The Development of 3D Graphics and VRML Libraries for Web3D Platform by Using Java

Bing-Yu Chen[1] and Tomoyuki Nishita[2]

[1]Department of Computer Science, The University of Tokyo, 113-0033 Japan

[2]Department of Complexity Science and Engineering, The University of Tokyo, 113-0033 Japan

SUMMARY

This paper proposes a new 3D graphics programming environment for Web3D on the Internet. To develop 3D graphics programs on the Internet is not easy because there is no popular 3D graphics library like OpenGL. For this purpose, we developed a 3D graphics library called jGL by using Java exclusively. jGL is a general-purpose 3D graphics library, and its API (Application Programming Interface) is defined in a manner quite similar to that of OpenGL. Since jGL offers the same functionalities as OpenGL, the programmers can use it easily. Furthermore, since people can use some simple Java programs on cellular phones recently, the migration experimentation toward i-αppli is also described in this paper. Moreover, VRML (Virtual Reality Modeling Language) is a standard file format for describing 3D models on the Internet. To display a 3D model on the Internet, people may like to use the VRML file format. Therefore, we also developed a VRML library called jVL by using Java exclusively as an extension of jGL. In this paper, the results and some detail descriptions are updated and different from those in [3].

Key words: OpenGL; VRML; Web3D; Java; graphics library.

1. Introduction

Recently, the Internet users are growing day by day, and so are the demands of 3D graphics on the Internet. However, when we develop a 3D graphics program on the Internet, since the Internet itself is a heterogeneous network environment, to offer several versions of the same program for several different platforms is not easy. This is so-called the platform dependant problem. Although Java[1] can be used to solve this problem now because of its hardware-neutral features, its 3D graphics support forms another problem.

To develop a 3D graphics program on a stand-alone computer, the programmer always uses a general-purpose 3D graphics library, like OpenGL[2]. However, there is no such a library on the Internet when using Java. Moreover, if there is a 3D graphics library for Java, it should be easy to learn and use. Therefore, we developed a 3D graphics library called jGL, and defined the API of it in a manner quite similar to that of OpenGL. Hence, while using jGL, the programmers do not need to learn how to use a entirely new 3D graphics library, so that to develop 3D graphics applications on the Internet becomes easier than before.

Unlike Java 3D and other similar 3D graphics libraries for the Internet; jGL does not need any native codes or pre-installed libraries, such as OpenGL, it is developed with pure Java only. To develop a 3D graphics application with jGL on the Internet, the programmer can just ignore how to do the 3D graphics rendering on the

[1] http://java.sun.com/
[2] http://www.opengl.org/

Java VM (Virtual Machine). The entire 3D graphics rendering is done by jGL. If the application is designed to be run by other users on the Web, all he or she has to do is just to put it on the Web server with jGL. Then, the users can use this program via the in-lined applet on any Java enabled Web browser. Moreover, the users who run any application developed with jGL do not need to install any package even jGL before using it; all of the required codes are downloaded at run-time from the server.

We released the first version of jGL at the end of 1997 [4]. That version provided only the basic rendering routines due to the machine performance. Moreover, some complex functions, such as texture mapping, are also ignored, since those functions are time-consuming. However, because the computer hardware has much improved and more and more fancy 3D graphics applications have been required for the Internet, we therefore decided to enhance the capabilities of jGL. Unfortunately, although the computer 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 size become too large to be transmitted over the Internet. To enhance the capabilities and minimize the code size at the same time, the kernel of jGL has been re-written.

Recently, some simple Java applets can be run on cellular phones, for example the i-αppli[3] of the 503i and 504i series provided by NTT DoCoMo in Japan. Since i-αppli is not standard and based on Java 2 SDK, Micro Edition (J2ME) that is different from the platform of jGL, which is Java 2 SDK, Standard Edition (J2SE), jGL can not be run on the i-αppli platform directly. To make the cellular phones to have 3D graphics supports, we ported some basic parts of jGL onto the i-αppli platform.

