Computer Assisted Surgery, 4:129-143, 1999.

AnatomyBrowser: A Novel Approach to Visualization and Integration of Medical Information

P. Golland[†], M.Sc., R. Kikinis[‡], M.D., M. Halle[‡], Ph.D., C. Umans[‡], B.A.,
W.E.L. Grimson[†], Ph.D., M.E. Shenton[‡], Ph.D., J.A. Richolt[‡], M.D.

[†]Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139
[‡]Surgical Planning Laboratory, Brigham and Women's Hospital, Boston, MA 02115

Contact information:
Polina Golland,
Artificial Intelligence Laboratory,
545 Technology Square, #737,
Cambridge, MA 02139
phone: (617) 253-8837
fax: (617) 258-6287
e-mail: polina@ai.mit.edu

Abstract

In this paper, we present a novel technique for visualization of 3D surface models, as well as its implementation in a system called AnatomyBrowser. Using our approach, visualization of 3D surface models is performed in two separate steps: a pre-rendering step, in which the models are rendered and saved in a special format, and an actual display step, in which the final result of rendering is generated using information from the pre-rendering step. While pre-rendering requires high-end graphics hardware, the final image generation and display can be implemented efficiently in software. Moreover, our current implementation of AnatomyBrowser interface uses the Java programming language and can therefore be readily run on a wide range of systems, including low-end computers with no special graphics hardware.

In addition to visualization of 3D models and 2D slices, AnatomyBrowser provides a rich set of annotation and cross-referencing capabilities. We demonstrate several possible applications for AnatomyBrowser, including interactive anatomy atlases, surgery planning and assistance in segmentation.

[1]This paper presents research that was supported in part by Mitsubishi Electric Research Laboratories and NSF grant IIS-9610249.

1 Objective

With recent developments in MRI technology, it has become possible to obtain high quality medical images that are extremely useful for clinical studies, surgical planning and other applications. This has spurred rapid development of a family of information extraction and analysis algorithms using medical imagery. Examples include segmentation, registration, shape and deformation modeling. And with the introduction of such images and methods of analysis, the importance of visualization of the results, especially 3D data, has become apparent as well.

In this paper, we introduce AnatomyBrowser, a visualization and integration package for medical applications. A novel approach to 3D surface model visualization has been developed and implemented in this system. Visualization is performed by two separate components of the system: the back-end component and the user-end interface. The back-end component uses special graphics hardware to render the models and save both intensity and depth information in a special representation called *multi-layer images*. The user-end interface reads the multi-layer images of the scene and generates an image to be displayed for the user. Since multi-layer images contain depth information, the user-end interface supports 3D model manipulation, such as partially transparent surfaces, changes in surface color, depth queries, etc.

The image generation process in the user-end interface requires little computational resources, and can be run on a low-end computer. The user-end interface is implemented as a Java applet, and as a consequence, it is platform independent. It can be readily run using any web browser that supports Java. In fact, it is available on-line [1].

Possible applications of AnatomyBrowser include interactive anatomy atlases, which was an original inspiration for this work, assistance in model-based segmentation, surgery planning and others.

1.1 Related Work

Work in digital anatomy atlases was pioneered by Höhne *et al.* [6], followed by other research groups [7, 11] in an effort to develop both extensive medical image data bases and visualization software suitable for display and manipulation of medical data. Clinical systems for surgical planning, surgical simulation and image guided surgery [5, 14] are other examples of applications that depend greatly on visualization tools. These systems usually provide features specific for the type of surgery, while the atlas oriented visualization systems tend to support fairly general visualization and annotation capabilities. A common feature among these systems is a need for high-end graphics hardware. To the best of our knowledge, AnatomyBrowser is the first system to provide an extensive set of visualization and cross-referencing capabilities without using specialized graphics support at the user-end of the system.

Because of its design, AnatomyBrowser is better suited for visualization of atlases, but the same approach to visualization can be used in a surgical tool. Section 5 contains examples of using AnatomyBrowser as a general visualization tool, as well as a reference tool for surgical applications.

Digital atlases are commonly used for model driven segmentation. In this application, the atlas is treated as a template that is to be warped by the algorithm to match the input grayscale data set ([10] contains a survey of deformable model methods). Another approach to segmentation that uses atlases is based on registration of grayscale images. The atlas grayscale data set is registered to the input grayscale data set, and the segmentation is mapped from the atlas onto the input data using the results of registration (see [9] for a comprehensive survey on registration methods). In both cases, AnatomyBrowser can provide visualization of intermediate results in the algorithms.

Before we proceed with the system description, we would like to stress that AnatomyBrowser is a visualization tool for displaying segmentation results combined with other types of medical information, but it is not directly a part of a segmentation process. A segmented data set can be an output of an automatic segmentation algorithm or a result of manual segmentation. It is processed by AnatomyBrowser in exactly the same manner, independently of the way it was obtained.

This paper is organized as follows. In the next section, our approach to 3D visualization is introduced and discussed. Sections 3 and 4 describe implementation of the AnatomyBrowser back-end components and its user-end interface respectively. Section 5 presents several examples of using AnatomyBrowser on various data sets, followed by the system performance evaluation and concluding remarks.

Materials and Methods

2 A Novel Approach to 3D Model Visualization

It has traditionally been accepted that special hardware and software is required for visualization of 3D surface models. Rendering of 3D surface models is a computationally intensive operation that requires special hardware to achieve acceptable update rates. And since rendering is considered an integral part of the visualization process, the visualization itself is thought to be feasible only on high-end computers with hardware graphics acceleration. In this paper, we propose a change in this paradigm. We separate the visualization process into two steps. First, the models are pre-rendered using a high-end graphics system, and the resulting images are saved in an intermediate format, called *multi-layer images*. Then multi-layer images are processed by the user-end interface to produce an image of a 3D scene. Multi-layer images are organized in such a way that the image generation by the user-end component requires only modest computational resources.

This approach can be considered a compromise between dynamic rendering of a scene and a set of static images of the same scene. Multi-layer images contain depth information, and therefore allow partial manipulation of the scene, such as removal of models, changing surface transparency, changing surface color, etc. But they are not equivalent to dynamic rendering, as they only provide the views that were selected and rendered in the pre-processing step.

