The *JavaGL* 3D Graphics Library & *JavaNL* Network Library

Student: Bing-Yu Chen
Advisor: Ming Ouhyoung, Ph.D.

Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, ROC

May 1997

Abstract

This thesis presents a three dimensional (3D) graphics library, *JavaGL*, and a multiparticipant network library, *JavaNL*, both of them are written in pure Java programming language. Therefore people can use the applications developed with these two libraries on Internet through a Java enabled browser or on a Java enabled machine. Besides *JavaGL* and *JavaNL*, this thesis also presents a new multiparticipant 3D graphics application interface on Internet. By using the *JavaGL* and *JavaNL*, people can develop multiparticipant 3D graphics applications in pure Java programming language and can be run on any kinds of Java enabled platforms.

With the growing popularity of Internet, 3D graphics and virtual reality (VR), more and more applications, for example, multiparticipant interactive 3D graphics games, require 3D graphics and network capabilities over network. Because Internet itself is a heterogeneous network environment, we need to have 3D graphics and network capabilities on each different platform. Furthermore, observing the development of Internet, we believe that "pay-per-use" software will be realized in the near future. Under this new paradigm, an application may need to be distributed from a server site to a client sites with different platforms.

To make *JavaGL* to be easy to learn and use, we define the application programming interface (API) in a manner quite similar to that of the OpenGL specification, since OpenGL is a de-facto industry standard for the programmers of developing 3D graphics applications. To make *JavaNL* survive under the heavy network traffic of Internet, we follow the concepts of Distributed Interactive Simulation (DIS), since DIS is a standard for the interactive simulation on Internet.

To develop such two libraries, there are some problems needed to be solved. 3D graphics rendering is a time consuming work and so is Java, using Java programming language to develop such a general purpose software based 3D graphics library, the performance is a great issue. Furthermore, how to make a 3D graphics application to be a 3D graphics multiparticipant application in a short time is an important consideration for the multiparticipant network library. In the development, we have made most efforts to solve these problems.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

As the world has become the Internet [1] world, all Internet services [2] have entered our daily life. World Wide Web (WWW) [3] is the most important service of these Internet services, because WWW has already contained all kinds of media in it, such as text, images, animation and sounds. The newest and powerful WWW browsers, such as Netscape Communicator of Netscape Communications Co. [4] and Microsoft Internet Explorer of Microsoft Co. [5], have already integrated all the Internet services, for example WWW, Electronic Mail (E-Mail), File Transfer Protocol (FTP), Network News and Internet Gopher, in them, one can view or use all of the services by clipping a house.

When WWW has become more and more popular, the development of three-dimensional (3D) computer graphics [6] and virtual reality (VR) [7] have also attracted more and more attention. Because the performance of computers is faster and faster day by day, several 3D graphics and VR applications can be realized on a cheaper computer, like the Intel chip's personal computers (PCs) or some low cost workstations. 3D graphics and VR have changed the computer world from flat to be three dimensional and from one way static output to two way interactive display.

Recently, several famous computer companies want to combine the 3D graphics and VR capabilities with Internet. The designer of Virtual Reality Modeling Language (VRML) [8] said: "because the world is not flat". Most media supported by WWW are 2D media, such as text, images and two dimensional (2D) animation. There are few 3D graphics and VR media on Internet.

Since Apple QuickTime VR of Apple Computer, Inc. [9] and VRML joined Internet, WWW was no more flat. A 3D object can be shown on WWW, and users can view the virtual object or walkthrough in a virtual building on WWW. But, the virtual objects and environments built by Apple QuickTime VR and VRML are all static. How can one interacts with the 3D graphics and VR projects on the Internet just as one does on a stand-alone computer?

Internet itself is a heterogeneous network environment. If we want to develop some 3D graphics and VR applications on Internet, we will need 3D graphics capability on different platforms which are connected to Internet. This is such a difficult task, but when Java programming language [10] comes to the Internet world, this difficult task seems have a solution. When Java programming language was introduced to Internet, it has spread into WWW very quickly. Today, one can run a Java applet or a Java application on all WWW browsers with all different platforms. Java virtual machine (VM) has become a new type of computer and all the platforms are like the devices of this kind of computer. So, if the program is written by Java programming language, it can be run practically on all kinds of computers.

For now, Java seems to be the solution of our problem. Using Java, we can write some simple program on Internet, but we can not run some larger 3D graphics and VR programs. The biggest problem is that there is no 3D graphics library in Java, such as OpenGL of Silicon Graphics, Inc. [11] or RenderWare of Criterion Software Ltd. [12] on the stand-alone computers. Without an efficient 3D graphics library, we must do the complex rendering work by ourselves each time we stand a 3D graphics projects, so we decide to develop a 3D graphics library for Java programming language, JavaGL [13]. To make the library to be easy to use, we follow the specification of OpenGL [14] to design the application programming interface (API) of the library, because OpenGL is a de-facto industry standard to all the 3D graphics and VR programmers.

From the experience of developing 3D graphics and VR projects on a stand-alone computer, we believe that, after the 3D graphics and VR applications shift to Internet, multiparticipant 3D graphics and VR applications will be the next focus on Internet, because the single user mode applications can not fulfill users need. For this reason, we decide to design a multiparticipant network library, JavaNL [15], to provide the network capability for the multiparticipant 3D graphics and VR applications. To develop a new network protocol for multiparticipant interactions is another research issue in the network area, so we decided to modify an existing protocol, Distributed Interactive Simulation (DIS) [16], an IEEE standard, to meet our requirements.

1.2 Development Environment

In resent years, Internet began to be popular exponentially. This is because computers have become more and more powerful and could be used easily for end users. Because a computer without network has limited functions, a stand-alone computer can not satisfy all users. At the same time, WWW has exploded on the Internet and has been popular in a short time, more and more useful services are offered by many companies and organizations on WWW.

When Internet and WWW become more and more popular, 3D graphics, and VR have also become more and more popular. Many stand-alone computers have supported high performance 3D graphics rendering or processing. Many companies, universities, and organizations paid much efforts to combine the 3D graphics and VR capabilities with Internet and WWW.

As Internet, WWW, 3D graphics, and VR have been used by many end users, they can see multimedia documents on Internet. But, lack of interactivity, people in the client sites can only view the well defined documents in the server sites. To make the documents more interactive, Java has joined the Internet family and the programmers can develop programs on it.

To combine the 3D graphics and VR capabilities with Internet or WWW, Java seems to be a suitable programming language for the Internet environment. Hence, the system development environment is on the Internet environment and using Java programming language to be the development language.

1.3 JavaGL– A 3D Graphics Library in Java

JavaGL [17] is a 3D graphics library with an API similar to that of OpenGL, a de-facto industry standard, and is developed by pure Java programming language. The performance is a great challenge for a general purpose software based 3D graphics library, and it is also a great challenge for a Java application. Unfortunately, JavaGL is a general purpose software based 3D graphics library developed by pure Java, of course, the performance will be the most important concern when developing JavaGL.

To keep JavaGL be platform independent, we gave up using any native code to implement the graphics kernel of it. OpenGL is a general purpose 3D graphics library, the performance of it is less than some specified 3D graphics library, but OpenGL is familiar with all 3D graphics programmers, and can be used to develop all kinds of the 3D graphics applications. To obey the specification of OpenGL, we must pay more efforts to enhance the performance of JavaGL.

OpenGL is not designed for Java programming language. Actually, when 3D graphics programmers develop some 3D graphics applications using OpenGL, there is no Java on the world. To follow the specification of OpenGL and satisfy the programmers' habit, we must pay much efforts to designed the API of JavaGL.

Since we built JavaGL's homepage on our web site four months age, there are more than 1000 people around the world visited this homepage. These people from more than 60 companies or organizations used ftp to download JavaGL, and sent us dozens of e-mails concerning the use of JavaGL. Some of them offered to collaborate and some just used the libraries, and this encourages us to improve JavaGL continuously.

1.4 JavaNL– A Network Library in Java

JavaNL is a multiparticipant network library, which follows the concepts of DIS, and is also developed by pure Java programming language. The API is the most important concern when we develop it, because we expect this library is easy to learn and use. Because DIS has not released all the specifications [18], we must pay our efforts to design the missing parts of JavaNL.

JavaNL is still not completely implemented so far. However it can offer most of the necessary network functions to the programmers. The goal of JavaNL is to present a prototype for constructing network applications as easy as possible, and to offer programmers a reliable and useful library to build multiparticipant network applications.

1.5 Multiparticipant 3D Graphics Application Interface

After developing the two libraries, we have already provided the 3D graphics and network capabilities on Internet. To develop a multiparticipant 3D graphics application, we can use JavaGL and JavaNL for the 3D graphics rendering and the network data exchange. As an example of a multiparticipant building walkthrough system, the “pay-per-use” multiparticipant 3D graphics application interface will be realized by using the two libraries in a short time.

For this application interface, we have done a multiparticipant building walkthrough system as a demonstration. This system is developed with JavaGL and JavaNL as the description of the application interface. People in the building model can see other users as a symbol in the same building environment and chat with each other.

1.6 Organization

This thesis is organized as follows, Chapter 2 introduces the current development of 3D graphics and VR on Internet. Chapter 3 and Chapter 4 describe the main parts of this thesis, *JavaGL* and *JavaNL*. Besides the implementation issues, there are some performance evaluation. Chapter 5 is a multiparticipant building walkthrough system with chat capability in it. It is developed with *JavaGL* and *JavaNL* as a testing example and presents a new multiparticipant 3D graphics application interface on Internet. Finally, there are some conclusions and future works in Chapter 6.

Chapter 2

Computer Graphics and Virtual Reality on Internet

This chapter surveys related technologies and systems that are important to the development of the thesis. Experienced readers can skip this chapter and go directly to the next chapter.

2.1 The Internet

The Internet was born about 25 years ago, as a U.S. Defense Department network called the ARPANET. The ARPANET was built by the Advanced Research Project Agency (ARPA) and was a computer network system for packet switching. The ARPANET was an experimental network designed to support military research – in particular, research about how to build networks that could withstand partial outrages and still function. In the ARPANET model, communication always occurs between a source and a destination computer. The network itself is assumed to be unreliable; any portion of the network could disappear at any moment.

About ten years later, Ethernet local area networks (LAN) and workstations came on the scene. Most of these workstations came with Berkeley UNIX, which came with Internet Protocol (IP) networking. This created a new demand: rather than connecting to a single large timesharing computer per site, organizations wanted to connect the ARPANET to their entire local network. This would allow all the computers on that LAN to access ARPANET facilities. About the same time, other organizations started building their own networks using the same communications protocols as the ARPANET: namely, IP and its relatives. It became obvious that if these networks could talk together, users on one network could communicate with those on another; everyone would benefit.

One of the most important of these newer networks was the NSFNET, run by the National Science Foundation (NSF), an agency of the U.S. Government. In the late 80's the NSF created five supercomputer centers. This created a communications problem: they needed a way to connect their centers together and to allow the clients of these centers to access them. At first, the NSF tried to use the ARPANET for communications, but this strategy failed because of bureaucracy and staffing problems.

In response, NSF decided to build its own network called NSFNET, based on the ARPANET's IP technology. It connected the centers with 56,000 bits per second (56K bps) telephone lines. In 1989, NSFNET used T1 (1.544M bps) to be the communication network and became the back bone of the Internet.

Now, many universities, research centers and business companies are connected by the Transmission Control Protocol (TCP)/IP protocol and all are the parts of the Internet. There are several services on the Internet, such as Remote Terminal Protocol (TELNET), FTP, E-Mail (Simple Mail Transfer Protocol; SMTP), Network News (Network News Transport Protocol; NNTP), Internet Gopher, Archie, Wide Area Information Servers (WAIS), World Wide Web (WWW) (Hypertext Transfer Protocol; HTTP [19]), Bulletin Board System (BBS), Internet Relay Chat (IRC), Multiple User Dimension (MUD)…..etc.

2.2 The World Wide Web (WWW) Explosion

The WWW is the newest information service to arrive on the Internet and is based on a technology called hypertext. Most of the development has taken place at CERN, the European Particle Physics Laboratory. The WWW merges the techniques of networked information and hypertext to make an easy but powerful global information system and represents any information accessible over the network as part of a seamless hypertext information space.

The WWW consists of documents and links. The format of documents is called Hypertext Markup Language (HTML) [20] which contains several kinds of formats, such like text, images, photos, animation, sounds and links to other documents. A simple protocol called HTTP is used to allow a browser program to request to see the documents that offered by a remote information server.

This kind of simple protocol and documents are originally used between the research centers and their members until a good browser exists. In 1993, the research members in the National Center for Supercomputing Applications (NCSA) released the NCSA Mosaic [21], a web browser that is easy to use. Then, the WWW is used via some powerful browsers by many people.

The WWW browsers can access many existing data systems via existing protocols, such as FTP and NNTP or via HTTP and a gateway. In this way, the critical mass of data is quickly exceeded, and the increasing uses of the system by readers and information suppliers encourage each other. Providing information is as simple as running the WWW (HTTP demon) server and pointing it at an existing directory structure. The server automatically generates the hypertext view of your files to guide the user around. To personalize it, you can write a few Standard Generalized Markup Language (SGML) or HTML hypertext files to give an even more friendly view. Also, any file available by anonymous FTP, or any Internet newsgroup can be immediately linked into the web.

Today, many users who are not familiar with computer science or any high technology are using the WWW. Many people even think that the WWW is the Internet, that means they think the WWW is all of the Internet.

2.3 Overview of Existing Standards and Applications

As Internet and WWW have become more and more popular, many companies, universities, and organizations have tried to combine the three-dimensional (3D) graphics and virtual reality (VR) capabilities with Internet. Now, there are some 3D graphics and VR applications, products, and standard on Internet.

2.3.1 VRML - Virtual Reality Modeling Language

The Virtual Reality Modeling Language (VRML) [22] is a language for describing multiparticipant interactive simulations – virtual worlds networked via the global Internet and hyper-linked with the WWW. All aspects of virtual world display, interaction and inter-networking can be specified using VRML. It is the intention of its designers that VRML become the standard language for interactive simulation within the WWW.

The VRML is a file format for describing 3D interactive worlds and objects. It may be used in conjunction with the WWW. It may be used to create 3D representations of complex scenes such as illustrations, product definition and VR presentations.

The first version of VRML allows for the creation of virtual worlds with limited interactive behavior. These worlds can contain objects which have hyper-links to other worlds, HTML documents or other valid Multipurpose Internet Mail Extensions (MIME) types. When the user selects an object with a hyper-link, the appropriate MIME viewer is launched. When the user selects a link to a VRML document from within a correctly configured WWW browser, a VRML viewer is launched. Thus VRML viewers are the perfect companion applications to standard WWW browsers for navigating and visualizing the WWW.

Moving Worlds [23] is the name of Silicon Graphics' submission to the Request-for-Proposals for VRML 2.0. It was chosen by the VRML community as the working document for VRML 2.0. It was created by Silicon Graphics, Inc. In collaboration with Sony and Mitra. Many people in the VRML community were actively involved with Moving Worlds and contributed numerous ideas, reviews, and improvements.

Our laboratory has developed a browser for VRML 1.0 file format [24] and is developing a browser for VRML 2.0 file format [25] using Java programming language. The goal of this VRML browser is to provide users all the necessary functions from servers so that users do not have to install additional hardware or software for 3D graphics.

2.3.2 Apple QuickTime VR

Traditionally, VR systems use 3D computer graphics to model and render virtual environments in real-time [26]. This approach usually requires laborious modeling and expensive special purpose rendering hardware. The rendering quality and scene complexity are often limited because of the real-time constraint. Apple QuickTime VR is a new approach which uses 360-degree cylindrical panoramic images to compose a virtual environment. The panoramic image is digitally warped on-the-fly to simulate camera panning and zooming. The panoramic images can be created with computer rendering, specialized panoramic cameras or by “stitching” together overlapping photographs taken with a regular camera. Walking in a space is currently accomplished by “hopping” to different panoramic points.

Now, Apple QuickTime VR is a standard format on Internet. People in a client computer can see the format of files via Internet by using Apple QuickTime VR viewer. The newest version of Apple QuickTime VR is version 2.0.

Like Apple QuickTime VR, our laboratory has a similar project which I have joint, named Photo VR [27]. Unlike Apple QuickTime VR, the scene of Photo VR is rendered by generating a sphere-like polyhedral environment map from photo-realistic images and using the generated maps to render the scene by techniques of computer graphics [28][29]. Because we use the sphere-like polyhedral to be the model of the environment map, the users can see the sky and the ground in Photo VR. Like Apple QuickTime VR, the users can use Photo VR via Internet as using the viewer of Apple QuickTime VR.

2.3.3 DIS - Distributed Interactive Simulation

Distributed Interactive Simulation (DIS) is a government/industry initiative to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual worlds for the simulation of highly interactive activities. This infrastructure brings together systems built for separate purposes, technologies from different eras, products from various vendors, and platforms from various services, and permits them to interoperate. DIS exercises are intended to support a mixture of virtual entities with computer controlled behavior (computer generated forces), virtual entities with live operators (human in-the-loop simulators), live entities (operational platforms and test and evaluation systems), and constructive entities (war games and other automated simulations). DIS draws heavily on experience derived from the Simulator Networking (SIMNET) program developed by the ARPA, adopting many of SIMNET's basic concepts and heading lessons learned.

In order for DIS to take advantage of currently installed and future simulations developed by different organizations, a means had to be found for assuring interoperability between dissimilar simulations. This means were developed in the form of industry consensus standards. The open forum (including government, industry, and academia) chosen for developing these standards was a series of semi-annual Workshops on Standards for the Interoperability of Distributed Simulations that began in 1989. The results of the workshops have been several IEEE standards. These standards provide application protocol and communication for Exercise Management and Feedback, provides user guidelines for setting up and conducting a DIS exercise.

