CHAPTER 1

Introducing Java 3D

The Java 3D API, a scene graph API developed by Sun Microsystems, provides a collection of high-level constructs for creating, rendering, and manipulating a 3D scene graph. A scene graph makes 3D 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 supports complex graphical elements such as 3D geometries, lighting modes, picking, and collision detection.

This chapter gives an overview of the main features and strengths of Java 3D, leaving program examples aside for the moment, and addresses the common complaints about the API (which are unfounded).

URLs are included that lead to more information, games, model loaders, games-related libraries, and alternative scene graph systems.

Overview of Java 3D

Prior to the most recent release, version 1.5, there were two Java 3D variants: one implemented on top of OpenGL, the other above DirectX Graphics. OpenGL (the Open Graphics Library) is a cross-language, cross-platform API for 3D (and 2D) computer graphics. The DirectX Graphics API supports a rendering pipeline quite similar (in concept) to OpenGL, describing all geometry in terms of vertices and pixels. It's part of DirectX, a collection of related gaming APIs aimed at Microsoft Windows (`http://www.microsoft.com/directx`). The other APIs support 3D audio, networking, input device integration, multimedia, and installation management.

Java 3D on Windows uses the OpenGL renderer by default and requires OpenGL 1.3 or later. DirectX rendered can be switched on by the user with a -Dj3d.rend=d3d command-line argument, and requires DirectX 9.0 or later.

A JOGL rendering pipeline was added to Java 3D 1.5, making it easier to develop future Mac versions. JOGL is a thin layer of Java over OpenGL, effectively hiding some of the low-level variations in the OpenGL API across different OSes. The JOGL pipeline also offers a lightweight JCanvas3D class as an alternative to the heavyweight Canvas3D class. Canvas3D is utilized as a drawing surface for rendering a 3D scene but can be tricky to combine with lightweight Swing GUI components; I explain how to safely use Canvas3D in Chapter 2.

One of the main aims of Java 3D 1.6 (due out by the summer of 2008) is to use the JOGL binding to combine OpenGL and Java 3D rendering more closely.

The principal Java 3D web site is `https://java3d.dev.java.net/`, where Java 3D can be downloaded as a binary installation for various platforms; for example, I retrieved the final release version 1.5 for Windows. Java 3D should be installed after Java SE, with Java SE 5 or later the recommended version. The API documentation and examples are separate (but essential) downloads from the same site.

The Java 3D roadmap site (`http://wiki.java.net/bin/view/Javadesktop/Java3DRoadmap`) details plans for versions 1.5.1, 1.6, and beyond. For instance, 1.5.1 will mainly add support for Microsoft Vista.

Overview of the Scene Graph

Java 3D uses a scene graph to organize and manage a 3D application. The underlying graphics pipeline is hidden, replaced by a treelike structure built from nodes representing 3D models, lights, sounds, the background, the camera, and many other scene elements.

The nodes are typed, the main ones being Group and Leaf nodes. A Group node has child nodes, grouping the children so that operations such as translations, rotations, and scaling can be applied en masse. Leaf nodes are the leaves of the graph (did you guess that?), which often represent the visible things in the scene, such as 3D shapes, but may also be nontangible entities, such as lighting and sounds. Additionally, a Leaf node may have node components, specifying color, reflectivity, and other attributes of the Leaf.

The scene graph can contain behaviors, represented by nodes holding code that affects other nodes in the graph at runtime. Typical behavior nodes move shapes, detect and respond to shape collisions, and cycle lighting from day to night.

The term *scene graph* is used, rather than *scene tree*, because it's possible for nodes to be shared (i.e., have more than one parent).

Before looking at a real Java 3D scene graph, look at Figure 1-1 that shows how the scene graph idea can be applied to defining the contents of a living room.