The input to the pre-rendering step is a set of surface models of the anatomical structures of interest. The surface models are represented using polygonal meshes. During the pre-rendering step, every model is rendered in isolation, with all other models removed from the image. Two images are computed and saved for every model: an intensity image generated by the renderer, and a depth image. To compute the depth values, every vertex of the model mesh is projected onto the optical axis of the imaginary camera, and the depth of the mesh faces is obtained by interpolating between the depth values of the vertices.

Intensity and depth images of all models are then combined in the following way: for every pixel, a stack of $\langle k, I, d \rangle$ tuples is saved, where k is a unique index associated with a surface model (i.e., an anatomical structure that this model represents), I is the surface intensity from the intensity image, and d is the depth value from the corresponding depth image. The tuples are sorted in the order of increasing depth (Fig. 1). In order to obtain the resulting image intensity for a particular pixel, intensity values in the corresponding stack are treated as colored (and maybe partially transparent) layers that are piled on top of each other, hence the name "multi-layer image".

This approach is reminiscent of *z-buffering* techniques [4], with one significant difference: for a particular pixel, a z-buffer stores a single intensity value, which gets overwritten every time a surface

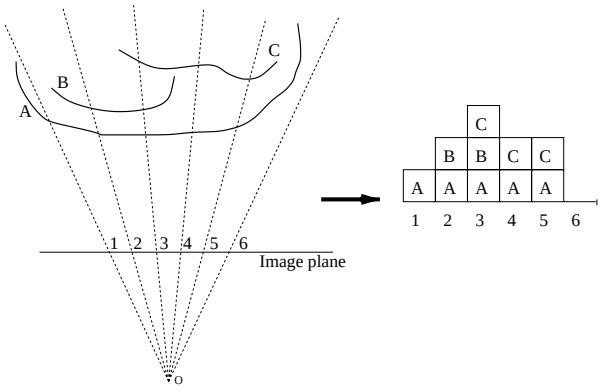


Figure 1: Multi-layer image. 3D scene consists of three surfaces (A,B,C). The view rays and the resulting stacks in the multi-layer image are shown for six different pixels. Intensity and depth values are omitted for simplicity, and only the structure indices are shown.

with a smaller depth value is encountered, while in a multi-layer image the previous intensity value gets stored in a stack when a new intensity with a smaller depth value is found for the pixel.

At rendering time, all models are assigned the color white and are treated as fully opaque. Surface colors and opacities are set at display time and are used to compute the resulting image intensity.

For modeling colored surfaces, we assume that the reflectivity properties of a surface are independent of the light wavelength [4]. Consider a white surface that produces an image of intensity (color) I_w at a certain pixel. Strictly speaking, the color of the image of the white surface is a 3-vector $[I_w\ I_w\ I_w]$, but we treat it as if it were a scalar value because its three components are always equal under white illumination. Suppose now we change the color of the surface to $\mathbf{C} = [R\ G\ B]$ without changing its reflectivity properties, such as diffuse and specular reflectivity coefficients. According to our model, the image intensity at the same pixel will change to $\mathbf{c} = [r\ g\ b]$ so that

$$\mathbf{c} = \mathbf{C}I_w \iff [r\ g\ b] = [RI_w\ GI_w\ BI_w]. \tag{1}$$

This implies that the color of a layer in a multi-layer image can be obtained by simply multiplying the layer intensity, which is computed during the pre-rendering step, by the color of the corresponding surface, which is set by a user at display time.

To handle partially transparent surfaces, every surface model gets assigned an *opacity value* α, ranging between 0 and 1 ($\alpha = 0$ corresponds to a fully transparent surface, $\alpha = 1$ corresponds to a fully opaque surface). We use a pre-multiplied color representation with a front-to-back compositing algorithm [2] for layer compositing. For a layer of color $\mathbf{c} = (r, g, b)$ and opacity α, a pre-multiplied color is defined as a 4-vector

$$\hat{\mathbf{c}} = [\alpha\mathbf{c}\ \alpha] = [\alpha r\ \alpha g\ \alpha b\ \alpha]. \tag{2}$$

Combining with Eq. (1), we obtain

$$\hat{\mathbf{c}} = [\alpha RI\ \alpha GI\ \alpha BI\ \alpha], \tag{3}$$

where I is the intensity of the layer computed in the pre-rendering step, and $\mathbf{C} = [R\ G\ B]$ is the user-defined surface color. Pre-multiplied colors provide a very efficient way to composite color layers. It can be shown [2] that a layer of (pre-multiplied) color $\hat{\mathbf{f}}$ composited on top of a layer of color $\hat{\mathbf{b}}$ is equivalent to a single layer of (premultiplied) color

$$\hat{\mathbf{f}} + (1 - \alpha)\hat{\mathbf{b}}, \tag{4}$$

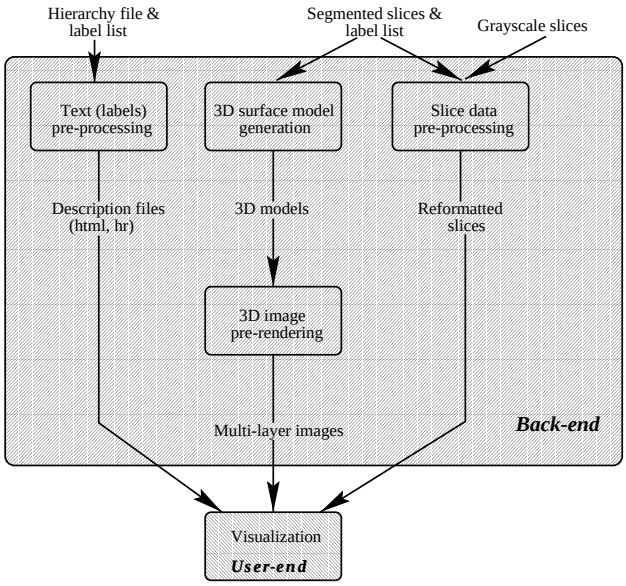


Figure 2: Data flow in AnatomyBrowser. Arrows indicate flow of data between different components of the system.

where α is the opacity of top layer $\hat{\mathbf{f}}$. This is a very simple compositing rule that requires only a few additions and multiplications and can be implemented efficiently using lookup tables for surface colors and opacities. Additional time savings come from the fact that for many pixels, it is not necessary to composite all the layers in their stacks, as the first visible surface is usually opaque. A simple test on the opacity of the resulting layer can be used to detect this situation and stop the compositing process.

