Adding a Floor Lamp to the 3D Scene

Prof. Carlos Carreto

This tutorial describes the creation of a floor lamp shape called FloorLamp to add to the 3D scene of the Ex7_1 project implemented in class. As in the case of the Table shape, geometric transformations are used to create the FloorLamp shape, both to group simpler shapes and to create the mesh polygon of one of the parts of the floor lamp.

It is assumed that the project already exists and the Table shape has been created already. The project made in class is in moodle's Computer Graphics page.

Start by adding a new class called FloorLamp to the project and add the following code to the class.

```
import javax.media.j3d.Appearance;
import javax.media.j3d.Group;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Vector3d;

import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Primitive;

public class FloorLamp extends Group {
  public FloorLamp(Appearance app) {
    // Base of the floor lamp
    Cylinder base = new Cylinder(0.3f, 0.05f, Cylinder.GENERATE_NORMALS, 30,
1, app);
    Transform3D tr = new Transform3D();
    tr.setTranslation(new Vector3d(0, 0.025, 0));
    TransformGroup tg = new TransformGroup(tr);
    tg.addChild(base);
    this.addChild(tg);

    // Body of the floor lamp
    Primitive body = new Cylinder(0.025f, 1.5f, Cylinder.GENERATE_NORMALS,
app);
    tr = new Transform3D();
    tr.setTranslation(new Vector3d(0, 0.75, 0));
    tg = new TransformGroup(tr);
    tg.addChild(body);
    this.addChild(tg);
  }
}
```

The last code creates part of the floor lamp (the base and the body) using two cylinders that are positioned and grouped through geometric transformations. This is the same technic used to construct the Table shape.

Add the following code to the createSceneGraph method of the main class of the project, just before the creation of the background of the scene. The code creates a FloorLamp object and adds it to the graph scene.

```
// Floor Lamp
FloorLamp floorLamp = new FloorLamp(app);
tr = new Transform3D();
tr.setScale(0.5f);
tr.setTranslation(new Vector3f(-0.3f, 0f, 0f));
tg = new TransformGroup(tr);
tg.addChild(floorLamp);
```

```
root.addChild(tg);
```

Run the application. Figure 1 shows the scene with the floor lamp partially built.

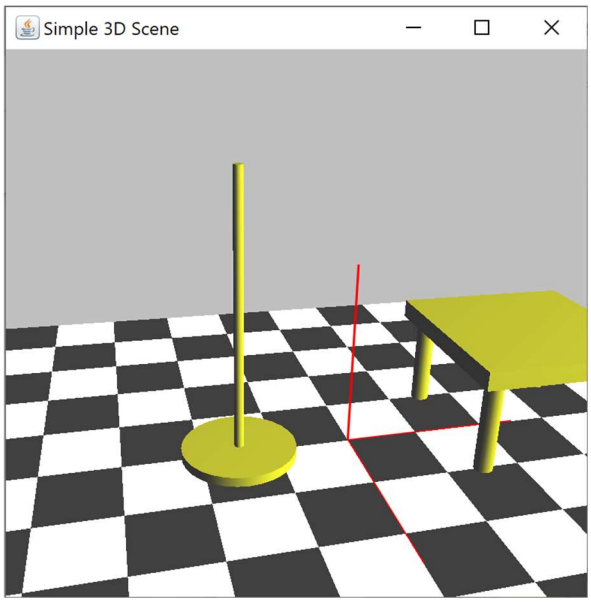

Figure 1 – Scene with the floor lamp.

The floor lamp is finished by adding the lampshade shape of figure 2 at the top of the body.

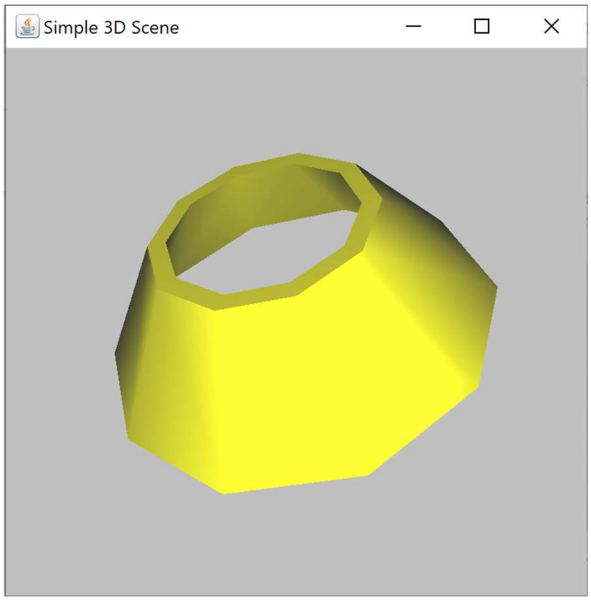

Figure 2 – Lampshade shape.

