MSIM 641 F06 Visualization I

Course Project

A Study to Investigate the Feasibility of Using Java3D and Rigid Body Dynamics in Games and Simulations

Prepared for

Dr. Yuzhong Shen

by

Randy Brooks

Assigned on

October 4, 2006

Final Submission
Due
December 12, 2006 at 11:59 PM

E-mail address:
randybrooks@charter.net

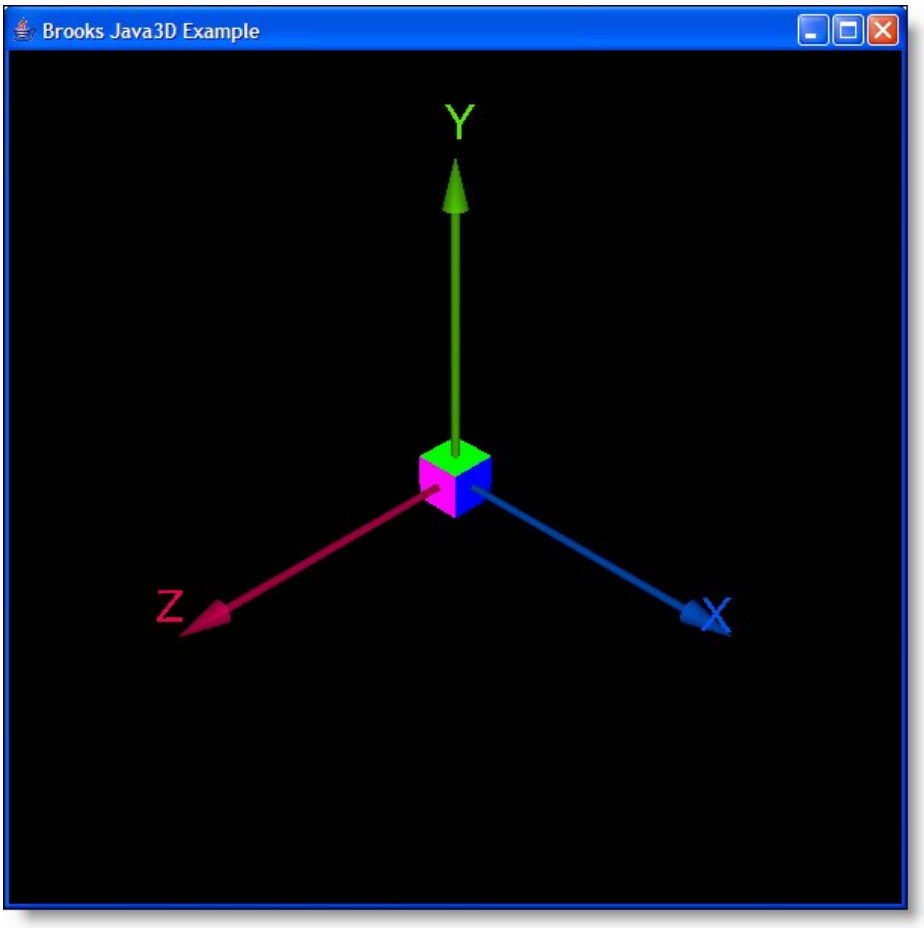
Phone number:
(757) 688-9739 - Work
(757) 365-9652 - Residence

Site location:
VMASC

Table of Contents

Executive Brief

The feasibility of using Java3D and rigid body dynamics (Odejava) in serious games and simulations is explored in this study. The combined power of Java3D's rich visualization environment and a powerful physics engine can produce visually pleasing and entertaining results. Shown below is a simple Java3D program that was written to understand the structure and richness of the API. The Java3D API is described along with examples of manipulating objects in 3D space using Odejava. All programs are installed and run from a Java development environment. Software features, licensing and history of Java3d are described in some detail. The advantage of using a scene graph approach is discussed. The summary details the challenges and lesson learned during the course of this study.



