3D Practical 1 – Basics: Checkers3D

In this practical we are going to look at the Checkers3D application from Andrew Davison's book, "Killer Game Programming in Java" (http://fivedots.coe.psu.ac.th/~ad/jg). After playing with the basic version, we will make some changes to explore 3D transformations, the effects of lighting and adding textures to objects.

Checkers3D

Copy the folder (you may need to copy files individually from within the folder) **'K:\CSCU9N6\Practicals\Checkers3D'** (also available at \\wsv\CSCU9N6\code\3D) to a sensible folder on your '**H**' drive.

Now create an **Eclipse** project (or you can use **BlueJ** or **TextPad**) in your copy of this folder, add all the files in the folder to your project and set the file **Checkers3D.java** as the 'main' file.

- **Note** that you will probably need to add the Java3D libraries to your Eclipse project manually: see instructions on the Practicals page of the module website; this may not be necessary with BlueJ and TextPad.

Compile this project and execute it. After a while the application window will appear, showing a 3D scene containing a lit, blue globe above a checkered floor, with a blue background.

You have some mouse control over the viewpoint (user or camera position). Try clicking-and-dragging in the scene with the left mouse button. You will quickly discover you can rotate the viewpoint around. If you hold the ALT key down at the same time as the mouse key, you can move the viewpoint in and out.

Lighting

In your IDE, find and open the file WrapCheckers3D.java. This file sets up the JPanel that holds the 3D scene and contains most of the code that creates the scene. All the changes we will make will be to this file (so if you mess it up you can always copy the original from the practicals folder!)

Let's first investigate the effects of lighting in this scene. It contains three light sources – one ambient light and two directional lights. These are created in the method lightScene() in the WrapCheckers3D class.

1. Remove the ambient light by commenting out the line:
   ```
   sceneBG.addChild(ambientLightNode);
   ```
2. Recompile and rerun Checkers3D. You will notice that the sphere is a little darker. In fact, it is black around the bottom edge where the two directional lights do not reach. However, the checkerboard floor and and the blue background are unchanged. Do you know why this is?
3. Now remove one of the directional lights, then both together, by commenting out the appropriate lines where the lights are added to the scene graph. What has happened?
4. Put all the lights back by uncommenting the appropriate lines.

5. Now we will play with the sphere properties, such as its shininess. So move down to the floatingSphere() method. Create a new specular colour with RGB values of (0.5f, 0.5f, 0.5f). Apply this to the sphere and try it out. Then use a value of (0.1f, 0.1f, 0.1f). This should give you an idea of what the specular colour adds to the sphere. Finally, revert to the original specular colour.

Sphere Properties

We'll now make a few more changes to the sphere, until we finally end up with a spinning world globe. At the moment a very basic constructor is used to create the sphere. We will replace that with a slightly more complex version, and then change the sphere's material so that it uses a texture.

6. Find the line where the sphere is created:

```
mySphere = new Sphere(2.0f, blueApp);
```

7. Replace this with the line:

```
mySphere = new Sphere(2.0f,Sphere.GENERATE_NORMALS,15,blueApp);
```

8. Try this out and you will see that nothing has changed.
9. All objects are constructed from vertices that are joined by straight lines. So the sphere is not truly a sphere. However, we can approximate a sphere as closely as we need by setting the number of vertices and lines used (but with performance implications!). Here the number 15 in the constructor call is the number of divisions around the circumference of the sphere. Try changing this to 5 and then to 30, and noting the appearance of the sphere in each case.
10. To see the structure of the sphere more clearly, uncomment the following line and try different numbers of divisions again. The sphere is now shown as its underlying wireframe.

```
//drawOutline(mySphere.getShape()); // display in outline
```

11. Comment out the call to drawOutline and set the number of divisions at 30. Now we will load a JPEG of the world and use this as a texture to colour the sphere. Firstly, after the lines creating the appearance blueApp, add the following lines to create an new appearance using the texture:

```
TextureLoader tex = new TextureLoader("images/earth.jpg", this);
Appearance texApp = new Appearance();
texApp.setTexture(tex.getTexture());
```

12. Modify the sphere creation line to the following, then compile and run again:

```
mySphere = new Sphere(2.0f,Sphere.GENERATE_NORMALS |
      Sphere.GENERATE_TEXTURE_COORDS,30,texApp);
```

3D Practical 1 – Basics: Checkers3D

Globe Rotation

Now we will make our world spin, using an Alpha object and a RotationInterpolator. This will require a new TransformGroup that will hold the rotation transformation required for the spin. The sphere will be added to this TG, which will in turn be added to the original translation TransformGroup, tg.

13. After the creation of the translation TG, tg, create a new TransformGroup called spinTg and call its setCapability() method to set ALLOW_TRANSFORM_WRITE so that the rotation transform can be changed at runtime (you might need to consult the Java 3D API to see how to do this.)
14. Now add the following lines of code to create an Alpha object and rotation interpolator. The rotation interpolator takes care of modifying this rotation incrementally to make the globe spin. Look up the Java 3D API to work out what rotationAlpha and rotator are actually doing. What does the schedulingBounds() do for the rotation?

```
Alpha rotationAlpha = new Alpha(-1, 5000);
RotationInterpolator rotator =
      new RotationInterpolator(rotationAlpha, spinTg);
rotator.setSchedulingBounds(bounds);
```

15. To make this work, the sphere must be added to spinTg instead of tg, and both spinTg and rotator must be added to tg. Do this, then compile and run to see the spinning globe!

Alternative Rotation

Let’s now make the globe spin around other than its own axis.

16. The globe is spinning around the y-axis. The globe itself is centred at (0,0) in x-z coordinates. Move the sphere along the z-axis a bit by changing the translation transform to (0,4,-4). Compile and run. The globe is still spinning around its own y-axis. This is because the spin rotation is applied after the translation of the globe.
17. Let’s rearrange how the translation and rotation transforms are applied. Change existing lines or add new lines of code so that now the sphere is added to the translation transform group tg (as it was initially), and tg is added as a child of spinTg, and both spinTg and rotator are children of sceneBG (instead of tg). Compile and run to see what happens.
18. You can either stick with this, or revert back to the original spin configuration before continuing.

Background Properties

Let’s now change the background so that it uses an image of a cloudy sky.

19. Find the line where the background is created in addBackground() and replace it with the following two lines. Comment out the line that sets the background colour. Compile and run to see the sky.

```
TextureLoader bgTexture = new TextureLoader("images/bg.jpg", this);
Background back = new Background(bgTexture.getImage());
```

New Objects

To finish off, try creating some new objects using the other primitive classes: Box, Cylinder and Cone (see the Java 3D API).

20. The easiest thing to try is to replace the sphere with a new geometric object.
21. To try something harder, create a new object entirely, placing it somewhere else in the world. You can reuse the Appearance objects already created for the sphere. You can also make the new object rotate and look at its wireframe view to see how the shape is constructed from triangles (note that these primitives are constructed from more than one Shape3D object, so you must specify which Shape3D to get in the call to drawOutline – check the Java 3D API for information on calling the getShape() method for each of the primitive classes.)