2.4 The Problems of Internet Programming Complexity

After putting the text, images, photos, animation and sounds on the WWW, we are thinking that what is the next thing we can do on the WWW or Internet. There are several answers of this question, but there are two important things that are noticed by people, 3D graphics and programming. Because all the media on the WWW are only two dimensional and are not interactive, people want to see something in three dimensional and has more interactivity. The three dimensional requirement will be described in another section, here we concerned is interactivity.

To make the materials on the WWW to be more interactive, we must write some program for the WWW. Before Java existed on the WWW, what we can do are Common

Gateway Interface (CGI) and MIME. But, CGI is the program in the server side, large CGI program may cause the performance of the WWW server down, and CGI can only show the text mode and limited results. The MIME format is a format which can contain several kinds of data format in one file, and the WWW browsers will call the viewer in the client side to view all the data. The program in the client side is platform dependent, one can not run the program which is originally run on the PC on workstation.

If the program is running on the server side, the load of the WWW server might be very heavy. If the network is broken when the user is running the program on the server side, he can not run the program until the client side connecting to the server side again. For this reason, we wish the program will be run on the client side. It is very difficult to program on the Internet, because we do not know what platform the client side is running on.

2.5 Java - A Programming Language for Internet

Java is a language designed for programming on the Internet [30]. Java enables the programmers to write completely new kinds of applications. With Java it is possible to imagine a true document-centric system, where a colleague can send you a report packaged with the word processing, spreadsheet, and database software you need to work with it.

Java offers the promise that the network will become the computer. If the Internet breaks down barriers, Java removes the last and most difficult of these: the barrier that prevents you from taking software from some random site and executing it on any platform.

Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language. Java is a simple language, because Java uses many of the same language constructs as C and C++. Java is an object-oriented language, so Java has the advantages of the object-oriented language.

Java is designed to support applications on networks; it is a distributed language. The Java compiler generates byte-codes, rather than the native machine code. To run a Java program, a Java interpreter is needed. Java byte-codes provide an architecture neutral object file format; the code is designed to transport programs efficiently to multiple platforms.

Java’s origin as a language for software for consumer electronics means that Java has been designed for writing highly reliable or robust software. One of the mostly highly touted of Java is that it is a secure language. Security is an important concern, as Java is meant to be used in networked environments. Java programs are compiled to an architecture neutral byte-code format. The primary advantage of this approach is that it allows a Java application to run on any system, as long as that system implements the Java Virtual Machine. Being architecture neutral is one big part of being portable.

Java is an interpreted language, so it is never going to be as fast as a compiled language like C, but it is still much better than the high-level scripting languages. Java is a multithreaded language; it provides support for multiple threads of execution that can handle different tasks. The Java language was designed to adapt to an evolving environment; it is a dynamic language.

Chapter 3

JavaGL– A 3D Graphics Library in Java

3.1 Introduction

As Internet, three-dimensional (3D) graphics, and virtual reality (VR) are getting more and more popular, there is increasing demand of 3D graphics applications over network. However Internet itself is a heterogeneous network environment. If we want to develop 3D graphics applications on Internet, we will need a 3D graphics capability in each different platform. Furthermore, observing the development of Internet, we believe that “pay-per-use” software will be realized in the near future. Under this new paradigm, a 3D graphics application may be distributed from a server to a client with a different platform. Therefore, we decide to develop a 3D graphics library that is platform independent, and Java is chosen for its platform independent feature.

At the same time, it is desired that this 3D graphics library be easy to learn, so we define the application programming interface (API) in a manner quite similar to that of OpenGL, since OpenGL is a de-facto industry standard. Hence, JavaGL is a 3D graphics library with an API which is similar to that of OpenGL and developed in pure Java programming language.

3.2 Related Researches and Products

There are several companies tried to combine the 3D graphics capability with Java programming language. Some of them modified the Java virtual machine (VM) directly, some used native code to enhance the performance, and some just tried to develop with pure Java programming language.

3.2.1 Java 2D and Java 3D from Sun Microsystems, Inc.

Java 2D [31] provides an abstract imaging model that extends the 1.0.2 AWT package, including line art, images, color, transforms and composting. Java 3D [32] provides an abstract, interactive imaging model for behavior and control of 3D objects. It is a specification extension of Java 2D and defined through a joint effort by Apple, Intel, Silicon Graphics, and Sun Microsystems

The Java 2D API is a set of classes for advanced 2D graphics and imaging, encompassing line art, text, and images in a single comprehensive model. The API provides extensive support for image composting and alpha channel images, a set of classes to provide accurate color space definition and conversion, and a rich set of display-oriented imaging operators. These classes will be provided as additions to the java.awt and java.awt.image packages contained in the Java Developers Kit (JDK) [33].

3.2.2 Liquid Reality from Dimension X, Inc.

Liquid Reality [34] is a modular and extensible Virtual Reality Modeling Language (VRML) 2.0 platform written in Java, providing both a browser for viewing VRML content and a developers toolkit of Java classes for creating powerful 3D applications.

Liquid Reality enables the creation of dynamically extensible VRML environments that seamlessly integrate motion, sound, and intelligent behaviors. With Liquid Reality users can create their own customized VRML 2.0 browser. Based upon ICE, Dimension X's 3D graphics library, Liquid Reality is not only extensible using Java but can be run on any Java-enabled browser.

With more than 250 classes for 3D content creation, Liquid Reality also provides custom nodes for a level of behavior programming way beyond the standard VRML 2.0

spec! In concert with ICE, Dimension X’s rendering API for Java, Liquid Reality-based applications achieve the speed and performance usually reserved for native 3D.

ICE is a 3D graphics library with Java API. To use the library, users and developers must install the native 3D graphics library, which is not platform independent and the API specification is not standard.

3.2.3 Cosmo 3D from Silicon Graphics, Inc.

Cosmo 3D [35] is a new platform independent graphics toolkit that brings ultra high performance, real-time 3D applications to the Internet and the desktop. Implemented in C++, this toolkit supports over 30 leading graphics file formats, including the Internet-standard VRML 2.0 and is accessible to Java applications through efficient Java bindings.

Silicon Graphics' Cosmo 3D enables Internet developers to incorporate dynamic 3D worlds, collaborative design environments, and virtual characters with spatialized and synchronized audio into their WWW applications.

Cosmo 3D is a graphics library which supports Java 3D and provides an efficient Java binding between a Java application and the natively-compiled Cosmo 3D libraries. Java 3D was defined through a joint effort by Apple, Intel, Silicon Graphics, and Sun Microsystems.

Cosmo 3D has been designed to be independent of the underlying rendering architecture. To achieve the highest performance and the broadest platform acceptance, it has been implemented on top of OpenGL. For un-accelerated, mainstream personal computer (PC) users running Microsoft Windows 95 or NT, Cosmo 3D uses Cosmo GL, a complementary special implementation of OpenGL tuned specifically for maximum software-rendering performance on the PC. Cosmo GL for the PC is available directly from Silicon Graphics.

3.2.4 Java3D from Jeff Orkin at Demon Systems, Inc.

Java3D [36] is an evolving set of classes used to display and manipulate 3D graphics. So far it displays in wire-frame, and can read in object in the PLG 3D file format. Eventually, it will handle solid modeling, shading, and texture mapping, and hopefully will read other file formats.

3.3 Implementation Issues

3.3.1 Introduction to OpenGL

OpenGL (for “Open Graphics Library”) graphics system is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of 3D objects.

To the programmer, OpenGL is a set of commands that allow the specification of geometric objects in two or three dimensions, together with commands that control how these objects are rendered into the framebuffer. For the most part, OpenGL provides an immediate-mode interface, meaning that specifying an object causes it to be drawn.

OpenGL allows a programmer to create interactive 3D applications that produce color images of moving 3D objects. With OpenGL, a programmer can control computer graphics technology to produce realistic pictures or ones that depart from reality in imaginative ways.

OpenGL is designed as a streamlined, hardware independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, a programmer must work through whatever windowing system controls the particular hardware he is using. Similarly, OpenGL does not provide high-level commands for describing models of 3D objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, a programmer must build up his desired model from a small set of geometric primitives–points, lines, and polygons.

3.3.2 OpenGL vs. JavaGL

The functions of OpenGL can be divided into three categories: the *OpenGL Utility Library (glu), OpenGL (gl),* and *OpenGL Extension to the X Window System (glX),* as shown in Figure 3-1 (a), where the hierarchy is for X Window system [37]. For Microsoft Windows system [38], there are some differences in platform dependent modules, as shown in Figure 3-1 (b).

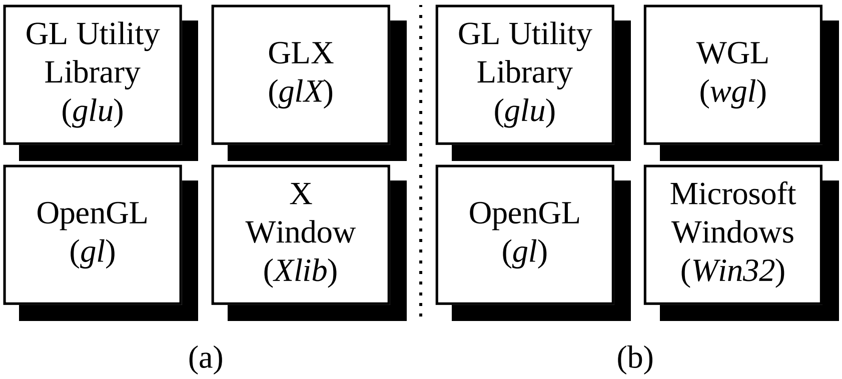

GL Utility Library (*glu*)
GLX (*glX*)
OpenGL (*gl*)
X Window (*Xlib*)
GL Utility Library (*glu*)
WGL (*wgl*)
OpenGL (*gl*)
Microsoft Windows (*Win32*)
(a)
(b)

Figure 3-1 OpenGL API hierarchy for (a) X Window and (b) Microsoft Windows.

OpenGL provides a small but powerful set of drawing primitive operations, and all higher-level drawing must be done in terms of these. The *glu* includes several commonly used graphics routines that encapsulate the OpenGL commands. The *gl* is the main part of OpenGL. The *glX* and *wgl* are the *OpenGL extension to the X Window System* (*Xlib*) and *Microsoft Windows System* (*Win32*), they are implemented depending on different platforms. Besides these three interfaces, there is an *OpenGL Programming Guide Auxiliary Library*, called *aux* or *glaux*, which is not an official part of OpenGL, but is useful for using OpenGL and familiar for the 3D graphics programmers.

We follow the above principles and concepts to develop the API hierarchy of JavaGL, as shown in Figure 3-2.

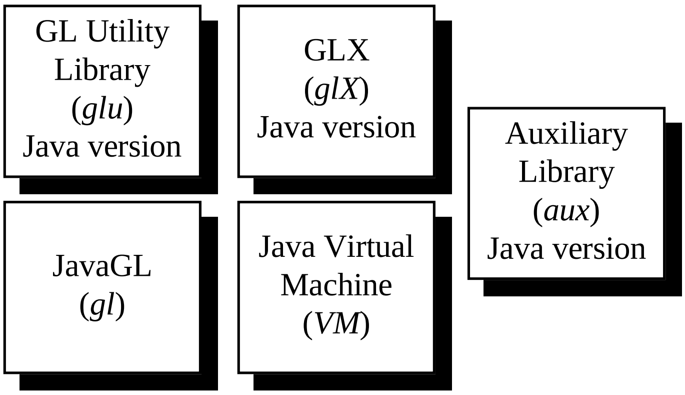

GL Utility Library (*glu*) Java version
GLX (*glX*) Java version
Auxiliary Library (*aux*) Java version
JavaGL (*gl*)
Java Virtual Machine (*VM*)

Figure 3-2 JavaGL API hierarchy.

The implementation is mainly based on the specification of OpenGL. For *OpenGL Utility Library (glu), JavaGL (gl),* and *OpenGL Extension to the X Window System (glX),* we follow the methodologies described in the *OpenGL Graphics System*, *OpenGL Graphics System Utility Library*, and *OpenGL Graphics with the X Window System* specifications. Because the Java virtual machine is not the X Window system, we originally plan to design the *OpenGL Extension to the Java Virtual Machine*; however to make the programmers use JavaGL similar to using OpenGL, we still follow the specification of the *OpenGL Graphics with the X Window System* to design the API of this part. For the purpose of easy using, we have also provided the *OpenGL Programming Guide Auxiliary Library (glaux)* as described in the *OpenGL Programming Guide* [39]. Besides these four interfaces, there is an underlying graphics kernel which is transparent to programmers.

3.3.3 Graphics Kernel

The hierarchy of OpenGL modules is shown in Figure 3-3. The functional hierarchy of

JavaGL is shown in Figure 3-4, like the hierarchy of OpenGL:

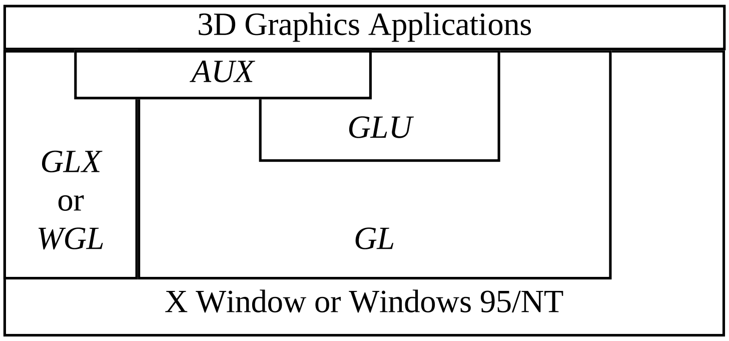


Figure 3-3 The hierarchy of OpenGL modules.

As the above figure, there are four interfaces which will be used by the 3D graphics applications in OpenGL, the *AUX*, *GLX* or *WGL*, *GLU*, and *GL*. The 3D graphics applications will call the functions in the four modules, where the contents of the four modules have been introduced in the about section. The hierarchy of JavaGL is designed to follow the above figure, and is like the following figure.

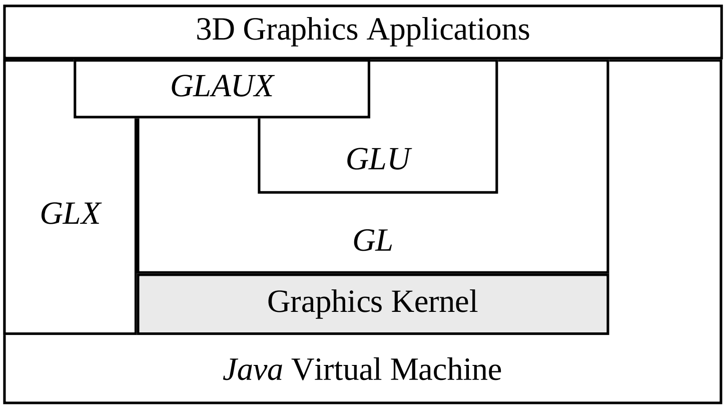


Figure 3-4 JavaGL implementation hierarchy.

There are three components which will be used by the 3D graphics applications, the *GLAUX*, *GLU*, and *GL*. For 3D graphics applications, these three components are classes or objects. What the applications need to do is just “new” the objects then using the member functions of these objects as using the commands of OpenGL. For the platform dependent *GLX* or *WGL*, we follow the specification of *GLX* to provide programmers a mechanism to bind Java’s graphics context with JavaGL. As the above figure, the *GLAUX*, *GLU*, and *GL* are only programming interface for programmers, the main part of JavaGL is the graphics kernel which is transparent to programmers. The graphics kernel provides a set of primitive rendering routines, and its Java class relationship is illustrated in Figure 3-5.

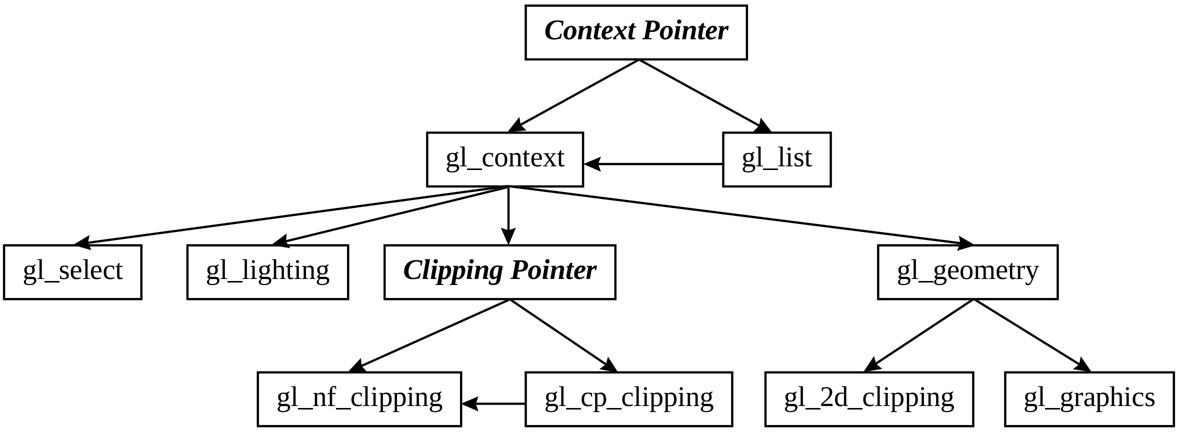


Figure 3-5 JavaGL Graphics Kernel hierarchy.

