Games Programming with Java and Java 3D

Andrew Davison
Dept. of Computer Engineering
Prince of Songkla University
HatYai, Songkhla 90112
E-mail: dandrew@ratree.psu.ac.th

Draft: 14th January 2003, #2

Abstract

This article looks at the advantages and disadvantages of using Java and Java 3D for games programming. It assumes the reader is familiar with Java, but presents short overviews of gaming, the low-level APIs OpenGL and DirectX, and Java 3D. No programming examples are included here, although links to online code are supplied.

1. Background to Gaming

Giving a definition for 'computer game' is problematic, due to the wide range of game types. For example, the ArcadePod site (http://www.arcadePod.com) divides its hundreds of Java games into more than ten categories: 3D games, multiplayer, action, classic, indoor sports, board, outdoor sports, card, mind, casino, educational, and the useful 'miscellaneous' catch-all. This makes it difficult to pin down the typical content of a game, and highlights the range of design and programming skills required to create one.

Another problem with giving a definition is that game platforms vary enormously, including PCs, dedicated game consoles (e.g. the Sony PlayStation 2), arcade machines, web-based games, hand-held consoles (e.g. the Nintendo Gameboy), interactive TV, set-top boxes (as supplied by some cable networks), cellular phones, and PDAs.

1.1. Revenue

PCs and game consoles account for almost all the income from games – about 1/3 from PCs (dominated by Windows), and most of the rest from three games consoles (Sony's PlayStation 2, Microsoft's Xbox, and Nintendo's GameCube) [Veronis Suhler Media 2000].

In October 2002, consulting firm *Strategy Analytics* (http://www.strategyAnalytics.com) predicted game console shipments to top 41.9 million units in 2002, an increase of 84% over the 2001 level. The PlayStation 2 is expected to account for 63% of sales, followed by the GameCube with 21% and the Xbox with 16%. These numbers highlight the domination of the PlayStation 2 which has accounted for 72% of global cumulative shipments, compared to 16% for the GameCube and 12% for the Xbox.

NPDFunworld (http://www.npdfunworld.com/) reported that console game sales in 2002 rose by about 20% over the record US$9.4 billion in 2001. Individual game sales are also increasing: *Grand Theft Auto: Vice City* from *Take Two Interactive* sold

an estimated 3 million copies in its first month of release. This game is likely to become the best selling title of 2002, and of all time.

1.2. Console Hardware

Since consoles are so central to gaming, it is interesting to consider their capabilities.

The PlayStation 2 processor runs with a clock speed of 294 MHz, has 32Mb of RAM, and a separate graphics chip that can render about 66 million polygons per second. The small amount of RAM is an important constraint upon games wishing to use Java.

The introduction of the Xbox changed the game rules (so to speak), with its Pentium III, 64Mb RAM, a 8Gb hard disk, and the ability to render 150-200 million polygons per second.

Sony and Microsoft have recently released network adapters for their consoles, highlighting the growing importance of multiplayer games.

Future console designs (e.g. the PlayStation 3 and Xbox 2) will bring PCs and consoles even closer, and further emphasize online gaming. The PlayStation 3 (slated to appear at the end of 2004) may use a 3GHz processor, 512Mb RAM, a 120Gb hard disk, and render 2 billion polygons per second. The Xbox 2 (due at the end of 2006) may employ a Pentium 4, a clock speed of 1GHz, 1Gb RAM, a 160Gb hard disk, and render 2-3 billion polygons per second.

1.3. High and Low Profile Games

Marner distinguishes between high profile and low profile games [Marner 2002].

A high profile game is endowed with massive development costs (perhaps US$5 million or more), a generous advertising budget, a large development staff, and a very visible presence for game retailers and magazines. To recoup the enormous upfront expenses, high profile games tend to utilise cutting-edge graphics (which require high hardware performance), and tie-ins with other media such as movies or books.

A low profile game is aimed at a smaller market, and may be limited to a single platform, or user community. It may have been developed by a single person (or small group), and be advertised in specialized newsgroups and mailing lists, leading to a substantial reduction in costs. Low profile games may have less ‘polish’, use less state-of the-art graphics, and place more emphasis on game design and characterization.

2. Java for Games

In this section, we identify reasons for using Java for games programming, and look at some of the popular arguments against it.

We focus on the Java language and its standard libraries (i.e. those found in J2SE 1.4); Java 3D will be considered in later sections.

2.1. Why use Java for Games?

The advantages of programming with Java are well known: object orientation, cross-platform support, code reuse, ease of development, the availability of tools, reliability and stability, good documentation, continuing support from Sun Microsystems, low development costs, the ability to use legacy code (e.g. C, C++), and increased programmer productivity.

The portability of Java is sometimes overstated – games in particular often require some 'tweaking' to improve their performance on different OSes/machines, such as in the scheduling of threads, data structure and/or algorithm design, or the choice of GUI components. Also, if legacy code written in a different language is utilised then portability is frequently compromised.

Productivity is an important advantage, although hard to quantify. An old study from 1998 suggested that software written in pure Java instead of C++ results in a 25% overall time/cost saving, corresponding to an overall productivity increase of 30% (the increase for the code phase alone is 65%) [Quinn and Christiansen 1998]. It is likely that these figures are greater today since the capabilities of the J2SE have improved (e.g. the libraries are larger, tool speed is better).

2.2. Ways of Using Java in Games

There are several approaches to writing games with Java:

- *Applets*. Usually the applet is a client for a multiplayer game, and communicates back to its home host where the game server is located. A drawback of applets is their security features, which need to be adjusted by the user before an applet can save files or communicate with different hosts.
- *Pure Java applications*. The meaning of 'pure' is somewhat vague since seemingly pure libraries, such as Java 3D, utilise code outside of Java, hidden from the programmer.
- *Dirty Java applications*. Dirty Java programs use a mix of Java and other languages (typically C or C++) by employing JNI, network links, or the Java Communications API [Kreimeier 1999]. This opens up Java to code which may be better optimized for the underlying hardware/OS, but affects portability.
- *Java as a scripting engine*. This is a variant of the dirty Java idea, where the majority of the application is implemented in another language. For example, C++ may be used to write the graphics rendering engine, image loaders, and the user interface for joysticks. Java is utilised for tasks that are hardware independent and/or less performance-oriented, such as user input validation and the game logic.

2.3. Misconceptions

There are several misconceptions about Java and gaming:

- *No one writes serious games in Java*. This rests on the definition of 'serious', which normally means commercial, high profile games. Of course, there are thousands of low profile, freeware/shareware Java games out on the Web.
- *Java is too big/slow for games programming*, especially when compared to C or C++;

- *Sun Microsystems is not really interested* in making Java suitable for the games industry.

We address each of these points in the following subsections.

2.3.1. Commercial Games

Java is nowhere near as popular a gaming language as C or C++, but it is being used, and in games which have become bestseller.