To summarize, our approach to visualization of 3D surface models consists of two separate stages. First, the models are rendered and for every pixel in the image, a set of surface intensities and depth values is saved in a multi-layer image. This operation requires high-end graphics hardware for rendering. Then at the display time, the intensity values are multiplied by the user defined colors and opacities and composited using front-to-back compositing rule. This process can be efficiently implemented in software, as it does not require a lot of computation.

The next two sections describe AnatomyBrowser, a system that employs the technique presented in this section for visualization of medical images.

3 AnatomyBrowser Back-end Components

AnatomyBrowser consists of two parts: the back-end system for data pre-processing and the user-end interface for data visualization. Fig. 2 shows the data flow in the system. The input to the back-end system includes grayscale and segmented data sets, a list of anatomical structures with the label values used for segmentation of the structures, and an optional hierarchy file that defines organization of the structures.

At the pre-processing stage, 3D surface models are generated, rendered and saved as multi-layer images, the slice data sets are resampled and compressed and the text information is processed to establish correspondence between different types of information available to the system. An anatomical structure can have information of a different nature associated with it: a name, a

3D surface model, a set of 2D slices, or text notes. In order to tie them together and provide efficient cross-referencing between them, AnatomyBrowser automatically assigns a unique index (id) to every anatomical structure in the input data set, i.e., in the list of structures provided by the user. The index is then used by the back-end components for tagging the data associated with the structure and by the user-end components for looking up information on the structure.

The current implementation of the model generation and the model rendering components is based on the Visualization Toolkit (VTK), a software package that uses available GL libraries to take advantage of graphics resources of a particular system [13]. Any other rendering library that supports scene manipulation (e.g., Inventor by SGI) could also be used for this purpose.

3.1 3D Model Generation

Surface models are constructed from segmented data sets using the Marching Cubes algorithm [8]. First, every border voxel in the segmented slice set is considered with its neighboring voxels, and a single patch of the surface is constructed based on the local behavior of the boundary. Next, patches from neighboring voxels are merged to form a single mesh that approximates the surface of the structure. In order to decrease the number of polygons (triangles) in the representation and to improve the surface appearance, some amount of decimation and smoothing is performed on the mesh under constraints on the accuracy of representation. Each surface model is saved in a separate file (one file per model) to be read by the pre-rendering component. We used an implementation of the algorithm that is publicly available as part of VTK [13].

Note that the models are specified in the coordinate system naturally defined by the slice volume. This simplifies implementation of cross-referencing between 3D models and cross-sectional views.

3.2 3D Image Pre-rendering

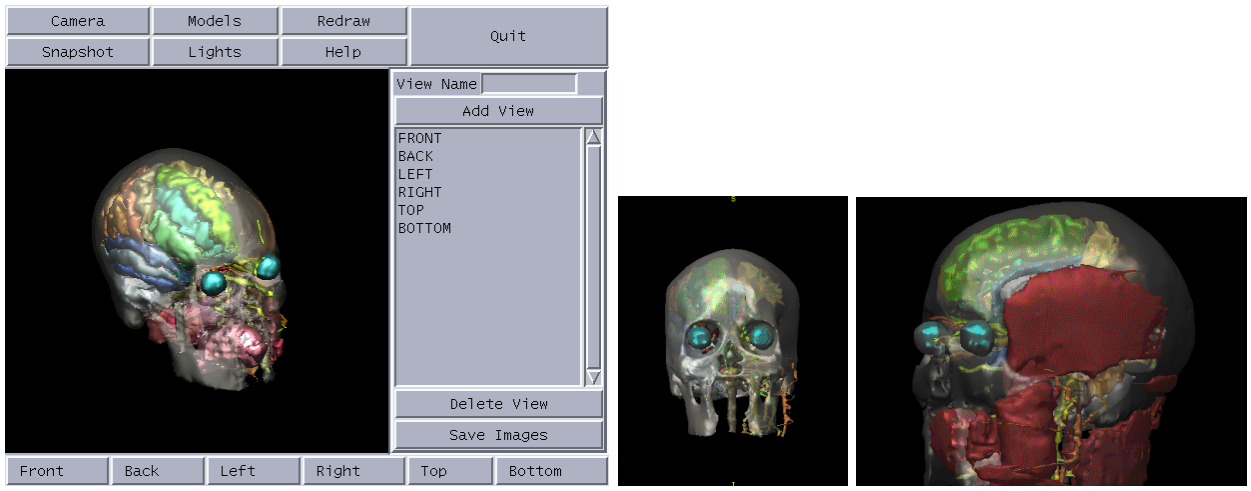
The main difference between AnatomyBrowser and other visualization packages lies in this component. It is in charge of model rendering and multi-layer image generation, and is essentially an implementation of the pre-rendering step described in Section 2.

The current implementation of this component is very similar to other 3D model viewers used in the field [5, 6, 14]. The viewer provides controls for adjusting global scene parameters, such as a viewpoint and lighting, as well as surface properties of individual models (color, transparency and reflectivity). Fig. 3a shows the viewer interface with the 3D surface models for the brain atlas data set. The menu buttons correspond to control panels for scene manipulation. The viewer also maintains a list of views to be saved as multi-layer images. The default is the six standard views (front, back, left, right, top and bottom), but other views can be added to the list by adjusting the scene parameters until the image is satisfactory and clicking on "Add view" button on the viewer control panel. Fig. 3b shows additional examples for the brain atlas data set generated by AnatomyBrowser.

After selection of views is completed, the rendering process is activated. For each view, a separate multi-layer image is generated and stored.

3.3 Slice Reformatting

The slice data sets undergo a set of transformations in the back-end component. The goal of this component is to provide the best representation of slice data for the user-end interface.



(a) Viewer interface (b) Additional examples of 3D views

Figure 3: Pre-rendering component of AnatomyBrowser. (a) The viewer interface with the brain atlas data set. The view list contains the six standard views. (b) Additional examples of the 3D views generated for the brain atlas data set.

Grayscale slice intensities are scaled from a 16 bit representation to an 8 bit representation based on the user defined window width w and window level l parameters. Window level l defines the grayscale level that will get mapped to 128, and window width w defines the width of the grayscale range to be mapped to 256 grayscale levels of the display:

$$\hat{I}(I) = \begin{cases} 0, & I < l - w/2 \\ 128 + \frac{256}{w}(I - l), & l - w/2 \leq I \leq l + w/2 \\ 255, & I > l + w/2. \end{cases} \tag{5}$$

Segmented slice intensities are remapped using the set of unique indices associated with the anatomical structures. For example, if a certain model was segmented using the value 15 and was assigned the index 8, all the pixels in the segmented data set with values of 15 will get assigned the value 8. After remapping is performed, the pixel value can be used as an index into a lookup table for the structure name, color, transparency, etc.

