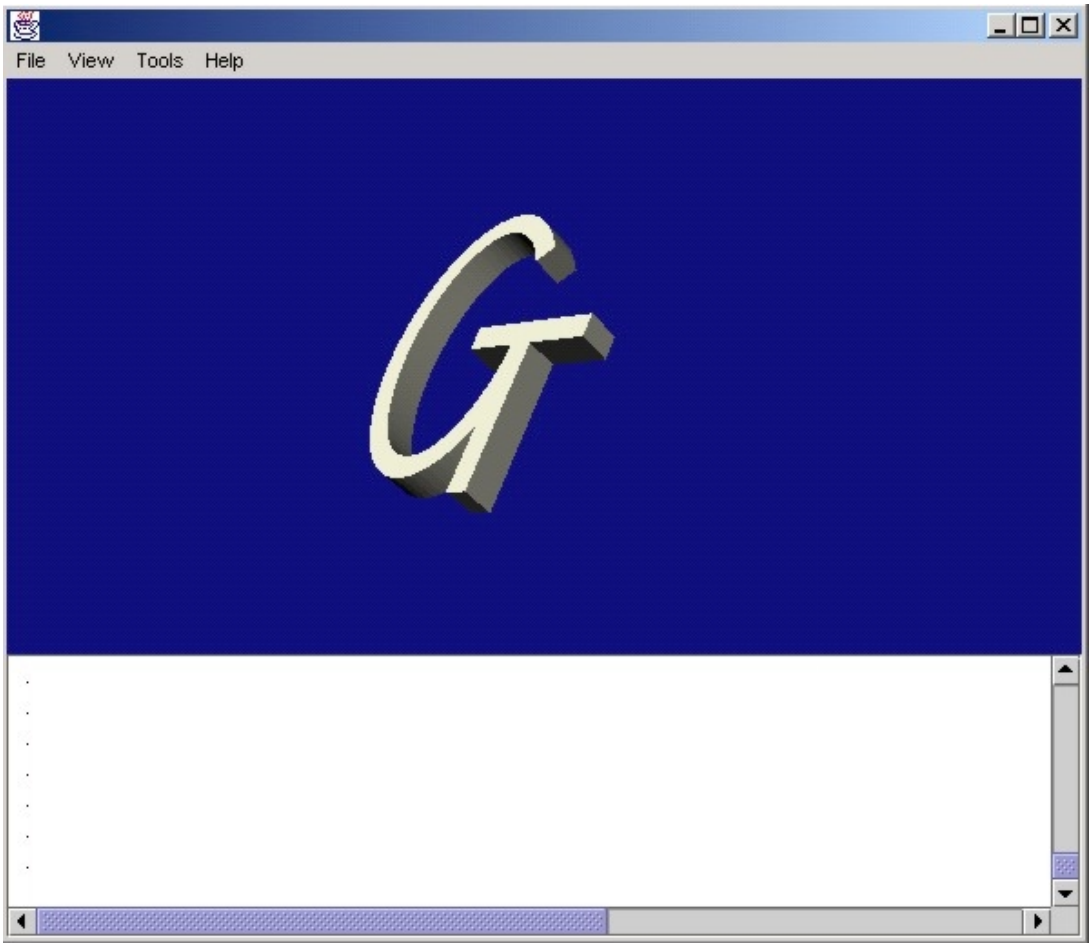
STLViewer



Matthew Chamberlain
Rahul Kulkarni
Christopher Williams

ME6104: Fundamentals of CAD
Dr. David Rosen
May 4th, 2001

Abstract

The rapid prototyping industry is foreseen as the key technology in merging design and manufacturing. It enables designers to take an idea from its original conception to prototype in just a few hours. As design moves onto the Internet to accommodate the distributed environment, many engineers are looking for software to bridge the inherent communication gap.

For this project we have developed a program that enables people of all different backgrounds view solid models over the Internet. Called "STLViewer," this software reads the STL (StereoLithography) file format (the *de facto* file format for rapid prototyping) and displays the model for the user to inspect. The user is able to rotate, pan, and zoom the model. Utilities are also provided for verification of the validity of the model.

STLViewer has a wide variety of potential applications. On-line 3D catalogs, infusion into an on-line ordering process for a SLA laboratory, and sharing of design concepts over the Internet are all potential uses for this software.

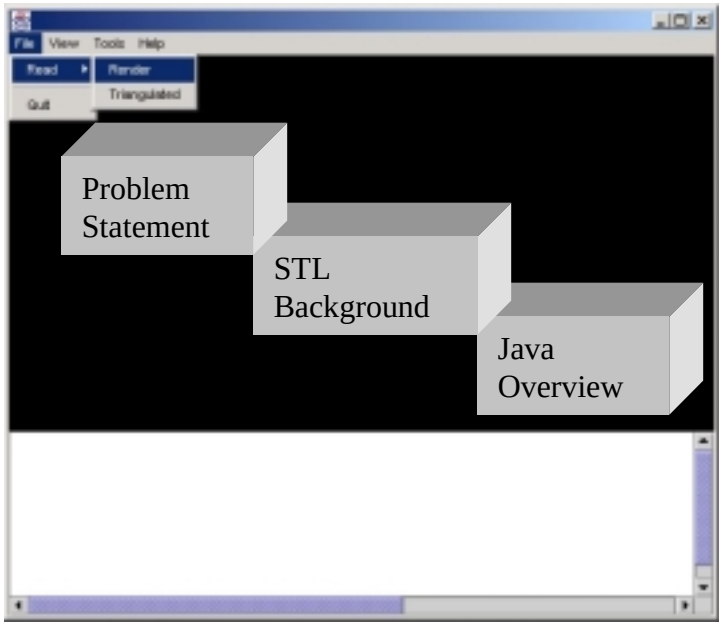
Through this project, the authors learned gained many valuable insights into STL file format issues, the Java programming language, the future of on-line software, and how components are described to CAD systems.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

In this chapter we introduce the problem to be studied. Background of STL files and the Java programming language is provided.

Insights into the details of STL files are provided from a literature survey. A discussion of common translation issues is given as well. Scenarios of the use for the program are also presented to illustrate the potential demand for its functionality.



1.1 PROBLEM STATEMENT

The rapid prototyping industry is growing each year. It is foreseen as the key technology in merging design and manufacturing. Because of this, rapid prototyping will prove to be even more useful as design and manufacturing move onto the Internet. This technology already is a step above other technologies because of its standard file format: STL (STereoLithography). In the distributed design environment of the future, designers from different parts of the globe will interact and attempt to design products over the Internet. Software that is able to provide designers the opportunity to collaborate on designs will be in high demand.

Through this project we hope to develop software that will facilitate communication between distributed designers working with the rapid prototyping technology. We will create a program that enables users to view solid models over the Internet that are saved in the STL file format – "an STLViewer." Through the development of this software, we hope to address the following foreseen problems that future designers will encounter:

- How can designs be effectively shared between both technical and non-technical people over the Internet?
- What types of attributes will be needed in future technical software to make communication and collaboration over the Internet more convenient and efficient?

In addition to these overall problems we plan to investigate the more concrete problems of the STL file format:

- How can you check to see if your STL file is valid model?
- Which normal vector (the one in the file, or the one calculated by the cross product of the vertices' vectors) is correct and when?

- How serious are the issues of redundant information, large significant figures, and the size of the file format in the development of the software?

All in all, our mission for this project is to develop software that will enable people of both technical and non-technical backgrounds to view STL files over the Internet.

1.2 SOME BACKGROUND ON STL FILES

1.2.1 What are STL Files?

Computer aided design and rapid prototyping technology become more and more prevalent every day as industry, academia, and researchers look for fast and inexpensive ways of creating virtual and physical models of objects. A key disjoint exists between the complicated three-dimensional renderings created by CAD systems and the capabilities of rapid prototyping machines. One way or another, it is necessary to break these models down to a more simple form in order to make them accessible to the machines that will realize the physical prototypes. The *de facto* readable file format (as it is often called) for this application is *.STL (Kai #1 1997).

The STL file format was developed by 3D Systems Corporation in 1987 as part of their effort to bring the first commercially available rapid prototyping technology (stereolithography) to market (Jacobs 1996). Since that time several dozen other different rapid prototyping methods have been developed and a number of deficiencies in the STL format have been identified, but, due to the simplicity of the format and its independence from any one CAD package, STL has remained industry standard (Kai, Jacob et al. 1997). Several other file formats have been developed:

- most notably CFL (Cubital Facet List) which is much more compact
- CLI (Common Layer Interface) which was developed specifically for medical scanning technology
- RPI (for Rensselaer Polytechnic Institute, where the format was developed)
- IGES (International Graphics Exchange Standard)

None have been universally adopted or interpreted in the same way as STL by the various makers of CAD packages (Fadel and Kirschman 1996).

The STL file format represents the surfaces of models as triangles, simple shapes that can be used to exactly break down rectangular sections or in larger numbers to closely

approximate curves. Generally, CAD packages include options for breaking down models into STL format through a process called tessellation. The result of the tessellation is a file in either ASCII or binary form that can then be read by an SLA machine (Ngoi, Chua et al. 1993). The ASCII file format has the advantage over the binary format of being readable to humans as well as machines, but has a distinct disadvantage in file size. According to Ngoi (1993) an STL file for a 12 facet object in ASCII will be 2.8 kilobytes while the same shape represented in binary form will take up only 0.68 kilobytes. In general, it is assumed that a binary STL file will be about ¼ to 1/6 as large as an ASCII file representing the same part (Fadel and Kirschman 1996).

1.2.2 A Look Inside an STL File

All STL files, whether they are in ASCII or binary format and regardless of the platform on which they were originally designed, have the same basic format. Each file starts out with a brief header that contains data about the object and the CAD system on which it was designed. This is followed by a number representing the number of facets present in the file and then the list of facets themselves (Ngoi, Chua et al. 1993).

Each facet is made up of four sets of X, Y, and Z coordinates, with each coordinate saved as a floating-point number with eight significant figures. The first set of coordinates represents the normal vector for the facet. The next three sets spell out the locations of the three vertices of the triangle in counter-clockwise order. At the end of each facet definition is a small piece of data for facet attributes (Jacobs 1996).

An ASCII STL, as previously mentioned, has the benefit of being readable by humans. An excerpt of an ASCII STL file of a cube is presented below:

```
solid cube
facet normal -1.000000e+000 0.000000e+000 0.000000e+000
   outer loop
      vertex 5.000010e-004 1.005000e-001 1.000000e-001
      vertex 5.000010e-004 1.005000e-001 0.000000e+000
      vertex 5.000010e-004 5.000010e-004 1.000000e-001
   endloop
```

```
endfacet
```

The keywords “normal” and “vertex” are extremely helpful to the human reader as it describes the numbers that follow.

For a binary STL file, the file size would include 84 bytes for the header and triangle count plus another 50 bytes for every triangle present. Thus, for the 12-facet object mentioned previously, the file size computation would be: (Jacobs 1996)

84 bytes (header) + 12 x 50 bytes (facets) = 84 bytes + 600 bytes = **684 bytes** (or roughly .68 kilobytes)

Two major rules ensure that an STL file is processed properly once it is passed on to the RP machine.

- First, the vertexes of each triangle must be ordered in a counter-clockwise manner. This serves to back up the built-in definition of the normal, since the right-hand rule can also be used to calculate which side of the face is outward. A clockwise-oriented set of vertexes indicates an inward-facing facet.
- Second, each set of adjoining facets must meet along just one common edge. This is the so-called “vertex-to-vertex” rule. The vertex of one triangle may not intersect the edge of an adjoining triangle. (also stated by (Chen, Ng et al. 1999))

1.2.3 Potential Problems with STL Files

Authorities on the subject of rapid prototyping seem to differ widely on the overall robustness of the STL file format, as evidenced by the following passages:

[STL] is conceptually simple, topologically robust and, when used with sufficient resolution, is capable of high accuracy.” - ((Jacobs 1996), page 5)

Several problems plague STL files and they are due to the very nature of STL files as they contain no topological data. Many commercial tessellation algorithms used by CAD vendors today are also no robust…" ((Leong, Chua et al. 1996), page 407)

Furthermore, another author claims that, in 10 percent of all cases, STL files created from solid models contain errors while a full 90 percent of all files based on surface models are invalid (Bailey 1995).

STL files, by their nature contain certain inherent weaknesses:

1. The format of STL files is redundant since information on the normal is contained twice. The normal is described explicitly in the definition of each triangle but could also be found quite simply by using the order in which the vertices are listed in combination with the right-hand rule. This problem results in huge STL files that take up space on computers.
2. A complete geometric description of the shape is not given by the STL format because some pieces of the detail are lost in the breakdown into the polygon representation. To recreate the model based on the vertices, matching vertices must be searched for and aligned. (This turns out to be a time consuming process.) At high levels of detail, this process can be dangerous, as round off error can cause vertices that are close to one another to be mistaken for the same point.
3. The method by which STL translators approximate the model is not consistent. Depending on whether or not a surface is convex or concave, its STL approximation will either add or subtract material.
4. The format lacks the ability to include any technological information on the facets (such as material or surface finish) (Stroud and Xirouchakis 2000).
5. The format is not very "information rich," especially compared to other representation schemes like NURBS or B-Splines. In order to achieve the same degree of mathematical precision in representing a curve as a few splines, thousands of tiny triangles would be needed (Kai, Jacob et al. 1997).

Beyond these inherent weaknesses to the general format of STL, many of STL files contain errors. The most typical of these errors are the following:

1. Gaps, cracks, holes, or punctures due to missing facets (see Figure 1a). This problem is often caused when a part contains multiple intersecting surfaces with large amounts of curvature.
2. Overlapping facets, which may be caused by round off errors in the handling of the eight significant figure floating-point variables that make up the vertices (see Figure 1b).
3. Degenerate facets in which all of the vertices (and thus all of the edges) are collinear, as seen in Figure 1c. These facets do not contain valid normal vectors. This may be caused by the algorithms that have been written to avoid missing facets.
4. Non-manifold topology conditions. This occurs when more than two facets share an edge. Figures 1d and 1e show two different types of non-manifold conditions in parts (Leong, Chua et al. 1996).

The potential presence of this many different types of error in STL files suggests that it might be wise to design software that would allow designers to check both visually and using exhaustive mathematical search methods for mistakes, then correct them in a manner consistent with the rest of the file. We hope to begin to address this need in our work.

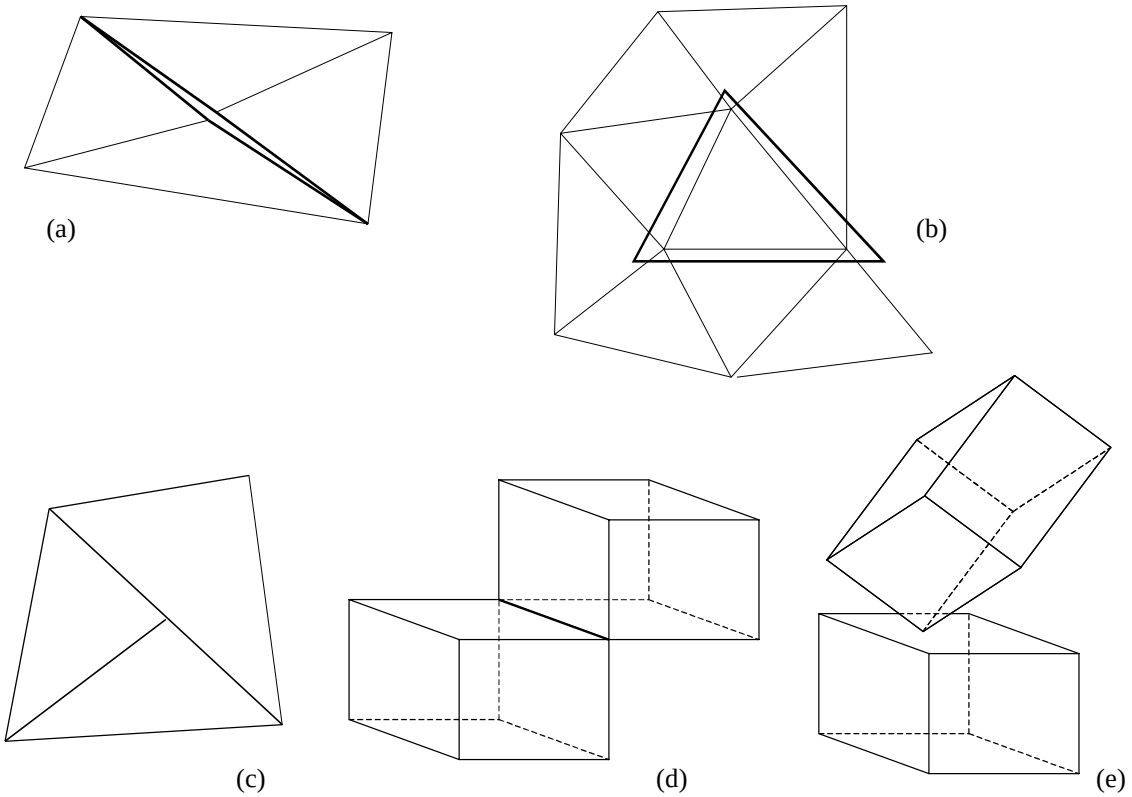


Figure 1: Common errors in .STL files

1.2.4 Where Could an STL Viewer Be Used?

We foresee several key uses for a software package that addresses our problem statement. Used as an intermediary between the CAD package and the actual RP machinery, this software could prove useful to a variety of different interest groups in a variety of different circumstances. Here, we outline just a few of them:

- *As a fast check for errors* – Visual inspection of the image in our viewer could provide designers with the earliest indication possible that some or all of the triangles in the model are invalid or missing. Holes, punctures, and gaps might be identified. The level of detail present in the triangle mesh could also be inspected to see if its resolution is fine enough for the tastes of the designer.
- *As an interface between CAD platforms* – Since it would run on any up-to-date web browser, this software could provide an easy means of file and image sharing between two designers who work on different CAD

platforms, since the STL code generated by translators is generic and universally readable.

