jGL and its Applications as a Web3D Platform

Bing-Yu Chen Tomoyuki Nishita
robin@is.s.u-tokyo.ac.jp nis@is.s.u-tokyo.ac.jp
Department of Information Science, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan

ABSTRACT

This paper proposes a new platform for 3D graphics on the Internet (and recently also for Web3D). To develop 3D graphics programs on the Internet is not very easy, because there are no high-quality tools like OpenGL. For this purpose, we have developed a 3D graphics library, called jGL, by using pure Java since the end of 1996. At that time, we ignored some functions, such as texture mapping, because these functions were too complex to be realized on low-cost machines, even on the fastest machine three years ago. jGL is a general-purpose 3D graphics library, and its application-programming interface is defined in a manner quite similar to that of OpenGL. Today, the hardware is better, but the network bottleneck is still the same as before, so we almost re-wrote all the code to enhance its capabilities and performance and minimized its code size to make it more suitable for running on the Internet.

Moreover, VRML is a standard 3D graphics file format and very popular. To display or play a 3D model or scene on the Internet, people would like to use the VRML file format, and use the VRML browser plug-in of the Internet browsers to view it. Unfortunately, besides the Microsoft Windows and SGI workstation environments, the support for displaying VRML models is not enough. Hence, we also developed a real platform independent VRML browser applet by using jGL, that can be used on any Java enabled Internet browsers. It is also a good example for jGL.

CR Categories: C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks – Internet; D.2.2 [Software Engineering]: Design Tools and Techniques – Software libraries; I.3.2 [Computer Graphics]: Graphics Systems – Distributed/network graphics; I.3.4 [Computer Graphics]: Graphics Utilities – Graphics packages, Software support

Additional Keywords: OpenGL, VRML, Web3D, Java.

1. Introduction

Since the end of 20th century, the Internet is getting more and more popular. More and more people use the services on the Internet everyday, like WWW and E-Mail. To produce more realistic and interactive contents and enhance the abilities of the Internet, people have rediscovered 3D graphics recently. Although there are many 3D graphics applications that could be used on the Internet, there is only a small number available today. The main problem is that they must offer several versions for different platforms, since the Internet itself is a heterogeneous network environment. Observing the development of the Internet, we believe that “pay-per-use” software will be realized in the near future. Under this new paradigm, we may need to distribute applications from servers to clients on different platforms. Therefore, we decided to develop a 3D graphics library that is platform independent. Java [1] is chosen as our programming language for its hardware-neutral features, and wide availability on many hardware platforms, even for embedded systems, such as mobile phones or PDAs (personal data assistant).

To develop 3D graphics applications on a stand-alone computer, programmers always hope to use a powerful 3D graphics library, like OpenGL [2]. But, there is no library to develop such applications on the Internet by using Java. Moreover, such a 3D graphics library should be easy to learn and use. Therefore, we defined an API (application-programming interface) called jGL in a manner quite similar to that of OpenGL, since OpenGL is an industry standard, and many programmers are familiar with OpenGL's API.

Unlike similar mostly commercial products; jGL does not need any native codes or pre-installed libraries, it is developed with pure Java only. Users who run any programs developed with it do not need to install any package before using them, all required code will be downloaded at run time.

We released the first version of jGL at the end of 1997 [3]. That version provided only the basic rendering routines, but lacked some complex functions, such as texture mapping. That is because those functions are time consuming and could not be realized on a low-end machine, even on the best machines at that time. But hardware has much improved since, and more and more fancy applications have been developed for the Internet. Hence, we decided to enhance the capabilities of jGL. Unfortunately, although the hardware is getting faster and faster day-by-day, the network bandwidth is still the same as, or even worse than before. Trying to increase the capabilities of jGL may make its code too large to be transmitted over the Internet. To enhance the capabilities and minimize the code size at the same time, we decided to re-write the kernel.

Besides the 3D graphics library as a tool for programmers, we also need a 3D model and scene browser. Hence, we also developed a VRML browser applet by using jGL, since people would like to use the VRML file format [4] to display or play a 3D model or scene on the Internet. Furthermore, since we use jGL for the 3D graphics rendering, the VRML browser applet is also a good example for proving the capability of jGL.

