Primitives and Transformations

Overview

In this tutorial we will explore primitives and transformations in the context of the jdw Java-Direct3D wrapper.

Contents

Primitives

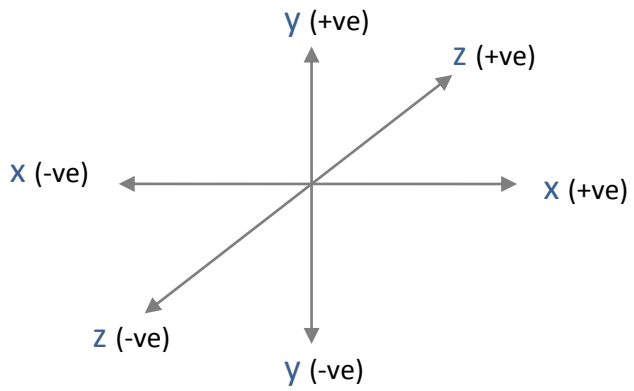
A primitive is a 3D object that you can place into a scene. Primitives are represented by `Primitive` objects, and are made up of a list of vertices (`Vertex` objects). The vertices define where edges of the shape meet. Create a Primitive object using the following line of Java code:

```
Primitive p = new Primitive(RenderMode.TRIANGLE_LIST);
```

The 3D coordinate system

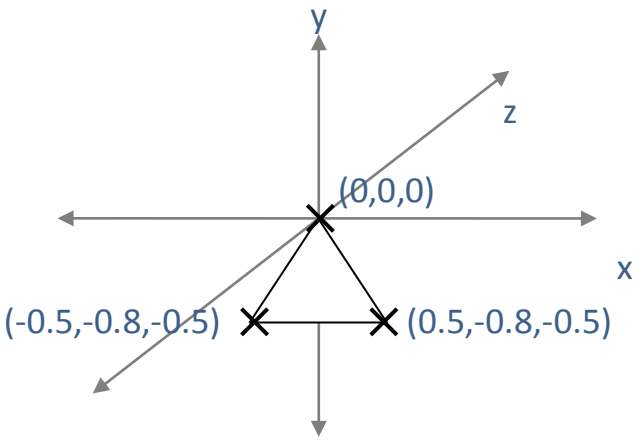
Vertices of 3D objects have three components: an X position, Y position and Z position.

The X, Y and Z-axes are defined according to the left-handed coordinate system (same as in Direct3D). The X-axis is a horizontal axis, the Y-axis is a vertical axis, and the Z-axis is perpendicular to the screen: larger Z values represent points further into the screen, and smaller Z values represent points further away from the screen.



To define a single triangle, we can use the following three vertices:

```
(0, 0, 0)
(0.5, -0.8, -0.5)
(-0.5, -0.8, -0.5)
```



As well as the X, Y, and Z-axis position, vertices can each have a color which contributes to the color of the associated primitive. If you do not set a color on the vertex explicitly, the color black is used.

Now we have all the information we need to define the `Vertex` objects in Java code. Coordinates are given as float values.

```
p.addVertex(new Vertex(0f, 0f, 0f, Color.RED));
p.addVertex(new Vertex(0.5f, -0.8f, 0f, Color.BLUE));
p.addVertex(new Vertex(-0.5f, -0.8f, 0f, Color.GREEN));
```

Note the order in which the vertices are added: they are added in clockwise order. If we added vertex (-0.5f, -0.8f, 0f) before vertex (0.5f, -0.8f, 0f), we would need to rotate the view around the Y-axis in order to be able to see the triangle.

Sample program

We can now create a simple program to see what this single triangle looks like.

1. Create a new file “Pyramids.java” and open it in your favorite text editor.
2. Create a non-public class named “PyramidFrame” extending the class `JFrame`, and add import statements.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import jdw.graphics.*;

class PyramidFrame extends JFrame {
}
```

3. Add a member variable to the `PyramidFrame` class.

