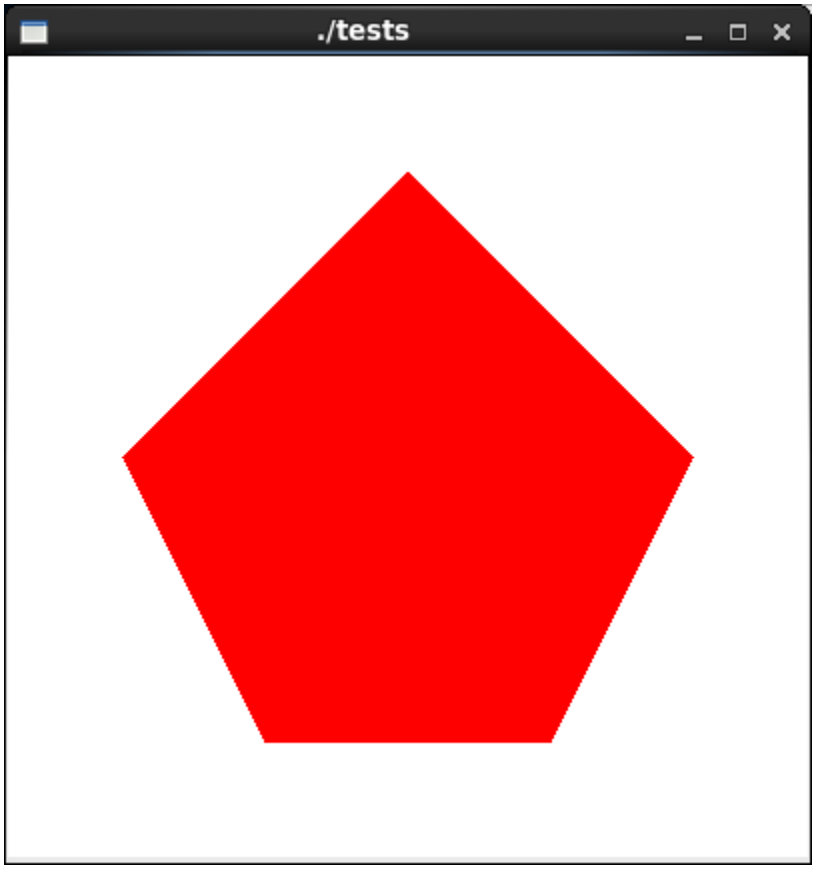
CSE520
Samuel Marrujo
Professor Yu
Lab 07

Draw a pentagon using GLSL

In this lab, we are to create a pentagon while hard-coding as a string in a GLSL program. I have successfully completed this task, and here is a picture of the following:



```
/*
  tests.cpp
  Sample program showing how to write GL shader programs.
  Shader sources are in files "tests.vert" and "tests.frag".
  @Author: T.L. Yu, 2008
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

#define GLEW_STATIC 1
#include <GL/glew.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <string.h>

using namespace std;

/*
   Global handles for the currently active program object, with its two shader
objects
*/
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;

int readVertexShaderSource( GLchar **shader )
{
    // Allocate memory to hold the source of our shaders.

    char str[] = " \
                  attribute vec3 temp; \
                        varying vec3 color; \
                  void main(void){ \
                              color = temp;\
                  gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;  \
           } ";

    int len = strlen ( str );
    *shader = (GLchar *) malloc( len + 1);
    strcpy ( *shader, str );
    (*shader)[len] = '\0';

    return 1;
}

int readFragmentShaderSource( GLchar **shader )
{
    // Allocate memory to hold the source of our shaders.