- *As a way of displaying CAD files without CAD* – Not all people in a company or organization are necessarily going to have the CAD package a part was designed in loaded onto their desktop computer. Our viewer would allow anyone with simple internet capability to inspect an image of the part being designed.
- *For demonstration purposes* – Again, not all laptop computers or those computers that are available in meeting rooms and lecture halls have CAD packages loaded onto them, but they oftentimes have internet capabilities. For the purposes of a demonstration concerned with the part under design (and not with the inner workings of the CAD package), a three-dimensional web-based image would probably be acceptable.
- *For web-based display of products* – Parts manufacturers and resellers could find an STL viewer to be a very useful way of displaying the geometry of their products to customers via the internet and online catalogs. Customers would benefit from being able to see the shape and arrangement of a part before they buy it. The manufacturers of custom parts might find further use for the software in the ability to view a part that a customer wants built quickly via the internet. This would give them a quick first look at the part and possibly the ability to accept or refuse jobs on the spot.
- *An extension of a 3D digitizer* – The use of a 3D digitizer to scan models and turn them into a STL mesh can now be extended to the Internet. Vendors, designers, and doctors that use 3D scanning software can now share physical parts over the Internet without the need or use of a complex CAD software package.

These points certainly do not represent the limit of the usefulness of an STL viewer. Indeed, a robust viewing package could provide the backbone for a suite of small utilities

for checking, correcting, altering, and even augmenting STL files (Ngoi, Chua et al. 1993). This concept is described further later in this paper.

1.3 OVERVIEW OF JAVA2 & JAVA3D

Due to our desire to offer this software over the Internet so that it is compatible with any computer currently on the market, we chose to design our software with the Java programming language. This sub-chapter contains an explanation of why we chose this programming language with a full description of its benefits and limitations.

1.3.1 Description

Java is a platform-independent programming language that supports many modern programming features and techniques. It is designed to run in a distributed network environment and run across heterogeneous platforms. This makes Java an ideal language for building modern Internet computing based applications.

Java is comprised of many Application Programming Interfaces (APIs) that add functionality to the basic features of Java mentioned above. The Java 3D™ API provides a set of object-oriented interfaces that support a simple, high-level programming model. This enables developers to build, render, and control the behavior of 3D objects and visual environments.

The Java 3D API takes advantage of existing hardware accelerators through low-level APIs such as OpenGL™ and Direct3D. This allows applications written using the Java 3D API to run on any platform with a Java virtual machine (JDK™ version 1.2 or higher software) and OpenGL or Direct3D implementation. To accommodate a wide variety of file formats, runtime loaders are supported. The Java 3D API benefits developers in diverse markets including scientific visualization, animation, Web site design, simulations, virtual world construction, training, games, and design automation. As an integral part of the Java™ Media API, the Java 3D API makes it easier for programmers to integrate 2D and 3D graphics, video, audio, and image processing, as well as multimedia and visualization features, in a single application.

The Java 3D API incorporates a high-level, scene-graph model that helps developers focus on objects and scene composition. This speeds application development, because programmers do not need to design specific geometric shapes or write rendering code for the scene display.

1.3.2 Features

Features of the Java language can be summarized as follows:

1. Object Oriented

   Java is an object-oriented language that allows the developer to develop applications as a set of self-contained objects that interact with each other. Each object encapsulates its data and exposes methods to other objects.

2. Portable

   One can compile Java code in a platform-independent byte-code format that is interpreted by a Java virtual machine (JVM). Java code can run on any platform with a JVM, making it ideal for running on heterogeneous networks.

3. Robust

   Java automatically deallocates memory using a garbage collector. This makes the program more robust because you do not have memory leaks and program crashes. Java also provides a powerful exception handling mechanism that simplifies error handling and recovery.

4. Multithreading

   Unlike other languages, Java has built in support for threads at the language level, making it much easier to build and deploy multithreaded Java programs across heterogeneous platforms. The thread library is not an add-on to the language, but part of the language.

5. Distributed

   Java comes with an extensive set of libraries that provide high-level support for networking standards and protocols, such as TCP/IP and HTTP. In addition, Java is tightly integrated with distributed computing standards, such as CORBA, this, along with RMI support, allow Java objects to communicate across a network.

6. Secure

   Java provides a sandbox security model that allows you to limit access to operating system resources for Java code downloaded from a network. This prevents inappropriate or harmful use of Java code, such as to grab operating system resources and access private data. Java also provides support for digital signatures that establish the creator of a Java application.

Features of the Java 3D API can be summarized as follows:

1. The flexible viewing model utilizes a broad range of display devices including flat-screen, head-mounted, and stereo displays as well as caves and portals -- without modifying the application.
2. Integrated 3D graphics and sound create a more exciting viewer experience.
3. Multiple levels of detail enable viewers to increase the resolution of near objects, improving application performance.
4. Support for continuous action devices, such as trackers used in immersive caves and portals, enhances the interaction of Java 3D applications.
5. Based on open standards, the Java 3D API specification was developed with input from leading technology vendors.
6. Because the Java 3D API is written for the Java platform, developers can freely mix it with other Java APIs in a single application.

1.3.3 Limitations

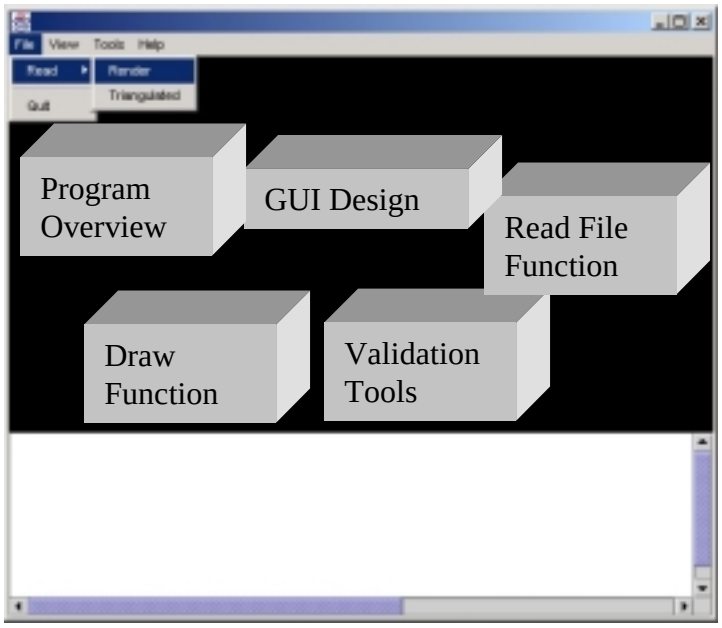
1. Java 3D does not have a file format of its own. As a result changes made to a Java 3D scene cannot be saved easily. Additional code needs to be written to save it in a standard file format.
2. Java 3D does not offer support for techniques for handling complexities in 3D graphics like advanced operations in BSP trees. This makes Java 3D slow when traversing large data structures.
3. Java 3D is designed for real-time animations and hence there is little support for printing and off-screen rendering.

1.3.4 Use and Installation Base

Java 3D is not a standard part of any web browser. Neither Internet Explorer nor Netscape Navigator has any 3D browser plug-in. One needs to separately download and install the Java 3D plug-in for a particular operating system. This can be cumbersome, especially for a user that are not familiar with computers or how to install plug-ins. Also this is the problem with persons not granted access to install files on a machine that is part of a network.

CHAPTER 2 “STLVIEWER” DESIGN

In this chapter we provide insight into how the program was designed. An overview of the program is followed by a thorough discussion of each module of the software. Important snippets of the code are provided for further understanding.



2.1 PROGRAM OVERVIEW

Our program, entitled “STLViewer,” operates through a Java applet in the user’s web-browser. The program is accessed by the user by connecting to the web-site containing the program (currently http://srl.marc.gatech.edu/~williams/Stlviewer.html).

The user selects an ASCII STL file they wish to view and pastes the text into the provided text area at the bottom of the Graphical User Interface (GUI). The program then parses through the file and displays the solid model. The algorithm reads the vertex and the normal information contained within the STL file. This data is then passed to a function that draws the model on the user’s screen. The program is able to display the model as a rendered solid or as a triangulated wireframe. Both models allow the user to visually inspect the model.

The user also has the option of comparing the normals provided by the STL file with normals calculated from the vertex information. If this check is completed, the user can choose to have the solid re-represented with the newly calculated normals. This allows the user to verify any errors caused by the translation of their CAD model into the STL file format.

2.2 GUI DESIGN

An important aspect in any piece of software is the design of the Graphical User Interface (GUI). As computers continue to spread throughout all levels of society, each piece of software needs to be as "user-friendly" as possible. Today's users no longer use DOS; they are only comfortable with the point-and-click interface of Windows.

The JAVA programming language provides many classes to assist the programmer in developing a GUI. Windows, dialog boxes, text areas, radio buttons, menus, and buttons are all at the programmer's disposal.

As mentioned earlier, one of the foreseen benefits of the STLViewer is its appeal to those users who do not have a technical background, do not own a high-end CAD software package, but want to view a solid model of a product (i.e. marketer, manager, or on-line customer). For this reason, we wanted to make sure that our GUI simple, yet efficient and effective.

An important part of this program, the GUI is divided into three main areas: the menubar, the 3D canvas, and the text area.

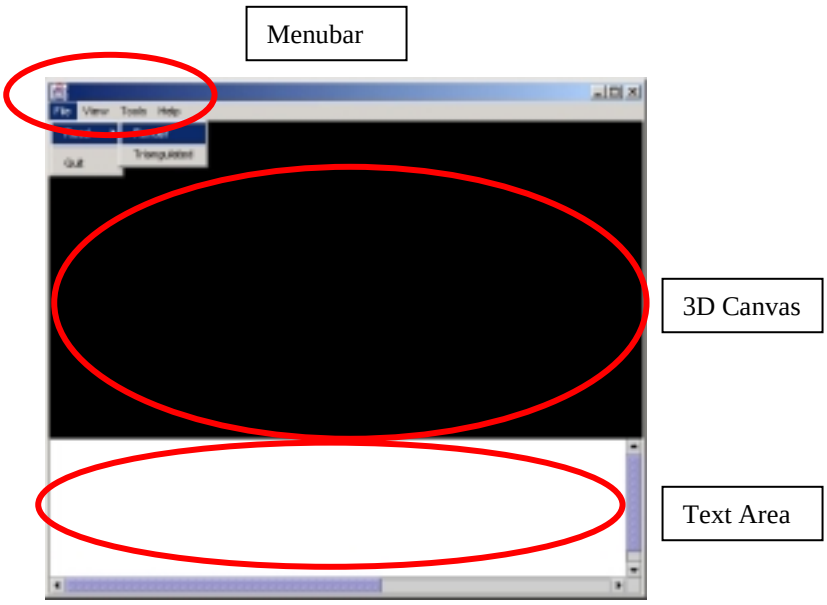


Figure 2: GUI Design

2.2.1 Menubar

Located at the top of the applet frame, the menubar provides the user with quick access to all of the program's main functions. Located at the top of the frame, the menubar contains four menus from which to choose: "File," "View," "Tools," and "Help."

File Menu

The file menu contains the two main functions that the user will use: "Read" and "Quit." "Read" is a submenu that enables the user to either open the STL file as a rendered solid, or as a triangulated wireframe. Choosing "Read -> Render" calls the readFile() function (discussed in Section 2.2). The solid model is displayed as a rendered object. The "Read -> Triangulated" option draws the triangles that were created in the STL file conversion process. The triangulated solid model serves as a wireframe model that the user can closely inspect the triangulation done by their CAD package for errors (ex. Gaps and vertices not matching (see Section 1.1.3)). The algorithms for drawing these separate types of models are presented in Section 2.3.3 - Drawing the Solid Model.

The "Quit" command closes the exits the program and applet window. The user may quit the program by either selecting this option, or by clicking on the "x" box in the upper-right hand corner of the frame.

View Menu

The "View" menu contains only one command for the user to execute: "Clear." This command simply clears the current model from the screen and also clears the information displayed in the text area. Once "Clear" is chosen, the user may open another STL file to view a new model.

Tools Menu

The commands from this menu provide technical tools for the user to evaluate the solid model. Containing two commands, "Normal Check" and "Redraw with New Normals," this menu is used by the user after a solid model is loaded.

"Normal Check" involves two steps. First, the three vertices coordinates of each triangle are used to calculate the triangle's normal. Second, it is compared to the normal provided by the STL file. If there is a discrepancy, it is reported to the user through the text area. The mathematical and programming details of this tool are described later in Section 2.4.1 – Normal Check Function.

"Redraw with New Normals" does exactly what it says it does. Used after "Normal Check," the user has the ability to redraw the solid model with the new normals calculated by "Normal Check." This enables the user to visually identify any sort of discrepancies between the two normal vector sets. Again, this is described in more detail in Section 2.4.2 – Redraw with New Normals Function.

Help Menu

This menu provides instructions on how to use the program as well as details about the software. This was done in an attempt to make our software even more user-friendly.

Choosing "Help -> Instructions" prints the following message in the text area:

- Paste (Ctrl-v) the ASCII contents of an STL file into the text area.
- Select 'Read' from the File menu.
- Once the model appears:
  - Drag the mouse with the left mouse button to rotate the object.
  - Hold the 'Alt' key while dragging the mouse to zoom in and out.
  - Drag the model with the right mouse button to pan.

With this the user is able to operate the program successfully.

Choosing "Help -> About STLViewer," the user is finds the following message in the text area:

Programming Details

The menubar was created using the AWT component class of the JAVA programming language. For this portion of the program, it was necessary to first define the menubar and add it to the applet frame (denoted by variable "f" in this example):

```
MenuBar menubar = new MenuBar();
f.setMenuBar(menubar);
```

Menus were added to then added to the menubar:

```
Menu file = new Menu("File");
menubar.add(file);
```

MenuItem's were then added to each menu. As each MenuItem is added to the menu, "ActionListeners" are added as well. This enables the applet to listen for selections by the user's mouse. Please note that since "Read" is a submenu, it is first classified as a "Menu." Its MenuItems are then added to it and they are then added to the "File" menu.

```
Menu read = new Menu("Read");
MenuItem render = new MenuItem("Render");
render.addActionListener(this);
render.setActionCommand("render");
read.add(render);
MenuItem triangle = new MenuItem("Triangulated");
triangle.addActionListener(this);
triangle.setActionCommand("triangle");
read.add(triangle);
file.add(read);
file.addSeparator();        // Put a separator in the menu
MenuItem quit = new MenuItem("Quit");
quit.addActionListener(this);
quit.setActionCommand("quit");
file.add(quit);
```

Later in the program, the events associated with menubar selections are defined:

```
public void actionPerformed(ActionEvent e) {
  String cmd = e.getActionCommand();
  if (cmd.equals("render")) {
        rendercheck = 1;
        readFile();
  } … }
```

This process is repeated for each menu item and its corresponding event.

This is a standard programming feature for Java GUI components. If future work is planned for this program, it should be noted that these components were made with Java's AWT Class and not the newer Swing Class. This was due to conflicts between "lightweight" and "heavyweight" components. The "heavyweight" Canvas3D of the GUI would have blocked the "lightweight" JMenuBar Class of the Swing class. When the user would have tried to pull down a menu, nothing would be visible, as the Canvas3D would be drawn over the menu. For our purposes, the AWT menubar suffices.

2.2.2 The 3D Canvas

The 3D Canvas is located at the middle of the applet frame. This canvas is where the solid model is displayed and is therefore the center point of the GUI. Created with the Java 3D API, the 3D Canvas is a Canvas3D() object. By following the instructions printed by the "Instructions" menu item (discussed above), the user is able to rotate, pan, and zoom in/out on the model.

More details on how the solid model is presented to the user on this canvas are presented in Section 2.3.1 - The 3D Canvas Setup and the Simple Universe.

2.2.3 The Text Area

The text area is another integral part of the program. The text area serves two primary purposes:

1. To display important messages and information to the user
2. It serves as the input field for the user.

As mentioned in the above description of the "Help" menu item, instructions and comments about the program are presented in the text area for the user to view. The text

area is also used during the “Check Normals” function. If the calculated normal does not match the one given by the STL file, the user is alerted to it through the text area.

