JavaGL[1] - A 3D Graphics Library in Java

Bing-Yu Chen, Tzong-Jer Yang, and Ming Ouhyoung

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

Abstract

With the popularity of Internet/Intranet and Virtual Reality (VR), more and more applications, for example, VRML browser, require 3D graphics capabilities over network. In this paper, we presented a 3D graphics library written in Java to fulfill this requirement. The performance evaluation is especially addressed for further studies in developing 3D graphics applications over network.

1. Introduction

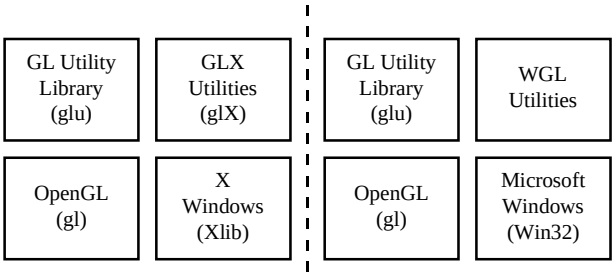
As the Internet/Intranet and virtual reality are getting more and more popular, it comes to demand a 3D graphics capability over network. Because Internet itself is a heterogeneous environment, we need to have 3D graphics capabilities in different platforms. Observing the development of Internet, we believe that "pay-per-use" software will be realized in the near future. Under this new paradigm, we will need to distribute applications from servers to clients in any platforms. Hence, we decide to develop a 3D graphics library that is platform-independent, and Java is chosen for its platform-independent feature.

Furthermore, it is desired that this 3D graphics library is easy to learn, so we define the API, in a manner, quite similar to OpenGL since OpenGL is a popular commercial standard.

The remainder of the paper is organized as follows. In Section 2, implementation issues are described. In Section 3, we apply our 3D graphics library, JavaGL, to render several models with the performance evaluated. We close with conclusions and future work in Section 4.

2. 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 in functions' names, as shown in Figure 1(b).



(a) (b)

Figure 1. (a) OpenGL API hierarchy for X. (b) OpenGL API hierarchy for Microsoft Windows.

Where *glu* is a set of commonly used graphics routines, and *gl* is the main part of OpenGL. *GLX* or *WGL* is the implementation depending on 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.

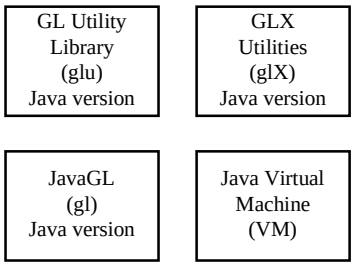


Figure 2. JavaGL API hierarchy.

The implementation is mainly 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 and the implementation in Mesa 3-D graphics library [2].

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

2. For the underlying graphics kernel, we put most of our efforts here. For instance, a drawing command in OpenGL may be executed immediately or be postponed in a display list, depending on the state of the display list. In C, we can simply use function pointers to solve this problem. In Java, since there are no pointers, we use class inheritance instead. We also refer to Graphics Gems [3, 4, 5] for performance enhancements.

3. Results

Currently, we have implemented over 150 functions in JavaGL, including functions for 3D model transformation, 3D object projection, depth buffer, smooth shading, lighting, material, and display list. The functions not yet supported so far are mainly for anti-aliasing and texture mapping.

To evaluate JavaGL's performance, we use a test program which renders 12 spheres with different materials assigned, as shown in Figure 3. We execute the test program on both a SUN Sparc20 and a Pentium-100 PC. We also rewrote the same test program in Mesa and OpenGL 1.1 for Microsoft [6], and measure the rendering time, as listed in Table 1.

On the SUN Sparc20, the test program with Mesa is about 20 times faster than JavaGL, as claimed by SUN that Java is about 20 times slower than C [8]. This can be improved greatly if a better Java interpreter exists. Table 2 shows the effects of Java compilers and interpreters. In the same configuration, if one replaces SUN JDK's class library with the one optimized by Symantec Café's compiler [9], there is no noticeable improvement; if one replaces SUN JDK's interpreter with Symantec Café's interpreter that supports Just-In-Time compiler (JIT), there is over 3 times performance speedup.

On the Pentium-100 PC, because Mesa for Windows 95 is not yet available, we compare JavaGL with OpenGL 1.1 for Microsoft. Though we have used Symantec Café's compiler and interpreter, OpenGL 1.1 for Microsoft is still 16 times faster than JavaGL. According to Microsoft's announcement [6], lots of optimizations have been done in OpenGL 1.1 for Microsoft.

Figure 4 is the teapot rendered with JavaGL and Mesa respectively.

4. Conclusions and Future work

The 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.

Performance is a great challenge for any Java applications. In the future, we will focus on the performance improvement by replacing some algorithms with faster ones. We also expect that the performance will be improved by faster Java interpreters and Java compilers.