Before the end of 2000, the jGL project has been known as JavaGL, since Java and all Java based marks are registered trademarks of Sun Microsystems, Inc., we have changed the name of JavaGL to be jGL from now on.

2. Related Projects

For the Web3D platform, there are some related projects on the Internet. One is a 3D graphics library; others are for displaying 3D objects or scenes.

Java3D – Java3D [5] is provided by Sun Microsystems Inc., but has not become a part of core Java package. Java3D needs support of OpenGL or DirectX, which has been pre-installed. To use a program based on Java3D, the users have to install the run-time package before using it. Moreover, Java3D has its own API, so people who want to write some 3D graphics programs by using it may spend much time to learn how to use it.

Shout3D – Shout3D [6] is a commercial product of Shout Interactive Inc. Shout3D uses pure Java and can display 3D models on the Internet. Although the file format of the 3D models is not VRML, it provides a converter from VRML to its own file format, but there is no one-to-one mapping. Like VRML, Shout3D also provides its own API to let people program animations of the 3D models.

blaxxun3D – blaxxun3D [7] is a commercial product of Blaxxun Interactive Inc. As Shout3D, it was also developed using pure Java and can display 3D models on the Internet. blaxxun3D reads VRML and draft X3D (Extensible 3D) file format, although it does not fully support both. Following the concepts of JavaEAI (Java External Authoring Interface), blaxxun3D also provides an API to let people program their 3D models.

Cortona Jet – Cortona Jet [8] is a commercial product of Parallel Graphics Inc. As Shout3D and blaxxun3D, it was also developed using pure Java and can display 3D models on the Internet. Cortona Jet reads VRML file format, but does not provide any API for programming.

Xj3D – Xj3D [9] is an open-source project of the Web3D Consortium. Xj3D is also developed using Java, but the 3D graphics rendering is based on Java3D. It can read VRML and draft X3D file format exactly; actually it is the testing platform for the X3D file format, the next generation of VRML.

Since Java3D is not platform independent and programmers need to pay more time to study how to use it, we think to provide a real platform independent 3D graphics library with familiar API is useful. Moreover, although we can use Shout3D or blaxxun3D to show some 3D objects, but we still need a browser to support more interactive mechanisms and to be programmable by ourselves. This is facilitated if the browser is implemented on top of a powerful API.

3. jGL - A 3D Graphics Library in Java

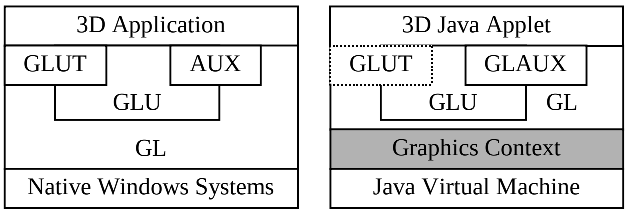
jGL is a 3D graphics library for Java. Unlike Java3D or other libraries, jGL does not need to be pre-installed; all the necessary codes will be downloaded at run time. jGL is written in pure Java, so it is really platform independent, and could be executed on any Java enabled machine. Since OpenGL is so famous and known by many 3D programmers, we followed the specifications of it to develop jGL. Therefore programmers who want to use jGL to develop their own 3D programs on the Internet do not need to learn how to use the new library, they can find a one-to-one mapping functions in jGL and those in OpenGL.

We began to develop jGL since the end of 1996, and released several versions in 1997. At that time, the performance was just 4 times slower than the Mesa-3D, which was developed by using the C language. Although the performance is not too bad, we still skipped some issues, which required heavy calculations, such as texture mapping. We began to re-develop jGL since summer 1999, almost all program code has been re-written. The class inheritance tree structures of jGL have also been re-designed again.

As the development experience of jGL, we find that the performance is not the most important problem, but the code size is. Within the four years, the performance of computer hardware improved several times, but network bandwidth is still the same. For example, four years ago, we used a PC with Intel Pentium 200MHz as our best testing platform and a 100BaseT as our Ethernet line, and 57600bps modem for dialing-up accesses. Now, we are using a PC with Intel PentiumIII 1.13GHz as our best testing platform, but the Ethernet line and modem are the same. Hence, we tried to develop jGL with the minimum code size.