In Figure 3-5, the sibling classes, *gl_context* and *gl_list*, have the same parent class and member functions. They both content graphics kernel functions, but *gl_context* is the really graphics routines, and *gl_list* is only for display list. The Context Pointer is a dispatcher that re-directs rendering commands to either *gl_list* or *gl_context*. When the state of GL is normal, the Context Pointer is pointing to *gl_context*, but when the state of GL is stalling to the display list, the Context Pointer is pointing to *gl_list*. The *gl_list* records a sequence of rendering commands, and eventually calls *gl_context* for rendering. The *gl_nf_clipping*, *gl_cp_clipping*, and Clipping Pointer have the same relation with that between *gl_context*, *gl_list*, and Context Pointer, where *gl_nf_clipping* is the clipping class for near and far clipping planes and *gl_cp_clipping* is the clipping class for user defined clipping planes.

The other classes are *gl_select* for selection, *gl_lighting* for lighting calculation, *gl_geometry* for drawing all kinds of geometric objects, *gl_2d_clipping* for 2D clipping functions, and *gl_graphics* is the basest drawing functions of the graphics kernel.

3.3.3.1 Association of Current Values

JavaGL, as OpenGL, is a state machine. Each vertex will be processed with the values of *current normal* and *current color*. *Current normal* vectors are used in lighting calculations. *Current color* is associated with each vertex as it is specified. The *associated color* is either the current color or a color produced by the lighting calculations depending on whether or not lighting is enabled. Figure 3-6 summarizes the associated data with a transformed vertex to produce a *processed vertex*.

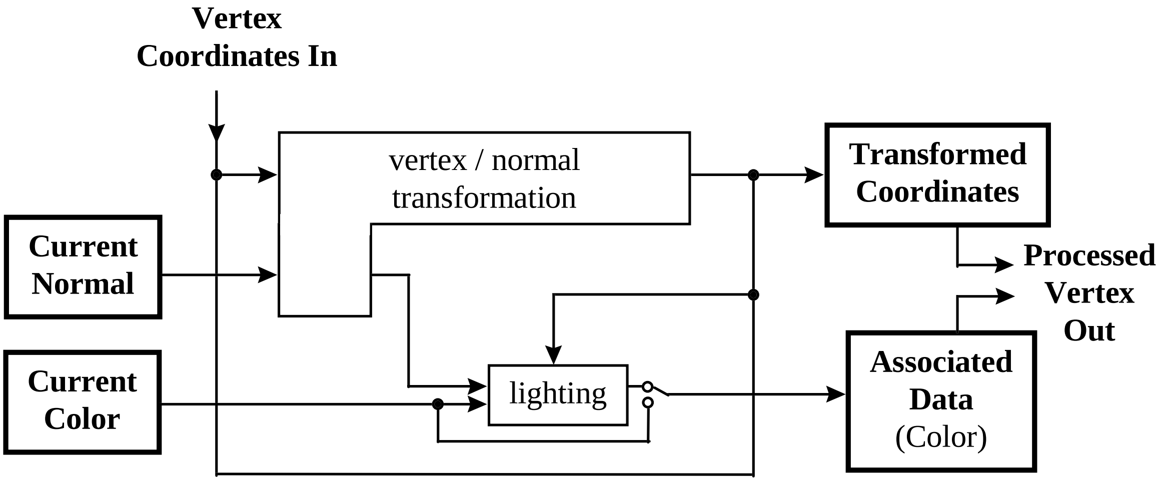


Figure 3-6 Association of current values with a vertex.

The current normal and current color are parts of the JavaGL state. The vertices and the normal vectors are transformed, color may be affected or replaced by lighting calculations. The processing indicated for each current value is applied for each vertex.

3.3.3.2 Coordinate Transformations

Figure 3-7 shows the vertex transformation sequence when the programmers call the JavaGL to draw a polygon on the screen.

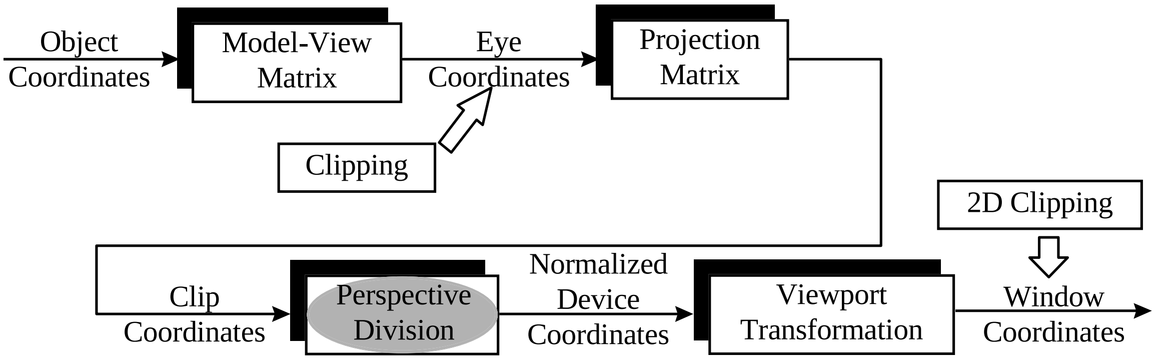


Figure 3-7 Vertex transformation sequence.

The coordinate system in which the programmers define objects is called *object coordinates*. After multiplying the *model-view matrix*, the coordinates are transformed into the *eye coordinate*. In the *eye coordinates*, we do 3D clipping, including the clipping planes clipping and the near and far planes clipping. After all the vertices are clipped, we multiply the vertices by the *projection matrix* to transform the vertices into the *clip coordinates*. After *perspective division*, we get the *normalized device coordinates*. After the *viewport transformation*, all the vertices are in the *window coordinates*. After the 2D clipping to cut the vertices which are not fitting into the screen, we can draw all the survival vertices to the screen.

The *object coordinates*, *eye coordinates*, and *clip coordinates* are four dimensional. The *model-view matrix* and *perspective matrix* are both four-by-four matrices.

If a vertex in the *object coordinates* is given by $\begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}$ and the *model-view matrix* is *M*, then the vertex's *eye coordinates* are found as

$$\begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = M \begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix}.$$

Similarly, if *P* is the *projection matrix*, then the vertex's *clip coordinates* are

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = P \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}.$$

The vertex's *normalized device coordinates* are then

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{pmatrix}.$$

Then, the vertex's *window coordinates* are given by

$$\begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix} = \begin{pmatrix} (p_x / 2)x_d + o_x \\ (p_y / 2)y_d + o_y \\ [(f - n)/2]z_d + (f + n)/2 \end{pmatrix},$$

where p_x and p_y are the viewport's width and height, (o_x, o_y) is the viewpoint's center, and z_d, n and f are the depth ranges.

3.3.3.3 Lighting

Before describing the lighting calculations, there are some introductions of operators and notations. A color parameter consists of four floating point elements, R, G, B, and A, where R, G, and B mean the elements of the RGB color model. A position parameter consists of four floating point elements, *x*, *y*, *z*, and *w*, that specify a position in the object coordinates. If c_1 and c_2 are colors defined by RGB color model (without the A element), where $c_1 = (r_1, g_1, b_1)$ and $c_2 = (r_2, g_2, b_2)$, then we define

$$c_1 * c_2 = (r_1 r_2, g_1 g_2, b_1 b_2).$$

If d_1 and d_2 are directions, then we define

$$d_1 \otimes d_2 = max\{d_1 \cdot d_2, 0\}.$$

If P_1 and P_2 are points then $\|P_1P_2\|$ is the distance between P_1 and P_2, and $\overrightarrow{P_1P_2}$ is the unit vector that points from P_1 to P_2. If P_2 has a zero *w* coordinate and P_1 has a non-zero *w* coordinate, then $\overrightarrow{P_1P_2}$ is the unit vector corresponding to the direction specified by the *x*, *y*, and *z* coordinates of P_2; if P_1 has a zero *w* coordinate and P_2 has a non-zero *w* coordinate, then $\overrightarrow{P_1P_2}$ is the unit vector that is the negative of that corresponding to the direction specified by P_1. If both P_1 and P_2 have zero w coordinates, then $\overrightarrow{P_1P_2}$ is the unit vector obtained by normalizing the direction corresponding to $P_2 - P_1$. If d is an arbitrary direction, then define $\hat{d}$ be the unit vector in d's direction. Finally, let V be the point corresponding to the vertex being light, and n be the corresponding normal, and let P_e be the eye-point.

The color c produced by lighting a vertex is defined the following description:

$$c = e_{cm} + a_{cm} * a_{cs} + \sum_{i=0}^{n-1} (att_i)(spot_i)\left[a_{cm} * a_{cli} + \left(n \otimes \overrightarrow{VP_{pli}}\right) d_{cm} * d_{cli} + (f_i)\left(n \otimes \hat{h}_i\right)^{s_{rm}} s_{cm} * s_{cli}\right]$$

where

$$f_i = \begin{cases} 1, & n \otimes \overrightarrow{VP_{pli}} \neq 0, \\ 0, & \text{otherwise,} \end{cases}$$

$$h_i = \begin{cases} \overrightarrow{VP_{pli}} + \overrightarrow{VP_e}, & v_{bs} = \text{TRUE}, \\ \overrightarrow{VP_{pli}} + \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T, & v_{bs} = \text{FALSE}, \end{cases}$$

$$att_i = \begin{cases} \dfrac{1}{k_{0i} + k_{1i}\left\|VP_{pli}\right\| + k_{2i}\left\|VP_{pli}\right\|^2}, & \text{if } P_{pli}\text{'s}\, w \neq 0, \\ 1.0, & \text{otherwise,} \end{cases}$$

$$spot_i = \begin{cases} \left(\overrightarrow{P_{pli}V} \otimes \hat{s}_{dli}\right)^{s_{rli}}, & c_{rli} \neq 180.0, \overrightarrow{P_{pli}V} \otimes \hat{s}_{dli} \geq cos(c_{rli}), \\ 0.0, & c_{rli} \neq 180.0, \overrightarrow{P_{pli}V} \otimes \hat{s}_{dli} < cos(c_{rli}), \\ 1.0, & c_{rli} = 180.0, \end{cases}$$

Parameter	**Type**	**Description**
Material Parameters		
a_{cm}	color	ambient color of material
d_{cm}	color	diffuse color of material
s_{cm}	color	specular color of material
e_{cm}	color	emissive color of material
s_{rm}	real	specular exponent (range: [0.0, 128.0])
a_m	real	ambient color index
d_m	real	diffuse color index
s_m	real	specular color index
Light Source Parameters		
a_{cli}	color	ambient intensity of light *i*
$d_{cli}\,(i{=}0)$ $d_{cli}\,(i{>}0)$	color color	diffuse intensity of light 0 diffuse intensity of light *i*
$s_{cli}\,(i{=}0)$ $s_{cli}\,(i{>}0)$	color color	specular intensity of light 0 specular intensity of light *i*
P_{pli}	position	position of light *i*
s_{dli}	direction	direction of spotlight for light *i*
s_{rli}	real	spotlight exponent for light *i* (range: [0.0, 128.0])
c_{rli}	real	spotlight cutoff angle for light *i* (range: [0.0, 90.0], 180.0)
k_{0i}	real	constant attenuation factor for light *i* (range: [0.0, ∞))
k_{1i}	real	linear attenuation factor for light *i* (range: [0.0, ∞))
k_{2i}	real	quadratic attenuation factor for light *i* (range: [0.0, ∞))
Lighting Model Parameters		
a_{cs}	color	ambient color of scene
v_{bs}	boolean	viewer assumed to be at (0, 0, 0) in eye coordinates (TRUE) or (0, 0, ∞) (FALSE)
t_{bs}	boolean	use two-sided lighting mode

Table 3-1 Summary of lighting parameters.

3.3.3.4 Clipping

The user defined clip plane will be described as an array of four floating point values, These are the coefficients of a plane equation in *object coordinates*. The inverse of the current *model-view matrix* is applied to these coefficients, at the time the plane is specified, as:

$$\begin{pmatrix} p_1' & p_2' & p_3' & p_4' \end{pmatrix} = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \end{pmatrix} M^{-1}.$$

To obtain the plane equation coefficients in *eye coordinates*. All points with *eye coordinates* that satisfy

$$\begin{pmatrix} p_1' & p_2' & p_3' & p_4' \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} \geq 0$$

lie in the half-space defined by the plane; points that do not satisfy this condition do not lie in the half-space.

If the primitive under this consideration is a point, the clipping will pass it if it satisfies this condition; otherwise, it will be discarded. If the primitive is a line, the clipping does nothing to it if the two endpoints of the line are entirely lie in the half-space, and discards if the endpoints are entirely out of the half-space. If part of the line segment lies in the half-space and part lies outside, then the line segment is clipped and new vertex coordinates will be computed for one vertices of the endpoints.

This clipping computation will produce a value, $0 \leq t \leq 1$, for each clipped vertex. If the coordinates of a clipped vertex are *P* and the original vertex's coordinates are *P₁* and *P₂*, then t is given by

$$P = tP_1 + (1 - t)P_2.$$

If the primitive is a polygon, then it is passed if all the edges of the polygon lie entirely inside the clip volume and discarded if all edges lie outside. If the polygon needs to be clipped, all edges which lie outside will be discarded, and there will be new edges that lie along the clip volume's boundary. Because polygon connectivity must be maintained, these clipped edges are connected by the new edges.

3.3.4 Implementation Problems

Developing such a 3D graphics library using Java programming language seems a tough task. During the design phase, we have some problems as following:

1. **Performance challenge** – On the underlying graphics kernel, we put most of our efforts here, since the performance is a great challenge for Java applets and also for general purpose software-based 3D graphics engines. For instance, drawing functions are subdivided into several smaller ones to optimize these functions and class inheritance is used to avoid the use of "if-then-else" statements in using these functions. We also refer to Graphics Gems [40][41][42] for performance enhancements. For the detail descriptions of the performance enhancement, please refer to section 3.3.6.
2. **Size consideration** – The byte-code size of the 3D graphics library is also a serious problem. Because the 3D graphics library will be downloaded in the run time, when end users run the 3D graphics applications developed with it. If the size of this library is too large, end users will wait for a long time for downloading the library. But, the size of the library and the performance of it seems be constrained, sometimes the size and the performance considerations are trade-off.
3. **Java is an object-oriented programming (OOP) language** – Because OpenGL API is defined for C programming language, while Java is an OOP language. We must design the API based on the OOP philosophy of Java, that is, the graphics engine should be designed as several classes, like Figure 3-5.
4. **No pointer data type in Java** – Pointers are very useful for programming. For instance, a drawing command in OpenGL may be executed immediately or be postponed in a display list, depending on the state of the graphics kernel. In C, we can simply use function pointers to solve this problem. In Java, however, since there are no pointers, we use class inheritance instead. Here, we use class reference instead of using

structure pointers, and use the Vector class in Java instead of using linked list which is also a useful data structure usually used by programmers.

5. **Java is only a virtual machine –** Java is not a real machine, but just a virtual machine. The machines which the Java programs run on are like the devices of this virtual machine. Therefore, we can not write any assembly routines for performance enhancement because we do not know which machine the library will be run on. Even if the real machine has hardware enhancement for 3D graphics algorithms, we can not use the feature, yet. Like our performance evaluation, when we test some examples on a machine with a hardware accelerated display card, all the 3D graphics enhancement features of the card are negligible for the examples.

3.3.5 The Differences Between JavaGL and OpenGL

Although the development of JavaGL follows the specification of OpenGL, there are still some differences between JavaGL and OpenGL due to the constraints of Java programming language.

1. **JavaGL has no structure –** Because Java is an OOP language, there is no structure data type as in C programming language. If there is a structure defined in OpenGL, there will be a class in JavaGL similar to the field definition of the structure. For example, there is a structure called GLUquadricObj in GLU for describing how a quadric should be constructed and rendered. Since JavaGL has no structure, there is a class with the same name and is put in javagl.glu.GLUquadricObj. Users can use this class instead of using the structure.

2. **JavaGL is platform independent –** OpenGL is a platform independent 3D graphics library, the primitive data type defined in OpenGL, for example GLint, is not the real primitive data type of the platform, OpenGL will convert them to the real primitive data type in the compile time. On the other hand, Java is also platform independent, but JavaGL does not need to convert the platform independent primitive data type defined in JavaGL to the primitive data type defined in Java. That is, JavaGL uses the

primitive data type defined in Java, and does not use the primitive data type defined in OpenGL.

3. **JavaGL has no dithering mode** – There is a rendering mode called *dithering*, which is used on the machine that does not support the true color mode, but it is not necessary for JavaGL to concern this problem because Java will handle it. When we want to draw something on the screen, all we have to do is just writing the true color to screen, if the real machine does not support the true color mode, Java will draw the dithered color on the screen.

4. **JavaGL has no single buffer mode** – Because JavaGL can not access real window framebuffer of the display card, JavaGL does not support single buffer mode. For the performance concern, if all the drawing works are done in the background, the drawing works will be faster than done in the foreground directly.

5. **JavaGL is a class and is designed for Java programming language** – JavaGL is designed for Java programming language, which is an OOP language. JavaGL itself is a class from programmers' view. To use the constants defined in OpenGL, programmers must use GL.GL_*CONSTANTS* instead of GL_*CONSTANTS* in C programming language.

3.3.6 Performance Enhancement

Performance is the great challenge for Java-based applications and also for general purpose software-based 3D graphics engines. Therefore, performance is also the great challenge for JavaGL which is a Java-based software 3D graphics library. Many methods are used to enhance the performance. However, JavaGL is a graphics library on Internet, that is, to make users download the library in the run time, the size of this library can not be too large. In other words, the size of the library is also needed to be optimized. In some cases, the size and the performance are trade-off.

