


BIBLIOGRAPHY


Figure C.3: The Viz with SPL Picasso Window Open
Figure C.2: An SPL Viz Rendering of Semi-Transparent Skin with Vessels Inside
Appendix C

Screen Shots

Figure C.1: The Viz in Action
Figure B.4: SPL Picasso Pop-Up Window
Figure B.3: Slice Viewer Controls
Figure B.2: Manipulator Panel and Top Control Panel
Appendix B

SPL Viz Controls

Figure B.1: User Interfaces in SPL Viz
5 Cochlea 153 230 153
29 Vestibule 153 230 128
19 Saccule 76 128 51
27 Utricle 51 204 76
6 Endolymphatic_Sac 0 179 0

[Bone] 255 255 240
25 Temporal_Bone 255 255 238
14 Mandible 255 255 204
22 Styloid_Process 179 204 204

[Vessels] 255 120 120
12 Internal_Jugular_Vein 0 51 204
20 Sigmoid_Sinus 0 128 255
24 Superior_Petrosal_Sinus 0 179 230
11 Internal_Carotid_Artery 179 0 0
1 Anterior_Cerebral_Artery 204 102 25
15 Middle_Cerebral_Artery 179 51 51
17 Posterior_Cerebral_Artery 153 0 0
where ? tells us that the element is an internal node, and index tells us that this is a leaf. The index field is used for reference among all the structures/images. when the labelmap images are encoded, we use the same index for the structure as in the hierarchy file.

Given below is an example hierarchy file containing the data for the model of an ear –

```
[Ear] 163 148 128
  ? Middle_Ear
  ? Inner_Ear

[Middle_Ear] 181 255 0
  26 Tympanic_Membrane 0 255 179
  16 Middle_Ear_Cavity 255 255 255
  3 Auditory_Tube 179 153 255
  9 Incus 179 76 76
  13 Malleus 179 128 76
  21 Stapes 255 153 0

[Inner_Ear] 181 255 99
  7 Facial_Nerve 204 204 25
  8 Geniculate_Ganglion 255 255 153
  10 Internal_Acoustic_Meatus 255 204 51
```
3 Structure1 100 100 100
1 Structure2 255 0 0
? SubGroup1
? SubGroup2
[SubGroup1] 100 150 40
...
[SubGroup2] 200 60 0
...

Describes the following tree:

```
 +---- Structure3
       |
       +---- Structure4   +---- Structure1
       |                   |
       +---- Group1       +---- Structure2
                           |
                           +---- SubGroup1
                           |
                           +---- SubGroup2
```

Each internal node is specified by its name in square brackets followed by its color (RGB, 0-255). Each child of the node is specified by a line
BINARY

DATASET POLYDATA

POINTS 5031 float

<< Binary data. 5031 \times 3 floating point numbers. Each
triplet specifies a vertex >>

TRIANGLE_STRIP 2242 16660

<< Binary data. 16660 integers, specifying 2242 individual
triangle strips >>

POINT_DATA 5031

NORMALS normals float

<< Binary Data. 5031 \times 3 floating point number. Each
triplet specifies the normal to the corresponding vertex >>

The numbers 5031, 2242 and 16660 are only sample values and will vary
depending on the actual model. The \texttt{vtk} file in this format is a mix of binary and ascii
data.

A.3 .hr

The hierarchy file specifies the hierarchy as a directed graph (some nodes can have
multiple parents) by listing its internal nodes with their children. For example:

2 Structure3 80 20 200
4 Structure4 60 0 60
[Group1] 200 150 200
• The next line specifies either the url or relative path for the directory containing the image files (the .gif and .label files)

Some sample files are given below :-

@SPL models/ #vtk models store locally
http://edlab-www.cs.umass.edu/ aali # slices stored remotely

Another sample file :-

@SPL # i have all the data at umass
http://abbeyroad.ecs.umass.edu/thesis/models/brain/ # all the slices are available at mit

A.2 .vtk

The .vtk files are used in this project to specify the geometry of the 3D models. The complete vtk file format can be found on pages 416–425 of the Visualization Toolkit textbook [4].

We describe below the vtk format in which all our medical data has been generated. All our medical data files have the structure as the sample file below, with only the actual data values varying –

# vtk DataFile Version 1.0
vtk output
Appendix A

File Formats

This appendix describes the main file formats which are read by the application.

A.1 .spl

A .spl files just contains the location of the 3D model files (.vtk) and the images files (.GIF and .label).

The following rules specify the format –

- The parser ignores everything after a ‘#’ character in a line it also ignore all blank lines and whitespaces.
- The first line must begin with ‘@SPL’, the signature for .spl files.
- The second line specifies either the absolute url or the relative path for the directory containing the vtk data files and the ’all.model’ file.
- If it is a relative path, it is just appended to the url for the directory where the .spl file is located. So ‘../’ will not work. if it cannot create a valid URL object from the string specified, it will treat it as a relative path.
If asked to develop this application from scratch again, probably the only major change would be to do all the interfaces in Swing. Other avenues for improvement would be to use the more standard file-loaders which are becoming available now. There are many parts of the UI which could be redesigned to conserve space and make the interface simpler for the user.

In conclusion, we have an application which can be used as the foundation of a more comprehensive Java-based visualization system. It is a tool which will be used at the SPL, and it highlights the power and features of Java3D. The process of developing such a major application has been a very educative and interesting experience.
CHAPTER 6

Conclusion

We have developed the SPL Viz, an advanced medical visualization tool. The Viz offers doctors a cross-platform visualization solution with many unique features. This has been an experimental project dealing with nascent technologies which have their shortcomings. Java3D has proved to be a successful 3D graphics API for handling the need of interactive renderings for medical visualization. See figures in appendix C for some screen shots of the Viz in action. The developed software will be further refined with future releases of Java3D.

There were a number of features which were initially planned for the project but never were implemented because of lack of time. Some of the more prominent of those features which we would really have liked to seen implemented are –

- Cross-correlation between the 2D image slices, 3D models and the hierarchy tree.

- Generation of oblique cross-sections and difference images.

- Save and load bookmarks.

- Multi-user control.
Deering, one of the creators of Java3D. The availability of geometry compression would help in reducing network delays by up to a factor of 10 (the amount of compression typically achieved).

Other features that are not working correctly in the current release are the Billboard and 3D Text. These were required in creating a frame object for the model with 3D labels for the various viewpoints. However, currently, Text3D can only support one text string, and the rendering of the subsequent strings is incorrect as it seems to render all the letters in a word as if they were located at the same point.

5.5 Chapter Conclusion

This chapter has provided an account of our implementation of the SPL Viz. The various components of the application have been described and their development process explained. The SPL Viz provides a basic framework for developing more advanced Medical Visualization applications. The architecture of SPL Viz can easily be extended to include more features as well as adapted for non-medical applications as well. Java3D is currently an alpha technology and a lot of features are still awaited for further development. However, the successful development of our application does reflect the power and flexibility of the Java3D API.
If, on the other hand, a very large number of frames is used, it’s like the renderer is running continuously, and that doesn’t help either.

5.4.2 Picking

One of the desired interactions for this application was to be able to click and select a particular anatomical structure. This picking ability was required in order to be able to show the name of structures by clicking on them, and may also be able to remove them using other sequence of commands involving picking.

Though the application contains code right now to implement the feature of displaying the labels for various structures when you click on them, this feature has been currently disabled. The reason is that the picking routines provided in the implementation do not work correctly. They use a bounding sphere approximation which is inadequate because of the complex and convoluted geometric shapes of the anatomical strictures. Using a bounding sphere approximation yields incorrect results in the picking routines.

So, this set of picking enabled feature will also have to wait till Sun brings out a better implementation.

5.4.3 Missing Features

A number of important features which were thought to be an important advantage while using Java3D have not yet been implemented. The most notable amongst those has been the Geometry Compression classes and utility routines. Geometry Compression is a fascinating concept for minimizing the storage space required to store 3D geometry information [18]. The concept of Geometry Compression was developed by Michael
5.4.1 Renderer Synchronization

The Java3D renderer is very processor hungry and heavily consumes CPU cycles when it is running. This causes the rest of the user interface to respond very sluggishly and sometimes become too slow for practical purposes. The solution to this problem was to shut down the renderer when no change is being made to the scene graph.

The SPLCanvas3D class was developed which extended the standard Canvas3D class. This class was initially designed such that it would stop rendering after one frame unless the mouse was being dragged. Thus once the user stopped dragging the mouse it would stop rendering. Subsequently, if the user made any more changes to the model the application would just start the renderer again which would be stopped by the SPLCanvas3D object after one frame.

Unfortunately, this did not work. For, even though the renderer stops after one frame, the image on the screen does not get updated. It turned out that the renderer had to run for more than one frame for it to update the image. The number of frames required for the image to update increased with the size of the model being displayed. Currently there is no way to ensure that the renderer just runs for one frame and updates the image.

A way out, after some feedback from the Java3D mailing list, was to use a behavior which stopped the renderer after a specified number of frames. The application was modified to allow for the number of frames to be passed as a command-line argument.

Currently, this is just a dirty way to get the job done, and developments are awaited to remedy this situation. The performance right now can get very erratic if a low number of frames is used. The user inputs on the various control panels and the renderer can get out of synchronization leading to delayed response in the application.
certificates with the three main browsers, Netscape, Internet Explorer and HotJava. However, with the introduction of the new Security classes in Java 1.2, the process should become simpler.

Security restrictions however, are not the problem in running our application in a browser. It is the Java3D support which is still missing from the main browsers and it will be some time before that becomes available.

5.4 The Bleeding Edge of Technology

Perhaps the most interesting thing about this project has been the experience of working right at the very frontier of current technology. Both the JDK version and the Java3D implementation are not a final release. The advantage of working with such software is that you have access to the very latest features, which are not available otherwise. The disadvantage is that such alpha and beta releases are full of bugs and unimplemented features. You can run into unexpected problems which are difficult to figure out because of a lack of documentation and because it's so new there are no experienced users to ask for help. In this context the Java3D mailing list was a very useful forum for discussing the issues involving Java3D and asking questions from fellow developers.

