Chapter 7. Introducing Java 3D

The next fifteen or so chapters will be about 3D games programming using Java 3D, Java's scene graph API which can run on top of OpenGL or DirectX.

I'll summarise the main elements of Java 3D, leaving program examples aside for the moment. Then, as in chapter 00, I'll examine Java 3D's suitability for games programming by considering the main criticisms leveled against it.

There's been an explosion in alternative ways of programming in 3D with Java. I'll list the main ones under the headings of Java bindings for OpenGL, scene graph APIs, and game engine bindings.

1. 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 more. Java 3D was developed by Sun Microsystems, and the most recent stable release is version 1.3.1. (There is a version 1.3.2, but it's a bug fix release which is still under review as I write this in July 2004.) In June 2004, Java 3D became a Sun community project, which allows external developers to submit modifications and new features.

There are two Java 3D 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.

Java 3D 1.3.1. can be downloaded from http://java.sun.com/products/java-media/3D/, together with ample documentation and a long tutorial.

1.1. The Scene Graph

Java 3D's scene graph is a directed acyclic graph (a DAG), so there's a parent-child relationship between most of its nodes, and the graph doesn't contain loops. It is possible for nodes to be shared in a graph, for example to duplicate geometry details.

A simplified scene graph for an office is shown in Figure 1. The office group contains nodes for a desk and two chairs. Each node utilizes shape and colour node components, and the chair shape information is shared.

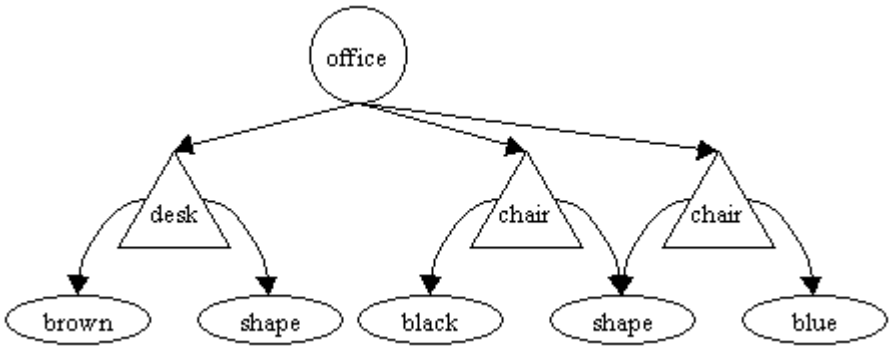

Figure 1. Scene Graph for an Office.

The choice of symbols in Figure 1 comes from Figure 2.

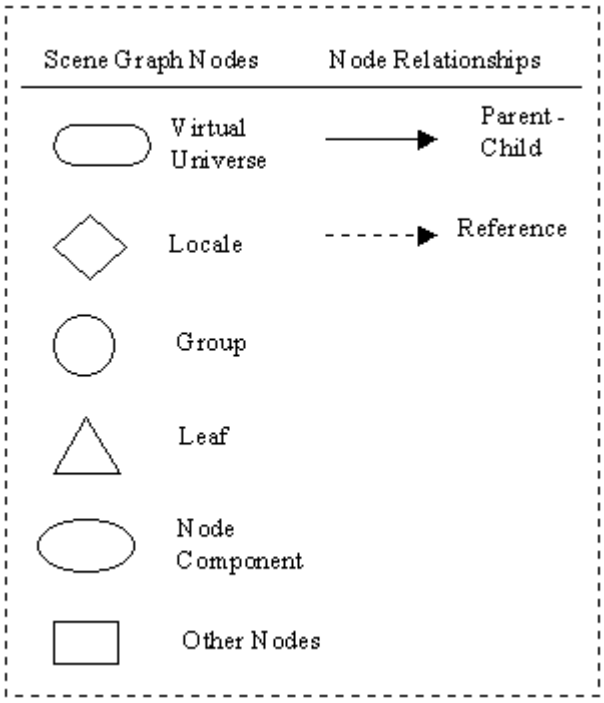

Figure 2. Scene Graph Symbols.

A shape node employs Shape3D (or its subclasses) 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.

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

Group nodes are containers for other groups or 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 from the graph at run time, while TransformGroup permits the position and orientation of its children to be changed.