3.1 OpenGL vs. jGL

Functions of OpenGL can be divided into two main categories: OpenGL Utility Library (GLU) and OpenGL (GL) as shown in Figure 1(a). jGL follows the same function hierarchy as shown in Figure 1(b) [10].



(a) (b)

Figure 1 The hierarchy of (a) OpenGL and (b) jGL modules.

GL implements a powerful but small set of drawing primitive 3D graphics operations, including rasterization, clipping, etc. GLU provides higher-level OpenGL commands to programmers by encapsulating these OpenGL commands with a series of GL functions. Besides these two main interfaces, there is an OpenGL Programming Guide Auxiliary Library, called AUX or GLAUX, which is not an official part of OpenGL API, but is widely used and familiar to many programmers. For this reason, we also include GLAUX in our package. Recently, GLUT has been widely used by programmers also, but we have not implemented this part yet.

The implementation of jGL is mainly based on the specifications of OpenGL, while the GLAUX library is implemented according to the *OpenGL Programming Guide*. Besides GL, GLU, and GLAUX, which are just interfaces for programmers, there is an underlying graphics context, which is transparent to programmers, and people cannot use this part directly. Hence, to enhance the performance and minimize the code size, we re-wrote all the code in the graphics context several times.

3.2 Implementation of Graphics Context

To reduce the code size and keep or enhance the performance of jGL, we utilize the class inheritance characteristics to re-design the system hierarchy of graphics context as Figure 2.

The graphics context of jGL can be divided into two parts. One part is for display lists; all the commands from the GL will not be executed and just stored as a sequence of rendering commands. The other part is for real graphics contexts, all the commands from the GL will be executed immediately.

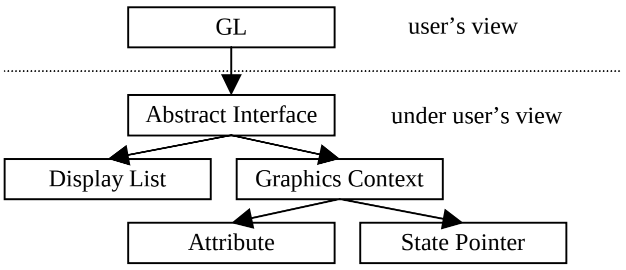


Figure 2 The graphics context of jGL.

The commands that will be executed in the real graphics context also can be categorized into two types. One type is just for changing some information stored in the graphics context. For example, when the user calls the glClearColor command, the color value to clear the display will just be stored; no real clear actions will be performed. Once the user calls the glClear command (with GL_COLOR_BUFFER_BIT), the real clear action will be executed by using the clear color value, which has been stored before.

The other type is like the previous glClear command. The commands in this type will perform some actions and directly produce some changes. Since jGL is defined as a state machine as OpenGL, we utilize class inheritance to avoid frequent checks here. The states of jGL have been classified into several states; including selection or not, flat or smooth shading, with depth test or without, texture mapping enabled or disabled, clipping for clip-plane or not. Therefore, running in different states will need different clipping, geometry and rendering routines.

3.3 Performance Enhancement Issues

Performance is a great challenge for both 3D graphics and Java, hence it is also a great challenge for jGL. Moreover, jGL is designed to operate over the Internet, where network bandwidth affects the overall performance significantly. Since we want to enhance the capabilities and minimize the code size without making the run-time performance worse, these considerations make the implementation of jGL complex.

Based on our experiences, we developed the following policies to speed up the performance and minimize the code size of jGL.

Utilize class inheritance to avoid "if-then-else" statements

OpenGL is a state machine, and it is usually necessary to determine if some status is enabled or not, which takes time to check. Hence, we utilize class inheritance to avoid these frequent checks, so that the status check will be realized only when the status changed. When the status changed, the rendering class pointer will be changed to point to its proper class type, so all the rendering commands followed will be routed to proper functions automatically without any further checks.

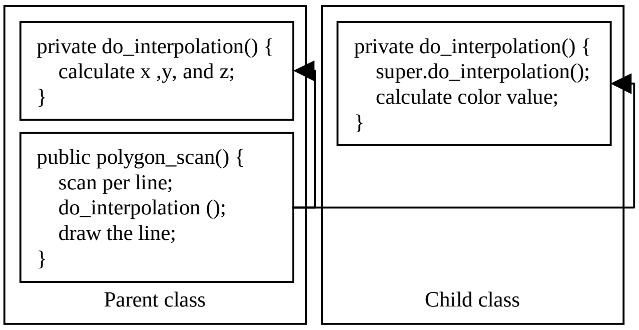
Divide a routine into several smaller ones

If a routine was very large and would be called in several situations, this routine must have some useless code segments for some states, while it is just called for a simple situation. So, it is worthwhile to divide this routine into several smaller ones.