The lampshade shape has to be built from scratch, creating a mesh of polygons that define its geometry. Figure 3 shows a possible quadrilateral polygon mesh to construct the shape's geometry.

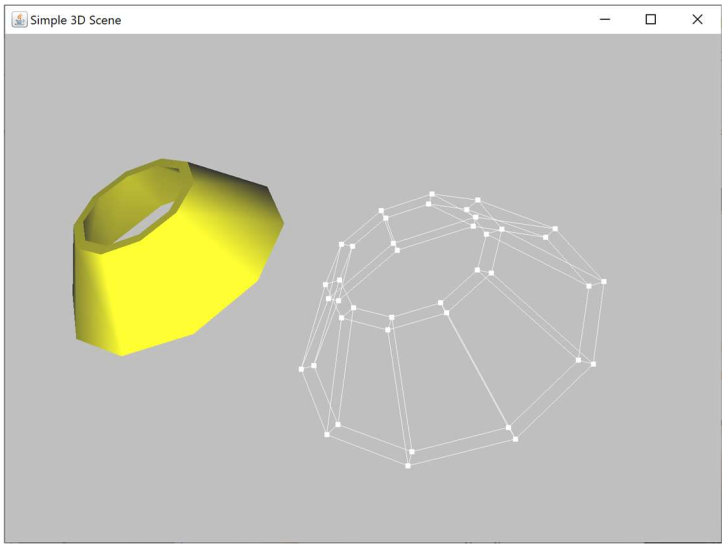

Figure 3 – Possible quadrilateral polygon mesh.

This polygon mesh can be created taking advantage of the symmetry that the geometry presents around the Y axis. Figure 4 shows a set of 4 vertices (a, b, c and d) that define a "slice" of the geometry. This initial set of vertices (which can easily be defined manually), can be used to generate the vertices of the next slice of the geometry (e, f, g, and h) by applying a rotation around the Y axis.

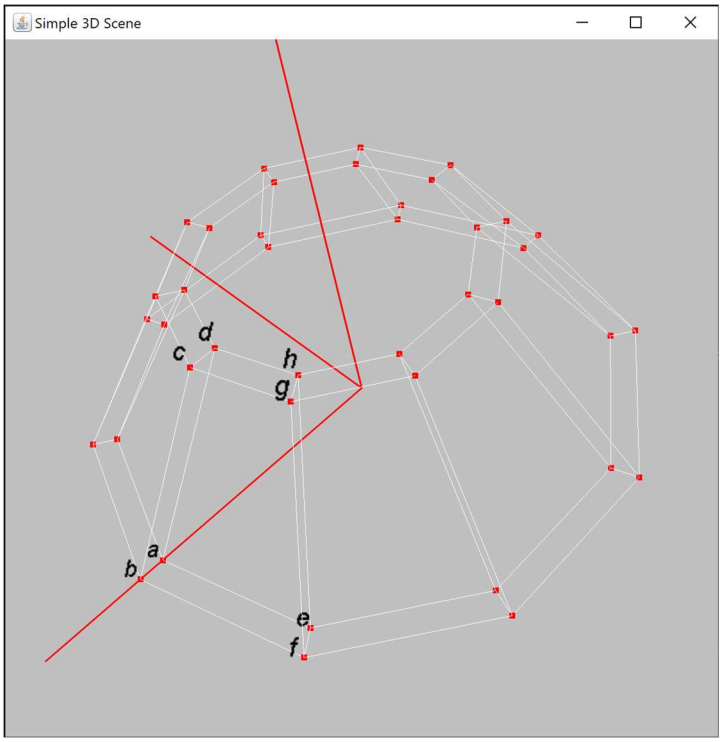

Figure 4 –Vertices of two slices.

The vertices of these two slices can then be used to define 4 quadrilaterals of the polygon mesh width the following sequences:

- Back: a, d, h, e
- Front: b, f, g, c
- Top: c, g, h, d
- Bottom: a, e, f, b

To construct the lampshade, add a new class called Lampshade to the project and add the following code to the class.

```
class Lampshade extends Shape3D {
  public Lampshade(int n, Appearance app) {

  }
}
```

The constructor receives 2 parameters, the number of slices to consider for the construction of the polygon mesh of the 3D shape and the appearance of the 3D shape.