A number of problems were encountered because of the early nature of the implementation, some of which are discussed below.
Figure 5.7: Pyramidal Step Artifacts with Low Crease Angle

The concept of crease angles must be borne in minds as there may be instances where the crease angle of \( \pi \) does not yield the right rendering.

5.3.2 Running as an Applet

This application can also be launched as an applet. As the application requires its own window to run, a button can be embedded in the web page to launch the application. However, the interesting issue is of applet security, which places severe restrictions. Normally an applet cannot read or write on the local disks. Also the applets are not allowed to connect to any other server than where they originated from.

However, browsers currently support features for providing greater access to trusted applets. For an applet to be trusted, it must have a digital certificate and the user must trust and allow the holder of that certificate access to the local file system. Currently there are three different sets of processes if you want to use digital
5.3 Other Features

5.3.1 Generating Normals

As discussed earlier, one of the steps taken to reduce the delay while loading the data over the network was to remove the normals information from the original *vtk* data files and then generate the normals on the fly.

The normals for the 3D geometry objects are generated by using the Java3D utility class `com.sun.j3d.utils.geometry.NormalGenerator`.

However, some care has to be taken while generating the normals. The normal generator uses a value called the crease angle to distinguish between sharp edges and smooth edges. If two or more triangles share the same coordinate index, then the normal generator needs to know whether it should generate just one normal to yield a smooth appearance, or to generate two separate normals and cause a *crease* in the appearance. The crease angle is used for this purpose. If the normals of the two triangles differ by more than the crease angle, the normal generator creates separate normals.

While generating the normals for the SPI models, the rendering did not look correct when the default crease angle of 44° was used. As illustrated in figure 5.7, the vessels in the model look like they have ‘pyramidal steps’ on them.

To avoid this artifact, the crease angle was increased, and as Java3D is optimized for crease angles of π and 0 radians, π was chosen (which corresponds to smooth shading). The resulting model renders exactly like the model with the pre-computed normals (see the right image in figure 5.7). Given the nature of anatomical structures, it makes sense to generate smooth shaded models, however, while using normal gener-
the following data is stored –

1. The current Transform3D containing information about the orientation and position of the model.

2. The BitSet containing information about what parts are visible and which are not.

3. The annotation text (it could be a comment or the name of a `.au` file).

4. The name of the bookmark.

All this data is stored in the ViewPoint object and is put in a hashtable data structure for quick access through the name of the bookmark.

Initially the aim was to develop another independent component for this control panel too, but seeing that the panel was needed to control a number of disparate features, this was not done. However, even though this part of the interface has not been developed as an independent component, there are some re-usable objects in the background. The class SPLAnimator is one such general class which can be used for animating a scene by interpolating between two Transform3D objects. It is basically a wrapper class for the RotPosPathInterpolator object which is used for interpolating the rotation and position of a given TransformGroup node.
5.2.6 Slice Viewer

The slice viewer is the interactive display for the 2D cross-sectional slices. These cross-sectional slices are generated a-priori for the three standard cross-sections - sagittal, axial and coronal. The significance of these grayscale images has already been explained in the second chapter. The user can select slices at various depths along a particular cross-section. Also, by cloning the slice viewer, the user can simultaneously view more than one slices.

Currently the slice viewer offers an option of the three standard cross-sections and the choice of grayscale and segmented images. Further work can be done here to include oblique cross-sections and blended images (blend the grayscale and segmented images together). Also, there is no image caching being done, which is another important way to improve performance.

The Slice Viewer can be used as an independent component for viewing the slices. It can be run from the command line using the command java SliceViewer path modelName, where the path is the URI of the directory containing the model slices and the modelName is the name of the model. It can also be used as an applet in web pages for providing access to slices. This is another example of how good object-oriented code can lead to the development of a lot of reusable software, and how this project has led to the creation of a lot of such reusable components.

5.2.7 Bookmarking and Multimedia Controller

The top panel of the display serves the role of controlling the features relating to bookmarking, animation and audio (Described earlier in section 4.5. The user can choose to bookmark a particular model orientation and state. For every bookmark,
by a suitable set of mouse movements directly on the model itself, the panel provides shortcuts for different orientations and reduces the user the amount of interaction required by the user. If the quality of a user interface can be judged by how few actions are required on part of the user to accomplish a particular task, then this manipulator panel significantly improves the quality of the application. Having such a panel is not merely a luxury item for the interface, but a necessity for the lay users who may find it very complicated to efficiently manipulate the 3D models directly. Also, when the frame rate becomes very low due to the large size of the model, it is very difficult to perform meaningful manipulations by dragging the mouse over the model.

This panel (like most of the other UI components) was initially developed using Symantec’s Visual Cafe. The images needed in the panel were created by first taking screen shots of the SPL Viz application to obtain an image of the 3D rendering of the head. Then some basic Photoshop work was done to reduce the images to the required size and draw the lines and arrows needed in them.

The left part of the panel, where the user clicks to go to pre-set standard viewpoints, was inspired by a similar control feature in the Slicer, at SPL. However, the rest of the panel was developed to make the rotations about the axes also as intuitive as going to pre-set viewpoints.

The ManipulatorPanel object only has access to the TransformGroup of the model which is being manipulated. Based on all the user inputs, the ManipulatorPanel class modifies the TransformGroup object. Thus, this is another useful component which can easily be used by other Java3D applications for manipulating models.
5.2.5 Manipulator Panel

The ManipulatorPanel class provides a very intuitive and easy to use interface for manipulation of 3D models. The main feature of this panel is that it provides some preset viewpoints where the user can view the model from. These are the anterior, posterior, superior, inferior, left and right views. See figure 5.6 for a screen shot of the panel in which the standard views are illustrated on the left part. The user clicks on the corresponding letters, A, P, S, I, L and R to go to those standard views. Note that even though the phrase being used is go to those views, its actually the model that is re-oriented rather than moving the camera. This distinction is only important from the technical implementation point of view and is transparent to the user.

![Figure 5.6: Screen Shot of the Manipulator Panel](image)

The user can also rotate the model about the three main axes by clicking on the corresponding ‘+’ and ‘−’ buttons on the panel. The scroller in the middle can be used for zooming in and out of the model.

While all the manipulations provided in this panel can also be accomplished
3D objects\(^4\). The original version, \texttt{JRembrandt} was a more powerful control panel as it allowed users to not only alter the RGB color and transparency values but also change the ambient, diffuse and specular coefficients as well as the specular power coefficient. However, as it occupied too much area and because all the functionality was not really required for ordinary users of the application, it was removed in favor of \texttt{JPicasso}.

Initially, there was also a separate Hierarchy Viewer window for viewing the hierarchy and manipulating the visibility of structures (i.e., switching them on or off). However, having two pop-up windows which had some overlapping functionality (selection of model parts), was not a very user-friendly design. This was then modified by merging the hierarchy viewer and \texttt{JPicasso} to provide \texttt{JAdjuster} which had the functionality of both. The pop-up window which appears by the name of SPL Picasso is actually the component \texttt{JAdjuster}. It now has four sliders on top for the manipulation of RGB and transparency values. The hierarchy is displayed as a tree in the panel on the bottom. Refer appendix B for details on how to use the interface.

This component has the reference to a \texttt{SPLModelData} object for the current model being displayed. Based on the user inputs, the various appearance objects are updated and the visibility is altered.

This component was designed entirely using the Java Foundation Classes, popularly known as Swing. As mentioned earlier, this panel was designed using Visual-Cafe2.5 (running JDK1.1.5) and the Swing code was then altered to the right package definition in JDK1.2beta3.

\(^4\)The prefix ‘J’ to the names adheres to the convention of naming all Swing based components with a ‘J’ prefixed to their name.
had the ability to set the Appearance object for the Shape3D node it returned.

**SPLModelLoader**

The next stage in development was the SPLModelLoader class. This class was initially developed to read in files of the ‘all.model’ format (refer to appendix A) This could read in the name of the various vtk data files located in the same directory as the ‘all.model’ file and also read their appearance attributes. It would then create an Appearance object using those attributes and call the Shape3DLoader object to read in the shape for each of the data files and set their Appearance object accordingly. The process of invoking the Shape3DLoader for all the separate data files is done in a separate thread so as to not hold up the main application while the model parts are being loaded up.

Later on another class, SPLParser was developed to parse ‘.spl’ files and the ‘.hr’ hierarchy files. The SPLModelLoader class then called the SPLParser class to generate the hierarchy tree and obtain the location of the ‘all.model’ file. All the information about the loaded model was passed to the main application, SPLViz, through the SPLModelData object

### 5.2.4 SPL Picasso

Picasso is the descendant of Rembrandt ... no, I am not discussing European Art, but JPicasso and JRembrandt were the names of two UI components that were developed using Swing and were used for manipulating the color and appearance of the
the size of the corresponding binary '.vtk' file.

An early alpha version of the VRML browser written in Java3D being developed at Sun was also used to view these data files. Interestingly, this test exposed a bug in the Java3D browser as it turned out the backface culling flag used in VRML2.0 was not being tested by the Java3D browser.

After all the experimentation with the file loader class, the current version of Shape3DLoader was developed which used the IndexedTriangleStripArray and also
as that was an even more efficient data structure for the geometry information. Also, an indexed array is necessary for generating normals and therefore changing the basic geometry structure served two purposes.

Shape3DLoader

The initial file loader class, Shape3DLoader, grew out of a test program to read in ‘.3dm’ data files, developed using one of the earliest pre-alpha releases of the Java3D implementation from Sun. The ‘.3dm’ format is a simple 3D graphics format for specifying geometry. The current version of the Shape3DLoader still has the ability to read in ‘.3dm’ objects even though it only supports a small subset of that format.