Besides the 3D graphics library as a programming environment for the Internet, we also need to show and use 3D models while developing 3D graphics applications. Hence, we also used pure Java to develop a VRML library called jVL by following the API of jGL and the specification of VRML [2], since VRML is a standard file format for describing 3D models on the Internet. Furthermore, the 3D graphics rendering of jVL is also done by jGL.

2. Related systems

For the Web3D platform, there are some related systems on the Internet. Some are 3D graphics libraries (Java 3D and JSparrow in the following); others are for displaying 3D objects or scenes (Eyematic Shout3D, blaxxun3D, Cortona Jet, and Xj3D in the following).

Java 3D – Java 3D[4] is provided by Sun Microsystems, Inc. as the 3D support for Java programming language, but has not become an official part of core Java package. Java 3D needs the support from OpenGL or DirectX, which has been pre-installed. Since it can get the benefits from the graphics hardware the run-time performance is good, but the platform dependent problem is occurred. To use a program based on Java 3D, the users have to install the run-time package of Java 3D before using it. Moreover, the API of Java 3D is specific, so people who want to develop some 3D graphics programs with Java 3D may spend much time to learn how to use it.

JSparrow – JSparrow[5] is an implementation of Java binding for OpenGL provided by PFU, Ltd. Since it needs the support from native OpenGL, it also has the platform dependent problem as Java 3D. Moreover, to use a program developed with it, the users also have to install the JSparrow package before using it.

Eyematic Shout3D – Eyematic Shout3D[6] is a commercial product of Eyematic Interfaces, Inc. It is developed using pure Java and can display 3D models on the Web. Although the file format of the 3D model is not VRML, it provides a converter from VRML to its own file format (not one-to-one mapping). Like VRML, it also provides its own API to let people program animation of the 3D model.

blaxxun3D – blaxxun3D[7] is a commercial product of Blaxxun Interactive, Inc. As Eyematic Shout3D, it is also developed using pure Java and can display 3D models on the Web. It reads VRML and draft X3D (Extensible 3D; the next generation of VRML) 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 Eyematic Shout3D and blaxxun3D, it is also developed using pure Java and can display 3D models on the Web. It reads VRML file for-

[3] http://www.nttdocomo.co.jp/p_s/imode/java/

[4] http://java.sun.com/products/java-media/3D/

[5] http://home.att.ne.jp/red/aruga/jsparrow/

[6] http://www.shout3d.com/products_shout3d.html

[7] http://www.blaxxun.com/products/blaxxun3d/

[8] http://www.parallelgraphics.com/products/jet/

mat, but does not provide any API for programming.

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

As listed in Table 1, since Java 3D 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 a familiar API is useful. Moreover, although there are several programs could be used to show 3D models, the users can not use them for programming such as changing the light sources or the actions of the 3D model. Therefore, the development of 3D graphics programs on the Internet is easier than before by using jGL and jVL.

Table 1. The comparison of related systems.

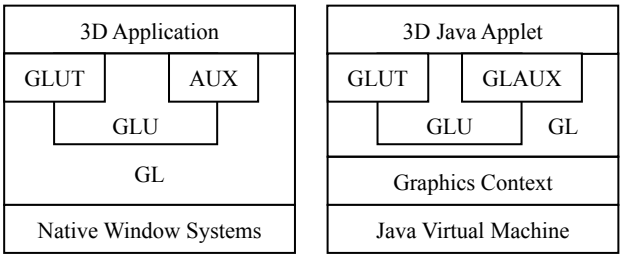
systems	pure Java	famous API	VRML
Java 3D	no	not at all	no
JSparrow	no	yes	no
Shout3D	yes	no	yes
blaxxun3D	yes	no	yes
Cortona Jet	yes	no	yes
Xj3D	no	no	yes
jGL + jVL	yes	yes	yes

3. Introduction of jGL

jGL is a 3D graphics library for Java and has no necessary to be pre-installed (all of the required codes are downloaded at run-time). Moreover, since OpenGL is so famous and has been known by many 3D graphics programmers, we defined the API of jGL in a manner quite similar to that of OpenGL by following the specifications of it [11]. Therefore, the programmers who want to use jGL to develop 3D graphics programs on the Internet do not need to learn how to use an entirely new library; they can find one-to-one mapping functions in jGL and those in OpenGL.