Commercial Java games are mostly CD-based applications, and use dirty Java (often Java and C++).

Tom Clancy's Politika (1997) from *Red Storm Entertainment* (http://www.redstorm.com/) was written in almost pure Java. *Tom Clancy's Rainbow Six* (1999) started out by mixing C++ and Java, but they dropped the Java when the code grew too complex [Upton 2000]. Both *Shadow Watch* (2000) and *Tom Clancy's ruthless.com* (1998) were written in Java mixed with C/C++.

Roboforge (2001) by *Liquid Edge Games* (http://www.roboforge.com) was coded in Java and Java 3D; it was given an "Excellent 87%" by *PC Gamer Magazine* in December 2001.

IL-2 Sturmovik (2001) by *Maddox Games* (http://www.il2sturmovik.com/) used dirty Java, with part of the game engine written in Java but all the graphics in C++.

At QuakeCon 2001, *Fullsail Real World Entertainment* showed a Quake clone called *Jamid* and *F1 Grand Prix Demo*, both written in Java using Java 3D. However, nothing much has been heard of them lately.

Dirty Java was used by *Jellyvision* (http://www.jellyvision.com/) in their popular *Who wants to be a Millionaire* (2000) and *You don't know Jack* (1995) games. They utilised C++ with Java for the game logic, which was also the approach in *Majestic* (2001) by *Electronic Arts* (http://www.ea.com/).

Java was used as a scripting language in the acclaimed *Vampire - the Masquerade: Redemption* (2000) from *Nihilistic software* (http://www.nihilistic.com/). The company was very happy with Java, although it only used JDK 1.1 [Huebner 2000].

Star Wars Galaxies from *LucasArts* (http://www.lucasarts.com/) is being scripted with a 'slimmed-down' version of Java.

Runescape (http://www.runescape.com) is a massive 3D multiplayer fantasy adventure game. It is probably the largest pay-to-play Java online game, with over 5000 paying members and 1 million free registrations. Clients can use a Java applet to play, or download a Windows-based client application.

A telling exception from this list are Java games on consoles. The only examples to date are the Sega Dreamcast games *Skies of Arcadia* (2000) and *Daytona USA* (2001), which contained a PersonalJava virtual machine [Patrizio 2000]. However, the Dreamcast is no longer in production.

2.3.2. Freeware/Shareware Games

There are many Java games out on the Web, but finding an entertaining game requires a careful search.

First some history: JDK 1.0 was released early in 1996, JDK 1.1 in early 1997, and Java 2 (JDK 1.2) at the end of 1998. Back then, there was a lot of hype promoting Java as a perfect way of programming networked, graphical programs (i.e. as applets), and this is reflected in the very large number of freeware game applets dating from 1996-1998.

Unfortunately, the early versions of Java were very slow, leading to disappointing game play. Also, many programmers ignored the overheads of downloading large code, images, audio, etc. Applet security restrictions and weaknesses in Java's media APIs (graphics, sounds, etc) were further problems. All of this contributed to a general feeling that Java was a toy language.

Recent versions of Java are quite different: speed is much improved, and APIs crucial to gaming, such as graphics and audio, are of a high quality. Also, there has been a move away from the use of applets towards the downloading of client-side applications. Java applications require less configuration, and once downloaded they don't need to be downloaded again.

Java's backward compatibility allows the applets from 1996-8 to be executed, and they will often run quicker than originally. However, it's probably best to steer clear of these Java dinosaurs, and look for more modern code.

There are many Web sites with Java games. The emphasis of the following list are on applications/applets for playing:

- ArcadePod.com, http://www.arcadepod.com/java/
  Over 700 Java games, nicely categorized.
- Java 4 Fun, http://www.java4fun.com/java.html
  Similar in style to ArcadePod, and a good set of links to other sites.
- Java Game Park, http://javagamepark.com
  Organized around game categories.
- Java Games Central, http://www.mnsi.net/~rkerr/
  A personal Web site which lists games with ratings and links.
- jars.com, http://www.jars.com
  This is a general Java site, but contains many games.
- Java Shareware, http://www.javashareware.com/
  Another general site: look under the categories: applications/games/ and applets/games.

Programmers looking for source code should start elsewhere, as detailed below:

- FreshMeat.com, http://freshmeat.net/
  Freshmeat maintains thousands of applications, most released under open source licenses. The search facilities are excellent, and can be guided by supplying game category terms. The results include rating, vitality, and popularity figures for each piece of software. A recent search for Java games returned over 100 hits.
- SourceForge, http://sourceforge.net/search/
  SourceForge acts as a repository, and management tool, for software projects, many with source code. A recent search for Java games returned over 250 hits.

However, many of the projects are at the planning stage, with some inactive for long periods.

- Code Beach, http://www.codebeach.com
  CodeBeach has a searchable subsection for Java games that currently contains nearly 90 example.
- Programmers Heaven, http://www.programmersheaven.com/zone13/
  It has a 'Java zone' containing some games.

2.3.3. Java is too big/slow

When people say that Java is too big/slow, they usually mean when compared to C or C++.

Earlier version of Java were *very* slow. Marner presents figures showing JDK 1.0 to be *20 to 40 times slower* than C++ [Marner 2002]. Fortunately, successive versions have brought Java much closer to C++: J2SE 1.4 is typically 1.2-1.5 times slower. These numbers depend greatly on the coding style used – Java coded in a straightforward, textbook-style may still be 2.5 to 4 times slower. This shows that Java programmers must be good programmers in order to utilise Java efficiently (but that's true of any language). Jack Shirazi's Java Performance Tuning website (http://www.javaperformancetuning.com/) is a good source for performance tips, and links to tools and other resources.

Recent numerical benchmarks on Linuxes found that compiled C++ and Fortran were at least twice as fast as Java byte code. But the performance was very dependent on the chosen JVM; IBM's implementation exceeded the performance of C++ code compiled with gcc [Ladd 2003].

A detailed comparison of difference versions of Java and other companies' virtual machines and native code compilers can be found in [Doederlein 2002].

An area of Java that is still slow is its GUI API, Swing. GUI components are created and controlled from Java, with little OS support: this increases their portability and makes them more controllable from within a Java program. The downside is speed since Java imposes an extra layer of processing above the OS. This is one reason why some games applications still utilise the original Abstract Windowing Toolkit (AWT) -- it is mostly just simple wrapper methods around OS calls. However, most games do not require complex GUIs: full-screen game play with mouse and keyboard controls are the norm, so GUI speed is less of a factor.

Another speed drain is Java's garbage collector, which is run automatically by the JVM, and may cause an appreciable slowdown in game animation. This is one area where good coding style can alleviate the problem, by the programmer reducing the number of temporary objects created during execution.

