
The following chapters of the textbook are relevant to this assignment:

- Chapter 6: Linear Algebra
- Chapter 7: Transformation Matrices
- Chapter 8: Viewing
- Chapter 9: The Graphics Pipeline

For OpenGL examples, consult the references:

- OpenGL slides from class
- OpenGL Programming Guide

This assignment is to be completed individually and the work you submit must be your own. You may edit any of the classes in the framework code and you may create new classes to complete the requirements. However, you do not need and should not use any other libraries.

In this programming assignment you will build an interactive 3D modeling system. Much of the system has been provided in the framework code, but key parts are missing that you will implement. This is part 1 of the assignment and includes rendering a triangle mesh, applying transformations in a scene graph, and using material properties with OpenGL.

1 XML Scene Files

For example, the XML file might look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<scene tolerance="0.014125375">
  <modeler.scene.Transformation
    name="Root"
    T="0.0 0.0 0.0"
    R="0.0 0.0 0.0"
    S="1.0 1.0 1.0"
  >
    <modeler.scene.PerspectiveCamera
      name="Camera"
      eye="5.0 5.0 3.0"
      target="0.0 0.0 0.0"
      up="-0.5581804 0.7591253 -0.33490822"
      fovy="45.0"
      near="2.0"
      far="1000.0"
    />
    <modeler.scene.Light
      name="Light 0"
```

```
      position="6.0 8.0 10.0"
      ambient="0.1 0.1 0.1"
      diffuse="0.8 0.8 0.8"
      specular="2.0 2.0 2.0"
    />
    <modeler.scene.Transformation
      name="Cube"
      T="-1.0 0.0 0.0"
      R="10.0 0.0 0.0"
      S="1.0 1.5 1.0"
    >
      <modeler.shape.Cube
        name="Cube"
        diffuseColor="0.8 0.8 0.8"
        specularColor="0.0 0.0 0.0"
        exponent="40.0"
      />
    </modeler.scene.Transformation>
  </modeler.scene.Transformation>
</scene>
```

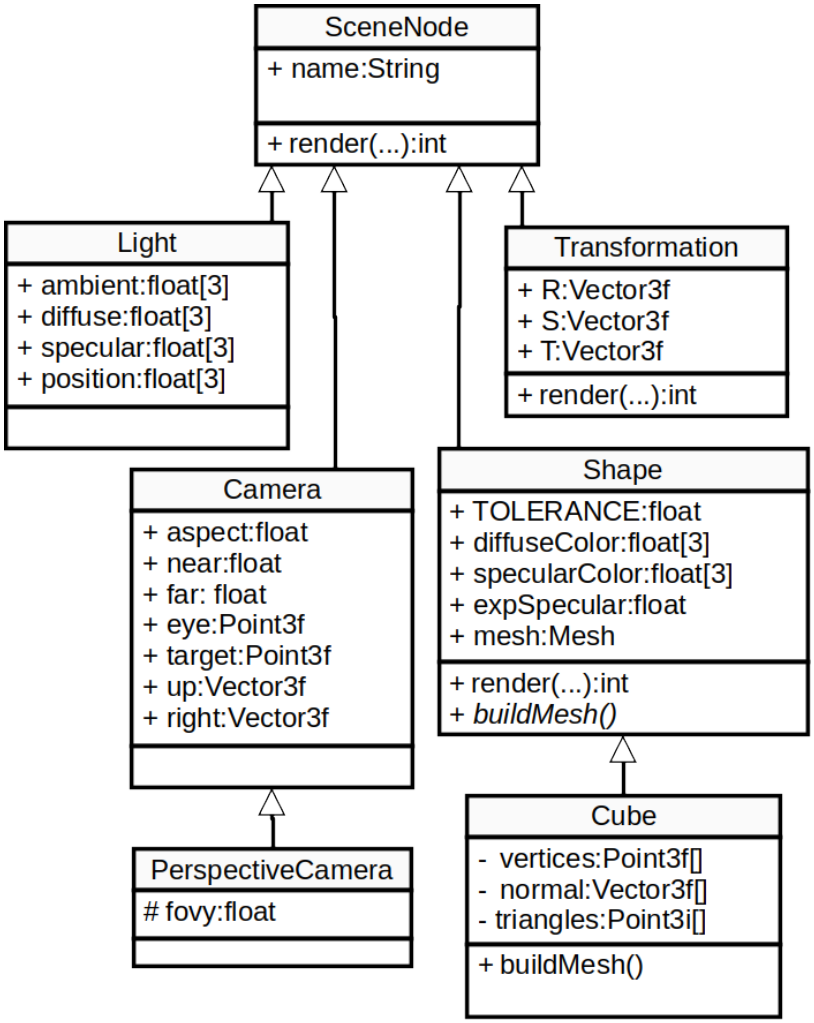
The scene contains a scene graph with the root node being a transformation node named *Root*. The root node applies no translation, rotation, or scaling but serves to contain all descendant objects and their transformations. Within the scene graph is a perspective camera, a light, and another transformation node containing a cube. Each transformation applies to all of its children. In this case, the default $2 \times 2 \times 2$ cube centered at the origin is first scaled 1.5 times in the y-direction, then rotated by 10 degrees counter-clockwise around the x-axis, and finally translated by -1 along the x-axis.

