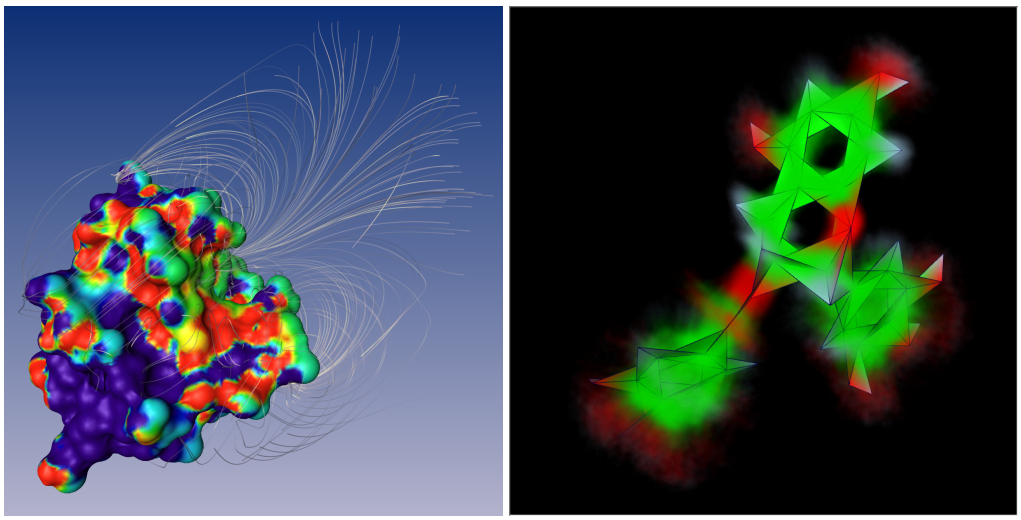

Figure 9: Left: Solvent accessible surface of the ribonuklease T1, pseudocoloured according to the molecule's electrostatic potential; illuminated streamlines depict the electrostatic field. Right: Configuration density and superposed bond angle representation of Epigallocatechin-Gallat.

techniques available in amira. For instance, the electrostatic field can be depicted using vector field visualization techniques, cf. Fig. 9 (left), or electron density isocontours can be computed.

However, molecules are not static, but rather in constant motion. Typically they fluctuate for long periods around a certain 'meta-stable' shape. Less frequently, they also undergo larger shape changes. Taken together, individual molecular configurations with similar shapes are called conformations. Conformation analysis is used to determine the 'essential shapes' of molecules and the probabilities of transitions between these shapes. amiraMol offers an extensive set of tools for visual analysis of trajectories from molecular dynamics simulations. This allows users (e.g.) to determine representatives of conformations and to depict conformations [19]. In Fig. 9 (right) a representative molecular shape is displayed, together with the shape density of the corresponding conformation.

5 Summary

We have presented the general design concepts behind amira, a 3D visualization and geometry reconstruction system. We have also given an overview of the different techniques and algorithms implemented in this system. It was shown that the combination of different concepts such as object orientation, a simple and well-structured user interface, the integration of highly-interactive components such as the segmentation editor, intuitive 3D interaction techniques, a powerful scripting interface, and a broad range of advanced and innovative algorithms for visualization and data processing have yielded a powerful software system which can be usefully applied to many problems in medicine, biology, and other scientific disciplines.

Acknowledgments We would like to sincerely thank the many students, researchers, and software developers who contributed to the amira suite: Maro Bader, Werner Benger, Timm Baumeister, Daniel Baum, Philipp Beckmann, Robert Brandt, Martina Bröhan, Liviu Coconu, Frank Cordes, Olaf Etzmuß, Andrei Hutanu, Ralf Kähler, Ralf Kubis, Hans Lamecker, Thomas Lange, Alexander Maye, André Merzky, Olaf Paetsch, Steffen Prohaska, Hartmut Schirmacher, Johannes Schmidt-Ehrenberg, Martin Seebaß, Georg Skorobohayj, Brygg Ullmer, Tino Weinkauf,

Natascha Westerhoff, Gregor Wrobel and Stefan Zachow. Special thanks go to Brygg Ullmer for his editing suggestions that improved grammar, vocabulary and style. We also would like to express our gratitude to Peter Deuflhard who supported this project over many years. Furthermore, we thank all research collaboration partners and early users who helped us with their requirements to shape the system and make it practically useful.