For example, to fill a polygon, we must do color interpolation if the polygon is filled using smooth shading. However if the polygon only requires constant shading, color interpolation would be unnecessary. Therefore, we divide the shading routines into two smaller ones.

Use function overriding to minimize code size

Once we divide some routines into several smaller ones, the total code size will be larger than before, because there are too many duplicated codes. Although we have utilized class inheritance to avoid "if-then-else" statements, here we also use this method to structure all the small routines and then use function overriding to reduce the duplicated codes.


Parent class:

```
private do_interpolation() {
    calculate x ,y, and z;
}

public polygon_scan() {
    scan per line;
    do_interpolation ();
    draw the line;
}
```

Child class:

```
private do_interpolation() {
    super.do_interpolation();
    calculate color value;
}
```

Figure 3 The example of using function overriding.

The same example as the one above, since we have divided the shading routines into two smaller ones, one for flat shading and the other for smooth shading. Because all of them need the polygon scan procedure to fill the polygon, we can put the two routines in the parent and child classes, and use the same code base to do the polygon scan. The difference between the two small routines is just for interpolation. In the flat shading, we only need to interpolate the point positions in the polygon, but in the smooth shading, we also need to calculate the color interpolation. Therefore, the function-overriding example is like Figure 3.

4. VRML Browser Applet by Using jGL

To test the capabilities of jGL, we also developed a VRML browser applet by using jGL, since VRML is also a standard for modeling 3D models and scenes on the Internet, and used by many people. Moreover, to provide a solution for Web3D, we will not only need to deliver the jGL, but also a testing platform, such as this VRML browser applet.

To provide such a browser applet by using pure Java is not very easy, but fortunately we have jGL to be our 3D graphics engine, and programming with jGL is almost the same as with OpenGL. Since VRML itself is object oriented, following the development policies of jGL to design the VRML browser applet by using Java is not too difficult, but performance and code size are still huge problems for it.

4.1 System Hierarchy

We followed the specification of VRML to develop the VRML browser applet. There are two main parts, which are nodes and fields, in the VRML specification. The 3D models or scenes are associated with several nodes with a tree structure, and the parameters of the nodes are stored in some fields.

Since VRML is object oriented, we also want to use the same performance enhancement methods as jGL. Therefore, we use class inheritance to organize fields in a class tree. For nodes, since there are several differences between nodes, we only classify all nodes into 6 main categories, and 4 sub-categories, and also organize nodes in a class tree.

The system hierarchy of the VRML browser applet is depicted in Figure 4, and the Node and Field are the two main parts. We isolate the rendering routines (the Render Interface in the Figure) as a stand-alone class located between the VRML browser applet and jGL, hence we can enhance the rendering performance by optimizing this one class.

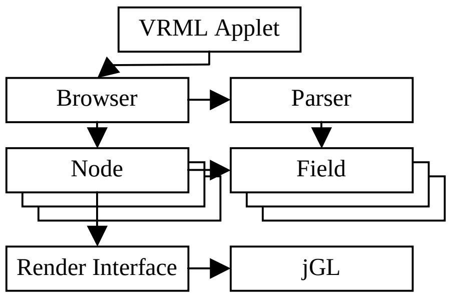


Figure 4 The system hierarchy of VRML browser applet.

The Browser and Parser as its class name are for all browser and parser functions. The Parser will be called only when loading the VRML file, and is used to parse the VRML file format and store all the information into Fields of Nodes.

4.2 Performance Enhancement Issues

From the experiences of developing jGL, we also utilize class inheritance and function overriding to minimize the code size and enhance the performance. Besides these, we also use the following policies.

Use display list mechanism of jGL

In OpenGL, display lists are used to store rendering commands, so that programmers could pre-calculate necessary functions and store the results and rendering commands into display lists to enhance the rendering performance. Because jGL has the same mechanism as OpenGL, we utilize display list of jGL in the same way.