Java3D example of Axis class and Cube class

1 Introduction

Java3D is a high level visualization API used with the Java programming language. Java 3D is a standard extension to the Java 2 JDK. Open Dynamics Engine (ODE) is an open source physics engine that is written in C. Odejava wraps ODE native functions into Java function calls. Odejava is an industrial quality library for simulating articulated rigid body dynamics.[1]

The Java 3D API is an interface for writing programs to display and interact with three-dimensional graphics. Whenever API functionality is ported to a different programming environment the speed of execution and the completeness of the port are concerns to the software developer. Java3D techniques used to increase performance are discussed.

Andrew Davison's book[2], "Killer Game Programming in Java", provided the two example programs used for this report.

2 Overview

2.1 Version History

Intel, Silicon Graphics, Apple, and Sun collaborated in making Java3D. A public beta version was released in March 1998 and a first version released version December 1998. The project died from mid-2003 through summer 2004 as development of Java 3D was discontinued.

Sun and volunteers have since been continuing its development since 2004 when Java 3D was released as a community source project. The current version is 1.4.0 and was released in February 2006. Version 1.5.0 is currently issued as a beta version release candidate. The community appears active and further support seems promising as version 1.6 is in the planning stages. Since the project was unsupported in 2003 similar Java3D implementations emerged, such as, JOGL, Xith3D, and others. A check of the Sun web site indicates that the official version 1.5 will be released this month.

2.2 Java3D Architecture

A Java 3D program creates instances of Java 3D objects and places them into a scene graph data structure. The scene graph is an arrangement of 3D objects in a tree structure that completely specifies the content of a virtual universe and how it is rendered.

A scene graph structure allows the software developer to easily manipulate the smallest object in a scene to the entire collection of objects. The details of rendering are handled automatically. There is a DirectX and OpenGL implementation of the software which can be changed by passing an argument to Java in the windows environment. The default implementation is OpenGL.

A graph is a data structure composed of nodes and edges. A node is a data element, and edge is a relationship between data elements. The nodes in the scene graph are the instances of Java 3D classes. The edges represent the two kinds of relationships between the Java 3D instances. The most common relationship is a parent-child relationship. A group node can have any number of children but only one parent. A leaf node can have one parent and no children. The other relationship is a reference. A reference associates a Node Component object with a scene graph Node. Node Component objects define the geometry and appearance attributes used to render the visual objects.[3]

Every path in a Java 3D scene graph completely specifies the state information of its leaf. State information includes the location, orientation, and size of a visual object. Consequently, the visual attributes of each visual object depend only on its path within the scene graph. This allows for easy manipulation of the entire scene or part of it. The following diagram show how the nodes are often drawn to communicate the graph structure. Lines with arrows are used to show the

edges between the nodes. The higher level relationship is much easier to use rather than direct manipulation of OpenGL or DirectX. Figure 1 shows a typical representation of nodes in Java3D.

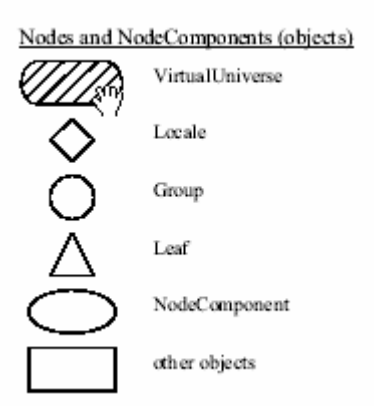


Figure 1.

2.3 Software Features

Java3D supports a multithreaded scene graph structure that is platform independent since it is based in Java. The generic real-time API is stable and performs well for visualization. It has support for retained, compiled-retained, and immediate mode rendering to enhance performance when required. It also has native support for head-mounted displays.

There is a 3D spatial sound feature to enhance the immersion sensation when game playing. There are programmable shaders that support both OpenGL Shading Language (GLSL) and C for Graphics (CG).

A Stencil buffer is supported along with the color and depth buffer (z buffer). The most typical application of the stencil buffer is to add shadows to 3D applications.

There are various file importers for most mainstream 3D formats, like 3DS, OBJ, VRML, X3D, NWN, and FLT. These file loaders are not an integral part of the API. They are widely available through third parties.

