Hardware Accelerated Graphics in Java

David McDonald and Robert F. Erbacher

Computer Science Dept., Utah State University, Logan, UT
david.mcdonald@ihc.com and Robert.Erbacher@usu.edu

Abstract

This paper examines the feasibility of Java as a development platform for high performance 3D graphics applications. With the ubiquity of java and its ease of development it must be considered as a potential development platform. However, traditionally java has been considered too slow for real 3D graphics. With recent advances in Java performance, the implementation of the OpenGL architecture within java (JOGL), and the direct linkages from the java virtual machine to the underlying graphics drivers the performance of Java must be fully evaluated in terms of its graphical performance capability. In this paper we examine both the JOGL API as well as the Java3D API.

Keywords: Performance Evaluation, 3D Graphics, Java

1. Introduction

The popularity of the Java programming language has exploded in the last five years. Java has found its way into a wide variety of markets, and developers world wide continue to find new application areas that are ideal for the language. Although much of the focus has been on building custom enterprise business software, there is a growing community that is interested in using Java to build sophisticated 3D applications. Such applications could be very useful in a wide variety of industries, particularly those that may have a need to distribute a 3D application over a wide area network or even the internet using applet technology. For organizations building 3D desktop applications that are targeted for multiple platforms, the cross-platform nature of Java could very well be the cost saving solution they have been searching for.

The question at this point is whether applications written in Java can really perform at the level required for realistic modeling of complex 3D environments. In an attempt to achieve this required level of performance, several Java APIs have been developed which use native libraries to allow for hardware acceleration of graphics.

This report focuses on two of the more popular APIs available for building 3D applications in Java: Java3D, and Java bindings for OpenGL (more commonly referred to as JOGL). Java3D and JOGL both provide the ability to write 3D applications that are hardware accelerated. Both APIs provide Java classes that are packaged into JAR files, along with native libraries that are used for hardware acceleration. Java3D is available with two different sets of native libraries: one for use with OpenGL drivers, and another for use with DirectX drivers.

The goal of this report is to understand how well these APIs really perform, and whether it is realistic to expect that a complex graphical application can be written in an interpreted language. While the programs written as part of this study are relatively simple, they demonstrate features that are commonly used in most 3D applications and are a good basis for performance testing.

2. The JOGL API

JOGL is a relatively new initiative from Sun Microsystems and SGI that has emerged within just the last couple of years [1]. JOGL is basically a straight port of OpenGL. All of the standard functions are available from both the GL and GLU libraries. This allows programmers who are familiar with OpenGL to quickly become accustomed to programming 3D applications in Java. JOGL is very easy to set up and start programming with. Installation is a snap by simply copying two DLL files and a JAR file into the appropriate Java directories.

The following code fragment is the display method of a simple Java application that uses JOGL to display a rotating white square against a blue background.

```
public void display(GLDrawable glDrawable) {
   GL gl = glDrawable.getGL();

   gl.glRotatef(ANGLE, 0, 0, 1);

   gl.glClear(GL.GL_COLOR_BUFFER_BIT);

   gl.glBegin(GL.GL_QUADS);
   gl.glVertex3f(-70, 70, 0);
   gl.glVertex3f(70, 70, 0);
   gl.glVertex3f(70, -70, 0);
   gl.glVertex3f(-70, -70, 0);
   gl.glEnd();
 }
```

```
private BranchGroup createScene() {
   BranchGroup scene = new BranchGroup();

   TransformGroup spin = new TransformGroup();
   spin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
   scene.addChild(spin);

   spin.addChild(new ColorCube(CUBE_SIZE));

   Alpha rotation = new Alpha(-1, 4000);

   RotationInterpolator rotator = new RotationInterpolator(rotation, spin);

   BoundingSphere bounds = new BoundingSphere();
   rotator.setSchedulingBounds(bounds);
   spin.addChild(rotator);

   scene.compile();
   return scene;
 }
}
```

Figure 1: Java3D sample code fragment.

With the exception of the first line in this method that is used to get access to the GL object, the method calls should all look familiar to those who have previously worked with OpenGL. Of note is the use of 'gl.' as a prefix to all OpenGL commands, i.e., gl is the instantiation of the OpenGL super class containing all of the OpenGL methods and variables.

3. The Java3D API

Traditionally, 3D programming has been very procedural. Much of the code base today is written in C, potentially with some inline assembly instructions. Java3D is an attempt to create a new, fully object-oriented paradigm for 3D programming. Even more, Java3D attempts to provide a complete framework for creating a 3D virtual universe. Everything in Java3D is centered on building a "scene graph", which is a tree data structure designed to couple together physical objects in the 3D world with their behaviors.