As seen in figure 3, the geometry is formed by quadrilaterals. Add the following code to the constructor to create an object of type IndextQuadArray to create such a geometry. The IndexedQuadArray allows to not have to repeat vertex coordinates.

```
int totalVertices = 4 * n; // Total number of unique vertices.
int totalIndices = 4 * 4 * n; // Total number of indices.

IndexedQuadArray qa = new IndexedQuadArray(totalVertices,
                          IndexedQuadArray.COORDINATES, totalIndices);
```

Next thing to do is to generate and set the coordinates of all the unique vertices of the geometry.

Add the following code to define the 4 initial vertices of a slice.

```
Point3f[] pts = {new Point3f(1, 0, 0), new Point3f(1.1f, 0, 0), new
Point3f(1.1f - 0.5f, 0.7f, 0f),new Point3f(1f - 0.5f, 0.7f, 0f)};
```

Define a transformation to generate the unique vertices. In this case it is a rotation around the Y axis of an angle of 360 divided by the number of slices.

```
Transform3D tr = new Transform3D();
tr.rotY(2 * Math.PI / n);
```

For each slice and for each vertex of the slice, add the vertex to the geometry. After the vertex is added, rotate the vertex around Y to generate the correspondent vertex of the next slice.

```
int index = 0;
for (int j = 0; j < n; j++) { // for each slice
  for (int i = 0; i < 4; i++) { // for each vertex of the slice
    qa.setCoordinate(index++, pts[i]);

    tr.transform(pts[i]);
  }
}
```

Now that we have set the unique vertices of the geometry, we can index them to define the quadrilaterals of the polygon mesh (the faces of the geometry).

The unique vertices were defined according to the sequence of the geometry slices. That is, the first 4 vertices with indices 0, 1, 2 and 3 correspond to the 4 vertices of the first slice identified in figure 4 with the letters a, b, c and d. The next 4 vertices with indices 4, 5, 6 and 7, correspond

to the 4 vertices of the second slice identified in figure 4 with the letters e, f, g and h. The following 4 vertices correspond to the next slice and so on.

Define the initial set of indexes of vertices of the first slice.

```
int a = 0;
int b = 1;
int c = 2;
int d = 3;
```

Add the following code to construct 4 faces using the 4 vertices of the actual slice and the 4 vertices of the next slice.

```
index = 0;
int e, f, g, h; // the next 4 vertices.
for (int j = 0; j < n; j++) {

  // Define the next 4 vertices from the actual ones.
  // The division by totalVertices makes the indices in the range
  // [0, totalVertices[.
  e = (a + 4) % totalVertices;
  f = (b + 4) % totalVertices;
  g = (c + 4) % totalVertices;
  h = (d + 4) % totalVertices;

  // Back face
  qa.setCoordinateIndex(index++, a);
  qa.setCoordinateIndex(index++, d);
  qa.setCoordinateIndex(index++, h);
  qa.setCoordinateIndex(index++, e);

  // Front face
  qa.setCoordinateIndex(index++, b);
  qa.setCoordinateIndex(index++, f);
  qa.setCoordinateIndex(index++, g);
  qa.setCoordinateIndex(index++, c);

  // Top face
  qa.setCoordinateIndex(index++, c);
  qa.setCoordinateIndex(index++, g);
  qa.setCoordinateIndex(index++, h);
  qa.setCoordinateIndex(index++, d);

  // Bottom face
  qa.setCoordinateIndex(index++, a);
  qa.setCoordinateIndex(index++, e);
  qa.setCoordinateIndex(index++, f);
  qa.setCoordinateIndex(index++, b);

  // Update the indices, the last 4 become the new 4.
  a = e;
  b = f;
  c = g;
  d = h;
}
```

Appearances based on interaction between light and material only work if the geometry includes the definition of the vertex normals. Add the following code to transform the IndexQuadArray geometry into GeometryInfo to be able to generate those normals.

```
GeometryInfo gi = new GeometryInfo(qa);
NormalGenerator ng = new NormalGenerator();
ng.generateNormals(gi);
```

Finally set the geometry and the appearance to construct the Shape3D object.

```
this.setGeometry(gi.getGeometryArray());
this.setAppearance(app);
```