1. **Utilize Java functions** – If there is a well-designed Java function, we will use the function instead of re-writing a simple one by ourselves. Because Java is like the

bridge between JavaGL to several platforms like the device drivers of several adapters. If some of the Java functions are implemented by using native codes or system calls, there will be a shorter path between JavaGL to several platforms.

2. **Use class inheritance to avoid "if-then-else" statements –** If we did not use class inheritance to avoid "if-then-else" statements, there must be several conditional jumpers all over the library, because the graphics library must determine which status is the current status, this is time consuming. Since many conditions will always need to be determined only once, after deciding which status is the current status, the status will not be changed and the graphics library will not need to determine the current status. For example, when implementing the display list, we set a flag to indicate whether the rendering commands are to be stored or to be executed. Therefore, for each rendering command, we need to check if the flag is set or not by "if-then-else" statements, and these "if-then-else" statements will slow down the execution speed. To solve this problem, we utilize class inheritance to avoid the use of "if-then-else" statements. For example, the class for the display list and the class for the executed functions are both inherited from the same parent class. After the dispatcher checks the flag, all the following rendering commands are realized in either the display list or the executed functions without any further checks of the flag.

3. **Use variables or tables to avoid re-calculation –** This is a common method which is always used by programmers. Because some parameters, like in the 3D transformation or in the lighting calculation, need complex calculations, but only once. Using variables or tables to register the parameters will save the re-calculation time.

4. **Make frequently used routines faster –** Polygon rasterization, shading, depth testing, clipping, etc., are frequently used routines. They are always the bottle necks of all the 3D graphics library, so we put most of our efforts to optimize these routines by introducing faster algorithms and by manual code optimization.

5. **Divide frequently used routines into smaller ones –** Some routines are the functions which will be called several times, to optimize all the drawing functions is therefore important, but many such routines have much unnecessary codes in them. For example, to fill a polygon, we must do the color interpolation if the polygon is filled by smooth

shading. If the polygon only needs flat shading, the color interpolation is not required. For this situation, we categorize all the drawing functions into several smaller ones, such as the drawing function with depth testing or without depth testing, the drawing functions with flat shading or smooth shading, and make these functions to be optimized and use class inheritance to use these functions.

6. **Group rarely used routines into a larger one –** When we divide the frequently used routines into smaller ones, the size of the library may be getting large. This is the side effect of performance enhancement. To reduce the total code size of the library, we examine all routines, and combine some similar routines that are rarely used into a larger one, contrarily. For example, we had two routines for rendering, including one with clipping and the other one without clipping originally. Since the former routine is mostly used, we combine these two routines, and optimize the conditional testing to redirect a rendering command to an appropriate code segment.

3.4 Results

The newest version of JavaGL is version 1.0 beta 3, and has been released in March 15th, 1997. The following descriptions are all for this version of JavaGL. JavaGL has been tested on several platforms and compiled by several compilers, such as SUN JDK 1.0.2 on SUN Solaris 2 and Microsoft Windows 95, SGI Cosmo Code 2.0 [43] on SGI IRIX 5 and Symantec Café 1.51 [44] and Microsoft Visual J++ 1.0 [45] on Microsoft Windows 95.

3.4.1 Supported Functions

Currently, we have implemented over **160** OpenGL functions in JavaGL (include GLAUX, GLU, and GL), including functions for two dimensional (2D)/3D model transformation, 3D object projection, depth buffer, smooth shading, lighting, material, display list and selection. The functions not yet supported so far are mainly for anti-aliasing and texture mapping. To provide the *OpenGL Utility Toolkit* (GLUT) [46] using JavaGL is also our future work.

3.4.2 Performance Evaluation

To test JavaGL's capability, we have provided **16** examples on WWW. These examples are selected from the *OpenGL Programming Guide*, which is the official programming guide of OpenGL. All of them can be executed on Internet via the JavaGL web site – Http://www.cmlab.csie.ntu.edu.tw/~robin/JavaGL/Example.html.

To evaluate JavaGL's performance, we use a testing program which renders twelve spheres with different materials, each sphere contains **256** polygons, as shown in Plate 1. This program is an example in the *OpenGL Programming Guide* (code from Listing 6-3, pp. 183-184, Plate 16). We execute the testing program on both a SUN Ultra-1 workstation and an Intel Pentium-200 PC. For comparison, we also rewrote the same program in Mesa 3-D graphics library [47], which is a software-based 3D graphics library with an API similar to that of OpenGL using C programming language, and measure the rendering time. We also rewrote the same program in hardware accelerated OpenGL on both the platforms, as listed in Table 3-2 and Table 3-3.

On the SUN workstation, the testing program with Mesa is about **4** times faster than that with JavaGL, as claimed by SUN that Java is about 20 times slower than C [48]. But, the performance can be further improved if a better Java interpreter or compiler exists. The testing program with hardware accelerated OpenGL [49] is of course the fastest one.

Graphics Library	**Environment**	**Rendering Time (ms)**
JavaGL 1.0 beta 3	SUN JDK 1.0.2 SUN JIT 1.0.2	**4984**
Mesa 2.1	GNU C 2.7.2.1	**1085**
OpenGL for Creator3D 1.0	GNU C 2.7.2.1 Hardware accelerated (Sun Creator3D)	**138**

Table 3-2 A performance comparison on a workstation. The workstation configuration is SUN Ultra-1 Model 170E, 128 MB memory, 24-bit display, Sun Solaris 2.5.1.

On the PC platform, we execute the testing program using the SUN JDK 1.0.2 and the Symantec Café 1.51 with JIT 2.0 beta 3. By using the Just-In-Time (JIT) compiler [50], there is over 4 times performance speedup. As the performance comparison on a SUN workstation, the testing program with the hardware accelerated OpenGL [51] is about many times faster than that with JavaGL.

Graphics Library	**Environment**	**Rendering Time (ms)**
JavaGL 1.0 beta 3	Sun JDK 1.0.2	**16700**
JavaGL 1.0 beta 3	Symantec Café 1.51 Symantec JIT 2.0 beta 3	**4070**
OpenGL for Windows 95 1.0	Microsoft Visual C++ 4.2 Hardware accelerated (ET-6000)	**189**

Table 3-3 A performance comparison on a PC. The PC configuration is Intel Pentium-200 CPU, 64 MB memory, 24-bit display, Microsoft Windows 95.

Plate 2 is the quality comparison between JavaGL and Mesa. There is a teapot which contains **604** triangles and is rendered by JavaGL and Mesa. It takes **550 ms** for JavaGL on an Intel Pentium-200 PC as in the above table.

3.4.3 How to Use JavaGL

1. **JavaGL in the server site –** The web master must put JavaGL in the same directory of the 3D graphics applications which are developed with JavaGL. Then, the users across the Internet can use the applications by downloading JavaGL in the same time when they download the byte-code of the applications. The size of JavaGL is only **58 KB**, this is very small and does only need to be downloaded only once. If the users use other 3D graphics applications which are using JavaGL, JavaGL does not need to be downloaded again.

2. **JavaGL in the client site –** If the users do not want to download JavaGL every time when they use the 3D graphics applications developed with JavaGL, they can download the newest JavaGL before using these applications. They can download the compressed library of JavaGL from Ftp://ftp.cmlab.csie.ntu.edu.tw/pub/cml/JavaGL/javagl.zip, uncompress it and add the path of JavaGL to the environment variable **CLASSPATH**. Then, when the users use the 3D graphics applications, they only need to download the byte-code of these applications. Because JavaGL is written by Java programming language, this 3D graphics library can be used on any platforms.

3. **To the programmers –** Please downloads the compressed library of JavaGL from the same position of the above and do the same thing. Then, the programmers can use this library like using OpenGL. To use JavaGL in the Java applets, the programmers must import the classes of JavaGL first, then declare the instances of the classes, then calling the member functions of the classes as calling the functions of OpenGL. There is a simple example in Figure 3-9, and the same example written with OpenGL in Figure 3-8.

```
#include <stdlib.h>

// must include GL/gl.h....
#include <GL/gl.h>
#include "glaux.h"

int main (int argc, char** argv) {
    auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
    auxInitPosition (0, 0, 500, 500);
    auxInitWindow (argv[0]);

    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    glBegin (GL_POLYGON);
        glVertex2f (-0.5, -0.5);
        glVertex2f (-0.5, 0.5);
        glVertex2f (0.5, 0.5);
        glVertex2f (0.5, -0.5);
    glEnd ();
    glFlush ();

    sleep (10);
}
```

Figure 3-8 A simple source code of OpenGL using the auxiliary library. This program is an example in *OpenGL Programming Guide* (code from Listing 1-2, pp. 13).

```
import java.applet.Applet;
import java.awt.*;

// must import javagl.GL....
import javagl.GL;
import javagl.GLAUX;

public class simple extends Applet {
    GL myGL = new GL ();
    GLAUX myAUX = new GLAUX (myGL);

    public void init () {
        myAUX.auxInitPosition (0, 0, 500, 500);
        myAUX.auxInitWindow (this);
    }

    public void paint (Graphics g) {
        myGL.glXSwapBuffers (g, this);
    }

    public void start () {
        myGL.glClearColor ((float)0.0, (float)0.0, (float)0.0, (float)0.0);
        myGL.glClear (GL.GL_COLOR_BUFFER_BIT);
        myGL.glColor3f ((float)1.0, (float)1.0, (float)1.0);
        myGL.glMatrixMode (GL.GL_PROJECTION);
        myGL.glLoadIdentity ();
        myGL.glOrtho ((float)-1.0, (float)1.0, (float)-1.0, (float)1.0, (float)-1.0, (float)1.0);
        myGL.glBegin (GL.GL_POLYGON);
            myGL.glVertex2f ((float)-0.5, (float)-0.5);
            myGL.glVertex2f ((float)-0.5, (float)0.5);
            myGL.glVertex2f ((float)0.5, (float)0.5);
            myGL.glVertex2f ((float)0.5, (float)-0.5);
        myGL.glEnd ();
        myGL.glFlush ();
    }
}
```

Figure 3-9 A simple source code of JavaGL according to Figure 3-8.

To use JavaGL, the program must import the JavaGL class named “javagl.GL”, if the programmers want to use GLU or GLAUX, they must import “javagl.GLU” or “javagl.GLAUX”, like include “GL/gl.h”, “GL/glu.h” and “glaux.h”. Because GL, GLU and GLAUX are three different classes, the programmers must declare the instances of these classes. GLU and GLAUX are two classes which will call the rendering functions of GL, when declare these two instances of them, we must give them the instance of GL. Then, they can use all the JavaGL functions as using the OpenGL functions. Like the above figures, comparing the two files, all the functions of JavaGL and OpenGL are one-by-one mapping.

3.4.4 JavaGL on Internet

There are over **1000** people have visited the JavaGL web site from Jan. 1 1997’ to Apr. 30 1997’ and over **135** organizations all over the world, include SUN, IBM, SAS, HP, Intel, S3, NASA, CERN, Purdue, CalTech, GaTech, Texas, Columbia, Ohio, Tokyo, Aizu, Taiwan, universities, have downloaded JavaGL from our ftp site from Jan. 1 1997’ to Apr. 30 1997’. The following log is the transfer log of downloading JavaGL.

cml19.csie.**ntu.edu.tw**
lamb.**sas.com**
brook.cwr.**uwa.edu.au**
203.71.132.23
pm-ok-0-24.athenet.net
as2511-28.sl017.cns.**vt.edu**
wwwdec.**cern.ch**
203.19.19.1
darkwing.cs.**utexas.edu**
61006d0011ny.concentric.net
194.206.118.19
TS1-Port27.Flagstaff.InfoMagic.net
130.209.133.55
gatekeeper.cms-stl.com
paloalto.access.**hp.com**
erdos.csie.**ncnu.edu.tw**
arn101.neocom.ca
dalet.belnet.be
snag.ug.eds.com
203.24.242.66
mas-isdn02-00.dial.xs4all.nl
saturn5.**Sun.COM**
charon.adisys.com.au
d156-1.cpe.Sydney.aone.net.au
linux1.HITC.COM
serin.deep.net
lemming.ericsson.cz
www-proxy-3.maz.net
208.193.160.89
207.106.30.102
peso.**ntu.ac.sg**
jeeves.mcs.anl.gov
insux1.cs.**jhu.edu**
bootes.ece.**arizona.edu**
arn101.neocom.ca
jesse.cis.**famu.edu**
paul.oz.net
sn.star.be
ant1.gulliver.fr
centaure.int-evry.fr
139.57.137.215
aegis.mcs.**kent.edu**
raf2p35.rafael.co.il
mp425.hknet.com
facesoft.port.net
algol.info.**ucl.ac.be**
206.251.226.48
mpngt1.ny.us.**ibm.com**

ple-ca17-13.ix.netcom.com
205.133.127.101
ccmppp05.ecn.**purdue.edu**
styx.**uwa.edu.au**
avery-49.**caltech.edu**
gr.**korea.ac.kr**
fw02-1.lerc.**nasa.gov**
intel-desk1.jf.**intel.com**
141.85.111.5
beker_l1.ece.**sc.edu**
a237-159.TS.**nctu.edu.tw**
sparc1.cs.**chpi.edu.tw**
204.215.156.90
140.112.18.248
xox.com
felix.cc.**gatech.edu**
hologram.cs.**wmich.edu**
mailhost.craycom.co.uk
gate.**dimensionx.com**
RAFFAEL.UNI-MUENSTER.DE
ds370613.**uncg.edu**
krebs.inet.dkfz-heidelberg.de
136.145.57.205
wyrm.its.**uow.edu.au**
21.p1.Nalice.MIA.Icanect.Net
200.9.61.125
ip23.bellsouth.cl
207.60.129.7
hei01.hei.com
internet.corelus.com
143.177.122.125
rocky.cs.**ucsb.edu**
physnuc.phy.**ulaval.ca**
dial2.alibrary.com
iesun6.en.tamuk.edu
207.17.218.207
194.94.26.44
kpc3.itia.mi.cnr.it
194.224.216.120
erchh001.nt.com
wsvst31.site.uni-wuppertal.de
austin.nk-exa.co.jp
du174-4.ppp.algonet.se
gateway.**s3.com**
tam6.mech.**nwu.edu**
rb-tc-ppp30.monmouth.com
async10.login.it

208.13.0.33
EFrench.burl.**brad.ac.uk**
sun26.hrz.th-darmstadt.de
141.30.118.16
gate.CRT.**UMontreal.CA**
ts23-7.homenet.**ohio-state.edu**
128.164.17.35
client-120-1.bellatlantic.net
ranger.demon.co.uk
ow68.wins.**uva.nl**
moby.idirect.com
halley.med.**jhu.edu**
c11-a.snvl1.sfba.home.com
ip226.lax.primenet.com
topher.wbr.rhno.**columbia.edu**
www.tui.co.uk
otto.pvv.ntnu.no
blitz.**iware.com**
goby.gen.**u-tokyo.ac.jp**
proxy0.FHG.DE
207.216.76.72
ircpc01.**swan.ac.uk**
inet-gw.indy.tce.com
205.153.215.128
192.101.181.123
theme6.owt.com
L170A.borg.com
muon.neuro.**soton.ac.uk**
cust91.max17.los-angeles.ca.ms.uu.net
ppp89.lewiston.com
zing.ncsl.nist.gov
196.31.58.190
206.79.6.241
mcase.cecer.**army.mil**
pc01lnx.intersoft.hu
csesun01.**u-aizu.ac.jp**
edradour.hgu.**mrc.ac.uk**
srini.dptechnology.com
ACCESSAIG1.AIG.COM
SLIP3-12.DIALIN.**UIC.EDU**
spleen.lim.univ-mrs.fr
hastur.finest.tm.fr
lynx.visix.com
128.32.37.200
proxyweb.worldnet.net
riogrande.se.houston.geoquest.slb.com
cagw2.att.com

Figure 3-10 The transform log of getting JavaGL from Ftp://ftp.cmlab.csie.ntu.edu.tw.

3.4.5 The Progression of JavaGL

We have released the first beta testing version of JavaGL at November 1996 and released the third beta testing version at March 1997. Among the four months, there is much advancement of JavaGL, Java technologies, include Java compiler and JIT, and the computer hardware technologies, include CPU and RAM.

Version	JavaGL 1.0 beta 1		JavaGL 1.0 beta 3	
Release Date	November 1996		March 1997	
Size (KB)	**116.5**		**58**	
Functions	**130**		**160**	
Time (ms)	**24391**	**8240**	**4984**	**4070**
Platform	SUN Sparc-20 64 MB memory	Intel Pentium-100 32 MB memory	SUN Ultra-1 128 MB memory	Intel Pentium-200 64 MB memory
Interpreter	SUN JDK 1.0.2	Symantec Café 1.5	SUN JDK 1.0.2 SUN JIT 1.0.2	Symantec Café 1.51 Symantec JIT 2.0 beta3

Table 3-4 Table of JavaGL progression. The testing program renders twelve spheres, one sphere contains **256** polygons, as shown in Plate 1.

As in Table 3-4, **30** functions are increased in JavaGL, but the size is reduced to a half of the size of the first version of JavaGL. Currently the whole package size of JavaGL is about **58 KB**, which is affordable for a network transmission via Internet, or even via a modem. In the workstation, after four months, we got SUN Ultra-1 with 128 MB RAM, and SUN JIT 1.0.2, so JavaGL gains five times speed up. In the PC platform, Intel Pentium CPU has sped up from 100 MHz to 200 MHz, memory has increased two times, and we got the Symantec JIT 2.0 beta 3, so JavaGL gains two times speed up.