2.4 Scene Graph Approach

There are several advantages of a scene graph approach. Object management is provided by the scene graph itself since it is a data structure. Rendering optimization is available though the use of the bounds that limit what is rendered. Behavior models allow for rotation, translations, and scaling. Collision detection in provided between objects in the scene graph. Java3D is multiple thread aware and makes use of multiple processing threads. Picking support is automatically supported by the scene graph. Hierarchical control permits complex compound structures to be developed and translated, rotated, or/and scaled with a simple transformation.[4]

2.5 Platform Support

The following platforms are fully supported.

- Windows (x86)
- Linux (x86)
- Linux (AMD64)
- Solaris (SPARC)
- Solaris (x86)

The Linux (Power PC) and Linux (IA64 Itanium) following platforms can be built from the Ant build.xml file, but are not supplied as pre-built downloads. Mac OS X 10.3.1 or later is not officially supported, but Java3D has been ported to it by the manufacturer. According to Apple, 10.4 (Tiger) is not supported, but user reports suggest that Java3D works fine on 10.4 with no problems.

2.6 Installation

Installation requires that the following jar files are located in the classpath.

- j3d-core.jar
- j3d-core-utils.jar
- vecmath.jar

Specific installation instructions are included in the readme.txt file included with the zipped installation files.

Odejava requires that the odejava.dll is placed put in the windows/system32 sub-directory. There are two jars needed to run the example programs.

- log4j-1.2.11.jar
- odejava.jar

2.7 License

There are two licenses required for Java3D. The source code for the j3d-core-utils is licensed under the open source Berkeley Software Distribution (BSD) License.

The source code for the j3d-core and vecmath projects is licensed under the Java Research License (JRL) for non-commercial use. The JRL allows users to download, build, and modify the source code in the j3d-core and vecmath projects for research use, subject to the terms of the license. The source for the j3d-core and vecmath projects is also licensed for commercial use under a no-fee Java Distribution License (JDL).

ODE stands for Open Dynamics Engine. It is an open source (BSD/LGPL) rigid body physics library written in C. The log4j-1.2.11.jar is part of the Apache Project. Odejava is released under the BSD License.

3 Example Programs

3.1 HelloWorld Example

The HelloWorld class is an Java3D program that uses a scene graph to contain the objects that are displayed. Figure 2 is a high level graphic example of the HelloWorld program.

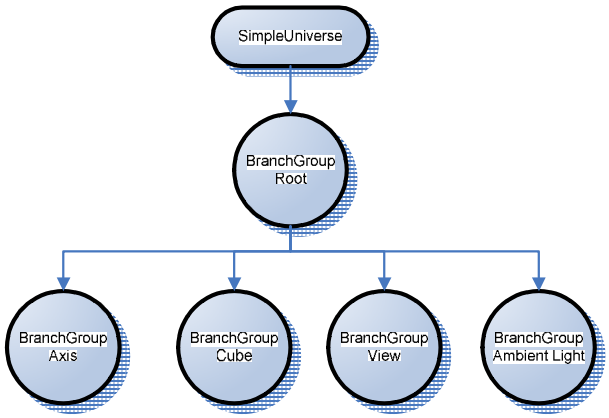


Figure 2.

3.2 Program Flow

The Axis class that was developed demonstrates several uses of transformations and transforms to positions the axis. HelloWorld creates an instance of the Axis class. Figure 3 is a graphic representation of the code that was developed for the Axis class. The X axis is shown in detail. Figure 4 shows a screen print from the HelloWorld program.

When the HelloWorld class main function is called a new instance of HelloWorld is created. The window is initialized and then Java3D is initialized including the axis and a cube at the origin. Lighting is enabled and mouse movement is captured by the obit behavior in the scene graph.

The axes are cylinders with cones attached. Since a cylinder's default position is pointing up the Y-axis when created, a TransformGroup is needed to move the cylinder into the correct position. A TransformGroup is needed for each axis. Material properties are enabled and color is applied to the appearance. Each axis has a 3D extruded letter next to the positive direction. The Java3D 'lookat' function is set to 35, 35, and 35 in the x, y, and z directions. The mouse wheel controls zooming and the left button moves the axes.

