Systems and Computers in Japan, Vol. 34, No. 10, 2003
Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol. J85-D-II, No. 6, June 2002, pp. 1047–1054

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, programmers can use it easily. Furthermore, since people have recently been able to use simple Java programs on cellular phones, 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 detailed descriptions are updated compared with those in Ref. 3. © 2003 Wiley Periodicals, Inc. Syst Comp Jpn, 34(10): 47–55, 2003; Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/scj.10393

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

1. Introduction

Recently, the number of Internet users has been growing day by day, as has the demand for 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, offering several versions of the same program for several different platforms is not easy. This is the so-called platform dependent problem. Although Java* 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.† However, there is no such library on the Internet when using Java. Moreover, if there were 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 its API in a manner quite similar to that of OpenGL. Hence, while using jGL, the programmer does not need to learn how to use an entirely new 3D graphics library, and developing 3D graphics applications on the Internet is easier than before.

Unlike Java 3D and other similar 3D graphics libraries for the Internet, jGL does not need any native codes or preinstalled libraries, such as OpenGL; it is developed in pure Java. To develop a 3D graphics application with jGL

*http://java.sun.com/
†http://www.opengl.org/

on the Internet, the programmer can ignore the specifics of 3D graphics rendering on the Java VM (Virtual Machine). All 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 to put it on the Web server with jGL. Anyone can then use this program via the in-lined applet on any Java-enabled Web browser. Moreover, users who run any application developed with jGL do not need to install any package, even jGL, before using it; all of the required code is 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 machine performance issues. Moreover, some complex functions, such as texture mapping, were also ignored, since those functions are time-consuming. However, because the computer hardware has greatly improved and more and more sophisticated 3D graphics applications have been required for the Internet, we decided to enhance the capabilities of jGL. Unfortunately, although computer hardware is getting faster and faster day by day, the network bandwidth is still the same as before, or even worse. Thus, increasing the capabilities of jGL might make its code size too large to be transmitted over the Internet. To enhance its capabilities and minimize the code size at the same time, the kernel of jGL was rewritten.

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

Besides the 3D graphics library as a programming environment for the Internet, we also need to display 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); others are for displaying 3D objects or scenes (Eyematic Shout3D, blaxxun3D, Cortona Jet, and Xj3D).

Java 3D – Java 3D[*] is provided by Sun Microsystems, Inc. as 3D support for the Java programming language, but has not become an official part of the core Java package. Java 3D needs support from OpenGL or DirectX, which has been preinstalled. Since it can take advantage of the graphics hardware, its run-time performance is good, but the platform-dependent problem occurs. To use a program based on Java 3D, the user must first install the run-time package of Java 3D. Moreover, the API of Java 3D is specific, so that persons who want to develop some 3D graphics programs with Java 3D may have to spend much time learning how to use it.

JSparrow - JSparrow[†] is an implementation of the Java binding for OpenGL provided by PFU, Ltd. Since it needs support from native OpenGL, it has the same platform-dependent problem as Java 3D. Moreover, to use a program developed with it, the user also must install the JSparrow package.

Eyematic Shout3D - Eyematic Shout3D[‡] 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 a one-to-one mapping). Like VRML, it also provides its own API to let people program animations of 3D models.

blaxxun3D - blaxxun3D[§] is a commercial product of Blaxxun Interactive, Inc. Like Eyematic Shout3D, it is developed using pure Java and can display 3D models on the Web. It reads the VRML and draft X3D (Extensible 3D; the next generation of VRML) file formats, although it does not fully support either. Following the concepts of JavaEAI (Java External Authoring Interface), blaxxun3D also provides an API to let people program 3D models.

Cortona Jet - Cortona Jet[**] is a commercial product of Parallel Graphics, Inc. Like Eyematic Shout3D and blaxxun3D, it is developed in pure Java and can display 3D models on the Web. It reads the VRML file format, but does not provide any API for programming.

Xj3D - Xj3D[††] 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 the VRML and draft X3D file formats exactly; actually it is a testing platform for X3D.

As shown in Table 1, since Java 3D is not platform independent and programmers must spend time learning how to use it, we believe that it is useful to provide a real platform-independent 3D graphics library with a familiar

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

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

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

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

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

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

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

Table 1. 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

API. Moreover, although there are several programs that could be used to display 3D models, they cannot be used for programming such as changing the light source or the actions of the 3D model. Therefore, the development of 3D graphics programs on the Internet is made easier than before by using jGL and jVL.