Pre-process the constant parameters

Since almost all the attributes of nodes will not be changed while re-drawing, we pre-process or pre-calculate all static information and store it in the nodes to enhance the performance.

Combine arbitrary geometric mesh data

Arbitrary geometric mesh data is more important than other nodes, since there are only few primitive geometric nodes supported by VRML. Most people prefer to use meshes instead of well-defined primitive geometric nodes. Therefore, it is important to display an arbitrary geometric mesh data efficiently.

In VRML, to show arbitrary geometric mesh data with different materials, we must use several Shape nodes with IndexedFaceSet nodes. Hence, there will be a huge branch in the 3D scene tree. Therefore, we combine such nodes to be just one node [11][12].

5. Results

5.1 jGL - A 3D Graphics Library in Java

Currently, we have implemented more than **220** OpenGL functions in jGL, including functions of GLAUX, GLU, and GL. These functions include 2D/3D transformation, 3D projection, depth buffer, smooth shading, lighting, material, display list, selection, texture-mapping, mip-mapping, evaluators, NURBS, and stippled geometry, etc. Functions not supported so far are mainly for anti-aliasing.

To test the capabilities of jGL, we have provided 26 examples on our jGL web page[1]. These examples are selected from the *OpenGL Programming Guide* [13] and can be executed directly on Java-enabled Internet browsers. Since jGL is developed in pure Java, it can be used on all Java-enabled machines. We have performed tests on all major operating systems including Microsoft Windows 98, NT, 2000, Sun Solaris (Sparc and x86), Linux, and SGI Iris.

Figure 5 shows a simple Java applet source code that draws a rectangle using jGL, which is similar to the simple example in the *OpenGL Programming Guide* (Listing 1-2, pp. 13, Figure 1-1), which is an official programming guide of OpenGL.

