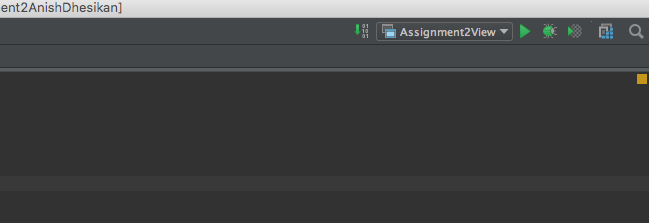
Anish Dhesikan

CS4300 Manual

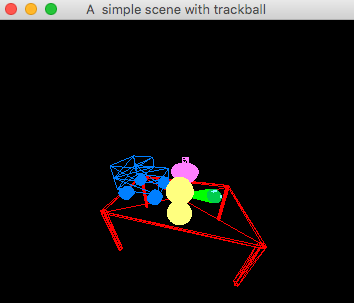
# Assignment 2 (02/16/16)

## How to Use The Program

To run the program in IntelliJ IDEA, set the Run Configurations to “Assignment2View.” Do this at the top right of the interface, as shown below in red:

Proceed to then press the run button, shown in blue above. This will open a window containing the graphics program. It may take a few seconds for the initial rendering.

Click and drag the mouse within the program’s window to rotate the scene like a trackball. Use the mouse scroll wheel to zoom in/out.



The scene is set up in the initObjects() method in the View class. In this method, 3D models are loaded from .obj files and placed in the scene. For example, the top of the table is drawn with the following block of code:

|  |
| --- |
| in = new FileInputStream("models/box.obj"); mesh = util.ObjImporter.*importFile*(in,true); o = new util.ObjectInstance(gl,program,mesh,new String("box")); mat.setAmbient(1, 0, 0); //only this one is used currently to determine color mat.setDiffuse(1, 0, 0); mat.setSpecular(1, 0, 0); meshObjects.add(o); meshMaterials.add(new Material(mat)); transform = new Matrix4f().translate(0, tableTop - tableThickness / 2, 0).scale(200,tableThickness,100); meshTransforms.add(transform); |

The user can change the camera before running the program in the View class’s draw() method. Here, there is a commented line that explains how to change the camera:

|  |
| --- |
| //set the camera at (0,0,150), looking at (0,0,0) with its 'up' direction as (0,1,0) modelView.peek().mul( new Matrix4f().lookAt(new Vector3f(0,0,150),new Vector3f(0,0,0), new Vector3f(0,1,0))); |

The trackball is implemented with a field on the View class called trackballTransform that is a matrix applied to the entire scene after all other rendering. This matrix is applied in the draw () method with the following lines:

|  |
| --- |
| //the trackball is essentially another transformation to the whole scene modelView.peek().mul(trackballTransform); |

This trackballTransform field is modified when the mouse is dragged / scrolled. The View class has a method called mouseDragged() in which the field is modified every time the mouse is dragged. Essentially, the currently x and y rotation of the scene are stored in the variables xRot and yRot. Each time the mouse is dragged, the difference in mouse position is factored into the xRot and yRot. For example, if the mouse is dragged from left to right, the change in mouse position is positive in the x-direction. Thus, xRot is accordingly increased. Then the trackballTransform matrix’s rotation is set based on the xRot and yRot. The trackballRadius determines how quickly the trackball is rotated. The larger the trackball, the slower it will rotate (or rather, the more the mouse will have to move to get the same result).

|  |
| --- |
| public void mouseDragged(int x,int y) {  float trackballRadius = 50f;  int deltaX = x - prevMouseX;  int deltaY = y - prevMouseY;  rotX += deltaX;  rotY += deltaY;   Vector3f prevScale = trackballTransform.getScale(new Vector3f().zero());  trackballTransform = trackballTransform.setRotationXYZ(rotY / trackballRadius, rotX / trackballRadius, 0);  trackballTransform = trackballTransform.scale(prevScale);    prevMouseX = x;  prevMouseY = y;  } |

## How to Create an Object of Revolution

First off, what is an object of revolution? An object of revolution is any object that can be created by rotating a curve about an axis. In doing this, any cross-section of the object is a perfect circular shape. For example, many vases are objects of revolution because any given horizontal cross section is a circle. However, a cell phone would not be an object of revolution because none of its cross sections (along any axis) are circles.

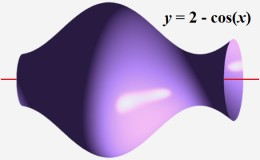


Figure - A vase and the axis around which it was rotated. Taken from <http://usercontent1.hubimg.com/7148892_f260.jpg>

To create an object of revolution, one must first create a curve, and then rotate this curve around an axis (for our purposes, we’ll assume this axis is the y-axis). This can be done with nested for-loops, the outer loop controlling the y-value of each layer, and the inner loop controlling the circle (and its radius) created at that layer.

|  |
| --- |
| for (int i from 0 to numLayers) {  for (int j from 0 to numVertsInCircle) {  theta = 2\*Math.*PI*\*j/(numVertsInCircle);   radius = 1; // this is the radius of this cross-section.  // this radius can depend on the current layer (i) e.g. cos(i) makes a  // vase-like curve   vertY = i / 5; // this is the y-position of the current layer. Should be dependent on i.   // calculate the position of a given vertex in this cross-section / circle  v = new Vector4f(radius \* (float) Math.*cos*(theta),  vertY,  radius \* (float) Math.*sin*(theta),  1.0f);   // add this vertex to your list of vertices  verts.add(v);  }  } } |

Once all the vertices are stored, you must connect them as triangles by storing indices in a list of triangles. Depending on the order in which you store your vertices, the process of connecting them as triangles may differ. However, the following code will be sufficient for the above method:

|  |
| --- |
| for (int i=0;i<numLayers-1;i++) {  for (int j=0;j<numVertsInCircle;j++)  {  triangles.add(i\* numVertsInCircle +j);  triangles.add(i\* numVertsInCircle +(j+1)% numVertsInCircle);  triangles.add(((i+1)% numLayers)\* numVertsInCircle +(j+1)% numVertsInCircle);   triangles.add(i\* numVertsInCircle +j);  triangles.add(((i+1)% numLayers)\* numVertsInCircle +(j+1)% numVertsInCircle);  triangles.add(((i+1)% numLayers)\* numVertsInCircle +j);  } } |

After this, there is sufficient information to create a mesh from vertices and triangles.