References

[1] G. Abram and L. A. Treinish. An extended data-flow architecture for data analysis and visualization. In *Visualization '95 proceedings*, pages 263–270. IEEE Computer Society Press, Oct. 1995.

[2] Amira - Advanced 3d Visualization and Volume Modeling. Software and user's guide available from *www.amiravis.com*.

[3] W. A. Barrett and E. N. Mortensen. Interactive live-wire boundary extraction. *Medical Image Analysis*, 1(4):331–341, 1997.

[4] D. S. Dyer. A dataflow toolkit for visualization. *IEEE Computer Graphics & Applications*, 10(4):60–69, July 1990.

[5] D. Foulser. Iris explorer: a framework for investigation. *ACM Computer Graphics*, 29(2):13–16, may 1995.

[6] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[7] C. Gunn, A. Ortmann, U. Pinkall, K. Polthier, and U. Schwarz. An extended data-flow architecture for data analysis and visualization. In *Visualization and Mathematics*, pages 249–265. Springer Verlag, 1997.

[8] H. Jin and R. I. Tanner. Generation of unstructured tetrahedral meshes by advancing front technique. *Int. J. Numer. Methods. Eng.*, 36:217–246, 1993.

[9] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *Academic Publishers*, 1987.

[10] H. Lamecker, T. Lange, and M. Seebass. Segmentation of the liver using a 3d statistical shape model. ZIB preprint 2002, submitted.

[11] R. Löhner and P. Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *Int. J. Numer. Methods. Fluids*, 8:1135–1149, 1988.

[12] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California, July 1987).

[13] Open Inventor from TGS. Software and user's guide available from *www.tgs.com*.

[14] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, 1994.

[15] S. G. Parker, D. M. Weinstein, and C. R. Johnson. The SCIRun computational steering software system. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools for Scientific Computing*, pages 5–44. Birkhauser (Springer-Verlag), Boston, 1997. U. of Utah.

[16] Qt whitepaper. Online available from *www.trolltech.com*.

[17] J. Rasure and C. Williams. An integrated data flow visual language and software development environment. *Journal of Visual Languages and Computing*, 2:217–246, 1991.

[18] M. F. Sanner, A. J. Olson, and J.-C. Spehner. Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers*, 38:305–320, 1995.

[19] J. Schmidt-Ehrenberg, D. Baum, and H.-C. Hege. Visualizing dynamic molecular conformations. In R. J. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of IEEE Visualization 2002*, pages 235–242, Boston MA, USA, October/November 2002. IEEE Computer Society, IEEE Computer Society Press.

[20] D. Stalling. *Fast Texture-Based Algorithms for Vector Field Visualization*. PhD thesis, Zuse Institute Berlin (ZIB), 1998.

[21] D. Stalling and H.-C. Hege. Fast and resolution independent line integral convolution. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 249–256, Aug. 1995.

[22] D. Stalling, M. Zöckler, and H.-C. Hege. Fast Display of Illuminated Field Lines. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):118–128, Apr. 1997.

[23] C. Upson, T. A. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[24] W. M. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multi-modal volume registration by maximisation of mutual information. *Medical Image Analysis*, 1(1):35–51, Mar. 1996.

[25] R. Whitaker. Reducing aliasing artifacts in iso-surfaces of binary volumes. In *IEEE Volume Visualization and Graphics Symposium*, pages 23–32, October 2000.