Using the basic structure of the Shape3DLoader class another class was developed to generate VRML2.0 data files from the vtk files. The motivation behind this class was to generate the corresponding VRML data files so that they could be viewed and manipulated using the VRML browsers available. Thus, it served to test the suitability of VRML for handling these kind of data sets, while at the same time, verifying if the Java3D code was being developed correctly, as at this point only the file loader had been developed and there was no application to do interactive manipulation of the Java3D objects being generated by the Shape3DLoader class. Screen shot of one such VRML file being viewed in CosmoPlayer2.0 is shown in figure 5.5.

As already discussed in section 2.2, VRML2.0 is not a solution for developing interactive medical visualization applications. In our case we found the performance of the 3D models on a 200MHz PentiumII (MMX) running CosmoPlayer2.0 in NT4.0, to be prohibitively slow. The ‘.wrl’ (VRML format) file generated was huge (as it had to be in ascii), and even after removing the normals data from it, it was more than twice
The `UniverseBuilder` object creates the `Universe` for this application and sets up the view parameters and lighting. The `StopRendererBehavior` is set up by `SPLViz\(^2\)` to shut off the renderer once the screen has been updated. The remaining scene graph constitutes the model. The model root is created by the `SPLModelLoader` and the behaviors for various mouse-drag movements are attached to it. Subsequently, the various shape nodes get added below the switch node, as they are loaded. Note that all the different `Shape3D` objects constituting the various anatomical structures are loaded in a separate thread so as to not hold up execution of the main program.

### 5.2.3 File Loaders

It’s chronologically appropriate to discuss the file loaders first as they were the first classes to be developed for this project. The application reads in a variety of different types of files, as described in section 4.3.2. The most important of these is obviously the `vtk` format in which the 3D model data is stored. As explained earlier, these 3D models are generated from the raw medical data obtained by CT scans, MRI and other techniques. The file format in which all the data is generated is actually a special case of the general `vtk` format, and is explained in detail in appendix A.

Initially a file loader was developed which generated a `TriangleStripArray`\(^3\) object, one of the basic constructs in Java3D for specifying geometry. As the `vtk` format of the original data also used triangle strips, it was easy to generate the `Shape3D` node using a `TriangleStripArray` to specify the geometry. Due to an error in the API documentation, `IndexedTriangleStripArray` was ignored in the initial development. However, later, the geometry object was changed to an `IndexedTriangleStripArray`\(^3\)

---

\(^2\)The class `SPLViz` is the main class of the SPL Viz application.

\(^3\)Please refer to the Java3D API documentation for details on all Java3D classes mentioned here.
UI components, it can easily be adapted to view, manipulate and animate other types of 3D models.

### 5.2.2 The Scene Graph

As explained earlier, Java3D is a scene-graph based API. The scene graph contains all the information that the Java3D renderer needs to generate the display for this application. The scene graph for this application is shown in figure 5.4.

![Scene Graph for SPL Visualizer](image)

**Figure 5.4: Scene Graph for SPL Viz**
access to the cross-sectional grayscale and segmented images associated with the model. These are pre-generated images for the sagittal, axial and coronal cross-sections and the user can clone the ‘slice viewer’ to view multiple such images simultaneously (see section 5.2.6 for details). The Viz also displays the hierarchical information associated with the model. The user can bookmark and annotate interesting viewpoints in the model which he can later re-visit. The annotations can be textual, or the user can specify an audio file (‘.au’ format) which is played back whenever he re-visits the bookmark. (Used in conjunction with Sun’s audio tool, it adds another dimension to the application whereby the user dictates and listens back to his annotation instead of typing and reading them). There is also provision for animating transitions while going from one viewpoint to another (the audio and animation features are introduced in section 4.5 and the implementation details are provided in 5.2.7). As the Viz is written purely in Java, it can be run over different platforms, and eventually, when the popular browsers start supporting Java3D, it can be launched as an applet from the SPL web page (see section 5.3.2 for these capabilities).

In Java3D terminology, the application loads the model data files and creates a scene graph which includes behaviors for mouse manipulation and picking (described in section 5.2.2). It also uses interpolators for animating transitions between viewpoints and sound nodes for audio annotation playback. It uses a switch node to allow removing and adding various anatomical structures from view. The appropriate capability bits (see section 3.4 for definition) are set to allows dynamic modification of the color and transparency attributes of the appearance objects for each structure.

Another important feature of the application is its generic nature. By substituting the vtk file loaders by other file loaders, and by removing some of the SPL specific
5.2.1 Meet the Viz

The SPL Viz (see figure 5.3) allows doctors and students to interactively browse 3D anatomical models. They can do 3D manipulation of the model by rotating, zooming and translating it.

These manipulations are performed by using mouse-drags on the model, as well as by clicking on the manipulation buttons on a control panel (described in section 5.2.4). The user can alter the appearance of individual structures by adjusting the RGB color values and the transparency value. Structures can also be switched on or off, i.e., they can be made to disappear and reappear from the screen. The Viz also provides
are used by one or more of the components shown in the figure.

The next section gives a brief introduction to the functionality that has been implemented in SPL Viz. That is followed by a description of the Java3D scene graph which is created in the application. The subsequent sections explain the main components of SPL Viz and their development in greater detail.
development cycle, but it was a necessity. As yet there are no visual environments for
JDK1.2beta3\footnote{Borland has recently released one with a pluggable JDK.} VisualCafe is very suitable for UI development with its powerful widget
set and Swing support, and thus proved to be a great help in generating all the uninter-

esting code for laying out the components in the right places. Some changes were
required later and then a great need was felt for the availability of visual development
environments on Solaris for advanced JDK versions.

\subsection{The SPL Viz}

This section describes the SPL Viz, its main components and their development pro-
cess. Appendix B contains snapshots of all the components with instructions on how
to use them.

The basic architecture of the SPL Viz is displayed in figure 5.2. It shows the
main components and how they interact with each other. Each component shown refers
to a class of that name and has a corresponding Java source code file. There are, of
course, many more classes in the project, but the ones presented in the figure provide
the basic functionality for SPL Viz.

The three main type of components in SPL Viz are those that manipulate the
scene graph, those involved in displaying 3D and 2D images and the file loader classes.
As shown in the figure, there is some overlap in the roles played by some of these
classes. The SPLCanvas3D class, which displays the 3D rendering of the scene graph,
is also involved in processing user input (as mouse drags) and altering the scene graph
based on the input. SliceViewerPanel also reads files which it displays as 2D slices.
All the other classes written for this project provide supplemental functionality and
used (Java3D 1.1 alpha3 and JDK1.2 beta3).

5.1.2 Development Platform and Setup

Most of the development took place on an UltraSparc 1 with Creator3D graphics accelerator installed. It was important to have an independent machine in order to keep the pre-alpha software and the medical data sets away from public access machines due to the proprietary and confidential nature. The Creator3D graphics accelerator gives much better graphics performance. The machine was installed in Knowles Engineering Building, Room# 304 with an IP address of abbeyroad.ecs.umass.edu. The initial development took place using JDK1.1.4 and the early-access Java3D pre-alpha implementation from Sun Microsystems. The first public alpha implementation of Java3D was released at the JavaOne conference in March 1998. The current development platform is Java3D alpha03 and JDK1.2 beta3.

All the final development was done by writing the code in the XEmacs (19.14) word editor and compiling the code using javac, the commandline Java compiler provided with the JDK. However, most of the initial UI code was generated using Symantec’s VisualCafe2.5 running on a PC. The Java code was then transferred to the UltraSparc along with the Symantec class files which were needed. This version of V Cafe also includes support for the Swing classes, though as it uses JDK1.1.x, the Swing package is included externally, and is not a part of the JDK itself, as in JDK1.2. So some of the components, which were designed in Swing in VisualCafe, required that the Swing package name be changed while compiling them in JDK1.2.

The whole process of designing the basic interface in Cafe and then transferring them for development on the Sparc did add a couple of additional steps in the
early February when the Sparc was finally installed. The initial efforts were aimed at building a basic viewer capable of reading \texttt{vtk} data files and displaying the models. Simple manipulation features were also added. Subsequently the more advanced capabilities were plugged in. These included network enabled dynamic loading, simpler model manipulation interface, hierarchy file viewer, cross-sectional slice viewer and the animation and sound features. Many of the features had to undergo revisions to improve the performance and simplify the interface.

5.1.1 Preparation

Most of the work done in this preliminary phase involved developing an understanding of the Java3D API and relating it to other paradigms for doing 3D graphics on the web. Sun Microsystems provided an early-access pre-alpha implementation of Java3D, which was tested by developing small programs. This phase also involved trips to the SPI in Boston to see the work being done in the area of medical visualization and the existing 3D visualization tools in action. The existing \texttt{vtk} software was studied and software was developed to convert the medical data sets into VRML2.0 format for visualizing them through VRML viewers.

The development of this application has followed a very experimental yet modular approach. Like any good object-oriented design, the application is very structured and one of its main advantages is that most of the code can easily be used for developing different applications with common features. The experimental nature of the application has arisen out of writing a lot of simple test code to implement various features and then refining and including it in the main application. This sort of experimental approach was necessary because of the pre-release nature of the software being
CHAPTER 5

Implementation

This chapter explains the development process and the various features of the application. It also describes the various trade-offs that were necessary to create the current version.

5.1 Project Time-line

The main stages over which the project evolved can be seen in figure 5.1. The project was initiated around October 1997 and it took till end of December to get things finalized and obtain the UltraSparc from Sun. The main development work started in

Figure 5.1: Project Timeline
critical views which highlight the pathologies in a case and its interesting feature, 
that case can subsequently be compared with other cases more easily.

Currently, the design approach used has been to playback the audio track as 
uniform non-localized background sound. Another interesting approach that was tried 
was to associate the audio with the a particular location in the model, so that whenever 
the user went close to a particular orientation, the audio should start playing. This kind 
of ‘Point sound’ is an interesting feature which is important while creating interactive 
virtual worlds, but this was not suited to our needs of clear audio which should only 
play, at the same intensity level at a particular orientation, and otherwise not play at 
all. (The ‘point sound’ feature was experimented with but its implementation did not 
prove very clear and consistent).