Knowing which version of Java you are using is important, especially when executing applets. Many browsers, such as Netscape 6 and Microsoft Internet Explorer 6 default to a JVM at version 1.1 unless the user configures things differently.

The speed-up in Java is mostly due to improvements in compiler design. The Hotspot technology introduced in J2SE 1.3 enables the run-time system to identify crucial areas of code that are utilised many times, and these are aggressively compiled. Hotspot technology is relatively new, and it's quite likely that future versions of Java

will find further speed-ups. The Hotspot technology has the unfortunate side-effect that program execution is often slow at the beginning until the code has been analyzed and compiled.

A final point about speed is *knowing what to blame* when a Java program runs slowly. An increasingly large part of the graphics rendering of a game is handled by hardware or software outside of Java. For example, Java 3D passes all of its rendering down to OpenGL or DirectX which may emulate hardware capabilities, such as bump mapping. Often the performance bottleneck in network games is the network. Dirty Java code passes control to libraries/functions which are quite independent of Java.

Java uses a large amount of memory at runtime, for two main reasons: it loads the JVM and associated libraries into memory (about 5-10 MB), and Java objects are stored on the heap rather than the stack [Hutchinson 2000]. Memory requirement is a serious issue when porting Java to consoles. For instance, the PlayStation 2 only has 32MB of RAM. It is being addressed in the Java Game Profile, described below, which uses the Java Micro Edition, so called because its memory needs are more suited to hand-held devices. A longer term solution is to simply wait – the memory specifications for the next generation of consoles are more than sufficient to cope with Java. However, waiting 2-4 years makes poor commercially sense.

2.3.4. Sun Microsystems isn't interested in Java games

The games market isn't a traditional one for Sun, and it will probably never have the depth of knowledge of a Sony, Sega, or Nintendo. However, the last few years have shown its increasing commitment to gaming.

J2SE has strengthened its games support: version 1.4 introduced full-screen imaging and page flipping using hardware. Faster I/O, memory mapping, and support for non-block sockets are all useful, the latter especially so in client/server multiplayer games. Version 1.3. introduced a timer API that can be employed in animation, and improved graphics and audio support. Recent Java extension libraries, such as Java 3D, the Java Media Framework (JMF), Java Sound, the Java Communications API, Jini, and JAXP (Java's peer-to-peer API) all offer something to games programmers.

At the 2001 Game Developers Conference, Sun announced a collaboration with several other companies, including Sega and Sony, to develop a Java gaming API called the Java Game Profile [JSR 134 2001]. It will be targeted at two markets: high-end game devices (using J2ME) and desktop machines (using J2SE).

The proposal lists several advantages of Java over other languages: its superior reliability, reduced time-to-market on games, device independence, platform scalability, the prevention of lock-in to a particular platform, and a broader target market.

It names several media API that may be useful, including Java 3D, JMF and Java Sound, and suggests the creation of some new ones: APIs for physics modeling, animation, and game marshalling. Companies involved in the project include *Math Engine* (http://www.mathengine.com/), who developed the *Karma* rigid body dynamics API, and *GameSpy* (http://www.gamespy.net/) who offer toolkits that allow players to utilise various multiplayer game servers.

The proposal does not say how these extensions will be crammed into a typical game console, which may explain why there hasn't been much activity in the last year.

Currently, the Java Game Profile specification is due out in the second quarter of 2003.

Part of the initiative was the creation of the JavaGaming.org website (http://www.javagaming.org), which has recently become more active, and offers some good forums on games programming.

2.4. When should Java be used for Games Programming?

Java is suited for programming high profile, commercial games, but in conjunction with C or C++ so that legacy code, such as a proprietary game engine, can be utilised. It should also be remembered that well-written Java code is approaching the speeds of C/C++.

A serious issue is when Java will be available on consoles. It seems likely that Java will appear on the PlayStation in the near future.

Java is more than adequate for low profile games on PCs,, and a quick search through the sites mentioned in section 2.3.2 will confirm its popularity.

3. Graphics APIs for Gaming

Java 3D is built on top of lower level graphics libraries: it can either use DirectX or OpenGL. It is useful to understand a little about these APIs before considering Java 3D itself.

3.1. Direct X

DirectX is a collection of related gaming modules that provide access to hardware via low-level function available through the Windows OS (http://www.microsoft.com/directx). Its principal aims are to make Windows suitable for game development and to simplify the software interface to the enormous variety of hardware that can be part of a typical PC. DirectX bypasses most of Windows to communicate directly with hardware through HAL (see figure 1). If the required feature is not present then HEL is brought into play to emulate the capability.

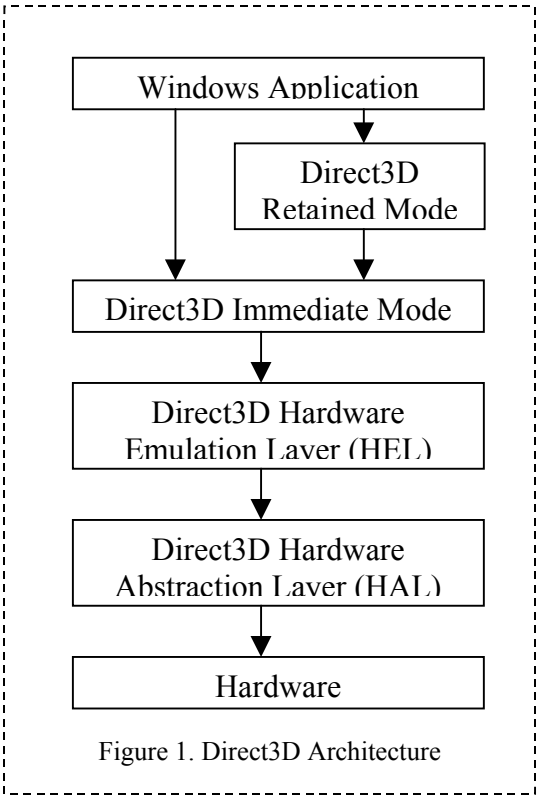

Figure 1. Direct3D Architecture

The graphics API, DirectX Graphics (formerly DirectDraw and Direct3D in v.7.0 and earlier), supports a traditional graphics pipeline, describing all geometry in terms of vertices and pixels. This places quite a burden on the programmer's shoulders, but the payoff is an API with a multitude of games-related features,

such as programmable vertex and pixel processing languages, multisampling, and point sprites.

DirectX Graphics offers some higher-level programming elements with its *retained mode* and the Direct3DX (D3DX) utility library. Retained mode provides a simple scene graph (called a frame hierarchy) and some geometry constructs. However, most DirectX programmers don't use it, claiming it to be poorly implemented and aimed at 3D modeling rather than gaming. D3DX is more widely appreciated: it includes several gaming elements such as a skinning library for working with meshes, and functions for vertex and pixel shading.