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

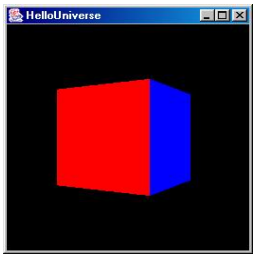

Figure 3. A Rotating Coloured Cube.

The scene graph for this application is given in Figure 4.

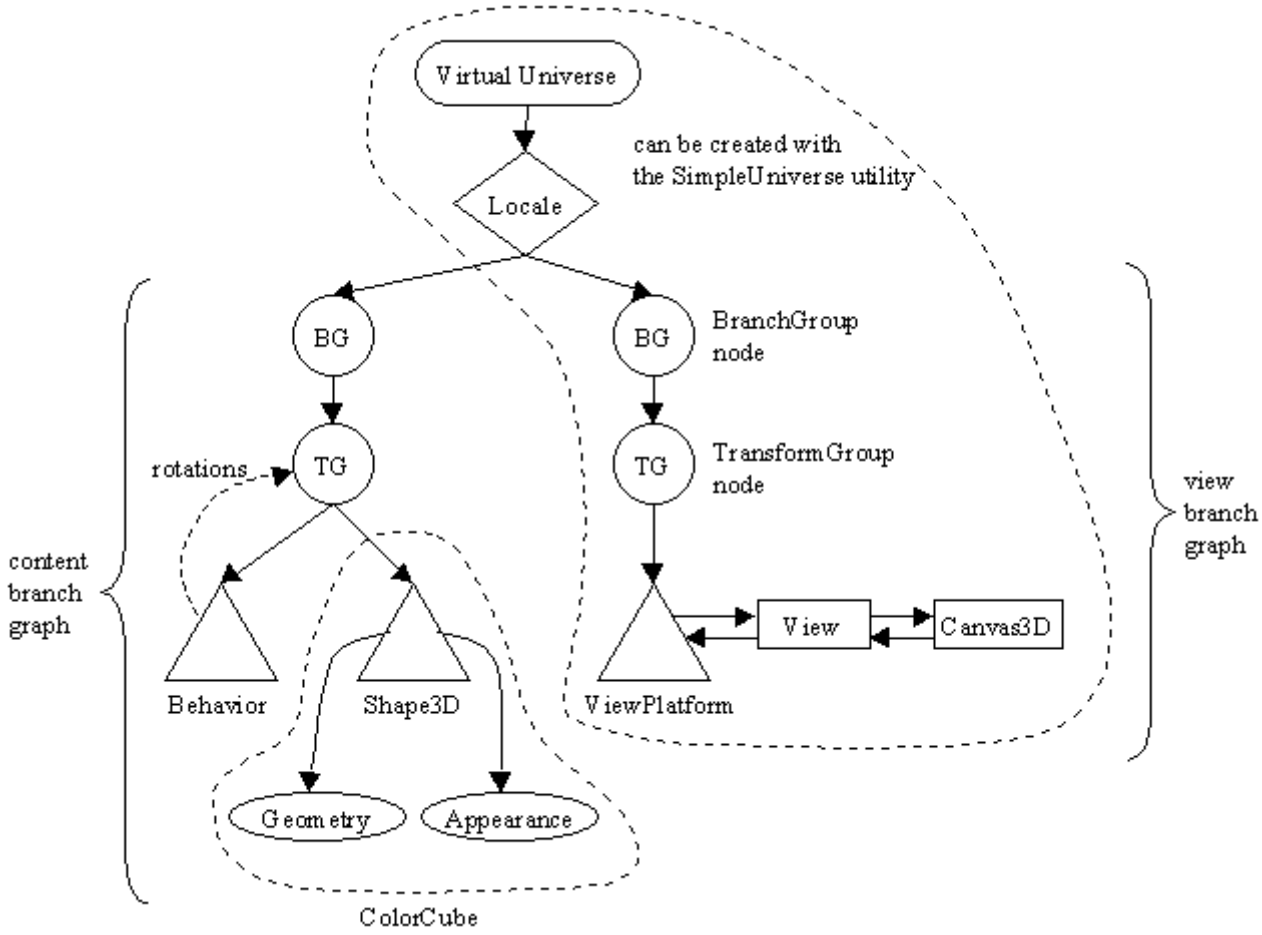

Figure 4. Scene Graph for HelloUniverse.