Javasoft also has plans to develop a 3D graphics API, Java3D. According to the schedule, an early release will be announced in 1997. We will also notice Java3D's development and shift our strategy to meet the trend.

Acknowledgements

This work is a part of the Multimedia Digital Classroom (MDC) project 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] Brian Paul, "The Mesa 3-D Graphics Library." Http://www.ssec.wisc.edu/~brianp/Mesa.html.

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

[4] James Arvo, "Graphics Gems II," Academic Press, Inc., 1991.

[5] David Kirk, "Graphics Gems III," Academic Press, Inc., 1992.

[6] "Sample: OpenGL 1.1 Release Notes & Components," Micrrosoft, 1996. Http://www.microsoft.com/kb/articles/q154/8/77.htm.

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

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

[9] "Café for Windows 95/NT," Symantec, 1996. Http://cafe.symantec.com/cafe/.

	JavaGL 1.0	Mesa 1.2.6	JavaGL 1.0	OpenGL 1.1 for Microsoft
Time (ms)	24391	1681	8240	500
Platform	SUN Sparc20 Model 71, 64 MB memory, 24-bit display (ZX). Solaris 2.5.		Intel Pentium 100, 32 MB memory, 24-bit display (ET 4000/W32p). Windows 95.	
Programming environment	Symantec Café 1.5 Java compiler, SUN JDK 1.0.2 Java interpreter.	GNU C 2.7.2.1	Symantec Café 1.5 Java compiler & interpreter	MS-Visual C 4.2

Table 1. Table of performance comparisons. The test program renders 12 spheres, as shown in Figure 3.

	SUN JDK classes + SUN JDK interpreter	Café classes + SUN JDK interpreter	SUN JDK classes + Café interpreter	Café classes + Café interpreter
Time (ms)	29490	29770	8680	8780

Table 2. The effects of Java compilers and Java interpreters. SUN JDK classes are not optimized while Café classes are optimized by Café's compiler. Café's interpreter supports Just-In-Time compiler (JIT) and performs better than SUN JDK's interpreter. These data are measured under the same PC configuration used in Table 1.

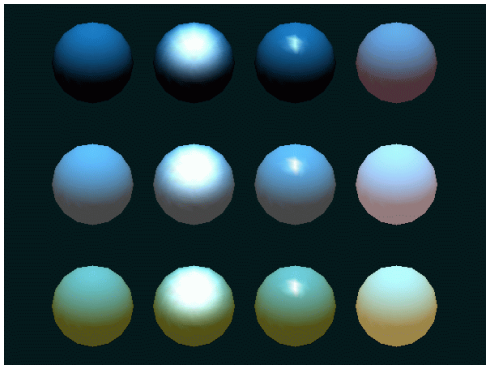
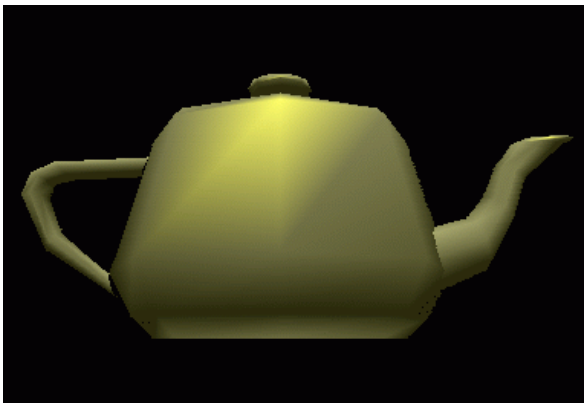
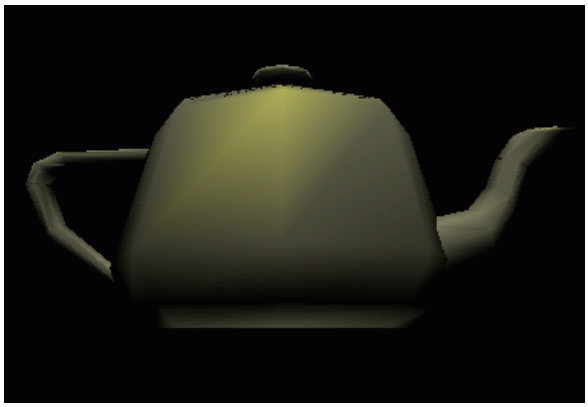


Figure 3. Twelve spheres are rendered to measure performances. Each sphere contains 256 polygons. This program is an example in *OpenGL Programming Guide* (code from Listing 6-3, pp. 183-184, Plate16) [7]. This figure is rendered with JavaGL.



(a)



(b)

Figure 4. The teapot rendered (a) with JavaGL, and (b) with Mesa 3-D graphics library. This teapot contains 604 triangles and takes 7.34 sec for JavaGL on the same SUN Sparc20 in Table 1.