Briefly, the other DirectX API are:

- DirectX Audio: for game soundtracks with 3D sound positioning and effects;
- DirectPlay: help for networked games, including guaranteed and non-guaranteed message delivery, traffic control for slow links, game session management, peer-to-peer, and voice transmission;
- DirectInput: offers interfaces for a variety of input devices, such as joysticks, headgear, and force-feedback devices;
- DirectShow: provides multimedia support for video streaming, editing, broadcasting;
- Direct Setup: aids the installation of the DirectX components necessary for the game application.

Naturally, the Xbox supports DirectX, but a somewhat different version from DirectX for PCs. Some of the main changes are: a simpler approach to object creation on screen, the replacement of DirectInput by a specialized game pad interface, and a new streaming API instead of DirectShow.

Java and DirectX. There is no technical reason why Java cannot utilise DirectX since JNI enables Java to call any C or C++ function, and in fact this approach has been used with Java 3D.

Microsoft offers Microsoft SDK for Java, but it is limited to an insubstantial version of Java and to DirectX 3.0 (http://www.microsoft.com/java/sdk/).

3.2. OpenGL

OpenGL is a software API for writing 3D (and 2D) graphics applications across a wide range of hardware and OSes (http://www.opengl.org/). It is a low-level API based around a graphics pipeline for pixel and vertex manipulation, similar in spirit to DirectX Graphics. It offers relatively few functions (about 200), avoiding high-level 3D commands for geometry and scene control, and has no support for windowing or input. The API has evolved slowly around a core set of features, but it is possible for extensions to be added, a common occurrence among graphic card vendors so that card-specific capabilities can be accessed.

The OpenGL specification is managed by the OpenGL Architecture Review Board (ARB) made up of many key industrial players in the 3D graphics market.

OpenGL is intended to be used in a wide range of application areas, such as CAD/CAM, visualization, geographic mapping, and also gaming. A current list of games using OpenGL is maintained at http://www.OpenGL.org/users/apps_hardware/applications/games.html, and includes favourites like Quake II, Descent 3, and Half Life.

Higher-level programming with OpenGL is possible. The OpenGL Utility Library (GLU) offers 2D image scaling, basic 3D geometry constructs, NURB curves and surfaces, and special-purpose matrix transformations. GLU is a standard part of every OpenGL distribution. The OpenGL Utility Toolkit (GLUT) is another standard component, made up of libraries for window management, call-back driven event processing, utilities for creating 3D solids and wireframes, fonts, and timers, all supplied in a platform independent manner. However, most programmers find GLUT too inflexible for serious applications, and tend to code up the necessary features themselves.

Open Inventor (http://www.sgi.com/software/inventor/) supports a scene graph interface on top of OpenGL, with a variety of pre-defined objects such as cubes, materials, cameras, lights, input devices, views, and editors. It simplifies tasks such as scene traversal, interactions, picking, and bounding box calculations. It is available on UNIX/Linux and Windows platforms. A drawback is the speed of its scene graph implementation. Kahlua is a Java wrapper for Open Inventor (http://www.igd.fhg.de/CP/kahlua/).

OpenGL Performer (http://www.sgi.com/software/performer/overview.html) addresses the speed problems of Open Inventor, with a scene graph API quite similar to Java 3D's, and employs many of the same optimizations. It is aimed at visual simulations, virtual reality and applications that require performance and a fixed frame rate. It is currently available on SGI IRIX and Linux platforms with C and C++ bindings.

There are several other OpenGL high-level tools: OpenGL Optimizer for rendering complex surface-based models as found in CAD/CAM, OpenGL Volumizer for medical and scientific imaging, and OpenGL Shader for visual effects like bump mapping, volume shading and multiple textures.

There is an alpha version of an OpenGL for the PlayStation 2, implemented by *DataPlus* (http://www.dataplus.co.jp/OpenGL4ps2.html). Part of the effort to support Linux on the PS2 includes a port of OpenGL (http://playstation2-linux.com/projects/OpenGLstuff/). UbisoftGL is another port, described in an article dating from 2001 at *Gamesutra.com* (http://www.gamasutra.com/features/20010618/ourdi_01.htm).

Java and OpenGL. As with DirectX, Java can easily utilise OpenGL by calls through JNI to the C or C++ interfaces, as done in Java 3D.

Several Java bindings for OpenGL have been released over the years, but they have a habit of being incomplete, containing bugs, lacking support and documentation, and eventually disappearing. One exception is GL4Java (also known as OpenGL for Java) from *Jausoft* (http://www.jausoft.com/gl4java.html), which works with the latest versions of OpenGL, many vendor extensions, GLU, and GLUT.

GL4Java has been the basis of several good games, including *Arkanae*, a role-playing fantasy game (http://arkanae.tuxfamily.org/), and *Java is Doomed*, a first person shooter closely related to DOOM (http://javaisdoomed.sourceforge.net). Both games utilised game engines to simplify the coding tasks.

3.3. DirectX versus OpenGL

A head-to-head comparison of DirectX and OpenGL isn't really fair: DirectX is a set of APIs for graphics, audio, multimedia, networking, and others, whereas OpenGL is aimed solely at the graphics domain. Even if we restrict the comparison to DirectX Graphics and OpenGL, we must still decide whether to limit OpenGL to its core features or to permit vendor-specific extensions.

In most technical areas, DirectX Graphics and OpenGL are almost equivalent, since both are based on the same graphics pipeline architecture, and there is a strong flow of ideas between the two. DirectX Graphics is aimed more towards game development, including such things as programmable vertex and pixel control, but these features are available as extensions to OpenGL on some platforms.

The most significant differences between the two APIs are unrelated to their functionality. OpenGL is ported to a wide range of platforms and OSes, while DirectX is limited to PCs running Windows, and the Xbox. DirectX is controlled by Microsoft alone, while the OpenGL ARB allows input from many partners, and successful extensions by vendors may be moved to the core in future releases. There is an open source implementation of the OpenGL API, released by SGI (http://oss.sgi.com/projects/ogl-sample/). The source for DirectX is not available.

4. Java 3D

The Java 3D API provides a collection of high-level constructs for creating, rendering, and manipulating a 3D scene graph composed of geometry, materials lights, sounds, and so on. Java 3D is being developed by Sun Microsystems and currently at v.1.3.1. There are two variants: one implemented on top of OpenGL, the other above DirectX Graphics. The low-level API handles the native rendering at the vertex and pixel levels, while the 3D scene, application logic, and scene interactions are carried out by Java code. This dual approach encourages application portability, hardware independence, and high-speed rendering. Intended application areas include visualization, CAD/CAM, virtual reality, simulation, and gaming.

4.1. The Scene Graph

Java 3D's scene graph is a directed acyclic graph (a DAG), so there is a parent-child relationship between most of its nodes, and the graph contains no loops. It is possible for nodes to be shared in a graph, typically node properties such as geometry.