3. Introduction of jGL

jGL is a 3D graphics library for Java and need not be preinstalled (all of the required code is downloaded at run-time). Moreover, since OpenGL is so well known and has been used by many 3D graphics programmers, we defined the API of jGL in a manner quite similar to that of OpenGL by following its specification [11]. Therefore, 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 a one-to-one mapping between functions in jGL and those in OpenGL.

3.1. OpenGL versus jGL

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

GL implements a powerful but small set of drawing primitive 3D graphics operations, including rasterization and clipping. GLU provides higher-level OpenGL commands to programmers by encapsulating them 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 the

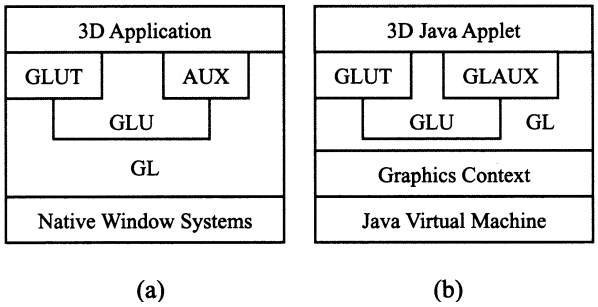


Fig. 1. The hierarchy of (a) OpenGL and (b) jGL modules.

OpenGL API, but is widely used and is familiar to many programmers. For this reason, we also include GLAUX in our package. Recently, the OpenGL Utility Toolkit (GLUT) too has been widely used by programmers, 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 the programmer.

3.2. Implementation of graphics context

To reduce the code size and maintain or enhance the performance of jGL, we utilized the class inheritance characteristics to design the system hierarchy of the graphics context shown in Fig. 2.

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

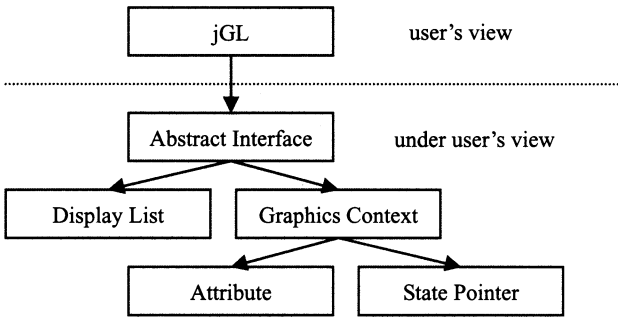


Fig. 2. The graphics context of jGL.

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 divided into two types. One type is just for changing information stored in the graphics context. The other type performs the real actions and directly produces changes due to the stored information. We also referred to Refs. 1, 6, 7, 9 to optimize the rendering algorithms for performance enhancement.

3.3. Performance enhancement issues

Run-time performance is a great challenge for both 3D graphics and Java applications. 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 include not only run-time performance but also capability enhancement and byte-code size minimization.

Therefore, we used 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 larger routines; (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 operates on cellular phones. To port jGL to it, we must comply with 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 10 kB; (3) there are only a few primitive drawing functions provided by i-αppli [10]. Therefore, to port jGL to the i-αppli platform, we reimplemented all of the necessary calculations using only integers. To minimize the jar-ball size, we removed all unnecessary constants and error checks. Finally, we have implemented more than 30 OpenGL functions in the i-αppli version of jGL, including 3D model transformation, 3D object projection, hidden-surface removal, and primitive geometry. The jar-ball size is about 4194 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, in addition to a 3D graphics library, a library for displaying 3D models is also necessary. Therefore, to test the capabilities of jGL, we used jGL and followed the VRML specification in developing jVL. jVL was also developed in pure Java, and hence it has no platform-dependent problem and has no necessity of being preinstalled. Moreover, to make jVL easy to use, we defined its API by following the API of jGL.