```
    private Environment3D env;
```

4. Add a method named `initScene` as shown.

```
private void initScene() {
```

Create an instance of `Environment3D` which represents the 3D scene.

```
    env = new Environment3D();
```

Create a triangle and add it to the environment.

```
    Primitive triangle = createTriangle();
    env.addPrimitive(triangle);
```

Position ourselves 3 units back on the Z-axis looking towards the origin.

```
    Vector3D eyePt = new Vector3D(0f, 0f, -3f);
    Vector3D lookAtPt = new Vector3D(0f, 0f, 0f);
    env.setView(eyePt, lookAtPt);
```

Now we need to write some standard Swing code to contain the `Environment3D` object. Add the following code to complete the `initScene` method.

```
    setLayout(new BorderLayout());
    add(env, BorderLayout.CENTER);
    setTitle("Pyramids");
    setSize(800, 600);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
}
```

5. Implement the `createTriangle` method referenced earlier. This method returns a Primitive containing three `Vertex` objects representing the three vertices of the triangle.

```
private Primitive createTriangle() {
    Primitive p = new Primitive(RenderMode.TRIANGLE_LIST);
    p.addVertex(new Vertex(0f, 0f, 0f, Color.RED));
    p.addVertex(new Vertex(0.5f, -0.8f, 0f, Color.BLUE));
    p.addVertex(new Vertex(-0.5f, -0.8f, 0f, Color.GREEN));
    return p;
}
```

6. To be able to run the program we need to add a main method. The main method will create and show the window on the Swing event thread.

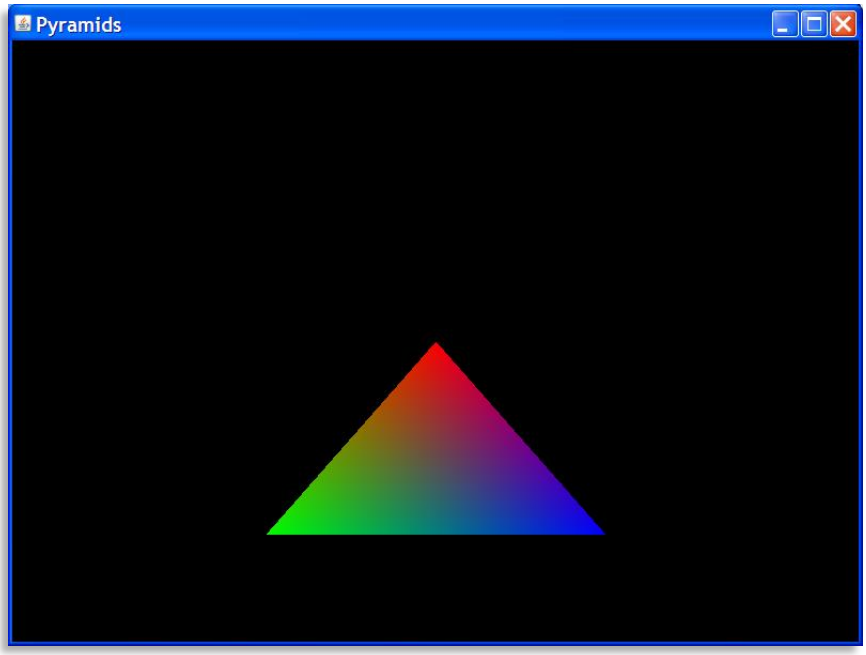
```
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            PyramidFrame frame = new PyramidFrame();
            frame.initScene();
        }
    });
}
```

7. Open a command prompt and compile Pyramids.java.

```
javac -classpath jdw.jar Pyramids.java
```

8. Run the program with the following command. A triangle is displayed in a window.

```
java -classpath jdw.jar;. PyramidFrame
```



Extending the sample

We will now extend the sample to display a set of square-based pyramids. Each pyramid will need four triangles for the sides and two triangles to make the square base.

1. Create a new class named "SquareBasedPyramid" extending `Primitive`. Add this class to Primitives.java or another file if you prefer.

