
# Creating a 3D Universe with Java3D

*by George Crawford III*

## Introduction

Java3D is part of the Java Media APIs and enables Java programmers to develop large interactive 3D universes. Java3D incorporates some of the best features of popular 3D programming toolkits, including Direct3D, OpenGL, and QuickDraw3D. The APIs also include support for 3D sound functionality.

## Creating a Java3D Application

In this article, I will describe how to build a simple Java3D program. Many of the gruesome details of Java3D programming will be deferred for a later article. For now, we just want to have fun.

### Step 1: Download and Install the JDK1.2beta 4 and Java3D SDK

The Java3D software development kit (SDK) and Java Development Kit (JDK) 1.2beta4 must be installed on your system before you can create 3D programs in Java. The most recent JDK implementation can be obtained from the Java Developer Connection (JDC) website. You will have to join the JDC before you can obtain either the JDK 1.2beta4 or the Java3D SDK. Registration is free. Check out http://www.javasoft.com/products for JDK 1.2beta4 download information. The latest Java3D implementation, currently version 1.1 Beta 1, can be downloaded from http://www.javasoft.com/products/java-media/3D.

The Java3D implementation that I use is built on top of Direct3D. Since this release is still in beta, there will undoubtedly be a few bugs. Also, each separate implementation will demonstrate quirks that cannot be duplicated in other implementations (e.g., OpenGL versus Direct3D). Bug reports are posted periodically at the Java Developer Connection Java3D page.

### Step 2: Create the Simple3D.java file

A Java3D program file has the same attributes as any other Java application. No new syntax or file naming scheme must be learned. The source code for the example used in this article can be found at the bottom of the page. Copy the source to a file called **Simple3D.java**. You will probably just copy-and-paste, but I find it easier to learn a new programming paradigm by actually *rewriting* program examples in my own particular style.

## Step 3: Import Java and Java3D Classes

This is an essential step in any Java program. Besides a few standard Java classes, a Java3D program must import class files that enable the creation of 3D universes. Table 1 lists and describes the classes required for **Simple3D.java**.

| Class | Package | Description |
|---|---|---|
| MainFrame | com.sun.j3d.utils.applet | Convenience class that enables a Java3D applet to be run as an application. |
| ColorCube | com.sun.j3d.utils.geometry | Convenience class used to create a multi-color cube. |
| SimpleUniverse | com.sun.j3d.utils.universe | Convenience class used to create a simple Java3D universe that is ready for viewing. |
| TextureLoader | com.sun.j3d.utils.image | Convenience class used to load textures. |
| Applet | java.applet | Used to create Java applet programs. |
| Frame | java.awt | Used to create a framed (windowed) Java application. |
| BorderLayout | java.awt | Used to layout components in an AWT *Container* (such as a *Frame* or *Applet*). |
| BranchGroup | javax.media.j3d | The root of a scene graph branch. |
| Background | javax.media.j3d | The background color or image that fills the window of each new rendering frame. |
| Canvas3D | javax.media.j3d | An extension of java.awt.Canvas that enables basic 3D rendering capabilities. |
| TransformGroup | javax.media.j3d | A group node that contains a transform. |
| Point3d | javax.media.j3d | Double precision floating point x, y, z coordinates. |
| Transform3D | javax.media.j3d | An abstraction that encapsulates a row-major, 4x4 double precision floating point matrix. Used for rotations, translations, and other transformations. |

Table 1: Classes Imported into Simple3D.java

The code for importing the classes presented in Table 1 is shown in Figure 1 below.

```
import java.applet.Applet;
import java.awt.Frame;
import java.awt.BorderLayout;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.image.TextureLoader;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
```

```
import javax.media.j3d.Background;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Point3D;
```

Figure 1: Import statements for Simple3D.java

## Step 4: Create the 3D Canvas and Container Layout

Java3D renders scenes on a special component called **javax.media.j3d.Canvas3D**. This component is an extension of the Abstract Window Toolkit (AWT) **java.awt.Canvas** class. While **Canvas3D** is also extensible, it is sufficient for most 3D applications.

