JAVAGL[1] - A 3D GRAPHICS LIBRARY IN JAVA FOR INTERNET BROWSERS

Bing-Yu Chen, Chee-Wen Shiah, Tzong-Jer Yang, and Ming Ouhyoung
Communications and Multimedia Laboratory,
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, R.O.C.

ABSTRACT

With the growing popularity of Internet and Virtual Reality (VR), more and more applications, for example, VRML browsers and 3D graphics network multiplayer games, require 3D graphics capabilities over network. This paper presents a 3D graphics library written in Java to satisfy this requirement. Performance evaluation is especially addressed for further studies in developing 3D graphics applications over network. Furthermore, we have also developed a network library and combined it into the 3D graphics library, so that this library has network capability and others can develop 3D graphics network multiplayer applications based on it.

INTRODUCTION

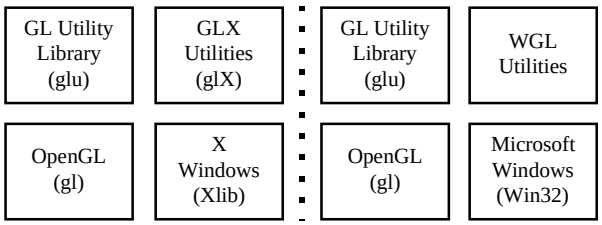
As Internet and VR are getting more and more popular, there is increasing demand of 3D graphics over network. Because Internet itself is a heterogeneous environment, we need to have 3D graphics capabilities in different platforms. Observing the development of Internet, we believe that "pay-per-use" software will be realized in the near future. Under this new paradigm, we may need to distribute applications from servers to clients in different platforms. Hence, 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 is easy to learn, so we define the Application Programming Interface (API) in a manner quite similar to that of OpenGL, since OpenGL is a popular standard.

In order to let JavaGL have network capability, we have also developed a network library in Java that follows the specification of "Distributed Interactive Simulation (DIS)" for network transparency.

IMPLEMENTATION ISSUES

OpenGL's functions can be divided into 3 categories: *GL utility library (glu)*, *OpenGL (gl)*, and *GLX utilities (glX)*, as shown in Figure 1 (a). For Microsoft Windows, there are some differences as shown in Figure 1 (b).



(a) X Windows (b) Microsoft Windows

Figure 1. OpenGL API hierarchy

In Figure 1, *glu* is a set of commonly used graphics routines, and *gl* is the main part of OpenGL, and *glX* or *WGL* utilities are the implementation depending on different platforms. Besides these 3 interfaces, there is an underlying graphics kernel which is transparent to programmers. We follow this principle to develop our JavaGL, as shown in Figure 2.

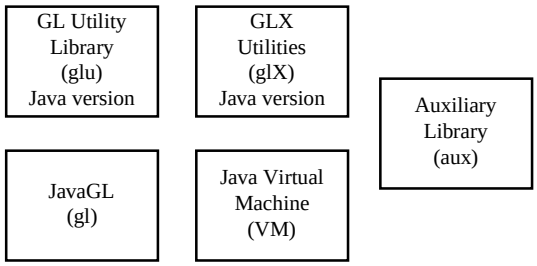


Figure 2. JavaGL API hierarchy.

The implementation is based on the specification of OpenGL [1], and some issues are summarized as follows:

1. For *GL utility library (glu), JavaGL (gl), and GLX utilities (glX)*, we follow the methodologies described in the OpenGL specification. For easy to use purpose, we have also provided the *auxiliary library (aux)* as included in OpenGL.
2. For 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, we categorize all the drawing functions to be several smaller ones to let these functions to be optimized We also refer to Graphics Gems [2] for performance enhancements.
3. Because the API of OpenGL is defined for C language, which is a structured programming language, but Java is an object-oriented programming (OOP) language, we must try to design the API to be Java version.
4. Java has no pointers, which is 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, since there are no pointers, we use class inheritance instead.

When 3D graphics applications based on JavaGL are used on Internet, there is usually interaction between users at the client site and the program at the server site, but lake of interaction between applications. Here we propose to add the network capability to JavaGL. To do this, we follow the specification of DIS [3], which is a standard for interactive simulations in different machines on Internet. To provide the network capability, the 3D graphics applications using JavaGL can send the Protocol Data Unit (PDU) among themselves.