4.6 Chapter Conclusion

This section has introduced the main design features involved in the development of 
SPL Viz. The design specifications have been presented and the various requirements 
have been explained. The components of SPL Viz introduced here are explained in 
greater depth along with their implementation details in the next chapter. Various 
optimization features which did not make it to the implementation stage have also 
been discussed.

The main focus of the design has been to provide an application which meets the 
basic requirements of medical visualization, while providing an intuitive GUI, network 
support and an extensible architecture.
4.5 Multimedia Features

Another set of features which were included in the application provided the user with the ability to create animated presentations with audio capability. These features are basically an extension of the ‘bookmarking’ feature. The user can add the current state of the model in a list of bookmarks. The state of the model refers to the orientation of the model and the visibility information of the various parts of the model (i.e., which model parts were turned on and which are turned off).

With every bookmark, the user can also associate an annotation by either typing it in or by specifying an audio file. The text annotation will be displayed or the audio file will subsequently be played back whenever the user visits that particular bookmark. The audio can be combined with the looping playback feature to go continuously from one bookmarked viewpoint to the next while playing the audio track associated with each\(^1\). This can be used in a number of ways –

- For teaching. The bookmarks can be used to prepare a lecture which the students can watch. Also, the student can at any point stop it and manipulate and study the 3D model in greater detail.

- For recording medical observations. An audio record for a particular case can be prepared by a doctor and used by other doctors later for assessing the same case.

- Another interesting use would be to use the same set of bookmarks for different models, for example, for different brain-tumor cases. Then by bookmarking the

\(^1\)The actual audio file creation can easily be done simultaneously using the AudioTool provided on the Sparc platform.
tools. The main conceptual difference between the two is that while manipulating models, the position and orientation of the camera in the virtual world remains fixed while the model is moved and rotated. On the other, while navigating, as when playing a game like Doom or Quake, the camera flies through the virtual world.

Two set of user manipulation capabilities were provided –

1. Mouse drags on the 3D model to rotate, translate and zoom.

2. Mouse clicks on the manipulator panel to go to preset viewpoints and do incremental rotations about the three axes.

The mouse drag features are standard 3D manipulation controls (code facilitating these behaviors is part of the standard Java3D release distribution). The basic controls are –

- Left Mouse Button Drag – Rotate.
- Middle Mouse Button Drag – Zoom.
- Right Mouse Button Drag – Translate.

Note that using the middle and right button actually takes some of the platform independence away from the application, given that the Windows platform usually has a 2 button mouse while a Macintosh has a single button mouse. The use of the manipulator panel explained in section 5.2.5 mitigates the problem somewhat as a Windows user can use it for the zoom instead of the middle mouse button.

The bookmarking feature allowed the user to remember selected orientations so that they can be recalled with a single click of the mouse.
images, provided the user is requesting successive images, which is most often the case while visualizing these kind of medical grayscale files. Also, the reconstruction step increases the computational overhead at the client side, and may not be feasible for a simple application.

We performed some simple tests to see the amount of saving achieved by following this technique. Two typical images were opened up in Adobe Photoshop and the difference filter was used to generate the difference image (see figure 4.1) Both the original images and the difference image were GIF encoded. While the original images were both 35 K in size the difference image was only around 9K in size, a saving of almost 75% in file size. For transmission over low bandwidth networks, this could significantly reduce the delay involved. Unfortunately, this was one of the features which was not included in the final implementation of the project as it was felt necessary to have working prototype for high bandwidth intranets rather than optimizing it for low bandwidth internet.

4.4 User Interface Design

The user interface design has been a critical part of the whole process, as the target audience for the application is people in the medical profession who do not have much experience or patience with complicated 3D manipulation controls. Providing good 3D navigation tools is not an easy design task, and ease of manipulation goes a long way toward making the user comfortable in dealing with 3D simulations. (See, for example, the user interface for the VRML plugin CosmoPlayer 2.0 from SGI, which is significantly better and more intuitive that the CosmoPlayer1.0 version).

In our example, we needed 3D model manipulation tools rather than navigation
Figure 4.1: Calculating the Difference Image

respectively. We compute

\[
I_3(x, y) = I_1(x, y) \oplus I_2(x, y)
\] (4.1)

Where \(\oplus\) denotes the bitwise XOR operation, which is more efficient than taking the mathematical difference of the two numbers and also avoids the problem of having negative grayscale values.

Now, we compress image \(I_3\) just like we would compress any other grayscale image, and transmit images \(I_1\) and \(I_3\) instead of \(I_1\) and \(I_2\) as we would normally. The image \(I_2\) can be reconstructed at the client side by the using the following equation

\[
I_2(x, y) = I_1(x, y) \oplus I_3(x, y)
\] (4.2)

As \(I_1\) and \(I_2\) are very similar, the image \(I_3\) will contain a value of zero (or very close to it), for most of the pixels. Thus any reasonable compression scheme will yield a much smaller file size for \(I_3\) than it would for \(I_1\) or \(I_2\). Thus we need to transmit only one complete image and subsequently only need send the difference
saving may not be significant, but if the application is used over the internet, it can save many minutes of download time.

4.3.4 Compressing the Difference Image

The grayscale cross-sections are an important part of the visualization process as they are the primary images used by doctors for diagnostic purposes. Usually a doctor wants to look at a few successive cross-sectional images along a particular axis. The axis could be one corresponding to the standard cross-sections - Sagittal, Coronal or Axial, or it could be an oblique axis (any arbitrary axis passing through the model). The interesting thing about looking at any set of successive images along an axis are that they are very similar to each other, i.e., successive images differ in only very small regions. See figure 4.1 for an example of two successive images along the coronal cross-section. That means that the amount of new information being transmitted by sending a second image is not as much as by sending the first image. However, by transmitting the second image as a complete image, we are sending a file of the same size as the first image. We can exploit the fact that we need to transmit less information the second time around by sending the difference image instead of the second image. The difference image will basically have zero values for all the pixels in the image where both the first and second image have the same grayscale value. Thus the difference image will contain significantly less information and can be compressed to much smaller size.

To explain this concept, let’s assume that we represent each pixel in the first image by a single byte, containing a grayscale value between 0 and 255. So let $I_1(x, y)$ and $I_2(x, y)$ contain the grayscale value of pixel $(x, y)$ for the first and second images.
Of the files listed above the ‘all.model’, ‘.hr’ and the ‘.spl’ files are very small and are just loaded once during start-up and so they do not require any size reduction efforts. The ‘.label’ files are already run-length encoded, which is a simple and efficient compression scheme for that data.

Some data reduction effort is possible for the other two file formats and is explained below.

4.3.3 Generating Normals for vtk Models

The \texttt{vtk} binary data files contain the 3D geometry data in three sets of points.

1. A list of floating point coordinates for all the vertices in the model.

2. A list of all the triangle strips in the model.

3. A list of floating point normals for all the vertices.

While the first two are necessary to define the model, the normals can actually be generated from the vertex information. The alpha implementation of the Java3D release contains a utility class \texttt{com.sun.j3d.utils.geometry.NormalGenerator}. Using that class the normals can be generated on the fly instead of having to be saved in the \texttt{vtk} data file. Thus we can reduce the \texttt{vtk} data file size by removing the normals information from them. The trade-off is, of course, more computation time at the client side, but this additional computational delay is usually 5-10 seconds for the larger \texttt{vtk} files (like those containing skin and brain data, each around 1.5 M in size). This delay is much less than what might be incurred by loading that same data (around 500 kilobytes for the larger files) over the network. For high bandwidth intranets the
this application and their typical sizes are listed below. See appendix A for a detailed description of the format of these files.

**.vtk** 3D model binary data files in the **vtk** format. They contain the vertex and normals data for the individual parts of a model. Size can vary from a few kilobytes to a couple of megabytes. The number of these files vary from model to model, from just 5-6 files to around 200 files for the brain atlas.

**.hr** The hierarchy text files containing the data about the hierarchy of the model and the color information which can be correlated with the label files. Size is just a few kilobytes.

**.gif** The grayscale image data files for the cross-sectional slices. These are currently pre-generated files in GIF format. They are usually around 50 kilobytes in size. A model can typically have 250 such pre-generated files associated with each of the three views - Sagittal, Axial and Coronal.

**all.model** The all.model file and the .spl file are very small text files. The former contains a list of the individual **vtk** files constituting the different parts of the model, and the color associated with each part. All the **vtk** data files are assumed to be at the same directory location as the ‘all.model’ file. The ‘.spl’ just contains the location of the ‘all.model’ data files and the cross-sectional GIF files in the form of absolute or relative URLs.

**.label** The label data files containing the segmented images. There is one such file corresponding to every grayscale image. These files contain an integer index value for every pixel in the image. The file is stored as a run length encoded binary file. These files are around a kilobyte in size.
features to mitigate this start-up delay.

**Background loading**

If the user is already working on a model and decides to switch to another model later, he can start loading the other model and still continue working with the current model. The loading takes place in the background (in another thread). Thus the user does not have to sit idly and wait for the second model to load. This works best for the case when the user needs to work on two models, one of which is available on his local machine while the other is located remotely. In this case he can start working on the local model while loading the remote model simultaneously in the background.

**Incremental Loading**

The different parts constituting a model are loaded one by one and the user can start working as soon as the first part has been loaded. As the different parts are loaded they are attached to the scene graph and become visible. This means that if the model is set up such that one of the smaller parts is the first one to be loaded, the user can start manipulating the model almost instantaneously. This also reduces the perceived start-up delay significantly.

**4.3.2 Data File Sizes**

The first step in designing any application which transfers data over the network should be to reduce the size of the data to the absolute minimum necessary. This kind of maximal compression is, however, often not possible because of an unacceptable increase in the complexity of the application. The different kinds of data files that are loaded by
The existing file formats for the \texttt{vtk} model files, the `all.model` file, the `hr` hierarchy files and the `label` segmented image files.

The data models were generated by a sequence of steps as described in section 2.1.2.