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

import jGL.GL;
import jGL.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(0.0f, 0.0f, 0.0f, 0.0f);
    myGL.glClear(GL.GL_COLOR_BUFFER_BIT);
    myGL.glColor3f(1.0f, 1.0f, 1.0f);
    myGL.glMatrixMOde(GL.GL_PROJECTION);
    myGL.glLoadIdentity();
    myGL.glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f);
    myGL.glBegin(GL.GL_POLYGON);
      myGL.glVertex2f(-0.5f, -0.5f);
      myGL.glVertex2f(-0.5f, 0.5f);
      myGL.glVertex2f(0.5f, 0.5f);
      myGL.glVertex2f(0.5f, -0.5f);
    myGL.glEnd();
    myGL.glFlush();
  }
}
```

Figure 5 A simple example of jGL to show a white rectangle.

To evaluate the performance of jGL, we used a test program that renders 12 spheres with different materials, where each

[1] http://nis-lab.is.s.u-tokyo.ac.jp/~robin/jGL

sphere contains 256 polygons, as shown in Figure 6. The performance is measured on both a SUN Ultra-10 workstation and an Intel PentiumIII-1G PC. The results are listed in Table 1. This test program is also an example in the *OpenGL Programming Guide* (code from Listing 6-3, pp. 183-184, Plate 16).

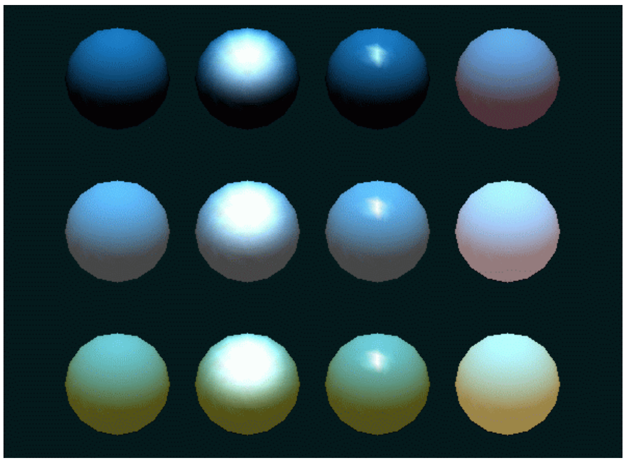
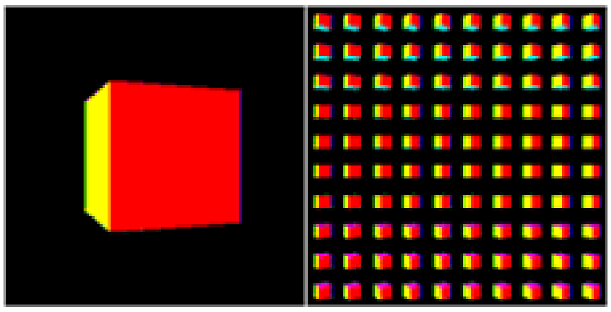


Figure 6 Twelve spheres are rendered to measure the performance. Each sphere contains 256 polygons and has different material. This figure was rendered with jGL.

Environment	Rendering Time	Platform
Sun JDK 1.3	**220** ms	Intel PentiumIII-1GHz, 512MB memory, Microsoft Windows 98
	1061 ms	Sun Ultra-10 360MHz, 256MB memory, Sun Solaris 7

Table 1 The performance testing of jGL on different platforms.

To compare our results with Java3D, we wrote a program that draws a cube with different colors similar to the HelloUniverse example in the package of Java3D, as shown in Figure 7 (a). We tested both programs on the same machine with Intel PentiumIII-1G CPU and NVIDIA GeForce2 GTS, which is a high-level graphics accelerator. The result shows both of them are rendered in real-time.



(a) **(b)**

Figure 7 The programs used to compare the performances of jGL and JavaGL. (a) A simple example to render a simple cube with different colors that is also an example (HelloUnverse) in the package of Java3D. (b) An example to repeat the same cube 100 times for testing the performance. These figures are rendered with jGL.

Since the performance could not be estimated by using a simple model, we use the same cube but repeat it 100 times as shown in Figure 7 (b) to measure the performance. The results are listed in Table 2.

Library	Rendering Time	Platform
jGL 2.3	**110** ms	PC as in Table 1 NVIDIA GeForce2 GTS
Java3D 1.2	**100** ms	

Table 2 The performance comparison for jGL and Java3D.

5.2 VRML Browser Applet by Using jGL

As of this writing, we have implemented more than 70% of all VRML nodes in our VRML browser applet. Besides the nodes, route and event transmission mechanisms have also been implemented.

To evaluate the performance of this VRML browser applet and to demonstrate that the applet can read 3D models from VRML files, we use a model that renders a table with different colors for the legs and top, which is selected from the *Virtual Reality Modeling Language (VRML 97)*, the specification of VRML. This model contains 204 polygons, as shown in Figure 8. The performance of the test file was also measured on both a SUN Ultra-10 workstation and an Intel PentiumIII-1G PC. The results are listed in Table 3.

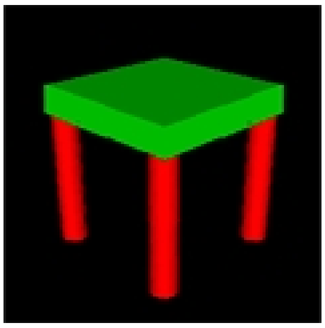


Figure 8 A simple table is rendered to measure performance. It contains 204 polygons and has two different colors for the legs and top. This model is an example in *Virtual Reality Modeling Language (VRML97)* (code pp. 211-213, Figure D.3). This figure is rendered with jGL.

Environment	Rendering Time	Platform
Sun JDK 1.3	**15** ms = **66.7** fps	PC as in Table 1
	23 ms = **43.5** fps	Workstation as in Table 1

Table 3 The performance testing of the VRML browser applet by using a simple table as shown in Figure 8.