A simplified scene graph for an office is shown in Figure 2. The office group contains three nodes for a desk and two chairs. Each node has a shape and colour (node components), and the shape for the two chairs are shared.

The choice of symbols in Figure 2 is not arbitrary. The symbols in Figure 3 are

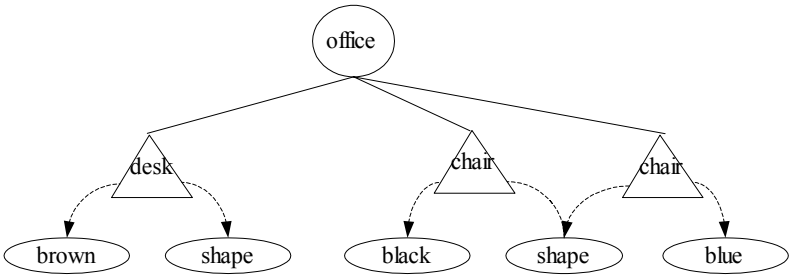

Figure 2. Scene Graph for an Office.

commonly used to draw Java 3D scene graphs; they are explained below.

In true object oriented style, all the nodes in a scene graph inherit from a SceneGraphObject class, but are subclassed in different ways.

Shape nodes are used for the graphical objects in the virtual world. The principal class is Shape3D, which acts as a container for Geometry and Appearance node components. The main geometry class is GeometryArray, used for drawing points, lines, and polygons. There are many Appearance subclasses, including ones for colour, texture, and transparency.

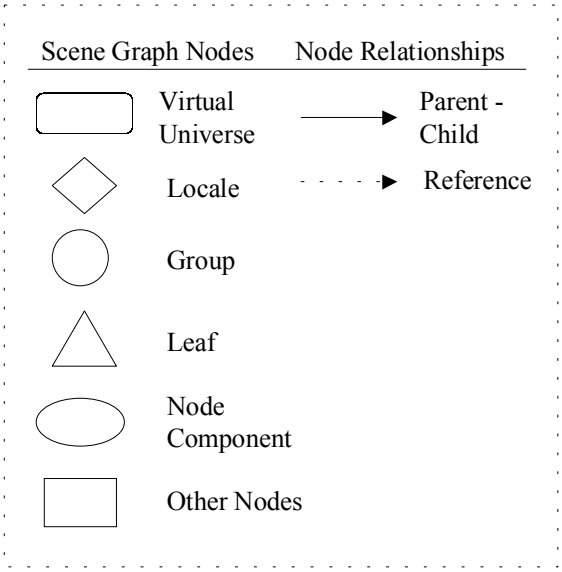

Figure 3. Scene Graph Symbols.

Environment nodes handle environmental concerns inside the virtual world, such as lighting, sound, and behaviours that change the world.

Group nodes are containers for other groups or for leaf nodes. Leaf nodes are usually shape or environmental nodes. The Group class supports node positioning and orientation for its children, and is subclassed in various ways. For instance, BranchGroup allows children to be added or removed at run time, while TransformGroup permits the position and orientation of its children to be changed.

4.2. HelloUniverse

The standard first example for Java 3D programmers is HelloUniverse (it appears in chapter 1 of the Java 3D tutorial). The program displays a rotating coloured cube, as in Figure 4.

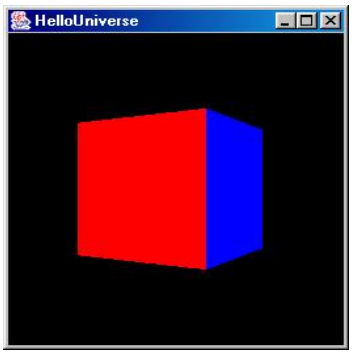

Figure 4. A Rotating Coloured Cube.

The scene graph for this application is given in Figure 5.

VirtualUniverse is the top node in every scene graph, and represents the virtual world space and its coordinate system.

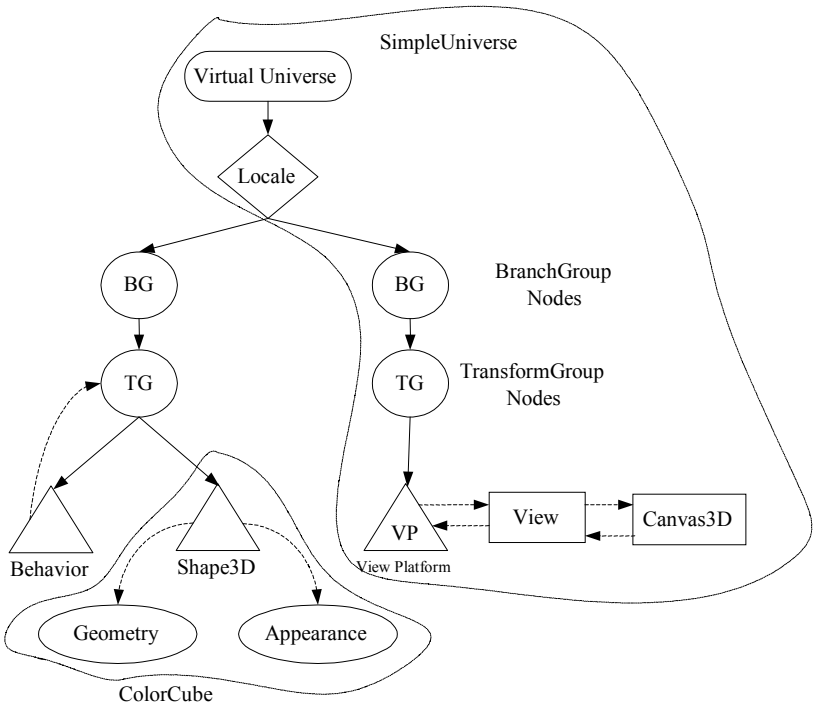

Figure 5. Scene Graph for HelloUniverse

Locale acts as the location in the virtual world where the scene subgraph is situated. In many programs there is only need for a single Locale, but it is possible to have many.

Below the Locale node there are always two subgraphs – the left branch is the *content branch graph*, holding program-specific content such as geometry, lighting, textures, and the background. The content-branch graph differs significantly from one application to another.

The right hand branch below Locale is the *view branch graph* and specifies the user's position, orientation, and perspective as they look into the virtual world from the physical world (e.g. from in front of a monitor). The ViewPlatorm node specifies the viewer's position in the virtual world; the View node states how to turn what the viewer sees into a physical world image (e.g. a 2D picture on the monitor). The Canvas3D node is a Java GUI component that allows the 2D image to be placed inside a Java application or applet.

The VirtualUniverse, Locale, and view branch graph often have the same structure across different applications since most programs use a single Locale and view the virtual world through a 2D image on a monitor. For these applications, the relevant nodes can be created with the SimpleUniverse utility class, relieving the programmer of a lot of graph construction work.