Java programming language is a hot topic in the computer world, there are many companies, universities, and organizations keep their focuses and do their best on the development of Java programming language. The performance of Java is the most important issue for them. The hardware enhancement is also rapid these years. Hence, the performance of JavaGL can be sped up five times in the short four months.

3.5 Applications Using JavaGL

JavaGL has already used to develop some applications in our laboratory and the Raster Graphics course in our department. The following sections describe two of them, one is a General Viewing System (GVS) in JavaGL which is a utility for viewing a Triangle file format and the other is a browser for viewing VRML 2.0 file format.

3.5.1 A General Viewing System (GVS) in JavaGL

GVS is used to view a simple file format of 3D object. The simple file format is called the Triangle file (*.tri). People use GVS can view a 3D object of this file format from all view points. We have developed GVS in both JavaGL and OpenGL. The two versions of GVS use the same OpenGL functions to keep the justness of performance evaluation. GVS supports all control functions like rotations, translations, and zoom in, zoom out of the 3D object. To use GVS, please enter the web site: Http://www.cmlab.csie.ntu.edu.tw/~robin/JavaGL/Example/GVS.

3.5.1.1 File Format

The Triangle file format is a sequence of the following blocks:

Triangle
frontcolor_**R** **f**rontcolor_**G** **f**rontcolor_**B** **b**ackcolor_**R** **b**ackcolor_**G** **b**ackcolor_**B**
vertex**1_X** **v**ertex**1_Y** **v**ertex**1_Z** **n**ormal**1_X** **n**ormal**1_Y** **n**ormal**1_Z**
vertex**2_X** **v**ertex**2_Y** **v**ertex**2_Z** **n**ormal**2_X** **n**ormal**2_Y** **n**ormal**2_Z**
vertex**3_X** **v**ertex**3_Y** **v**ertex**3_Z** **n**ormal**3_X** **n**ormal**3_Y** **n**ormal**3_Z**

Figure 3-11 The Triangle block of the Triangle file format.

The “Triangle” is the beginning tag of each block, after the “Triangle”, all the 24 numbers describe the parameters of a triangle include the foreground color and background color as RGB color model, the 3D positions and normal vectors of the three pixels of this triangle. Because each 3D object can be divided into several small triangles, putting all the triangles in the Triangle file in the 3D space will reconstruct the 3D object.

3.5.1.2 Performance Evaluation

We have developed GVS with JavaGL and OpenGL and do the performance evaluation by viewing a teapot model of **604** triangles as shown in Plate 2. To render the teapot using the GVS with JavaGL takes **550 ms** on an Intel Pentium-200 PC as in Table 3-3. Using the GVS with OpenGL to render the same teapot file with Mesa 3-D graphics library takes **80 ms** on a SUN Ultra-1 workstation as in Table 3-2.

Plate 3 is a complex model that contains **5273** triangles, and the rendering time is **6150 ms** on an Intel Pentium-200 PC as in Table 3-3. Because we implement JavaGL with pure Java, an application written with JavaGL can be run on different platforms without any additional local 3D graphics capabilities. The complex model in Plate 3 is our computer science department building, which is rendered by an applet running in a Netscape Communicator web browser, where all the 3D graphics functions are obtained directly from a server.

3.5.2 JavaVRML - A VRML 2.0 Browser Using JavaGL

VRML is a common standard file format which is used to construct a virtual world or 3D objects on Internet. To view a file with VRML file format, a useful browser is needed. There are some VRML browsers all over the world, but all of them are platform dependent because they do not have a real platform independent and efficient 3D graphics engine. To develop a real platform independent VRML browser, we choose Java programming language to be the development environment and JavaGL to be the 3D graphics engine. This Java-based VRML browser, named JavaVRML, now supports VRML 2.0 file format and can be run as an applet on all kinds of the Java enabled platforms.

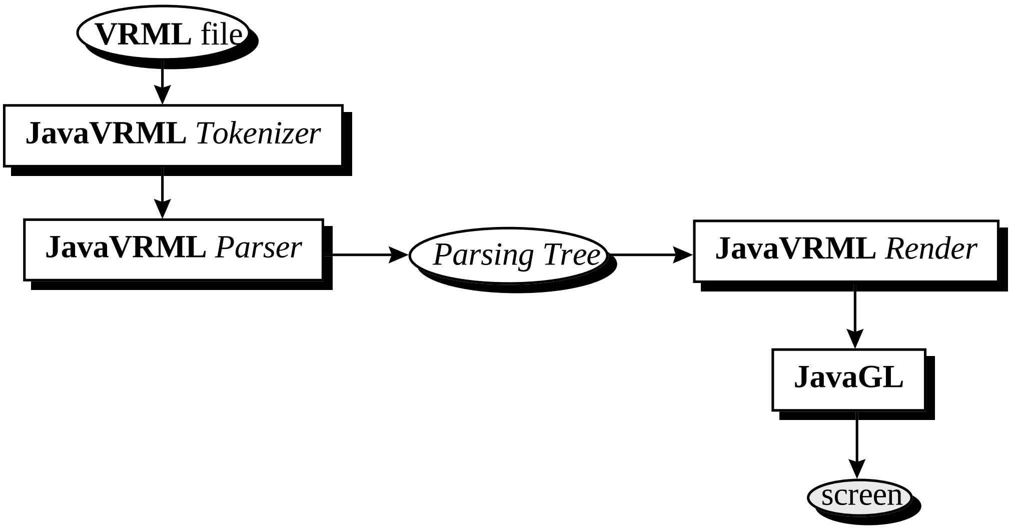


Figure 3-12 The system hierarchy of JavaVRML.

To view a VRML file, we must parse the file first and construct a paring tree for rendering. The JavaVRML *Tokenizer* and *Parser* are all developed using Java and are only run once when reading the VRML file. The parsing tree is the main structure we used for every time rendering the 3D object or the virtual world defined in the VRML file. To render the parsing tree, we just put the parsing tree to the JavaVRML *Render*, it is developed using Java and JavaGL for the rendering engine. Then, users can see the VRML file on the screen and view it from all view points.

JavaVRML has not been completed yet, it only supports some usually used or some special nodes of the specification of VRML 2.0. But, users can already use it to view some simple VRML files on Internet via Http://www.cmlab.csie.ntu.edu.tw/~unicorn/VRML/browser.html.

Plate 4 is a room of VRML 2.0 file format rendered by JavaVRML on Netscape Communicator 4.0 Preview Release 2.

Chapter 4
JavaNL– A Network Library in Java

4.1 Introduction

When the three dimensional (3D) graphics applications based on JavaGL have been developed on Internet, there is usually interaction between users at the client sites and the application at the server site, but lack of interaction between the applications. So that we want to give the 3D graphics applications more interactivities and give them the network capability. To do this, we reference the specifications of Distributed Interactive Simulation (DIS) [52][53], which is a standard for interactive simulations in different machines and platforms on Wide Area Network (WAN).

Because this network library is also an additional part of JavaGL, to keep the platform independent feature of JavaGL, we also chosen Java for the programming language to develop the network library. Hence, we can use JavaGL and JavaNL to develop 3D graphics multiparticipant applications based on the two libraries.

4.2 Implementation Issues

4.2.1 Introduction to Distributed Interactive Simulation (DIS)

The basic architecture concepts of DIS are an extension of the Simulator Networking (SIMNET) program developed by the Advanced Research Project Agency (ARPA). The basic architecture concepts for DIS, add their implications as they apply to DIS, are:

1. *No central computer controls the entire simulation exercise* – DIS uses a distributed simulation approach in which the responsibility for simulating the state of each entity rests with separate simulation applications residing in host computers connected via a network.
2. *Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities* – Simulation applications (or simulations) are autonomous and generally responsible for maintaining the state of at least one entity.
3. *A standard protocol is used for communicating ground truth data* – Each simulation application communicates the state (which is herein called ground truth) of the entity it controls/measures (location, orientation, velocity, articulated parts position, etc.) to other simulations on the network.
4. Changes in the state of an entity are communicated by its controlling simulation application.
5. Perception of events of other entities is determined by the receiving application.
6. *Dead reckoning algorithms are used to reduce communications processing* – A method of position/orientation estimation, called dead reckoning, is used to limit the rate at which simulations must issue state updates for an entity.

4.2.2 DIS vs. JavaNL

It is our goal to design a network agent of network applications and offer a simple application programming interface (API) to make programmers develop the network applications as easy as possible. This network agent is called JavaNL, also Java based.

As the above descriptions, although DIS is a standard and has many advantages, it still has some problems to be implemented. First, the specifications of DIS are only IEEE Std 1278.1-1995 for application protocols and IEEE Std 1278.2-1995 for communication services and profiles. IEEE P1278.3 for exercise management and feedback has not been standardized, so we can not get this part of the specifications of DIS.

Because DIS itself is originally designed for the military use and seems not for the general use, but the concepts of DIS are still very nice to develop the network library on Internet, we choose to not follow the entire specifications of DIS to design the Java based network API, but just use the concepts of DIS to design JavaNL. The concepts of JavaNL following the specifications of DIS are as following:

1. There is no central computer that controls the entire simulation exercise.
2. Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities.
3. Changes in the state of an entity are communicated by its controlling simulation application.
4. Perception of events of other entities is determined by the receiving application.

As shown in Figure 4-1, each DIS application uses protocol data units (PDUs) to communicate with each other, and keeps all simulation information locally.

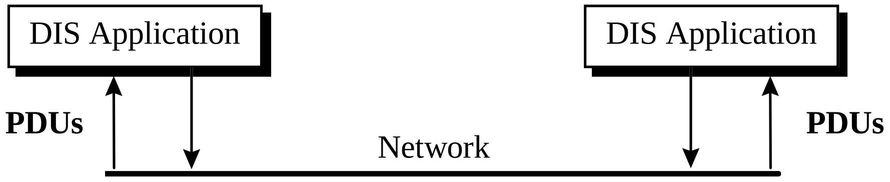


Figure 4-1 The control flow of DIS. A DIS application needs to maintain all the simulation information necessary, and uses PDUs to communicate with each other.

JavaNL follows the above guide line and adopts the PDU format for data exchange as described in the specifications of DIS. When one application wants to send some information to other ones, it only needs to tell JavaNL and JavaNL will send the information for it. When JavaNL receives the information from other applications, it will notify the application by sending some messages to it. The system hierarchy is shown in the following figure.

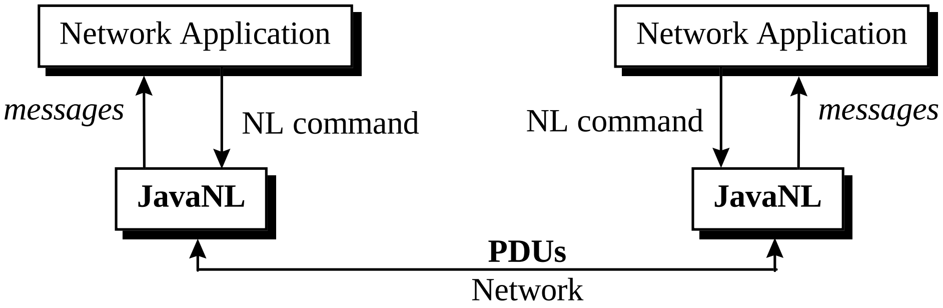


Figure 4-2 The control flow of JavaNL that provides PDU transmission capability.

4.2.3 The Concepts of JavaNL

The network applications using JavaNL will play the roles as the simulation applications and manager in DIS. There will be one simulation manager and several simulation applications in a simulation exercise, of course, the simulation manager is also a simulation application. When the simulation manager creates a simulation exercise, it is waiting for other simulation applications joining the simulation exercise. After the simulation manager creating the simulation exercise, it must register all the entities owned by each simulation application to the simulation exercise. When one simulation application joins the simulation exercise, the simulation manager will tell it all the information of this simulation exercise, includes how many simulation applications have already been in the simulation exercise and how to communicate with them, and tell the new simulation application to create the entities which are used in the simulation exercise controlled by each simulation application. Figure 4-3 is the flow chart of the behavior of the simulation manager and applications as a state machine.

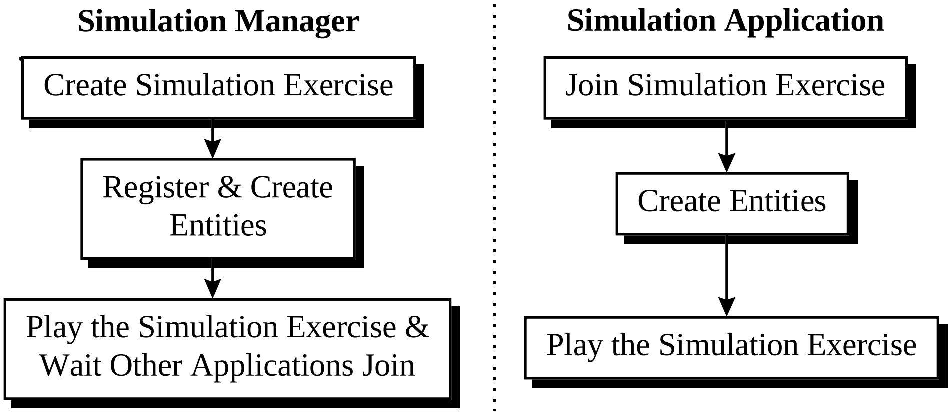

Simulation Manager
Create Simulation Exercise
Register & Create Entities
Play the Simulation Exercise & Wait Other Applications Join
Simulation Application
Join Simulation Exercise
Create Entities
Play the Simulation Exercise

Figure 4-3 The flow chart of the behavior of simulation manager and application.

The simulation manager is only needed at the beginning phase of the simulation exercise or when some simulation applications want to join the simulation exercise even when the simulation exercise has already been started. After all the simulation applications have joined in the simulation exercise, the role of the simulation manager is the same as other simulation applications, because they do not need simulation management any more. All the simulation applications exchange the PDU by themselves, even when the network connection is broken, all the simulation applications can also be run dividually.

4.2.4 PDU Sending and Receiving

There are 27 PDUs defined in the specifications of DIS, but some of them are designed for military use, so we just choose some useful PDUs to be the PDUs used in JavaNL. There are no PDUs for network management in all the 27 PDUs, hence, we follow the structure of the PDU format to construct the PDUs for JavaNL and modify the Entity State PDU to be enabled to transmit some customized data to other simulation applications.

The PDUs used by JavaNL from the 27 PDUs are:

a) Entity Information/Interaction
   1) Entity State PDU
   2) Collision PDU
d) Simulation Management

1) Start/Resume PDU
2) Stop/Freeze PDU
3) Acknowledge PDU
11) Create Entity PDU
12) Remove Entity PDU

The PDUs designed for JavaNL are:

g) Network Management
   1) Join Request PDU
   2) Join Accept PDU
   3) Join Reject PDU
   4) Disconnect PDU

Some of the PDUs are sent via the Transmission Control Protocol (TCP) and some are via the User Datagram Protocol (UDP) due to the specifications of DIS or the sending reliable concern and shown in the following table:

PDU Type	Service Requirement	Sending Protocol
Entity State PDU	Best Effort Multicast	UDP
Collision PDU	Best Effort Multicast	UDP
Start/Resume PDU	Best Effort Unicast / Best Effort Multicast	UDP
Stop/Freeze PDU	Best Effort Unicast / Best Effort Multicast	UDP
Acknowledge PDU	Best Effort Unicast / Best Effort Multicast	UDP
Create Entity PDU	Best Effort Unicast / Best Effort Multicast	UDP
Remove Entity PDU	Best Effort Unicast / Best Effort Multicast	UDP
Join Request PDU	Reliable Unicast	TCP
Join Accept PDU	Reliable Unicast	TCP
Join Reject PDU	Best Effort Multicast	UDP
Disconnect PDU	Best Effort Unicast / Best Effort Multicast	UDP

Table 4-1 The service requirements and sending protocol of PDUs.

In JavaNL, there is a network agent, named *nl_network_agent* for controlling the sending and receiving of the PDU. The *nl_network_agent* will fork two threads *nl_tcp_receiver* and *nl_udp_receiver*. These two threads are like the watching dog to snoop the network, if there is a TCP stream or a UDP datagram sent to this host computer, they will catch it and put it to the PDUInQueue as the PDU format. The *nl_network_agent* always polls the PDUInQueue, if there is a PDU comes, it will put a message to the MSGQueue to JavaNL or just deal the requirement by itself. The network application using JavaNL always also polls the JavaNL to perceive if there is a message comes. The *nl_network_agent* always also polls the PDUOutQueue to wait JavaNL put some PDUs, if there is a PDU appears in the queue, the *nl_network_agent* will call the *nl_tcp_sender* or *nl_udp_sender* to send the PDU due to the type of the PDU. If the application wants to send some information to other ones, it will call JavaNL commands to do this, and the JavaNL will put some PDUs in the PDUOutQueue, then the PDUs will be sent by the *nl_network_agent* through the above path.