Both grayscale and segmented slice volumes are resampled along three standard directions (axial, coronal, sagittal) using an isotropic voxel size. The size of the new voxels is determined by the smallest dimension of the original voxels. This operation is needed to speed up the display of slices, trading storage space for performance time. We have also implemented and tested a version that resamples the volume on-line in the user-end interface, but it proved to be significantly slower than the current implementation that fetches resampled slices off a disk. The system can switch between the two implementations in a manner transparent to the user. After resampling, grayscale slices are stored as *gif* files and segmented slices are compressed using run-length encoding.

3.4 Structure Hierarchy

AnatomyBrowser supports hierarchical organization of anatomical structures. A simple text file format is used to specify the nodes of the hierarchical tree. For example,

```
[Head]
? Brain
x Skin
x Skull

[Brain]
x Temporal lobe
x Frontal lobe
x Parietal lobe
```

defines a hierarchy tree with a root named "Head" that has three children: two leaves "Skin" and "Skull", and node "Brain" that has in turn three children of its own: "Temporal lobe", "Frontal lobe" and "Parietal lobe".

A name can be declared to be a single structure (`x`) or a group of structures (`?`). The group of structures is specified by listing all the elements (structures or subgroups) right after the declaration of the group name (`[...]`). Hierarchy files can be as simple as a list of structures, or as complicated as a tree of several levels with more than a hundred structures in it (e.g., the brain atlas in Fig. 4).

The back-end system reads the hierarchy file, verifies that there exists a surface model for every name that was specified as a single structure, and attaches the structure index to the name in the hierarchy. A new, augmented, hierarchy file is then written out for the user-end interface. In addition to the hierarchy file, an *html* file is generated by this component. The *html* file contains all the parameters required by the interface, such as file names, slice volume size, voxel size, and others.

To summarize, after processing the input data, the back-end part of the system generates the following files:

- an html file that defines all the necessary parameters for the interface;
- a hierarchy file that specifies organization of the anatomical structures;
- multi-layer images (one file per view) for the 3D display;
- resampled and compressed slices (two files per slice) for the slice displays.

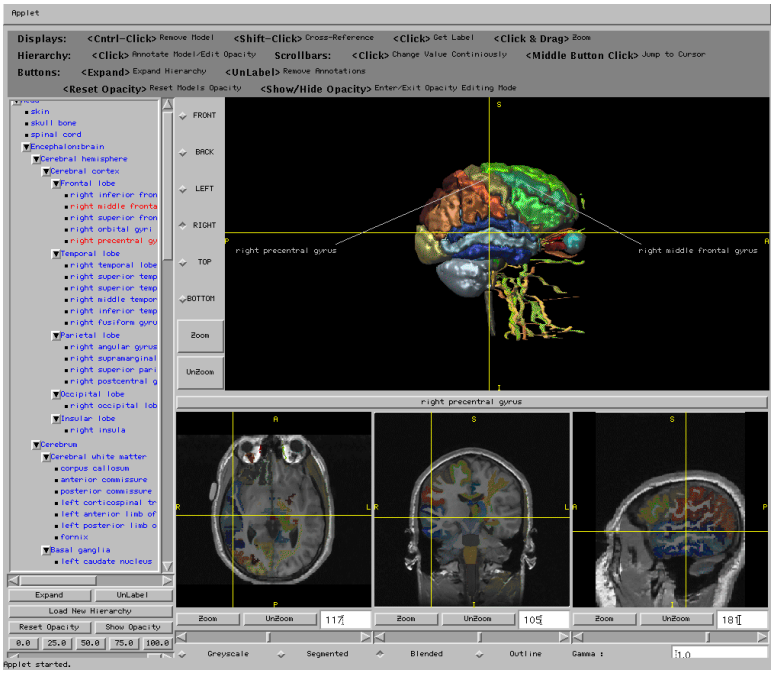
These files are used by the user-end interface at display time. The next section describes the interface components in detail.

4 AnatomyBrowser User-end Interface

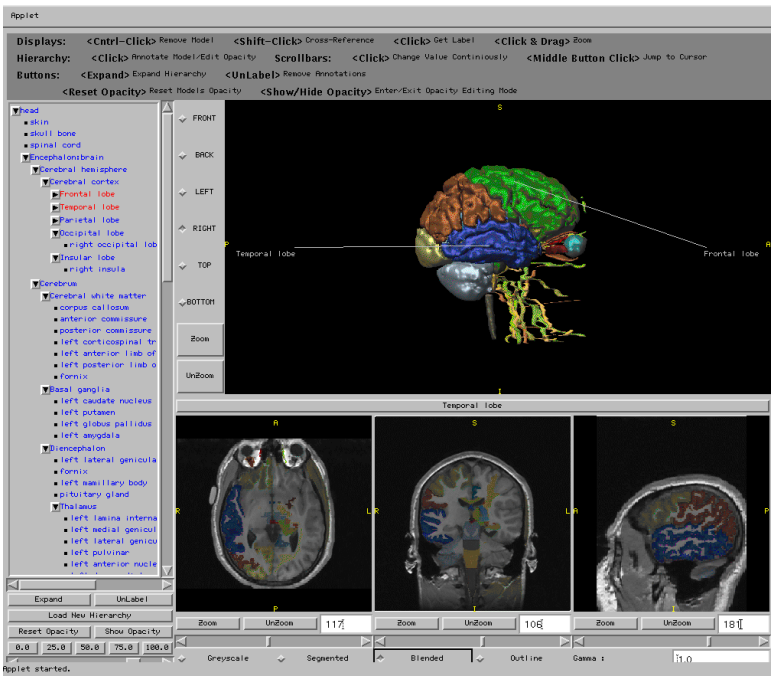
AnatomyBrowser integrates three main types of information: 3D surface models, slice data sets and text. Accordingly, the user-end interface consists of three main components: a 3D display, three slice displays and a hierarchy panel (Fig.4a). In addition to visualization capabilities, AnatomyBrowser provides cross-referencing among all types of displayed information. The user-end interface is implemented as a Java applet, and is therefore platform independent.