3.1 OpenGL vs. jGL

The functions of OpenGL can be divided into two main categories: OpenGL Utility Library (GLU) and OpenGL (GL) as shown in Fig. 1 (a). jGL follows the same function hierarchy and is defined as Fig. 1 (b).



(a) (b)

Fig. 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, OpenGL Utility Toolkit (GLUT) has been widely used by programmers also, so we also implemented this part.

The implementations of GL, GLU, and GLUT of jGL are mainly based on the specifications of OpenGL, GLU [5], and GLUT [8]. Besides GL, GLU, GLUT, and GLAUX, which are just interfaces for programmers, there is an underlying graphics context, which is transparent to programmers.

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 design the system hierarchy of the graphics context as shown in Fig. 2.

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

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. The other type performs the real

[9] http://www.web3d.org/TaskGroups/source/xj3d.html

actions and directly produces some changes due to the stored information. We also refer to [1] [6] [7] [9] to optimize the rendering algorithms for performance enhancements.

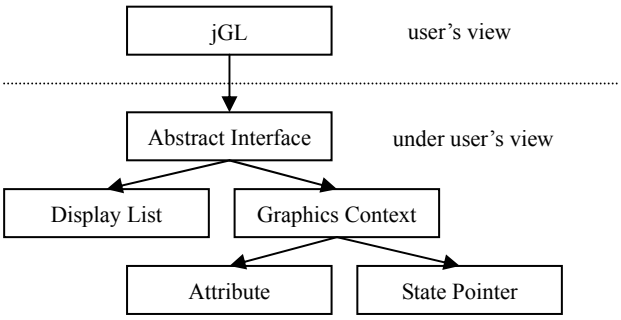


Fig. 2. The graphics context of jGL.

3.3 Performance enhancement issues

The run-time performance is a great challenge for both 3D graphics and Java application. Moreover, jGL is designed to operate over the Internet, where the network bandwidth affects the overall performance significantly. Hence, the performance enhancement issues of jGL do not only contain the run-time performance but also the capability enhancement and the byte-code size minimization.

Therefore, we use the following four policies to develop jGL: (1) utilize class inheritance to avoid "if-then-else" statements; (2) divide a routine into several smaller ones; (3) group rarely used routines into a larger one; (4) use function-overriding to minimize code size.

3.4 Porting experimentation for i-αppli

The development environment of jGL is different from the i-αppli platform which is a specific Java VM and operated on cellular phones. To port jGL onto it, we have to obey the following limitations due to the i-αppli platform: (1) there is no floating-point data type; (2) the size of the jar-ball must be smaller than 10KB; (3) there is only a few primitive drawing functions provided by i-αppli. [10] Therefore, to port jGL onto the i-αppli platform, we re-implemented all of the necessary calculations by using only the integers. To minimize the jar-ball size, we removed all of the unnecessary constants and error checks. Finally, we have implemented more than 30 OpenGL functions in the i-αppli version of jGL, which include 3D model transformation, 3D object projection, hidden-surface removal, primitive geometry, etc. Moreover, the jar-ball size is about 4,194 bytes.

4. Introduction of jVL

jVL is a VRML library for Java and also an extension of jGL. To provide a new programming environment for the Internet, besides a 3D graphics library, a library for displaying 3D models is also necessary. Therefore, also to test the capabilities of jGL, we use jGL and follow the specification of VRML to develop jVL. jVL is also developed with pure Java only, hence it has no platform dependant problem and has no necessary to be pre-installed. Moreover, to make jVL to be easy to use, we define its API by following the API of jGL.