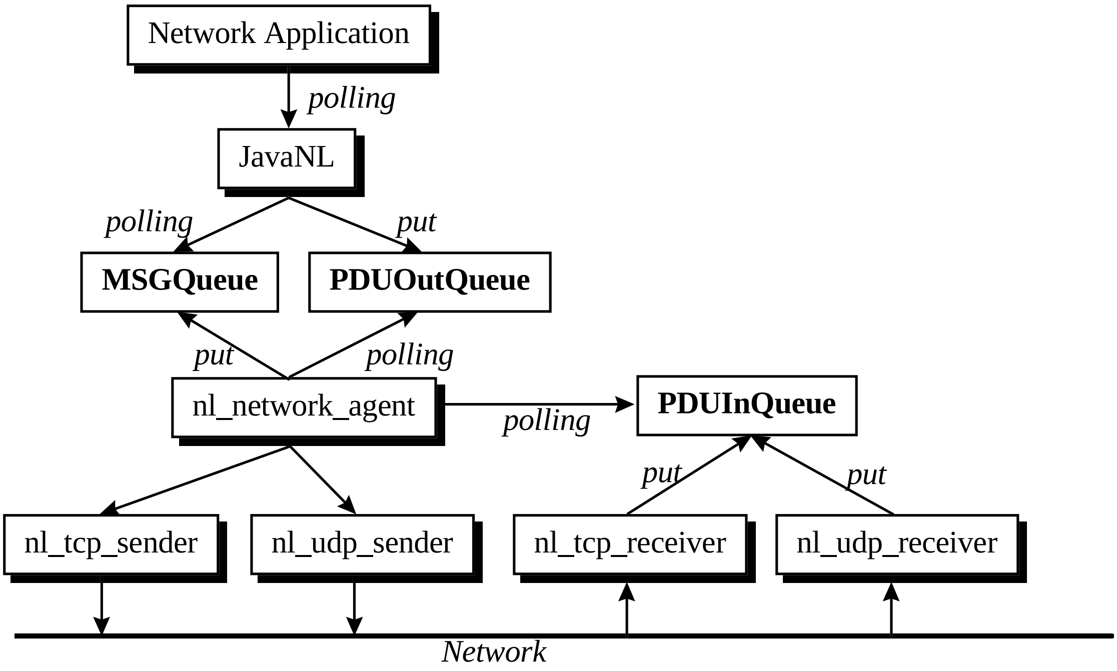


Figure 4-4 PDU sending and receiving of JavaNL.

4.2.5 Formats of modified PDUs

There are five modified PDUs used in JavaNL, they are Entity State PDU, Join Request PDU, Join Accept PDU, Join Reject PDU, and Disconnect PDU.

We appended the customized data format to the Entity State PDU as the following table:

Field size (bits)	**Entity State PDU fields**	
1152	Original Entity State PDU with no articulation parameter	
96 + 32i + 32j + 1k	Customized Information	Size of Integer Array (i) - 32-bit integer
		Size of Floating Array (j) - 32-bit integer
		Size of Boolean Array (k) - 32-bit integer
		Integer Array - i * 32-bit integer
		Floating Array - j * 32-bit floating point
		Boolean Array - k * Boolean number
Total Entity State PDU size = (1248 + 32i + 32j + 1k) bits where i, j, k are size of each array in the customized information field		

Table 4-2 The format of the modified Entity State PDU

The formats of the Join Request PDU, Join Accept PDU, Join Reject PDU, and Disconnect PDU are as the following tables:

Field size (bits)	**Join Request PDU fields**	
96	PDU Header	
112 + 16n	Site Information	Site - 16-bit unsigned integer
		Length of IP address(n) - 32-bit unsigned integer
		IP address - n * 16-bit unsigned integer
		TCP port - 32-bit unsigned integer
		UDP port - 32-bit unsigned integer
Total Join Request PDU size = (208 + 16n) bits where n is length of IP address		

Table 4-3 The format of the Join Request PDU

Field size (bits)	**Join Accept PDU fields**	
96	PDU Header	
112 + 16n	Site Information of Simulation Manager	Site - 16-bit unsigned integer
		Length of IP address(n) - 32-bit unsigned integer
		IP address - n * 16-bit unsigned integer
		TCP port - 32-bit unsigned integer
		UDP port - 32-bit unsigned integer
32	Number of Sites(m)	32-bit unsigned integer
(112+16n)m	Information of Sites	Site - 16-bit unsigned integer
		Length of IP address(n) - 32-bit unsigned integer
		IP address - n * 16-bit unsigned integer
		TCP port - 32-bit unsigned integer
		UDP port - 32-bit unsigned integer
Total Join Accept PDU size = [240 + 16n + (112 + 16n)m] bits where n is length of IP address m is the number of sites already in the simulation exercise		

Table 4-4 The format of the Join Accept PDU

Field size (bits)	**Join Reject PDU fields**
96	PDU Header
Total Join Reject PDU size = 96 bits	

Table 4-5 The format of the Join Accept PDU

Field size (bits)	**Disconnect PDU fields**
96	PDU Header
16	Site - 16-bit unsigned integer
Total Disconnect PDU size = 112 bits	

Table 4-6 The format of the Disconnect PDU

4.2.6 Network Management

When a simulation application creates a simulation exercise, it will become a simulation manager and wait for other simulation applications to join the simulation exercise. If there is a simulation application that wants to join the simulation exercise, it must know the Internet Protocol (IP) address and the TCP port number of the host computer of the simulation manager first, then send a Join Request PDU to the simulation manager with the network information including the IP address, TCP and UDP port number of itself. If the simulation manager agrees the join request from the simulation application, it will send a Join Accept PDU with the network information of all the simulation applications already in the simulation exercise, or just send a Join Reject PDU to reject the join request to the simulation application.

When the simulation manager sends a Join Accept PDU to a newly joined simulation application, it also sends a Join Request PDUs to all other joined simulation applications with updated simulation information to inform them there is a new simulation application joined this simulation exercise.

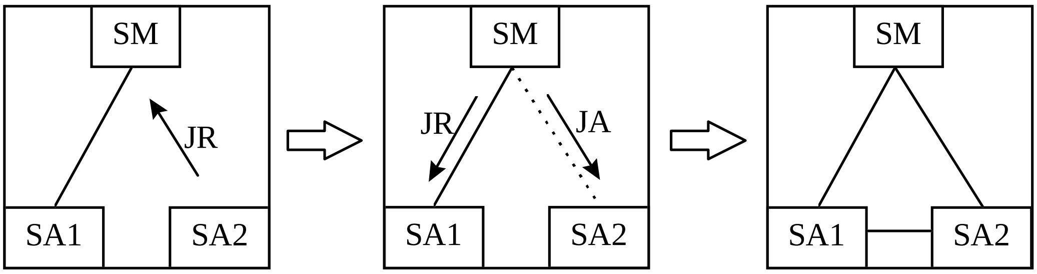


Figure 4-5 Join sequence in JavaNL, where SM is a simulation manager, SA1 and SA2 are simulation applications, JR means the Join Request PDU and JA means the Join Accept PDU. SM and SA1 are in a simulation exercise originally, and SA2 is the late join simulation application.

If a simulation application wants to leave, it sends a Disconnect PDU to the simulation manager, and the simulation manager cuts the connection, and broadcasts the Disconnect PDU to all other remain joined simulation applications.

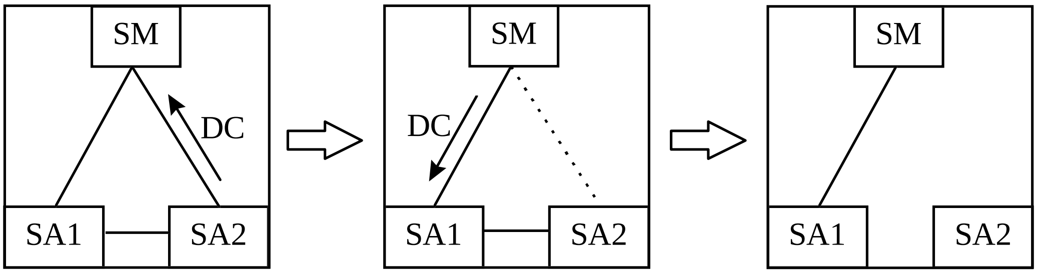


Figure 4-6 Disconnect sequence in JavaNL, where SM, SA1 and SA2 are the same in the Figure 4-5, and DC means the Disconnect PDU. SM, SA1 and SA2 are in a simulation exercise originally, and SA2 wants to leave the simulation exercise.

If the network is malfunction, the simulation application will not be able to communicate with some other ones, and will cut the communications to unreachable ones. If the network malfunction is perceived by the simulation manager while doing the network management work, it will try to notify other simulation applications by sending the Disconnect PDUs.

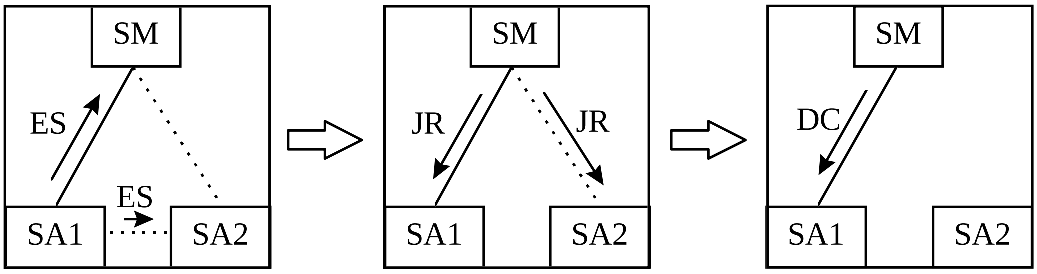


Figure 4-7 Disconnect sequence in JavaNL due to network malfunction, where SM, SA1, SA2 and DC are the same in the Figure 4-5, ES means the Entity State PDU and JR means the Join Request PDU. SM, SA1 and SA2 are in a simulation exercise originally, but SA2 is disconnected due to network malfunction. When SA1 can not send Entity State PDU to SA2, it will cut the connection with SA2. When SM can not send Join Request PDU to SA2, it will cut the connection with SA2 and broadcast the Disconnect PDU to other simulation applications in the simulation exercise.

4.2.7 Simulation Management

Once the network connection between all the simulation applications in a simulation exercise has been established, the simulation manager will tell all the simulation applications the initial or current situation of the simulation exercise. JavaNL of the simulation manager will send the Create Entity PDU to all the simulation applications to tell them to create the entities which have been registered by the simulation manager when it creates the simulation exercise.

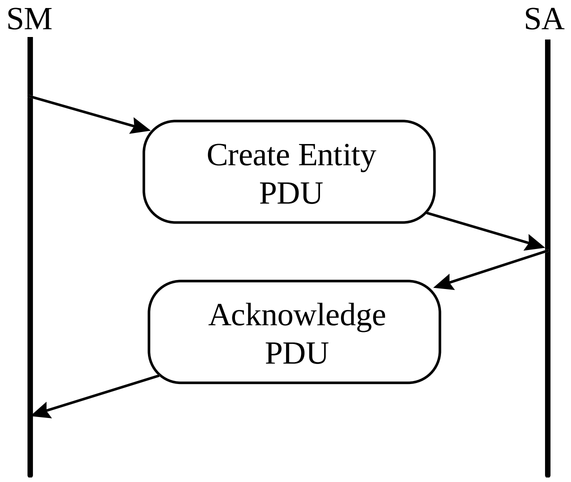


Figure 4-8 Entity creation, where SM is a simulation manager and SA is a simulation application.

After creating the entities, all the simulation applications in the simulation exercise will send the Entity State PDU to the new simulation application.

4.2.8 Implementation Problems

To develop and use the network library using Java programming language have some problems as following:

1. **DIS is not complete and not for general use** – The specifications of DIS have not been fully released, so we can not get all the specifications. DIS itself is developed for the military use and seems can not support the general use of the network data exchange. If we want to obey the specifications of DIS to develop JavaNL seems have some problems.

2. **Java is not a real multithreaded programming language** – When the network applications using JavaNL have been run, there must be many threads running concurrently in the same time. Although Java is a multithreaded programming language, but it is not really multithreaded. If there is a thread does not release the thread control to the system, the system can not stop the tread and other threads will be starvation. So we make all the threads of JavaNL to go to sleep about 20 ms every time

when the threads complete their work. But, if the application itself has a non-stopping thread, the starvation situation must be occurred.

3. **JavaNL has the network secure problem** – Because Java is a networked programming language, security is an important concern. If an applet is loaded from a remote web server via a network protocol, it can not create a network connection to any computer other than the one from which it was itself loaded. Therefore, if an applet using JavaNL to be a network applet, it can not be run on Internet. So the network application using JavaNL can only run from the local file system.

4.3 Results

4.3.1 Performance Evaluation

To evaluate the performance of JavaNL, we have written two applications using JavaNL. One is put on the server side, and the other is put on the client site. When the server starts up, all it has to do is waiting for others to join. When there is one client program joining the server, it will send a PDU which contains two integer numbers in it to the client site. When the client program runs, it will join the server and ready to receive the PDU, after receiving the PDU, it will send the same PDU returned to the server, then we can calculate the performance of JavaNL. The size of the PDU is about **192** bytes, and is transmitted via UDP.

We put the two programs on the same workstation, SUN Solaris 2.5.1 on SUN Ultra-1 Model 170E with 10 base 2 Ethernet coaxial cable as shown in Table 3-2, and we measure the round trip time of a PDU sending from the server program. Because the PDU is sent via UDP, we also measure the round trip time of a UDP packet of the same size of the PDU in both a Java program and a C program. The results are summarized in the following table. The bottleneck of JavaNL is mainly in the polling of PDUs, in the concurrent control of Java threads, and in the pack of PDUs.

Round trip time of	PDU in JavaNL	UDP packet in Java	UDP packet in C
Time (ms)	**338**	**4**	**1**

Table 4-7 The performance of JavaNL.

We have also done the testing on the PC platform, Microsoft Windows 95 on Intel Pentium-200 with 10 base T Ethernet twisted pair line as shown in Table 3-3, the round trip time is **380 ms**. If the server is set on the workstation, and the client is on the PC as above, across the network, the round trip time is **356 ms**.

4.3.2 How to Use JavaNL

4.3.2.1 To Program With JavaNL

If programmers want to develop network applications using JavaNL, they can download the library of JavaNL from Ftp://ftp.cmlab.csie.ntu.edu.tw/pub/cml/JavaNL/javanl.zip, uncompress it and add the path of JavaNL to the environment variable **CLASSPATH**. Then, the programmers can use this library to develop the network applications. To use JavaNL in the Java applets, the programmers must import the classes of JavaNL first, declare the instances of the classes, then call the member functions of JavaNL. If a network application wants to send some information to other ones, the application must encapsulate all the information into the NLOBJ structure, then send the NLOBJ to other ones. There are two simple examples, one is for the simulation manager, and the other is for other simulation applications, in Figure 4-9 and Figure 4-10.

```
import java.lang.*;
import java.util.*;
import java.net.*;
import java.io.*;

// must import javanl.NL....
import javanl.NL;

class simpleServerApp extends Thread {
    static NL myNL = new NL ();
    private Thread MyThread;

    public void run () {
        while (true) {
            while (myNL.nlHaveMessage ()) {
                switch (myNL.nlGetMessage ()) {
                    case NL.NL_JOIN_REQUEST:
                        if (myNL.nlIsServer ()) { myNL.nlJoinAccept (); }
                        break;
                    case NL.NL_DISCONNECT:  break;
                }
            }
             try { MyThread.sleep (20); } catch (InterruptedException e) {}; }
    }

    public static void main (String argv[]) {
        myNL.nlInit ();
        myNL.nlOpenSession ();
        new simpleServerApp ();
    }

    public simpleServerApp () {
        MyThread = currentThread ();
        this.start   ();
    }
}
```

Figure 4-9 A simple examples for server site

```
import java.lang.*;
import java.util.*;
import java.net.*;
import java.io.*;

// must import javanl.NL....
import javanl.NL;

class simpleClientApp extends Thread {
    static NL myNL = new NL ();
    private Thread MyThread;

    public void run () {
        while (true) {
            while (myNL.nlHaveMessage ()) {
                switch (myNL.nlGetMessage ()) {
                    case NL.NL_JOIN_REQUEST:     break;
                    case NL.NL_JOIN_ACCEPT: break;
                    case NL.NL_JOIN_REJECT: break;
                    case NL.NL_DISCONNECT: break;
                }
            }
            try { MyThread.sleep (20); } catch (InterruptedException e) {}; }
    }

    public static void main (String argv[]) {
        myNL.nlInit ();
        myNL.nlJoinSession (new String ("cmlab"), 4321); }
        new simpleClientApp ();
    }

    public simpleClientApp () {
        MyThread = currentThread ();
        this.start  ();
    }
}
```

Figure 4-10 A simple examples for client site

The applets or applications using JavaNL must be threads or fork their own thread classes to poll JavaNL to get the messages from JavaNL or other applets or applications in the same simulation exercise (session). To make sure all the threads of JavaNL will be executed by the system, all the threads created by the applets or applications must go to sleep when they finished one job. When the server program wants to create a session, it will call the **nlOpenSession** after calling **nlInit** for initialized and the client program can call the **nlJoinSession** after calling **nlInit** to join the session, of course the client program must know where the server is.

4.3.2.2 Messages of JavaNL

There are seven messages which the JavaNL will send to the network applications as the following description:

1. **NL_JOIN_REQUEST**: Received when one client wants to join or has joined this exercise due to the receiver is server or client.
2. **NL_JOIN_ACCEPT**: Received when the server accepts the join request from me..
3. **NL_JOIN_REJECT**: Received when the server rejects the join request from me.
4. **NL_DISCONNECT**: Received when one application has left this exercise.
5. **NL_CREATE_OBJECT**: Received when the server tells me to create object.
6. **NL_REMOVE_OBJECT**: Received when someone tells me to remove an object.
7. **NL_OBJECT_STATUS**: Received when someone tells me the status of an object.

4.3.2.3 The NLOBJ Structure of JavaNL

The **NLOBJ** is a general data structure which is used to exchange the information between all the simulation applications in a simulation exercise. There are only four pointers in it, they are pointers for character, integer, floating point, and Boolean number arrays.