4.3. Java 3D Strengths

Java 3D's scene graph makes programming much easier for novices (and even for experienced programmers) because it emphasizes scene design rather than rendering, by hiding the graphics pipeline. The scene graph naturally supports complex graphical elements such as 3D geometries, lighting modes, picking, and collision detection. At the Java 3D implementation level, the scene graph can be used to group shapes with common properties, carry out view culling, occlusion culling, level of detail selection, execution culling, and behaviour pruning – all optimizations which must be coded directly by the programmer in lower-level APIs. Java 3D utilises Java's multithreading to carry out parallel graph traversal and rendering, both very useful optimizations.

Java 3D is designed with performance in mind, which it achieves at the high-level by scene graph optimizations, and at the low-level by being built on top of OpenGL or DirectX.

Some programmer-specified scene graph optimizations are available through the use of capability bits to state what operations can/cannot be carried out at run time (e.g. prohibiting a shape from moving). Java 3D also permits the programmer to bypass the scene graph, either totally by means of the *immediate mode*, or partially by using a combination of immediate and retained modes (called *mixed mode*). Immediate mode gives the programmer complete control over rendering and scene management, and is similar in style to coding in DirectX Graphics and OpenGL.

Java 3D's view model separates the virtual and physical worlds (through the ViewPlatform and View nodes). This makes it straightforward to reconfigure an application to utilise a range of output device, from a monitor, to stereo glasses, to room-sized 'caves'.

Virtual world behaviour is coded with Behaviour nodes in the scene graph, which are triggered by events. Among other things, this offers a different style of animation, instead of the usual update-redraw cycle, by employing multithreading to execute behaviours as events occur.

The core Java 3D API package, javax.media.j3d, supports basic polygons and triangles within a scene graph, while the com.sun.j3d packages add a range of utility classes including SimpleUniverse, mouse and keyboard navigation behaviours, audio device handling, and loaders for some 3D file formats.

Java 3D's sound support is based on the Java Sound API and allows 2D and 3D audio output, ambient and spatialized sound.

Geometry compression is possible, often reducing size by an order of magnitude. When this is combined with Java's NIO and networking, it facilitates the ready transfer of large quantities of data between applications such as multiplayer games.

Java 3D is integrated with Java and its other libraries. This brings the benefits of object orientation (classes, inheritance, polymorphism), threads, exception handling, and APIs such as JMF and JAI.

Java 3D's reliance on Java and OpenGL/DirectX gives it a high degree of portability, with related advantages such as reduced development time and cost.

4.4. Java 3D Weaknesses

Java 3D's scene graph is often considered an unreasonable overhead, especially by programmers with experience of OpenGL and DirectX. Although it does introduce overheads, they should be judged against the optimizations which the scene graph allows. These can be implemented in a low-level API by an experienced programmer, but at what cost in time and maintainability? Also most large OpenGL and DirectX applications need a data structure like a scene graph anyway, in order to manage code complexity.

The high-level nature of the scene graph makes Java 3D code harder to tune for speed, unlike OpenGL/DirectX. However, a programmer does have the option of moving to the mixed or immediate modes of coding if absolutely necessary.

Similarly, the hiding of the low-level graphics API makes it harder for a programmer to code around bugs in the APIs or the drivers.

Since Java 3D is available on top of OpenGL and DirectX, it is forced to offer a common denominator of features from both APIs. For example, this is why vertex and pixel shading is absent. However, Java 3D version 1.4 is scheduled to allow extensions to its core features, in a similar vein to extensions in OpenGL.

Java 3D requires the installation of Java and a graphics API, which can easily amount to 10-20MB on a hard disk. This is a serious concern when attempting to port Java 3D to consoles, but a problem that will disappear in a few years as the next generation of consoles appear.

The amount and diversity of software required for Java 3D makes it difficult to automatically install everything along with the application. Sun Microsystems needs to develop an API similar to DirectSetup to simplify this process. Currently a developer must utilise software like InstallAnywhere (http://www.zerog.com/) or JExpress (http://www.denova.com/).

It is interesting to examine how successful 3D multiplayer games, such as EverQuest (http://everquest.station.sony.com/), handle installation issues. Most of the EverQuest software comes on a large set of CDs, requiring as much as 1Gb of disk space. Also, when a EverQuest client logins to a server, it may download additional player data. This indicates the need for two forms of installation: CD-based for the majority of the application (e.g. via InstallAnywhere) and network-based for updates, perhaps by using Java Web Start (http://java.sun.com/products/javawebstart/).

5. Java 3D Compared to OpenGL and DirectX

We have already discussed the central difference between Java 3D and OpenGL / DirectX : Java 3D uses a scene graph while OpenGL / DirectX Graphics supply interfaces to a graphics pipeline.

5.1. Java 3D Compared to OpenGL

OpenGL focuses on graphics while Java 3D has wider concerns, including support for audio and output devices.

OpenGL is not tied to any particular language: there are bindings for many languages (see http://www.opengl.org/developers/documentation/implementations.html). This means that OpenGL code cannot assume the existence of standard libraries outside of itself. By contrast, Java 3D is tightly linked to Java and programmers must use Java libraries for I/O, networking, multimedia, and so on. While this is a restriction, it enhances the portability of applications.

OpenGL is controlled by the ARB, while Java 3D is a Sun Microsystems product.

OpenGL is stable, evolves slowly, and has been ported to a wide range of OSes and devices. Java 3D is starting to settle down around core functionality, and is only available on the main OSes and PCs at present.

Marner has carried out comparative performance tests on OpenGL and Java 3D versions of the same 3D non-interactive space of randomly positioned, scaled, and rotated boxes [Marner 2002]. He used the C++ and GL4Java bindings for OpenGL, and v.1.3.0 beta 1 of Java 3D.

The C++ version was fastest, the GL4Java implementation a little slower, and Java 3D was about 2.5 times slower. This benchmark has been somewhat invalidated by reports that the code slowdown was due to a performance bug in that version of Java 3D, and a poorly optimized scene graph (http://www.rolemaker.dk/articles/evaljava/errata.htm). Unfortunately, the timings have not been repeated with the latest version of Java 3D.

Marner's code shows other striking differences between Java 3D and OpenGL. The C++ and GL4Java programs are of comparable sizes (about 10 classes; 30 pages of code with documentation), while the Java 3D application is much smaller (5 classes and 11 pages). Marner comments on the complexity of the OpenGL code which required a kd-tree data structure, a culling algorithm around the view frustum, and preprocessing vertex operations. All of these capabilities are built into Java 3D, and so were not required in the Java 3D program. In the GL4Java source, the optimized view frustum algorithm is very hard to understand, but is responsible for an order of magnitude speedup over the simpler version. Also, the OpenGL applications could have been considerable faster if extensions available for the graphics card were employed.