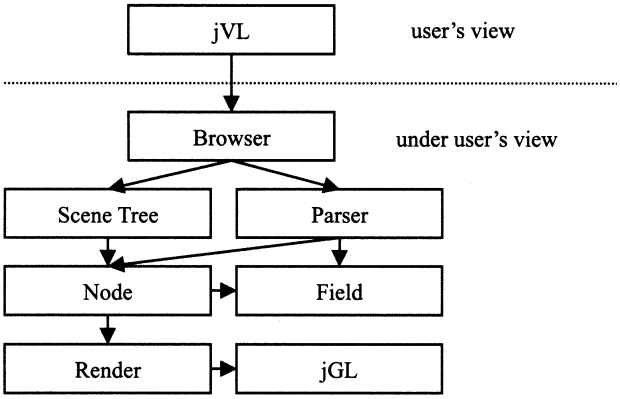


Fig. 3. The system hierarchy of jVL.

4.1. System architecture

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

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 and 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

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

5. Results

Currently, we have implemented more than 300 commonly used OpenGL functions in jGL, including GL, GLU, GLUT, and GLAUX. These functions include 2D/3D model transformation, 3D object projection, depth buffer-

ing, smooth shading, lighting, material properties, display lists, selection, texture-mapping, mip-mapping, evaluator, NURBS (Non-Uniform Rational B-Splines), and stippled geometry. The functions not supported so far are mainly for anti-aliasing.

To test the capabilities of jGL, we provided 30 examples on the jGL Web page.* These examples were 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 run on all Java-enabled machines. We have performed 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.

Figure 4 shows the source code of a Java applet, 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, pp. 18–19, Fig. 1-1).

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 the Bézier surface generating functions of jGL (evaluator functionality of OpenGL) and contains 12,544 polygons. The rendering result is shown 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, pp. 148–150, Fig. 3-25). Moreover, using the buttons on the cellular phone, the robot arm can be operated as in the example in the OpenGL Programming Guide.

To compare with Java 3D, we wrote a program that draws a rotating cube with different color values similar to "HelloUniverse," which is a simple Java 3D example in the Java 3D API Specification [12] (Section 1.6.3, pp. 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.

Since run-time performance cannot be evaluated by a simple model, we use the same cube but repeat it 6400 times to measure the run-time performance. Figure 7(b) shows a similar example of repeating the same cube only 100 times. On the desktop PC described in Table 2, the rendering time of jGL and Java 3D was 406 and 353 ms, respectively. The graphics accelerator installed in the desktop PC used an NVIDIA Quadro 4 900XGL GPU.