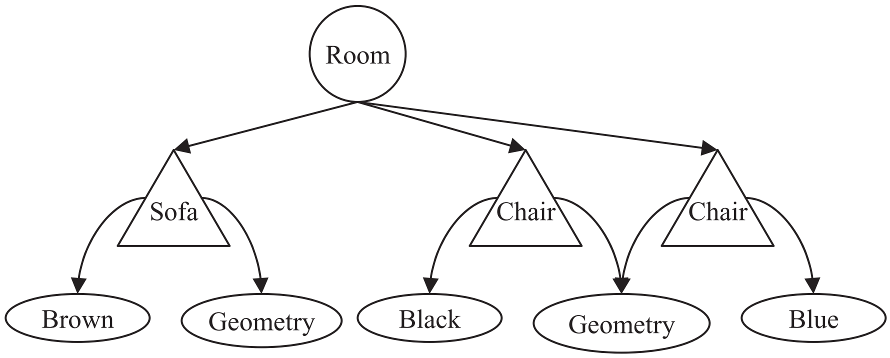

Figure 1-1. *Scene graph for a living room*

The room Group node is the parent of Leaf nodes representing a sofa and two chairs. Each Leaf utilizes geometry (shape) and color node components, and the chair geometry information is *shared*. This sharing means that both chairs will have the same shape but be different colors.

The choice of symbols in Figure 1-1 comes from a standard symbol set (shown in Figure 1-2), used in all of this book's Java 3D scene graph diagrams. I explain the VirtualUniverse and Locale nodes and the Reference relationship in the "HelloUniverse Scene Graph" subsection.

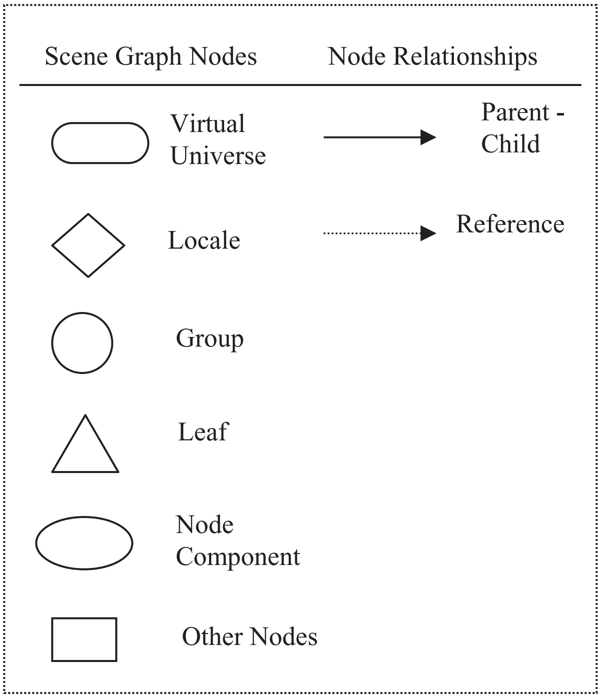

Figure 1-2. *Scene graph symbols*

Some Java 3D Scene Graph Nodes

The Java 3D API can be viewed as a set of classes that subclass the Group and Leaf nodes in various ways. The Leaf class is subclassed to define different kinds of 3D shapes and environmental nodes (i.e., nodes representing lighting, sounds, and behaviors).

The main shape class is called Shape3D, which uses two node components to define its geometry and appearance; these are represented by classes called Geometry and Appearance.

The Group class supports basic node positioning and orientation for its children, and is subclassed to extend those operations. For instance, BranchGroup allows children to be added or removed from the graph at runtime, while TransformGroup permits the position and orientation of its children to be changed.

The HelloUniverse Scene Graph

The standard first code example for a Java 3D programmer is HelloUniverse (it appears in Chapter 1 of Sun's Java 3D tutorial at `http://java.sun.com/developer/onlineTraining/java3d` and in the Java 3D examples collection at `https://java3d.dev.java.net/`). The HelloUniverse program displays a rotating colored cube, as shown in Figure 1-3.

Figure 1-3. *A rotating colored cube*

The scene graph for this application is given in Figure 1-4.