4.1 Interface Components

3D Display The 3D display uses multi-layer images to generate and display images of the 3D models using Eq. (3)-(4). It provides the user with controls for view selection, zoom in/out, and manipulation of surface colors and opacities.

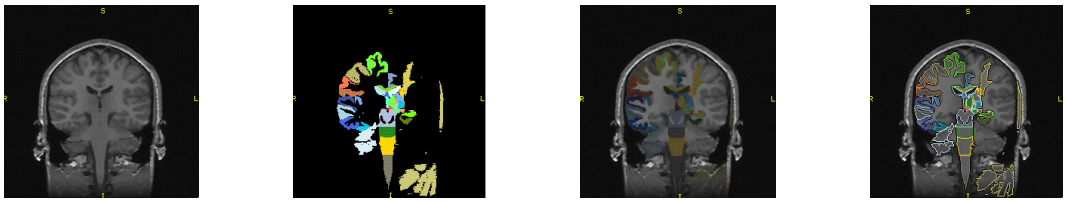


(a) Interface example



(b) Brain data set with coarse hierarchy

Figure 4: AnatomyBrowser user-end interface. (a) An example of the interface for the brain data set. The cross-hair positions correspond to the same 3D point. Some of the structures are annotated on the 3D display. The sagittal slice is slightly zoomed in. (b) AnatomyBrowser for the brain data set with the frontal, temporal and parietal lobes collapsed into single nodes. Compare to (a) for the differences in the hierarchy panel, the colors of the structures and annotation labels.



(a) Grayscale (b) Segmented (c) Blended (d) Outline

Figure 5: Slice display. The same coronal slice from the brain data set is displayed using four different ways of combining grayscale and segmented images.

Cross-Sectional Slices Axial, coronal and sagittal cross-sections are displayed on the three slice displays. The cross-sectional slices are computed from the original slice volume by resampling it along the three orthogonal directions. Similarly to the 3D display, zoom in/out functionality is implemented for the slice displays. AnatomyBrowser provides four different ways to view the slices. A user can choose to view just the grayscale images, or just the segmented set, or blend them together, or view the grayscale images with the outlines of the segmentation. Fig. 5 demonstrates all four choices for one slice from the brain atlas data set.

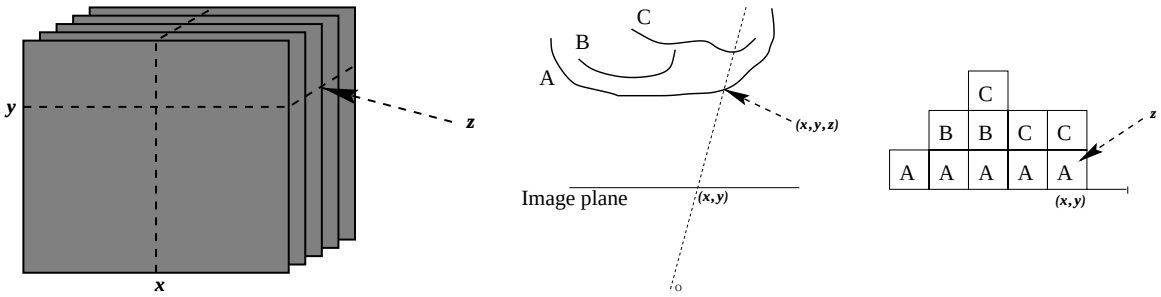
Hierarchy Panel A hierarchy tree (graph) is a natural, systematic way to classify and present anatomical structures. AnatomyBrowser uses a directed acyclic graph (DAG) to model anatomical hierarchy. The hierarchy file generated by the back-end system is parsed by the interface and a corresponding graph structure is constructed. However, we use a tree model to display the structure hierarchy, as it is more suitable than a DAG for visualization purposes. If a particular structure (or a group of structures) is shared by several parents, several instances of the corresponding node (or a subgraph) are generated and displayed under each of the parents. Since the internal representation uses a DAG, and all the instances of a group or a structure point back to the same element in the representation, the element properties can be changed using any of the instances.

The hierarchy panel allows the user to set a desirable level of detail for displayed images. Any subtree in the hierarchy panel can be collapsed into a single node. This causes all the structures in the sub-tree to be re-rendered using the color of the new collapsed node and to be treated as one structure from that point forward (Fig. 4b). This feature can be useful for studying anatomy with the help of AnatomyBrowser. A user can start with a very coarse hierarchy tree and expand group nodes gradually, rather than try to learn the whole tree at once. It can also be used to change properties of all elements in a subtree in one step. For example, if we wish to remove all muscles from the knee atlas in Fig. 7a, collapsing the muscle group and removing it as one element is easier than removing the muscles one by one.

4.2 Cross-Referencing Between Displays

An important capability provided by AnatomyBrowser is cross-referencing among the displays. If a user clicks on one of the display areas (<*Shift-Click*>), a cross-hair appears on all four displays at locations corresponding to the same 3D point.

Although all four displays exhibit very similar behavior, the cross-referencing definition and implementation is different for the 3D display and the slice displays. Any pixel in the slice display is characterized by three numbers: the slice index and the pixel coordinates in the slice (Fig. 6a). These three numbers are sufficient to compute the 3D coordinates of the corresponding physical point in the volume defined by the slice set. Therefore, the mapping between pixels in the slice



(a) Slice display coordinates (b) 3D display coordinates (c) Multi-layer image

Figure 6: Coordinate correspondence between the volumes and the physical world: (a) A point in the slice display defines a 3D point in the physical volume; (b) A point in the 3D display defines a view ray in the physical world; the depth value can be determined by identifying the intersection point of the view ray with the nearest surface; (c) The depth value can be looked up in the multi-layer image of the scene.

set and physical points in the 3D volume is well-defined and can be readily computed from the physical dimensions of the voxel. This is not true for the 3D display. Each pixel in the 3D display is described by only two coordinates, and a user has no means to specify nor to obtain depth values directly in the 3D display. A natural one-to-one correspondence between pixels in the 3D display and entities in the 3D space is the one from pixels to view rays and *vice versa* (Fig. 6b). This construction provides us with a useful interpretation of the cross-referencing for the 3D display.