Only a few lines of code are required to create a **Canvas3D** component and add it to a container. Figure 2 shows the code in **Simple3D.java** used to create a 3D canvas.

```
public void layoutComponents() {
    setLayout(new BorderLayout());
    canvas3D = new Canvas3D(null);
    add("Center", canvas3D);
}
```

Figure 2: Container Layout and *Canvas3D* Creation Code

The first line in the method simply sets the layout so that components are added to specific regions of the container (e.g., "East", "West", or "Center"). The second line creates the **Canvas3D** object. There is only one constructor method in **Canvas3D**, and that method requires a **java.awt.GraphicsConfiguration** object as a parameter. **GraphicsConfiguration** is an abstract class that encapsulates information about a particular graphics device. On a PC, for example, a **GraphicsConfiguration** object might represent a particular screen mode. In our example, the **GraphicsConfiguration** is set to **null**, so the default device configuration is used. The last line simply adds the 3D canvas to the current container.

## Step 5: Create the Scene Graph

In general, Java3D uses a *scene graph* for rendering purposes (as you may find out in your perusal of the Java3D documentation, this is not *always* true, e.g., *immediate mode* does not require the use of a scene graph). A scene graph is a tree structure that contains Java3D *nodes*. Each node connection represents a parent-child relationship. A scene graph is constructed in such a way that state information cannot be shared among subgraphs. This enables Java3D to render scenes concurrently.

In our example, the creation of the scene graph requires the following substeps:

1. Create a branch group object.

2. Create the background and add it to the branch group.
3. Create the transform group and transformation and add the transform group to the branch group.
4. Compile the branch group.

The next few sections explain the scene graph creation process in detail.

**Step 5-1: Create a Branch Group Object**

A **BranchGroup** object represents the root of a particular scene graph. The **BranchGroup** object is created in **Simple3D.java** in the *createSceneGraph* method as follows:

```
sceneGraph = BranchGroup();
```

Once the root of a scene graph has been defined, a scene graph can be constructed and readied for rendering.

**Step 5-2: Create the Background**

By default, the background of our 3D scene is null and void (black). Just to make the scene a little more intersting, we will add an image to the background. The *createBackground()* method in **Simple3D.java** contains the code required to add a background image to the scene. This code is shown in Figure 3.

```
public void createBackground() {
    BoundingSphere boundingSphere =
        new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);
    TextureLoader backgroundTexture =
        new TextureLoader(backgroundImage, this);
    Background background =
        new Background(backgroundTexture.getImage());
    background.setApplicationBounds(boundingSphere);
    sceneGraph.addChild(background);
}
```

Figure 3: Code to Add a Background to a Scene

The first object created is the **javax.media.j3d.BoundingSphere**. The **BoundingSphere** object specifies the spherical region in which the background will be active. There are two parameters: The sphere center location and the sphere radius. In **Simple3D.java**, the **BoundingSphere** is located at x, y, z coordinates 0.0, 0.0, 0.0, respectively, and has a radius of 100.0 meters. Java3D uses the right hand coordinate system, meaning that the x-axis points toward the right, the y axis points up, and the z-axis points out of the screen. In Java3D, the meter is the default unit of measurement.

The **com.sun.j3d.utils.image.TextureLoader** object is used to load the background image specified by the user at start-up. The specification for the **TextureLoader** constructor is as follows:

```
    TextureLoader(java.lang.String fname, java.awt.Component
                  observer)
```

Where **fname** is the name of the file that contains the texture, and **observer** is the image observer. An image observer receives asynchronous image update notifications during image construction.

The background is constructed with a **javax.media.j3d.Background** object. The **Background** constructor expects a **javax.media.j3d.ImageComponent2D** object. The **ImageComponent2D** class represents a 2D image and can be applied to 3D objects in addition to backgrounds. The **ImageComponent2D** object is obtained by calling the **TextureLoader.getImage()** method.

Once the background image has been obtained, the application bounds of the background is set with the **Background.setApplicationBounds()** method. This is necessary to specify the region in which the texture is to be applied.

Finally, the background is added to the scene graph with the **BranchGroup.addChild()** method, which is inherited from **javax.media.j3d.Group**.