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

Locale acts as the scene graph's location in the virtual world.

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 ColorCube is composed from a Shape3D node and associated geometry and appearance components. Its rotation is carried out by a Behavior node which affects the TransformGroup parent of the ColorCube's shape.

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 ViewPlatform node holds 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 Java 3D's SimpleUniverse utility class, relieving the programmer of a lot of graph construction work.

2. Java 3D Strengths

The Scene Graph. 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 underlying graphics pipeline. A scene graph naturally supports complex graphical elements such as 3D geometries, behaviours, 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 utilizes Java's multithreading to carry out parallel graph traversal and rendering, both very useful optimizations.

Performance. 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 capability bits, which 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 an *immediate mode*, or partially by using a combination of immediate and retained modes (called *mixed mode*). Immediate mode gives the programmer greater control over rendering and scene management.

Novel Features. 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 utilize a range of output device, from a monitor, to stereo glasses, to CAVES.

Virtual world behaviour is coded with Behaviour nodes in the scene graph, and triggered by events. Among other things, this offers a different style of animation based on responding to events, instead of the usual update-redraw cycle.

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 ColorCube and SimpleUniverse, mouse and keyboard navigation behaviours, audio device handling, and loaders for several 3D file formats.

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 allows 2D and 3D audio output, ambient and spatialized sound. Unfortunately, there are bugs in the sound system. In the long term, it will probably be replaced by JOAL, a Java binding for a 3D audio API called OpenAL which is supported in hardware on many sound cards.

Java Integration. Java 3D is Java, so offers object orientation (classes, inheritance, polymorphism), threads, exception handling, and more. Java 3D can easily make use of other Java APIs, such as JMF and JAI.

Portability. Java 3D's implementation in Java and OpenGL or DirectX gives it a high degree of portability, with related advantages such as reduced development time and cost.

Documentation and Examples. The Java 3D distribution comes with about 40 small-to-medium examples. They are a great help, but somewhat lacking in documentation. 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/. The tutorial is a good introduction to Java 3D but sometimes rather confusing for beginners.

I recommend two Java 3D textbooks:

- *Java 3D API Jump-Start*
  Aaron E. Walsh, Doug Gehringer
  Prentice Hall, 2001
  ISBN: 0-1303-4076-6
  http://vig.prenhall.com:8081/catalog/academic/product/0,1144,0130340766,00.html
- *Java 3D Programming*
  Daniel Selman
  Manning Pub, 2002
  ISBN: 1-9301-1035-9
  http://www.manning.com/selman/

The Walsh and Gehringer text is an excellent overview, using code snippets rather than page after page of listings. It complements the Java 3D tutorial.

The Selman book is more advanced. For the games enthusiast, Selman describes a Doom-like world, utilizing first-person perspective keyboard navigation, and scene creation from a 2D map. The world contains bookcases, pools of water, flaming torches, and animated guards.

3. Criticisms of Java 3D for Games Programming

The misconceptions and complaints about Java 3D closely match those used against Java, which I discussed back in chapter 00:

- Java 3D is too slow for games programming;
- Java 3D is too high-level;
- Java 3D isn't supported on games consoles, so why bother using it;
- No one uses Java 3D to write real games;
- Sun Microsystems isn't interested in supporting Java 3D.

As in chapter 00, I'll address each of these statements in detail.

3.1. Java 3D is Too Slow for Games

There is virtually no evidence for this claim. The only serious test was done by Jacob Marner back in 2002. Marner 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. He used the C++ and GL4Java bindings for OpenGL, and version 1.3.0 beta 1 of Java 3D. His Masters thesis, "Evaluating Java for Game Development" can be obtained from http://www.rolemaker.dk/articles/evaljava/.

The C++ version was fastest, the GL4Java implementation a little slower, and Java 3D about 2.5 times slower. However, the slowdown was due to a performance bug in that version of Java 3D, and a poorly optimized scene graph. Unfortunately, the timings have not been repeated with the latest version of Java 3D, or with more recent Java bindings to OpenGL such as JOGL or LWJGL.

Marner's code highlights some 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 requires 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, so don't need to be implemented in the Java 3D application. 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.

The OpenGL applications could have been considerable faster if extensions available on the graphics card were employed.