The object-oriented framework approach that Java3D takes means that the programmer is working with a much more abstracted API than with an API such as JOGL.

The concept of the "scene graph" itself takes some getting used to. There is a definitive set of rules about how to construct a legal scene graph which must be adhered to in order for the program to run. Even after reading through the documentation and understanding these rules, it still takes a fair amount of experimentation before one is able to append the various behaviors and objects into a scene graph that are required to produce the desired result.

The code fragment in figure 1 is a portion of a Java application that uses Java3D to display a cube rotating on the Y axis. The difference in styles between JOGL and java3D are clear.

4. The API Debate

There has been a fair amount of debate as to which API should be used. Sun has thrown more of its weight behind the JOGL initiative recently, which has left some wondering about the fate of Java3D. However, at the JavaOne 2004 conference, Sun announced that Java3D would be open sourced. This should insure that both APIs will be available (and improved) for the foreseeable future. Continuing both projects will create the competition and choice that ultimately fuels better technology.

5. Testing Strategy

In order to compare these APIs as fairly as possible, seven pairs of programs were developed (one using JOGL and one using Java3D) which were very close in features and behaviors. Benchmarking was then performed against each implementation, and the results recorded. The metric used for all tests was frames per second.

5.1. Development Tools

All of the Java programs were written and compiled using the Java 2 Software Development Kit version 1.4.2_05, available from Sun Microsystems (http://java.sun.com/j2se/1.4.2).

Programs written using JOGL were tested using JOGL version 1.1b05, available from Sun Microsystems (https://JOGL.dev.java.net).

Programs written using Java3D were tested using Java3D version 1.3.1, available from Sun Microsystems (http://java.sun.com/products/java-media/3D). For each program written using Java3D,

	Rotate Cube	Rotate Sphere A	Rotate Sphere B	Rotate Sphere C	Stress Test A	Stress Test B
Java3D OpenGL	1466	671	672	681	198	176
Java3D DirectX	2006	783	768	786	179	169
JOGL	1678	728	452	455	*	*
DirectX Improvement	37%	17%	14%	15%	**-10%**	**-4%**
JOGL Improvement	15%	9%	**-33%**	**-33%**	*	*

Table 1: Summary of results generated by the six performance tests.
*The performance of JOGL in these experiments was too poor too compare effectively.

both the OpenGL implementation and the DirectX implementation were tested.

5.2. Testing Tools

Benchmarking of the programs was performed using Fraps, which shows an on screen display of the frames per second that any hardware accelerated graphics application is producing. Fraps is available from http://www.fraps.com.

5.3. Configuration

All programs were tested on a Dell Inspiron XPS laptop running Windows XP with the following hardware configuration:

CPU: 3.4 GHz
RAM: 1 GB
Video: ATI Mobility 9800 256 MB

All results are shown in the relevant columns of table 1. The rows relate the different implementation modes: Java3D-OpenGL, Java3D-DirectX, and JOGL. The final two rows show the percentage improvement using the Java3D-OpenGL results as a basis for comparison.

5.4. Test #1: Rotating Cube

The first pair of applications displays a rotating cube. More specifically, the application displays a cube in which each side of the cube is a different color, and rotates at a constant rate that is independent of the frames per second.

Although this is a very simple application, the frames per second are outstanding. In this test, JOGL performed 15% better than Java3D with OpenGL. However, Java3D with DirectX was the real winner here; it performed 20% better than JOGL, and 37% better than Java3D with OpenGL. These results are exemplified in the first column of table 1.

5.5. Test #2: Rotating Sphere A

This pair of applications each displays a rotating sphere that is defined using 10,000 polygons. This application pair displays a solid sphere that is rotating at a constant rate. The performance results for this experiment are shown in the second column of table 1. The results in this test are closer than those in Test #1 but the differences are still significant.

5.6. Test #3: Rotating Sphere B

This pair of applications is the same as the "Sphere A" set but has generated normals and material attributes applied for specular, diffuse, and ambient lighting. A light source is added that is directed straight down the Z axis.

In this test, both versions of Java3D out performed the JOGL application. We suspect that the Java3D framework is imposing some automatic optimizations to keep performance high that the JOGL implementation does not incorporate. These results are related in column 3 of table 1.

5.7. Test #4: Rotating Sphere C

This pair of applications is the same as the "Sphere B" set but adds texture mapping to the sphere. The texture map is a 512 × 512 linear image of the earth. The light source is moved to the right side to produce a different lighting angle. The results for this experiment are nearly identical to that of test #3 both in terms of raw results and percentage improvement from the base.

