

22.11 Three-Dimensional Shapes

8

Throughout this chapter, we've demonstrated many two-dimensional graphics capabilities. In Java SE 8, JavaFX added several three-dimensional shapes and corresponding capabilities. The three-dimensional shapes are subclasses of `Shape3D` from the package `javafx.scene.shape`. In this section, you'll use Scene Builder to create a `Box`, a `Cylinder` and a `Sphere` and specify several of their properties. Then, in the app's controller, you'll create so-called *materials* that apply color and images to the 3D shapes.

FXML for the Box, Cylinder and Sphere

Figure 22.16 shows the completed FXML that we created with Scene Builder:

- Lines 16–21 define the `Box` object.
- Lines 22–27 define the `Cylinder` object.
- Lines 28–29 define the `Sphere` object.

We dragged objects of each of these `Shape3D` subclasses from the Scene Builder **Library's Shapes** section onto the design area and gave them the `fx:id` values `box`, `cylinder` and `sphere`, respectively. We also set the controller to `ThreeDimensionalShapesController`.⁵

5. At the time of this writing, when you drag three-dimensional shapes onto the Scene Builder design area, their dimensions are set to small values by default—a Box’s **Width**, **Height** and **Depth** are set to 2, a Cylinder’s **Height** and **Radius** are set to 2 and 0.77, and a Sphere’s **Radius** is set to 0.77. You may need to select them in the **Hierarchy** pane to set their properties.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- ThreeDimensionalShapes.fxml -->
3  <!-- FXML that displays a Box, Cylinder and Sphe
     4
      5  <?import javafx.geometry.Point3D?>
      6  <?import javafx.scene.layout.Pane?>
      7  <?import javafx.scene.shape.Box?>
      8  <?import javafx.scene.shape.Cylinder?>
      9  <?import javafx.scene.shape.Sphere?>
         10
11  <Pane prefHeight="200.0" prefWidth="510.0"
12    xmlns="http://javafx.com/javafx/8.0.60"
13    xmlns:fx="http://javafx.com/fxml/1"
14    fx:controller="ThreeDimensionalShapesControll
         15        <children>
16            <Box fx:id="box" depth="100.0" height="100
17              layoutY="100.0" rotate="30.0" width="10
         18                  <rotationAxis>
19                    <Point3D x="1.0" y="1.0" z="1.0" />
         20                  </rotationAxis>
         21          </Box>
22            <Cylinder fx:id="cylinder" height="100.0"
23              layoutY="100.0" radius="50.0" rotate="-
         24                  <rotationAxis>
25                    <Point3D x="1.0" y="1.0" z="1.0" />
         26                  </rotationAxis>
```

