CMSC427 fall 2017 Lab 5

Fan Yang

Tetrahedron: A tetrahedron consists of only 4 triangles and the 4 triangles have 12 vertices in total, so to draw a tetrahedron we only need to provide 12 vertices. They are actually 4 vertices but each vertex appears 3 times. Also, the piece of code "gl.glDrawArrays(GL_TRIANGLES, 0, 36)" was changed to "gl.glDrawArrays(GL_TRIANGLES, 0, 12)".

Cylinder: A double for-loop was used to create the slices of the cylinder. In the code I used 11 circles (10 slices) and each circle consists of 20 vertices, so there are in total 400 triangles and 1200 vertices. In the double for-loop, two triangles were generated at each vertex, which required 18 lines of code. The piece of code "gl.glDrawArrays(GL_TRIANGLES, 0, 36)" was changed to "gl.glDrawArrays(GL_TRIANGLES, 0, 1200)"

```
// Gordon program 4.1
// Shows display of cube
// Show mesh, transformations, camera, projection
package code;

import graphicslib3D.*;
import java.nio.*;
import javax.swing.*;
import static com.jogamp.opengl.GL4.*;
import com.jogamp.opengl.*;
import com.jogamp.opengl.awt.GLCanvas;
import com.jogamp.common.nio.Buffers;
import com.jogamp.opengl.GLContext;

public class Code extends JFrame implements GLEventListener
{  private GLCanvas myCanvas;
   private int rendering_program, count;
   private int vao[] = new int[1];
   private int vbo[] = new int[2];
   private float cameraX, cameraY, cameraZ;
   private float cubeLocX, cubeLocY, cubeLocZ;
   private GLSLUtils util = new GLSLUtils();

   public Code()
   {  setTitle("Chapter4 - program1a");
      setSize(600, 600);
      // Added for Mac OS
      GLProfile glp = GLProfile.getMaxProgrammableCore(true);
      GLCapabilities caps = new GLCapabilities(glp);
      myCanvas = new GLCanvas(caps);
      // Replaced
      //myCanvas = new GLCanvas();
      myCanvas.addGLEventListener(this);
      getContentPane().add(myCanvas);
```

```
  this.setVisible(true);
}

public void display(GLAutoDrawable drawable)
{  GL4 gl = (GL4) GLContext.getCurrentGL();
  gl.glClear(GL_DEPTH_BUFFER_BIT);
  gl.glUseProgram(rendering_program);

  int mv_loc = gl.glGetUniformLocation(rendering_program, "mv_matrix");
  int proj_loc = gl.glGetUniformLocation(rendering_program, "proj_matrix");

  float aspect = (float) myCanvas.getWidth() / (float) myCanvas.getHeight();
  Matrix3D pMat = perspective(60.0f, aspect, 0.1f, 1000.0f);

  Matrix3D vMat = new Matrix3D();
  vMat.translate(-cameraX, -cameraY, -cameraZ);

  Matrix3D mMat = new Matrix3D();
  mMat.translate(cubeLocX, cubeLocY, cubeLocZ);

  Matrix3D mvMat = new Matrix3D();
  mvMat.concatenate(vMat);
  mvMat.concatenate(mMat);

  gl.glUniformMatrix4fv(mv_loc, 1, false, mvMat.getFloatValues(), 0);
  gl.glUniformMatrix4fv(proj_loc, 1, false, pMat.getFloatValues(), 0);

  gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
  gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
  gl.glEnableVertexAttribArray(0);

  gl.glEnable(GL_DEPTH_TEST);
  gl.glDepthFunc(GL_LEQUAL);
  gl.glDrawArrays(GL_TRIANGLES, 0, count);
  System.out.println("Display");
}

public void init (GLAutoDrawable drawable)
{
  rendering_program = createShaderProgram();
  cameraX = 0.0f; cameraY = 0.0f; cameraZ = 10.0f;
  cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
  setupVertices();
}

private void setupVertices() {
   count = 36;
   float[] vertex_positions =
   {
    -1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,
```

```
     1.0f, -1.0f, -1.0f, 1.0f,  1.0f, -1.0f, -1.0f,  1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, 1.0f, -1.0f,  1.0f, 1.0f,  1.0f, -1.0f,
     1.0f, -1.0f,  1.0f, 1.0f,  1.0f,  1.0f, 1.0f,  1.0f, -1.0f,
     1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f, 1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f, -1.0f,  1.0f,  1.0f, 1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f, 1.0f,  1.0f, -1.0f, 1.0f,  1.0f,  1.0f,
     1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f, -1.0f
   };
   setupHelper(vertex_positions);
 }

 private void setupVerticesCylinder()