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

3.1.1. Java 3D is Slow Because Java is Slow

Java 3D performance is often equated with Java performance: the myth of Java's slowness somehow demonstrating 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. Many successful games rely less on special effects, more on gaming characterization and story. Of course, 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.

3.2. Java 3D is Too High-Level

Java 3D's scene graph is often considered an unreasonable overhead, especially by programmers with experience of OpenGL or DirectX. Although it does introduce overheads, they should be judged against the optimizations which the scene graph brings. These can be implemented in a low-level API by an experienced programmer, but at what cost in time and maintainability?

Most large OpenGL and DirectX applications need a data structure like a scene graph, in order to manage code complexity, so the scene graph versus no scene graph argument is often invalid.

A powerful, high-level, and flexible 3D graphics API will need a scene graph *and* a way to efficiently access the graphics pipeline. These mechanisms are aimed at different levels in 3D graphics programming, called the *entity level* and the *rendering level*. An application's entity level requires a data structure for organizing the scene objects, while the rendering level handles light mapping, shadows, radiosity, vertex shading, and so on. Great games are designed at the entity level, in terms of game play, characters, scenarios, and story elements. The look and feel of a great game, the light and dark, the atmosphere, is created at the rendering level.

Although Java 3D has highly developed tools for entity level programming, it is deficit at the rendering level. For example, Java 3D cannot carry out vertex and pixel shading. Part of this is due to the desire to support Java 3D portability across OpenGL and DirectX, preventing it from making assumptions about which low-level features are present. Nevertheless, it is possible to achieve some striking rendering effects in Java 3D by employing multi-textures.

The high-level nature of the scene graph makes Java 3D code harder to tune for speed, unlike programs using OpenGL or DirectX directly. However, a programmer does have the option of moving to Java 3D's mixed or immediate modes.

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

3.3. The Lack of Console Support

The lack of a console implementation for Java 3D is a serious problem, but if Java and OpenGL are available on a game machine, then Java 3D should be readily portable. The Game Cube already uses OpenGL.

Linux for the PlayStation 2 includes OpenGL support (http://playstation2-linux.com/projects/openglstuff/). There is an old alpha version of an OpenGL for the PlayStation 2, implemented by DataPlus (http://www.dataplus.co.jp/OpenGL4ps2.html). However, the future for OpenGL on consoles, and other small devices, is OpenGL ES, a subset of OpenGL (http://www.khronos.org/opengles/).

A Java binding is being developed for OpenGL ES, managed by JSR 239 (http://www.jcp.org/en/jsr/detail?id=239). (A JSR is a Sun-sanctioned process for defining a new Java API.) Much of the work is derived from JSR 231, which will be based around JOGL and/or LWJGL (both explained below). JSR 239 is scheduled to be finished early in 2005.

3.4. No Real Java 3D Games

Java 3D has been employed in relatively few games, but they include bestsellers and award winners. I mentioned the commercial games back in chapter 00.

- *Law and Order II*, by Legacy Interactive. (http://www.lawandordergame.com/index2.htm)
- *Pernica* by Starfire Research (http://www.starfireresearch.com/pernica/pernica.html)
- *Cosm* from Navtools, Inc. (http://www.cosm-game.com/)
- *Roboforge* by Liquid Edge Games (http://www.roboforge.com).
- *FlyingGuns* (http://www.flyingguns.com/),.
- *CazaPool3D* (http://membres.lycos.fr/franckcalzada/Billard3D/Pool.html).
- *Out Of Space* (http://www.geocities.com/Psionic1981)
  An arcade game by Chris Forrester -- fly around your opponents and kill them.
- At QuakeCon 2001, Fullsail Real World Entertainment showed a Quake clone called *Jamid* and a motor racing simulator called *Java Gran Prix*.
- *The Virtual Fishtank* (http://www.virtualfishtank.com/main.html)
  A distributed simulation of a 24,000 gallon aquarium, rendered to 13 large projection screens, and running on 15 networked machines. The fish migrate from server to server as they swim from screen to screen. It was shown at the Boston Museum of Science, and the St. Louis Science Center, to teach children about emergent self-organizing behaviour in decentralized rule-based systems.
- *DALiWorld* (http://www.dalilab.com/)
  Another distributed aquatic virtual world, inhabited by autonomous artificial life.