```
class SquareBasedPyramid extends Primitive {
}
```

2. Add a constructor to the class which will create all the required vertices.

```
    public SquareBasedPyramid() {
        super(RenderMode.TRIANGLE_LIST);

        // Front of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.BLUE));

        // Back of pyramid
```

```
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.GREEN));
        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.GREEN));

        // Left of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.YELLOW));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.YELLOW));

        // Right of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.RED));

        // Base of pyramid (2 triangles)
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.BLUE));

        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.BLUE));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
    }
```

3. Modify the `PyramidFrame` class to create a pyramid instead of a triangle. Delete these two lines:

```
        Primitive triangle = createTriangle();
        env.addPrimitive(triangle);
```

   Replace with:

```
        SquareBasedPyramid py1 = new SquareBasedPyramid();
        env.addPrimitive(py1);
```

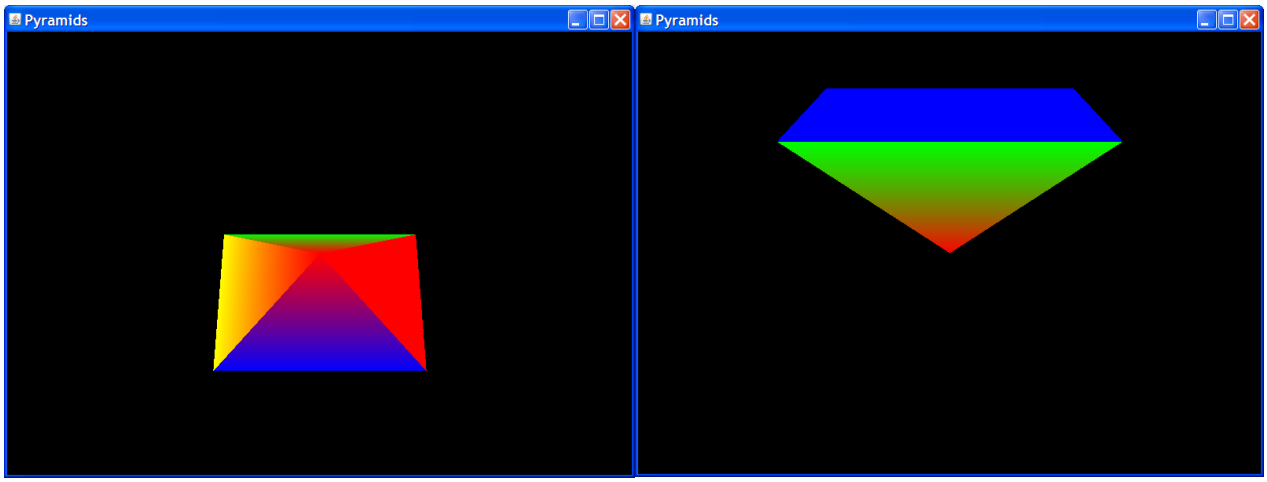
4. If you run the program now, you will see the front of the pyramid only. In the `initScene` method, just after the call to `setView`, add the following code. This `RenderListener` rotates the pyramid about the X-axis.

```
        env.addRenderListener(new RenderListener() {
            public void frameRendering(FrameRenderingEvent e) {
                Matrix worldMat = env.getWorldTransform();
                Matrix rot = Matrix.rotationYawPitchRoll(0, 0.03f, 0);
                worldMat = Matrix.multiply(worldMat, rot);

                env.setWorldTransform(worldMat);
            }
        });
```

5. Compile Pyramids.java, and run `PyramidFrame`. All faces of the pyramid can be seen as it rotates.