Another outcome of Marner's work is the negligible overhead of JNI: GL4Java uses JNI to interface Java to OpenGL, and its performance is only slightly less than the C++ binding.

5.2. Java 3D Compared to DirectX

A comparison is complicated by DirectX being a set of related APIs and Java 3D being able to call upon other Java libraries. For example, DirectX Graphics can be equated with Java 3D *and* Java 2D, and Direct Audio can be compared to the audio elements of Java 3D and the Java Sound API.

One area where DirectX appears superior is in its support for application installation with Direct Setup.

As far as I know, there have been no attempts to compare the performance of similar applications written in DirectX and Java 3D.

Java 3D implemented on top of DirectX has a reputation of being a little slower than Java 3D / OpenGL, and less stable. The speed reduction is due to a mismatch between Java 3D's scene graph data structures and those maintained by DirectX Graphics.

6. Is Java 3D suitable for writing 3D Games?

The question of whether to use Java 3D for games programming is complicated by the wider question of the capabilities of Java, which was considered in section 2.

Another aspect of the question is how Java 3D compares to the popular APIs, OpenGL and DirectX, which was addressed in section 5.

Aside from these, there are three main issues when considering Java 3D:

- performance;
- console support;
- the availability of existing Java 3D games.

6.1. Performance

Performance data on Java 3D is scarce – only the Marner study has seriously considered Java 3D's performance against one of its rivals, OpenGL [Marner 2002]. At present, performance 'figures' seem mostly based on hearsay. A common example is to give DirectX 10/10, OpenGL 7.5/10, and Java 3D 3/10.

Java 3D performance is often equated with Java performance: the perceived slowness of Java 'demonstrates' the slowness of Java 3D. In fact, since Java 3D relies on OpenGL or DirectX for rendering, much of the graphics processing speed of Java 3D is independent of Java.

History suggests that performance will become a less important consideration as the base speed of hardware keeps increasing. Even today, most low profile games rely less on dazzling 3D special effects, more on gaming characterization and story. High profile games will always need performance, but the real bottleneck will not be the platform but the network, as multiplayer games begin to dominate.

Performance should be considered alongside issues such as code complexity, productivity, maintainability, and portability. These criteria strongly influence a move towards higher-level API, as typified by Java 3D.

6.2. Console Support

The lack of a console implementation for Java 3D is rather ironic considering Java's "write once run anywhere" slogan. As mentioned earlier, the Java Game Profile [JSR 134 2001] may eventually address the problem on the Sony PlayStation 2 (or 3).

6.3. Existing Java 3D Games

The existence of games is important mainly for its psychological influence of developers and players: commercially successful games, which are fun and exciting to play, and implemented with Java 3D, would go a long way to persuading developers to start using Java 3D, and reassure users that they are not riding around in a Model T Ford.

It is true that Java 3D has been employed in many fewer gaming examples than OpenGL or DirectX. Nevertheless there are several excellent high profile games and many low profile examples.

High Profile Games

- Pernica / Starfire Research (http://www.starfireresearch.com/pernica/pernica.html)
  An online role-playing game similar in nature to EverQuest, Asheron's Call, Ultima Online, and others.
- Cosm / Navtools, Inc. (http://www.cosm-game.com/)
  Another fun online fantasy-based role-playing game.
- RoboForge / Liquid Edge Games (http://www.roboforge.net/)
  The object is to build and train a robot to fight in online tournaments.
- DALiWorld / DALi, Inc. (http://www.dalilab.com/)
  A distributed aquatic virtual world, inhabited by autonomous artificial life.
- Terra Live Map / JackedIn (http://www.terracorps.com/cgi-bin/clanmap.pl)
  A Java 2D/Java 3D application that lets Web visitors peer into a massively multiplayer online game by showing the terrain map, object ownership, and other data.

Low Profile Games

The 'other sites' page at j3d.org (http://www.j3d.org/sites.html) is the most comprehensive source for Java 3D examples, and includes a games section, and a demos section with many links to games.

Another strategy for finding Java 3D games and source is to visit the sites listed in section 2.3.2, and restrict the search to strings containing "3D". For example, a search at sourceForge for "java3d" returned seven hits with source code available, including 3 multiplayer games, a tank simulator, and a first person shooter.

6.4. Summary

Java 3D is suitable for writing 3D games when high performance is less of a priority than high-level programming capabilities.

The scene graph approach provides its own optimizations but its main benefit is increased programmer productivity. This shows itself in the developer's ability to write larger programs with less complex coding, program portability across a range of platforms, increased maintainability, and reliability.

Java 3D's main weakness, and a serious one, is the lack of a console version.

References

Doederlein, O.P. 2002. "The Java Performance Report", November, http://www.javalobby.org/

Hutchinson, R. , 2001. "The Evolution of Performance on the Java Platform: A Panel Discussion". *JavaOne 2001*. http://java.sun.com/javaone/javaone2001/pdfs/2647.pdf

Huebner, R. 2000. "Postmortem of Nihilistic Software's Vampire: the Masquerade – Redemption", *Game Developer Magazine*, July, http://www.gamasutra.com/features/20000802/huebner_01.htm

Java Specification Requests (JSR) 134, 2001. Java Game Profile, http://jcp.org/jsr/detail/134.jsp

Kreimeier, B, 1999. "Dirty Java – Using the Java Native Interface within Games", *Game Developer Magazine*, July, http://www.gamasutra.com/features/19990611/java_01.htm

Ladd, S.R. 2003. "Linux Number Crunching: Benchmarking Compilers and Languages for ia32", *Coyote Gulch Productions*, January, http://www.coyotegulch.com/reviews/almabench.html

Marner, J. 2002. "Evaluating Java for Game Development", Dept. of Computer Science, Univ. of Copenhagen, Denmark, March, http://www.rolemaker.dk/articles/evaljava/

Patrizio, A. 2000. "Dreamcast Gets Java Jolt", *Wired News*, http://www.wired.com/news/culture/0,1284,36810,00.html

Strategy Analytics. 2002. "Games Console Sales To Hit 41.9m Units In 2002", http://www.strategyanalytics.com/press/PR00023.htm, December

Upton, B. 2000. "Postmortem: Read Storm's Rainbow Six", *Game Developer Magazine*, May, http://www.gamasutra.com/features/20000121/upton_01.htm

Veronis Suhler Media Merchant Bank, 2000. "Entertainment Industry Segment Performance", http://www.veronissuhler.com/filmed/segment.html

Quinn, E. and Christiansen, C. 1998. "Java Technology Pays Positively", *IDC Bulletin*, #W16212, May.

Appendix: Using Java 3D

The following sections provide details about Java 3D-related hardware, software, text books, and how to find help and advice. There are also sections on tools for Java/Java 3D programming, and books on Java gaming and 3D graphics.