The text area is not only capable of handling output for the user; it is also the source of input for the program. Once the software is opened, the user is asked to paste their STL file (in ASCII format) into the text area. Once the user has done this and the “Read ->Render/Triangulated” menu item is chosen, the text area is parsed, read, and the normals and vertices of the STL file are passed to a draw function (described in Section 2.2.3 – Parsing a Line for Geometry).

It is important to discuss the use of the text area as the input source. This is a rather unconventional way of receiving user input. Usually, programs use file dialog boxes where the user is able to browse their computer for the appropriate file and then simply clicks “Open.” This method is easy to use and familiar to most users.

This method of opening files was our first choice (there is a working version of the code where this is an option), but we encountered many difficulties with this method. When using Java applets stored on a server (separate from your computer), there are many Java security restrictions. These security restrictions are in place so to keep users uploading /downloading viruses to/from the server. As mentioned earlier, a primary goal for the software is its ability to work over the Internet. Because of current limitations in security software, we were unable to allow the user to upload a file to the server to be read and therefore had to find another way to input the file.

Using the text area as an input source seemed like the most logical solution. Acting on this idea we first designed the text area using Java’s AWT component, TextArea(). At first it seemed to work well, but as we ran more complex examples, we found a major limitation in the code. After some research we found that the AWT TextArea() component had an input size limitation of only 50kB. Since ASCII STL files are so large (as discussed in Section 1.1.3 - Potential Problems with STL Files) we faced a potential dilemma:

- We were unable to offer the user the ability to upload a file to the server for security reasons.
- The text area meant for input, could not manage large ASCII STL files.
- Since it is impossible to paste a Binary STL file into a text area, which would not solve our problem.

After more research, we found that the answer to our problem came from the Java Swing component JtextArea(). This more advanced, lightweight, component enables the user to paste any amount of text that the user's computer can withstand. Unlike the menubar (discussed above), we had no problems mixing the Swing text area with the rest of the "heavyweight" components.

Although the problem was solved, this was our first encounter of problems with the size of the STL file in ASCII format.

2.3 READ FILE FUNCTION

There are two ways in which an STL file can be read by the program:

1. Taking input as a file
2. Taking input from a text area where the contents of the STL file are pasted

Both these features have been implemented in the program. However since an applet running in a web browser does not have access to local files due to security restrictions, the feature enabling file input has been disabled in the program. However, documentation has been provided for reading the file for future use.

2.3.1 Taking Input as a File

Clicking on the *Read* button brings up the *Open* Dialog box as shown in Figure 3.

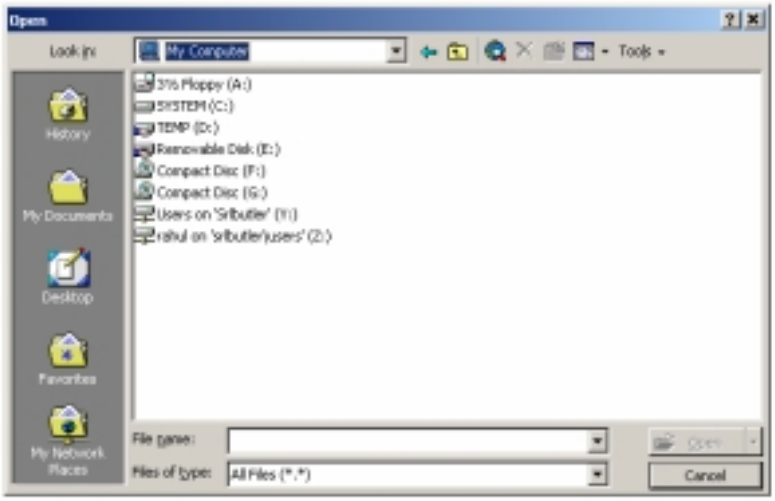


Figure 3: Open File Dialog Box

The corresponding code is as follows:

```
FileDialog d = new FileDialog(f, "Open File", FileDialog.LOAD);
d.show();
currentfile = ""+d.getDirectory()+d.getFile();
readfile();
```

The variable *currentfile* stores the complete path and filename for the file selected. Control is then passed on to the *readfile()* function, which parses the file and extracts geometry data.

The *readfile()* function then inputs the file using a *BufferedReader.* One line at a time can then be extracted from the *BufferedReader* using its *readLine()* function. Code reading each line is given below:

```
URL source=null;
try
{
     source = new URL("file:///"+currentfile);
} catch (MalformedURLException  e ) { }

try
{
     BufferedReader br = new BufferedReader(
                  new InputStreamReader(source.openStream())
                                                 );
     while(null != (aLine = br.readLine()))
     {
       // Main string parsing code here
     }
} catch (IOException e ) {}
```

2.3.2 Taking Input From the Text Area

For reading a line at a time from a text area we used a construct called *StringTokenizer*. The *StringTokenizer* construct allows us to split the string read using “/n” which is used to represent a carriage return. Thus we get one line at a time.

```
String stlfile = textarea.getText();
StringTokenizer st = new StringTokenizer(stlfile, "\n");

st = new StringTokenizer(stlfile, "\n");
while (st.hasMoreTokens())
{
      aLine = st.nextToken();
      // Main string parsing code here
}
```

2.3.3 Parsing a Line for Geometry Data

The entire set of lines in the STL file are parsed twice:

1. To get the number of vertices so that the array for storing vertices and normals may be defined
2. To get each vertex and normal and store them in the respective arrays

On the first pass through the file, we need to count the number of vertices in order to initliaze the array that will store the vertices. This counter is labeled “numverts.”

```
StringTokenizer st = new StringTokenizer(stlfile, "\n");

//register # of vertices for array lengths
numverts = 0;
String aLine = "";
while (st.hasMoreTokens()) {
    aLine = st.nextToken();
    if (aLine.indexOf("vertex") != -1)
    numverts++;
}
```

On the second pass, we fill the initilized arrays with the values read from the STL file. The basic idea is to find the keywords, “normal” or “vertex,” in the line and get the coordinates for each of them.

```
if (aLine.indexOf("vertex") != -1) {
     // Parse string
}
if (aLine.indexOf("normal") != -1) {
     // Parse string
}
```

The string is parsed using the method *substring*. Using *substring* we can get any part of a string by specifying the start and end points. For a vertex and a normal statement, there is the keyword followed by spaces and then a space separating each coordinate. This property of the syntax of the STL file has been used to parse it. The coordinates obtained are converted to *float* for the normals and for the vertices.

For the vertices the corresponding code is:

```
int startat  = aLine.indexOf("vertex");
```

```
int endat    = aLine.indexOf(" ", startat+8);
String vert1 = aLine.substring(startat+7, endat);
double vert1_flt = Float.parseFloat(vert1);
startat = endat+1;
endat   = aLine.indexOf(" ", startat+2);
String vert2 = aLine.substring(startat, endat);
float vert2_flt = Float.parseFloat(vert2);
startat = endat+1;
String vert3 = aLine.substring(startat);
float vert3_flt = Float.parseFloat(vert3);
```

Each vertex is stored into the "tricoords" Point3f array. This array is later passed to a function that draws the solid.

```
tricoords[vert_count] = new Point3f(vert1_flt,
                                                vert2_flt, vert3_flt);
vert_count++;
```

Another variable in this code is the float "bigpoint." This float records the largest (absolute) value of a vertex. The vertex furthest away from the origin is saved as the "bigpoint."

```
if (bigpoint < Math.abs(vert1_flt)) { bigpoint = vert1_flt; }
if (bigpoint < Math.abs(vert2_flt)) { bigpoint = vert2_flt; }
if (bigpoint < Math.abs(vert3_flt)) { bigpoint = vert3_flt; }
```

This value is retained and is later passed to a function which automatically scales the model display. This is discussed in further detail in Section 2.3.2 – Scaling the Simple Universe.

When the algorithm finds the keyword "normal" it runs the corresponding code:

```
int startat  = aLine.indexOf("normal");
int endat    = aLine.indexOf(" ", startat+8);
String norm1 = aLine.substring(startat+7, endat);
float norm1_flt = Float.parseFloat(norm1);
startat = endat+1;
endat   = aLine.indexOf(" ", startat+2);
String norm2 = aLine.substring(startat, endat);
float norm2_flt = Float.parseFloat(norm2);
startat = endat+1;
String norm3 = aLine.substring(startat);
float norm3_flt = Float.parseFloat(norm3);
```

Each normal is stored in the "normals" Vector3f array. Note that in Java3D a normal has to be specified for every vertex.

```
normals[norm_count] = new Vector3f(norm1_flt,
                                                norm2_flt, norm3_flt);
normals[norm_count+1] = new Vector3f(norm1_flt,
                                                norm2_flt, norm3_flt);
normals[norm_count+2] = new Vector3f(norm1_flt,
                                                norm2_flt, norm3_flt);
norm_count+=3;
```

The vertices and normals are then stored in the corresponding arrays. Counts for vertex and normal arrays are updated for the next line. They are now ready to passed to the solid model draw function.

2.4 DRAW FUNCTION

In this section we will discuss how the 3D canvas displays the solid model. We begin with a description of how the canvas is setup. We proceed to a description of how the model is made using the array of normals and coordinates passed from the readFile() function.

2.4.1 The 3D Canvas Setup and the Simple Universe

As soon as the program is started the three main GUI components (menubar, 3D canvas, and text area) are instantiated. While the menubar and text area are common components that are easily configured, the 3D canvas requires the assistance of Java3D. A lot of code needs to be written just to properly get the canvas ready for the solid.

The core of any Java3D program is its Scene Graph data structure. The Scene Graph is an arrangement of three-dimensional objects in a tree structure that completely specifies the content of a Virtual Universe, and how it is to be rendered. Java 3D supplies a type of Virtual Universe called Simple Universe. As can be seen from Figure 4, a Simple Universe contains all the ingredients necessary for a basic 3D program. For this project, the Simple Universe was used.

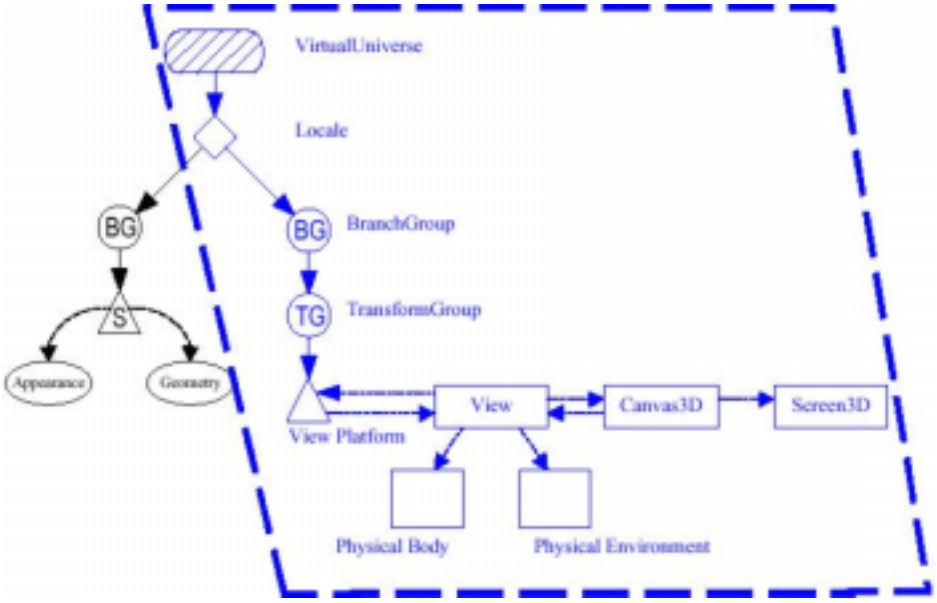


Figure 4: Simple Universe Diagram (cite Java3D tutorial)

The scene graph for our program is presented below in Figure 5.

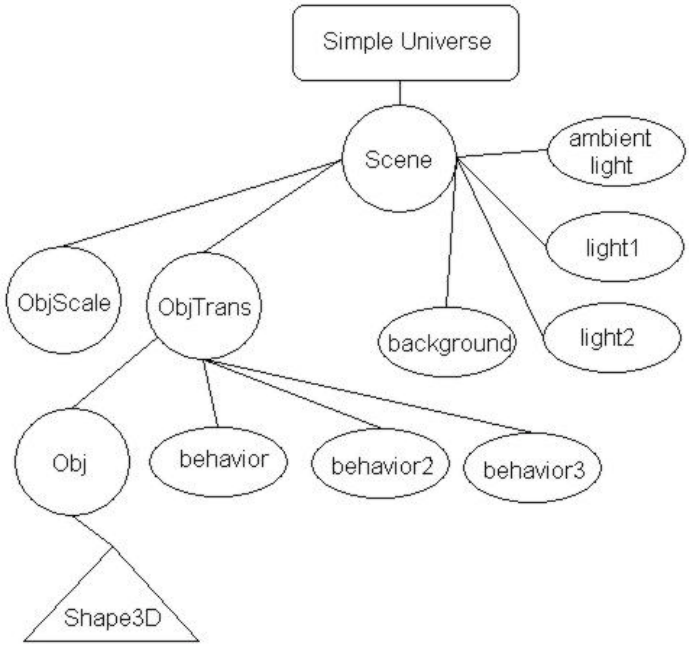


Figure 5: Scene Graph of "STLviewer"

The Simple Universe is located at the top of the graph. The Scene Graph "Scene" is attached to it. "Scene" is given the ability to detach from the Simple Universe so that the user is able to clear the current solid model from the 3D Canvas (using the "View -> Clear" menu item). From this Scene Graph two different types of Branch Groups are attached:

- "objScale" – a Transform Group that manages the Bounding Sphere of the model.
- "objTrans" - a Transform Group that manages the behaviors of the mouse.

Also attached to "Scene" are three different sources of lighting (two directional and one ambient) and one background node that takes care of the background color.

Attached to the "objTrans" Branch Group is three behavior nodes that assist the program in recognizing the mouse movements that rotate the object ("behavior"), zoom in and out ("behavior2"), and pan the object ("behavior3"). Also attached to the "objTrans" Branch Group is the "obj" Branch Group. Finally "Shape3D", which contains the actual solid model, is added to "obj".

2.4.2 Scaling the Simple Universe

An interesting feature of this program is the above mentioned branch group "objScale." During preliminary testing we found that some parts were too big for the viewing screen while others were too small. After looking at the STL files of our examples, we realized that some models would have to be scaled down because of a difference of units.

Java3D's default units are in meters. Since the vertices recorded in an STL file are recorded in the units of the CAD program in which they were originally designed, we found many discrepancies of units. Parts made in millimeters were drawn well, while parts drawn in inches were too large and exceeded the defined bounding sphere. In order to solve this problem, we incorporated an algorithm to automatically scale the view of the universe so that the user would be able to view any object they want.

This scaling algorithm is integrated into the readFile() method described in Section 2.2. As each vertex is read, it is compared to the stored (absolute) value of the largest vertex encountered thus far. If the vertex is larger than the one previously stored, the vertex replaces the old vertex (variable name: "bigpoint"). This value is then passed to the

createScene() method where the Simple Universe is defined. The “bigpoint” is then used to appropriately scale the universe. If it is relatively large, the universe is scaled to 0.0075 of its normal value. If it is relatively small, it is scaled up to 1.5.

```
if (bigpoint < 0.6f) {t3d.setScale(1.5);}
else if ((bigpoint > 0.59f) && (bigpoint < 0.9))
   {t3d.setScale(1.0);}
else if ((bigpoint > 0.89f) && (bigpoint < 1.3f))
   {t3d.setScale(0.5);}
else if ((bigpoint > 1.29f) && (bigpoint < 2.5f))
   {t3d.setScale(0.02);}
else { t3d.setScale(0.0075); }
```

It must be noted that this algorithm was based on the assumption that this program would be used to view models meant for the rapid prototyping. This algorithm, therefore, is therefore does not accurately scale very large STL parts.

Now that our Simple Universe is ready for the solid, we are able to discuss how to actually display the solid on the 3D canvas.

2.4.3 Drawing the Solid Model

As mentioned in Section 2.1.1, the user can choose how he/she would like to view the solid: in a rendered format, or a triangulated wireframe format. These two different visualization features are shown in Figure 6.