```
javac -classpath jdw.jar Pyramids.java
java -classpath jdw.jar;. PyramidFrame
```



Render modes

When a `Primitive` object is constructed, a render mode is given as a parameter to specify how the vertices contained in the primitive are interpreted. We have only used "triangle list" so far, but there are six render modes which are summarized below.

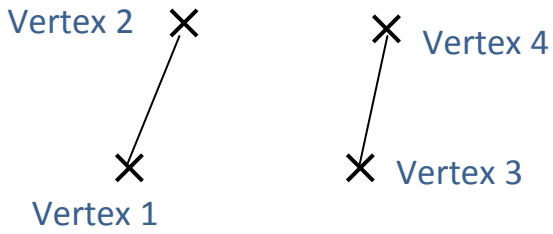
Mode	RenderMode constant	D3DPRIMITIVETYPE constant
Point list	`RenderMode.POINT_LIST`	**D3DPT_POINTLIST**
Line list	`RenderMode.LINE_LIST`	**D3DPT_LINELIST**
Line strip	`RenderMode.LINE_STRIP`	**D3DPT_LINESTRIP**
Triangle list	`RenderMode.TRIANGLE_LIST`	**D3DPT_TRIANGLELIST**
Triangle strip	`RenderMode.TRIANGLE_STRIP`	**D3DPT_TRIANGLESTRIP**
Triangle fan	`RenderMode.TRIANGLE_FAN`	**D3DPT_TRIANGLEFAN**

Point list

This render mode causes the vertices of the primitive to be rendered as standalone dots.

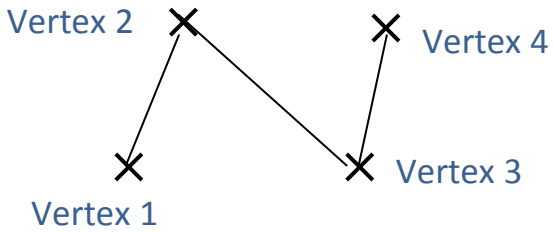
Line list

This render mode specifies that each pair of vertices added to the primitive are linked together to form a line.



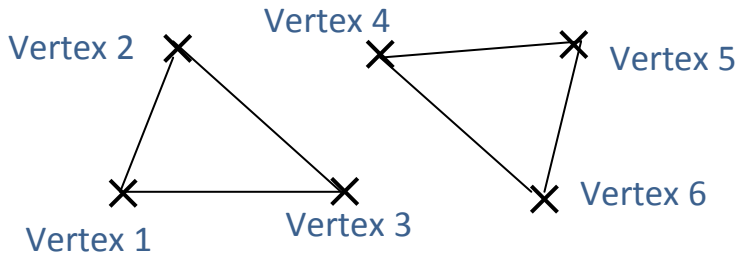
Line strip

Under the line strip render mode, each vertex is linked together to form a continuous line.



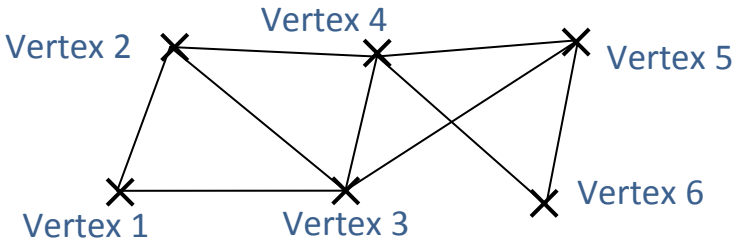
Triangle list

The triangle list mode causes each set of three vertices to be linked together to form a triangle.



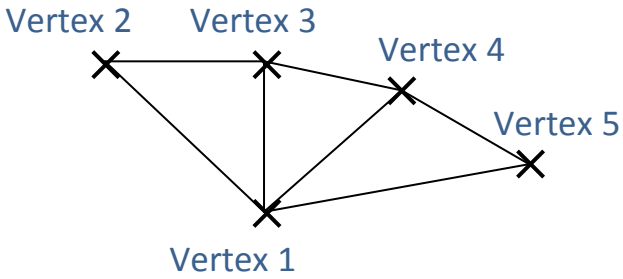
Triangle strip

The triangle strip mode specifies that the first three vertices make up a triangle, and each subsequent vertex along with the previous two, make up additional triangles.



Triangle fan

The first three vertices make up a triangle, and subsequent triangles are formed using the first vertex as a common point.



Transformations

A transformation specifies how an object or a scene's position is affected. A 3D transformation is given as a 4x4 matrix (represented by an instance of `jdw.graphics.Matrix`). You can translate, scale and rotate objects about each of the three axes. A translation and rotation for example, can be combined into one transformation by multiplying the matrices together.

Extending the sample

We will now modify the `PyramidFrame` class to perform various transformations.

1. From the `PyramidFrame` class, delete the `createTriangle` method and the `addRenderListener` method call.
2. Below the call to `addPrimitive`, insert the following line of code.

