Tutorial 1

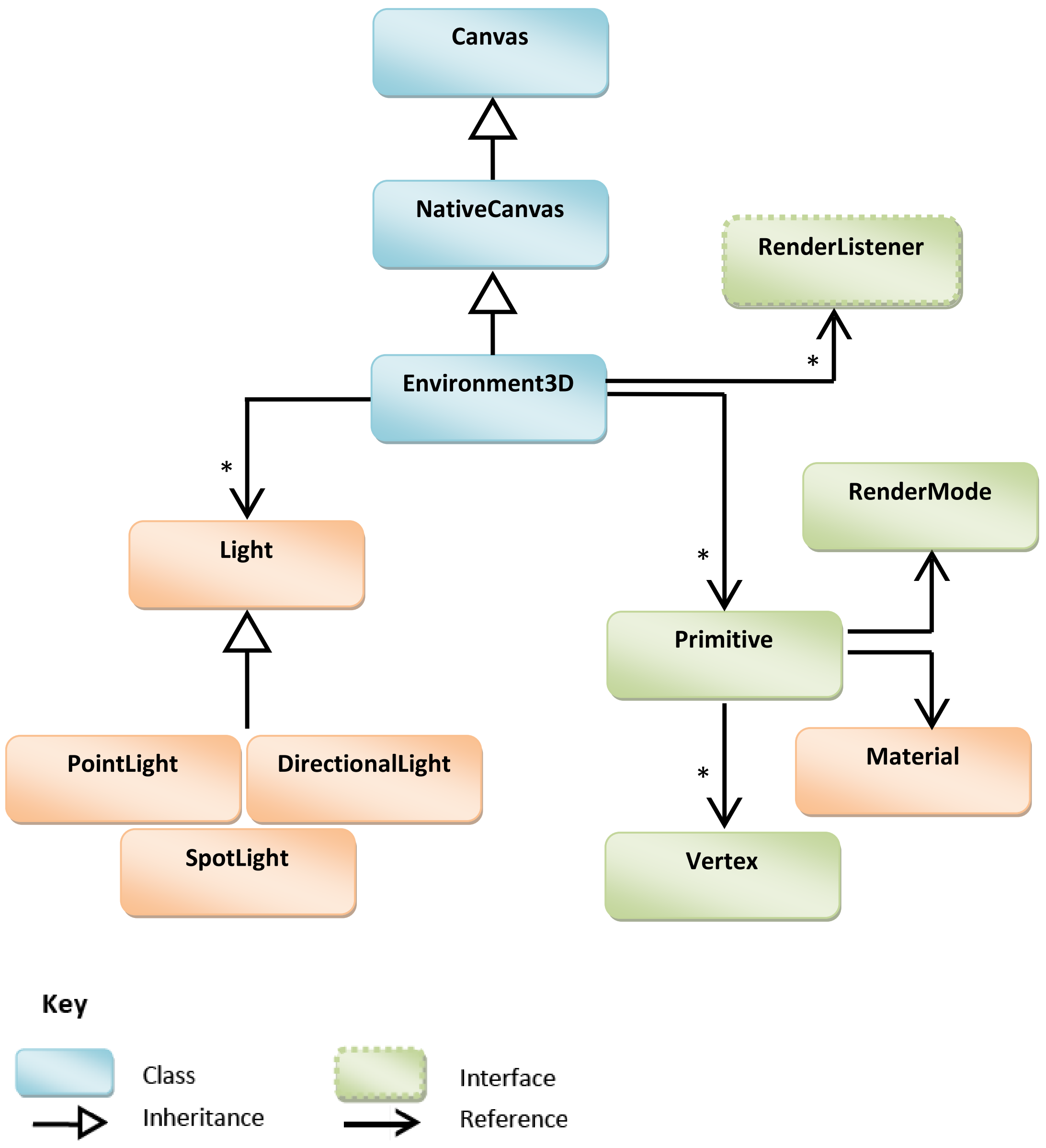
Getting started

In the first tutorial we will see an overview of the *Java-Direct3D Wrapper* (jdw) class library, and how to create and rotate some basic 3D shapes. For the tutorials, it is assumed that you have the Java 5 SDK installed, and understand Java AWT and Swing. The programs you will create are available in the `jdw.demo` package.

Contents

The class library

The diagram below illustrates the classes and interfaces provided by the jdw wrapper.



Environment3D – This class represents a 3D environment. `Environment3D` objects contain multiple lights and primitives (both of which are explained shortly). `Environment3D` objects are placed into your Swing and AWT programs (it is a subclass of `java.awt.Canvas`).

Primitive – A `Primitive` represents a 3D object of arbitrary complexity. A `Primitive` object could represent a simple cube, or a complex character in a game. Primitives are made up of a list of vertices (i.e. `Vertex` objects).

Vertex – A `Vertex` represents a single point of a 3D object. For example, to define a cube, you need 8 vertices, one for each corner of the cube. Each `Vertex` can have a color which contributes the color of the associated primitive.