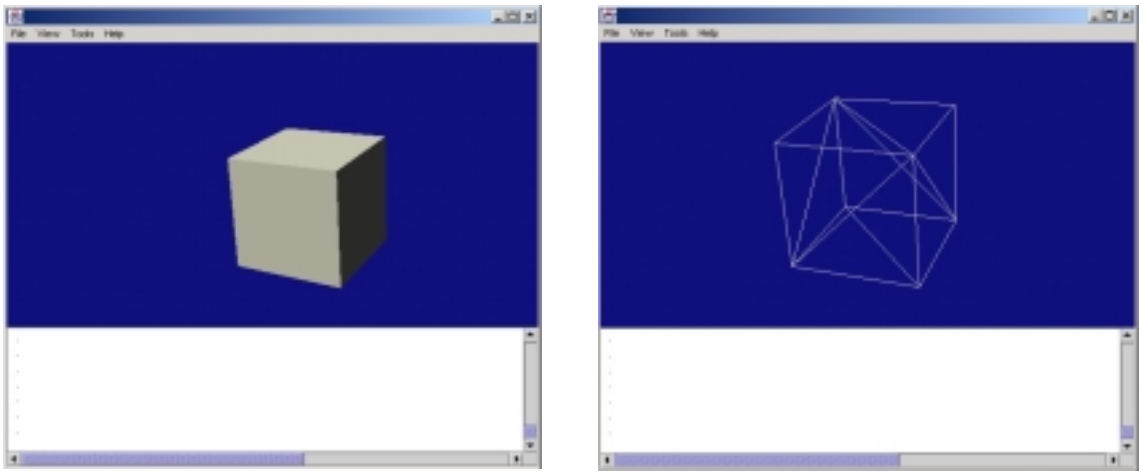


Figure 6: Rendered and Triangulated Cube

The algorithms to display these two renditions of the solid model are surprisingly not complicated. Java3D's built in methods enable the user to quickly define lines and triangles. Both of these algorithms take the same input from the readFile() method: an array of normals (stored as a Vector3f array) and an array of coordinates (stored in a Point3d array).

Rendered

The integer flag, rendercheck, tells the render() method which drawing algorithm to use. If rendercheck is 1, the rendering algorithm is run. Amazingly enough, the rendering algorithm is as simple as what follows:

```
if (rendercheck == 1){
 TriangleArray ta = new TriangleArray(numverts,
TriangleArray.COORDINATES | TriangleArray.NORMALS);
  ta.setCoordinates(0, tricoords);
  ta.setNormals(0, normals);
  part = new Shape3D(ta);
}  //end of if rendered
```

The solid model is represented by the Java3D class, Shape3D ("part"). The Shape3D is constructed by passing it a TriangleArray object of the GeometryArray Class.

A TriangleArray coordinate array ("tricoords") contains the list of vertices read from the STL file. The TriangleArray class takes three vertices at a time from the "tricoord" array and automatically constructs a triangle with them. This is illustrated in Figure 7.

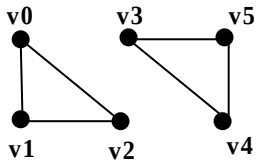


Figure 7: TriangleArray Coordinate Manipulation

The normals of each vertex of the TriangleArray are defined by the array of normals ("normals") read from the readFile() function.

This is the clear benefit of programming graphics with Java3D: the ability to quickly and easily define geometry.

Triangulated

When users choose to display their part as a triangulated wireframe, the following algorithm is followed:

```
else if (rendercheck == 0) {
  int newnumverts = (numverts * 2);
  LineArray la = new LineArray(newnumverts, LineArray.COORDINATES);
  //fill array using coordinates
  int i = 0;    //line array counter
  int j = 0;    //triangle array counter
  while (i < newnumverts) {
    la.setCoordinate(i, tricoords[j]);
    la.setCoordinate(i+1, tricoords[j+1]);
    la.setCoordinate(i+2, tricoords[j+1]);
    la.setCoordinate(i+3, tricoords[j+2]);
    la.setCoordinate(i+4, tricoords[j+2]);
    la.setCoordinate(i+5, tricoords[j]);
    i = i + 6;
    j = j + 3;
  }
  part = new Shape3D(la);
} //end of elseif Triangulated
```

This algorithm uses the same arrays as the rendered function: normals and tricoords. The main difference between the “rendered” and “triangulated” functions is the type of GeometryArray. While the rendered algorithm uses a TriangleArray, the triangulated algorithm uses a LineArray.

The LineArray class takes a pair of vertices from the array list and simply connects them with a line. This is illustrated in Figure 8.

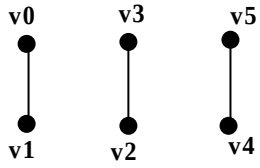


Figure 8: LineArray Coordinate Manipulation

Because of this, it now takes six vertices to draw a triangle (start and end point for each line). Therefore, the first step of the algorithm is to multiply the number of vertices ("numverts") by two. The algorithm listed above simply steps through the "tricoords" array and reconfigures the model's coordinates appropriately and passes them to the LineArray.

To draw two triangles with the coordinates above like we did with the Triangle Array, the coordinate array would consist of:

LineArray[0]	v0
LineArray[1]	v1
LineArray[2]	v1
LineArray[3]	v2
LineArray[4]	v2
LineArray[5]	V0
LineArray[6]	v3
LineArray[7]	v4
LineArray[8]	v4
LineArray[9]	v5
LineArray[10]	v5
LineArray[11]	v3

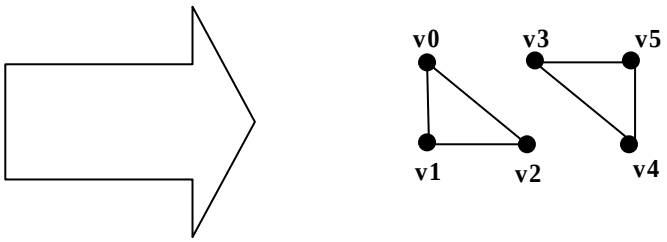


Figure 9: LineArray Coordinate Example

Although it increases the size of the working array (and makes the vertex redundancy problem worse), this is the simplest way to draw a wireframe of the object.

2.5 VALIDATION TOOLS

One of the focuses of this project was to identify the common problems of STL files listed in Section 1.1.3. For this reason, we developed a set of tools for identifying errors in the STL files that are passed through the STLViewer.

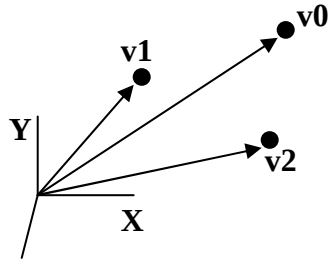
These commands were briefly described in Section 2.1 - GUI Design. Once the solid is loaded, the user has the ability to compare the normals given by the STL file to those calculated from the given vertex information. Choosing "Redraw New Normals" provides the user with a visual validation of the model as it is re-represented with the newly calculated normals.

2.5.1 Normal Check Function

The first step of the "Normal Check" function is to calculate the new normal for each triangle. This is accomplished using the three vertices of each triangle.

At first glance it may seem impossible to determine the triangle's true normal from a list of its three vertices. The STL file has a special feature, however, that enabled us to do just that. STL files list the vertices of each triangle in a counter-clockwise fashion. The normal can therefore be calculated by simply taking the cross product of the two vectors connecting the first vertex to the second and third vertices.

For each triangle of the model, three vectors ("pt1," "pt2," and "pt3") are drawn from the origin to its three vertices. These vectors are constructed with the vertex information received from the readFile() function.

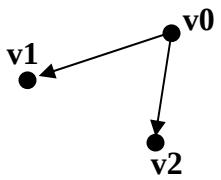



```
while (i < numverts) {
      Vector3d pt1 = new
Vector3d(tricoords[i++]);
      Vector3d pt2 = new Vector3d(tricoords[i++]);
```

```
Vector3d pt3 = new Vector3d(tricoords[i++]);
```

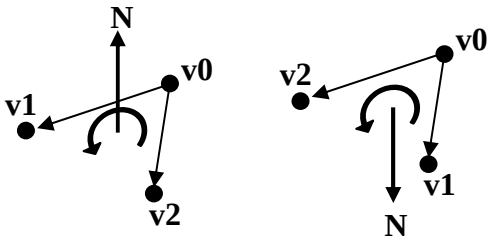
These vectors are subtracted to create two vectors (“test1” and “test2”) originating from “pt1.”

```
Vector3d test1 = new Vector3d();
test1.sub(pt2, pt1);
Vector3d test2 = new Vector3d();
test2.sub(pt3, pt1);
```



The cross product of these two vectors is then calculated.

```
Vector3d cross = new Vector3d();
cross.cross(test1, test2);
cross.normalize();  //normalize the normal
crossf[k++] = new Vector3f(cross);
crossf[k++] = new Vector3f(cross);
crossf[k++] = new Vector3f(cross);
```



The two normals are then compared.

```
if (crossf[k-1] == normals[j]) { ; }
else {
  textarea.append("*** Normals don't match!! ***" + "\n");
  textarea.append(" Given vector value:  " + normals[j] + "\n");
  textarea.append(" Calculated vector value:  " + crossf[k-1] +
                                                        "\n");
  errorcount++;
}
j = j+3;
} //end of while
```

This algorithm will provide a warning in the text area for the user if there is some sort of discrepancy between the two normals. For a more visual interpretation of the differences, the “Redraw New Normals” function is available to the user.

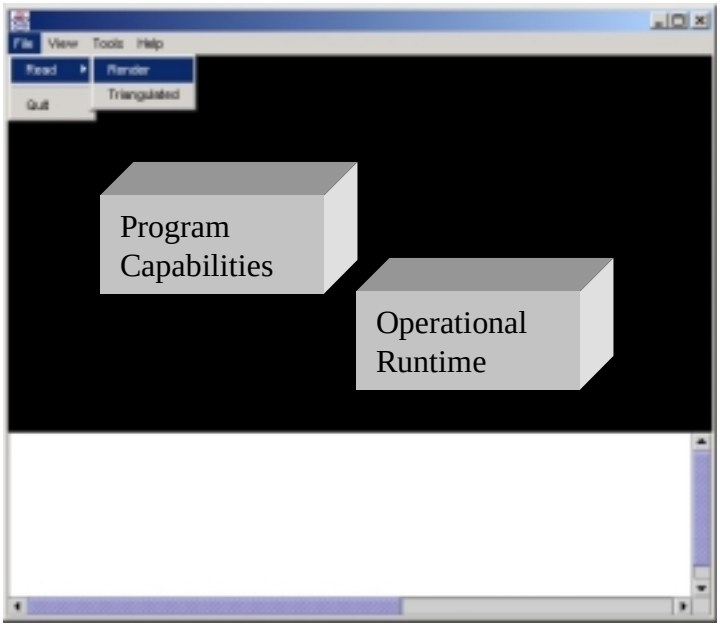
2.5.2 Redraw with New Normals Function

This method is very similar to the draw method as described in Section 2.3.3 – Drawing the Solid Model. The only difference between the two is that this method passes the newly calculated normals to the TriangleArray instead of those read by the readFile() function.

The “crossf” array contains all of the new normals. Coupled with the vertex coordinate array (“tricoords”), the solid model is quickly redrawn.

CHAPTER 3 RESULTS

We discuss the results of our programming in this chapter. The capabilities and limitations of each module of the software are discussed. Multiple screendumps of the software are included as examples of results from the program. The chapter concludes with a discussion of the amount of time needed to read and display a solid.



3.1 STLVIEWER'S CAPABILITIES

3.1.1 GUI

As expected, the GUI is rather easy to use. Once the website is accessed, the frame containing the program pops open for the user to begin viewing. The instructions prove to be a sufficient guide to the program.

The menubar works well and provides easy access to all the necessary commands for the user. The 3D Canvas displays the solids very well and allows the user to rotate, pan, and zoom the object just as expected. The text area also works very well. The text area can take any amount of input at once (limited only by the computer's CPU). Messages are also displayed to the text area to assist the user.

The pasting of the code seems to be the only tedious part of the GUI. Due to the lack of a "Paste" menu-item, or a context-sensitive menu (accessed by the right-mouse button), the user is forced to use "Control – V" as their paste command. Luckily, the text area is very similar to Window's Notepad – commands like "ctrl-v" (Paste), "ctrl-a" (Select All), "ctrl-c" (Copy), and "ctrl-x" (Cut) work. Although the pasting process is described once the "Help -> Instructions" menu item is chosen, many users may not be familiar with this command.

3.1.2 File Read Function

The function that parses the files works extremely well. It has had no problems with any STL file passed to it. It efficiently and effectively parses the STL file for the two keywords "normal" and "vertex." Once these keywords are found, the algorithm stores the corresponding float values as coordinates or normals in appropriate arrays to be passed to the Draw function.

The file read function sometimes takes a few seconds to finish for larger files. Since ASCII files can be very large, it takes a while for the algorithm to parse the file. This is described in more detail in Section 3.2 – Operational Run-time.

While this algorithm can read any ASCII STL file presented to it, it does have a limitation: the design of the file reading function is not robust. If the standard for the STL file format changes in the future the file reader will not run. This is only true, however, if lines involving the keywords ("vertex" and "normal") are changed. For example, if the keyword "vertex" is followed by three spaces instead of one, the function will not recognize the vertex's coordinates. From this we recognize the importance of keeping file formats standard on each and every CAD package.

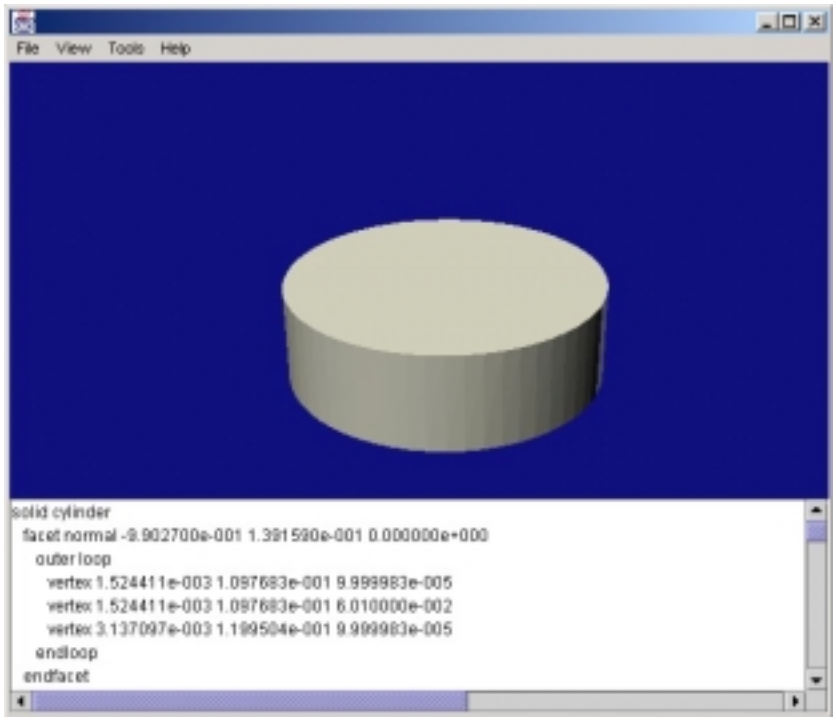
After the demonstration for Dr. Rosen, we realized the severity of this lack of robustness. For this reason, we are currently attempting to fix this problem. We will substitute the dependence of "white space" with a search for the beginning of new data. This will ensure that all STL files will be readable by our program.

While this lack of robustness is a flaw of the program, it must be noted that it is very easy to update the code so that it will work with a new file format. One can either update the STL file to have appropriate spacing (or fix any other errors), or update the actual code to incorporate the new standards.

3.1.3 Draw File Function

Once the file is pasted into the text area, the user must choose which visualization of the model they would like to view – render or triangulated. Both methods have proven to display the model exactly as the STL file dictates.

Figure 10 shows a simple cylinder represented in both its rendered and wireframe format.