```
        transform(py1);
```

3. To the `PyramidFrame` class insert an empty `transform` method.

```
    private void transform(Primitive p) {
    }
```

4. Your class should look like the one below. Verify it compiles without errors.

```
class PyramidFrame extends JFrame {
    private Environment3D env;

    private void transform(Primitive p) {
    }

    private void initScene() {
        env = new Environment3D();
        SquareBasedPyramid py1 = new SquareBasedPyramid();
        env.addPrimitive(py1);
        transform(py1);

        Vector3D eyePt = new Vector3D(0f, 0f, -3f);
        Vector3D lookAtPt = new Vector3D(0f, 0f, 0f);
        env.setView(eyePt, lookAtPt);

        setLayout(new BorderLayout());
        add(env, BorderLayout.CENTER);
        setTitle("Pyramids");
        setSize(800, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                PyramidFrame frame = new PyramidFrame();
                frame.initScene();
            }
        });
    }
}
```

We will experiment with different transformations by modifying the `transform` method. No transformation is currently carried out so the pyramid is in its default position.

Scaling

To scale primitives, use the `Matrix.scale` method. This returns a `Matrix` object representing a scaling transformation. Scaling factors are specified for each axis allowing an object to be stretched or squashed.

Javadoc

Method Summary	
static Matrix	**scale**(float x, float y, float z)

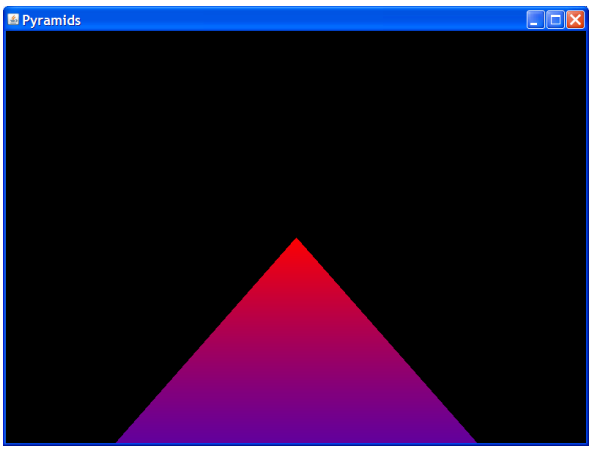
Try a scaling transformation

1. Insert the following lines of code into the `transform` method. This code creates the scaling matrix, and passes it to the pyramid.

```
Matrix m = Matrix.scale(2f, 2f, 2f);
p.setTransform(m);
```

2. Compile Pyramids.java and run `PyramidsFrame`. The pyramid is twice its original size.

```
javac -classpath jdw.jar Pyramids.java
java -classpath jdw.jar;. PyramidFrame
```



3. Repeat steps 1 and 2 with different scaling factors.

Translation

To translate primitives, use the `Matrix.translate` method. This returns a `Matrix` object representing a translation. Offsets are specified for each of the three axes.

Javadoc

Method Summary	
static Matrix	**translate**(float x, float y, float z)

Try a translation

1. Replace any code presently in the `transform` method with the following lines. This code translates the pyramid 2 units along the X-axis (right), 1 unit along the Y-axis (up), and 3 units along the Z-axis (into the screen).