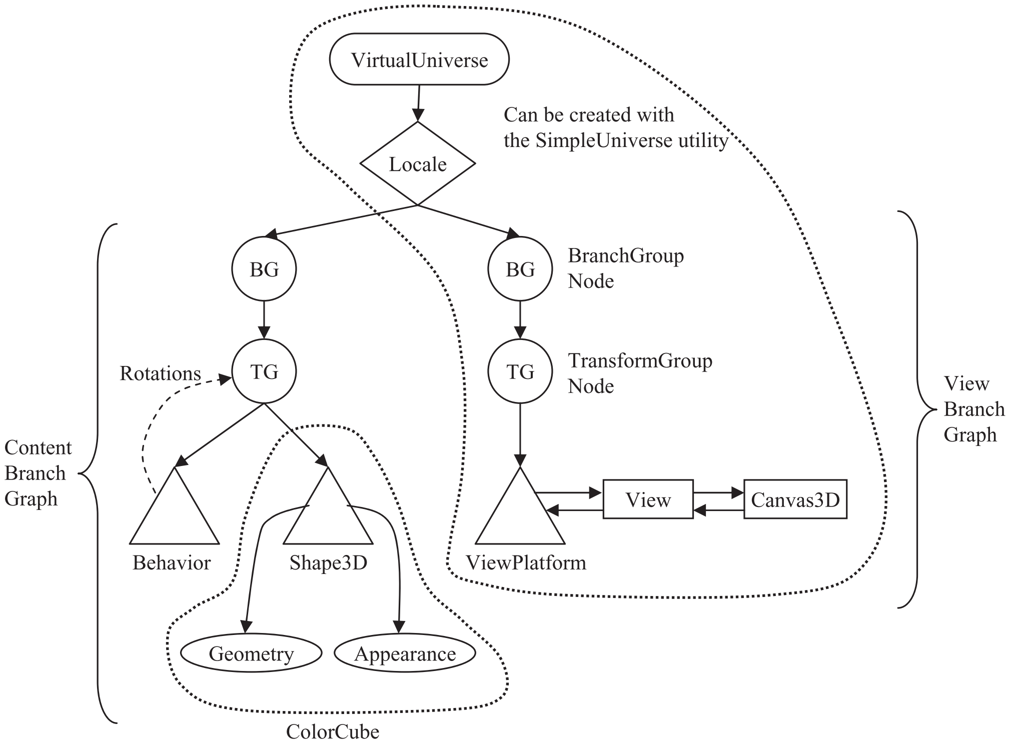

Figure 1-4. *Scene graph for HelloUniverse*

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 world's 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 branch below Locale is the *view branch graph*, and specifies the user's position, orientation, and perspective as he looks into the virtual world from the physical world (e.g., from in front of a monitor). The ViewPlatform node stores 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 as 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.

Java 3D Strengths

The core strengths of Java 3D are its scene graph, performance, collection of unique features, Java integration, and extensive documentation and examples.

The Scene Graph

The scene graph has two main advantages: it simplifies 3D programming and speeds up the resulting code. The scene graph hides low-level 3D graphics elements and allows the programmer to manage and organize a 3D scene. The scene graph supports a wide range of complex graphical elements.

At the Java 3D implementation level, the scene graph is used to group shapes with common properties and carry out view culling, occlusion culling, level of detail selection, execution culling, and behavior pruning—all optimizations that 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 Graphics.

Some programmer-specified scene graph optimizations are available through capability bits, which state what operations can/cannot be carried out at runtime (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, via the mixed mode. Immediate mode gives the programmer greater control over rendering and scene management, but isn't often required. Mixed mode "mixes" the immediate and retained modes so a program can utilize lower-level rendering and a scene graph together. Retained mode programs (the default Java 3D coding style) only use the scene graph API. Almost all of my Java 3D examples employ retained mode, except in Chapter 9, which looks at mixed mode rendering.

Unique 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 devices, from a monitor to eyeglasses with stereo displays to CAVEs (rooms with projected images covering every wall).