4.1 System architecture

The VRML data structure can be mainly divided into two parts, which are nodes and fields. A 3D model is associated with several nodes with a tree structure, and the parameters of the nodes are stored in some fields. Then, we designed the system hierarchy of jVL as shown in Fig. 3 by following the VRML data structure.

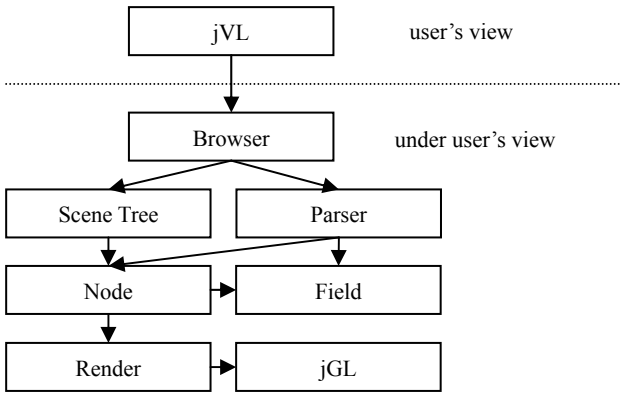


Fig. 3. The system hierarchy of jVL.

As shown in Fig. 3, the Node and Field are the two main parts for constructing a 3D model, jVL is only an interface for programmers, and all of the algorithms are contained in the Browser. While loading a VRML file, the Parser is called to parse the file to generate the Scene Tree which represents the model. When rendering the Scene Tree, jGL is used to generate the image by using the data stored in the Nodes and Fields.

4.2 Performance enhancement issues

From the experiences of developing jGL, we use the following two policies to develop jVL: (1) utilize class inheritance and function-overriding as jGL; (2) use display list mechanism of jGL.

5. Results

Currently, we have implemented more than 300 common used OpenGL functions in jGL, including GL, GLU, GLUT, and GLAUX. These functions include 2D / 3D model transformation, 3D object projection, depth buffer, smooth shading, lighting, material property, display list, selection, texture-mapping, mip-mapping, evaluator, NURBS (Non-Uniform Rational B-Splines), stippled geometry, etc. Functions not supported so far are mainly for anti-aliasing.

```
import jgl.GL;
import jgl.GLApplet;

public class hello extends GLApplet {
    public void display () {
        myGL.glClear (GL.GL_COLOR_BUFFER_BIT);
        myGL.glColor3f (1.0f, 1.0f, 1.0f);
        myGL.glBegin (GL.GL_POLYGON);
            myGL.glVertex3f (0.25f, 0.25f, 0.0f);
            myGL.glVertex3f (0.75f, 0.25f, 0.0f);
            myGL.glVertex3f (0.75f, 0.75f, 0.0f);
            myGL.glVertex3f (0.25f, 0.75f, 0.0f);
        myGL.glEnd ();
        myGL.glFlush ();
    }

    public void init () {
        myUT.glutInitWindowSize (500, 500);
        myUT.glutInitWindowPosition (0, 0);
        myUT.glutCreateWindow (this);
        myGL.glClearColor (0.0f, 0.0f, 0.0f, 0.0f);
        myGL.glMatrixMode (GL.GL_PROJECTION);
        myGL.glLoadIdentity ();
        myGL.glOrtho (0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
        myUT.glutDisplayFunc ("display");
        myUT.glutMainLoop ();
    }
}
```

Fig. 4. The source code of a simple example of jGL to show a white rectangle.

To test the capabilities of jGL, we provided 30 examples on jGL Web page[10]. These examples are selected from the OpenGL Programming Guide [13], which is an official programming guide of OpenGL. Since jGL is developed in pure Java, these examples can be executed on all of the Java-enabled machines. We have performed the tests on all of the major operating systems including Microsoft Windows 98/NT/2000/XP, Sun Solaris 7/8/9 (SPARC and Intel platform editions), SGI IRIX 6.3/6.4/6.5, and Linux.

Fig. 4 shows the source code of a Java applet as a simple jGL program using GLUT to show a white rectangle, which is similar to a simple example provided in the OpenGL Programming Guide (code from Example 1-2, pages 18-19, Figure 1-1).