### 4.3 Internet Features

The application was written so as to make use of remotely located data resources and make it easily and widely available to other users on different platforms. The application has two different target environments – high bandwidth (fiber-optic and/or 100baseT ether) intranets, as in SPL, and low bandwidth external access (over the web when browser support for Java3D arrives). The trade-offs between development time and low bandwidth features have been made in the favor of the former so as to have at least a working prototype for the intranet environment.

#### 4.3.1 Start-Up Delay

The application allows a user to load 3D data models from anywhere over the internet by specifying the URL. This works fine if the data file is located in network proximity where it can be downloaded at a high bandwidth. However, for loading over low bandwidth connections, the delay can be significant, as the amount of data involved can be of the order of a few megabytes.

It's expected that the user will generally allow for the start-up delay before starting work on the models. In case the user needs the same models frequently, he can download them on his local machine. However, the application does provide two
• Support for *bookmarking* viewpoints, i.e., the user can save a particular model orientation and state so as to come back to it later. He can also save comments or annotations for these bookmarked views.

• Support for animating transitions between view orientations. This is not only an interesting aesthetic feature but can be used for preparing animated presentations as well as while leaving the application in an idle ‘demo’ mode.

• Support for audio annotations. This exploits the audio capabilities of Java3D and adds another dimension to the application. Coupled with the animation feature this allows for powerful presentation development as well as for preparing animated lectures.

Each of the above design requirement is further explained in the next chapter along with the implementation details.

### 4.2 Available Software and Tools

The SPL has a number of applications which are used for viewing 3D models and related data files. The main design requirements for this application, already stated above, were a cross between the existing features of the Anatomy Browser Applet and the Slicer (see sections 2.3 and 2.3) The main shortcoming of the Anatomy Browser applet was that it did not provide true 3D manipulation, while the slicer was a platform specific application which could not be run across the web.

The available infrastructure at SPI was invaluable as the model data already existed and did not have to be generated afresh. The application was designed to read
• Have a simple and intuitive user interface. Though this seems like another obvious requirement, developing an easy-to-use interface is a non-trivial task. The main goal here was to make the model manipulation as simple and intuitive as possible while reducing the effort and time required on the part of the user to obtain a particular desired orientation.

• Reduce the perceived start-up and execution delay while running over the network. This is an important requirement while running the application over the internet. There are two separate issues here – reduce the initial start-up time and reduce the execution time delay. The first deals with starting the application and loading the first model. The second refers to delays while the model is being manipulated. This could be delay in getting the GIF files or in loading subsequent models. We present a number of ideas which help in reducing the perceived delays.

• The design should be easily extensible, i.e., it should later be possible to add further visualization features to the application. This condition basically implies that the program should be very modular and object-oriented. Fortunately, Java is a wonderful language for writing structured code and goes a long way towards facilitating good design. On the flip side, because of the alpha nature of the technology, a lot of the code was first written as experimental code and then later, in the interest of time, integrated with the main application with minimal changes.

Some extra features which were incorporated in the design and were not part of the original specification are -
uses the Visualization Toolkit for application and tools development.

- Allow the user to alter visibility, color and transparency of separate model parts. This is important for the user to be able to focus on particular parts of the models which are of interest.

- Allow the user to load models from anywhere on the internet. This is necessary in order for the application to run over a variety of platforms and over the web, as it will not always be possible for the user to have the model data on the local machine.

- Allow the user to change the orientation of the model. The main advantage of having 3D models is that they can be viewed from arbitrary orientations to gain a better understanding of the shape, relative sizes and spatial relationships.

- Allow manipulation of and cross-correlation with the hierarchy files associated with the models. The hierarchy files contain the data about the anatomical hierarchy and are necessary for understanding the relationship between the various structures while looking at the models. The hierarchy display can also be used by the user for selecting/reselecting model parts and selectively altering the color, transparency and visibility.

- Load and display the cross-sectional GIF files associated with the models. The cross-sectional gifs are the pre-generated 2D slices containing the grayscale and segmented images. The grayscale images are the images that are used by doctors while diagnosing patients, and having them in relation to the 3D model significantly enhances their accessibility.
CHAPTER 4

Project Design

The main goal of this project was to develop an application in Java3D that can be used by doctors and students for browsing 3D anatomical models. The guiding principles for the design were extensive 3D functionality, network awareness and making use of the existing visualization infrastructure at the Surgical Planning Laboratory. The other important design issue was having an extensible architecture, i.e., as the number of features that can be implemented in a semester’s time frame is limited, the design should be such that other features can easily be integrated later by other developers at the Surgical Planning Laboratory. The main challenge lay in developing the application in an alpha technology and dealing with the issues involved in having the application run over the network.

4.1 Design Specifications

The main design requirements for the application were:

- Read 3D model data in vtk format (see next chapter) as supported by SPL. All the models available are in this format as most of the existing software at SPL.
compiled. Having capability bits enables the renderer to make optimizations on scene graphs.

3.5 Chapter Conclusion

Sun Microsystems has been promoting Java not merely as another powerful programming language, but as a whole new paradigm for internet based computing. It is already widely popular and has seen tremendous development effort on part of Sun, leading to the development of APIs for almost every possible application area, including Java3D for 3D graphics.

This chapter introduced Java3D and explained some of its basic features. In the context of internet based platform independent applications, Java3D provides a powerful API for 3D graphics. The scene graph paradigm makes it easier for developers to build complex applications without worrying about the low level details. We explained some of the key features of Java3D which make it suitable for developing a medical visualization application. Some of the common Java3D terms were also explained. This chapter thus provided an insight into our main development tool.
Java 3D scene graph, change the behavior’s internal state—in general, perform any computation it wishes.

**Interpolators** These are a class of predefined behavior nodes used for interpolating between two extreme points. They can be used, for example, to vary the color of an object over time. Our application uses a RotPosPathInterpolator which is used for animation by interpolating the rotation and position components of the Transform3D object associated with a particular TransformGroup.

**Shape3D** The Shape3D leaf node object specifies all geometric objects. It contains two components: a reference to the shape’s Geometry and a reference to its Appearance component.

**Appearance** This node component object contains all the information about the color and material of a Shape3D object.

**Geometry** This node component object contains all the information about the shape of the object. It stores the individual vertices and normals in one of the geometry node component objects like a LineArray or IndexedTriangleStripArray.

**Switch** The Switch group node allows a Java 3D application to choose dynamically among a number of subgraphs.

**Renderer** The Java3D renderer is the actual rendering engine which traverses the scene graph and generates the output on the screen.

**Capability Bits** These are flags which must be set for any given node in order that some of that node’s properties can be read and/or written once that node has been
presently standard computer monitors would most often be used, more sophisticated
devices could be used, especially for applications like simulating surgery. Java3D would
be able to handle these different display devices without any significant effort on the
part of the application developer.

3.4 Java3D Terminology

This section explains some of the terms which come up in discussions in later chapters.
Most of these terms refer to particular types of nodes found in the Java3D scene graph.

VirtualUniverse The root node for a complete scene graph which is rendered by the
Java3D renderer.

BranchGroup A group node that is the root of a subgraph of a scene. It may be
compiled as a unit, attached to a virtual universe, or included as a child of a
group node in another subgraph. A subgraph, rooted by a BranchGroup node,
can be thought of as a compile unit.

TransformGroup A group node object which specifies a single spatial transformation
to orient, position and scale all of its children nodes. The transformation is stored
as a Transform3D node component object.

Behaviors Behavior nodes provide the means for animating objects, processing key-
board and mouse inputs, reacting to movement, and enabling and processing
pick events. Behavior nodes contain Java code and state variables. A Behavior
node’s Java code can interact with Java objects, change node values within a
3.3.3 Geometry Compression

Java3D has built in support for geometry compression, a scheme for reducing the amount of geometric data. The algorithm is based on Michael Deering’s paper in SIGGRAPH’95 [18]. Michael Deering, along with Henry Sowizral and Kevin Rushforth, developed the Java3D API. Although it is a lossy\(^3\) method, it reduces the memory requirement for specifying 3D geometry by up to an order of magnitude without a significant loss in object quality. Geometry compressed data also provides a compact format for transferring the models over the network. Having data in this format at the server would also reduce the startup time at the client – it would be faster to use built-in Java3D commands to extract Java3D geometry objects from the geometry compressed bit streams than to read binary data files through separate file loaders.

3.3.4 Dynamic Loading

A Java3D scene graph can be modified while it is being rendered. As Java is a multi-threaded language, a different thread can load the 3D objects over the network and keep attaching them to the scene graph as they become available. This allows the user to get started with less delay and to start work as soon as the first object becomes available.

3.3.5 Display Devices

Java3D provides support for different kinds of display device including normal computer screens, stereo screens, head mounted displays and room mounted displays. Although

\(^3\)The term *lossy* implies that some of the original geometry information is lost during compression.
that Java3D uses in rendering; specifically geometry, light, and sound. *Node component* objects are objects which are referred to by the node objects for specific properties, i.e., the Shape3D leaf node refers to the Geometry node component object to define its geometrical structure.

The scene graph can contain *behavior* and *interpolator* nodes for animation, morphing, user interaction and other features. Instead of specifying a camera as in traditional 3D graphics APIs, in Java3D the user specifies a View Platform object, which is more general in nature. Section 3.4 explains some of the common Java3D terms.

It is beyond the scope of this report to discuss all the Java3D Scene Graph objects; please refer to the Java3D API specification for a complete description.

### 3.3.2 Layered Architecture

Java3D implementations will be layered on lower level APIs like OpenGL and Direct3D. It can thus exploit the graphics acceleration available for these APIs. The speed of a cross-platform graphics application not only depends on the power of the client machine but also on the graphics acceleration available and the ability of the application to use it. As Java3D implementations will be able to use the native graphics acceleration on the different platforms, the application developer does not have to worry about making platform specific optimizations. Java3D thus relieves the developer from the chore of programming at a lower level.
A scene graph contains two main kinds of nodes, leaf nodes and group nodes. Group nodes serve to aggregate other nodes with common features (like same 3D transformations, location, behavior etc). The leaf node objects contain the elements stored in node objects which are organized together to form a directed acyclic graph. Figure 3.1 shows a simple scene graph containing one geometric object.