If a cross-reference is initiated from the slice display, the corresponding 3D point can be uniquely computed. For the 3D display, we place the cross-hair on the pixel corresponding to the selected 3D point, but we might not see the point itself on the display. Instead, we will see a surface that occludes the selected 3D point. The cross-hair on the 3D display should be interpreted as a view ray along which the selected 3D point lies.

In a similar way, by clicking on the 3D display, a user specifies a view ray rather than a single 3D point. AnatomyBrowser uses depths information available to it from the multi-layer image to disambiguate this point referenced by the ray. The assumption used by the system is that the point of interest lies on the intersection of the view ray with the closest visible surface (Fig. 6b,c). This decision is justified by a natural notion of pointing: when interpreting somebody's pointing, an observer follows an imaginary ray defined by the pointer until it meets an object that is not fully transparent.

4.3 Text & Annotation

AnatomyBrowser also provides annotation capabilities, which can be considered as cross-referencing between text and images. This feature can be invoked from any of the main components of the AnatomyBrowser interface.

To use the hierarchy panel for annotation of a particular structure, a user clicks on the name of the structure on the panel. If the structure is visible on the 3D display, a pointer to the structure will appear in the 3D image. The name of the structure will be displayed on the title bar between the 3D display and the slice displays. The implementation of this feature is fairly straightforward: a multi-layer image contains information on the structures currently visible on the 3D display.

If a user clicks on any of the displays, the structure index is extracted from the corresponding segmented slice for the slice displays, or from the multi-layer image for the 3D display. Again, the

name of the structure is located in the hierarchy graph and a pointer with the structure name is displayed on the 3D image.

As mentioned before, AnatomyBrowser allows text notes to be attached to a particular structure. The text notes are saved in a file with the structure name, which is used by the interface to locate the notes. We use a title bar with the name of the structure as a 'hypertext' link to a collection of text notes. Clicking on it brings up a text window with all the text information available on the structure.

Both annotation and text documentation can be invoked for all leaves in the hierarchy panel, whether they represent real structures or groups of structures that have been collapsed into a single node.

5 Example Applications

AnatomyBrowser provides a framework for visualization of medical images and integrating them with other information. Anatomy studies are the most obvious application for such a system. AnatomyBrowser can effectively replace a conventional anatomical atlas book, as it provides at least as much information as the book. Students can see anatomical structures, view an MR or CT scan of those structures, read an attached description and study the hierarchy of the structures. Visualization and full cross-referencing make AnatomyBrowser a useful interactive teaching tool. And the fact that the user-end interface can run essentially on any computer makes AnatomyBrowser particularly well suited for this purpose.

We have generated interactive atlases using AnatomyBrowser for brain (Fig. 4), knee (Fig. 7a), abdomen, inner ear and other cases. Building digital atlases is an ongoing effort in our group, and they become available on-line [1] as new data sets are segmented. All the atlas cases are based on scans of individual healthy subjects, but the system can potentially be used on phantom data, as well as on atlases derived from a group of subjects. Most of the cases are based on a single segmented scan, but some examples were generated by merging several different modalities using registration. These include the brain atlas that was generated from two different MRI scans that covered overlapping, but different, parts of the subject head and were merged together using rigid registration techniques, and the inner ear atlas that was generated by merging bone segmentation from a CT scan and vessel segmentation from an MR angiogram. Once on the web-site, the brain atlas has been extensively used as a reference tool by the researchers in our laboratories.

In addition to anatomy studies, digital atlases can also be used for model driven segmentation. We can view the process of model-based segmentation as a loop, in which the results of the segmentation of the previous scans are used in segmentation of the new data sets. And the digital atlas is an intermediate step in this loop, where knowledge about anatomy is accumulated in a particular representation, visualized for the user, and possibly processed for future use by the segmentation algorithm. In a way, its purpose is to "close the loop" from one stage of the segmentation to the next one, which uses the result of the first stage. AnatomyBrowser can be helpful for visualization both of the atlas and of the intermediate results in this process.

As we pointed out in the introduction, AnatomyBrowser is different from a digital atlas. It is a visualization system that can be used on any scan, not necessarily the reference (atlas) data. This naturally brings up another application for AnatomyBrowser. It can be employed as a visualization and annotation tool for clinical cases, when the analysis must be performed on a per-case basis. We have used AnatomyBrowser for visualization of several tumor cases in different anatomical areas: pelvis (Fig. 7b), sacrum, brain, and kidney. These cases are also available on-line [1].

The tool was tested for both surgical planning and as a reference tool during surgery. Before the

surgery, an MRI (or/and CT) scan of a patient was taken and segmented in the Surgical Planning Laboratory at Brigham and Women's Hospital. Then the 3D surface models were generated and presented to the surgeons in the back-end viewer. The surgeons used the viewer to familiarize themselves with the 3D shape and location of a tumor and structures around it and to plan the surgery. They were also asked to select about a dozen views that would be useful during surgery. During surgery, the user-end interface with the selected views and the slice data was made available to the surgeons for general reference on a personal computer (a laptop) in the OR.

The surgeons' response to the system was positive in all cases. It provided them with much greater visualization capabilities than they had had before, while establishing clear correspondences between the 3D models and the more commonly used cross-sectional slice sets. We would like to mention two cases in which AnatomyBrowser allowed the surgeons to significantly improve their performance. In case involving a sacral tumor, the visualization system made it obvious that certain vertebrae were covered by the tumor on the right side, but not on the left side of the sacrum, which was of a great help in planning the surgery. More information on this case can be found in [12]. In the case of a pelvic tumor, an initial surgery plan was changed after the analysis of the 3D models using AnatomyBrowser had demonstrated that the originally planned cuts would not be feasible. Then in the OR, the surgeons consulted the user-end interface to establish correspondence between different structures on the slices and on the 3D display. The surgeons reported that the information on spatial relationships between different structures provided by AnatomyBrowser allowed them to substantially reduce the time of the surgery.