Fig. 5. 24 teapots with different material properties are rendered with jGL.

Table 2. The performance testing of Fig. 5.

rendering time	Platform
609 ms	Intel Pentium 4 2.8GHz, 1 GB memory, Microsoft Windows XP
4,507 ms	Sun UltraSPARC IIi 360MHz, 256MB memory, Sun Solaris 9

To evaluate the run-time performance of jGL, we used a test program that renders 24 lighted, smooth-shaded teapots drawn with different material properties that approximate real materials, where each teapot is generated by Bésier surface generating functions of jGL (evaluator functionality of OpenGL) and contains 12,544 polygons. The rendering result is shown

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

in Fig. 5. This test program is also an example in the OpenGL Programming Guide (Plate 17). The results are listed in Table 2. The Java environment used in this paper is J2SE v 1.4.1_01.

To test the i-αppli version of jGL, we used a movable robot arm as shown in Fig. 6. This test program is also an example in the OpenGL Programming Guide (code from Example 3-7, pages 148-150, Figure 3-25). Moreover, to use the button on the cellular phone, the robot arm can be acted as the example in the OpenGL Programming Guide.

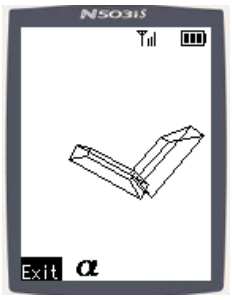
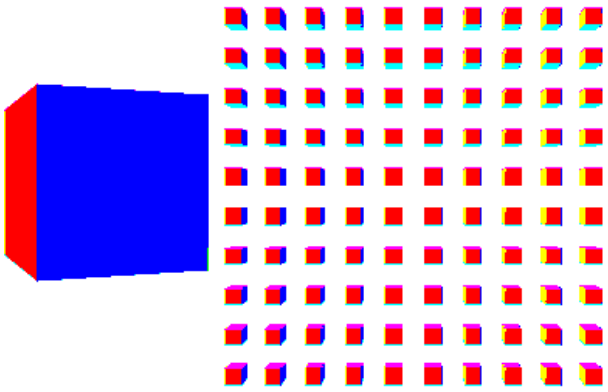


Fig. 6. A robot arm is rendered on a mobile phone.

To compare with Java 3D, we wrote a program that draws a rotating cube drawn with different color values similar to the "HelloUniverse", which is a simple Java 3D example in the Java 3D API Specification [12] (Section 1.6.3, pages 9-10) as shown in Fig. 7 (a). We tested it and the Java 3D example on the same machine. The results show that both of them are rendered in real-time.



(a) (b)

Fig. 7. The programs used to measure the performances of jGL and Java 3D. (a) A rotating cube drawn with different color values. (b) An example to repeat the same cube 100 times.

Since the run-time performances can not be estimated by using a simple model, we use the same cube but repeat it 6,400 times to measure their run-time performances. Fig. 7 (b) shows a similar example of repeating the same cube only 100 times. By using the desktop PC used in Table 2, the rendering time of jGL and Java 3D are 406 ms and 353 ms, respectively. The graphics accelerator installed in the desktop PC is using an NVIDIA Quadro 4 900XGL GPU.

As of this writing, we have implemented more than 70% of all VRML nodes in jVL. Besides the nodes, route and event transmission mechanisms have also been implemented. To evaluate the run-time performance of jVL, we used a simple table with variable colors for the legs and top might be prototyped as shown in Fig. 8, which is selected from the specification of VRML (code from Section D.4, pages 211-213, Figure D.3). The results are listed in Table 3.

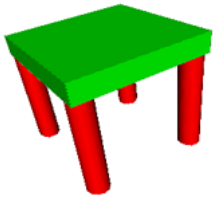


Fig. 8. A simple table is rendered with jVL.