The scene graph specifies all the details that the renderer needs to display the scene – the geometric data, attribute information and the viewing setup. The application developer constructs a scene by attaching various node objects to his scene graph. As a more complicated example, see section 5.2.2 for a discussion on the scene graph for our application.
3.3 Java3D Concepts

The Java3D API was designed with the following goals in mind [15]:

- High Performance.
- Rich set of 3D features.
- High level, object-oriented paradigm.
- Wide variety of file-formats.

Java3D is a scene graph based API. A scene graph (explained below), provides a high level description of a 3D virtual world including all the information required to render it, i.e., display it on the screen. It improves on the previous APIs by allowing the programmer to focus on the scene, the geometric objects and their composition, rather than worrying about low-level details like how to best write the rendering code. This high level abstraction also allows Java3D to make its own optimizations and to be able to exploit the graphics acceleration available on a wide variety of different platforms.

Java3D also includes many new features, like spatialized sound, which are not traditionally considered part of graphics but are crucial for realistic immersive virtual experiences.

The following sections elaborate on some of the features of the API and explains how they are important for developing a visualization application.

3.3.1 The Scene Graph

A Java3D scene graph is a high level hierarchical description of a virtual world. The term scene graph comes from the fact that all the information about a virtual scene is
3.2.1 GUIs and Swing

Java provides a variety of Graphical User Interface (GUI) classes. It is very easy to use the built-in GUI components in Java and to customize and extend them. In the first release of the Java Development Kit (JDK version 1.0), Java provided the Abstract Windowing Toolkit (AWT), a set of classes for developing user-interfaces. The AWT provides basic implementation of common UI components like buttons, menus, scrollbars, labels and many others. Developers can use these classes to do rapid prototyping of applications.

Perhaps one of the major developments to the Java language has been the recent introduction of the Swing classes, which extend (but do not replace) the AWT. The Swing classes are part of the Java Foundation Classes in JDK1.2, and provide a very rich set of UI capabilities, including features like – lightweight components, pluggable look and feel, and new components like internal frames. A full discussion of Swing is beyond the scope of this report. The reader is referred to the Sun website for a comprehensive description.

In the context of our application, the Swing classes have some incompatibility with the current release of the Java3D implementation. So, even though Swing is perfect for developing the UI for a Java application, it could not be used for all the UI requirements of SPL Viz. Swing was used, however, to develop some of the pop-up windows in the application. This was possible because the incompatibilities with Swing arise if Swing and non-Swing components occupy the same window, and putting them in different windows avoids the problem.

With this brief look at the Java language, we are now ready to discuss Java3D and its many features.
Having an interpreter in between the compiled code and the machine’s native environment obviously has a performance penalty and is the main reason why Java is not yet popular for computation intensive applications like 3D computer games. Bridging the performance gap between Java and C/C++ is one of the key areas of development right now. Just-In-Time (JIT) Compilers are also being used to improve the performance. A JIT compiler compiles the byte code to native code during execution to obtain significant speed-up. Symantec has developed a JIT compiler which is distributed along with Sun’s JDK (Java Development Kit).

Java provides high-level support for networking, as the language was developed with the internet in mind. This is very useful for developing applications where data location is specified as a URL, and Java takes care of the lower level semantics of setting up IP connections and transferring data.

Perhaps the main advantage in using Java is its wide availability. The major internet browsers, Netscape and Internet Explorer, come with their own JVM, and are capable of running Java applets. Thus by developing the application in the form of a Java applet, it can automatically be used by millions of people using these internet browsers. There is no overhead involved in deploying the application at every possible site where it may be used.

Another major feature responsible for Java’s popularity is its support for Graphical User Interface (GUI) development. When Java was first released, it caught on like a wildfire because people could easily develop visually appealing interactive applets which resided on web-pages, and thus add life to their static HTML pages.
3.1.3 Java3D Time-line

The development of Java3D has been an open process, and the Java3D API specification version 1.0 was released on August 1, 1997. Sun provided early access of its API implementation to a select group of pre-alpha developers before making the alpha implementation public at the JavaOne conference in March 1998. At the time of writing, Java3D is still an alpha technology. The Java3D website currently features the Java3D1.1alpha3 specification as well as the same version of the implementation provided by Sun Microsystems. Java3D is supposed to provide the first beta version later this summer.

The next section gives a brief introduction to the Java programming language which is necessary to fully appreciate the features of Java3D.

3.2 Java

Java is a high level object oriented programming language. Since its inception only a few years back, it has taken the world of computing by storm. Perhaps its most revolutionary feature is the concept of platform independence. As today’s internet consists of different kinds of computers linked together it becomes important to develop applications which can run on all the various platforms. Java provides it through the concept of byte-codes and the Java Virtual Machine [17]. A Java program is first compiled by the Java compiler into byte-codes, which is a platform independent binary format for storing executable classes. The Java Virtual Machine (JVM) is the platform dependent interpreter which executes the byte-codes. By providing JVMs for different platforms, the platform independent Java byte-codes can be executed on all those
Table 3.2: JavaMedia APIs

<table>
<thead>
<tr>
<th>API Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java 2D</td>
<td>Provides an abstract imaging model that extends the JDK 1.0.2 AWT package, including line art, images, color, transforms, and compositing</td>
</tr>
<tr>
<td>Java Media Framework</td>
<td>Specifies a unified architecture, messaging protocol, and programming interface for media players, media capture, and conferencing; Comprises three separate APIs (Java Media Player, Java Media Capture, and Java Media Conference)</td>
</tr>
<tr>
<td>Java Collaboration</td>
<td>Allows for interactive, two-way, multi-party communications over a variety of networks</td>
</tr>
<tr>
<td>Java Telephony</td>
<td>Integrates telephones with computers and provides basic functionality for first-party and third-party call control</td>
</tr>
<tr>
<td>Java Speech</td>
<td>Provides Java based speech recognition and speech synthesis (text-to-speech)</td>
</tr>
<tr>
<td>Java Animation</td>
<td>Provides for motion and transformations of 2D objects while utilizing the Java Media Framework for synchronization, composition, and timing</td>
</tr>
<tr>
<td>Java 3D</td>
<td>Provides an abstract, interactive imaging model for behavior and control of 3D objects</td>
</tr>
</tbody>
</table>

for applications ranging from simple 3D logos on web pages to complex multi-user simulations with sound, movie textures and immersive displays.

The Java 3D specification is the result of a joint collaboration between Silicon Graphics, Inc., Intel Corporation, Apple Computer, Inc., and Sun Microsystems, Inc. All had advanced, retained mode APIs under active internal development, and were looking at developing a single, compatible, cross-platform API in Java.
3.1.2 The Internet and the Origin of Java3D

The emergence of Java3D should best be understood in the context of the Java-Media suite of APIs being developed at Sun Microsystems. With the emergence of internet technologies and the popularity of the Java language as an easy to use tool for developing web based applications, there has been increasing demand for integrating full multimedia capabilities in these applications. Earlier, the only way that could be done was to use browser plug-ins. The main problem with using plug-ins is that they are non-flexible, i.e., they are separately compiled, platform-specific applications which must be individually installed on each client platform. Also, though plug-ins serve the needs of content developers, they do not provide application developers with an easy to use standard API. What that means is that a VRML world creator can create his world using an application like CosmoWorlds and put it on the web for anybody with a VRML enabled browser to see. However, if an application developer wants to create a web based 3D modeling application with custom GUI and I/O features, he has to master a whole set of diverse technologies and do a complex balancing act to get them working together (As an example, a simple web-based audio application can require understanding RealAudioTM clients and servers, cgi, html, and then praying that it works on all platforms).

The JavaMedia suite has been designed to provide a rich set of multimedia features for the already popular Java programming language. The various APIs shown in table 3.2 cover the needs of application developers to include a diverse set of multimedia feature in Java applications and applets [16].

Java3D was created to provide developers with a high level API for producing stand-alone 3D graphics applications as well as web based 3D applets. It can be used
M&S Computing (later to become Intergraph). The hardware configuration was raster terminals attached to 32 bit minicomputers like Digital VAX, Prime 400, and Data General MV Series. PHIGS, 3D GKS and PEX emerged in the early 1980s as the API standards for the new generation of workstations from companies like Sun and Apollo (now part of HP). These API were based on display lists and were very successfully used.

OpenGL was the path breaking 3rd generation API released in the late 80s by SGI. It had advance support for immediate mode graphics and allowed the user very powerful low level programming capabilities along with features like texture mapping and complex lighting models. This was a really revolutionary API and companies like Alias Wavefront, SoftImage and many others came into existence to take advantage of this new advanced graphics API and the resultant hardware and systems that were developed around it.

The 90s have seen the emergence of a new generation of graphics APIs, all of which make use of Scene Graphs along with a host of other new ideas in graphics. Some of these new efforts are – Direct Model from HP, the OpenGL Scene Graph API and Optimizer toolkit from SGI, and the recently announced Fahrenheit project from SGI and Microsoft. However, with the explosive growth of the internet and the emergence of Java as the programming language of choice for cross platform applications, the most awaited fourth generation API was undoubtedly Java3D. Java3D combines the power of the Java platform along with the most advanced concepts in graphics to provide users with a simple yet very powerful paradigm for platform independent graphics.
key concepts which are necessary to understand the development of SPL Viz.

### 3.1.1 Growth of Graphics APIs

In a historical perspective, Java3D can be considered as a fourth generation graphics API [16] (See table 3.1^2).

The 1970s saw the emergence of the first generation of graphics API in the form of the SIGGRAPH standard called CORE. This period saw the emergence of new CAD (Computer Aided Design) based companies like Computervision, Applicon, Calma and

---

^2Table used with permission. Courtesy Ken Tallman, Sun Microsystems

---
CHAPTER 3

Java3D$^{TM}$

3.1 Introduction

Java3D has been developed by Sun Microsystems as a next generation 3D graphics API which has its foundation in the Java platform [15]. An API, or Application Programming Interface, is a set of available software routines which facilitate programming by providing ready to use functionality. In other words, it is software which you can directly use in your own application without worrying about how it was written. In the context of an object-oriented language like Java, the API constitutes a set of classes\(^1\) which the application developer can use.