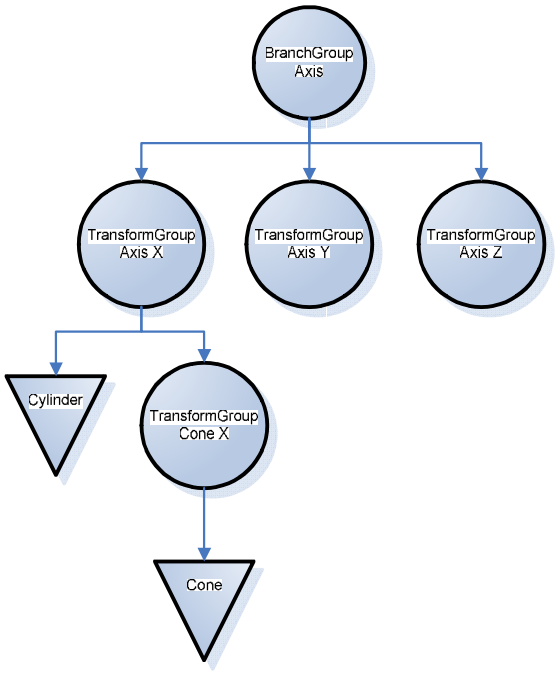


Figure 3.

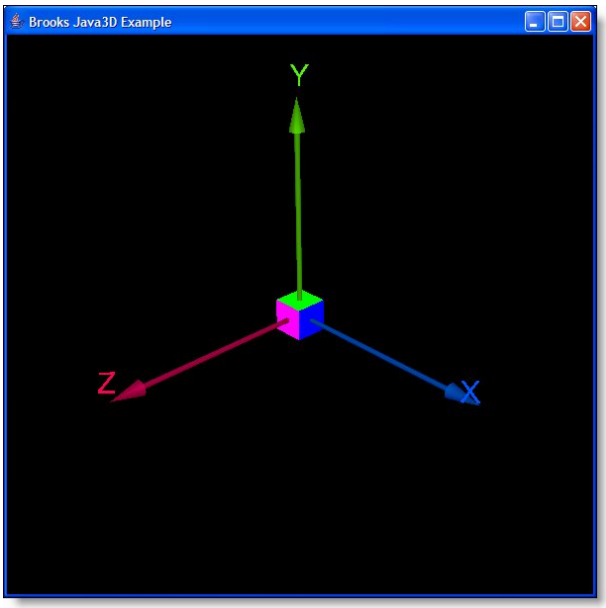


Figure 4.

3.3 Bounding Balls

This bounding ball program written by Andrew Davison is an example of the power of Java3D. Several textured spheres are created and allowed to react to the 3D environment based. The spheres spin independently and have gravity and friction properties. They bounce against each other and the bounding walls. A special behavior is created to allow the spheres to interact with the environment. Odejava handles the calculations when one sphere is in contact with another. Performance seems good with about 10 to 15 spheres using standard windows hardware.

Figure 5 shows the screen during execution of the program. A short video capture is included with the delivered report showing the movement and interactions.

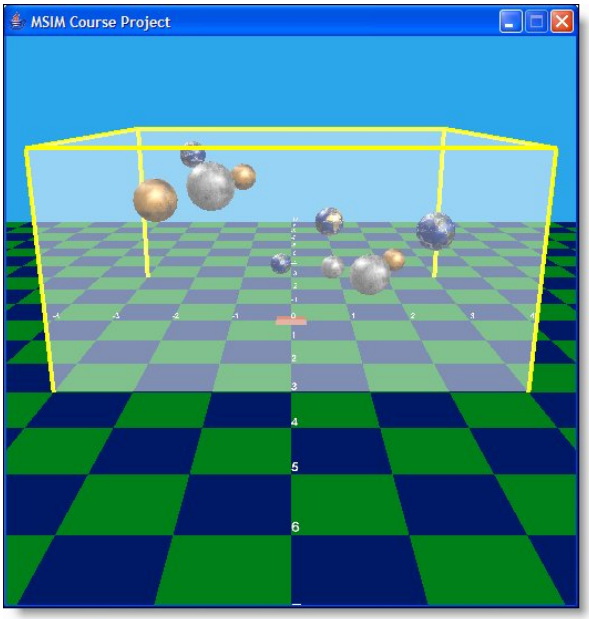


Figure 5.