To test the performance of the VRML browser applet, we use a model, which has 5,273 polygons and 12 kinds of materials as shown in Figure 9. The testing platform is also Intel PentiumIII-1GHz, 512MB memory, Microsoft Windows NT 4.0, Sun JDK 1.3. The rendering time is about 2.7 fps (frame per second).

To compare with Shout3D and blaxxun3D, we also want to use the same model as we compare the performance of jGL and Java3D. Unfortunately, the model could only be run on Shout3D and our VRML browser applet, but it could not be run on blaxxun3D. So we use the same method as in the previous section, we make a model to display the large model, as shown in Figure 9, and repeat 100 times to estimate the rendering time; the results are listed in Table 4.

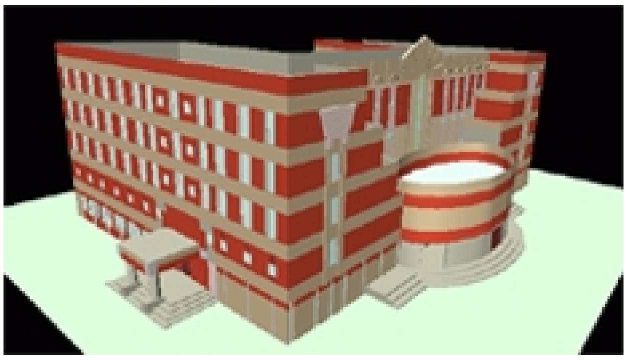


Figure 9 This building model contains 5,273 polygons with 12 kinds of materials with VRML format.

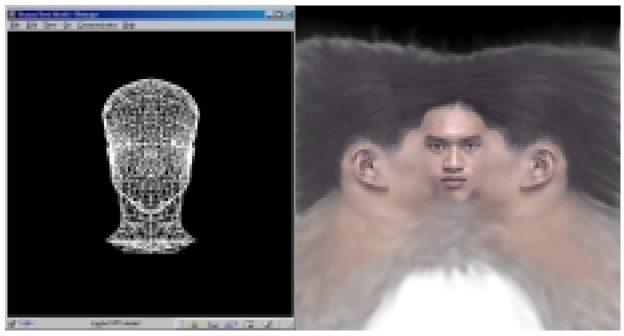
Package	Rendering Time	Platform
Shout3D 1.03	**2.25** sec	PC as in Table 1
Blaxxun3D 1.18	**1.31** sec	
VRML browser applet using jGL	**22.31** sec	

Table 4 The performance comparison for Shout3D, blaxxun3D and our VRML browser applet using jGL. The testing model using the same model as shown in Figure 9, but repeat it 100 times, so it can be seen as a model with 527,300 polygons.

The performance of the VRML browser applet using jGL is much worse than Shout3D and blaxxun3D, and of course worse than other traditional VRML browser, such as Cosmo Player of SGI. That is because we have not optimized the performance of the VRML browser applet, and the purpose of the applet is not only to show the 3D model with VRML file format, but also to support all the browsing functionalities and provide a good testing platform with good modularization.

5.3 3D Human Head Texture Mapping

To test the capability of texture mapping, we also develop a 3D human head texture mapping [13] and build it with jGL. To do this, we use a 3D human head model with 2,185 polygons, and a 512 x 512 human face image as shown in Figure 10.



(a) (b)

Figure 10 The data for the 3D human head texture-mapping (a) the 3D human head model, and (b) the human face image.

Since the entire source code was written in pure Java, and so is jGL, the program can be run on all kinds of Java-enabled platform directly from our web page. The performance results are listed in Table 5. The rendered model is shown in Figure 11.

Environment	Rendering Time	Platform
Sun JDK 1.3	**165** ms	PC as in Table 1
	434 ms	Workstation as in Table 1

Table 5 The performance testing of 3D human head texture mapping on a PC and a workstation.

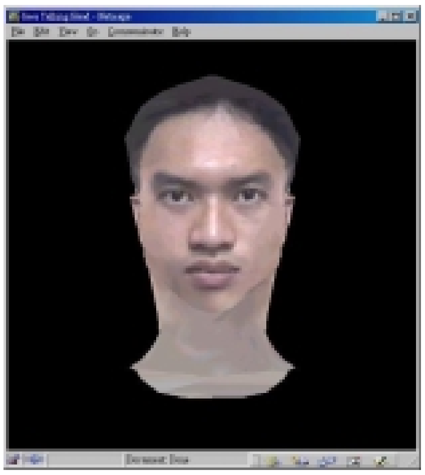


Figure 11 The result of 3D human head texture mapping.

6. Conclusions and Future Work

Since we offer jGL on our web server for download, many people around the world have visited our web page. We also received dozens of e-mails concerning the use of jGL. Some would like to collaborate with us, and some want to use jGL to develop their applications. This encourages us to further improve jGL.

Performance is still the great challenge for any Java applications. We expect that the performance will be improved by better Java interpreters and compilers, and will be greatly improved by new Java chips and faster CPUs.

Sun Microsystems, Inc. has combined its Java2D into Java2, and released the newest version of Java3D recently. Because Java2D is part of Java core packages, it can benefit from hardware acceleration, though this will need many efforts on porting it to each platform. Using Java2D to be the base of jGL may be a solution of the performance problem. But the main contribution of jGL is not only to provide a 3D graphics library for Java, but also to offer a familiar programming environment on the Internet for all 3D programmers.