2 Assignment

Every object in the scene is a subclass of the abstract `SceneNode` class. For example, a `Transformation` is a `SceneNode` that applies a transformation matrix (scale, rotation, and translation) to all descendant nodes. The abstract `Shape` class is a `SceneNode` that renders shapes (draws them) by cycling through the triangles that define them.

When the scene is rendered, each node is recursively rendered using `SceneNode.render()`. If it's a `Shape`, `Shape.render()` sets the appropriate material properties and calls `mesh.render()`. If it's a transformation, `Transformation.render()` modifies the current model view matrix to scale, rotate, and translate subsequent calls to `glVertex`. Importantly, the `Transformation` node is the only `SceneNode` that has children, and therefore must call `SceneNode.render` to render it's children.

Make sure you understand what the program is intended to do first. You can download the framework code from here: `https://cs.appstate.edu/rmp/cs5465/model.zip`



For this assignment, you will implement `Mesh.render`, `Transformation.render`, and `Shape.render`.

2.1 Scene drawing using OpenGL

OpenGL reference: `https://cs.appstate.edu/rmp/cs5465/openglprogramming-guide-v2.pdf`

The official documentation is for the C programming language. It's pretty easy to translate to Java. For example in C, OpenGL functions start with `gl` and use `GLenum` types as constants:

```
glBegin(GL_POINTS);
...
glEnd();
```

In Java you will need a `GL` object, `gl` in these examples, and the OpenGL methods are members of that object. The constants are declared statically in the `GL` class:

```
gl.glBegin(GL.GL_POINTS);
...
gl.glEnd();
```

Mesh generation for cubes has already been implemented, so you can focus on rendering a scene using OpenGL:

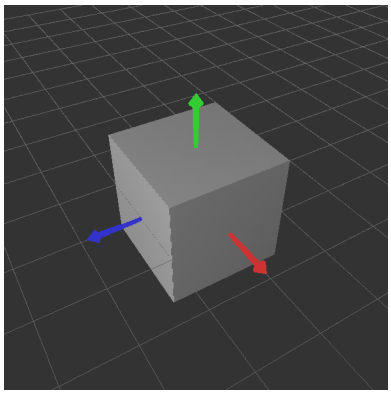
2.2 `Mesh.render`

Implement `Mesh.render` to draw the mesh geometry. Here are some helpful functions:

- glBegin()...glEnd()
- glNormal
- glVertex
- Back Face Culling

You will use OpenGL to render *triangles* with "back face culling" turned on.

After completing this part, your cube should look like this:



cube

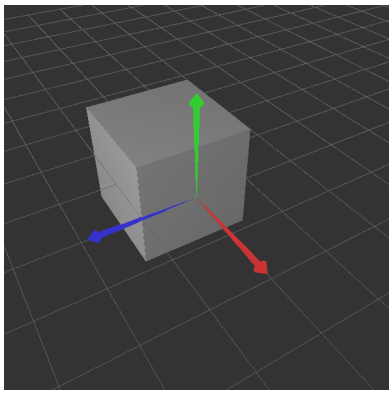
2.3 `Transformation.render`

Implement `Transformation.render` to make sure the transformation hierarchy is applied properly.