The Java3D classes provide a broad range of functionality to 3D application developers. Complex interactive virtual worlds can be created with relative ease. Java3D not only meets all the requirements of traditional 3D graphics like lighting, camera, texture mapping and animation, but it also provides a host of new features like spatialized sound, geometry compression and support for different display devices.

This chapter introduces the main features of Java3D and explains some of the

---

\(^1\)A *Class* is an Object-Oriented programming concept, which refers to a piece of software which acts as an independent entity and interacts with other software through clearly defined interfaces.
Axial Cross-section A cross-section of the human body looking at it from the top.

Oblique Cross-section A cross section taken along any arbitrary axis passing through the body.

Segmentation The process whereby regions in a medical data set are identified as belonging to various anatomical structures.

Registration The process of transforming the segmented structures to new reference frames that best utilize the extracted information.

2.5 Chapter Conclusion

This chapter introduced the field of Medical Visualization and provided the context for our project. The 3D visualization pipeline used for generating models from raw medical data was explained and brief definitions of some relevant medical terms were provided. We also described some of the current research work in this area which has been relevant to the development of SPL Viz.
frame. Then, instead of viewing each volume by observing parallel slices as one image at a time, the 3D Slicer allows up to three planes to slice through the volumes simultaneously at any orientation. Any volume may be displayed on each plane, or two volumes could be displayed simultaneously such that one is semi-transparently overlaid on the other.

Surfaces of 3D structures, such as the skin, brain, blood vessels, ventricles, tumor, and functional areas, are extracted, colored, and displayed with varying opacity. Surfaces and slices are combined in one interactive, 3D graphics display to illustrate spatial relationships between tissue boundaries.

### 2.4 Medical Terminology

This chapter has introduced a number of medical terms, some of which might not have been clear from context. Here is brief explanation of some of the relevant terms.

- **CT** Computed Tomography – A technique for medical data acquisition using the attenuation of X-rays as they pass through the body.

- **MRI** Magnetic Resonance Imaging – Another medical data acquisition technique which uses the phenomenon of regional differences in proton concentrations and the physico-chemical environment of protons.

- **Coronal Cross-section** A cross-section of the human body looking at it as if you were standing right in front of it. See the left image in figure 2.1.

- **Sagittal Cross-section** A cross-section of the human body looking at it from the side. See the right image in figure 2.1.
Image Guided Neurosurgery

The SPL is involved in the development, testing and deployment of image guided surgery systems, which register pre-operative scans with the actual position of the patient in the operating room. Such registered scans can be used by the surgeon both to plan the procedure (e.g. what path to take, where critical structures lie) and to support navigation (e.g. the position of surgical instruments relative to the registered scans, proximity to critical structures, positions of key landmarks) [14].

An example of using a prototypical surgical navigation tool is to point a surgical instrument at a key landmark within the surgical cavity, and to then display the position of that landmark on the segmented medical scans. By using such tools, the neurosurgeon should be able to visualize the full anatomical environment within which he is currently working, even though only a small opening is made in the skull, and only a small portion of the patient’s anatomy is actually directly visible to him.

Slicer

The 3D Slicer assists doctors in surgical planning and navigation by providing an interactive means of exploring several different radiological data sets. Volumes of diagnostic, anatomical data are acquired prior to surgery using MRI or CT. Other techniques like MR Angiograms reveal blood vessels, whereas MRI exam or PET can be used to obtain functional or metabolic information. The functional information is needed for example, when a tumor is located near portions of the cortex that are vital for speech, vision, or motor coordination, then functional data guides the surgeon in avoiding damage to these critical areas.

These varied medical data sets are registered, or aligned, to one coordinate
hand, while the applet can be used over the web, it does not support full 3D manipulation and uses an image based approach. In this context, our application seeks to integrate the features of the above two applications by developing an interactive visualization tool with true 3D manipulation capabilities which is platform independent.
automatically or semi-automatically) and prepare 3D models for interactive visualization, 3D MR makes anatomical detail readily apparent, instead of being relegated to traditional diagnosis from cross-sectional images. Consequently, there is now a great deal of interest in developing a digital 3D atlas of the human body for the purposes of teaching, pre-operative planning, and for the basis of template driven segmentation leading to the automatic delineation of the same structures in new imaging data sets (so-called “segmentation by matching” or template driven segmentation, TDS). This work affords a unique opportunity to evaluate structures in the body in 3D using information derived from routine medical imaging scans.

The SPL, together with Division of Neurosciences at Harvard Medical School, has been engaged in making a detailed morphological brain atlas for a number of years [13]. They are now extending the work to other parts of human anatomy, namely the chest and abdomen, the inner ear, and the knee. Figure 2.4 shows the brain atlas being accessed through the SPI-NSL Anatomy Browser applet.

The goals for the development of the atlas are -

- Develop it as an educational tool.
- Use it for pre-surgical planning and reference.
- Apply the anatomy as a template for segmentation by matching.

The development of the atlas has led to the generation of 3D models of various anatomical structures. In particular, a very detailed hierarchical brain atlas has been developed. The atlas can be browsed by either the stand alone viewer developed using \texttt{vtk}, or through its WWW interface, the SPI-NSL Anatomy Browser applet [10]. The former provides full 3D manipulation features but is platform specific. On the other
dent. Java is the language of choice for developing web based applications. However, until the release of Java3D, it was not really possible to do powerful 3D graphics using pure Java. Some OpenGL bindings (see [12]) were available as well as a number of other libraries for 3D graphics by independent developers, but none as powerful and sophisticated as Java3D.

2.3 Visualization at S.P.L.

The Surgical Planning Laboratory (SPL), part of the MRI Division of the Department of Radiology at Harvard Medical School’s Brigham and Women’s Hospital, is a computer science-based, application-oriented image-processing laboratory that is securely established in a rich clinical environment. The main research of the SPL has been the development of post-processing methods for digital medical imaging data and to use these algorithms for clinical applications.

The SPL has been conducting pioneering research in 3D medical visualization and has developed a number of systems for integrating 3D visualization in pre-operative and intra-operative procedures, as well as for educational and instructional purposes.

Some of the relevant projects which use 3D visualization techniques are described here –

Anatomy Atlas

Current advances in imaging technology, such as Magnetic Resonance Imaging (MRI), yield isotropic data sets at high spatial resolutions and quality, in relatively small imaging times. Coupled with methods to quickly delineate structures of interest (either
2.2 Web Based Visualization

The internet has become the premiere mode for the dissemination of information. There has been a lot of work in bringing visualization applications to the web. Most approaches to web-based visualizations can be classified into client-based rendering systems (like VRML), server based rendering systems (see [8]) and some interesting variations in between (see [9]). Some image based approaches have also been used as a substitute for true 3D rendering (for example, Multi Layered Imaging [10]). Only client-based rendering systems are capable of true 3D interaction in real time. All other approaches either have unacceptable bandwidth requirements or else only manage to simulate 3D interaction by using 2D images. Another problem with a server based system is that it can easily be flooded by multiple requests resulting in loss of performance.

Presently, the most popular way of doing client based interactive 3D on the web is to use VRML [11]. Even though VRML2.0 provides support for animation and scripting, it offers more from a content developer’s point of view than from an application developer’s perspective. So, though it would be misleading to call it merely a data-file format, it is not a general purpose programming language, which is a requirement in order to develop useful visualization software. Another reason that rules out VRML is that it does not provide any support for developing user interfaces, as it is only meant for creating 3D content, and without a custom designed intuitive interface it is impossible to provide a useful visualization tool for doctors.

In order for the software to be widely accessible it needs to be platform indepen-
2.1.3 The Visualization Toolkit

The Visualization Toolkit, \texttt{vtk}, is a C++ visualization library written at General Electric Corporate Research and Development [7]. \texttt{vtk} efficiently handles large volumetric datasets like those found in 3D medical visualization [4].

Most of the visualization work at SPL uses \texttt{vtk} and the models used in this project are in the standard \texttt{vtk} format (see appendix A), generated through a sequence of steps using different routines available in the toolkit. Figure 2.3 shows the \texttt{vtk} visualization pipeline used for generating the 3D model data files from the raw volume data-sets. These 3D models are subsequently interactively rendered by software like the one we have developed.

![vtk Pipeline for 3D Model Generation](image)

\texttt{vtk} uses a demand-driven visualization pipeline architecture to process volumetric data through a series of filters. After processing, the filtered data can be displayed using commercial computer graphics software. \texttt{vtk} supports OpenGL, Silicon Graphics’ GL, Hewlett Packard’s Starbase and Sun’s XGL. Vtk has a general image/volume import mechanism and exports Wavefront OBJ, OpenInventor, Renderman RIB and VRML geometry descriptions. The \texttt{vtk} routines have \textit{wrapper classes} so that they can easily be used by scripting languages like \texttt{lymb} and \texttt{Tcl} and also with Java. \texttt{vtk} is thus a very powerful toolkit and forms the back-end for our project by generating the 3D...
by CT, MRI or other techniques.

- **Image Processing:** Extracting useful grayscale images by filtering and processing the raw data.

- **Segmentation:** Identifying and marking various pixels as belonging to different anatomical structures (Automating this step is an interesting area of current research).

- **Modeling:** Generation of 3D model by extracting polygons from the volume data (an aggregate of successive cross-sectional data files). Various techniques are used to reduce the size of the obtained model.

- **Rendering:** The actual display of the 3D models, this is usually in the form of an interactive application with a graphical user interface.

One popular toolkit which is used to implement parts of the above mentioned visualization pipeline is described below.
Though these 2D cross sectional grayscale images are used by doctors all over the world for diagnostic purposes, they are limited inasmuch as it is very difficult to do mental reconstructions of the actual three dimensional anatomical structures they represent. Having a clear idea of the spatial relationship between the structures is not merely important for diagnoses but is critical while performing, say, a brain tumor surgery.