To use this data structure, the application must allocate the memory of the array size of these four primitive data types and put data in them when it wants to send the data. Then, call the **nlObjectStatus** to broadcast the changed information and other applications will receive the **NL_OBJECT_STATUS** message and a **NLOBJ** instance of the changed data.

4.3.2.4 Functions of JavaNL

Now, JavaNL only releases 12 functions to be the prototype of it. Here is some brief description of them.

1. **nlInit**: Call it when initialize JavaNL.
2. **nlIsServer**: If this application is a server, then returns true, otherwise returns false.
3. **nlHaveMessage**: If there comes a message.
4. **nlGetMessage**: Get the message.
5. **nlGetMessageSite**: Get the site of the message.
6. **nlGetMessageEntity**: Get the entity of the message.
7. **nlGetObject**: Get the object of the message.
8. **nlOpenSession**: Call it to open a session.
9. **nlJoinSession**: Call it to join a session.
10. **nlJoinAccept**: If the server accepts the join request from a client, calls it.
11. **nlRegisterObject**: Call it to register some objects.
12. **nlObjectStatus**: Call it if the application wants to broad cast the status of the objects.

Chapter 5

Multiparticipant
Building Walkthrough System
Using JavaGL and JavaNL

5.1 Introduction

After developing the two libraries, we want to combine them for more powerful applications. The first one is the prototype of the multiparticipant three dimensional (3D) graphics application interface. To build a multiparticipant interactive 3D graphics game, there are three important parts of it, they are the body of the game, the network agent, and the 3D graphics engine of it.

There are many multiparticipant games in the market and Internet, but many of them can be only run in the Local Area Network (LAN) environment or are only text mode not 3D graphics games. After Java existing on World Wide Web (WWW), word processing, spreadsheet, and database software may be all on Internet. The "pay-per-use" software model may be realized in the near future. We believe that after the applications people usually used are able to be used through Internet, the multiparticipant 3D graphics games will be the next target on Internet.

If a multiparticipant 3D graphics game is on Internet and "pay-per-use", people on any client sites of any kinds of platforms, even the platform is just a TV, Personal Digital Assistant (PDA) or something else, can play the game together. They can run the game on Internet by downloading the game at the play time, and create or join a session of the game as they want.

To build such a multiparticipant 3D graphics game on Internet, we need the two most important parts, the 3D graphics engine and network support, of it. Fortunately, we have JavaGL and JavaNL which are developed by pure Java programming language. Based on these two libraries, we can build a multiparticipant 3D graphics game as soon as possible.

To test the multiparticipant 3D graphics application interface, we follow the idea of the walkthrough system, which is an important topic of our laboratory. In our laboratory, we have developed a single user mode model-based building walkthrough system on an SGI workstation with several virtual reality (VR) devices, and a single user mode image-based walkthrough system on a personal computer (PC). So, we try to build a multiparticipant model-based building walkthrough system on Internet using Java programming language as usual, as an example of the multiparticipant 3D graphics application interface.

To give more interactions with all the participants in the building walkthrough system, we have also implemented a chat tool based on JavaNL. Using this chat tool, people in the same building environment can chat with each other just like they really walk in the building environment.

5.2 Implementation Issues

5.2.1 Building Model

This multiparticipant building walkthrough system uses a building model based on a simplified fifth floor of our department building. The file format of this model is the Triangle file as the descriptions in session 3.5.1.1. The building model contains **84** triangles in it. Because we want to develop an interactive multiparticipant building walkthrough system, the performance is the most important thing we must concern. The size of the building can not be too large to use, but if the model is too small it will not have the realness.

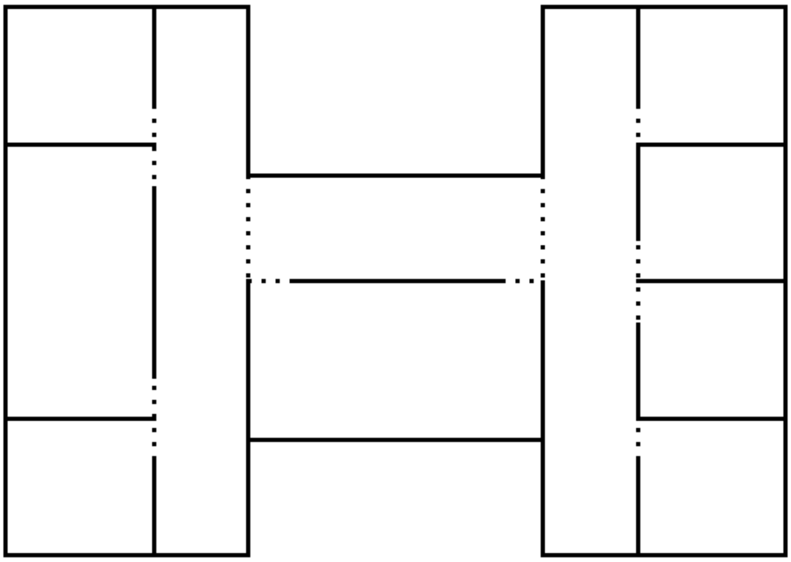


Figure 5-1 The bird's eye view of a building model.

Using the General Viewing System (GVS) in session 3.5.1 to show the building model, it takes **230 ms** in average, sometimes it only takes **130 ms** to show the model due to which parts of the model the user seen. So, to show the model in GVS, we can get **7.7 fps** (frames per second), it is not real-time but is acceptable.

5.2.2 System Hierarchy

Originally, we want to choose the building model of Virtual Reality Modeling Language (VRML) file format, but VRML itself is a general purpose 3D modeling language, includes the information of lights, transformations, cameras...etc. If we use the building model of VRML file format to be our building model, the performance of the building walkthrough system may be worse, because the system will pay more efforts to calculate so many parameters every time it renders the parsing tree of the building model and these parameters actually need to be calculated only once. The Triangle file format is also a general purpose modeling format, but it is simpler than VRML, because it only describes the geometric data of the 3D model.

To simplify the complexity of the building walkthrough system, we give all the polygons in the building model the same material parameters and give two fixed lights for lighting the building model. The participant in the building model can go ahead, back and turn right or left in the building, that means the participants have only two dimensional (2D) motion capabilities.

Because we choose the Triangle file format to be the file format of the building model, to parse and render this file format, we use the parsing and rendering parts of GVS, to simplify the development phase. Figure 5-2 is the system hierarchy of the multiparticipant building walkthrough system. In this figure, JavaGL and JavaNL act as the 3D graphics engine and network agent.

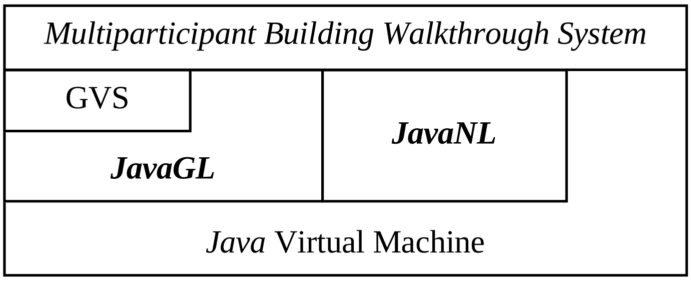


Figure 5-2 System hierarchy of the multiparticipant building walkthrough system.

Like the example code in Figure 4-9, if a participant wants to create a session of the multiparticipant building walkthrough system, the participant’s application will be the session server of this session. If he chooses the application to be a client site, the application will just join the session like the example code in Figure 4-10.

All the participants’ applications of the multiparticipant building walkthrough system must record all the statuses of other participants. When the status of itself is changed, it will send a NLOBJ to other participants. If the participants receive a NLOBJ from one participant, they will modify their own record table by themselves, and update the output screen if it is necessary.

If there is a new participant wants to join the session of the multiparticipant building walkthrough system and the server agrees, all the participants must add an entry in the record table for the new participant. When a participant received the disconnect message, maybe caused someone left or network malfunctioned, it just removes an entry in the record table and the application of this participant can still be run as usual.

The most important feature of multiparticipant applications is interaction with other participants in the same environment. To do this, we have also implemented a chat tool with the multiparticipant building walkthrough system. The chat tool is developed with JavaNL only, using JavaNL to develop such a network application only takes 10 or 20 minutes.

5.3 Results

Because the multiparticipant building walkthrough system is developed using Java programming language, the 3D graphics rendering work is done by JavaGL and the network transmission work is done by JavaNL, this system can run on any Java enabled platforms.

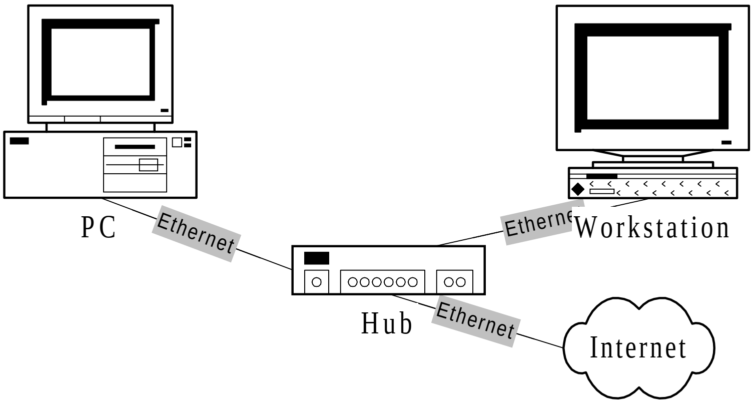


Figure 5-3 The testing environment of the multiparticipant building walkthrough system.

The system can be run on different machines or the same machine with different ports and communicated with other systems through the network. Actually, we have tried to put more than ten people in the same building environment, some of the people are in different machines, some are in the same one with different ports. They can see other participants as symbols in the building environment and chat with them.

To evaluate the performance of the multiparticipant building walkthrough system, we put the system on a workstation and a PC. The performance evaluation is as the following table.

Platform	**Workstation**	**PC**
Refresh Time (ms)	**230**	**130**
Refresh Rate (fps)	**4.3**	**7.7**
Environment	SUN Ultra-1 Model 170E 128 MB memory 24-bit display (Creator 3D) SUN Solaris 2.5.1 10 Base 2 Ethernet	Intel Pentium-200 64 MB memory 24-bit display (ET 6000) Microsoft Windows 95 10 Base T Ethernet
Interpreter	SUN JDK 1.0.2 SUN JIT 1.0.2	Symantec Café 1.51 Symantec JIT 2.0 beta3

Table 5-1 Performance of a multiparticipant building walkthrough demo system.

If the participant at the PC site changes his position, the participant at the workstation side will be notified after about **250 ms**. The delay is due to the refresh rate and the network latency.

Plate 5 shows the result of running the multiparticipant building walkthrough system. In this simulation exercise, there are three participants in it, and Plate 5 shows one participant's view. The other two participants are represented by red cubes.

With the chat tool in the system, users can chat with each other in the building environment. Although the transmission rate is not very well, but it is accessible for the chat tool.

5.4 Conclusions

Although the performance of the multiparticipant building walkthrough system is not impressive, this application demonstrates as an example of a multiparticipant 3D graphics application interface on Internet.

From the research experience of developing a model-based building walkthrough system in our laboratory [54], the performance will be sped up about **6.6 times** by applying all the dynamic visibility computation. Then, the frame rate of the multiparticipant building walkthrough system will be up to **50 fps** or the size of the building model will be larger to make users feel more realistic.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

This thesis presents two useful libraries. One is JavaGL, a three dimensional (3D) graphics library with the application programming interface (API) witch is similar to that of OpenGL. The other is JavaNL, a network library. Both of them are developed by pure Java programming language and are platform independent. By using the two libraries, the thesis also presents a multiparticipant 3D graphics application interface on the Internet by using the Java programming language, and gives a multiparticipant building walkthrough system as an example of this interface.

POPULARITY: Since we put JavaGL on our web pages three months ago, there are more than **1000** people all over the world already visited this page, and sent us dozens of e-mails concerning the use JavaGL. Some of them offer to collaborate and some want to use JavaGL, and this encourages us to improve both JavaGL and JavaNL.

Following the development of multiparticipant 3D graphics applications on Internet, we believe that these applications will be the next “pay-per-use” stream on the Internet. In this thesis, we offer JavaGL and JavaNL for these applications.

Although the performance of the multiparticipant building walkthrough system is not so well, but by the experience of developing JavaGL, the performance may be enhanced by the improvement of the computer hardware or the Java technology. Maybe we can design a sexy multiparticipant 3D graphics game on the Internet by using Java programming language when the performance of the Java environment is acceptable, and the performance of JavaGL can be boosted up to render some complex environments or objects in real-time.

Using JavaNL to develop a network utility is easier than before, just like our multiparticipant building walkthrough system, we add the chat capability in it after the system have been established. To add the chat capability, we just take less than ten minutes to finish this work by using JavaNL.

6.2 Future Work

6.2.1 JavaGL

JavaGL now has not fully supported all the OpenGL functions, only the daily used ones. Although JavaGL has not accomplished, people have already been able to use this 3D graphics library for 3D graphics applications on Internet. By the counter on the web site, the log file on the FTP site and some mails from several organizations, we have found that there are several people and organizations have noticed and used JavaGL, such as SUN, IBM, SAS, HP, Intel, S3, NASA, CERN, Purdue, CalTech, GaTech, Texas, Columbia, Ohio, Tokyo, Aizu, Taiwan, universities. To enhance JavaGL to support all the OpenGL functions seems to be a necessary goal.

Rendering performance is always a great challenge for Java and software based 3D graphics library. Since the development of JavaGL, many researchers and organizations are facing this problem. So, besides the performance enhancement of JavaGL, we expect that the performance will be improved by faster Java interpreters and Java compilers from the software side, and will be greatly improved by the new Java chips from the hardware side.

One of the characteristic of OpenGL is that OpenGL is a platform independent 3D graphics library, no mater which machine or display card a programmer uses, he can call the same OpenGL functions to do the 3D graphics work. But, OpenGL itself is designed for C programming language, it offers several redundant functions to make programmers to call the function by different parameters, such as glColor3f, glColor3d...etc., all of the glColor*NTV* (where *N* means the number of parameters, *T* means the data type of the parameters and *V* means if the parameter is an array) are the same function to set the foreground color of the GL. JavaGL is developed by Java programming language which is one of the object-oriented programming (OOP) language. Polymorphism is one of the characteristic of the OOP language and so is Java programming language. We will consider to use this characteristic to simplify these redundant function names because gl*FunctionNTV* is not necessary for JavaGL.

6.2.2 JavaNL

JavaNL is not yet complete by now. Currently, it can not offer all the network functions to the programmers, however, the non-military functions are ready. The goal of JavaNL is to present a prototype for constructing network applications as easy as possible, and offer programmers a reliable and useful library to build network applications.

The API and the transmitted protocol data unit (PDU) of JavaNL are defined by ourselves. To make JavaNL useful to more programmers, we must pay more efforts to make the API and the PDU of JavaNL to obey some standards or rules of some products, such as Microsoft DirectPlay [55], Distributed Interactive Simulation (DIS) or Department of Defense (DOD) High Level Architecture (HLA) [56].