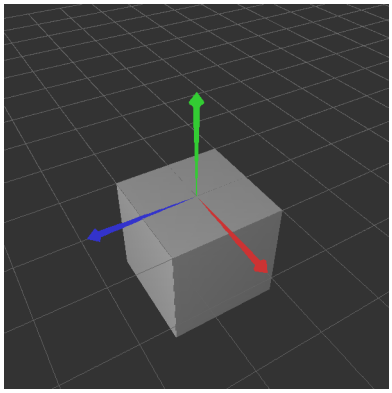
Helpful methods:

- glMatrixMode()
- glPushMatrix()...glPopMatrix()
- glRotate
- glScale
- glTranslate

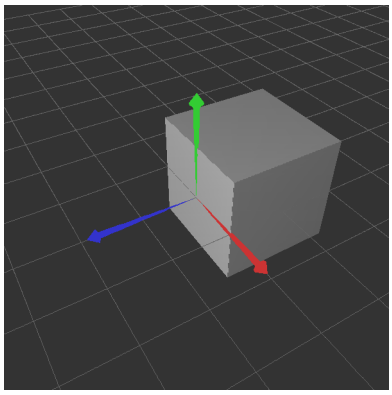
After completing this part, the following examples should work:



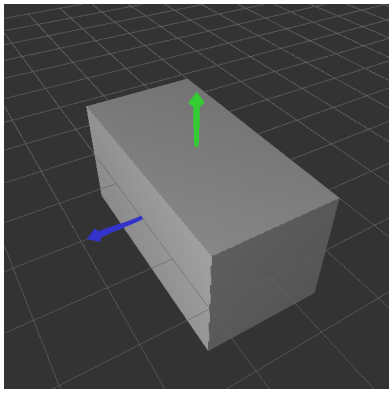
cube-tx



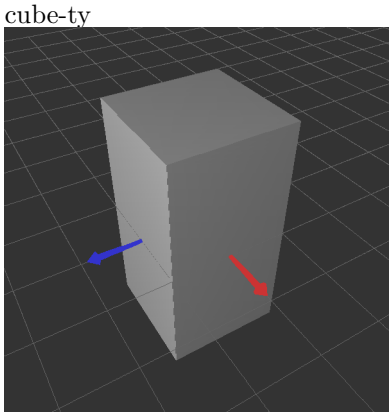
cube-ty



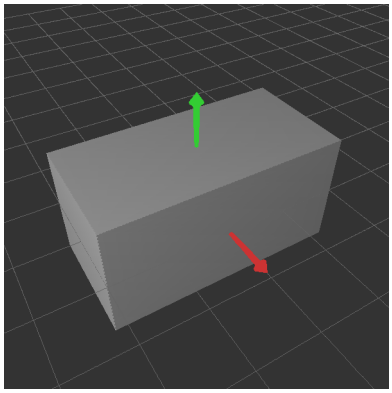
cube-tz



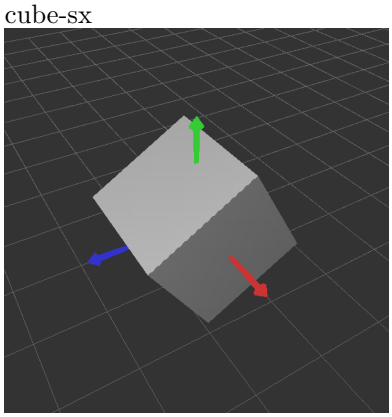
cube-sx



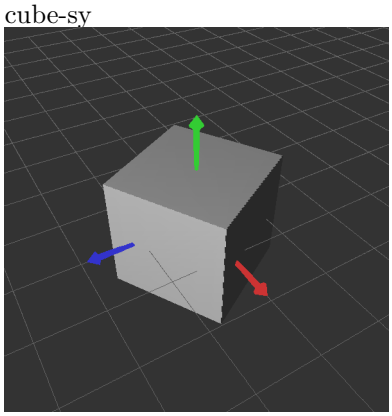
cube-sy



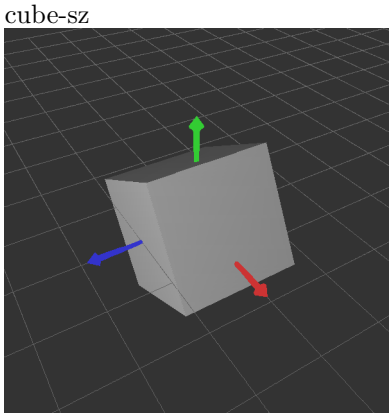
cube-sz



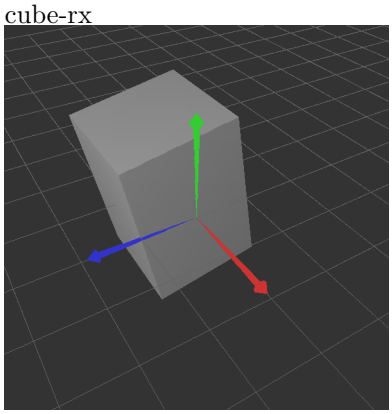
cube-rx



cube-ry



cube-rz



cube-rst

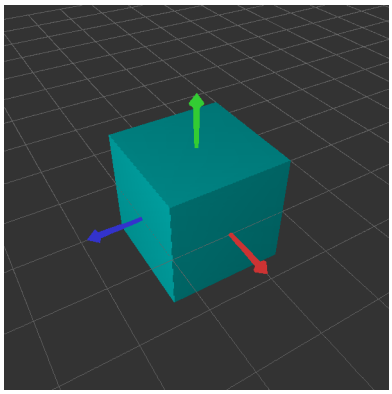
2.4 `Shape.render`

Complete `Shape.render` to set the material parameters for each mesh. We will use the "diffuse color" for the diffuse and ambient color of the material.

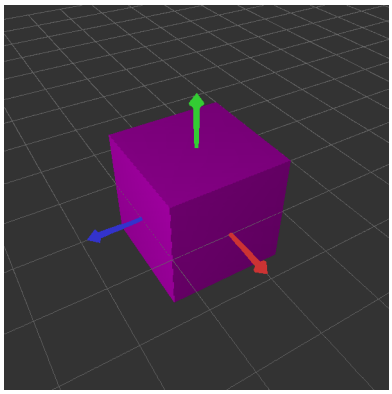
Helpful methods:

glMaterial