The "Other Sites" page at j3d.org (http://www.j3d.org/sites.html) is a good source for Java 3D examples, and includes games and demos sections with many links.

The Java Games Factory (JGF), http://grexengine.com/sections/externalgames/, places its games into 2D and 3D categories, further subdivided by the 3D API being used, such as Java 3D, JOGL, and LWJGL.

A good strategy for finding Java 3D games and source code is to visit SourceForge (http://sourceforge.net/search/) and FreshMeat.com (http://freshmeat.net/) and search for keywords such as "Java", "3d", and "game".

Two very exciting Java 3D projects, which aren't really games:

- *Project Looking Glass* (https://lg3d.dev.java.net/). A prototype 3D desktop offering rotating, transparent windows, multiple desktop workspaces, and an API for developing applications. It received a great deal of attention at JavaOne in 2004.
- *The Mars Rover Mission* (http://www.sun.com/aboutsun/media/features/mars.html). Java 3D and JAI are being used to render and interpret the real-time images captured by the rover. There's also a rover simulator implemented in Java 3D, which is a sort of game.

3.4.1. Java 3D Loaders for Games

A loader is an essential tool for quickly populating a game with people, artifacts, and scenery. All the model loaders listed below are for popular games formats, and all support animation.

- Quake Loaders (http://www.newdawnsoftware.com/)
  The loaders support Id Software's Quake 2 MD2 and BSP, and Quake 3 MD3 formats. A morphing animation example using the MD3 loader can be found at http://www.la-cfd.com/cassos/test/md3/index.html
- JAVA is DOOMED (http://javaisdoomed.sourceforge.net/)
  This is a complete 3D engine, including loaders for Quake 2 MD2 and 3D Studio Max .3ds files.
- The Java XTools (http://www.3dchat.org/dev.php)
  This package offers a range of Java 3D extras, including loaders for Renderware, Caligari TrueSpace, Alias/Wavefront Maya .obj and .mtl files. Other elements include a lens flare mechanism, a text-to-texture converter, and a sky box class.
- NWN Java 3D utilities (http://nwn-j3d.sourceforge.net/)
  It handles Neverwinter Night models, including animation and emitters.
- Java 3D 3DS Loader (http://sourceforge.net/projects/java3dsloader/)
  The loader supports 3D Studio Max models, including cameras, point and directional lights, animation, and hierarchy textures.
- Anim8or Loader (http://anim8orloader.sourceforge.net/)
  It can load 3D models and scenes saved in the Anim8or file format. Anim8or is a 3D modeling and character animation program (http://www.anim8or.com/main/index.html).

- Xj3D (http://www.xj3d.org/)
  The loader implements the X3D standard, a successor to VRML 97, and provides for keyframe animation. Xj3D also contains it own OpenGL renderer, which is reportedly much faster than the one inside Java 3D.

3.4.2. Add-ons for Gaming

- Yaarq (http://www.sbox.tugraz.at/home/w/wkien/)
  Yaarq, by Wolfgang Kienreich, offers APIs for several gaming-related features, including texturing, bump maps, reflection maps, overlays, and particle systems. It also demonstrates how to achieve stable frame rates.
- Alessandro Borges' Examples (http://planeta.terra.com.br/educacao/alessandroborges/java3d.html).
  Java 3D is often criticized for lacking sophisticated lighting effects. Alessandro has developed several examples showing how to utilize bump maps to generate irregular surface lighting and cube map textures for reflection effects.
- Toon Shaders (http://www.antiflash.net/java3d/comicshader.html).
  This demonstrates how simple cartoon-style shading can be added to shapes.
- A CSG (geometry boolean operators) API by Danilo Balby Silva Castanheira is available at http://www.geocities.com/danbalby/.
- A skeletal animation and skinning system by Mark McKay can be found at http://www.kitfox.com/salamander/3d/skinAndBones.html
- Java 3D Game SDK (https://java3dgamesdk.dev.java.net/)
  The extra functionality includes a menu to let the user choose between full screen and window mode, a game mouse, and a collision box for the precise steering of objects.
- The j3d.org Code Repository (http://code.j3d.org/) includes code (or partial code) for ROAM terrain rendering, particle systems, and 2D overlays.