Color Plate

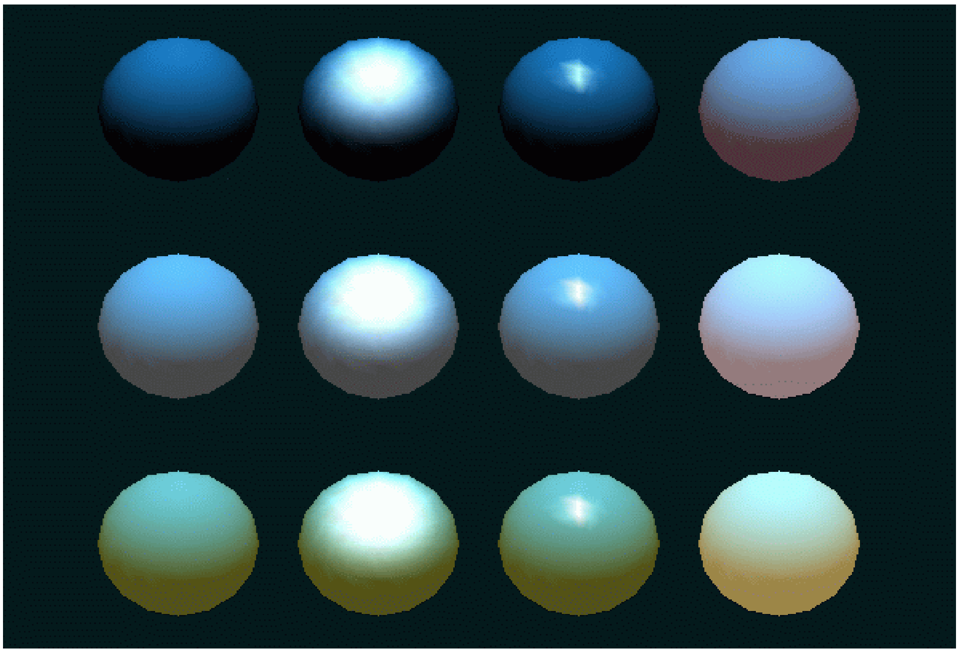
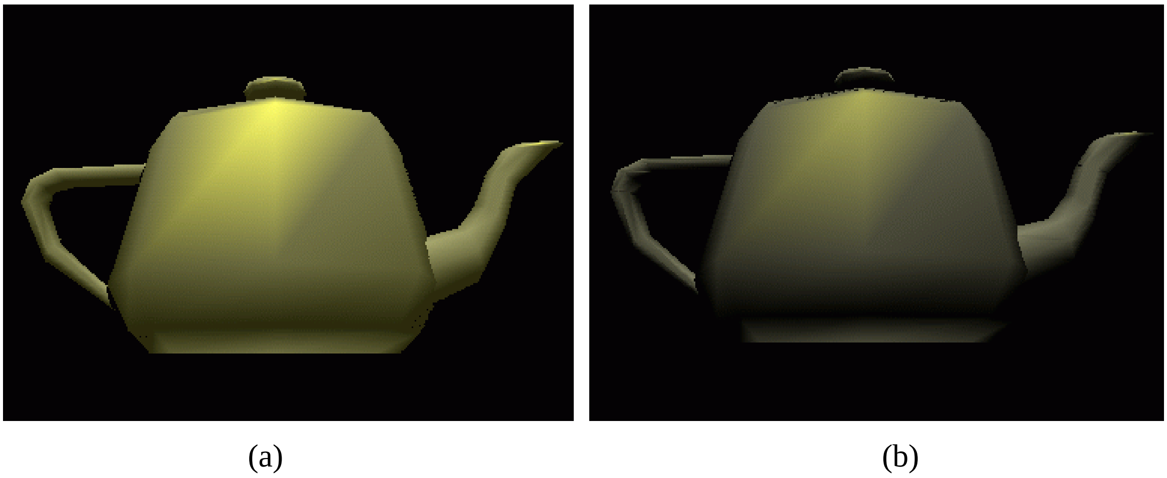
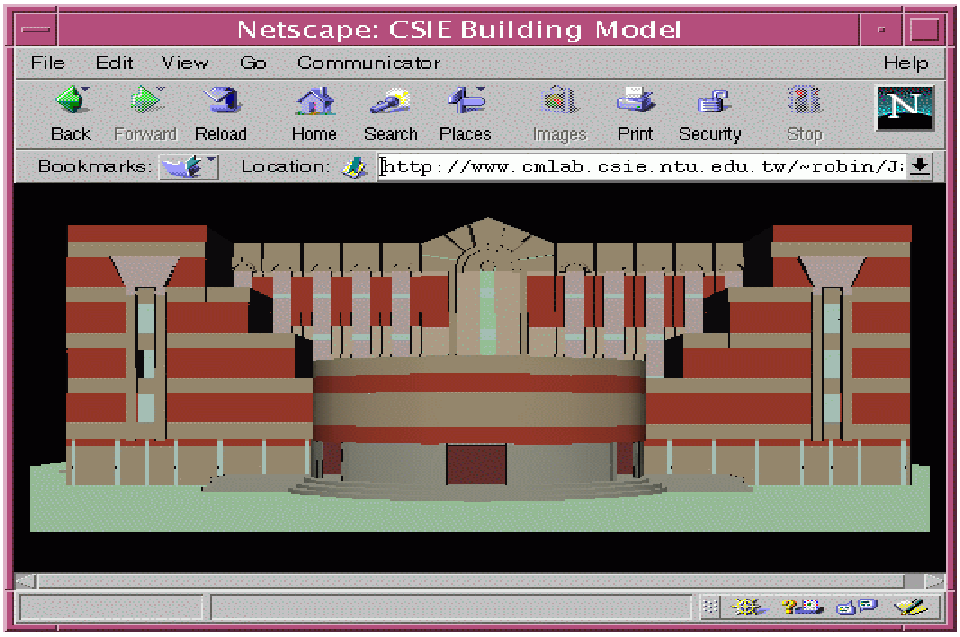


Plate 1 Twelve spheres are rendered to measure performance. Each sphere contains **256** polygons and has different materials. This program is an example in *OpenGL Programming Guide* (code from Listing 6-3, pp. 183-184, Plate 16). This photo is rendered with JavaGL.



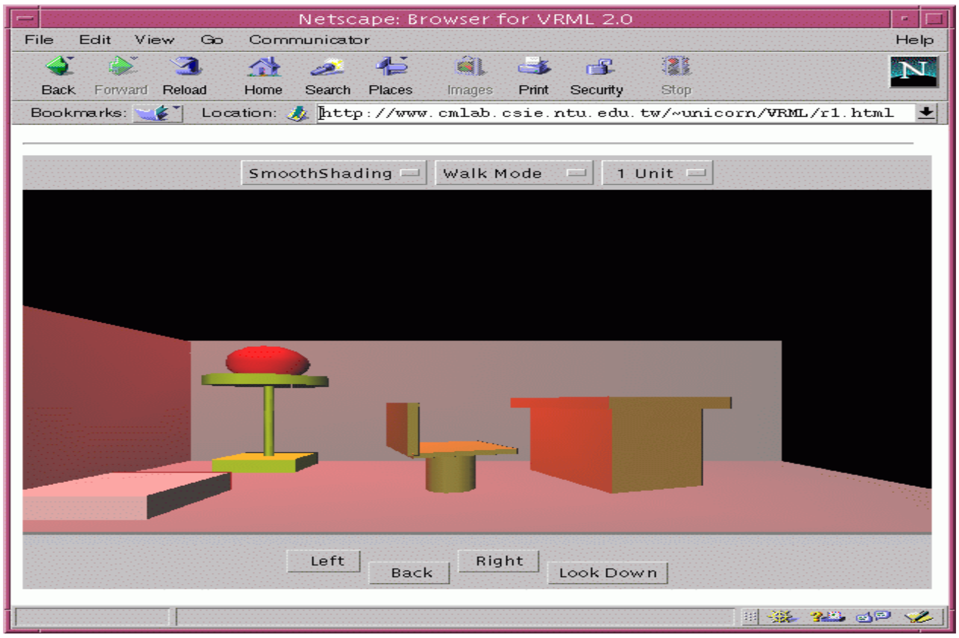
(a) (b)

Plate 2 The teapot rendered (a) with JavaGL (**550 ms**), and (b) with Mesa (**122 ms**).


Netscape: CSIE Building Model
File Edit View Go Communicator Help
Back Forward Reload Home Search Places Images Print Security Stop
Bookmarks:
Location:

http://www.cmlab.csie.ntu.edu.tw/~robin/J:

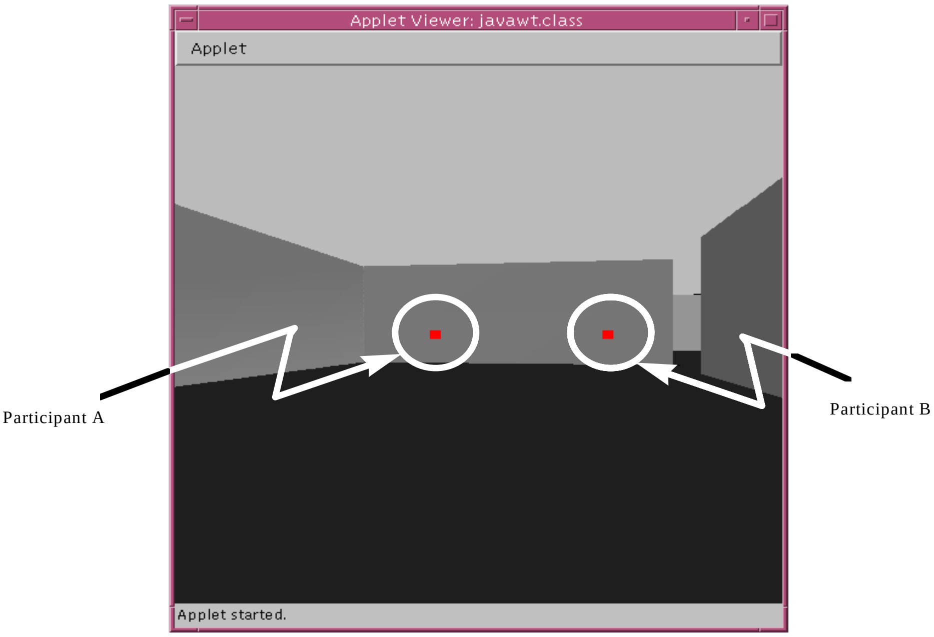
Plate 3 A complicated example rendered by GVS on Netscape Communicator 4.0pr2. This is the model of our department building which contains **5273** triangles and takes **6150 ms** on a PC as shown in Table 3-3.


Netscape: Browser for VRML 2.0
File Edit View Go Communicator Help
Back Forward Reload Home Search Places Images Print Security Stop
Bookmarks:
Location:
Left
Back
Right
Look Down

http://www.cmlab.csie.ntu.edu.tw/~unicorn/VRML/r1.html

SmoothShading Walk Mode 1 Unit

Plate 4 The room rendered by JavaVRML on Netscape Communicator 4.0pr2



Applet started.

Plate 5 The photo of a multiparticipant building walkthrough system.

Bibliography

[1] Ed Krol, "*The Whole Internet & User's Guide*," O'Reilly & Associates, Inc., 1992.

[2] Fred Halsall, "*Data Communications, Computer Networks and Open Systems*," Addison-Wesley, 1992.

[3] "*W3C - The World Wide Web Consortium*," World Wide Web Consortium, 1997, Http://www.w3c.org/pub/WWW.

[4] "*Netscape Communicator*," Netscape Communications, Co., 1997. Http://home.netscape.com/comprod/products/communicator.

[5] "*Microsoft Internet Explorer*," Microsoft, Co., 1997. Http://www.microsoft.com/ie/default.asp.

[6] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, "*Computer Graphics Principles and Practice*," Addison-Wesley, 1990.

[7] R. A. Earnshaw, M. A. Gigante, H. Jones, "*Virtual Reality Systems*," Academic Press, Inc., 1993.

[8] "*VRML.SGI.COM*," Silicon Graphics, Inc., 1997, Http://vrml.sgi.com.

[9] "*Apple QuickTime VR*," Apple Computer, Inc., 1997. Http://qtvr.quicktime.apple.com.

[10] "*Java Programming Language*," Sun Microsystems, Inc., 1997. Http://www.javasoft.com.

[11] "*OpenGL WWW Center*," Silicon Graphics, Inc., 1997. Http://www.sgi.com/Technology/openGL.

[12] "*RenderWare*," Criterion Software Ltd., 1996. Http://www.canon.co.uk/csl/

rwmain.htm.

[13] Robin Bing-Yu Chen, "*JavaGL - A 3D Graphics Library in Java,*" 1997. Http://www.cmlab.csie.ntu.edu.tw/~robin/JavaGL.

[14] Mark Segal, and Kurt Akeley, "*The OpenGL Graphics Systems: A Specification (Version 1.1),*" Silicon Graphics, Inc., 1996. Http://www.sgi.com/Technology/openGL/glspec/glspec.html.

[15] Robin Bing-Yu Chen, "*JavaNL - A Network Library in Java,*" 1997. Http://www. cmlab.csie.ntu.edu.tw/~robin/JavaNL.

[16] "*Distributed Interactive Simulation,*" Institute for Simulation and Training, University of Central Florida, 1997. Http://www.ist.ucf.edu/labsproj/projects/dis.htm.

[17] Bing-Yu Chen, Chee-Wen Shiah, Tzong-Jer Yang, and Ming Ouhyoung, "*JavaGL - a 3D Graphics Library in Java for Internet Browser,*" to appear in Proc. of IEEE International Conference on Consumer Electronics, Chicago, Illinois, USA, 1997.

[18] "*IEEE Standard for Distributed Interactive Simulation – Application Protocols (IEEE Std 1278.1-1995),*" Institute of Electrical and Electronics Engineers, 1996.

[19] "*HTTP - Hypertext Transfer Protocol,*" World Wide Web Consortium, 1997, Http://www.w3.org/pub/WWW/Protocols.

[20] "*HyperText Markup Language (HTML),*" World Wide Web Consortium, 1997, Http://www.w3.org/pub/WWW/MarkUp.

[21] "*NCSA Mosaic,*" National Center for Supercomputing Applications, University of Illinois at Urbana/Champaign, 1997 Http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html

[22] Gavin Bell, Anthony Parisi, and Mark Pesce, "*The Virtual Reality Modeling Language: Version 1.0C Specification,*" 1996. Http://vag.vrml.org/vrml1.0c.html

[23] "*The Virtual Reality Modeling Language Specification Version 2.0*," Silicon Graphics, Inc., 1996. Http://vrml.sgi.com/moving-worlds.

[24] "*SSB VRML Group*," Communications and Multimedia Lab., National Taiwan University, 1996, Http://www.cmlab.csie.ntu.edu.tw/~ssb/gvrml.

[25] Jen-Wei Chen, "*JavaVRML - A Browser for VRML 2.0*," 1997. Http://www.cmlab. csie.ntu.edu.tw/~unicorn/VRML/borwser.html

[26] Shenchang Eric Chen, "*QuickTime VR - An Image-Based Approach to Virtual Environment Navigation*," SIG GRAPH '95, p.29 - p.38, 1995

[27] "*Photo VR*," Communications and Multimedia Lab., National Taiwan University, 1997. Http://www.cmlab.csie.ntu.edu.tw/cml/g/pvr/PVR.html.

[28] Wen-Kae Tsao, Bing-Yu Chen, Jiunn-Jia Su, and Ming Ouhyoung, "*Rendering Real World Scenes and Objects for Virtual Environment Navigation*," in Proc. of International Computer Symposium, p.1 - p.8, NSYSU, Kaohsung, ROC, 1996.

[29] Wen-Kae Tsao, "*Rendering Scenes In the Real World for Virtual Environments Using Scanned Images*," Master Thesis, Dept. of Computer Science and Information Engineering, National Taiwan University, 1996.

[30] David Flanagan, "*Java in a Nutshell*," O'Reilly & Associates, Inc., 1996.

[31] "*Java 2D API*," Sun Microsystems, Inc., 1997. Http://www.javasoft.com/ products/java-media/2d.

[32] *Java 3D*," Sun Microsystems, Inc., 1997. Http://www.javasoft.com/products/ api-overview.html#3d.

[33] "*The Java Developers Kit Version 1.0.2*," Sun Microsystems, Inc., 1996. Http:// www.javasoft.com/products/jdk/1.0.2.

[34] "*Liquid Reality*," Dimension X, Inc., 1997. Http://www.dimensionx.com/ products/lr.

[35] "*Cosmo 3D*," Silicon Graphics, Inc., 1997. Http://www.sgi.com/Products/

cosmo/cosmo3D.

[36] Jeff Orkin, “*Java3D,*” Demon Systems, Inc., 1997. Http://www.demonsys.com/jorkin/java3d.

[37] Mark J. Kilgard, “*OpenGL Programming for the X Window System,*” Addison-Wesley, 1996.

[38] Ron Fosner, “*OpenGL Programming for Windows 95 and Windows NT,*” Addison-Wesley, 1996.

[39] Jackie Neider, Tom Davis, and Mason Woo, “*OpenGL Programming Guide,*” Addison-Wesley, 1993.

[40] Andrew S. Glassner, “*Graphics Gems,*” Academic Press, Inc., 1990.

[41] James Arvo, “*Graphics Gems II,*” Academic Press, Inc., 1991.

[42] David Kirk, “*Graphics Gems III,*” Academic Press, Inc., 1992.

[43] “*Cosmo Code,*” Silicon Graphics, Inc., 1997, Http://www.sgi.com/Products/cosmo/code.

[44] “Symantec *Café,*” Symantec, Co., 1997. Http://www.symantec.com/cafe.

[45] “*Microsoft Visual J++,*” Microsoft Co., 1997, Http://www.microsoft.com/visualj.

[46] Mark J. Kilgard, “*Graphics Library Utility Toolkit,*” Silicon Graphics, Inc., 1996. Http://www.sgi.com/Technology/openGL/glut.html.

[47] Brian Paul, “*The Mesa 3-D Graphics Library,*” 1997. Http://www.ssec.wisc.edu/~brianp/Mesa.html.

[48] Arthur van Hoff, Sami Shaio, and Orca Starbuck, “*Hooked on Java,*” Addison-Wesley, 1996.

[49] “*Solaris OpenGL Ultra Creator3D Edition,*” Sun Microsystems, Inc., 1997. Http:// www.sun.com/solaris/graphics/openglOverview.html

[50] “*The JIT Compiler Interface Specification,*” Sun Microsystems, Inc., 1996. Http://www.javasoft.com/doc/jit_interface.html.

[51] "*OpenGL for Windows 95*," Microsoft, Co., 1997. Http://www.microsoft.com/kb/softlib/mslfiles/Opengl95.exe

[52] "*IEEE Standard for Distributed Interactive Simulation – Communication Services and Profiles (IEEE Std 1278.2-1995)*," Institute of Electrical and Electronics Engineers, 1996.

[53] "*Enumeration and Bit-encoded Values for Use with IEEE Std 1278.1-1995, Standard for Distributed Interactive Simulation – Application Protocols*," Institute for Simulation and Training, University of Central Florida, 1996. Http://ftp.sc.ist.ucf.edu/SISO/dis/library/enumerat.doc.

[54] Yuong-Wei Lei, "*The SpaceWalker Walkthrough System for Unrestricted Three-Dimensional Polygon Environments*," Ph.D. Thesis, Dept. of Computer Science and Information Engineering, National Taiwan University, 1996.

[55] "*Microsoft DirectPlay*," Microsoft, Co., 1997. Http://www.microsoft.com/devonly/tech/dx3doc/dire0340.htm

[56] "*High Level Architecture*," Defense Modeling and Simulation Office, Department of Defense, 1997. Http://www.dmso.mil/projects/hla.

[57] Chung-Hsi Huang, "*The Implementation of Distributed Interactive Simulation and Its Applications in Virtual Reality*," Master Thesis, Dept. of Computer Science and Information Engineering, National Taiwan University, 1996.