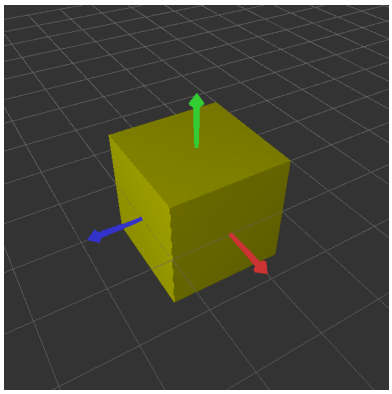
After completing this part, the material properties should work:



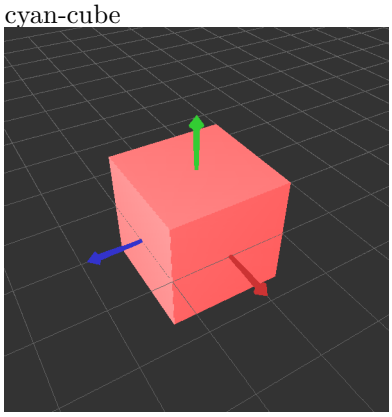
cyan-cube



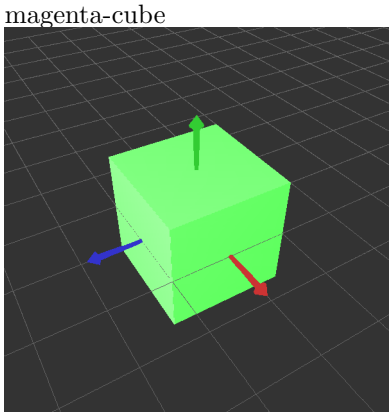
magenta-cube



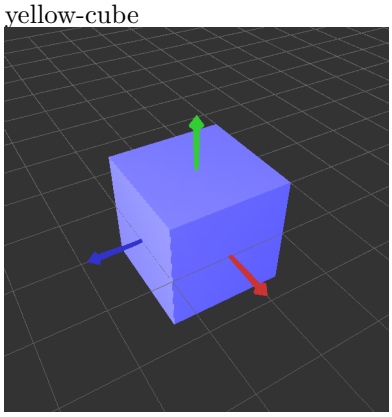
yellow-cube



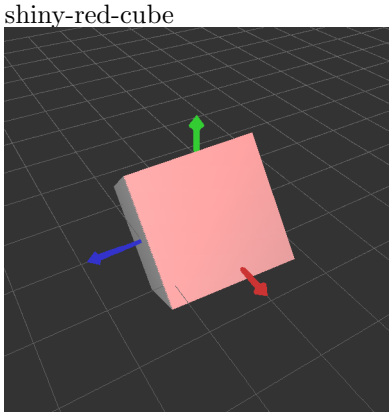
shiny-red-cube



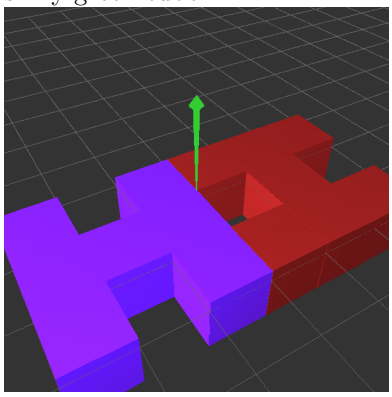
shiny-green-cube



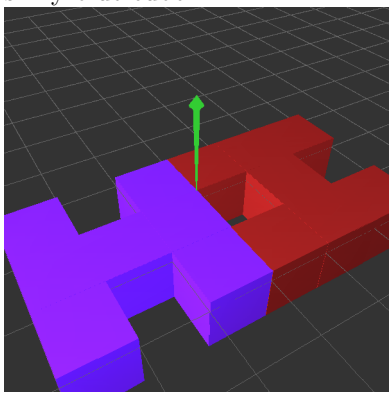
shiny-blue-cube



super-shiny-cube



hi



hi2

3 Java Development

To work on this program you can use your favorite Java IDE such as Eclipse or VS Code. We will be using **Java 8** with **JUnit4**. Later versions of Java will not work for some of the assignments. The working directory for your project should be the `ray1` folder. The source location should be the `ray1/src` folder.

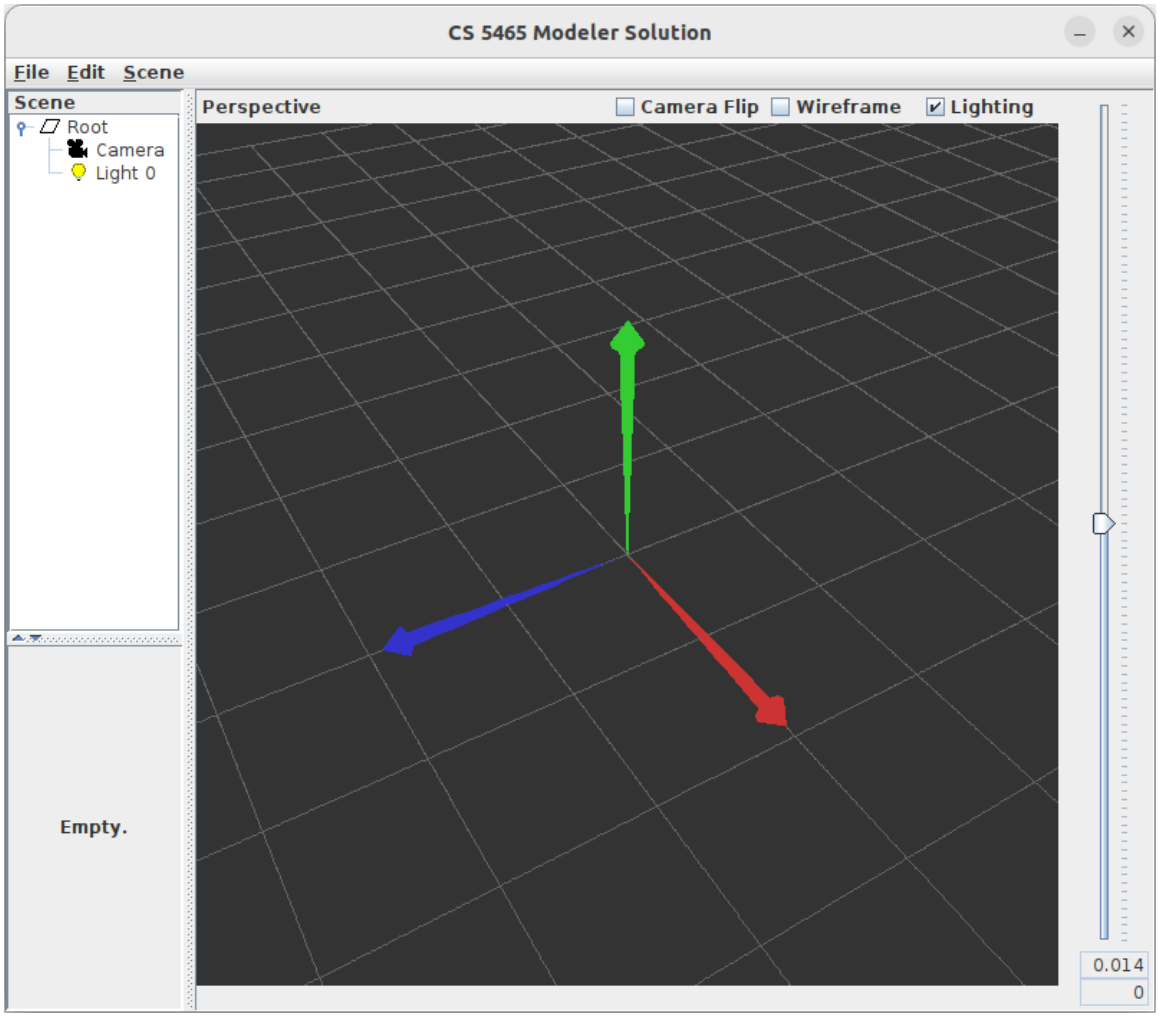
You can install the JDK for Java 8 here:
`https://adoptium.net/temurin/releases/?version=8`