A.1. Hardware

It is difficult to give minimum hardware requirements for Java 3D applications since it depends on the amount and complexity of the operations carried out by the code. I have had satisfactory results with a 900 MHz processor, 256 MB of RAM, and a display card with 64 MB of memory. When I ran the Java 3D Fly-through example (http://www.javasoft.com/products/java-media/3D/flythrough.html), I obtained a bit over 30 frames per second, which is quite acceptable.

The choice of graphics card is important, and a rough guide is to go for one with OpenGL support -- this will have a knock-on effect on the speed of Java 3D using OpenGL A list of cards that support OpenGL can be found at http://www.OpenGL.org/users/apps_hardware/accelerators.html

Another solution is to search the Java 3D newsgroup and mailing list, both described below, for discussions about cards.

A.2. Software

J2SE should be installed before Java 3D. You can get the latest version from http://java.sun.com/j2se/downloads.html (v.1.4.1 as I write). Before installing it, remove any previous versions you might have on your machine.

You should also download the API documentation and tutorial, which are essential reading. Get these from http://java.sun.com/docs/.

The latest version of the Java 3D SDK (currently v.1.3.1) can be found at http://www.javasoft.com/products/java-media/3D/. On the download page (http://java.sun.com/products/java-media/3D/download.html) there is a choice between different platforms and OpenGL/DirectX; the OpenGL version is reportedly more stable than the DirectX implementation. While on the download page, the "implementation documentation" should be retrieved: it describes both the core API and the utility classes.

When Java 3D is installed, various JAR files (and DLLs in the Windows version) are added to the Java SDK, in subdirectories jre\lib\ext and jre\bin. The JRE, typically located in "C:\Program Files\Java\j2re", is also updated.

A \java3d folder, containing about 40 small-to-medium examples, is added to the \demo subdirectory of the Java SDK. They are a great help, but somewhat lacking in documentation, and some of them need rewriting to use more recent features of Java 3D and the utility classes.

The Java 3D tutorial is available online at http://java.sun.com/products/java-media/3D/collateral/; you should also download the examples which are in a zipped JAR file. The tutorial is a reasonable introduction to Java 3D but is sometimes rather confusing for beginners.

The source code for the Java 3D utility package com.sun.j3d.utils is available in the file java3d-utils-src.jar, which is usually to be found in the Java SDK folder. The code for geometry like Sphere and Cylinder, and behaviours like OrbitBehaviour and MouseTranslate can be a useful source of ideas.

A.3. Java 3D Books

Two good textbooks on Java 3D:

- *Java 3D API Jump-Start*, Aaron E. Walsh, Doug Gehringer, Prentice Hall; ISBN: 0-1303-4076-6, 2001
- *Java 3D Programming*, Daniel Selman, Manning Pub.; ISBN: 1-9301-1035-9; 2002

The Walsh and Gehringer text is an excellent overview, using code snippets rather than page after page of listings. This is a good book to read in conjunction with the tutorial.

The Selman book is more advanced and has some very sophisticated examples. For the games enthusiast, there is a DOOM-like 3D world using first-person perspective, which covers keyboard navigation, world creation from a 2D map, including walls, bookcases, pools of water, flaming torches, and animated guards.

A complete online version of the book was available for free at the Manning Web site (http://www.manning.com/selman/), but has lately been replaced by an ebook version which costs money. However, the full source code is still available, and two sample chapters and errata.

A.4. Help and Advice

The *Java3D interest* mailing list is the best place to go for answers. It can be searched from http://archives.java.sun.com/archives/java3d-interest.html; subscription is also possible from that site. A searchable-only interface can be found at http://www.mail-archive.com/java3d-interest@java.sun.com/

There is a newsgroup for Java 3D, comp.lang.java.3d, which can be conveniently searched and mailed to from Google's Groups page http://groups.google.com/groups?group=comp.lang.java.3d

The official source for all things Java 3D is Sun's site http://java.sun.com/products/java-media/3D/, which has links to demos, a basic FAQ, and links to several interesting application sites such as the Virtual Fishtank.

The best independent Java 3D Web site is j3d.org (http://www.j3d.org), which is actively maintained, has a great FAQ, and a growing collection of tutorials, utilities and a code repository.

The JavaGaming.org site (http://www.javagaming.org) has some good discussion forums.

A.5. Tools

Shareware and freeware tools that can help when programming Java and Java 3D.

- EditPlus (http://www.editplus.com)
  A text editor with syntax highlighting for a number of languages, including Java and HTML. It can be configured to use the Java compiler and JVM.
- Windows Grep (http://www.wingrep.com)
  A Windows version of UNIX grep for quickly searching through the text of files. This is particularly useful when examining collections of (often poorly documented) Java source.
- Jad (http://kpdus.tripod.com/jad.html)
  A decompiler for Java class files. A decompiler can help reveal the programming techniques behind examples that come with no source code.
- J-Sprint (http://www.j-sprint.com/)
  A shareware Java profiler which can help detect slow spots in your code.
- Fraps (http://www.fraps.com/)
  A simple-to-use frame counter which can be used to measure the animation speed of code.
- ESS-Model (http://www.essmodel.com/)
  A UML modeling tool which also extracts class diagrams from existing Java code. This is a good way of getting an overview of other people's programs.
- Reverse Engineering Project (http://www.neiljohan.com/projects/reverse/)
  This 3rd year student project by Neil Johan generates UML class diagrams from Java source which show the class dependencies.
- ac3d (http://www.ac3d.org/)
  A 3D model editor and viewer; useful for preparing content for Java 3D applications.
- JCreator (http://www.jcreator.com/)
  A good, simple programming environment for Java, recommended by many Java 3D programmers. Details on how to set it up can be found in http://www.j3d.org/faq/editors.html at j3d.org.

A.6. Other Books

Java games programming:

- *Java 2 Game Programming*, Thomas Petchel, Andre LaMothe (Editor). Premier Press, December 2001. ISBN: 1-9318-4107-1. Lots of introductory Java (8 chapters). Concentrates on 2D arcade-style games; one chapter on networking; no 3D.
- *Micro Java Game Development*, David Fox, et al. Addison Wesley Professional, April 2002. ISBN: 0-6723-2342-7. Uses J2ME.

3D graphics with a gaming emphasis:

- *3D Games: Real-time Rendering and Software Technology*, Alan Watt and Fabio Policarpo, Vol. 1, Addison-Wesley, 2001.

The standard text on computer graphics:

- *Computer Graphics Principles and Practice*, James Foley, Andries van Dam, Steven Feiner, and John Hughes, Addison-Wesley, 1990, ISBN: 0-2018-4840-6.

Two games books using OpenGL and DirectX:

- *OpenGL Game Programming*, Kevin Hawkins and Dave Astle, Prima Pub., 2001, ISBN: 0-7615-3330-3
- *The Zen of Direct3D Game Programming*, Peter Walsh, Prima Pub., 2001, ISBN: 0-7615-3429-6.