**Step 5-3: Create a 3D Transformation**

A 3D transformation is the movement of a 3D point from one location to another. Rotations, scales, translations, and reflections are all examples of transformations. In Java3D, a 3D transformation is represented by the **javax.media.j3d.Transform3D** class. The scene graph group node that contains a **Transform3D** object is an instance of **javax.media.j3d.TransformGroup**.

The transformation performed in **Simple3D.java** simply rotates the cube a specified number of degrees about the x axis. The code required to perform this transformation is shown in Figure 4.

```
public void createTransformation() {
    // Create a 3D transformation that will
    // rotate the cube a specified number of degrees on the x axis
    Transform3D transform3D = new Transform3D();
    transform3D.rotX(xRotationAngle);

    // Create the transform group node and add the transformation
    // to the node. Add the transformation group to the scene graph.
    TransformGroup transformGroup = new TransformGroup(transform3D);
    sceneGraph.addChild(transformGroup);

    // Add a colored cube to the transform group node.
    transformGroup.addChild(new ColorCube(cubeScale));
}
```

Figure 4: Code to Perform a 3D Transformation

The first two lines of code create a Java3D transformation that will be used to rotate a 3D object about the x-axis.

The method used to perform the rotation, **javax.media.j3d.Transform3D.rotX()**, accepts a single double precision floating point number that specifies the angle in radians. The next two lines creates a transform group node with the specified 3D transformation. Once this group node has been properly initialized, it is added to the scene graph.

The last line of code simply adds the colored cube to the transform group node. The transform group node now has two children: the 3D transformation object and a colored cube.

**Step 5-4: Compile the Scene Graph**

After the scene graph has been constructed, it can be compiled. This is done by simply calling the **BranchGroup.compile()** method.

## Step 6: Create the Universe

Now that you have a scene graph, you are ready to create a Java3D universe. In more sophisticated 3D application, this may require some complex steps. However, since our scene is very simple, we will use the convenience classes provided by Sun. In **Simple3D.java**, the **createSimpleUniverse()** method is used to construct the universe and add it to the scene graph. The code required to do this is shown in Figure 5.

```
public void createSimpleUniverse() {
    SimpleUniverse simpleUniverse = new SimpleUniverse(canvas3D);
    simpleUniverse.getViewingPlatform().setNominalViewingTransform();
    simpleUniverse.addBranchGraph(sceneGraph);
}
```

This segment of source code readies the scene graph for rendering. First, a **javax.java-media.j3d.SimpleUniverse** object is created. This constructor method accepts a **Canvas3D** object that will be used for rendering. Next, the nominal viewing distance is set by calling the **com.sun.j3d.utils.universe.ViewingPlatform.setNominalViewingTransform()** method. Finally, the scene graph is added to the universe by calling the **com.sun.j3d.utils.universe.SimpleUniverse.addBranchGraph()** method.

## Step 7: Compile and Run the Program!

That's all there is to our **Simple3D.java** application. To compile the program, use JDK1.2beta 4 **javac**, as illustrated below:

```
javac Simple3D.java
```

The main method simply retrieves any arguments passed from the command line and creates a **Simple3D** object. Table 2 shows the valid command line arguments and their semantics.

| Argument | Java Type | Description |
|---|---|---|
| background | java.lang.String | Image to be used as a background. |

| | | |
|---|---|---|
| scale | double | Used to set how much the object is to be scaled. |
| degrees | double | The angle to rotate the cube on the xaxis. This angle is converted to radians in the **Simple3D** constructor. |

Table 2: Command Line Arguments Accepted by **Simple3D.java**

Use the standard JDK1.2beta4 **java** program to run the application, like so:

```
    java Simple3D -background m18.jpg -scale 0.5 -degrees 65.5
```

All command line arguments except background are optional.