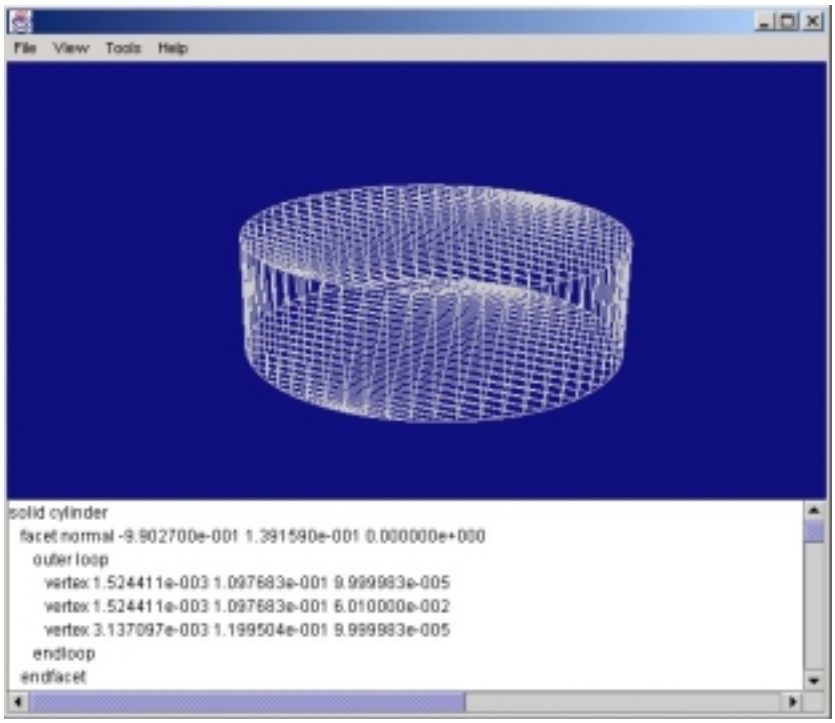
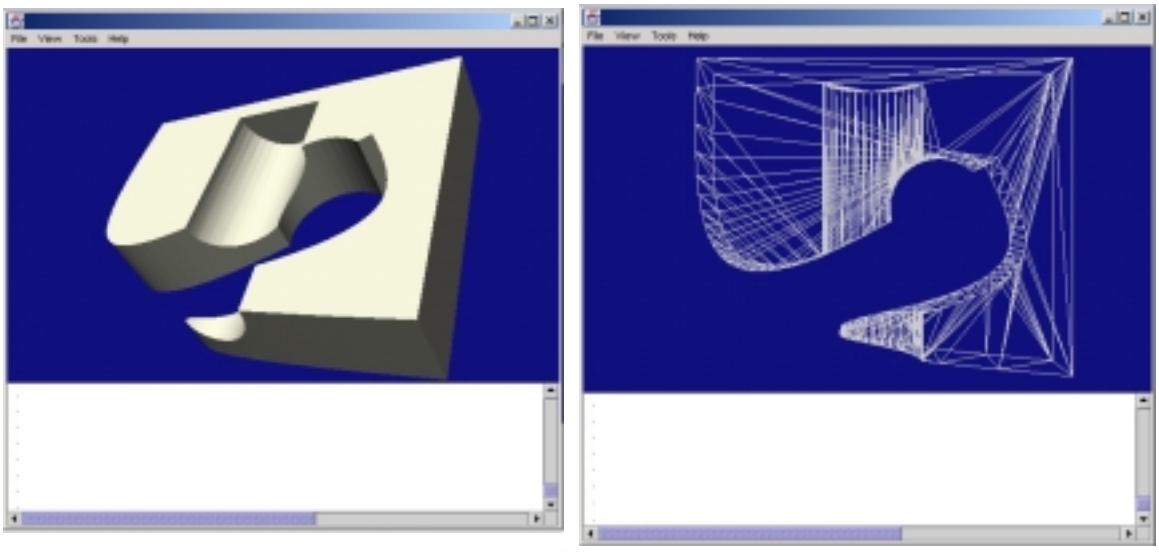


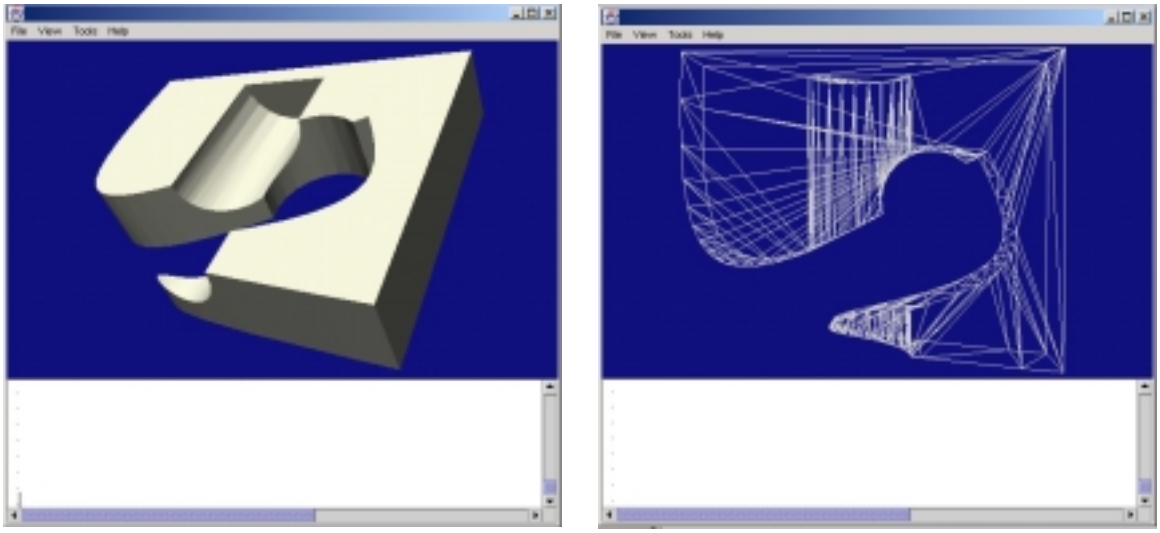
Figure 10: Cylinder in Rendered and Triangulated Format

The triangulated version of the model provides the user with a visual verification of the part. It can also provide insight into how accurate the STL file is. Are the curves accurate enough? Are they represented by too few or too many lines? These questions can be answered from a simple investigation of the triangulated solid model.

Figure 11 shows a part we named “obscure shape.” Its profile consists of a couple of splines, and a cylinder has been subtracted (boolean wise) from the solid. This part was created with SolidWorks and was translated twice – once with “Coarse Triangulation” chosen, and once with the “Fine Triangulation” option chosen in the “STL Options” menu. Figure 11 shows the two different versions of the obscure part in both their rendered and their triangulated formats.



(a)



(b)

Figure 11: Complex Shape in "Fine" (a) and "Coarse" (b) Triangulation

As can be seen in the above figure, the user can easily identify differences between the two modes of triangulation with the combined use of the render and triangulated models. It is obvious from looking at the two versions that the “Coarse” model approximates curves with fewer line segments. With this tool, the user is able to visually judge if this approximation is accurate enough for his/her needs.

This program can read and draw any STL file. As can been from the complex models of Figure 12, as long as the CAD model appropriately converts the solid model to STL, our program will be able to view it.

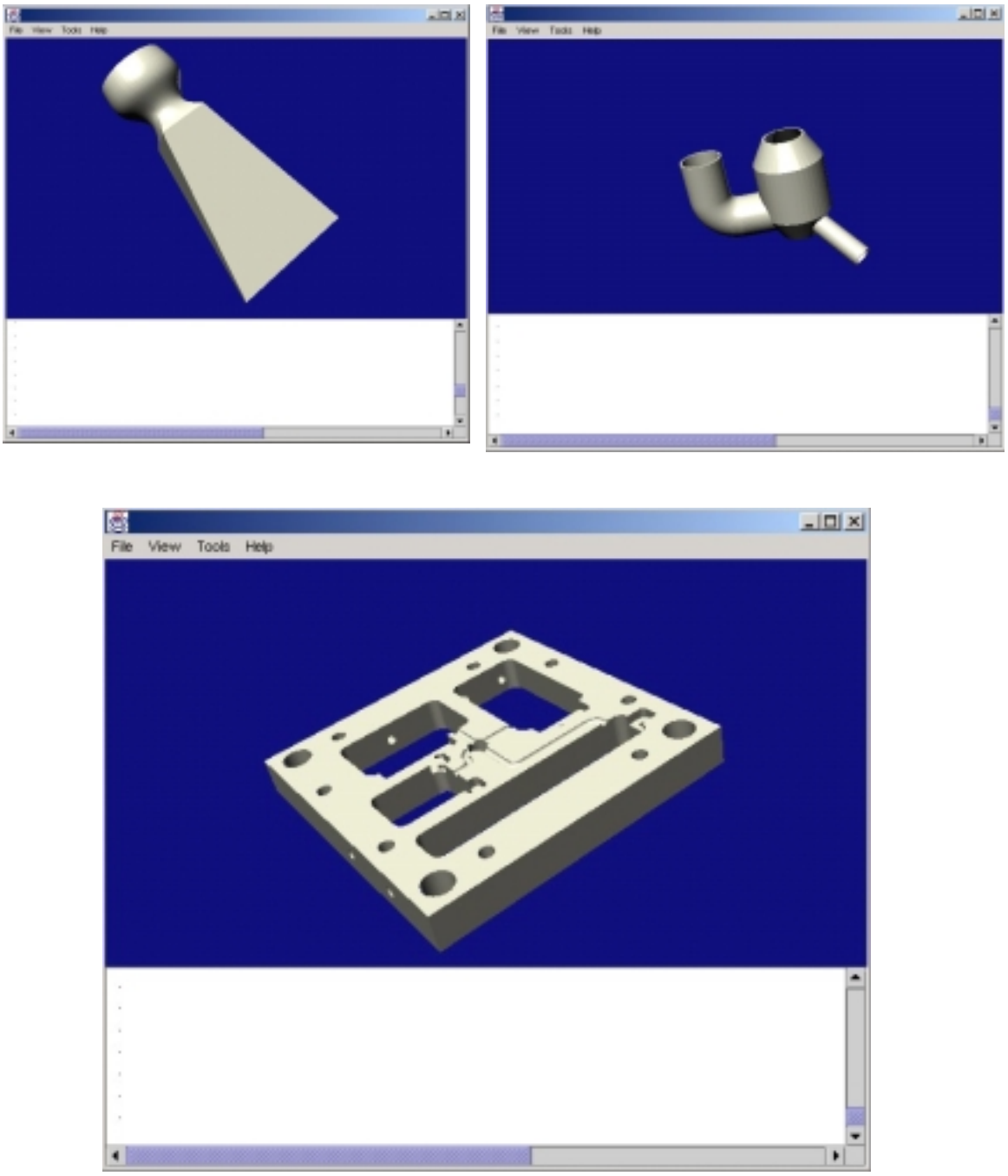


Figure 12: Complex Shapes Visualized by STLViewer

3.1.4 Initializing the Simple Universe: Scaling & Manipulating the Model

The feature that provided the most trouble in programming was the auto-scaling feature. For the majority of models loaded into the STLViewer, this portion of the program works well. For other models, however, this feature does not work as well as some parts are too small or run off the screen.

As described in Section 2.4.2 the auto-scaling function scales the view area, using the vertex that is furthest from the global origin ("bigpoint") as a reference.

```
if (bigpoint < 0.6f) {t3d.setScale(1.5);}
else if ((bigpoint > 0.59f) && (bigpoint < 0.9))
   {t3d.setScale(1.0);}
else if ((bigpoint > 0.89f) && (bigpoint < 1.3f))
   {t3d.setScale(0.5);}
else if ((bigpoint > 1.29f) && (bigpoint < 2.5f))
   {t3d.setScale(0.02);}
else { t3d.setScale(0.0075); }
```

As can be seen in the above code, certain ranges are provided for scaling. This is obviously not the most robust method of scaling. More ranges could have been added, but these ranges seemed to do well for most sizes. It must be noted, however, that this program is meant for parts meant for the SLA process. Therefore, the program should not be used for very large parts (ex. whole car bodies, etc.).

Other problems were encountered when models were very poorly scaled. If a part appears very small on the 3D Canvas, it becomes very difficult to zoom in on the part. With most parts, zooming, rotating, and panning occur very quickly with no lag in user input. With parts that are very far away from the view screen, the user may become discouraged with the slow nature of the zooming feature.

The auto-scaling feature works very well overall. For all examples passed to it, the STLViewer handles the scaling effectively. Figure 13 contains two parts: a part meant to test the micro-SLA process (width of 5 mm), and a light switch cover plate (length of 5

inches). As can be seen, the auto-scaling feature did a good job with both parts despite their differences in sizes.

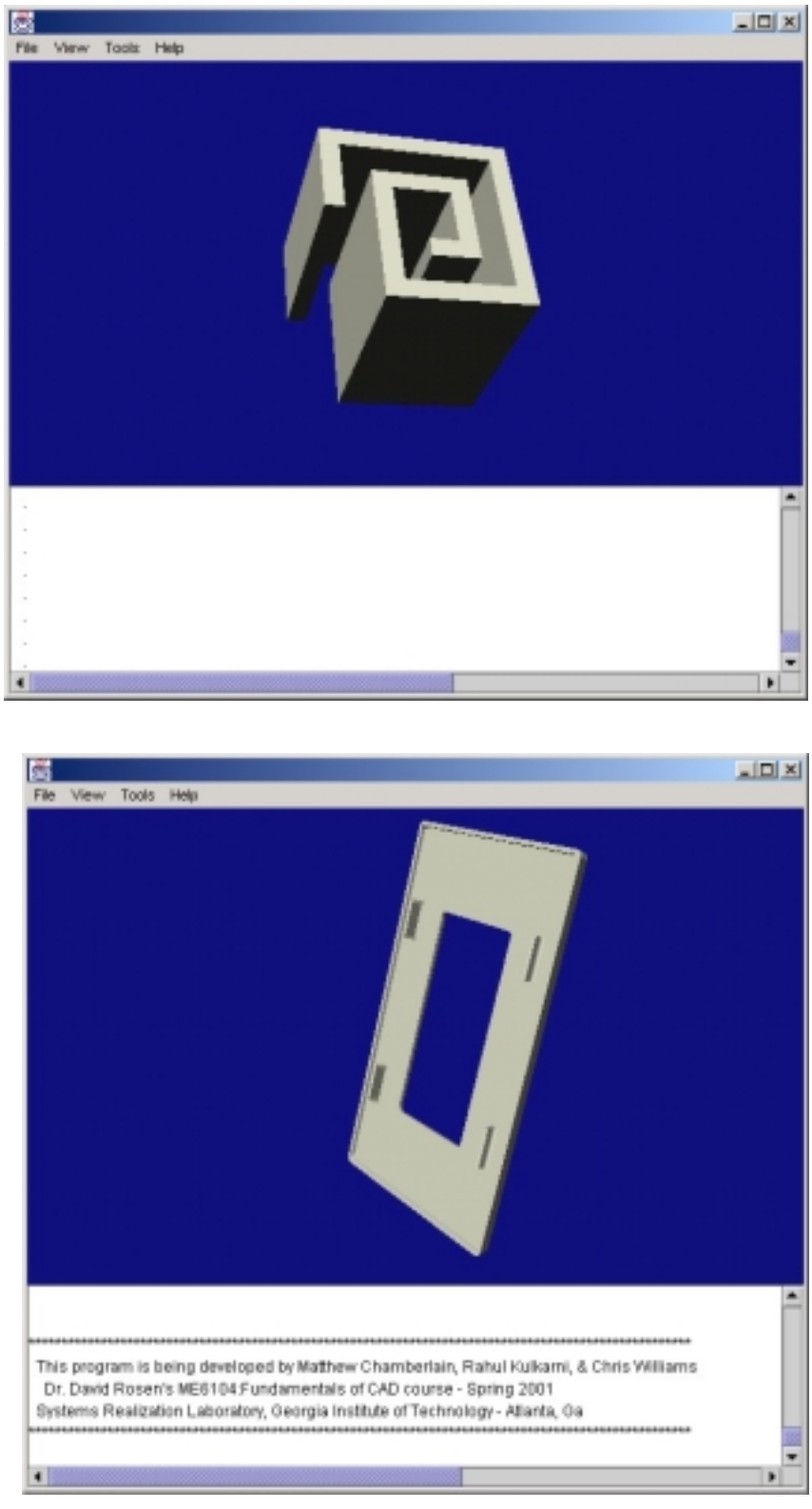


Figure 13: Examples of the Robustness of the Auto-Scaling Feature

Manipulating the solid, overall, also shows no faults. Although panning and zooming in on the solid can be very tedious if the part is incorrectly scaled, there are really no other difficulties due to the code. The user may be discouraged, however, when the part is not centered about the Java3D origin. The coordinates received from the STL file are directly translated to the origin of the Simple Universe. This can cause problems, however, if the part is far from the origin. Instead of simply rotating about its own center, the part will rotate around the origin and will orbit the origin like a satellite. Because of this, it can be sometimes difficult to properly rotate the model. Ideally, the centroid of the part would be located at the origin. It should be noted, however, that many CAD models automatically translate the solid to the origin. As a result, one vertex of the solid is attached to the origin. Thus, the solid is not usually very difficult to rotate.

3.1.5 Extra Tools: Normal Check & Redraw with New Normals

As described in Section 2.5, these extra tools are provided for the experienced user. Once chosen, the normals provided by the STL file are compared with normals calculated through manipulations of the triangle's coordinates. The solid model can then be redrawn with the new normals for a visual check of the approximation.

These two validation tools, although limited in function, work well. The following figure, a close-up of the obscure part of Figure 11, shows the effects of these tools. Figure 14a shows the part as it was originally drawn. The text area shows the results of the normal check. Figure 14b shows the part redrawn with the newly calculated normals.