Virtual world behavior is coded with Behavior nodes in the scene graph and triggered by events. Among other things, Behavior nodes offer a different style of animation based on responding to events instead of the usual update-redraw cycle used in most games programs.

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 behaviors, 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 (new I/O capabilities present since version 1.4) and networking, it facilitates the ready transfer of large quantities of data between applications, such as multiplayer games.

Java 3D supports both of the popular programmable shader languages, OpenGL's GLSL (the default), and NVIDIA's Cg. This allows the programmer to create specialized rendering effects—such as bump mapping and shadows—very easily.

Java 3D offers 2D and 3D audio output, with ambient and spatialized sound. Unfortunately, there are bugs in the sound system, so spatialized sound isn't available by default in Java 3D 1.5. Version 1.6 will probably include a JOALMixer class instead, which will act as a programming interface to a JOAL-based audio device. JOAL is a Java binding for a 3D audio API called OpenAL, which is supported by many sound cards. As of January 2007, there's a partially completed version of JOALMixer at the j3d-incubator site (`https://j3d-incubator.dev.java.net/`). In Chapter 13, I develop a JOAL sound class that can be used with Java, Java 3D (in Chapter 14), and JOGL (Chapter 17).

Java Integration

Java 3D is Java, so it offers object orientation (classes, inheritance, polymorphism), threads, exception handling, and more. Java 3D can easily make use of other Java APIs, such as Java Media Framework (JMF) and Java Advanced Imaging (JAI). The JMF includes mechanisms for playing audio and video segments and can be extended to support new forms or audio and video (`http://java.sun.com/products/java-media/jmf`). JMF is utilized alongside Java 3D in Chapters 9 and 10. JAI provides many additional image processing features, including more than 100 imaging operators, tiling of large images, network-based capabilities, and the means to add new forms of image processing (JAI can be found at `http://java.sun.com/products/java-media/jai`).

Documentation and Examples

The Java 3D distribution doesn't come with any programming examples, which are a separate download at `https://java3d.dev.java.net/`; the 40-plus small to medium examples are a great help, but somewhat lacking in documentation.

Sun's Java 3D tutorial, available at `http://java.sun.com/developer/onlineTraining/java3d`, is quite old, dating from 2002 and focusing on version 1.2, but still useful for understanding Java 3D's core features.

Ben Moxon has a very nice introductory Java 3D tutorial based around getting a 3D figure to move over a hilly terrain (`http://www.benmoxon.info/Java3d/`); it's called "The Little Purple Dude Walks."

I recommend three Java 3D textbooks as supplemental reading:

- *Java 3D API Jump-Start* by Aaron E. Walsh and Doug Gehringer (Prentice Hall, 2001)
- *Java 3D Programming* by Daniel Selman (Manning Publications, 2002)
- *Java Media APIs: Cross-Platform Imaging, Media, and Visualization* by Alejandro Terrazas, John Ostuni, and Michael Barlow (Sams, 2002)

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.

Terrazas is involved in virtual reality research and business, so there's a heavy slant in the 3D part of his book toward less common topics such as sensors and head tracking, and a bit on CAVEs. There's an example combining Java 3D and JMF to create a 3D video chat room.

Criticisms of Java 3D for Games Programming

The misconceptions and complaints about Java 3D closely match those used against Java, which I mention in the introduction to this book. I'll focus on three criticisms specific to Java 3D:

- Java 3D is too high-level.
- No one uses Java 3D to write real games.
- There is a lack of support for Java 3D.

Java 3D's Level of Abstraction

Java 3D's scene graph is often considered a source of unreasonable overheads, especially by programmers with experience of OpenGL or DirectX. Although it does introduce some overheads, they should be judged against the optimizations that the scene graph brings. These optimizations 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 needs both a scene graph and a way to efficiently access the graphics pipeline. These two mechanisms are aimed at different levels in 3D graphics programming, sometimes 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 is best known for entity-level programming (via the scene graph), it also supports rendering-level coding. For example, Java 3D can perform vertex and pixel shading using either the GLSL or Cg shading languages. It's also possible to achieve some striking effects by employing multitextures, rendering attributes, and blending functions, as I show in Chapter 6.