 {
    count = 1200;
    int numSlices = 11, numStacks = 20;
    float[] vertex_positions = new float[numSlices*numStacks*18];
    for(int i = 0; i < numSlices-1; i++)
       for (int j = 0; j < numStacks; j++) {
          vertex_positions[(i*numStacks+j)*18] = i - numSlices/2;
          vertex_positions[(i*numStacks+j)*18+1] = (float) Math.cos(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+2] = (float) Math.sin(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+3] = i - numSlices/2;
          vertex_positions[(i*numStacks+j)*18+4] = (float) Math.cos(Math.PI*2*(j+1)/numStacks);
          vertex_positions[(i*numStacks+j)*18+5] = (float) Math.sin(Math.PI*2*(j+1)/numStacks);
          vertex_positions[(i*numStacks+j)*18+6] = i - numSlices/2 + 1;
          vertex_positions[(i*numStacks+j)*18+7] = (float) Math.cos(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+8] = (float) Math.sin(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+9] = i - numSlices/2;
          vertex_positions[(i*numStacks+j)*18+10] = (float) Math.cos(Math.PI*2*(j+1)/numStacks);
          vertex_positions[(i*numStacks+j)*18+11] = (float) Math.sin(Math.PI*2*(j+1)/numStacks);
          vertex_positions[(i*numStacks+j)*18+12] = i - numSlices/2 + 1;
          vertex_positions[(i*numStacks+j)*18+13] = (float) Math.cos(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+14] = (float) Math.sin(Math.PI*2*j/numStacks);
          vertex_positions[(i*numStacks+j)*18+15] = i - numSlices/2 + 1;
          vertex_positions[(i*numStacks+j)*18+16] = (float) Math.cos(Math.PI*2*(j+1)/numStacks);
          vertex_positions[(i*numStacks+j)*18+17] = (float) Math.sin(Math.PI*2*(j+1)/numStacks);
       }
       setupHelper(vertex_positions);
 }

 private void setupVerticesTetra() {
    count = 12;
    float h = (float) Math.sqrt(3);
    float v = (float) Math.sqrt(6)*2/3;
    float[] vertex_positions =
    {
```

```
      -1.0f,  -h/3, 0.0f,    1.0f, -h/3, 0.0f,    0.0f, h*2/3, 0.0f,
      -1.0f,  -h/3, 0.0f,    1.0f, -h/3, 0.0f,    0.0f, 0.0f, v,
      -1.0f,  -h/3, 0.0f,    0.0f, h*2/3, 0.0f,   0.0f, 0.0f, v,
      1.0f,  -h/3, 0.0f,     0.0f, h*2/3, 0.0f,   0.0f, 0.0f, v
    };
    setupHelper(vertex_positions);
  }

  private void setupHelper(float[] vertex_positions){
    GL4 gl = (GL4) GLContext.getCurrentGL();
    gl.glGenVertexArrays(vao.length, vao, 0);
    gl.glBindVertexArray(vao[0]);
    gl.glGenBuffers(vbo.length, vbo, 0);

    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    FloatBuffer vertBuf = Buffers.newDirectFloatBuffer(vertex_positions);
    gl.glBufferData(GL_ARRAY_BUFFER, vertBuf.limit()*4, vertBuf, GL_STATIC_DRAW);
  }

 private Matrix3D perspective(float fovy, float aspect, float n, float f)
 {  float q = 1.0f / ((float) Math.tan(Math.toRadians(0.5f * fovy)));
   float A = q / aspect;
   float B = (n + f) / (n - f);
   float C = (2.0f * n * f) / (n - f);
   Matrix3D r = new Matrix3D();
   r.setElementAt(0,0,A);
   r.setElementAt(1,1,q);
   r.setElementAt(2,2,B);
   r.setElementAt(3,2,-1.0f);
   r.setElementAt(2,3,C);
   r.setElementAt(3,3,0.0f);
   return r;
 }

 public static void main(String[] args) { new Code(); }
 public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {}
 public void dispose(GLAutoDrawable drawable) {}

 private int createShaderProgram()
 {  GL4 gl = (GL4) GLContext.getCurrentGL();

   String vshaderSource[] = util.readShaderSource("code/vert.shader");
   String fshaderSource[] = util.readShaderSource("code/frag.shader");

   int vShader = gl.glCreateShader(GL_VERTEX_SHADER);
   int fShader = gl.glCreateShader(GL_FRAGMENT_SHADER);

   gl.glShaderSource(vShader, vshaderSource.length, vshaderSource, null, 0);
   gl.glShaderSource(fShader, fshaderSource.length, fshaderSource, null, 0);
```

```
    gl.glCompileShader(vShader);
    gl.glCompileShader(fShader);

    int vfprogram = gl.glCreateProgram();
    gl.glAttachShader(vfprogram, vShader);
    gl.glAttachShader(vfprogram, fShader);
    gl.glLinkProgram(vfprogram);
    return vfprogram;
  }
}
```