```
Matrix m = Matrix.translate(2f, 1f, 3f);
p.setTransform(m);
```

2. Compile Pyramids.java and run `PyramidsFrame` (same commands as before) to see the result.

Rotation

To rotate primitives there is a choice of methods in the `Matrix` class. To rotate around a single axis use one of the `rotateX`, `rotateY`, or `rotateZ` methods. To rotate around all three axes at once, use the `rotationYawPitchRoll` method. Angles are given in radians[1].

Javadoc

Method Summary	
`static Matrix`	`rotationX(float angle)` Creates a new matrix to represent a rotation around the X-axis.
`static Matrix`	`rotationY(float angle)` Creates a new matrix to represent a rotation around the Y-axis.
`static Matrix`	`rotationYawPitchRoll(float yaw, float pitch, float roll)` Creates a matrix with the specified yaw, pitch and roll.
`static Matrix`	`rotationZ(float angle)` Creates a new matrix to represent a rotation around the Z-axis.

Try some rotations

1. Replace any code presently in the `transform` method with the following lines. This rotates the pyramid about 45 degrees around the X-axis.

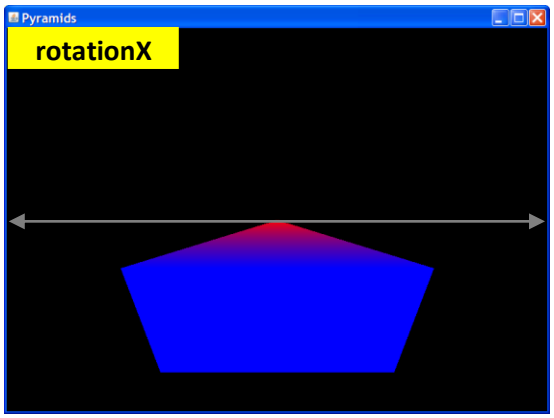
```
Matrix m = Matrix.rotationX(0.78f);
p.setTransform(m);
```

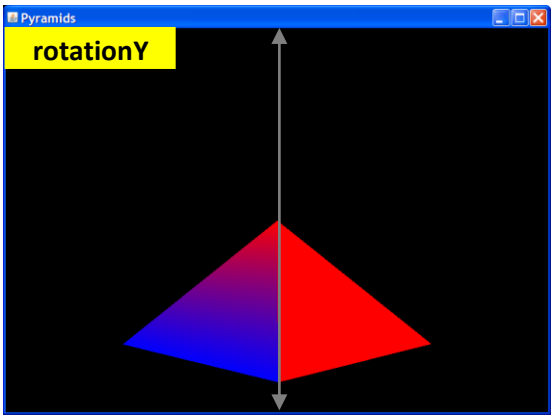
2. Compile Pyramids.java and run `PyramidsFrame` to see the result.

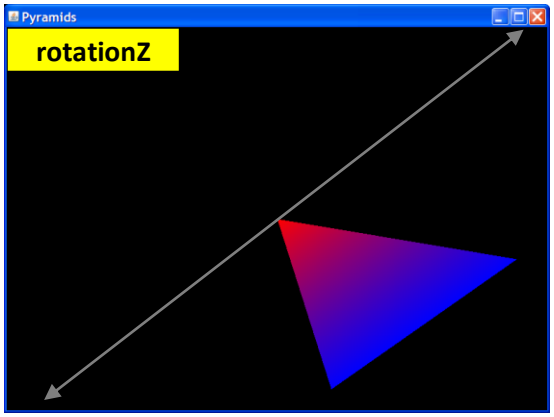
[1] 2 radians equals 360 degrees. 1 radian = 180/ degrees. 1 degree = /180 radians.

3. Replace the call to `rotationX` with a call to `rotationY`. Recompile and run to see the result.
4. Replace the call to `rotationY` with a call to `rotationZ`. Recompile and run to see the result.

Below are some screenshots of what you will see for each rotation (with the axis overlayed).



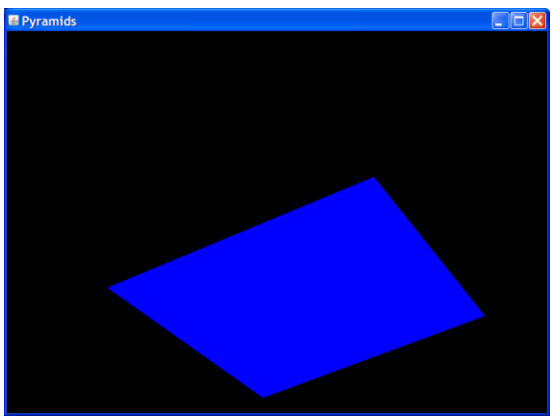




5. The `Matrix.rotationYawPitchRoll` method can be used to rotate around more than one axis at once. The yaw corresponds to the Y-axis, pitch: X-axis, and roll: Z-axis. Replace any code presently in the `transform` method with the following lines. This rotates the pyramid 45 degrees around all three axes.

