Java 3D – Part 2
Positioning Objects

- So far, the examples have created objects in the same place, the center of the universe.

- In Java 3D, locations are described by using x, y, z coordinates.

  - Increasing coordinates go along the x-axis to the right, along the y-axis upwards, and along the z-axis out of the screen.

  - In the picture, x, y and z are represented by spheres, cones and cylinders.
• **Positioning Objects**
  ◦ This is called a “right-handed” coordinate system because the thumb and first two fingers of your right hand can be used to represent the three directions

  ◦ All the distances are measured in meters

  ◦ To place your objects in the scene, you start at point (0, 0, 0), and then move the objects wherever you want

  ◦ Moving the objects is a transformation, so the classes you use are: TransformGroup and Transform3D

  ◦ You add both the object and the Transform3D to a TransformGroup before adding the TransformGroup to the rest of your scene
### Positioning Objects

<table>
<thead>
<tr>
<th>Step</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a transform, a transform group and an object</td>
<td>Transform transform = new Transform3D(); TransformGroup tg = new TransformGroup(); Cone cone = new Cone (0.5f, 0.5f);</td>
</tr>
<tr>
<td>Specify a location for the object</td>
<td>Vector3f vector = new Vector3f (-.2f, .1f, -.4f);</td>
</tr>
<tr>
<td>Set the transform to move (translate) the object to that location</td>
<td>transform.setTranslation (vector);</td>
</tr>
<tr>
<td>Add the transform to the transform group</td>
<td>tg.setTransform (transform);</td>
</tr>
<tr>
<td>Add the object to the transform group</td>
<td>tg.addChild (cone);</td>
</tr>
</tbody>
</table>
Positioning Objects

- This may seem complicated, but the transform groups enable you to collect objects together and move them as one unit (similar to a composite object in Java 2D graphics).

- For example, a table could be made up of cylinders for legs and a box for the top.

- If you add all the parts of the table to a single transform group, you can move the whole table with one translation.
Positioning Objects

- The Transform3D class can do much more than specifying the coordinates of the object.

- The functions include `setScale` to change the size of an object and `rotX`, `rotY`, and `rotZ` for rotating an object around each axis (counter clockwise).

- This example displays the different objects on each axis.
• **Positioning Objects**
  
  ◦ Example:

  ```java
  import com.sun.j3d.utils.geometry.*;
  import com.sun.j3d.utils.universe.*;
  import javax.media.j3d.*;
  import javax.vecmath.*;

  public class Position {
  public Position () {
    SimpleUniverse universe = new SimpleUniverse ();
    BranchGroup group = new BranchGroup ();
  }
  ```
**Positioning Objects**

- Example:

```java
// X axis made of spheres
for (float x = -1.0f; x <= 1.0f; x = x + 0.1f)
{
    Sphere sphere = new Sphere (0.05f);
    TransformGroup tg = new TransformGroup ();
    Transform3D transform = new Transform3D ();
    Vector3f vector = new Vector3f ( x, .0f, .0f);
    transform.setTranslation (vector);
    tg.setTransform (transform);
    tg.addChild (sphere);
    group.addChild (tg);
}
```
Positioning Objects

Example:

```java
// Y axis made of cones
for (float y = -1.0f; y <= 1.0f; y = y + 0.1f) {
    TransformGroup tg = new TransformGroup ();
    Transform3D transform = new Transform3D ();
    Cone cone = new Cone (0.05f, 0.1f);
    Vector3f vector = new Vector3f (.0f, y, .0f);
    transform.setTranslation (vector);
    tg.setTransform (transform);
    tg.addChild (cone);
    group.addChild (tg);
}
```
Positioning Objects

Example:

```java
// Z axis made of cylinders
for (float z = -1.0f; z <= 1.0f; z = z + 0.1f)
{
    TransformGroup tg = new TransformGroup();
    Transform3D transform = new Transform3D();
    Cylinder cylinder = new Cylinder(0.05f, 0.1f);
    Vector3f vector = new Vector3f(.0f, .0f, z);
    transform.setTranslation(vector);
    tg.setTransform(transform);
    tg.addChild(cylinder);
    group.addChild(tg);
}
```
Positioning Objects