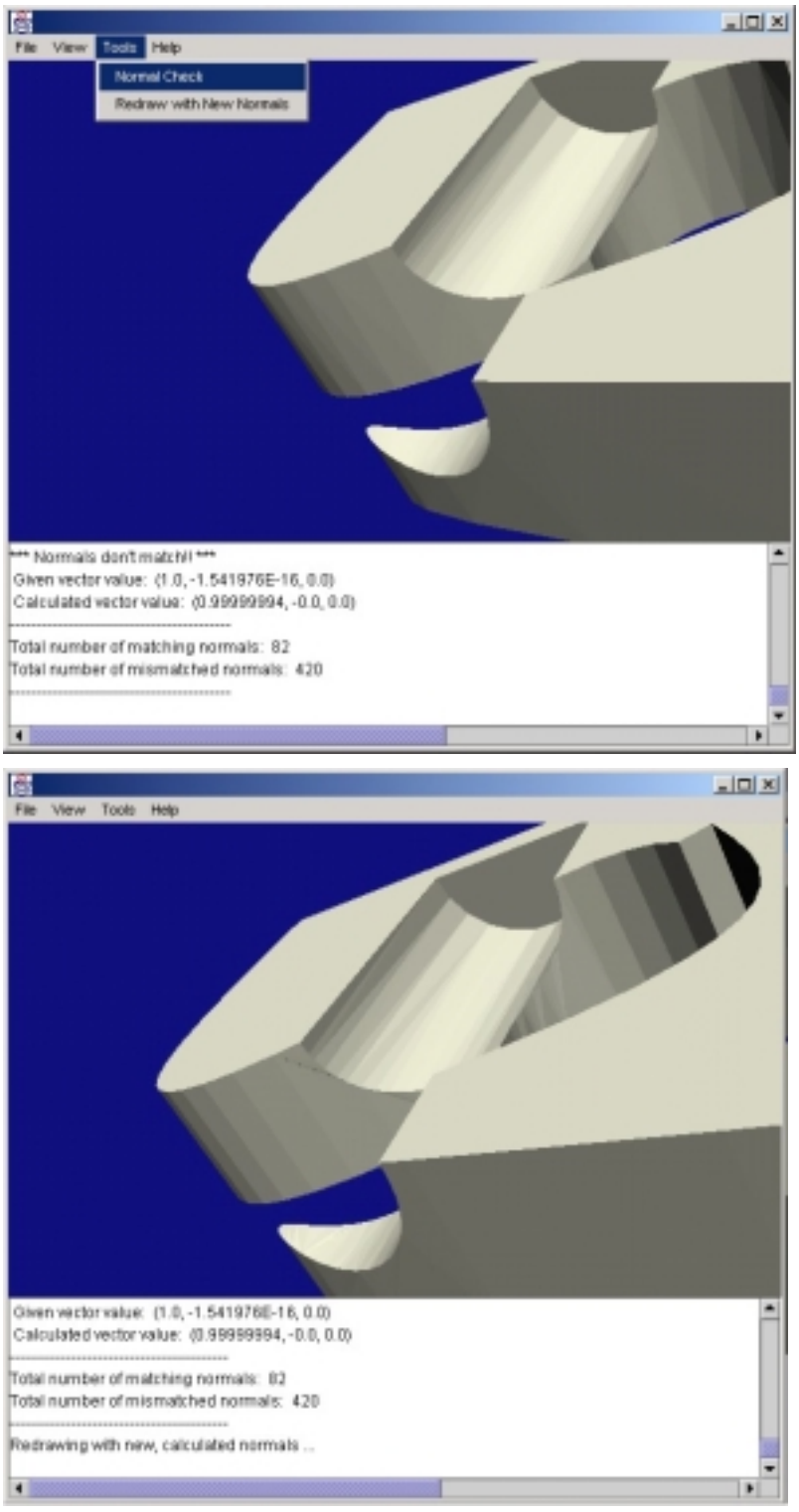


Figure 14: Normal Check & Normal Redraw Effects on Obscure Part

As can be seen from the figure, there are definite visual differences between the two models. The redrawn version has visual continuity issues.

3.2 OPERATIONAL RUN-TIME

As stated in the earlier sections of this chapter, the program has a tendency of loading and manipulating the solid quite slowly if it is very large. The large size of ASCII STL files is the obvious cause of these hiccups. The table below lists the load times of example STL files as compared to their file size. The table also includes a column for load times of the same files in SolidViewerPro as a benchmark.

File Size	STLViewer	SolidViewerPro
19 kB	instantaneous	instantaneous
218 kB	0.5 sec.	instantaneous
488 kB	4 sec.	2 sec.
1.462 MB	13 sec.	3 sec.
1.563 MB	16 sec.	4 sec.
2.579 MB	16 sec.	5 sec.

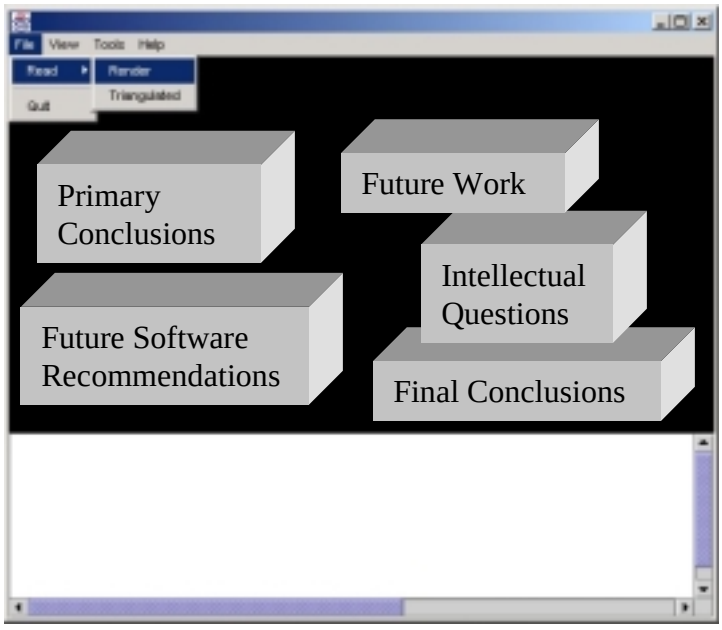
Figure 15: Run Time Comparisons

As can be expected, the STLViewer runs a few seconds behind SolidViewerPro. It must be noted that comparing these two programs is narrow-minded. SolidViewerPro has the benefit of running of the users CPU directly, whereas STLViewer is running through the user's web-browser.

While the wait for the model to be read is not terribly bad, one wonders whether or not it is due to a performance limitation of Java. It seems, however, that it is due to the tedious reading method. Having to pass the file through a text area is definitely less efficient as passing it through a standard I/O method as a file dialog box.

CHAPTER 4 CONCLUSIONS

The paper concludes with a reflection on the software. Primary conclusions are presented as a list of benefits and limitations of the software. Ideas for future work are discussed as well. Through the development of this Internet-based program, we gained insight into the future of on-line CAD software. These insights are shared in the Future Software Recommendations section as a requirements list. The course's intellectual questions are also answered in the context of the project. The "Final Conclusions" section provides a summary of the learning obtained from working on this project.



4.1 PRIMARY CONCLUSIONS

From the results noted in Chapter 3, we are able to draw a few primary conclusions on the performance of the program.

Overall, this program works very well. It has achieved all of the goals set for it:

- It effectively reads STL files.
- It effectively displays STL files.
- Users are able to rotate, zoom, and pan the models.
- It runs over the Internet in a distributed design environment.
- It can display the STL file in a rendered or triangulated wireframe format.
- It has a few features that enable the user to visually confirm the validity of the model.

Of course, with all programs, there are a few limitations as outline in this chapter:

- The file reading function is limited by the user's CPU power.
- The dependence of ASCII files limits the program's performance, as ASCII STL files are inherently large.
- The user can only paste text in the text area with "Ctrl-V."
- The file reading function is not very robust. If file format changes, it will no longer work.
- The auto-scaling feature is not very robust. Although it's limited power adds robustness to the program, it has limitations that keep it from appropriately scaling all parts.
- The part is not always located around the origin. As a result, rotating the part can be difficult at times.
- The program does not load files as fast as other commercial software. This is due to the reliance on ASCII files as well as the fact that the program runs over the Internet.

4.2 FUTURE WORK

As stated in the previous section, this project has met all of its goals. The program functions as we expected and it has directed us to learning of STL files, rapid prototyping file transfer, realm of distributed computing, and the intellectual questions. This project, however, could be extended to include new features and ideas.

4.2.1 Thoroughly Check STL File

The primary purpose of this program was to develop a viewer for STL files over the Internet. We attempted to add a few extras to the program – notably, a routine for comparing normals. As discussed in the first chapter, however, many problems can occur with a STL file conversion. One idea for future work would be to increase the "Tools" menu list to include checks for gaps, missing facets, degenerate facets, and overlapping facets.

At first glance, this would be a difficult task when only provided with basic information about a facet. Many algorithms have been written and published, however, that provide easy routines for selection of common facet errors. These algorithms would have to be converted into Java and incorporated into the existing code. Luckily, the program is very modular. Any tool can be easily added to its functionality.

4.2.2 Correct the STL File

Of course, once the STL file is thoroughly checked, the program could correct the file. Again, some algorithms have been written that correct common problems with STL files.

This addition to STLViewer would make it an even more useful program. Customers of Rapid Prototyping labs could submit jobs through the Internet. The STLViewer would show the customer their part as interpreted by the STL file. The program could then alert

the user to problems and then automatically fix the file. Finally, the program could show the customer the new part for their approval. Most customers of rapid prototyped parts are only concerned with the look and the feel of the part, thus STLViewer would be a magnificent utility for them to use.

4.2.3 Increase Inherent Robustness

The overall design of STLViewer is fairly robust. As mentioned earlier, the program is modularly designed. Any new tool can be added to the menubar to provide more functionality. Adding an actionListener() for the new choice and the appropriate analysis code is all that is required when one wishes to add a new option.

As mentioned earlier, however, the readFile() function is not very robust. If the file standard changes by a few spaces, it could be disastrous for the function since it relies on spacing to correctly read in the file. The code can be easily re-written for changes in file format, but that is not a robust way of programming code. Future work on this program will entail a rewrite of the readFile() function so that it is based solely on a keyword search and not on the combination of keywords and accompanying spaces. The improvement on this function would add a new level of robustness to the readFile() function and to the program as a whole.

This work is currently underway as result of the discouraging results of the demonstration.

4.3 RECOMMENDATIONS FOR FUTURE INTERNET BASED CAD SOFTWARE

From the experience gained in designing STLViewer, we are able to extract some insight into the future of Internet-based CAD software. This insight is compiled into the requirements list below:

	Requirements List for Internet CAD - ME6104
	Problem Statement: *Develop a Internet based CAD software package to be used by distributed designers.*
D W	**Requirements**
	User Inteface:
D	- The user interface must be compatible with different PC platforms.
W	- The interface must be easily navigable by the user
D	- The inteface must not use a lot of memory to run and to navigate.
	File Format:
D	- Must be standard among all software packages.
D	- Files should be as small as possible so as to minimize load time.
D	- Files should contain as much information about the solid as possible.
W	- Different formats containing different types of information (material property, surface finish, coordinate information) could be used to minimize file size.
	Internet Security:
D	- Security protocol must be reconfigured so that complex information can be trusted when shared
W	- CAD Intranet memberships may provide secure CAD developing shelters
	Data Exchange:
D	- Data exchange should not be limited by security violations
D	- Data exchange should not be limited by file size
	Analysis Packages:
D	- Anaylsis code should be run off of the user's CPU to minimze calculation time
D	- Results should be easly uploaded to the core CAD software located on the server
W	- Updates to analyiss packages should be easily retrievable and added to existing framework.
D	- Internet-based CAD program must be robust enough to handle updates

4.4 ANSWERS TO INTELLECTUAL QUESTIONS

Our insight into these questions comes from our experience with the STL file format. We achieved an understanding of how CAD software (as well as our own software) communicates to this file format. This is the context, therefore, in which the three intellectual questions will be answered.

4.4.1 Why are components shaped the way they are?

Referring back to our working context, it should be obvious that, at first, we have no basis for an answer to this question on a concrete level. We did not design any parts and have no immediate insight into why components are shaped the way they are. The question can be restated, however, to apply more to our project without losing its importance. *Why are STL files translated the way they are?*

STL files, as any other exported solid model file format, are just approximations of how the model was originally designed. Using STLViewer, the user is able to visually verify the accuracy of the approximation. Multiple straight lines now represent curves. Small cracks develop due to sudden changes in curvature. Tessellation errors cause convex boundaries to flatten.

Dave Albert of the Albert Consulting Group created the STL format in 1987. He chose to use the facet approach because you can create a faceted representation from most CAD systems that offer surfacing and solid modeling (Wohlers 1992). Triangles are the simplest polygons and can closely approximate all surfaces. Two triangles are needed for a rectangle, while curved surfaces can be closely approximated with many small triangles.

STL files are translated the way they are because of their application. Used primarily as file transfer file format from the CAD package to the SLA machine, STL files only need

surface information. STL files do not need information on material properties or parameterization strategies. They only require information on the vertices and normals of each facet covering its face. This simple requirement is probably the reason for its incredibly inefficient development.

4.4.2 How are component shapes described to CAD systems?

Our work with the STL file format has provided us with a great opportunity to answer this question. Through this project we are working on a very standardized file format that is both read and written by many commercial CAD software packages.

For rapid prototyping, the STL file is the *de facto* file format. It consists of a list of triangular facets, which provide a wireframe for the solid's surface. This list contains the three vertices as well as the normal for each facet.

The STL file's description of a component is very limited and inefficient in this manner. ASCII STL files are very large due to the repeated vertices and extra normal information. The STL file's size could be reduced by 25% if each line describing a facet normal was left out. The file size could also be reduced if each vertex of each facet wasn't repeated each time was shared with another facet (ex. a single vertex on a cube is referenced six times).

What is missing from the STL file format is edge, face, and body information. Listing facets is sufficient for making a triangular surface mesh. This is not efficient, however, if more information is needed about the solid. For comparison, a typical ACIS ".sat" file is 40kB. This file format contains all major coordinate information and also contains information about faces, shells, edges, coedges, and curve type. Usually, components are described to CAD systems through a series of pointers to vertex, edge, face, etc. information. This structure allows the CAD package and the user to step through the solid model in order to extract a wide-range of information as efficiently as possible.

The STL file format, however, only provides information for its triangular mesh. This information is useless to many CAD packages. They can use it to redraw the model, but no editing may be done. In fact some CAD packages only export STL files; they do not read them.

From our work in developing this visualization tool, we realized the power of a standardized file format. The STL is a very standardized file format because of the extreme growth of the rapid prototyping industry. Components must be described to CAD systems in a standardized fashion, otherwise the predicted distributed design environment will not exist.

4.4.3 How can a top-down, product-wide approach to CAD modeling work?

This is a difficult answer for us to answer in the context of our project. Our project did not directly lead to insight into an answer for this question as say, a project about product family design would have. Therefore, this question needs to be augmented to better fit our context. *How can a Internet-based approach to CAD modeling work?*

There are four major components of CAD programs:

1. User Interface
2. Solid Modeling Engine
3. Engineering Calculation Modules
4. Data Interchange Module

We present a brief overview of the current implementations of each of these components in commercial CAD systems and also on our vision of the future CAD systems.

User Interface:

The user interface consists of the main screen where the CAD model of the part under construction is displayed. Different tools that are used for object creation are also displayed. The user can click on the different tool buttons and work on the CAD

model. This is the standard form of user interface adapted by all commercial CAD systems. Variation occurs in terms of where each tool bar goes or how customizable is the user interface.

In the future we believe a user would design in an immersive CAD environment. A virtual reality headset would be a part of every computer system and a user would be able to visualize the part in 3D. Commands can be given to the CAD system through voice. For specialized applications there could be devices that would enable the designer to design easily. For example, a mouse kind of device that could be bent would be ideal for sheet metal bending applications.

Solid Modeling Engine

This is the most crucial part of the CAD system. Different commercial CAD applications use different solid modeling techniques like B-Rep, CSG etc. Each of these have their own distinct set of advantages and disadvantages.

Solid Modeling engines of the future would have better integration between these different forms of representation. We feel a volumetric solid modeler would be a better way of modeling solids rather than the current B-Rep modelers.

Solid Modeling engines of the future need not be loaded on the same machine as the one the user is working on. They could be run remotely via the Internet, thus eliminating the need for loading heavy CAD applications on a single computer.

Engineering Analysis Modules

These are add-on modules to the CAD system which enable different kinds of analysis performed on the CAD model generated. Examples of such modules include stress analysis, simulation, interference check etc.

In today's CAD systems, each commercial CAD system has their own set of modules for engineering analysis. We believe in the future each of these modules would be interoperable. So there would not be three different stress analysis modules by

different companies containing the same functions, but there could be three stress analysis modules with different functions, that can be combined into one on the fly. These modules could be loaded on different computers across the globe and could be used remotely using remote invocation technologies.

Data Interchange Module

The Data Interchange Module is the one that is responsible for storing the CAD model created in a format that the CAD system can understand. Each of today's CAD system has its own file format. Hence it becomes very difficult for a designer to use different CAD systems to design the same CAD model.

Lately we have seen the STEP standard evolving as the standard for representing CAD data. However we feel in the future instead of a common format, there would be formats based on XML. Hence whatever be the order of information in the file, using the appropriate tags and document type definitions, a CAD system would be able to retrieve information from the file.

4.5 FINAL CONCLUSIONS

During the course of the semester, we had a sudden change of plans for the project. We dropped an attempt to merge this project with another class's project and found ourselves with few options for available projects. We are happy and satisfied with our selection of this project because:

The program met our overall goals:

- It effectively displays STL solids
- The user is able to fully visually inspect the model
- It is able to run over the Internet with no security violations
- It incorporated a small set of tools to enable the user to visually and mathematically identify potential errors.

We learned about:

- We learned about and gained experience with the Java programming language and its new 3D API.
- We learned about Java security restrictions.
- We learned about the STL file format and its common issues.
- We gained experience with a variety of CAD packages in the design and the export of different examples.
- We learned about file transfer in the rapid prototyping process.

We gained insight into the course's intellectual questions.