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.

Java 3D Games

Java 3D has been employed in relatively few games, but they include best sellers and award-winners, including the following:

- *Law and Order II* (`http://www.lawandordergame.com`). This is a detective game based on the TV show.
- *Pernica* (`http://www.starfireresearch.com/pernica/pernica.html`). This is a sword-and-sorcery role-playing world.
- *RoboForge* (`http://www.roboforge.com/`). Build and fight robots.
- *FlyingGuns* (`http://www.flyingguns.com/`). World War I planes dogfight in the skies.
- *CazaPool3D* (`http://cazapool3d.sourceforge.net/cazapooljws/Pool.html`). Want to shoot some pool?
- *Cassos* (`http://www.la-cfd.com/cassos/english/index.php`). Race monkeys with a dragon.
- Games by IMI Labs (`http://www.imilabs.com/`). Several impressive game demos over the years include *Java 3D Grand Prix* (a racing game), *JAMID* (a first-person shooter in the *Quake* mold), and *Underworld Assault* (a two-person fighting game). The latest is *Cosmic Birdie*, a networked racing game using the Sun Game Server.
- *Out of Space* (`http://www.geocities.com/Psionic1981/oos.html`). This features jet-packed flying and shooting with explosions.
- *StreamGlider* (`http://www.streamglider.net/`). Blow up space gliders.
- *The Virtual FishTank* (`http://www.virtualfishtank.com/main.html`). This is a distributed simulation of a 24,000 gallon aquarium, rendered to 13 large projection screens and running on 15 networked machines.

The "Useful Sites" page at `http://www.j3d.org/sites.html` is a good source for Java 3D examples and includes a games and demos sections. Other good general lists of code examples can be found at `http://java3d.virtualworlds.de/projects.php` and the Java 3D Users Wiki at `http://wiki.java.net/bin/view/Javadesktop/Java3DUsers`.

The games listed at the Java Games Factory, `http://javagamesfactory.org/`, can be viewed by technology categories, which include several 3D-related sections.

The third-year computer graphics course in the Computer Science Department of the University of Applied Sciences in Biel, Switzerland, maintains a site of student projects using Java 3D (`https://prof.hti.bfh.ch/swc1/DemoJ3D/`). Many of them are games, including *Battleship3D-Net* (a networked version of the *Battleships* game), *Billard-3D* (pool), *Glymp3D* (role-playing action), *JBomba* (based on *Bomberman*), and *TriChess* (3D-networked chess).

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

Sun's Project Darkstar (`http://games-darkstar.dev.java.net/`) is aimed at developing tools for supporting massive multiplayer online games. The Sun Game Server is its server-side platform, and there are client APIs for C++, Java SE, and Java ME. The first two demos, *Battle Trolls* and *Cosmic Birdie* (mentioned previously), both use Java 3D.

Two very exciting Java 3D projects that aren't really games are the following:

- Project Looking Glass (`https://lg3d.dev.java.net/`). This is a prototype 3D desktop offering rotating transparent windows, multiple desktop workspaces, and an API for developing applications.
- 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 (`http://www.distant-galaxy.com/MERalizer/MERalizer.html`).

Java 3D Model Loaders for Games

A *loader* is an essential tool for quickly populating a game with people, artifacts, and scenery. A good first stop for loader listings is the j3d.org page at `http://java3d.j3d.org/utilities/loaders.html`. All the model loaders in the following list are for popular game formats, and they all support animation:

- Quake Loaders (`http://www.newdawnsoftware.com/`). The loaders support id Software's *Quake II* MD2 and BSP, and *Quake III* MD3 formats. A morphing animation example using the MD3 loader can be found at `http://www.la-cfd.com/cassos/test/md3/index.html`.
- The Java XTools (`http://www.3dchat.org/dev.php`). This package offers a range of Java 3D extras, including loaders for Renderware, Caligari TrueSpace, and Wavefront OBJ and MTL files. Other elements include a lens flare mechanism, a text-to-texture converter, and a sky box class.
- NWN Java3d Loader (`http://nwn-j3d.sourceforge.net/`). It handles *Neverwinter Nights* models, including animation and emitters.
- 3DS Java3D Loader (`http://sourceforge.net/projects/java3dsloader/`). The loader supports 3D Studio Max models, including cameras, point and directional lights, animation, and hierarchy textures.
- Anim8orLoader (`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 its own OpenGL renderer, which is reportedly much faster than the one inside Java 3D.

Extra Gaming Libraries

Three places to start looking for additional games-related libraries/APIs for Java 3D are the following:

- The j3d-incubator project (`https://j3d-incubator.dev.java.net/`) is for sharing examples and utility code.
- The j3d.org Code Repository (`http://code.j3d.org/`) includes code (or partial code) for ROAM terrain rendering, particle systems, and 2D overlays. ROAM (Real-Time Optimally Adapting Meshes) automatically adjusts the amount of terrain detail that's rendered, depending on its distance from the user's viewpoint. This speeds up rendering times considerably for large landscape.

- The Java Games Middleware project (`https://games-middleware.dev.java.net/`) contains game engines and special-purpose libraries.

Specialized add-ons include the following:

- 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.
- Various lighting examples (`http://planeta.terra.com.br/educacao/alessandroborges/java3d.html`). Java 3D is often criticized for lacking sophisticated lighting effects. Alessandro Borges has developed several examples showing how to utilize bump maps to generate irregular surface lighting and cube map textures for reflection effects. Mike Jacobs wrote a JDJ article on bump mapping at `http://mnjacobs.javadevelopersjournal.com/bump_mapping_in_java3d.htm`, and Joachim Diepstraten has one at `http://java3d.j3d.org/tutorials/quick_fix/dot3_bumps.html`.
- Comic/cel shaders (`http://www.antiflash.net/java3d/comicshader.html`). This demonstrates how simple cartoon-style shading can be added to shapes.
- A constructive solid geometry (CSG) API (`http://www.geocities.com/danbalby/`). A new shape is created using set operations (e.g., union, intersection, difference) applied to groups of simpler shapes.
- jgeom (`https://jgeom.dev.java.net/`). This is a 3D geometry library including NURBS, subdivision surfaces, and boolean operators. NURBS are used to define smooth curves and surfaces.
- skinandbones (`https://skinandbones.dev.java.net/`). This is a skeletal animation and skinning system.
- Odejava (`https://odejava.dev.java.net/`). This is an ODE binding for Java. ODE (Open Dynamics Engine) is a rigid body physics library suitable for simulating articulated rigid body dynamics, such as ground vehicles and legged creatures. I describe how it can be used with Java 3D in Chapter 5.
- java3dgamessdk (`https://java3dgamesdk.dev.java.net/`). The extra functionality includes a menu to let the user choose between full-screen and window mode. There is support for a game mouse, and a collision box for the precise steering of objects.
- Genesis FX (`http://www.indietechnologies.com/`). This is a commercial particle system for 3D game special effects. There's a full-featured version for personal use, which is free to the community. The techniques are explained in Mike Jacobs' JDJ article at `http://jdj.sys-con.com/read/99792.htm`.

Java 3D Support

If *support* means a pool of knowledgeable people ready to offer advice and large archives of technical information, Java 3D has an abundance of support.

Java 3D is a community project managed by the Java Media, Imaging, and Graphics (JMIG) Group at Sun, with more than 400 registered members. Java 3D's license allows developers to download the source code and to contribute bug fixes and utilities. Modifications are allowed for research purposes, and there is a no-fee commercial license.

An important aspect of a community project is that much of the implementation work comes from the community, a strategy also successfully employed to develop the JOGL, JOAL, and JInput APIs. JOGL is considered in Chapters 15 through 17, JOAL is utilized in Chapters 13, 14, and 17, and JInput, to connect a game pad to Java 3D, is in Chapters 11 and 12.