On linux, you may install OpenJDK version:
`https://openjdk.org/install/`

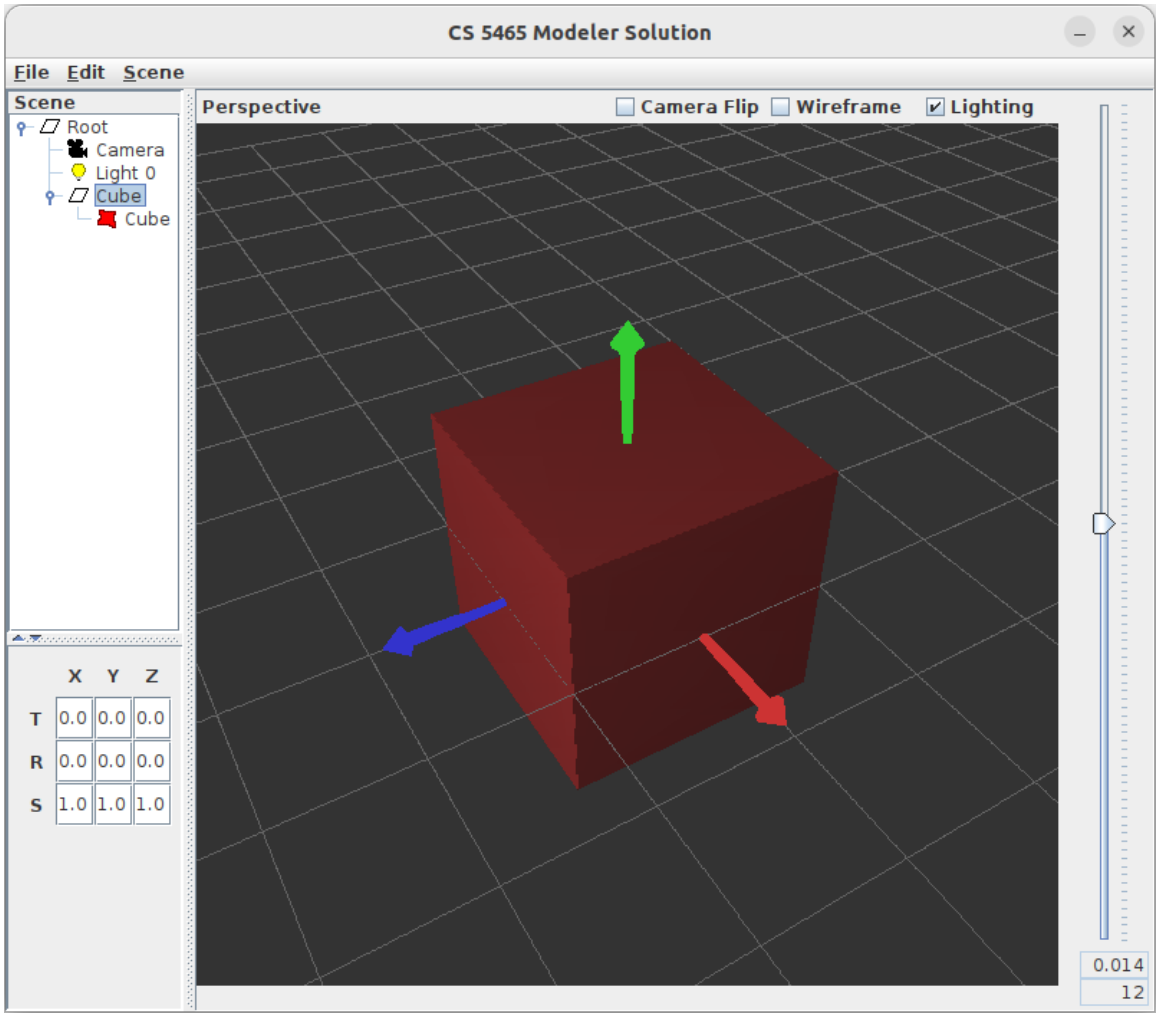
You can get JUnit4 here:
`https://github.com/junit-team/junit4/wiki/Download-and-Install`

3.1 Running the Program

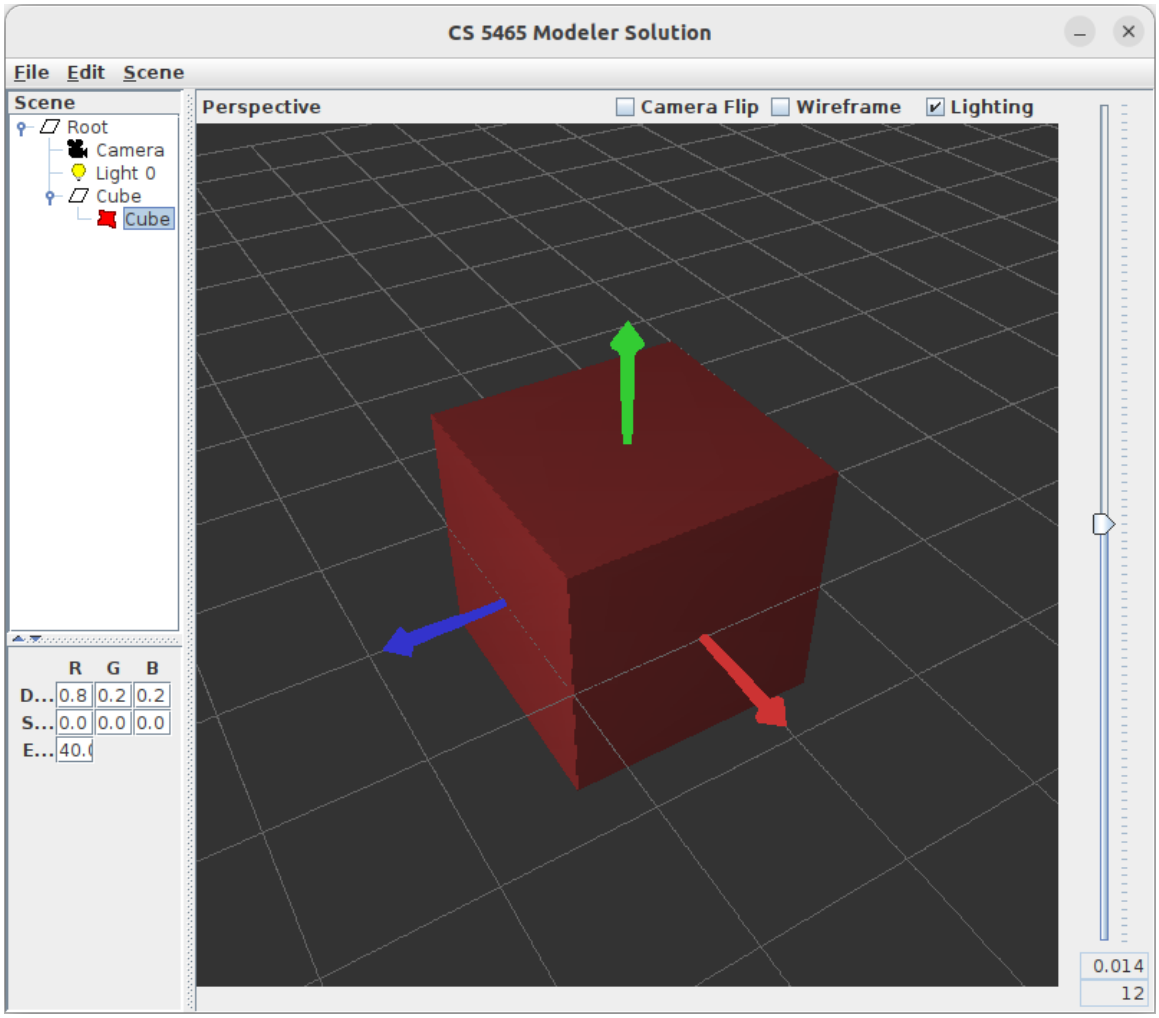
To run the program on a particular XML file, you can run `modeler.MainFrame` as a Java Program with zero or more XML files as arguments. If you provide no arguments, the GUI opens and you can interact with it:



Add a cube by clicking "Scene" and then "Add Cube." The cube will appear in the scene graph panel on the left, including a transformation node (notice the **T**ranslate, **R**otate, and **S**cale details in the editable boxes in the bottom of the panel.):



In addition, specifics for the cube are editable (Diffuse RGB, Spectral RGB, and spectral exponent):