```
27          </Cylinder>
28      <Sphere fx:id="sphere" layoutX="430.0" lay-
29          radius="60.0" />
30      </children>
31  </Pane<
```

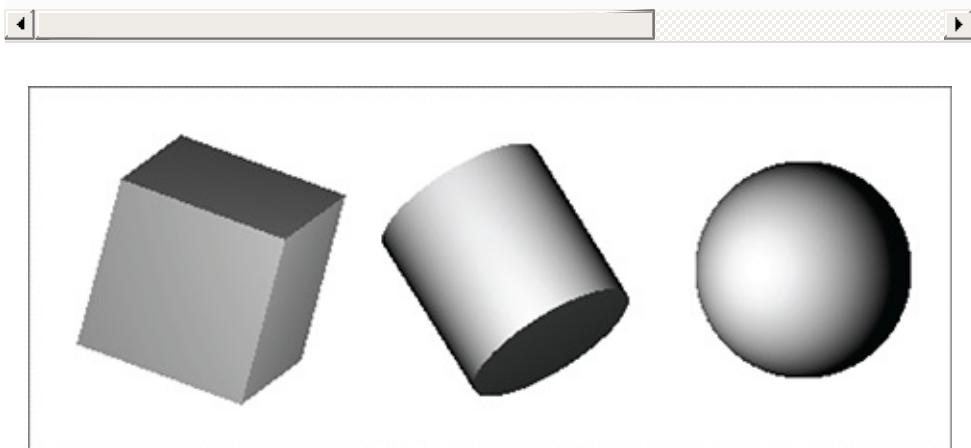


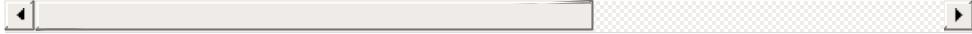
Fig. 22.16

FXML that displays a Box, Cylinder and Sphere.

Description

As you can see in the screen capture of [Figure 22.16](#), all three shapes initially are gray. The shading you see in Scene Builder comes from the scene's default lighting. Though we do not use them in this example, package `javafx.scene`'s `AmbientLight` and `PointLight` classes can be used to add your own lighting effects. You can also use camera objects to view the scene from different angles and distances. These are located in the Scene Builder **Library**'s **3D** section. For more information on lighting and cameras, see

<https://docs.oracle.com/javase/8/javafx/graphics-tutorial/>



We ask you to investigate these capabilities and use them in [Exercise 22.11.](#)

Box Properties

Configure the Box's properties in Scene Builder as follows:

- Set **Width**, **Height** and **Depth** to **100**, making a cube. The depth is measured along the z-axis which runs perpendicular to your screen—when you move objects along the z-axis they get bigger as they're brought toward you and smaller as they're moved away from you.
- Set **Layout X** and **Layout Y** to **100** to specify the location of the cube.
- Set **Rotate** to **30** to specify the rotation angle in degrees. Positive values rotate counter-clockwise.
- For **Rotation Axis**, set the **X**, **Y** and **Z** values to **1** to indicate that the **Rotate** angle should be used to rotate the cube 30 degrees around *each* axis.

To see how the **Rotate** angle and **Rotation Axis** values affect the Box's rotation, try setting two of the three **Rotation Axis** values to **0**, then changing the **Rotate** angle.

Cylinder Properties

Configure the Cylinder's properties in Scene Builder as follows:

- Set **Height** to `100.0` and **Radius** to `50`.
- Set **Layout X** and **Layout Y** to `265` and `100`, respectively.
- Set **Rotate** to `-45` to specify the rotation angle in degrees. Negative values rotate clockwise.
- For **Rotation Axis**, set the **X**, **Y** and **Z** values to `1` to indicate that the **Rotate** angle should be applied to all three axes.

Sphere Properties

Configure the Sphere's properties in Scene Builder as follows:

- Set **Radius** to `60`.
- Set **Layout X** and **Layout Y** to `430` and `100`, respectively.

ThreeDimensionalShape sController Class

Figure 22.17 shows this app's controller and final output. The colors and images you see on the final shapes are created by applying so-called materials to the shapes. JavaFX class `PhongMaterial` (package `javafx.scene.paint`) is used to define materials. The name “Phong” is a 3D graphics term—*phong shading* is technique for applying color and shading to 3D surfaces. For more details on this technique, visit

https://en.wikipedia.org/wiki/Phong_shading



