Assignment 2

Maximum 100 marks. Start this assignment after you have completed Unit 10. Answer five of the following questions (20 marks each).

1. Using **`MyOwn3D`** from Assignment 1,

a) Set the camera/view to in middle of one wall, two-thirds of the way up the wall and looking down at a 45-degree angle.
b) Make the camera stationary; all future moves are in relation to the angle of the base of the camera.

N/A

2. Using a sphere in **`MyOwn3D`**, capture key presses such that such pressing predefined keys moves the sphere around the room—left, right, forward, and backward.

See BlueJ project file *Question2*.

The controls are:
up arrow = -z-axis movement
down arrow = +z-axis movement
left arrow = -x-axis movement
right arrow = +x-axis movement

3. To learn more about the interaction between material colors and light colors, create a scene with red, green, and blue visual objects. Light the objects with a single-color light. What did you see? You may use **`LitSphereApp.java`** or **`MaterialApp.java`** as a starting point.

See the BlueJ project file *Question3.*
*Note: I added an ambient light to the scene to help visualize the objects. However, it has little effect on the resulting overall colour produced from the directional light.

A command line integer argument can be used to change the colour of the directional light:
0 = red, 1 = green, 2 = blue, 3/default = magenta, 4 = cyan, 5 = yellow, 6 = white

I created the objects as specified in the question. I lit the scene with several different colours to test the effects. A white light produced the expected results, with red, green, and blue objects. The default magenta light caused the red object to appear pink, while the other two objects took on differing purple shades, the green object being darker than the blue one. A red light makes the red object red, the other objects appear as a muted dark red. A green light makes all objects appear green, with the actual green object being the brightest and most vivid. A blue light makes all objects appear similarly blue, with the green object appearing the darkest. A cyan light makes all objects appear bluish-green, but the green object is more of a greenish-blue. Finally, a yellow light produces the strangest result. The

red object appears orange with a yellow reflection, the green object appears mostly green with a yellowish tinge, and the blue object is an ugly brownish-yellow colour.

It is interesting to note that all objects retain some colour and will reflect some light, even if the colour of the light has no values similar to the colour of the object. Also, the reflection area always appears as a more vibrant version of the shadowy portion of the object, no matter the colour of the light. What I found particularly strange was the fact that the sphere object seemed to be much more reflective than the cube and cylinder objects, even when all values are the same except for the colour.

4. Answer the following:

a) What is the use of texturing?

The use of texturing is to give a more real-world appearances to the geometry in a virtual scene. When a shape is created, it can have a solid color or lighting can be used to give it some effects. However, the object is still a solid mass with a solid color which may have some appearance effects due to lighting. Texturing a shape gives it an actual texture that most real-world objects have. For example, a wall of a building may be made of bricks. A brick texture can be applied to give this effect to an object. Another example is a tree. A tree is a complicated structure that can partially been seen through. Applying a tree texture to a model will give it these properties and make it appear more realistic. The advantage of using a texture is that the complicated geometry of an object can be rendered without modeling each individual 3D shape. It reduces the rendering cost of the object in terms of processing.

b) How would you animate a shadow that moves across a stationary visual object as the shadow moves?

To animate a shadow moving across a stationary visual object as the shadow moves involves the use of texturing. More specifically, multi-texturing can be used if the stationary visual object already has a texture. I will assume this is the case. To begin, I would make sure the Appearance object for the shape refers to the original texture indirectly through a TextureUnitState object. If the texture objects are referred to directly, then multi-texturing can not be used. Next, assuming that the shadow texture has already been created, I would need to ensure that the texCoordSetMap array refers to the right set of texture coordinates for both the original and shadow textures. The shadow texture could then be implemented with its own Texture and TextureAttributes objects. These would be referenced by the Appearance object of the visual object.

To animate the shadow, and Alpha object and Behavior are needed. These would be set with values to have the shadow move in the desired motion at the desired rate. It can even be used to change the shape of the shadow if necessary. The texture itself would be manipulated by changing the properties of the associated Transform3D object. However, the capabilities of the texture would need to be set first. Finally, the algorithm to move, rotate, or change the shape of the object would be written and implemented using the objects mentioned above.

c) How would you animate a shadow that moves across a visual object as the object passes through a stationary shadow?

   To animate a shadow moving across a visual object as the object moves through a shadow, a similar process to the one in the previous question can be used. However, the process is a bit more complicated because the movement of the object and the movement of the shadow across the object would need to be coordinated. This also requires the use of additional Alpha and Behavior objects for the movement of the object. To reduce this complexity, the shadow texture can created with the EYE_LINEAR field set as demonstrated in the program TexCoordGenApp.java. As shown in the program, the texture remains stationary as the object moves through it. The same property can be used on the moving visual object in this example.

5. Answer the following:

a) Turn the rectangular column in **`MyOwn3D`** into a brick column with textures.
b) Convert a sphere object in **`MyOwn3D`** into a ball by changing its texture.

See the BlueJ project file *Question5*.

6. Answer the following:

a) What are sprites? Give two examples as part of a Java3D program.
b) What are obstacles? Give two examples as part of a Java3D program.
c) How would you disallow a sprite from passing through an obstacle? Explain with an example Java3D program.

N/A

7. Perform the following:

a) Add five new spheres to **`MyOwn3D`**.
b) Change the characteristics of a sphere so that it matches the characteristics of a real tennis ball.

N/A

8. Implement the notion of gravity for the tennis ball from **`MyOwn3D`** (see the previous question 7). Place the ball on the ceiling, and let it go. Comment on how your gravity code works on the falling tennis ball.

See the BlueJ project file *Question8*.

I have to say that I am fairly pleased with how the gravity code turned out in this project. After some initial mishaps, I finally produced some results. The code uses physics based calculations for the height of the ball only. It ignores the effects of wind resistance, friction,

and small values that have little effect of the physics of the ball. The code works by calculating the initial drop using the equation for a falling body ($d = 0.5gt^2$). Once the ball hits the ground for the first time, and all subsequent times, the height is calculated using the velocity of ball ($h = v^2/2g$. The velocity is calculated based on the height that the ball has fallen and the coefficient of restitution (0.728 for a tennis ball, $v = cor * \sqrt{(2gh)}$). The time until the ball reaches the maximum height is calculated to discover when the next bounce will occur ($t = v/g$).

I discovered that the ball falling straight down was a little bit boring, so I added rotation and lateral movement to the ball after the first bounce for the ball movement to appear more realistic. The ball will bounce off the left and right walls, as well as reverse rotation when bouncing off the wall. The ball will stop bouncing when the height of the bounce becomes negligible. Once the ball reaches this point, it will continue to roll for approximately 2 seconds, and the lateral motion will decelerate as a function of time. All these lateral effects add a more pleasing motion to the ball, but are not based on any real world physics. The real-world physics applies to the vertical position of the ball only.

Overall, the ball behaves quite realistically. However, the size of the ball has been massively increased (100x) to allow it to be easily seen while the program runs. This makes the motion appear to be a little slow, because the ball seems quite massive and is not scaled to the environment. However, when the ball is re-sized to its actual dimensions, the effect appears much more natural. Nevertheless, I felt that seeing the object more easily was the more important factor, and hence, I kept the ball size large.