If you run the program with an XML file as command line argument, the GUI will open, create the scene, render the image and save it to a PNG file in the same directory as the XML, then close the window.

For example, when I run the program in VS Code I get:

```
$  /usr/bin/env /usr/lib/jvm/java-8-openjdk-amd64/bin/java -cp
↪  /tmp/cp_b250zfl4icb5tf1dzxslg8obd.jar modeler.MainFrame scenes/cube.xml
Loading scenes/cube.xml
GLView.GLView
GLView.getCamera
GLView.processEvent
GLView.init
GLView.componentResized
GLView.myReshape
GLView.display
GLView.componentMoved
GLView.init
GLView.componentResized
GLView.myReshape
GLView.display
GLView.componentResized
```

```
GLView.myReshape
GLView.display
GLView.componentResized
GLView.myReshape
GLView.display
GLView.display
GLView.refresh
Root
Camera
GLView.setCamera
Light 0
Cube
Cube
GLView.display
GLView.refresh
GLView.display
GLView.refresh
```

3.2 Running the Tests

Use the standard way to run JUnit4 tests in your IDE to run the tests in `tests.ModelTest1`.

4 User's Manual

4.1 Menu Items relevant to this assignment

- File→Open: Open a scene from an XML file.
- File→Save as: Save the current scene as an XML file.
- File→Exit: Exit the program.
- Edit→Group selected: groups nodes into a new parent transformation.
- Edit→Reparent selected: move nodes under an existing parent transformation.
- Edit→Delete selected: removes the selected nodes from the tree. Cannot be undone. Lights and cameras cannot be removed.
- Edit→Select: Enter select mode so that clicking on an object in the view port will select it in the tree
- Scene→Add Light: Add a new light source in the scene at the root. A maximum of eight light sources can be added.
- Scene→Add Cube: Add a new $2 \times 2 \times 2$ cube to the scene at the origin.

4.2 Scene Panel

The scene panel provides a hierarchical display of the scene graph. Each element is either an object or a transformation. Selecting an object will reveal its modifiable properties in the Property Panel in the lower-left corner.

4.3 Property panel

Depending on the selected node, different modifiable properties are shown:

- Transformations: **T**ranslation, **R**otation, and **S**cale in x, y, and z, respectively
- Objects: **Diff**use and **Spec**ular color in red, green, and blue, respectively. Specular **Exp**onent
- Lights: **Pos**ition in x, y, and z; and intensity (**Pow**) in red, green, and blue.

4.4 Viewport

- Alt/Option Left-click and drag: orbits around target point
- Alt/Option Right-click and drag: translate along view direction: up$\rightarrow$ toward the origin; down $\rightarrow$ away from the origin
- Ctrl-alt Left-click and drag: Track mouse.
- Left-click: select object's transform
- Shift-Left-click: select multiple objects

5 Submissions

Your assignment will be graded by Web-CAT. Compress your `src` directory into a ZIP file and upload it here:
`http://webcatvm.cs.appstate.edu:8080/Web-CAT`