Of particular interest is that all of the results for Test #4 actually achieved higher results than Test #3

(columns 3 and 4 of table 1) when the expectation would be for performance to decrease because of the additional properties added to the object. However, this improved result is not statistically significant, amounting to approximately a 1% improvement and is in part a random variation in the performance results. The main conclusion from this improvement in performance is that the number of polygons incorporated into the scene is the limiting factor. The added load of the texture mapping has no significant impact since it is likely handled by a separate functional unit of the Graphical Processing Unit (GPU).

5.8. Test #5: Stress Test A

These pair of applications attempts to put more load on the system and effectively reduce the frames per second. This environment again is based on displaying a number of high resolution spheres. All of the spheres are texture mapped with the same image, and the light source is the same as that in the "Sphere C" set.

JOGL performed particularly ineffective in this experiment. The JOGL version of this application was limited to displaying a total of 17 spheres while the Java3D version displays a total of 32 spheres. Even with this advantage JOGL was only able to achieve 39 frames per second. With the rate of decrease in performance with increasing numbers of spheres, the frame rate using JOGL was reduced below an effective rate of measurement for use with Fraps.

Java3D, however, seemed to have some built in optimizations that allowed the continued addition of spheres without impacting performance significantly. After adding 32 spheres, the visible screen space was almost completely occupied and adding non-visible spheres did not decrease performance further. The results are listed in column five of table 1.

This stress test shows an extreme difference in performance between the Java3D and JOGL implementations. One can only assume that Java3D is performing significant optimizations. It is interesting to note that in stress testing, Java3D with OpenGL finally performed better than Java3D with DirectX by 10%. Considering that all other tests showed Java3D with DirectX out performing Java3D with OpenGL by approximately 15% this is quite a significant turn-around and shows the OpenGL drivers advantages for very complex scenes.

5.9. Test #6: Stress Test B

This pair of applications is the same as the "Stress Test A" set except that each sphere is texture mapped with one of five different textures. Additionally, the JOGL application was reduced from 17 spheres to just 7, since performance dropped below a realistically measurable level with anything above this.

These results are very similar to those in Test #5. The introduction of additional textures decreased performance across the board. Compared with the results of Test #5, the reduction in performance in Java3D with OpenGL was 11%, while Java3D with DirectX performance was reduced by 6%. JOGL again performs too poorly to compare effectively only providing 29 frames per second with measly seven spheres.

6. Conclusion

Both of these APIs exhibited excellent performance overall. The automatic optimizations being performed by the Java3D framework gave it much better performance than JOGL in many of the tests. This advantage partly involves the fact that JOGL is a new environment that still requires enhancement. Given the number of polygons in the spheres a robust implementation could easily improve performance through application of effective minimization algorithms.

The abstraction in Java3D can be very powerful. It is certainly convenient to be able to create a SimpleUniverse object and start adding other objects and behaviors to it. The abstraction also allows a single API to be used with either OpenGL drivers or DirectX drivers. However, there are tradeoffs to the extra abstraction. For example, it is much more difficult to get at the details of what the framework is doing and have the fine-grained control that is sometimes needed in 3D programs.

By comparison, the JOGL API has the most flexibility (since it is just a basic port of OpenGL), but the programmer is forced to code each and every optimization required to get their application to perform at an acceptable level. Additionally, JOGL has the advantage of providing a familiar environment to many graphics programmers.

Ultimately each API has its place, depending on the requirements of the application that is being developed. Java3D could speed the development of applications that can be done at a more abstract level, while other programs that require more low level control would probably benefit most from JOGL.

The other conclusion one must come to is that choice is the most important aspect of all. In most of the tests, the DirectX version of Java3D out-performed both of the OpenGL implementations.

7. References

[1] SGI and Sun Microsystems, *SGI and Sun Microsystems Software Platforms to Work Seamlessly Together with Java Bindings to OpenGL*, Press Release, 2003. (http://www.sun.com/smi/Press/sunflash/2003-07/sunflash.20030728.1.html)

[2] James Keogh, *J2EE: The complete Reference*, McGraw-Hill Osborne Media, 2002.

[3] Gene Davis, *Learning Java Bindings For Open Gl JOGL*, Authorhouse, 2004.

[4] Henry Sowizral, Kevin Rushforth, Michael Deering, *The Java 3D(TM) API Specification*, Addison-Wesley Pub Co., 2000.

[5] Fraps (http://www.fraps.com)