```
1 // Fig. 22.17: ThreeDimensionalShapesController.
2 // Setting the material displayed on 3D shapes.
3 import javafx.fxml.FXML;
4 import javafx.scene.paint.Color;
5 import javafx.scene.paint.PhongMaterial;
6 import javafx.scene.image.Image;
7 import javafx.scene.shape.Box;
8 import javafx.scene.shape.Cylinder;
9 import javafx.scene.shape.Sphere;
10
11 public class ThreeDimensionalShapesController {
12     // instance variables that refer to 3D shapes
13     @FXML private Box box;
14     @FXML private Cylinder cylinder;
15     @FXML private Sphere sphere;
16
17     // set the material for each 3D shape
18     public void initialize() {
19         // define material for the Box object
20         PhongMaterial boxMaterial = new PhongMaterial();
21         boxMaterial.setDiffuseColor(Color.CYAN);
22         box.setMaterial(boxMaterial);
23
24         // define material for the Cylinder object
25         PhongMaterial cylinderMaterial = new PhongMaterial();
26         cylinderMaterial.setDiffuseMap(new Image(
27             "cylinder.jpg"));
28         cylinder.setMaterial(cylinderMaterial);
29
30         // define material for the Sphere object
31         PhongMaterial sphereMaterial = new PhongMaterial();
32         sphereMaterial.setDiffuseColor(Color.RED);
33         sphereMaterial.setSpecularColor(Color.WHITE);
34         sphereMaterial.setSpecularPower(32);
35         sphere.setMaterial(sphereMaterial);
36     }
37 }
```

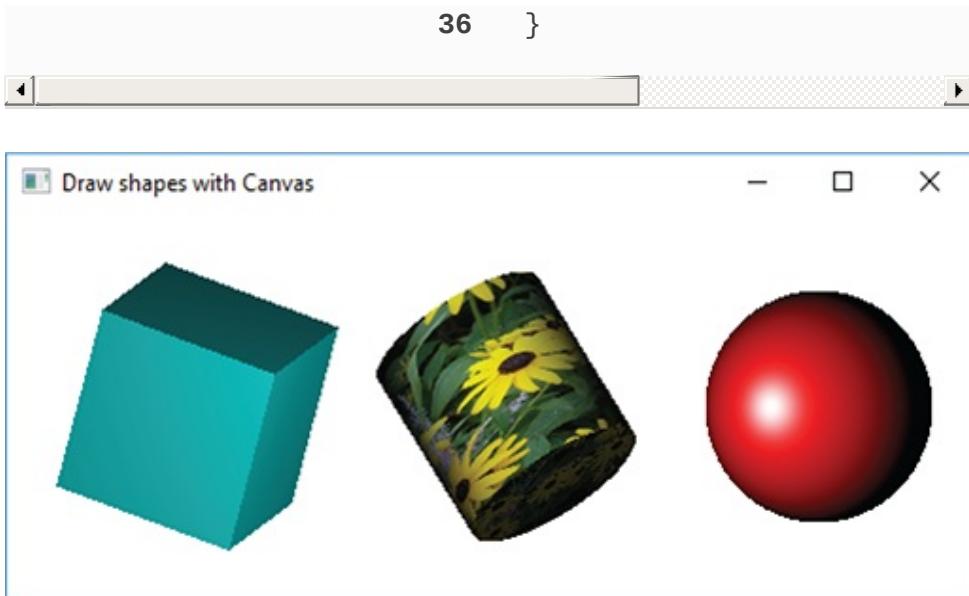


Fig. 22.17

Setting the material displayed on 3D shapes.

Description

PhongMaterial for the Box

Lines 20–22 configure and set the `Box` object's `PhongMaterial`. Method `setDiffuseColor` sets the color that's applied to the `Box`'s surfaces (that is, sides). The scene's lighting effects determine the shades of the color applied to each visible surface. These shades change, based on the angle from which the light shines on the objects.

PhongMaterial for the Cylinder

Lines 25–27 configure and set the `Cylinder` object's `PhongMaterial`. Method `setDiffuseMap` sets the `Image` that's applied to the `Cylinder`'s surfaces. Again, the scene's lighting affects how the image is shaded on the surfaces. In the output, notice that the image is darker at the left and right edges (where less light reaches) and barely visible on the bottom (where almost no light reaches).

PhongMaterial for the Sphere

Lines 30–34 configure and set the `Sphere` object's `PhongMaterial`. We set the diffuse color to red. Method `setSpecularColor` sets the color of a bright spot that makes a 3D shape appear shiny. Method `setSpecularPower` determines the intensity of that spot. Try experimenting with different specular powers to see changes in the bright spot's intensity.