```
        Matrix m = Matrix.rotationYawPitchRoll(0.78f, 0.78f, 0.78f);
        p.setTransform(m);
```

6. Compile Pyramids.java and run `PyramidsFrame` to see the result.



Summary

In this tutorial we have seen how to construct primitives and how to apply various transformations to them. The source code is in the file “Primitives.java” in the jdw.demo package.

In the next tutorial we will examine the world and view transformations, and explore how to combine multiple transformations with matrix multiplication.

Complete source code

For reference, here is the last iteration of the source code described in this tutorial.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import jdw.graphics.*;

/**
 * Renders and transforms a set of pyramid shapes.
 */
class PyramidFrame extends JFrame {
    private Environment3D env;

    private void transform(Primitive p) {
        Matrix m = Matrix.rotationYawPitchRoll(0.78f, 0.78f, 0.78f);
        p.setTransform(m);
    }

    /**
     * Creates the 3D scene.
     */
    private void initScene() {
        // Create a 3D environment into which objects can be placed
        env = new Environment3D();

        // Create a square-based pyramid
        SquareBasedPyramid py1 = new SquareBasedPyramid();
        env.addPrimitive(py1);
        transform(py1);

        // Eye: 3 units back on the Z-axis
        Vector3D eyePt = new Vector3D(0f, 0f, -3f);
        // Look at: the origin
        Vector3D lookAtPt = new Vector3D(0f, 0f, 0f);
        // Set the view using these two points
        env.setView(eyePt, lookAtPt);

        setLayout(new BorderLayout());
        add(env, BorderLayout.CENTER);
        setTitle("Pyramids");
        setSize(800, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    /**
     * Creates a new instance of PyramidFrame on the Swing event thread.
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                PyramidFrame frame = new PyramidFrame();
                frame.initScene();
```

```
            }
        });
    }
}

class SquareBasedPyramid extends Primitive {

    public SquareBasedPyramid() {
        super(RenderMode.TRIANGLE_LIST);

        // Front of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.BLUE));

        // Back of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.GREEN));
        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.GREEN));

        // Left of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.YELLOW));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.YELLOW));

        // Right of pyramid
        addVertex(new Vertex(0f, 0f, 0f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.RED));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.RED));

        // Base of pyramid (2 triangles)
        addVertex(new Vertex(-0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.BLUE));

        addVertex(new Vertex(0.5f, -0.8f, 0.5f, Color.BLUE));
        addVertex(new Vertex(-0.5f, -0.8f, 0.5f, Color.BLUE));
        addVertex(new Vertex(0.5f, -0.8f, -0.5f, Color.BLUE));
    }
}
```