3.5. Sun isn't Supporting Java 3D

Perhaps this statement was true in 2003, but Java 3D is now a community project managed by the Advanced Software Development Group at Sun. If 'support' means a pool of knowledgeable people ready to offer advice, and large archives of technical information, then Java 3D has an abundance of support.

In the middle of 2003, Doug Twilleager issued the now infamous message "Java 3D 1.4 is currently in a holding pattern" (read it in full at http://www.javagaming.org/cgi-bin/JGNetForums/YaBB.cgi?board=3D;action=display;num=1054567731). Doug Twilleager is the chief architect of the Game Technologies Group at Sun, and one of the designers of Java 3D.

His message appeared just before JavaOne 2003, a conference which emphasized the JOGL, JOAL, and JInput APIs. Many people interpreted this as meaning that Java 3D was heading for the dustbin (the trash can) of history.

A possible reason for the "holding pattern" was Java 3D's development origins in the 3D Graphics Hardware Group at Sun. As graphics cards from companies such as ATI and nVidia caught up and surpassed Sun's hardware, the group started to become less profitable. Layoffs occurred, and unprofitable group projects, such as Java 3D, were given low priority.

In March 2004, Doug Twilleager was back, this time announcing that Sun was making Java 3D available through a public source license at https://java3d.dev.java.net/.

The reemergence of Java 3D is due to the work of a few key people, including Doug Twilleager, and high profile uses in Sun projects such as Mars Rover and Looking Glass. Java 3D has moved to the Advanced Software Development Group, a unit within Sun that is actively supported by upper management.

The new Java 3D project site (https://java3d.dev.java.net/) hosts the source code for Java 3D 1.3.2, a bug fix release. The current stable version is 1.3.1, which is used in this book.

Java 3D's license allows developers to download the source code, and to contribute bug fixes and utilities. Modification are allowed for research purposes, and there is a no-fee commercial license.

An expert group is being formed to define and implement future versions of the Java 3D API. An important point is that much of the implementation work is expected to come from the community, a strategy successfully employed to develop the JOGL, JOAL, and JInput APIs.

There are four new Java 3D mailing lists:

interest@java3d.dev.java.net
announce@java3d.dev.java.net
issues@java3d.dev.java.net
cvs@java3d.dev.java.net

There is a new Java Desktop 3D Forum:

http://www.javadesktop.org/forums/forum.jspa?forumID=55

Older Java 3D information sources are still around.

The Java 3D Product Page: http://java.sun.com/products/java-media/3D/, with links to demos, a basic FAQ, and several application sites such as the Virtual Fishtank.

The Java 3D Gaming Forum: http://www.javagaming.org/cgi-bin/JGNetForums/YaBB.cgi?board=3D.

The Java 3D Interest Mailing list. 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/

The Java Technology Forum for Java 3D, http://forum.java.sun.com/forum.jsp?forum=21

The best independent Java 3D site is *j3d.org* (http://www.j3d.org); it has a great FAQ, and a large collection of tutorials, utilities, and a code repository.

The USENET newsgroup *comp.lang.java.3d* can be searched and mailed to from Google's Groups page http://groups.google.com/groups?group=comp.lang.java.3d

3.5.1. Roadmaps for the Future

It's hoped that Java 3D 1.4 will be released in the next 6-9 months (I'm writing this in July 2004), and may include programmable shaders and other features that can be quickly added. There have been discussions about using JOAL to replace Java 3D's buggy sound, and adding in character animation, terrain utilities, improved collision detection and avoidance, NURBs, CSG (geometry boolean operators), and more loaders. As the earlier "Loaders" and "Add-ons for Gaming" subsections indicate, many of these extensions already exist.

Whether these plans for version 1.4 actually bear fruit depends on the Java 3D developer community; Sun is involved mainly as a manager and adjudicator.

It's interesting to look at the future plans list for Project Looking Glass (https://lg3d.dev.java.net/), which is built on top of Java 3D. It includes some of the Java 3D wish list, and also a physics engine (perhaps using odejava, https://odejava.dev.java.net/) and a particle system.

Java 3D 1.5 (or perhaps 2.0) will take longer to arrive, since major changes are planned, such as pluggable renderers and extensibility. Athomas Goldberg, the head of the Game Technologies Group, has remarked that JOGL and JOAL may come into the picture at this stage.