Table 3. The performance testing of Fig. 8.

rendering time	Platform
10 ms = 100 fps	desktop PC as Table 2
47 ms = 21.3 fps	workstation as Table 2

Table 4. The performance testing for data size of jVL.

number of cubes	rendering time
100	30 ms
400	90 ms
900	190 ms
1,600	320 ms
2,500	490 ms
3,600	691 ms
4,900	931 ms
6,400	1,202 ms
8,100	1,512 ms
10,000	1,853 ms

Since the performance could not be estimated by using a simple model, we use more than 100 rotating cubes drawn with different colors as shown in Fig. 7 (b),

which is the example of only 100 rotating cubes. Table 4 lists the performance of evaluating the rendering time according to the model data size on the laptop PC as Table 2.

Fig. 9 shows the source code for displaying a VRML file by using jVL and jGL. Moreover, to change or control the objects, lights, and sensors included in the VRML file is also possible.

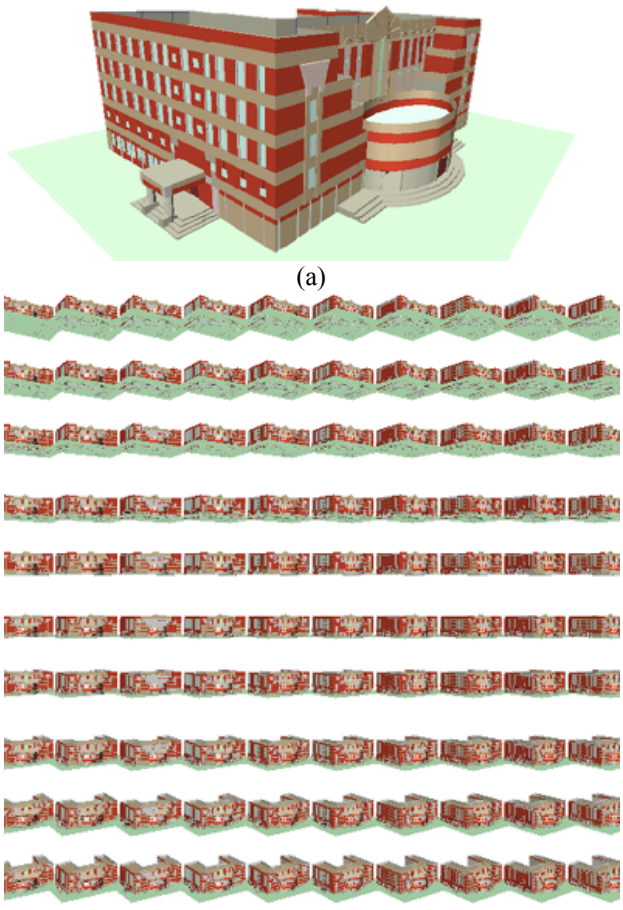
```
import jvl.VL;
import jgl.GLApplet;

public class viewer extends GLApplet {
    VL myVL = new VL (myGL);

    public void display () {
        myVL.vlRenderWorld ();
    }

    public void init () {
        myUT.glutInitWindowSize (500, 500);
        myUT.glutInitWindowPosition (0, 0);
        myUT.glutCreateWindow (this);
        myVL.vlSetWorldURL (getDocumentBase(), "exampled.4.wrz");
        myVL.vlInit ();
        myVL.vlViewpoint (getSize ().width, getSize ().height);
        myUT.glutDisplayFunc ("display");
        myUT.glutMainLoop ();
    }
}
```

Fig. 9. The source code using jVL for displaying a VRML file.

To compare with Eyematic Shout3D and blaxxun3D, we used a VRML file which contains 100 3D building models as shown in Fig. 10 (b). Each building model has 5,273 polygons with 12 different material properties as shown in Fig. 10 (a). By using the desktop PC as shown in Table 2, the run-time performances of all of them are about 2 seconds per frame.