RenderMode – This is an enumeration with constants specifying how the vertices of a primitive should be interpreted. For example, with the "triangle list" mode, each set of three vertices in the vertex list make up a triangle of the primitive.

Light – Light objects generate lighting for a scene. There are three types represented as three different subclasses: `PointLight`, `DirectionalLight`, and `SpotLight`. Lighting is optional, and is disabled by default. To enable lighting call `setLightingEnabled(true)` on the `Environment3D` instance.

PointLight – A point light is a source light having a specific position and light color. The light is emitted in all directions.

DirectionalLight – A directional light is a source light where the light appears to come from a far distance. It shines in a specific direction using a specific color.

SpotLight – A spot light emits a cone of light from a specific position in a specific direction. Like the other source lights, spot lights have a color.

Material – A material specifies the types and colors of light reflected by a primitive. Equivalent to a Direct3D material.

RenderListener – `RenderListeners` receive callbacks just before a frame is rendered. This allows any required graphical processing to be repeated without requiring extra threads.

Creating a 3D environment

We will create a Swing program containing a 3D environment. This program will consist of a single class which will contain code to create and rotate a cube.

1. Create a new file “ObjectRotator.java” and open it in your favorite text editor.
2. Create a class named “ObjectRotator” extending the class JFrame. The Direct3D wrapper classes we’ll be using are in the `jdw.graphics` package, so add an import statement for this package.

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

public class ObjectRotator extends JFrame {
}
```

3. Add some member variables to the ObjectRotator class.

```
    private Environment3D env;
    private Primitive obj;
```

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

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

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

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

We need to add some objects to the scene otherwise nothing will be visible. Create a cube primitive and add it to the scene.

```
        obj = Primitive.createCube();
        env.addPrimitive(obj);
```

Now we need to setup how the scene will be displayed to the user. To do this we need to define the “eye” or “camera” position, and the direction in which we are facing. These position and direction values are defined in terms of Vector3D objects. We pass the view information to the Environment3D instance using the `setView` method.