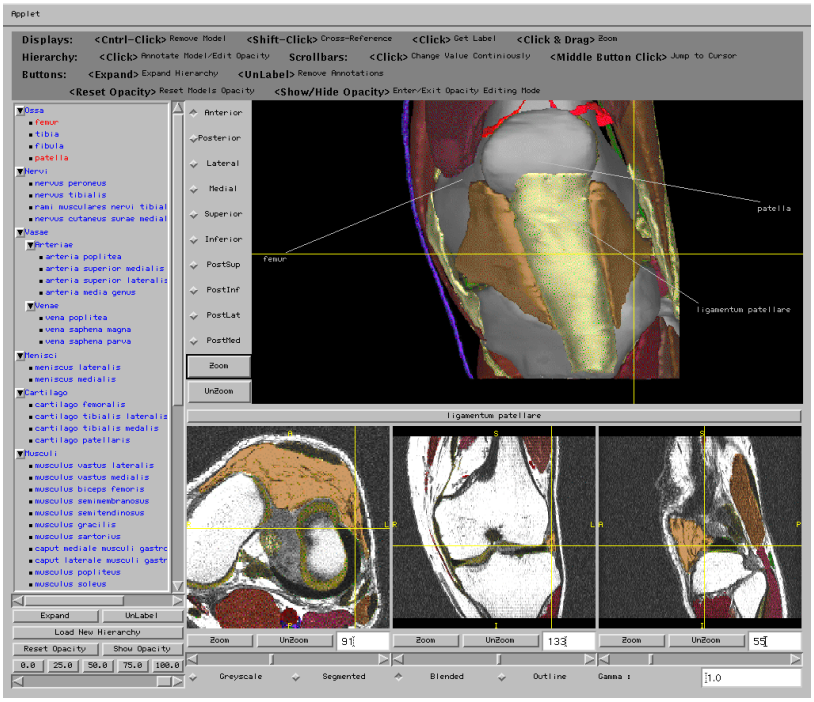
Although AnatomyBrowser cannot provide as much functionality as the specialized surgical systems [5, 14], it can definitely be useful for surgeons that do not have access to high computational resources and the specialized surgical systems. In particular, AnatomyBrowser can be considered a testing tool, allowing the surgeons to assess the value of 3D image visualization before committing to a certain surgical system.

Another interesting (future) application where AnatomyBrowser can be of help is remote data processing. Consider an imaging center that receives medical scans from other hospitals or imaging facilities that are not equipped to perform segmentation and visualization on-site. The scans are processed in the imaging center and are sent back to the users, who can them employ AnatomyBrowser interface on a workstation or a personal computer to visualize the data. This model of medical image processing follows the same principle of uneven distribution of computational resources between the processing site and the visualization site that was used in AnatomyBrowser design and implementation. Thus it can naturally provide visualization services in such a scenario.

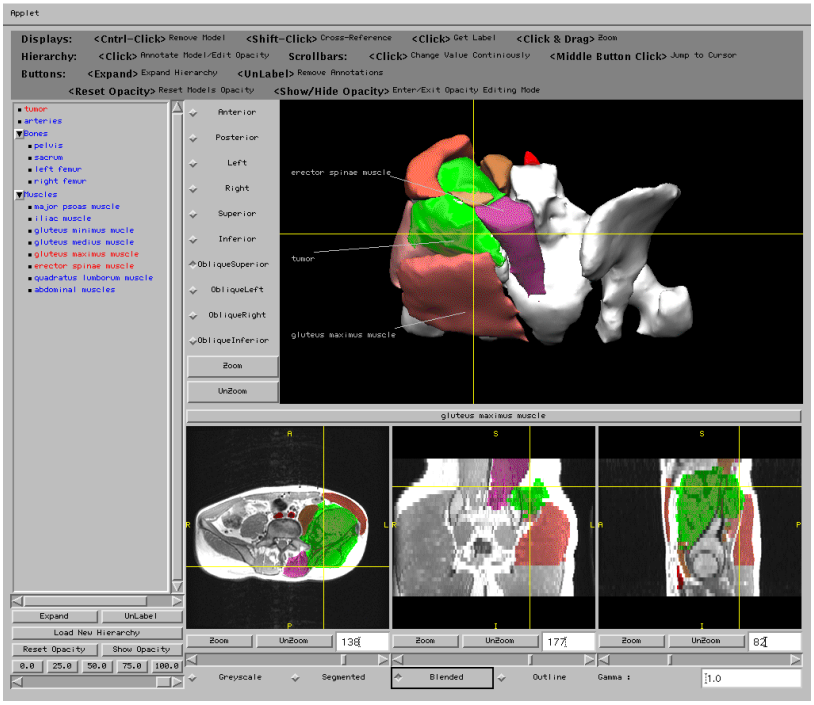
6 Current Implementation Evaluation

In this section, we report statistics collected on 12 cases that were processed and visualized using AnatomyBrowser. They range from tumor cases, with a small number of structures to atlas cases with over 100 structures. We note that the current system was implemented as a proof of concept for the new visualization approach we presented in the paper. It can be improved to decrease significantly the size of the intermediate data used by AnatomyBrowser, as well as the running time of all the components.

Table 1 compares sizes of slice data sets generated in the pre-processing step and of the original data sets. In most cases, the output size is comparable to the input size, but for some cases (for example, case 5) the size of the output is significantly higher than the size of the input data set. Note that these are statistics reported for the implementation that resamples a slice volume off-line and stores all resampled slices. In this implementation, if an aspect ratio between slice thickness



(a) Knee atlas



(a) Pelvic tumor

Figure 7: Additional examples of AnatomyBrowser. Both examples demonstrate cross-reference and label correspondence. (a) Atlas example: knee atlas. (b) Clinical case example: pelvic tumor.

Case number	1	2	3	4	5	6	7	8	9	10	11	12
Input size(MB)	30.7	52.5	10.0	40.0	8.7	12.7	13.7	70.0	30.0	28.5	30.7	40.0
New size(MB)	25.3	40.9	10.2	8.2	20.9	10.7	24.1	23.3	41.3	20.3	18.5	21.8

Table 1: Comparison between the original data set size and the size of the data set produced by the back-end component for 12 different cases.

Case number	1	2	3	4	5	6	7	8	9	10	11	12
Num. of models	6	8	10	13	14	15	29	45	48	56	120	120
Image size(MB)	0.22	0.07	0.19	0.09	0.38	0.26	0.18	0.09	0.74	0.60	0.55	0.44

Table 2: Number of surface models (anatomical structures) and an average size of multi-layer images generated by the pre-rendering component for 12 different cases.

and slice resolution is high, the number of slices in the output data set is significantly higher than the number of slices in the input data set. In the future, when it becomes feasible to resample slices on-line, the intermediate data sets will be significantly smaller than input data sets.