Example

Color3f light1Color = new Color3f (.1f, 1.4f, .1f); // green light
BoundingSphere bounds = new BoundingSphere (new Point3d
(0.0,0.0,0.0), 100.0);
Vector3f light1Direction = new Vector3f (4.0f, -7.0f, -12.0f);
DirectionalLight light1 = new DirectionalLight (light1Color,
light1Direction);
light1.setInfluencingBounds (bounds);
group.addChild (light1);
universe.getViewingPlatform ().setNominalViewingTransform ();

// add the group of objects to the Universe
universe.addBranchGraph (group);
Positioning Objects

- Example

```java
public static void main (String [] args)
{
    new Position ();
}
```
• **Appearance**
  ◦ There are many ways to change the way that objects in your scene look
  ◦ You can change their color, how much light they reflect
  ◦ You can paint them with two-dimensional images, or add rough textures to their surfaces
  ◦ The Appearance class contains the functions for making these changes
• **Appearance**
  - The simplest way of setting the appearance is by specifying only the color and the shading method
  
  - This works for setting an object to being a simple color, but to make an object look realistic, you need to specify how an object appears under lights
  
  - You do this by creating a Material
- Appearance
  - Example:
### Appearance

<table>
<thead>
<tr>
<th>Step</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an object</td>
<td>Sphere sphere = new Sphere ();</td>
</tr>
<tr>
<td>Create an appearance</td>
<td>Appearance ap = new Appearance ();</td>
</tr>
<tr>
<td>Create a color</td>
<td>Color3f col = new Color3f (0.0f, 0.0f, 1.0f);</td>
</tr>
<tr>
<td>Create the coloring attributes</td>
<td>ColoringAttributes ca = new ColoringAttributes (col, ColoringAttributes.NICEST);</td>
</tr>
<tr>
<td>Add the attributes to the appearance</td>
<td>ap.setColoringAttributes (ca);</td>
</tr>
<tr>
<td>Set the appearance for the object</td>
<td>sphere.setAppearance (ap);</td>
</tr>
</tbody>
</table>
• **Appearance**
  ◦ **Materials**
    • Materials have five properties that enable you to specify how the object appears

    • There are four colors: Ambient, Emissive, Diffuse, and Specular

    • The fifth property is shininess, that you specify with a number

    • Each color specifies what light is given off in a certain situation
• **Appearance**
  ◦ **Materials**
    • Ambient color reflects light that has been scattered so much by the environment that the direction is impossible to determine
      
        • This is created by an AmbientLight in Java 3D
        
        • Emissive color is given off even in darkness
          
            • You could use this for a neon sign or a glow-in-the-dark object
- **Appearance**
  - **Materials**
    - Diffuse color reflects light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface.
    - This is used with a DirectionalLight.
    - Specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction.
    - Shiny metal or plastic have a high specular component.
    - The amount of specular light that reaches the viewer depends on the location of the viewer and the angle of the light bouncing off the object.
• Appearance
  ◦ Materials
    • Changing the shininess factor affects not just how shiny the object is, but whether it shines with a small glint in one area, or a larger area with less of a gleaming look

    • For most objects you can use one color for both Ambient and Diffuse components, and black for Emissive (most things don’t glow in the dark)

    • If it’s a shiny object, you would use a lighter color for Specular reflections
• **Appearance**
  
  ◦ **Materials**

  - For example, the material for a red billiard ball might be:

    ```java
    // billiard ball
    //                ambient emissive diffuse specular shininess
    Material mat = new Material (red, black, red, white, 70f);
    ```

  - For a rubber ball, you could use a black or red specular light instead of white which would make the ball appear less shiny

  - Reducing the shininess factor from 70 to 0 would not work the way you might expect, it would spread the white reflection across the whole object instead of it being concentrated in one spot
• **Appearance**
  ◦ **Texture**
    • Materials may change the appearance of a whole shape, but sometimes even the shiniest objects can seem dull
    
    • By adding texture you can produce more interesting effects like marbling or wrapping a two-dimensional image around your object
    
    • The TextureLoader class enables you to load an image to use as a texture
    
    • The dimensions of your image must be powers of two, for example 128 pixels by 256
• **Appearance**
  ◦ **Texture**
    • When you load the texture you can also specify how you want to use the image
      
    • For example, RGB to use the color of the image or LUMINANCE to see the image in black and white
      