The eventual release dates for Java 3D will probably be closely linked to those for Java. J2SE 1.5, now renamed to J2SE 5, will be available in the third quarter of 2004. J2SE 5.1 (code named "dragon fly") in early 2005 (6 to 9 months from now), version 6 in early 2006, version 7 in late 2007.

4. Alternatives to Java 3D

There are a large number of ways of programming in 3D with Java without employing Java 3D. I've divided them into three categories: Java bindings to OpenGL, scene graph APIs, and game engine bindings.

4.1. Java Bindings to OpenGL

Several Java OpenGL bindings have been released over the years, but they tend to be incomplete, contain bugs, lack support and documentation, and often disappear suddenly. A (slightly out of date) list of Java bindings is maintained at the OpenGL site, http://www.opengl.org/resources/java/. It includes links to JOGL, LWJGL, Java 3D, GL4Java, and a few older projects. I'll only describe the active ones here.

- GL4Java (http://gl4java.sourceforge.net/docs/overview/benefits.html)
  GL4Java, also known as "OpenGL for Java Technology", was one of the most popular OpenGL bindings until the arrival of JOGL. It can be used with AWT and Swing, and links to OpenGL 1.3 and vendor extensions.

- Lightweight Java Game Library (LWJGL) (http://www.lwjgl.org/)
  LWJGL utilizes OpenGL 1.5 with vendor extensions. It works with the latest versions of Java, so can use the NIO and full screen capabilities of J2SE 1.4. However, it doesn't support AWT or Swing. LWJGL is quite small, as the name suggests, so is suitable for devices with limited resources.

  The documentation for LWJGL is a little scanty, although ports of the Nehe OpenGL tutorials have started to appear; they're at the end of the original Nehe tutorials (http://nehe.gamedev.net).

- JOGL (https://jogl.dev.java.net/)
  JOGL is the most recent of the Java bindings for OpenGL, and promoted by the Game Technologies Group at Sun. In common with LWJGL, it supports the latest versions of Java, OpenGL and extensions. It differs in being integrated with AWT and Swing, and is considerably larger.

  JOGL and LWJGL are the main contenders for the Java OpenGL reference binding being developed as part of Sun's JSR 231 specification process (http://www.jcp.org/en/jsr/detail?id=231). JSR 231 will become the official Java binding for OpenGL. It should be finished by the start of 2005.

  There is a growing amount of tutorial material on JOGL.

  The JOGL Forum at javagaming.org is a rich information source (http://www.javagaming.org/cgi-bin/JGNetForums/YaBB.cgi?board=jogl). One very good JOGL introduction, by Gregory Pierce, can be found at http://www.javagaming.org/cgi-bin/JGNetForums/YaBB.cgi?board=jogl;action=display;num=1058027992

  Another introductory article, "Jumping into JOGL" by Chris Adamson, is at http://today.java.net/pub/a/today/2003/09/11/jogl2d.html

  The eBook "Learning Java Bindings for OpenGL (JOGL)" by Gene Davis is available from http://www.genedavissoftware.com/books/jogl/. It starts with basic JOGL examples, suitable for beginners. Several chapters and appendices are online for free.

  All the Nehe OpenGL tutorials have been ported to JOGL; they can be downloaded from http://nehe.gamedev.net or http://pepijn.fab4.be/nehe/.

  JOGL's access to OpenGL and its extensions means that it can utilize shading languages for special effects like fish eyes and spherization, and can generate various types of shadow using textures. Java 3D can mimic a few of these (see the add-ons mentioned in section 3.4.2). Crucially, JOGL can employ program code to dynamically affect the graphics pipeline, whereas Java 3D mostly uses capability bits and simple get/set methods.

  A recent posting to the Java Desktop 3D forum (http://www.javadesktop.org/forums/thread.jspa?threadID=3222) describes the use of JOGL's GLCanvas class to create a HUD (heads up display) within a Java 3D application. The canvas can be manipulated in the pre- or post- rendering phases of Java 3D's immediate mode.

4.2. Scene Graph APIs

The creation of scene graph APIs for Java is something of a growth area, greatly aided by the existence of lower-level OpenGL bindings. Most of the systems are open source.