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

```
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.

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 which might be prototyped as shown in Fig. 8, which is selected from the VRML specification (code from Section D.4, pp. 211–213, Fig. D.3). The results are listed in Table 3.

Since the performance could not be evaluated by a simple model, we used more than 100 rotating cubes drawn

Fig. 5. Twenty-four teapots with different material properties rendered with jGL.

with different colors as shown in Fig. 7(b), which is an example with only 100 rotating cubes. Table 4 lists the rendering time according to the model data size on the laptop PC listed in Table 2.

Figure 9 shows the source code for displaying a VRML file by using jVL and jGL. Changing or controlling the objects, lights, and sensors included in the VRML file is also possible.

To compare with Eyematic Shout3D and blaxxun3D, we used a VRML file containing 100 3D building models as shown in Fig. 10(b). Each building model has 5273

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

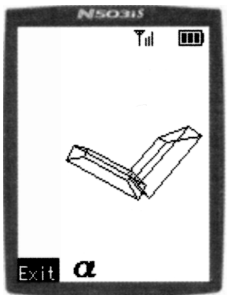
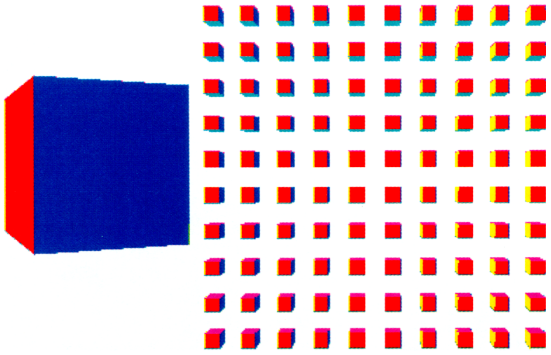


Fig. 6. Robot arm rendered on a mobile phone.



(a) (b)

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

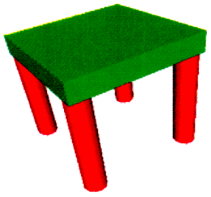


Fig. 8. A simple table rendered with jVL.

Table 3. Performance testing of Fig. 8

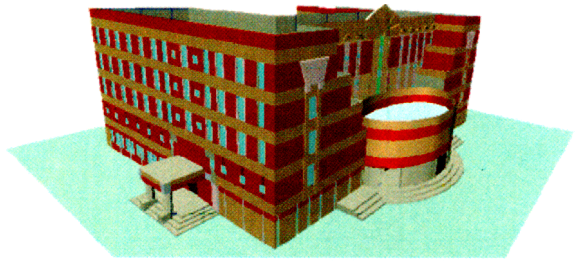
rendering time	Platform
10 ms = 100 fps	desktop PC as Table 2
47 ms = 21.3 fps	workstation as Table 2

Table 4. Performance testing for data size of jVL

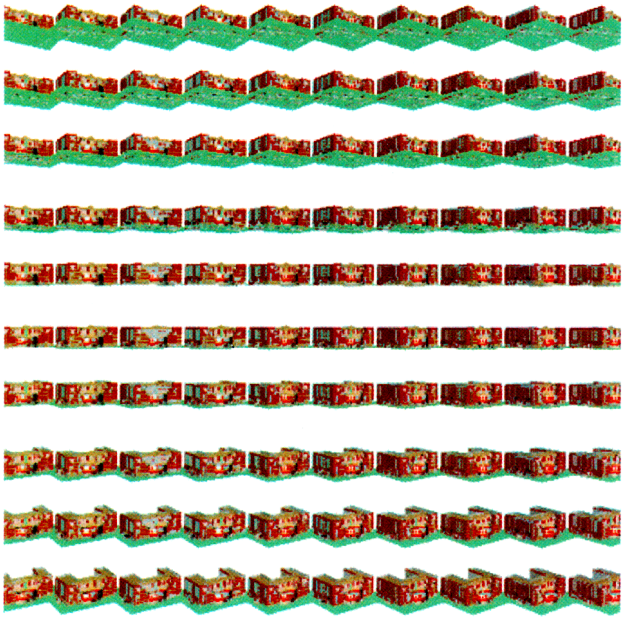
number of cubes	rendering time
100	30 ms
400	90 ms
900	190 ms
1,600	320 ms
2,500	490 ms
3,600	691 ms
4,900	931 ms
6,400	1,202 ms
8,100	1,512 ms
10,000	1,853 ms

```
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.



(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.

Table 5. Response to questionnaire on jGL

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

polygons with 12 different material properties as shown in Fig. 10(a). On the desktop PC shown in Table 2, the run-time performances in all cases were about 2 seconds per frame.

6. Conclusions

This paper has described the implementation of jGL and jVL using pure Java and an experimental effort to port jGL to the i-αppli platform. Our comparisons indicate that although Java 3D uses OpenGL or DirectX as its graphics engine, the performance of jGL is not worse for simple models. 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 a program developed with it. Since we offer jGL on our Web server, many people around the world are using it to convert their previous OpenGL works to Web-enabled versions or to teach and learn computer graphics algorithms. In order to get feedback from users, we sent a questionnaire on jGL to 50 users around the world. The results are shown in Table 5. Most of the users thought that jGL was easy to use and had good reliability.

Although the run-time performance of jVL is not fully satisfactory, because it provides an API for programming, changing the light source and the activity 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 made easier than before by using jGL and jVL. Run-time performance is still the great challenge for jGL and jVL. We expect that its performance will be improved by better Java interpreters and compilers, and will be greatly improved by new Java chips and faster CPUs.

REFERENCES

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

AUTHORS (from left to right)

Bing-Yu Chen received his B.S. and M.S. degrees in computer science and information engineering from the National Taiwan University, Taipei, in 1995 and 1997, and Ph.D. degree in information science from the University of Tokyo 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 Web graphics. He is a member of ACM and IEEE.

Tomoyuki Nishita received his B.S., M.S., and Ph.D. degrees in electrical engineering from Hiroshima University in 1971, 1973, and 1985. He worked for Mazda Motor Corp. from 1973 to 1979. He then became a lecturer at Fukuyama University, and subsequently an associate professor and professor. He moved to the Department of Information Science of the University of Tokyo in 1998 and has been a professor in the Department of Complexity Science and Engineering since 1999. His research interest is mainly computer graphics. He is a member of IEICE, IPSJ, ACM, and IEEE.