Popular Java 3D forums include the following:

- The forum in Java.Net's Java Desktop Technologies category: `http://forums.java.net/jive/forum.jspa?forumID=70`
- The Java Games forum: `http://www.javagaming.org/forums/index.php?board=14.0`
- The forum at the Sun Developer Network site: `http://forum.java.sun.com/forum.jsp?forum=21`

Other information sources include the following:

- The Wiki page at `http://wiki.java.net/bin/view/Javadesktop/Java3D`, which includes installation details, a roadmap for future versions, pointers to good books, and a great list of applications.
- The old Java 3D product page at `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 Interest Mailing list, which was closed in July 2005 but still contains years of good advice. It can be searched from `http://archives.java.sun.com/archives/java3d-interest.html` and `http://www.mail-archive.com/java3d-interest@java.sun.com/`.
- The Java 3D Programming forum hosted at Manning Publications at `http://www.manning-sandbox.com/forum.jspa?forumID=31`. This is a good place to contact Daniel Selman, the author of *Java 3D Programming* published by Manning.
- The Java 3D section of j3d.org (`http://java3d.j3d.org`), which has a great FAQ, a large collection of tutorials, utilities, and a code repository. Some of the information is a bit out of date.
- Java 3D at VirtualWorlds (`http://java3d.virtualworlds.de/`), which is a German/English site with sections on loaders, input devices, add-on libraries, documentation links, and a forum.
- The USENET newsgroup comp.lang.java.3d, which can be searched and mailed to from Google's Groups page, `http://groups.google.com/groups?group=comp.lang.java.3d`.

Alternatives to Java 3D

There are many ways of programming in 3D with Java without employing Java 3D. One major approach is to use a more direct Java binding to OpenGL, as typified by JOGL, covered in Chapters 15 through 17, or the Lightweight Java Game Library (LWJGL).

The following lists 3D graphics APIs based around the scene graph idea:

- Xith3D (`http://xith.org`) uses the same basic scene graph structure as Java 3D but can also directly call OpenGL operations. 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 and LWJGL.
- jME (jMonkey Engine) (`http://www.mojomonkeycoding.com/`) was inspired by the scene graph engine described in *3D Game Engine Design* by David H. Eberly (Morgan Kaufmann, 2000). jME is built on top of LWJGL.
- JAVA is DOOMED (`http://javaisdoomed.sourceforge.net`) includes loaders for *Quake II* MD2 and 3D Studio Max 3DS files. The implementation uses JOGL, and the distribution includes *Escape*, a *Doom*-like game.

- Aviatrix3D (`http://aviatrix3d.j3d.org/`) is a retained-mode Java scene graph API above JOGL. Its tool set is aimed at data visualization rather than gaming, and supports CAVEs, domes, and head-mounted displays.
- JView (`http://www.rl.af.mil/tech/programs/JVIEW/`) is another visualization API, supporting both 2D and 3D graphics, developed by the U.S. Air Force Research Lab. GL4Java, an older low-level Java API for OpenGL, was used to build it.
- Espresso3D (`http://www.espresso3d.com/`), a games-oriented library, includes OpenAL audio, sprites, collision detection, input, and rendering support. It's built using LWJGL.
- AgentFX (`http://www.agency9.se/products/agentfx/`) includes character animation, shader support using Cg, and texture compression. It's a commercial product, implemented using JOGL.

Summary

This chapter summarized the main features of Java 3D, including its many strengths (the scene graph, performance, its unique features, integration with Java, and an extensive body of documentation and examples). I examined the criticisms of Java 3D (i.e., being too high-level, not being used for "real" games, and having a lack of support), and found them to be groundless. I finished by briefly listing some of the alternatives to Java 3D; the most popular, JOGL, is the subject of Chapters 15 through 17.

This chapter contains many URLs leading to more information on Java 3D, including games, model loaders, and games-related libraries.