- We gained insight into how solid models are decomposed into a triangulated surface mesh.
- We learned about the importance of file standardization.
- We gained insight into the future of Internet-based CAD software.
- We achieved an understanding of the STL file format and how it communicates with CAD software.

References

Bailey, M. J. (1995). "Tele-Manufacturing: Rapid Prototyping on the Internet." ISEE Computer Graphics and Applications 15(6): 20-26.

Chen, Y. H., C. T. Ng and Y. Z. Wang (1999). "Generation of an STL File from 3D Measurement Data with User-Controlled Data Reduction." International Journal of Advanced Manufacturing Technology 15(2): 127-131.

Fadel, G. M. and C. Kirschman (1996). Accuracy Issues in CAD to RP Translations. The First Internet RP Conference: Rapid Product Development, MCB University Press.

Jacobs, P. F. (1996). Stereolithography and Other RP&M Technologies: From Rapid Prototyping to Rapid Tooling. New York, ASME Press.

Kai, C. C., G. G. K. Jacob and T. Mei (1997). "Interface Between CAD and Rapid Prototyping Systems. Part 1: A Study of Existing Interfaces." International Journal of Advanced Manufacturing Technology 13(8): 566-570.

Leong, K. F., C. K. Chua and Y. M. Ng (1996). "A Study of Stereolithography File Errors and Repair. Part 1: Generic Solution." International Journal of Advanced Manufacturing Technology 12: 407-414.

Ngoi, B. K. A., C. K. Chua, S. Ngai and F. T. L. Tay (1993). "Development of a Stereolithography Preprocessor for Model Verification." Computing & Control Engineering Journal 4(5): 218-224.

Stroud, I. and P. C. Xirouchakis (2000). "STL and Extensions." Advances in Engineering Software 31(2): 83-95.

Wohlers, T. (1992). "CAD Meets Rapid Prototyping." Computer-Aided Engineering 11(4).

Appendix A: STLViewer Code

This appendix contains the code written for STLViewer. STLViewer.java is the main program. From it the applet is initialized, the GUI is defined, the file is read, and the vertex and coordinate arrays are filled. These arrays are then passed to Draw.java where the 3D Canvas is initialized and the model is drawn.

```
//*********************************************************************//
// An applet with a frame or an application with pulldown menus      //
// that reads .stl files                                             //
//*********************************************************************//

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.text.*;
import java.util.Properties;
import java.lang.String.*;
import javax.vecmath.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.lang.Math.*;

public class Stlviewer extends Applet implements ActionListener {


/*********************************************************************/
  // Variables

/*********************************************************************/
  Frame f;
  JTextArea textarea;                  // Message window
  BufferedReader input;
  BufferedWriter output;
  Draw draw;
  String currentfile;
  int rendercheck;
  int numverts;
  float bigpoint;
  Point3f tricoords[];
  Vector3f normals[];
  Vector3f crossf[];


/*********************************************************************/
  // Member function:    Main()

/*********************************************************************/
  public static void main(String[] args) {
      Stlviewer stl = new Stlviewer();
      stl.create_frame() ;
  } //end of main


/*********************************************************************/
  // Member function:    Init()

/*********************************************************************/
  public void init() {
      Stlviewer stl = new Stlviewer();
      stl.create_frame() ;
```

```
  } //end of init()

/*************************************************************************/
  // Member function:    CreateFrameFunction()

/*************************************************************************/
  protected void create_frame() {

    f = new Frame();                          // Create a window

    // Create Java 3D object Draw
    draw = new Draw();
    f.add(draw, "Center");

    // Create Jtextarea
    textarea = new JTextArea(9, 90);
    textarea.setEditable(true);
    JScrollPane scroll = new JScrollPane(textarea);
    f.add(scroll, "South");

    // Create a menubar and tell the frame about it
    MenuBar menubar = new MenuBar();
    f.setMenuBar(menubar);

    // Create three pulldown menus for the menubar
    Menu file = new Menu("File");
    Menu view = new Menu("View");
    Menu tools = new Menu("Tools");
    Menu help = new Menu("Help");

    // Add the menus to the bar, and treat Help menu specially.
    menubar.add(file);
    menubar.add(view);
    menubar.add(tools);
    menubar.add(help);
    menubar.setHelpMenu(help);

    // Add menu items
    Menu read = new Menu("Read");
    MenuItem render = new MenuItem("Render");
    render.addActionListener(this);
    render.setActionCommand("render");
    read.add(render);
    MenuItem triangle = new MenuItem("Triangulated");
    triangle.addActionListener(this);
    triangle.setActionCommand("triangle");
    read.add(triangle);

    file.add(read);
    file.addSeparator();      // Put a separator in the menu
    MenuItem quit = new MenuItem("Quit");
    quit.addActionListener(this);
    quit.setActionCommand("quit");
    file.add(quit);

    MenuItem clear = new MenuItem("Clear");
```

```
    clear.addActionListener(this);
    clear.setActionCommand("clear");
    view.add(clear);

    MenuItem normcheck = new MenuItem("Normal Check");
    normcheck.addActionListener(this);
    normcheck.setActionCommand("normcheck");
    tools.add(normcheck);
    MenuItem normdraw = new MenuItem("Redraw with New Normals");
    normdraw.addActionListener(this);
    normdraw.setActionCommand("normdraw");
    tools.add(normdraw);

    MenuItem inst = new MenuItem("Instructions");
    inst.addActionListener(this);
    inst.setActionCommand("instructions");
    help.add(inst);
    help.addSeparator();
    MenuItem about = new MenuItem("About STLViewer");
    about.addActionListener(this);
    about.setActionCommand("about");
    help.add(about);

    // Handle window close requests
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        f.dispose();
        System.exit(0);
      }
    });

    f.setSize(600, 500);     // Set the size of the window
    f.show();                // Finally, pop the window up.

  } //end create_frame


/*****************************************************************/
  // Action Performed Interface

/*****************************************************************/
  public void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();
    if (cmd.equals("render")) {
      rendercheck = 1;
      readFile();
    }
    else if (cmd.equals("triangle")) {
      rendercheck = 0;
      readFile();
    }
    else if (cmd.equals("quit")) {
      f.dispose();
      System.exit(0);
    }
    else if (cmd.equals("clear")) {
      textarea.setText("");
```

```
      draw.detachBG();
    }
    else if (cmd.equals("normcheck")) {
      textarea.append("Checking normals ..." + "\n");
      normCheck();
    }
    else if (cmd.equals("normdraw")) {
      textarea.append("Redrawing with new, calculated normals ..." +
"\n");
      normDraw();
    }
    else if (cmd.equals("instructions")) {
      textarea.append(" Instructions:"+ "\n");
      textarea.append(" - Paste (Ctrl-v) the ASCII contents of an STL
file into the text area."+ "\n");
      textarea.append(" - Select 'Read' from the File menu."+ "\n");
      textarea.append(" Once the model appears:" + "\n");
      textarea.append(" - Drag the mouse with the left mouse button to
rotate the object."+ "\n");
      textarea.append(" - Hold the 'Alt' key while dragging the mouse
to zoom in and out."+ "\n");
      textarea.append(" - Drag the mouse with the right mouse button to
pan." + "\n");
    }
    else if (cmd.equals("about")) {

textarea.append("******************************************************
*********************************************" + "\n");
      textarea.append("  This program is being developed by Matthew
Chamberlain, Rahul Kulkarni, & Chris Williams" + "\n");
      textarea.append("    Dr. David Rosen's ME6104:Fundamentals of CAD
course - Spring 2001" + "\n");
      textarea.append("  Systems Realization Laboratory, Georgia
Institute of Technology - Atlanta, Ga" + "\n");
      textarea.append("**************************************************
************************************************" + "\n");
    }

   } //end of actionPerformed


/*****************************************************************/
  // Member function:    readFile()

/*****************************************************************/
  public void readFile() {
      textarea.append("\n" + "Loading .");

      String stlfile = textarea.getText();
      StringTokenizer st = new StringTokenizer(stlfile, "\n");

      //register # of vertices for array lengths
      numverts = 0;

      String aLine = "";

      while (st.hasMoreTokens()) {
```

```
        aLine = st.nextToken();
        if (aLine.indexOf("vertex") != -1)
        numverts++;
      }

      tricoords = new Point3f[numverts];
      normals = new Vector3f[numverts];

      bigpoint = 0.0f; //stores largest point value for scaling

      aLine="";
      int vert_count = 0;
      int norm_count = 0;

      st = new StringTokenizer(stlfile, "\n");
      while (st.hasMoreTokens()) {
        aLine = st.nextToken();

        if (aLine.indexOf("vertex") != -1) {
          int startat  = aLine.indexOf("vertex");
          int endat    = aLine.indexOf(" ", startat+8);
          String vert1 = aLine.substring(startat+7, endat);
          float vert1_flt = Float.parseFloat(vert1);
          if (bigpoint < Math.abs(vert1_flt)) { bigpoint = vert1_flt; }
          startat = endat+1;
          endat   = aLine.indexOf(" ", startat+2);
          String vert2 = aLine.substring(startat, endat);
          float vert2_flt = Float.parseFloat(vert2);
          if (bigpoint < Math.abs(vert2_flt)) { bigpoint = vert2_flt; }
          startat = endat+1;
          String vert3 = aLine.substring(startat);
          float vert3_flt = Float.parseFloat(vert3);
          if (bigpoint < Math.abs(vert3_flt)) { bigpoint = vert3_flt; }

          tricoords[vert_count] = new Point3f(vert1_flt, vert2_flt,
vert3_flt);
          textarea.append("   ." + "\n");
          vert_count++;
        }

        if (aLine.indexOf("normal") != -1)   {
          int startat  = aLine.indexOf("normal");
          int endat    = aLine.indexOf(" ", startat+8);
          String norm1 = aLine.substring(startat+7, endat);
          float norm1_flt = Float.parseFloat(norm1);
          startat = endat+1;
          endat   = aLine.indexOf(" ", startat+2);
          String norm2 = aLine.substring(startat, endat);
          float norm2_flt = Float.parseFloat(norm2);
          startat = endat+1;
          String norm3 = aLine.substring(startat);
          float norm3_flt = Float.parseFloat(norm3);

          normals[norm_count] = new Vector3f(norm1_flt, norm2_flt,
norm3_flt);
          normals[norm_count+1] = new Vector3f(norm1_flt, norm2_flt,
norm3_flt);
```

```
          normals[norm_count+2] = new Vector3f(norm1_flt, norm2_flt,
norm3_flt);
          //textarea.append("Normal: "+norm1_flt+" "+norm2_flt+"
"+norm3_flt+"\n");
          textarea.append("   ." + "\n");
          norm_count+=3;
        }
      } //end of while

      //draw stl file
      draw.createScene(bigpoint);
      draw.render(rendercheck, numverts, tricoords, normals);

  } // end of readFile()


/**********************************************************************/
  // Member function:    normCheck()

/**********************************************************************/
  public void normCheck() {
    crossf = new Vector3f[numverts];  //array for new calculated
normals
    int i = 0; //array index for tricoords[]
    int j = 0; //array index for normals[]
    int k = 0; //array index for crossf[]
    int errorcount = 0; //counter for # of mis-matched normals
    int correctcount =0; //counter for # of correct normals

    while (i < numverts) {
      Vector3f pt1 = new Vector3f(tricoords[i++]);
      Vector3f pt2 = new Vector3f(tricoords[i++]);
      Vector3f pt3 = new Vector3f(tricoords[i++]);
      Vector3f test1 = new Vector3f();
      test1.sub(pt2, pt1);
      Vector3f test2 = new Vector3f();
      test2.sub(pt3, pt1);
      Vector3f cross = new Vector3f();
      cross.cross(test1, test2);
      cross.normalize();
      crossf[k++] = new Vector3f(cross);
      crossf[k++] = new Vector3f(cross);
      crossf[k++] = new Vector3f(cross);

      if (crossf[k-1].equals(normals[j])) { correctcount++; }
      else {
        textarea.append("*** Normals don't match!! ***" + "\n");
        textarea.append(" Given vector value:  " + normals[j] + "\n");
        textarea.append(" Calculated vector value:  " + crossf[k-1] +
"\n");
        errorcount++;
      }
      j = j+3;
    } //end of while

    textarea.append("----------------------------------------" +
"\n");
```

```
    textarea.append("Total number of matching normals:  " +
correctcount + "\n");
    textarea.append("Total number of mismatched normals:  " +
errorcount + "\n");
    textarea.append("----------------------------------------" +
"\n");

  }//end of normCheck()


/*************************************************************************/
  // Member function:    normDraw()

/*************************************************************************/
  public void normDraw() {
      draw.detachBG();
      draw.createScene(bigpoint);
      draw.render(rendercheck, numverts, tricoords, crossf);
  } //end of normCheck()
/*************************************************************************/
}//end of class Satviewer
```

```
//*********************************************************************//
// A Java 3D class that uses the vertices of a .stl file to display  //
// the solid on the screen.                                          //
//*********************************************************************//

import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.loaders.ParsingErrorException;
import com.sun.j3d.loaders.IncorrectFormatException;
import com.sun.j3d.loaders.Scene;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.io.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import java.lang.Double;

public class Draw extends Applet {


/**********************************************************************/
  // Variables

/**********************************************************************/
  SimpleUniverse u;
  BranchGroup scene;
  TransformGroup objTrans;


/**********************************************************************/
  // Constructor:       Draw

/**********************************************************************/
  public Draw(){

    setLayout(new BorderLayout());
    Canvas3D c = new Canvas3D(null);
    add("Center", c);

    u = new SimpleUniverse(c);

    // This will move the ViewPlatform back a bit so the
    // objects in the scene can be viewed.
    u.getViewingPlatform().setNominalViewingTransform();

  } // end constructor Draw


/**********************************************************************/
  // Member function:    CreateScene()

/**********************************************************************/
  public void createScene(float bigpoint){
    // Create a simple scene and attach it to the virtual universe
    scene = createSceneGraph(bigpoint);
```

```
    u.addBranchGraph(scene);
  } // end of createScene()


/**************************************************************************/
  // Member function:    CreateSceneGraph()

/**************************************************************************/
  public BranchGroup createSceneGraph(float bigpoint){
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();
    objRoot.setCapability(BranchGroup.ALLOW_DETACH);

    // Create a Transformgroup to scale all objects so they
    // appear in the scene.
    TransformGroup objScale = new TransformGroup();
    Transform3D t3d = new Transform3D();
    if (bigpoint < 0.6f) {t3d.setScale(1.5);}
    else if ((bigpoint > 0.59f) && (bigpoint < 0.9))
{t3d.setScale(1.0);}
    else if ((bigpoint > 0.89f) && (bigpoint < 1.3f))
{t3d.setScale(0.5);}
    else if ((bigpoint > 1.29f) && (bigpoint < 2.5f))
{t3d.setScale(0.02);}
    else { t3d.setScale(0.0075); }
    objScale.setTransform(t3d);
    objRoot.addChild(objScale);

    // Create the transform group node and initialize it to the
    // identity.  Enable the TRANSFORM_WRITE capability so that
    // our behavior code can modify it at runtime.  Add it to the
    // root of the subgraph.
    objTrans = new TransformGroup();
    objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
    objTrans.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
    objScale.addChild(objTrans);

    BoundingSphere bounds = new BoundingSphere(new
Point3d(0.0,0.0,0.0), 350.0);

    // Create the rotate behavior node
    MouseRotate behavior = new MouseRotate();
    behavior.setTransformGroup(objTrans);
    objTrans.addChild(behavior);
    behavior.setSchedulingBounds(bounds);

    // Create the zoom behavior node
    MouseZoom behavior2 = new MouseZoom();
    behavior2.setTransformGroup(objTrans);
    objTrans.addChild(behavior2);
    behavior2.setSchedulingBounds(bounds);
    // Create the translate behavior node
    MouseTranslate behavior3 = new MouseTranslate();
    behavior3.setTransformGroup(objTrans);
    objTrans.addChild(behavior3);
    behavior3.setSchedulingBounds(bounds);
```

```
    // Set up the background
    Color3f bgColor = new Color3f(0.05f, 0.05f, 0.5f);
    Background bgNode = new Background(bgColor);
    bgNode.setApplicationBounds(bounds);
    objRoot.addChild(bgNode);

    // Set up the ambient light
    Color3f ambientColor = new Color3f(0.1f, 0.1f, 0.1f);
    AmbientLight ambientLightNode = new AmbientLight(ambientColor);
    ambientLightNode.setInfluencingBounds(bounds);
    objRoot.addChild(ambientLightNode);

    // Set up the directional lights
    Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
    Vector3f light1Direction  = new Vector3f(4.0f, -7.0f, -12.0f);
    Color3f light2Color = new Color3f(0.3f, 0.3f, 0.4f);
    Vector3f light2Direction  = new Vector3f(-6.0f, -2.0f, -1.0f);

    DirectionalLight light1 = new DirectionalLight(light1Color,
light1Direction);
    light1.setInfluencingBounds(bounds);
    objRoot.addChild(light1);

    DirectionalLight light2 = new DirectionalLight(light2Color,
light2Direction);
    light2.setInfluencingBounds(bounds);
    objRoot.addChild(light2);

    return objRoot;
  }


/**************************************************************************/
  // Member function:    DetachBG()

/**************************************************************************/
  public void detachBG() {
    scene.detach();
  } //end of detach


/**************************************************************************/
  // Member function:    render()

/**************************************************************************/
  public void render(int rendercheck, int numverts, Point3f
tricoords[], Vector3f normals[]) {

    Shape3D part = new Shape3D();

    if (rendercheck == 1){
      TriangleArray ta = new TriangleArray(numverts,
TriangleArray.COORDINATES | TriangleArray.NORMALS);
      ta.setCoordinates(0, tricoords);
      ta.setNormals(0, normals);
      part = new Shape3D(ta);
```

```
    }  //end of if rendered

    else if (rendercheck == 0) {
      int newnumverts = (numverts * 2);
      LineArray la = new LineArray(newnumverts, LineArray.COORDINATES);
      //fill array using coordinates
      int i = 0;    //line array counter
      int j = 0;    //triangle array counter
      while (i < newnumverts) {
        la.setCoordinate(i, tricoords[j]);
        la.setCoordinate(i+1, tricoords[j+1]);
        la.setCoordinate(i+2, tricoords[j+1]);
        la.setCoordinate(i+3, tricoords[j+2]);
        la.setCoordinate(i+4, tricoords[j+2]);
        la.setCoordinate(i+5, tricoords[j]);
        i = i + 6;
        j = j + 3;
      }
      part = new Shape3D(la);
    } //end of elseif Triangulated

    Appearance app = new Appearance();
    ColoringAttributes ca = new ColoringAttributes();
    ca.setColor (0.8f, 0.8f, 0.8f);
    app.setColoringAttributes(ca);
    Material material = new Material();
    app.setMaterial(material);
    part.setAppearance(app);

    BranchGroup obj2 = new BranchGroup();
    obj2.addChild(part);
    objTrans.addChild(obj2);

  } //end of render

} //end of class Draw
```

Appendix B: Learning Essays

This appendix contains the learning essays of each member of the project group.

i) Matthew Chamberlain
ii) Rahul Kulkarni
iii) Christopher Williams

Matthew Chamberlain

ME6104: Fundamentals of CAD – Critical Evaluation

May 3, 2001

I think I signed up for this class for all of the wrong reasons. Now that we have that out of the way, I can say that I think I stayed in this class and learned from it for all of the right reasons. It is my task in these two short pages to explain why I make these two claims.