    • After the texture is loaded, you can change the TextureAttributes to say whether you want the image to replace the object underneath or modulate the underlying color
      
    • You can also apply it as a decal or blend the image with the color of your choice
• Appearance
  ◦ Texture
    • If you are using a simple object like a sphere then you will also have to enable texturing by setting the “primitive flags”
    • These can be set to Primitive.GENERATE_NORMALS + Primitive.GENERATE_TEXTURE_COORDS when you create the object
    • In case this is starting to sound a bit complicated, here is an example
    • You can experiment with the texture settings in this example and compare the results
    • You can download the picture I used from http://www.java3d.org/Arizona.jpg
• **Appearance**
  ◦ Example:

```java
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.image.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.Container;

public class PictureBall
{
    public PictureBall ()
    {
        // Create the universe
        SimpleUniverse universe = new SimpleUniverse ();

        // Create a structure to contain objects
        BranchGroup group = new BranchGroup ();
    }
}
```
• **Appearance**
  
  Example:

  ```java
  // Set up colors
  Color3f black = new Color3f (0.0f, 0.0f, 0.0f);
  Color3f white = new Color3f (1.0f, 1.0f, 1.0f);
  Color3f red = new Color3f (0.7f, .15f, .15f);

  // Set up the texture map
  TextureLoader loader = new TextureLoader
      ("K:\\3d\\Arizona.jpg", "LUMINANCE", new Container ());
  Texture texture = loader.getTexture ();
  texture.setBoundaryModeS (Texture.WRAP);
  texture.setBoundaryModeT (Texture.WRAP);
  texture.setBoundaryColor (new Color4f (0.0f,
      1.0f, 0.0f, 0.0f));

  ```
**Appearance**

Example:

```
// Set up the texture attributes
// could be REPLACE, BLEND or DECAL
// instead of MODULATE
TextureAttributes texAttr = new TextureAttributes();
texAttr.setTextureMode(TextureAttributes.MODULATE);
Appearance ap = new Appearance();
ap.setTexture(texture);
ap.setTextureAttributes(texAttr);
// set up the material
ap.setMaterial(new Material(red, black, red, black, 1.0f));
```
### Appearance
Example:

```java
// Create a ball to demonstrate textures
int primflags = Primitive.GENERATE_NORMALS + Primitive.GENERATE_TEXTURE_COORDS;
Sphere sphere = new Sphere (0.5f, primflags, ap);
group.addChild (sphere);

// Create lights
Color3f light1Color = new Color3f (1f, 1f, 1f);
BoundingSphere bounds = new BoundingSphere (new Point3d (0.0, 0.0, 0.0), 100.0);
```
• **Appearance**
  ◦ Example:

```java
Vector3f light1Direction = new Vector3f (4.0f, 7.0f, -12.0f);
DirectionalLight light1 = new DirectionalLight (light1Color, light1Direction);
light1.setInfluencingBounds (bounds);
group.addChild (light1);

AmbientLight ambientLight = new AmbientLight (new Color3f (.5f, .5f, .5f));
ambientLight.setInfluencingBounds (bounds);
group.addChild (ambientLight);
```
• **Appearance**
  ◦ Example:

    // look towards the ball
    universe.getViewingPlatform().setNominalViewingTransform();

    // add the group of objects to the Universe
    universe.addBranchGraph(group);

```java
public static void main (String[] args)
{
    new PictureBall();
}
```

**Java 3D**
• **Appearance**
  ◦ You can also set up three-dimensional textures, using shapes instead of a flat image
  ◦ Unfortunately, these do not currently work very well across different platforms
• **Appearance**
  ◦ **Special Effects**
    • Look at the AppearanceTest example that comes with Java 3D for more effects you can use.
    
    • For example, you can display objects as wireframes, display only the corners of an object, etc.
    
    • You can even make objects transparent, with the following settings:
• Appearance
  ◦ Special Effects

TransparencyAttributes t_attr = new
  TransparencyAttributes
    (TransparencyAttributes
     .BLENDED, 0.5f,
     TransparencyAttributes
     .BLEND_SRC_ALPHA,
     TransparencyAttributes
     .BLEND_ONE);

  ap.setTransparencyAttributes (t_attr);
• Questions/comments/views on post-revolutionary Russian literature and poetry?