    char str[] = "      varying vec3 color;\
                  void main(void){   \
        gl_FragColor = vec4( color, 1); \
      } ";
```

```
    int len = strlen ( str );
    *shader = (GLchar *) malloc( len + 1);

    strcpy ( *shader, str );

    (*shader)[len] = '\0';

    return 1;
}

//  public
int installShaders(const GLchar *vertex, const GLchar *fragment)
{

    printf("------------\n");
     printf("%s\n", vertex );
     printf("%s", fragment );
    printf("\n-------------\n");
    GLint  vertCompiled, fragCompiled;  // status values
    GLint  linked;

    // Create a vertex shader object and a fragment shader object

    vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
    fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);

    // Load source code strings into shaders, compile and link

    glShaderSource(vertexShaderObject, 1, &vertex, NULL);
    glShaderSource(fragmentShaderObject, 1, &fragment, NULL);

    glCompileShader(vertexShaderObject);
    glGetShaderiv(vertexShaderObject, GL_COMPILE_STATUS, &vertCompiled);

    glCompileShader( fragmentShaderObject );
    glGetShaderiv( fragmentShaderObject, GL_COMPILE_STATUS, &fragCompiled);

    printf("vertCompiled, fragCompiled: %d, %d\n", vertCompiled, fragCompiled);
    if (!vertCompiled || !fragCompiled)
        return 0;

    // Create a program object and attach the two compiled shaders

    programObject = glCreateProgram();
    glAttachShader( programObject, vertexShaderObject);
    glAttachShader( programObject, fragmentShaderObject);

    // Link the program object

    glLinkProgram(programObject);
    glGetProgramiv(programObject, GL_LINK_STATUS, &linked);

    printf("linked=%d\n");
    if (!linked)
        return 0;

    // Install program object as part of current state
```

```
    glUseProgram(programObject);


    return 1;
}

int init(void)
{

   const char *version;
   GLchar *VertexShaderSource, *FragmentShaderSource;
   int loadstatus = 0;

   version = (const char *) glGetString(GL_VERSION);
   if (version[0] != '2' || version[1] != '.') {
      printf("This program requires OpenGL 2.x, found %s\n", version);
    //  exit(1);
   }
   readVertexShaderSource( &VertexShaderSource );
   readFragmentShaderSource( &FragmentShaderSource );
   loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);

   return loadstatus;
}

static void Reshape(int width, int height)
{
   glViewport(0, 0, width, height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0, 1.0, -1.0, 1.0, 5.0, 25.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(0.0f, 0.0f, -15.0f);
}

void CleanUp(void)
{
   glDeleteShader(vertexShaderObject);
   glDeleteShader(fragmentShaderObject);
   glDeleteProgram(programObject);
   glutDestroyWindow(win);
}

static void Idle(void)
{
   glutPostRedisplay();
}


static void Key(unsigned char key, int x, int y)
{
   switch(key) {
   case 27:
      CleanUp();
      exit(0);
      break;
```

```
   }
   glutPostRedisplay();
}

void display(void)
{
   GLfloat vec[4];
   int loc;
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glClearColor( 1.0, 1.0, 1.0, 1.0 );    //get white background color
   glColor3f ( 0, 0, 1 );           //red, this will have no effect if shader is
loaded
  // glutWireSphere(2.0, 10, 5);
   loc = glGetAttribLocation(programObject, "temp" );
   glPointSize ( 4 );
//   glBegin (GL_POINTS);  //need GL_POINTS; "GL_POINT" doesn't work

   glBegin ( GL_POLYGON );
     glVertexAttrib3f(loc,1,0,0);
     glVertex3f(-1, -2, 1);
     glVertexAttrib3f(loc,1,0,0);
     glVertex3f( 1,  -2, 1);
     glVertexAttrib3f(loc,1,0,0);
     glVertex3f( 2, 0, 1);
     glVertexAttrib3f(loc,1,0,0);
     glVertex3f( 0,  2, 1);
     glVertexAttrib3f(loc,1,0,0);
     glVertex3f( -2, 0, 1);
   glEnd();
   glutSwapBuffers();
   glFlush();
}


int main(int argc, char *argv[])
{
   int success = 0;

   glutInit(&argc, argv);
   glutInitWindowPosition( 0, 0);
   glutInitWindowSize(400, 400);
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
   win = glutCreateWindow(argv[0]);
   glutReshapeFunc(Reshape);
   glutKeyboardFunc(Key);
   glutDisplayFunc(display);
   glutIdleFunc(Idle);

   // Initialize the "OpenGL Extension Wrangler" library
   glewInit();

   success = init();
   printf("success=%d\n", success );
   if ( success )
     glutMainLoop();
   return 0;
}
```