First of all, I'll have to address the first point: why I signed up for this course originally. I will freely admit to the fact that I originally signed up for this class thinking that I was going to learn to use a CAD package. I had just spent a summer using ProEngineer with only a 5-day crash course on how to use it. I guess I was hoping that by taking an actual class on a CAD package, I would learn more about how CAD was supposed to be used (as opposed to my self-taught approach, which could be best described as "find a way to draw it and go with it –as long as it works it's okay.") I was quickly straightened out about the real content of ME6104, however, when somebody emphasized to me that the course's name was *fundamentals* of CAD.

I suppose that this word is quite important in describing ME6104. As a freshman at Carnegie Mellon, I took a class called *Fundamentals of Mechanical Engineering*, in which we studied and did homework on just about every possible subject in ME, from statics to thermodynamics, touching quickly on everything. The type of *fundamentals* I found in ME6104 was very different, and this is where I begin to address my second point: that I stayed in the class for all the right reasons.

What I found interesting about this class is that it has pushed my thinking about CAD in two completely opposite directions, away from my old utilitarian view of CAD. In one respect, I have come to have a greater respect for the intricate mathematics that are the essence of CAD. In another respect, I have begun to think about what the future holds for CAD, with the focus of my thoughts on how users can be presented with the lowest level of detail possible.

I think the first move in my thinking –the one towards considering all of the intricate details of CAD packages- is based primarily in the early course lectures and the homework assignments. At first, I found the lectures for this class to be anything but interesting, as they were purely mathematical with little direct relation to CAD packages as we know them. I guess I felt a little bit like the 16-year-old kid in driver's ed who is stuck studying all the theory for the written test but itches to go for a ride. I got this same feeling at first from the homework assignments. They were excruciatingly long and I did *not* understand why we were focusing so much on the details.

The change in my point of view came, I believe, with the completion of homework assignment #3, in which we had to plot the 3-D contours of the path of the laser beam. Taking all of the small building blocks that we had developed over the course of the semester and the two previous assignments and putting them all together to make something as cool as the contours with the SLA machine was really gratifying. This in part made me realize that each and every rendering done by a CAD package is backed by the mathematical theory that we had developed in class to that point, giving me a new respect for that theory.

The second move in my thinking came later on in the semester. Two activities in the class really precipitated this move. The first was the in-class description of one of the alternate problems for homework assignment #4, which concerned a small company that makes after-market parts (spoilers, etc) for cars and wants to move to CAD. The other was the in-class activity in which we attempted to identify features of CAD packages that would be built for very specific manufacturing processes. These two activities, along with our project, really got me thinking about the future of CAD long before the assignment to write this 'Critical Review' came out.

More specifically, what all of these activities have me thinking about is the relationship or interface between CAD packages and their users. In the case of the custom-parts company, the problem was that people who had never used CAD before and who were used to handcrafting parts were going to need to use it. In the case of the manufacturing processes, the problem was probably to tailor a software package to be understandable and useful for engineers who are only concerned with one manufacturing process. In either case, the key aspects in addressing the problem are the transfer of design intent from the mind of the user to the CAD package and the transfers of design information back from the CAD package to the user. In order to tailor the CAD package to these two cases, careful study of the way the engineers function and interact with their customers would have to be carried out.

Our project, an STL file viewer, deals with this issue as well. Every possible use that we could think of for our viewing involves sharing design intent or shape (information) between two people or between CAD packages and people. Of particular interest are the uses it could have for people who are not engineers and/or do not actually have a CAD package on their computer. This possible extension of the use of CAD and/or CAD files has me thinking about where CAD will be going in the near future. I wonder whether it will become more and more accessible in the way computer graphics and drawing has to this point. I also wonder whether RP technology will become a mainstream activity as computerized printing has over the last 40 years or so. Will we one day have RP machines hooked to our PC's at home or will the 'mainstreaming' of this technology stop with some sort of CAD/RP printing offices a la Kinkos? I think the second option is more likely, but it's still very interesting to think about.

So what have I learned? Basically, I learned what *fundamentals* are with respect to CAD –and that they are *not* just the basics of how to use a CAD package. The *fundamentals* are the designer's intent, the geometric modeling that will represent that intent, the interface that allows the designer to express that intent to the software package, and the means of expressing the geometric model back to the user (the graphical representation of the model.)

The stated goals of this course are "*to introduce fundamental technologies underlying CAD and show how they can be used to aid designing, not just geometry documentation. To provide an open classroom where students can learn by doing and relate the course material to their research and experience*." I think that all of these goals were achieved in the class. I wish I had paid more attention to the stated goals before I started the class –not because I would have dropped it because it didn't fit my original idea of the class- but because I would have realized what I was going to get out of it ahead of time.

ME 6104: COMPUTER AIDED DESIGN, SPRING 2001
RAHUL KULKARNI

1. *What were your goals for this course at first? What are your goals now? Comment on the changes in your goals and how they affect your approach to solving design problems using CAD. Have you achieved your goals?*

   When I first came to class for this course I did have a good idea about different techniques of 3D modeling in CAD. I believed that creating solid models in CAD software and then simulating its behavior under different loading conditions was all it was about. I also believed that the main reason why CAD is in its infancy in its use over the Internet because there was not a common file format and CAD software was too huge to be put up as web services over the Internet. My objectives initially were:
   1. To sharpen my skills in using a solid modeling package
   2. To come up with a file format based on XML that would be useful as the de facto standard for use over the Internet.

   Though my thoughts were right in a way, those were not the major reasons why CAD has not been widely used over the Internet. The main thing I learnt in the course is the importance of the use of CAD models in manufacturing. One needed to have lots of information to be able to generate a CAD model that could be manufactured straight away. The advent of the Internet did not just translate into CAD software being run as web services, but more importantly, it was the real time flow of information from different sources at different stages of the design process that really made the Internet such an important proposition.

   I also realized that just having a common file format was not the only goal. STEP is already the de facto standard for transferring CAD models. STEPML is the standard touted to be the one, which would be widely accepted for use over the Internet. So it was not the availability of a common file format, but the general acceptance and use of it as a standard that was a showstopper.

   My refined objectives were:
   1. To sharpen my skills in using a solid modeling package
   2. To explore a file format in detail and its use over the Internet
   3. Explore the feasibility of using CAD software over the Internet

   I am pleased to say that I achieved all my objectives through this course. It has not only enlightened me a lot on the use of Internet for CAD and the problems involved, but has also changed my perception of using CAD - from a process of creating a solid model with a few mouse clicks, to a tool which can use data from disparate sources to help design a solid model to translate functional requirements into form.

2. *What are the skills you picked up / refined as a result of taking this course and doing the exercises? What have you learned as a result of taking this course, doing exercises and working on your project?*

   I had expected to just update my skills on CAD when I came to the course, but at the end I feel I learned many new software programs thanks to this course. The first program I learned was Matlab. The assignments were challenging and at the same time were interesting since they were real world problems. I liked the connectivity between the assignments, which made us feel that we were progressing towards a common objective. We went a long way from plotting a point using mirrors modeled in Matlab for a SLA machine, to plotting lines and curves and finally surfaces. This was extremely challenging and it really required complete understanding of the basics of geometry representation. Through the CAD modeling assignments, I took up a different CAD program for each of the assignments. As a result of this I first learnt SolidWorks and then sharpened my skills on Pro / Engineer for the next assignment. The assignment on winged edge data structure and its implementation in a software program was also very interesting. The course gave a great overview on

the importance of CAD models for Rapid Prototyping. I enjoyed thoroughly the discussion on the different SLA machines and how they differed in the manufacturing process. I was also amazed to learn of the fast adaptation of Rapid Prototyping in the industry.

For my project I teamed up with Chris and Matt, to create a STL File Viewer over the Internet. I got a great opportunity to explore how a file format affects a program's design. I also got a great feel over the use of CAD software over the Internet.

3. *What are the 'fundamentals' of CAD? What have you learned about geometric modeling, human-computer interaction, and information modeling relative to engineering design?*

   Fundamentals of CAD to me, now, are not just knowing geometric primitives and how to put them together to get a solid model. Fundamentals of CAD also include the role of a designer and how design requirements need to be met through creation of a solid model. We learnt about different techniques for designing a part like feature- based design which was very useful. We were not taught which commands to use in CAD systems, but the basic concepts behind their use. Also we learnt a lot about rapid prototyping and how it is the next step in the way CAD models are used. The lecture on human-computer interaction was extremely interesting. The presentations by different groups after the lecture spoke of creativity and gave an insight into how human-computer interaction could take place in the future. I particularly liked the example shown in class for sheet metal bending. The course gave a great overview on information modeling in CAD systems. While teaching any concept, what information would be needed to model the construct was stressed upon. This gave us a good idea on the use of information in CAD especially in this Internet age.

4. *What is geometric modeling? What do you see as the role of geometric models, geometric modeling, and CAD in engineering design and product development? How does your project fit into this framework?*

   Geometric modeling is the creation of geometry in space. Concepts for geometric modeling include coordinate systems, representation of points, curves, solids and their transformations and operations on them. As long as we need products in three dimensions, geometric modeling is here to stay. Representing a CAD model without geometric modeling is not possible. In the future this concept would graduate to the representation of geometry along with functional requirements of that part of geometry. Also manufacturing parameters would be specified along with the geometry itself. File formats need to be adapted which would enable storage of such design data with the geometric data. This is crucial for the use of CAD in engineering product development of the future.

   Our project deals with the STL file format to represent geometry. This ties very well into the above framework, since through this project we have got to know how the STL format only supports representation of geometry data. This format would not even work for specifying colors in Rapid Prototyping machines which is possible today. Also we pointed out some shortcomings of the format especially since it represents redundant information.

5. *What is the role of CAD in engineering product development process? What should it be?*

   Before companies started using the Internet, most companies involved in software product development carried out the different phases of the product development sequentially. If, during the later stages of product development, the company came across new information or the user needs changed then these changes would be incorporated into the next version of the product otherwise risk shipping the product late. Rapid innovation in the technological areas and the Internet has created very dynamic environment in all walks of life. In this environment, the user needs are changing very rapidly resulting in new challenges for the companies and its product development managers. CAD plays an important role in this process. Every time new information arrives the first thing to change is the CAD model and its associated data. This has given rise to parametric CAD modelers. However in the future, I feel parametric CAD is not very healthy since it dampens creative spirits of the designers.

Georgia Institute of Technology
THE GEORGE W. WOODRUFF SCHOOL OF MECHANICAL ENGINEERING
ME 6104: FUNDAMENTALS OF CAD
SPRING 2001 - DR. ROSEN
Critical Evaluation

Christopher Williams
May 4th, 2001

Goals

Entering ME6104 on the first Tuesday of this semester, I was very excited. I was excited about taking a course that I had a solid background in. I was excited about finally taking an *entire* class with my co-advisor. I was excited about meeting new people and experiencing a new class setting. I was also excited about learning more about a subject that I have a great interest in.

As I walked into the class that day a few expectations for the semester popped into my head. I wanted to do well in the course. I wanted to further my knowledge of CAD. I wanted to use the opportunity to work on an interesting project. I wanted to observe a new teaching style in order to build upon my on.

Now that the semester is complete, I am satisfied with my growth during the semester. I feel like I have a deeper understanding of CAD and its applications to the design and manufacturing processes. Though it was doubtful at first, my group members and I finished a very interesting project that could have immediate impacts on the RPMI website and the RTTB project.

During the course of the semester I also took mental notes along with my class notes of the way Dr. Rosen taught the class. I observed how he addressed the class, how he dealt with questions, and how he presented material. The one teaching skill I learned from Dr. Rosen was how to manage students' differing levels of understanding. Some students had a great deal of experience with various CAD packages, while others had never opened AutoCad in their academic career. No matter what the case, Dr. Rosen was patient and understanding and was able to adjust his responses appropriately.

All in all, this semester's class went very well for me. Moving onward, I am changing my focus to working over the summer to find a Master's thesis topic while learning more about mass-customization and product families. I know that the learning I am taking away from this class will help me reaching these goals. The knowledge I gained from this course regarding using CAD to bridge design and manufacturing – this is crucial in design for mass-customization where one must concurrently design the product and its production process.

Skills

During the course of the semester I added many new skills to my repertoire. As I finished each homework assignment I became more and more proficient in Matlab. I had previous experience with Matlab in an undergraduate Controls lab, but I gained a whole new set of skills through this class (i.e., 3D plotting).

I also gained more experience with IDEAs through the homework assignments. I worked extensively with IDEAs in my first design course. Unfortunately that semester was the first semester that the course used IDEAs, as a result; I learn something new about IDEAs each time I use it.

One important discovery I made during the semester was while I was completing the last homework assignment. While attempting to design components of a beverage merchandiser for a parametric assembly, I realized that I did not know how to assemble parts in IDEAs. I thought I knew how, but when I put my ideas into practice, I found that I had no clue of what to do. This revelation, the "know you don't know" portion of the learning square, is providing me with a start for new learning. I am discouraged (and a little embarrassed) that I don't have this skill. Thanks to this class, I can now work on this skill over the summer semester.

The Future of CAD

The distributed design environment of the future will find design and manufacturing merging over the Internet. The role of a CAD software package will be changing as the functional requirements of distributed designers changes.

From the course and the project, I have developed many insights into the future of CAD and its role in design and manufacturing:

To assist distributed designers, CAD software will become Internet based. Designers will be able to work together and share design ideas through the use of a common CAD software package. The main engine of the CAD package will remain on the intranet server, while separate analysis codes and plug-ins will be purchased by the user and stored on his/her computer.

The infusion of the Internet into a CAD package will make it the integral piece of merging design and manufacturing. With internet capabilities, CAD software will enable users to share designs, share ideas, and share prototypes. Designers will be able to concurrently design a product and its production process with this new tool.

The role of CAD in the design and manufacturing process will increase ten fold. Designs will separate from CAD modeling and the virtual environment at very late stages in the design process. Since information will be easily transferred over the Internet, prototypes will be made in the embodying stages of design only when necessary. The actual part and assembly will not be constructed until the final stages of design in order to conduct final testing.

Overall Learning

Entering the course this semester I was comfortable with my understanding of CAD. Over the course of the semester, my horizons were greatly expanded as I soon learned more about the internal mathematics behind CAD modeling. NURBS, B-Splines, and parametric representations are now newly added parts of my engineering lexicon. This class was essential to my overall learning experience of graduate school not because it will help me for the Ph.D. Qualifying exam, but because it gave me the opportunity to formulate my own insight into the inner works of CAD. I am surprised, and a little disappointed, that we could afford to take the last month off for projects – there is still so much more to learn.

Appendix C: Final Presentation Slides

This appendix contains the slides of the presentation as presented to the Spring 2001 class of ME6104: Fundamentals of CAD on May 4th, 2001.