6. Conclusions

This paper presented the implementation of jGL and jVL using pure Java and the experimentation for porting jGL onto the i-αppli platform. From our comparisons, although Java 3D uses OpenGL or DirectX as its graphics engine, the performance of jGL is not worse for a 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 which is developed with it. Since we offer jGL onto our Web server, many people around the world are using it to provide their previous OpenGL works to be Web-enabled versions or to teach and learn the computer graphics algorithms. In order to get the feedbacks from the users, we make a questionnaire of jGL and mail to 50 users around the world. The result is shown in Table 5. Most of the users thought that jGL is easy to use and has good reliability.



(a) (b)

Fig. 10. The model used to compare the performances. (a) The rendering of a 3D building model. (b) An example to repeat the same model as (a) 100 times.

Although the run-time performance of jVL is not so good, because it provides an API for programming, to change the light source and the activities of the 3D model is possible. Moreover, since it has the same advantages as jGL, it is possible to use jVL to develop programs on the Web3D platform. Hence, the development of 3D graphics applications on the Internet is easier than before by using jGL and jVL. The run-time performance is still the great challenge for jGL and jVL. 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.

Table 5. The questionnaire result of jGL.

	excellent	good	bad	very bad
easy to use	5	45	0	0
reliability	8	42	0	0

REFERENCES

1. Arvo J. Graphics Gems II. Academic Press, Inc.; 1991.
2. Carey R., Bell G., and Marrin C. ISO/IEC 14772-1: 1997 Virtual Reality Modeling Language (VRML97). VRML Consortium, Inc.; 1997.
3. Chen B.-Y. and Nishita T. The development of 3D graphics and VRML libraries for Web3D platform by using Java. IEICE Trans Information and Systems 2002; J85-D-II(6): 1047-1054. (in Japanese)
4. Chen B.-Y., Yang T.-J., and Ouhyoung M. JavaGL - A 3D graphics library in Java for Internet browsers. IEEE Trans Consumer Electronics 1997;43(3): 271-278.
5. Chin N., Frazier C., Ho P., Liu Z., and Smith K. P. The OpenGL® Graphics Systems Utility Library (Version 1.3). Silicon Graphics, Inc.; 1998.
6. Foley J. D., van Dam A., Feiner S. K., and Hughes J. F. Computer Graphics: Principles and Practice in C. Addison-Wesley; 1996.
7. Glassner A. S. Graphics Gems. Academic Press, Inc.; 1990.
8. Kilgard M. J. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3. Silicon Graphics, Inc.; 1996.
9. Kirk D. Graphics Gems III. Academic Press, Inc.; 1992.
10. NTT DoCoMo, Inc. i-mode Java Contents Development Guide; 2001. (in Japanese)
11. Segal M. and Akeley K. The OpenGL® Graphics Systems: A Specification (Version 1.4). Silicon Graphics, Inc.; 2002.
12. Sowizral H., Rushforth K., and Deering M. The Java 3D™ API Specification. Addison-Wesley; 2000.
13. Woo M., Neider J., Davis T., and Shreiner D. OpenGL® Programming Guide, Version 1.2. Addison-Wesley; 1999.

AUTHORS (from left to right)

Bing-Yu Chen received the B.S. and M.S. degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1995 and 1997, respectively, and received the Ph.D. degree in Information Science from the University of Tokyo, Japan, in 2003. He worked for the GHQ of the Taiwan Army from 1997 to 1999. He is currently an assistant researcher in the Department of Computer Science of the University of Tokyo. His research interest is mainly for Web Graphics. He is a member of ACM and IEEE.

Tomoyuki Nishita received the B.S., M.S., and Ph.D. degrees in Electrical Engineering from the Hiroshima University, Japan, in 1971, 1973, and 1985, respectively. He worked for Mazda Motor Corp. from 1973 to 1979. He has been a lecturer at the Fukuyama University since 1979, then became an associate professor in 1984, and later became a professor in 1990. He moved to the Department of Information Science of the University of Tokyo as a professor in 1998 and now is a professor at the Department of Complexity Science and Engineering of the University of Tokyo since 1999. His research interest is mainly for Computer Graphics. He is a member of IEICE, IPSJ, ACM, and IEEE.