From our comparison, although Java3D uses OpenGL or DirectX as its graphics engine, the performance of jGL is not worse for the simple model. Hence, jGL seems more suitable for small 3D models on the web, since the user does not need to install any run-time library before using the program. Moreover, the API of jGL is similar to that of OpenGL, so the programmers can use it easily.

When developing jGL and other applications on the Internet, we found that run time performance is so far not the main problem for Java applets, since the machine's performance is getting better and better day-by-day. But the code size problem is a major problem, no mater for byte code size or the model data size, since the network bandwidth is still very narrow.

At this moment, jGL is being applied to develop a VRML browser applet and other Java-based projects in our laboratory. The goal of the Java-based projects is to provide users all the necessary functionality by direct download from the web server, so that users do not have to install additional hardware or software for 3D graphics applications on the web. jGL meets this requirement because it is implemented purely in Java, which is designed for mobile code on the Internet.

The implementation of the VRML browser applet using jGL is not completed yet. jGL as well as the applet were not our original goals. The applet and jGL were developed as useful tools to help us to do some research on the Web3D platform. So we only develop the necessary parts in these packages when we need them. That is also the reason that the VRML browser applet does not support all nodes and jGL does not support all functions of the respective standards yet.

Since the network bandwidth is the most important problem for 3D graphics on the Internet, after minimizing the code size of jGL and our VRML browser applet, we will try to minimize the size of the model or use progressive refinement technologies to this end. Of course, to enhance the performance of the VRML browser applet is part of our future work plans.

We have developed jGL for almost four years, to make intend to release it into open-source in the near future.

Acknowledgements

We would like expressing our appreciation to Prof. Ming Ouhyoung, who supported the development of jGL since 1996, and always gives us many suggestions for developing it. We also want to thank the Graphics Group of Communication and Multimedia Lab., National Taiwan University, which created the model for 3D human head texture mapping.

References

[1] "The Source for Java™ Technology," Sun Microsystems, Inc., 2000. http://java.sun.com.

[2] "OpenGL – High Performance 2D/3D Graphics," OpenGL, Org., 2000. http://www.opengl.org.

[3] Bing-Yu Chen, Tzong-Jer Yang, and Ming Ouhyoung. JavaGL - a 3D Graphics Library in Java for Internet Browsers. In IEEE Trans. on Consumer Electronics, Vol. 43, No. 3, pages 271 - 278, 1997.

[4] Rikk Carey, Gavin Bell, and Chris Marrin. ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97). The VRML Consortium Incorporated, 1997.

[5] JavaSoft. The Java 3D API Specification Version 1.2. Sun Microsystems, Inc., 2000.

[6] "Shout3D v1.03," Shout Interactive, Inc., 2000.
http://www.shout3d.com

[7] "blaxxun3D v1.18," Blaxxun Interactive, Inc., 2000.
http://www.blaxxun.com/products/blaxxun3d

[8] "Cortona Jet," Parallel Graphics, Inc., 2000.
http://www.parallelgraphics.com/products/jet

[9] "Web3D Consortium," Web3D Consortium, 2000.
http://www.web3d.org.

[10] Mark Segal, and Kurt Akeley. The OpenGL Graphics Systems: A Specification (Version 1.1). Silicon Graphics, Inc., 1996.

[11] Hugues Hoppe. Efficient Implementation of Progressive Meshes. In Computer & Graphics, Vol. 22, No. 1, pages 27-36, 1998.

[12] Hugues Hoppe. Progressive Meshes. In Computer Graphics (SIGGRAPH 96 Conference Proceedings), pages 99-108, 1996.

[13] Jackie Neider, Tom Davis, and Mason Woo. OpenGL Programming Guide. Addison-Wesley, 1993.

[14] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Fredrick Pighin. Making Faces. In Computer Graphics (SIGGRAPH 98 Conference Proceedings), pages 55-66, 1998.