The complete code of the Lampshade class is shown next.

```
class Lampshade extends Shape3D {
  public Lampshade(int n, Appearance app) {

    // n = number of slices
    int totalVertices = 4 * n; // Total number of unique vertices.
    int totalIndices = 4 * 4 * n; // Total number of indices.

    // The geometry is constructed with quadrilaterals.
    // The IndexedQuadArray allows to not have to repeat vertex coordinates,
    IndexedQuadArray qa = new IndexedQuadArray(totalVertices,
                            IndexedQuadArray.COORDINATES, totalIndices);

    // Transformation to generate the unique vertices
    Transform3D tr = new Transform3D();
    tr.rotY(2 * Math.PI / n); // Rotation around Y axis.

    // The 4 initial vertices that define a slice.
    Point3f[] pts = {new Point3f(1, 0, 0), new Point3f(1.1f, 0, 0), new
Point3f(1.1f - 0.5f, 0.7f, 0f),new Point3f(1f - 0.5f, 0.7f, 0f)};

    // Generate all the unique vertices from the 4 initial vertices
    int index = 0;
    for (int j = 0; j < n; j++) { // for each slice
      for (int i = 0; i < 4; i++) { // for each vertex of the slice
        // Add the 'i' vertex to the geometry.
        qa.setCoordinate(index++, pts[i]);

        // After the vertex 'i' is added, the vertex is rotated around Y to
        // generate the correspondent next one.
        tr.transform(pts[i]);
      }
    }

    // Initial set of indexes to define the faces.
    int a = 0;
    int b = 1;
    int c = 2;
    int d = 3;

    // Construct 4 faces using the actual 4 vertices and the next 4.
    index = 0;
    int e, f, g, h; // the next 4 vertices.
    for (int j = 0; j < n; j++) {

      // Define the next 4 vertices from the actual ones.
      // The division by totalVertices makes the indices in the range [0,
      // totalVertices[.
      e = (a + 4) % totalVertices;
      f = (b + 4) % totalVertices;
      g = (c + 4) % totalVertices;
      h = (d + 4) % totalVertices;

      // Back face
      qa.setCoordinateIndex(index++, a);
      qa.setCoordinateIndex(index++, d);
      qa.setCoordinateIndex(index++, h);
      qa.setCoordinateIndex(index++, e);

      // Front face
      qa.setCoordinateIndex(index++, b);
```

```
            qa.setCoordinateIndex(index++, f);
            qa.setCoordinateIndex(index++, g);
            qa.setCoordinateIndex(index++, c);

            // Top face
            qa.setCoordinateIndex(index++, c);
            qa.setCoordinateIndex(index++, g);
            qa.setCoordinateIndex(index++, h);
            qa.setCoordinateIndex(index++, d);

            // Bottom face
            qa.setCoordinateIndex(index++, a);
            qa.setCoordinateIndex(index++, e);
            qa.setCoordinateIndex(index++, f);
            qa.setCoordinateIndex(index++, b);

            // Update the indices, the last 4 become the new 4.
            a = e;
            b = f;
            c = g;
            d = h;
        }

        // Transform the IndexQuadArray geometry into GeometryInfo to be able to
        // generate the normals.
        GeometryInfo gi = new GeometryInfo(qa);
        NormalGenerator ng = new NormalGenerator();
        ng.generateNormals(gi);

        this.setGeometry(gi.getGeometryArray());
        this.setAppearance(app);
    }
}
```

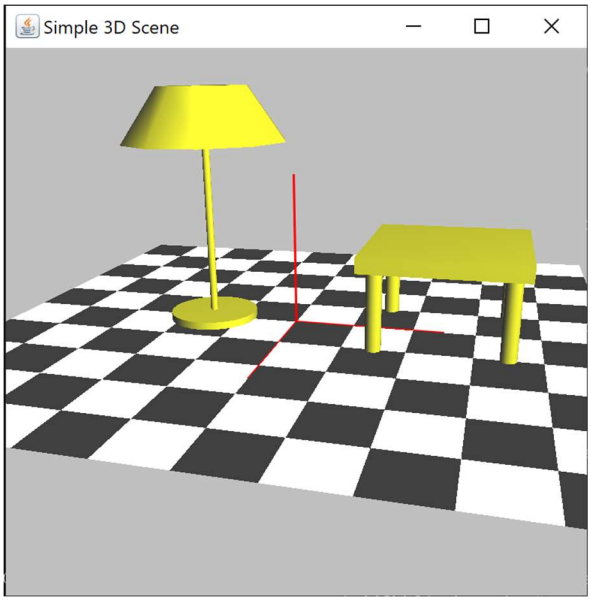

Figure 5 – Final 3D scene with the floor lamp and the table.

Add the following code to the FloorLamp constructor to finish the construction of the floor lamp.

```
// Top
Lampshade lampshade = new Lampshade(10, app);
tr = new Transform3D();
tr.setScale(0.5);
tr.setTranslation(new Vector3d(0, 1.2, 0));
```

```
tg = new TransformGroup(tr);
tg.addChild(lampshade);
this.addChild(tg);
```

Run the application. Figure 5 shows the final 3D scene with the floor lamp and the table.