3.4 Bird Flocking

Another program example used to study the feasibility of Java3D and Odejava is the bird flocking program written by Andrew Davison. He creates flocks of bird objects and there is a predator and prey relationship between them that slowly decreases the bird population. It uses the same Java3D environment as shown in the last example. This is a good example of the power of the combination of using Java3D

for visualization and Odejava as the rigid dynamic engine. It produces an environment showing movement that appears to be flocks of birds flying through 3D space.

Figure 6.

4 Future Growth

Java3D is a well supported and active community that appears to have a robust future. The Sun tutorial, although dated has great examples of how to use the API. A roadmap for Java3D at Sun's web site indicates that many more features will be implemented. Features, such as, OpenGL's auto mip-map generation is planned for version 1.6. As more features are added without the underlining complexity of OpenGL or DirectX I believe that the user community will continue to support Java3D. Another indication that Java3D is actively supported is the number of RSS list servers keeping contact with subscribers.

Unfortunately, Odejava has recently become unsupported. I was quite surprised to find that the main hosting web site http://odejava.org is not currently available due to message that indicates excessive bandwidth usage. My research indicates that this message has appeared since September 2006. Sun Microsystems does host the source code and the binary download for all platforms, although the dates are few years old and it does not appear very active. This means that newer features in Java 1.5.0, such as, the use of generics types will not be supported in Odejava.

5 Summary

This report is based on three technical subjects that were studied. First, the Java3D API was studied in detail. Secondly, Odejava examples were explored to gain a better understanding of its potential use in games and simulations. Thirdly, a small software application was developed to demonstrate knowledge of using scene graph architecture. The majority of the effort on this project was spent on software development and learning Java3D well enough to produce a simple application.

Learning Java3D is made difficult due to the lack of quality published books on the subject. The reference section includes a few of the most popular books. There are several good web tutorials but the levels of complexity vary greatly. A lot of time can be wasted in searching for good code examples when implementing unfamiliar features of the API.

Many web sites and books have references to 3D file loaders and other information that are no longer supported or available. This can be very frustrating when researching the current levels of support in the user community.

One lesson learned from the experience of writing a Java3D program is to fully understand the scene graph structure before starting to code. There can be many branch groups and transformation groups all to manipulate the objects in a scene.

A good diagram needs to be created at the beginning of the exercise. It is very easy to lose the context of where one is in the scene graph structure. If you add child node to the

wrongly intended branch, strange behavior can follow. Java3D will not produce the intended results and there will be no error.

At times during this study, I was developing code for both the C++ OpenGL projects and for the Java Axis class. I found that I wanted to perform pushMatrix and popMatrix operations in Java3D. This is an example of the need to understand the context of where you are in the scene graph. Since in Java3D you do not have access to the renderer function (GLUT display callback function) you must change your thinking perspective.

In Java3D, the programmer must think more like an artist who is creating a scene of objects and wiring them together in some logical order. Furthermore, objects like lighting and viewing can be artistically added and changed in the same way as other object in the scene.

The mix of licenses for Java3D is very confusing. I am hopeful that Sun and the development community will release Java3D under the GNU (General Public License) as they did recently with Java SE.

Java3D and Odejava are very reliable and feasible tools for use in game creation and simulation visualization. It was seen many times that very few lines of code could produce dramatic changes in the scene. During the course of software development it was clear that Java3D has the power to product visually pleasing and rich graphical environments.

Appendix

References

1. Odejava web site. https://odejava.dev.java.net/

2. Killer Game Programming in Java, Andrew Davison, O'reilly Press, May 2005

3. Java3D sun tutorial. http://java.sun.com/products/java-edia/3D/collateral/j3d_tutorial_ch1.pdf

4. Java 3D Programming, Daniel Selman, Manning Publication, 2002. Page 51.

5. 3D API Jump-Start, Aaron E. Walsh, Doug Gehringer, Sun Microsystems Press, 2002

Development Environment

- Eclipse IDE 3.2
- Java 5.0.8
- Java3D 1.4.1
- Odejava 2004.05.30

Deliverables Included

- Java jar files to run HelloWorld program
- Short Video AVI screen captures
- Project proposal and this report
- Java source code