Table 2 reports an average size of a multi-layer image generated and stored by the back-end system. We can see that there is some correlation between the number of models and a size of a multi-layer image, but it is obvious this is not the only factor affecting the size of the images. Another important parameter (that is difficult to quantify) is "density" of a scene, i.e., an average number of models per pixel in the image. But since the size of the multi-layer images is two orders of magnitude smaller than the size of the slice data set, it is clearly not a bottleneck in the system.

An amount of time required by the back-end system to process the data varies significantly depending on the number, size and complexity of surface models. Table 3 summarizes our experience with the system. In general, it takes under two hours to process the data on Sun Ultra Sparc10 workstation with graphics acceleration. Most of the time is taken up by rendering and file I/O. While the processing time for the slice data sets does not change as the number of models increases, the time for surface mode generation and model rendering has a high correlation with the number of models in the system.

In the process of system development and use, several different modes of running the user-end interface emerged. The first and the simplest one is when the data files and the user-interface reside in the same file system, and the files are accessed through the file system. In this case, the update speed is sufficient for most applications. Once the applet is loaded, the interface reacts to user commands almost instantaneously. Later in the project, we put a few example cases on our web-site, and users started accessing the system through an http connection. The system performance dropped, but was still acceptable if the access was through the local network (within the same domain). However, it proved to be too slow for remote access from other sites. Analysis of the data size above provides an important insight into this problem: an amount of data than needs to be transfered over the network makes remote use difficult. That is why we encourage our users to obtain a local copy of the data and the interface.

Improving the system performance for remote access is an exciting and promising direction for future work for this project. The current implementation, however, is not suited for remote use, as it is not optimized with respect to the amount of transmitted data, which is known to be a bottleneck for many distributed applications.

Component	Surface model generation	Model pre-rendering	Slice pre-processing
Clinical cases	5-10min	10-15min	2-3min
Atlas cases	30-60min	20-30min	2-3min

Table 3: Running time for different back-end components reported separately for clinical cases (fewer than 20 structures) and atlas cases (more than 20 structures).

7 Concluding Remarks

We have presented a novel framework for visualization of 3D models of anatomical structures. In this framework, the visualization process is divided between the back-end system that renders the images and saves them in a special format, and the user-end interface that reads pre-rendered images and displays 3D models, while providing a set of 3D scene manipulation capabilities similar to visualization packages based on true dynamic rendering.

The main advantage of this method is that the computational resources can be allocated unevenly in the system. In particular, the back-end components requires graphics acceleration hardware, but the user-end interface does not need any special hardware or software support.

We demonstrated an implementation of this approach in a system called AnatomyBrowser. The user-end interface of AnatomyBrowser is implemented in Java and can run on any platform. The current implementation proved to be efficient if the data resided in the same file system with the interface. Remote access through an http connection does not yield sufficient update speed for the system to be used over the network. Future optimizations of the interface implementation can help reduce the amount of information transmitted over the network and thus make remote use of the system feasible.

We demonstrated several example applications for AnatomyBrowser as a visualization system. They included studies of anatomy, or visualization of anatomical atlases in general, as well as visualization of clinical cases for medical research or surgery.

References

[1] AnatomyBrowser URL.
http://splweb.bwh.harvard.edu:8000/pages/papers/browser_update/index.html,
Mirror site: http://www.ai.mit.edu/projects/anatomy_browser.

[2] Blinn JF. Jim Blinn's corner: Compositing, part I. *IEEE Computer Graphics and Applications*, 14(5):83-87, 1994.

[3] Blinn JF. Jim Blinn's corner: Compositing, part II. *IEEE Computer Graphics and Applications*, 14(6):78-82, 1994.

[4] Foley JD, Van Dam A, Feiner SK, Hughes JF. Computer Graphics: Principles and Practice. 2nd edition, Addison-Wesley, 1990.

[5] Grimson WEL, Leventon ME, Ettinger GJ, Chabrerie A, Ozlen F, Nakajima S, Atsumi H, Kikinis R and Black PML. Clinical Experience with a High Precision Image-guided Neurosurgery System, *In Proc. MICCAI'98: The First Int. Conf. Medical Image Computing and Computer Assisted Intervention*, Lecture Notes in Computer Science Series, 1496:64-73, 1998.

[6] Höhne KH, Bomans M, Riemer M, Schubert R, Tiede U and Lierse W. A 3D anatomical atlas based on a volume model. *IEEE Computer Graphics Applications*, 2:72–78, 1992.

[7] Kikinis R, Shenton ME, Iosifescu DV, McCarley RW, Saiviroonporn P, Hokama HH, Robatino A, Metcalf D, Wible CG, Portas CM, Donnino R and Jolesz FA. A Digital Brain Atlas for Surgical Planning, Model Driven Segmentation and Teaching. *IEEE Trans. on Visualization and Computer Graphics*, 2(3):232-241, 1996.

[8] Lorensen WE and Cline HE. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21:163–169, 1987.

[9] Maintz JBA and Viergever MA. A survey of medical image registration. *Medical Image Analysis*, 2(1):1-36, 1998.

[10] McInerney T and Tersopoulos D. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91-108, 1996.

[11] Nowinski WL, Fang A, Nguyen BT, Raphel JK, Jagannathan L, Raghaven R, Bryan RN and Miller GA. Multiple Brain Atlas Database and Atlas-Based Neuroimaging System *Computer Aided Surgery*, 2(1):42-66, 1997.

[12] Richolt JA, Case of the month report on the sacral tumor case. http://splweb.bwh.harvard.edu:8000/pages/comonth/97/may2/com.html, Surgical Planning Lab, Brigham and Women's Hospital, Boston, MA, 1997.

[13] Schroeder W, Martin K, and Lorensen WE. The Visualization Toolkit : an Object-Oriented Approach to 3D Graphics. Prentice-Hall, NJ, 1996.

[14] Serra L, Nowinski WL, Poston T, Chua BC, Ng H, Lee CM, Chua GG and Pillay PK. The Brain Bench: virtual tools for neurosurgery. *Medical Image Analysis*, 1(4):317-329, 1997.

[15] Umans C, Halle M, and Kikinis R. Multilayer Images for Interactive 3D Visualization on the World Wide Web. http://splweb.bwh.harvard.edu:8000/pages/papers/mli/paper.html, *Technical Report #51*, Surgical Planning Lab, Brigham and Women's Hospital, Boston, MA, 1997.