# Conclusion

**Explore and Enjoy!**

# Source Code

```
/*
 * Simple3D.java
 *
 * Author: George Crawford III
 *
 * Permission to use and modify this source code is hereby granted provided
 * the author's name is included in all modified versions of this software.
 */
import java.applet.Applet;
import java.awt.Frame;
import java.awt.BorderLayout;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.image.TextureLoader;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Background;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Point3d;

public class Simple3D extends Applet {
    public void createSimpleUniverse() {
        SimpleUniverse simpleUniverse = new SimpleUniverse(canvas3D);
```

```
    simpleUniverse.getViewingPlatform().setNominalViewingTransform();
    simpleUniverse.addBranchGraph(sceneGraph);
}
public void createBackground() {
    BoundingSphere boundingSphere =
        new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);
    TextureLoader backgroundTexture =
        new TextureLoader(backgroundImage, this);
    Background background =
        new Background(backgroundTexture.getImage());
    background.setApplicationBounds(boundingSphere);
    sceneGraph.addChild(background);
}
public void createSceneGraph() {
    // Create the root of the scene graph
    sceneGraph = new BranchGroup();

    // Create the background for the scene and add it to the scene graph
    createBackground();

    // Create a 3D transformation and add it to the scene graph.
    createTransformation();

    // Allow Java3D to optimize the scene graph.
    sceneGraph.compile();
}
public void createTransformation() {
    // Create a 3D transformation that will
    // rotate the cube a specified number of degrees on the x axis
    Transform3D transform3D = new Transform3D();
    transform3D.rotX(xRotationAngle);

    // Create the transform group node and add the transformation
    // to the node. Add the transformation group to the scene graph.
    TransformGroup transformGroup = new TransformGroup(transform3D);
    sceneGraph.addChild(transformGroup);

    // Add a colored cube to the transform group node.
    transformGroup.addChild(new ColorCube(cubeScale));
}
public void layoutComponents() {
    setLayout(new BorderLayout());
    canvas3D = new Canvas3D(null);
    add("Center", canvas3D);
}
public Simple3D(String background, double scale, double xAngle) {
    backgroundImage = background;
```

```
        cubeScale = scale;
        xRotationAngle = (Math.PI/180)*xAngle; // Convert to radians
        layoutComponents();
        createSceneGraph();
        createSimpleUniverse();
    }
    public static void main(String[] args) {
        String backgroundImage = "m18.jpg";
        double scale = 0.5;
        double xAngle = 90.0;
        if(args.length  0)
            try {
                for(int i = 0; i < args.length; i++)
                    if(args[i].equalsIgnoreCase("-background"))
                        backgroundImage = args[++i];
                    else if(args[i].equalsIgnoreCase("-scale"))
                        scale = Double.valueOf(args[++i]).doubleValue();
                    else if(args[i].equalsIgnoreCase("-xAngle"))
                        xAngle = Double.valueOf(args[++i]).doubleValue();
             } catch(ArrayIndexOutOfBoundsException exception) {
                 handleException();
             } catch(NumberFormatException exception) {
                 handleException();
             }
        Frame frame = new MainFrame(new Simple3D(backgroundImage,
                                                 scale, xAngle),
                                                 300, 300);
    }
    public static void handleException() {
        System.err.print("Usage: java Simple3D -background [image] ");
          System.err.println("<-scale [scale]> <-xAngle [degrees]>");
          System.err.println("Where: ");
          System.err.print("\t[image] is the image to display in the");
          System.err.println("background");
          System.err.println("\t[scale] is the size of the bounding sphere");
          System.err.print("\t[degrees] is the angle in degrees ");
          System.err.println("for axis rotation");
          System.exit(0);
    }
    private String backgroundImage;     // Background image filename
    private double cubeScale;           // Used to scale the 3D cube
    private double xRotationAngle;  // The rotation angle in degrees
    private Canvas3D canvas3D;  // Used to render the 3D scene
    private BranchGroup sceneGraph; // Contains 3D scene information
}
```

## Resources

1. Sun Microsystems, Inc. The Java 3D API. Sun Microsystems, Inc. October 4, 1997.
2. Sun Microsystems, Inc. Java 3D API Documentation. July, 1998.
3. Sun Microsystems, Inc. Java 3D API Specification Version 1.1 Beta 01. Sun Microsystems, Inc. July, 1998.

Biography: George Crawford III is a software engineer at MPI Software Technology, Inc. and completing his M. S. degree in computer science at Mississippi State University. His technical areas of interest include software design, distributed object technology, Java programming, games programming, and DirectX/Direct3D programming.