- Xith3D (http://xith.org)
  Xith3D uses the same basic scene graph structure as Java 3D, but can also directly call OpenGL operations. This means it supports functionality like shadow volumes and vertex and fragment programs.

  This is the ideal situation for a 3D graphics API in my opinion, making Xith3D a strong contender as an alternative to Java 3D.

  Since the high-level APIs of Xith3D and Java 3D are so similar, porting Java 3D code over to Xith3D is fairly straightforward. There are versions of Xith3D that run on top of JOGL or LWJGL.

  A good tutorial for Xith3D beginners can be found at http://xith.org/tiki-index.php?page=Docs

  There is a Xith3D forum at javagaming.org:
  http://www.javagaming.org/cgi-bin/JGNetForums/YaBB.cgi

  Two problems with Xith3D are its restriction to OpenGL (there is no DirectX version), and the lack of scene graph thread safety.

- OpenMind (http://sourceforge.net/projects/open-mind/)
  The OpenMind API contains the expected elements, including hierarchical scene management and object transforms, dynamic cameras, lights and fog. OpenMind is implemented on top of JOGL (it formerly used GL4Java).
- jME Graphics Engine (http://www.mojomonkeycoding.com/)
  jME (jMonkey Engine) was inspired by the scene graph engine described in "3D Game Engine Design" by David H. Eberly (http://www.magic-software.com/Books.html). Currently, jME is built on top of LWJGL, with plans for JOGL support in the near future.
- Jist3D (http://www.jist3d.com/)
  The alpha version of this engine will be released at the end of 2004. Many of its features are described in "Practical Java Game Programming" by Clingman *et al.* (see chapter 00 for publication details).

  A key element of Jist3D is its utilization of JOGL, JOAL, and JInput. The rendering engine uses JOGL to support the scene graph, and includes utilities for working with Java 3D graphs, a collision system, and 2D overlays.
- JAVA is DOOMED (http://javaisdoomed.sourceforge.net)
  It includes loaders for Quake 2 MD2 and 3D Studio Max 3DS files. The implementation uses JOGL. The distribution includes Escape, a homage to Doom.
- Aviatrix3D (http://aviatrix3d.j3d.org/index.html)
  Aviatrix3D is a retained-mode Java scene graph API above JOGL. Its toolset is

aimed at data visualization rather than gaming, and supports CAVEs, domes, and HMDs.

- Kahlua (http://www.igd.fhg.de/CP/kahlua/)
  Kahlua is a Java wrapper for Open Inventor (http://www.sgi.com/software/inventor/) a scene graph API available on the UNIX/Linux and Windows platforms.
- jFree-D2 (http://sourceforge.net/projects/jfreed2/)
  jFree-D2 is a reincarnation of the open source Java 3D implementation JFree-D, developed by Jean-Christophe Taveau back in 1999. It provides a workable (but incomplete) implementation of Java 3D on top of GL4Java. Support for JOGL is planned in the future.

4.3. Game Engine Bindings

The following APIs emulate well-known game engines (e.g. Quake) or are Java wrappers around existing engines.

- Auriga3D (http://www.auriga3d.org/)
  Auriga3D works with Quake3 maps. There are versions on top of JOGL and LWJGL.
- Jake 2 (http://www.bytonic.de/html/jake2.html)
  Jake2 is a port of the GPL'd Quake2 game engine. It uses JOGL for the graphics and JOAL for the 3D sound. In tests, it achieves better than 85% of the speed of the original C ─ 210 fps compared to 245 fps.
- Ogre4J (http://www.sourceforge.net/projects/ogre/)
  Ogre4J is a binding for the OGRE 3D Engine (http://www.ogre3d.org/) using JNI. OGRE 3D supports both Direct3D and OpenGL, and runs on all the main desktop platforms.
- Jirr (http://sourceforge.net/projects/jirr/)
  Jirr is a binding of the open source Irrlicht game engine (http://irrlicht.sourceforge.net/), which is written in C++. Jirr is still at an early stage in development.
- Odejava (https://odejava.dev.java.net/)
  Odejava is a binding around ODE, the Open Dynamics Engine, an industrial quality library for simulating articulated rigid body dynamics. Typical applications include ground vehicles, legged creatures, and moving objects in VR environments. ODE is coded in C. The project contains tools for linking Odejava into Xith3D, OpenMind, and jME.