Fortunately, these 2D cross sectional slices can be gathered into volumes and processed to generate complete 3D anatomical structures. Doctors now have access to interactive 3D renderings of the data which allows them to not only see the 3D models but also manipulate the models and explore them to better appreciate the spatial relationships.

### 2.1.2 3D Visualization Pipeline

A number of techniques exist for interactive 3D visualization of the acquired data [5]. The approach used depends on the software and hardware available. Direct volume rendering of the data is computationally very expensive and is generally not used for interactive visualization, though with the advances in processor technology, it might become feasible later. One technique which is more commonly used is to extract geometric primitives from the volume data and display them using standard 3D graphics APIs [6]. The main steps in the data flow pipeline for generating these 3D models are illustrated in figure 2.2

The main steps in going from a human patient to the 3D visualization of his anatomical structures are –

- Acquisition: Obtaining the raw medical data in the form of cross sectional slices
2.1.1 2D Medical Visualization

Hospitals and laboratory use a wide variety of methods for obtaining data from the patient. Some of the popular data acquisition techniques are – X-Ray Computed Tomography (CT), Single Photon Emission Computed Tomography (SPECT), Positron Emission Tomography (PET), Magnetic Resonance Imaging (MRI), and Ultrasound. The obtained data is in the form of slice planes or cross-sectional images of the patient, similar to the conventional photographic X-Ray images. Each slice consists of a 2D array of numbers which may represent the attenuation of X-Rays (CT) or the relaxation of nuclear spin magnetization (Nuclear Magnetic Resonance Imaging). Grayscale values are assigned to these numbers for display on to a computer screen, whereby the structures in the image becomes apparent as a result of the interaction of the human visual system with the spatial organization of the data and the grayscale values we have chosen [4]. See figure 2.1 for an example of some of these images.

Figure 2.1: Coronal and Sagittal Grayscale Cross-Sections
CHAPTER 2

Medical Visualization

2.1 Introduction

Medical visualization is one of the most important areas of the broader discipline of scientific data visualization. Visualization can be defined as the process of exploring, transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data [4].

In the context of modern medicine, the goal is to present the doctor with the maximum information about a patient in an easily understandable format. Visualization uses image processing, computer graphics, and interactive tools to separate relevant from extraneous information. The results of the visualization process can be images, movies, or dynamic models with which the viewer can interact. No matter what form the processed data is in, the purpose of its presentation is to help the researcher, scientist, or physician better understand his\(^1\) data.

\(^{1}\)The pronoun his is merely being used as a convenience and should be read as his or her.
Surgical Planning Lab. Chapter 3 introduces Java3D, which is still an alpha technology, and was the development vehicle of choice. Chapter 4 explains the project design while Chapter 5 provides the implementation details. The appendix provides help on how to use the SPL Viz and also documents the various file formats the application uses.
With the growth of the Internet, and networked computing facilities in general, recent trends have been towards cross-platform, ‘write-once, run anywhere’ applications. However, the goal of platform independence generally runs counter to the goal of high speed interactive graphics, which seeks to exploit the platform architecture to get the maximum performance. Sun Microsystems has recently released Java3D, which is the first major, completely platform independent, API for 3D graphics. Java3D has been developed with performance in mind, and its layered architecture seeks to exploit the platform specific graphics acceleration by abstracting the API so that the low level rendering is transparent to the programmer.

We have developed the SPL Viz, a pure Java application for the 3D visualization of medical data. The Viz can be used for interactive manipulation of 3D anatomical models. Though this application is more suited for high-speed intranets, it can also be run over the internet with a data download delay.

This work has been in collaboration with the Surgical Planning Laboratory (SPL)\(^1\) at Brigham and Women’s Hospital (a teaching affiliate of Harvard Medical School). SPL provided all the medical input for the project in the form of medical datasets and feature specifications. Existing visualization tools at SPL were useful for understanding the features required for this application. Sun Microsystems provided early access to the Java3D software before it became available to the public at the JavaOne conference this year [3]. Sun Microsystems also loaned an UltraSparc workstation for the duration of the project.

The next chapter provides the context for this project by giving an introduction to the field of medical visualization and some of the current research work at the

---

\(^1\)See http://splweb.bwh.harvard.edu:8000/pages/index.html
CHAPTER 1

Introduction

Since the early days of Ivan Sutherland’s Sketchpad at MIT [1], the field of interactive computer graphics has developed tremendously, both in terms of the power and sophistication of hardware, algorithms and APIs, as well as in the scope of application areas. Whereas Ivan Sutherland used a 320 K Byte, 9 inch CRT and punch card input TX-2 computer, to basically just draw lines, today we have millions of times more powerful graphics accelerated workstations and high level software. They are used for applications ranging from special effects for the motion picture Titanic to networked military war-game simulations [2]. This thesis involves one such important and interesting application area, namely, Medical Visualization.

Medical data obtained through MRI (Magnetic Resonance Imaging) and CT (Computed Tomography) from patients is used by doctors for diagnoses, treatment and also for planning of neurosurgical processes. 3D reconstructions of this data provide the clinician with a more accessible visualization which is invaluable for understanding the spatial relationships between anatomical structures. This understanding is critical for planning surgical procedures and is also of immense benefit to students trying to understand the human anatomy.
List of Figures

2.1 Coronal and Sagittal Grayscale Cross-Sections .......................... 5
2.2 3D Visualization Pipeline .................................................. 7
2.3 \textit{vtk} Pipeline for 3D Model Generation ............................ 8
2.4 SPL/NSL Anatomy Browser Applet ........................................ 12
3.1 A Simple Java3D Scene Graph .............................................. 25
4.1 Calculating the Difference Image ......................................... 40
5.1 Project Timeline .............................................................. 45
5.2 Main Components of SPL Viz ............................................... 49
5.3 Screen Shot of the SPL Viz ................................................ 50
5.4 Scene Graph for SPL Viz ..................................................... 52
5.5 VRML Model Generated from \textit{vtk} Data ............................... 55
5.6 Screen Shot of the Manipulator Panel .................................. 58
5.7 Pyramidal Step Artifacts with Low Crease Angle ....................... 63
B.1 User Interfaces in SPL Viz .................................................. 7
B.2 Manipulator Panel and Top Control Panel ............................... 8
B.3 Slice Viewer Controls ....................................................... 9
B.4 SPL Picasso Pop-Up Window ............................................... 10
C.1 The Viz in Action ........................................................... 11
C.2 An SPL Viz Rendering of Semi-Transparent Skin with Vessels Inside . 12
C.3 The Viz with SPL Picasso Window Open ................................ 13
List of Tables

3.1 Graphics APIs ......................................... 17
3.2 JavaMedia APIs ........................................ 20
B SPL Viz Controls ............................................. 7
C Screen Shots ......................................................... 11
BIBLIOGRAPHY .......................................................... 14
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Medical Visualization</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1</td>
<td>2D Medical Visualization</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2</td>
<td>3D Visualization Pipeline</td>
<td>6</td>
</tr>
<tr>
<td>2.1.3</td>
<td>The Visualization Toolkit</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Web Based Visualization</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Visualization at S.P.L.</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>Medical Terminology</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Chapter Conclusion</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Java$^TM$</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>16</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Growth of Graphics APIs</td>
<td>17</td>
</tr>
<tr>
<td>3.1.2</td>
<td>The Internet and the Origin of Java3D</td>
<td>19</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Java3D Time-line</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Java</td>
<td>21</td>
</tr>
<tr>
<td>3.2.1</td>
<td>GUIs and Swing</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>Java3D Concepts</td>
<td>24</td>
</tr>
<tr>
<td>3.3.1</td>
<td>The Scene Graph</td>
<td>24</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Layered Architecture</td>
<td>26</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Geometry Compression</td>
<td>27</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Dynamic Loading</td>
<td>27</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Display Devices</td>
<td>27</td>
</tr>
</tbody>
</table>
ABSTRACT

MEDICAL VISUALIZATION USING JAVA3D™

SEPTEMBER 1998

ARISH ALI, B.TECH., I.I.T. KANPUR
M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Francis S. Hill, Jr.

One of the many applications of 3D Computer Graphics is in the area of Medical Visualization. Computer generated 3D reconstructions of medical data are used by doctors for planning neurosurgical processes as well as for instructional purposes. Java3D has recently been released by Sun Microsystems as a powerful cross-platform 3D graphics API. We have developed the SPL Viz, a Java3D application for medical visualization.

SPL Viz provides the doctors and researchers at the Surgical Planning Laboratory (Brigham and Women’s Hospital, Boston) with a platform-independent, network-aware, graphics accelerated application for visualizing the medical data obtained through medical imaging techniques like MRI and CT. This work builds upon the existing visualization tools at SPL, by providing their functionality in the context of a pure Java application. Many innovative features, like support for audio annotations and animated presentations, have been provided. Various techniques to mitigate the network delays in running the application over low bandwidth networks are also presented.
ACKNOWLEDGMENTS

This thesis would not have been possible without the support and guidance from Prof. Francis S. Hill, Jr. (University of Massachusetts, Amherst), Dr. Ron Kikinis (SPL, Brigham and Women’s Hospital, Boston), and Travis Bryson (Sun Microsystems, Mountain View, CA). I would also like to thank Polina Golland (AI Lab, MIT) for help with \texttt{vtk}. On a less technical note, I would like to thank Nirupama for taking my cooking turns.

Arish Ali

University of Massachusetts Amherst

SEPTEMBER 1998
To Ishrat
MEDICAL VISUALIZATION USING JAVA3D<sup>TM</sup>

A Thesis Presented

by

ARISH ALI

Approved as to style and content by:

________________________________________
Francis S. Hill, Jr., Chair

________________________________________
Maciej J. Ciesielski, Member

________________________________________
Beverly P. Woolf, Member

________________________________________
D. H. Schaubert, Department Head

Department of Electrical and Computer Engineering
Medical Visualization Using Java3D\textsuperscript{TM}

A Thesis Presented

by

ARISH ALI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

SEPTEMBER 1998
Electrical and Computer Engineering