[1] Web site: Http://www.cmlab.csie.ntu.edu.tw/~robin/JavaGL/.

RESULTS

Currently, we have implemented over 160 OpenGL functions in JavaGL, including functions for 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. The additional network capability has also been implemented.

To test JavaGL's capability, we have provided 16 examples on line. The examples are selected from the *OpenGL Programming Guide* [4], which is the official programming guide of OpenGL. All of them can be executed on Internet via the JavaGL web site.

To evaluate JavaGL's performance, we use a test program which renders 12 spheres with different materials, as shown in Figure 3. We execute the test program on both a SUN Ultra-1 workstation and an Intel Pentium-200 PC. We also rewrote the same program in Mesa 3-D graphics library [5], which is a software-based 3D graphics library with an API similar to that of OpenGL using C language, and measure the rendering time, as listed in Table 1.

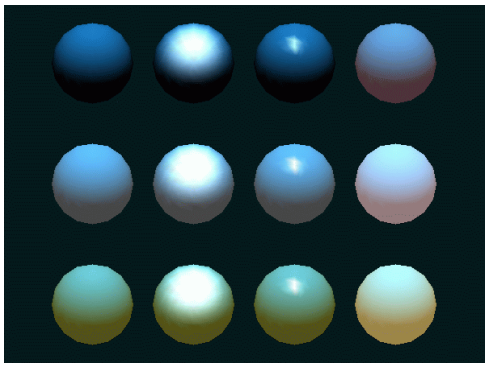


Figure 3. 12 spheres are rendered to measure performance. Each sphere contains 256 polygons. This program is an example in *OpenGL Programming Guide* (code from Listing 6-3, pp. 183-184, Plate 16) [4]. This figure is rendered with JavaGL.

On the SUN workstation, the test program with Mesa is about 4 times faster than JavaGL, as claimed by SUN that Java is about 20 times slower than C [6]. But, the performance can be further improved if a better Java interpreter exists. On the PC platform, we execute the test program using the SUN JDK 1.0.2 and the Symantec Café 1.51 with JIT 2.0. By using the Just-In-Time (JIT) compiler, there is over 4 times performance speedup.

CONCLUSIONS AND FUTURE WORK

JavaGL is being applied to develop our next generation VRML browser running across Internet. 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. JavaGL meets this requirement because it's purely implemented by Java which is designed for Internet.

We will also apply JavaGL to develop 3D graphics network multiplayer games. At present, we are developing the JavaNL [8], a network library in Java, using the DIS protocol, trying to enhance the network capability of JavaGL.

Performance is a great challenge for any Java applications. We expect that the performance will be improved by faster Java interpreters and Java compilers in the software side, and will be greatly improved by the new Java chips in the hardware side.

All the demo codes and examples are put in our web site at "Http://www.cmlab.csie.ntu.edu.tw/~robin/JavaGL", and are welcome to interested readers visit it.

ACKNOWLEDGMENTS

This work is a part of the Multimedia Digital Classroom (MDC) project at National Taiwan University sponsored by National Science Council (NSC) under the grant NSC 85-2622-E-002-015.

REFERENCES

[1] 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.

[2] Andrew S. Glassner, "Graphics Gems," Academic Press, Inc., 1990.

[3] "IEEE Std 1278.1-1995 and 1278.2-1995," IEEE, 1996.

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

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

[6] Arthur van Hoff, Sami Shaio, and Orca Starbuck, "Hooked on Java," Addision-Wesley, 1996.

[7] "Café for Windows 95/NT," Symantec, 1997. Http://cafe.symantec.com/cafe.

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

Graphics Library	JavaGL 1.0beta3	Mesa 2.1	JavaGL 1.0beta3	JavaGL 1.0beta3
Time (ms)	5031	1085	16700	4070
Platform	SUN Ultra-1 Model 170E, 128 MB memory, 24-bit display (Creator 3D). SUN Solaris 2.5.1.		Intel Pentium-200, 64 MB memory, 24-bit display (ET 6000). Microsoft Windows 95.	
Programming Environment	SUN JDK 1.0.2. SUN JIT 1.0.2	GNU C 2.7.2.1	SUN JDK 1.0.2.	Symantec Café 1.51 Symantec JIT 2.0beta3

Table 1. Table of performance comparisons. The test program renders 12 spheres, one sphere contains 256 polygons, as shown in Fig. 3.