```
        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("Tutorial 1: ObjectRotator");
        setSize(800, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
```

5. 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.

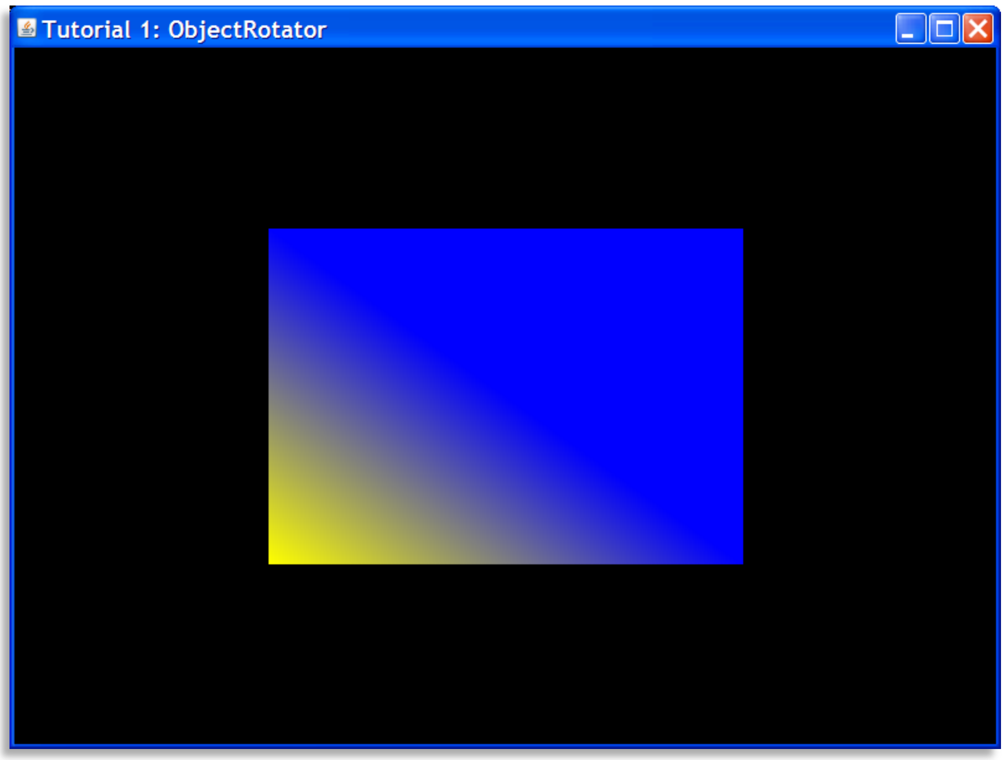
```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                ObjectRotator o = new ObjectRotator();
                o.initScene();
            }
        });
    }
```

6. Open a command prompt and compile ObjectRotator.java.

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

7. Run the program with the following command. A cube is displayed in a window.

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



Animating the scene

Right now the scene is not very interesting, we are looking straight at the front of the cube object. A `RenderListener` can be implemented to make the cube rotate a little bit every time a frame is to be rendered.

1. In the `initScene` method, just after the call to `setView`, add the following code.

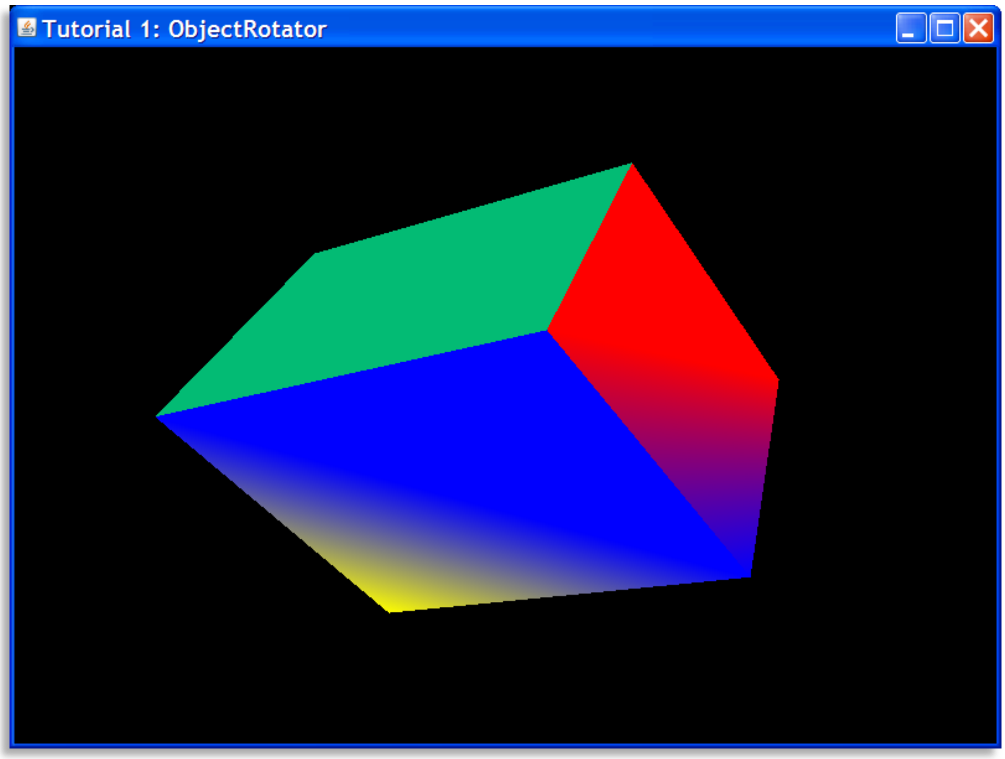
```
        env.addRenderListener(new RenderListener() {
            public void frameRendering(FrameRenderingEvent e) {
                Matrix worldMat = env.getWorldTransform();
                Matrix rot = Matrix.rotationYawPitchRoll(.005f, -.008f, .005f);
                worldMat = Matrix.multiply(worldMat, rot);
                env.setWorldTransform(worldMat);
            }
        });
```

The `frameRendering` method is called just before a frame is rendered. The `rotationYawPitchRoll` function is used to rotate the world transform on the X, Y and Z axes. Angles are specified in radians.

2. Compile and run ObjectRotator.java once again. You will see a more interesting result.

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

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



The `Primitive.createCube` method automatically assigned colors to the cube's vertices.

Summary

In this tutorial we have seen how easy it is to get up and running with the Java-Direct3D wrapper. Some details were ignored in this tutorial (e.g. the world transform), but these will be examined in later tutorials.

Complete source code

For reference, here is the complete ObjectRotator class described in this tutorial.

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

/**
 * Creates a cube and places it into an <tt>Environment3D</tt>.
 */
public class ObjectRotator extends JFrame {
    private Environment3D env;
    private Primitive obj;

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

        // Create a cube object and add to the environment
        obj = Primitive.createCube();
        env.addPrimitive(obj);

        // 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);

        // Automatically rotate the cube
        env.addRenderListener(new RenderListener() {
            public void frameRendering(FrameRenderingEvent e) {
                Matrix worldMat = env.getWorldTransform();
                Matrix rot = Matrix.rotationYawPitchRoll(.005f, -.008f, .005f);
                worldMat = Matrix.multiply(worldMat, rot);

                // Update the world transformation
                env.setWorldTransform(worldMat);
            }
        });

        setLayout(new BorderLayout());
        add(env, BorderLayout.CENTER);
        setTitle("Tutorial 1: ObjectRotator");
        setSize(800, 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
    /**
     * Creates a new instance of ObjectRotator on the Swing event thread.
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                ObjectRotator o = new ObjectRotator();
                o.initScene();
            }
        });
    }
}
```