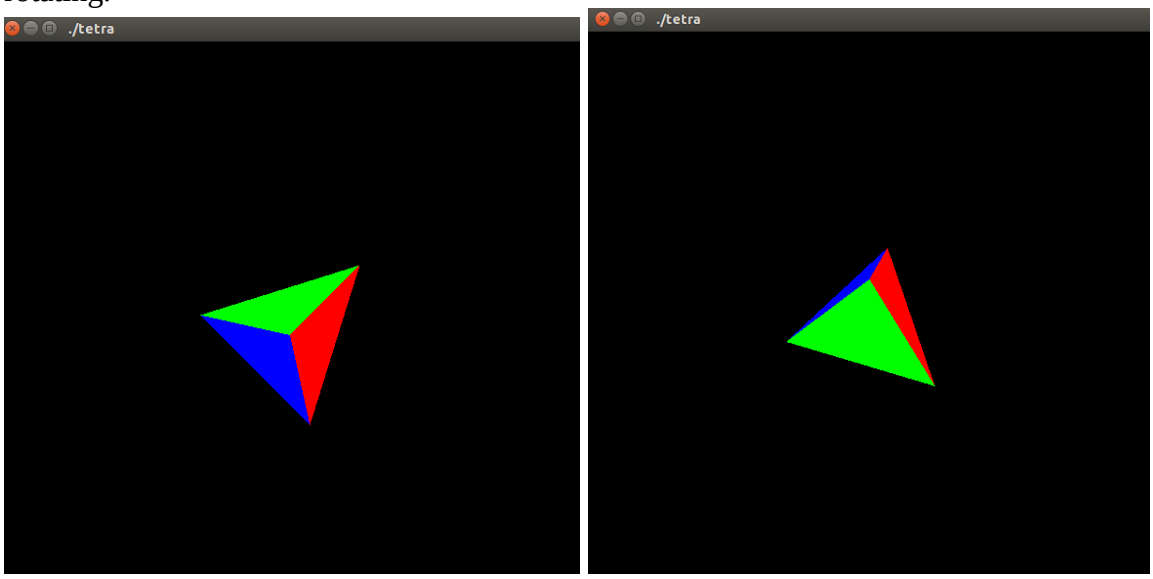
Erik Anchondo
2-6-19
cse 520
lab 4

1. Wrote a glsl program that displays a colored tetrahedron. The tetrahedron starts to rotate on the x axis when the mouse button is clicked once. If the mouse button is a second time, it starts rotating on the y axis. If the mouse button is clicked a third time, it starts rotating on it z axis. After the fourth time of clicking the mouse button, it stops rotating.



Here is the source code:

```
#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>

using namespace std;

GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;

//colors
```

```
int triangle_1 = 1;
int triangle_2 = 2;
int triangle_3 = 3;

//rotations
float theta[3];
int  thetaLoc[3];
int  aLoc;
float x = 0;
float y = 0;
float z = 0;

//scaling
float thes[3];
int thesLoc[3];
int sLoc;
float s_x = 1;
float s_y = 1;
float s_z = 1;

//animation
int rot_state = 0;
float animation_speed = 0.10;
int timeCounter = 0;    // time counter
int time_interval = 20;// how much time has to be before action is done

int readShaderSource(char *fileName, GLchar **shader )
{
   FILE *fp;
   int count, pos, shaderSize;
   fp = fopen( fileName, "r");
   if ( !fp )
   {
      return 0;
   }
   pos = (int) ftell ( fp );
   fseek ( fp, 0, SEEK_END );
   shaderSize = ( int ) ftell ( fp ) - pos;
   fseek ( fp, 0, SEEK_SET );
   if ( shaderSize <= 0 )
   {
      printf("Shader %s empty\n", fileName);
      return 0;
   }
   *shader = (GLchar *) malloc( shaderSize);
```

```
    if ( *shader == NULL )
    {
        printf("memory allocation error\n");
    }
    count = (int) fread(*shader, 1, shaderSize, fp);
    (*shader)[count] = '\0';
    if (ferror(fp))
    {
       count = 0;
    }
    fclose(fp);
    return 1;
}

int installShaders(const GLchar *vertex, const GLchar *fragment)
{
    GLint  vertCompiled, fragCompiled;  // status values
    GLint  linked;
    vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
    fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);
    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;
    }
    programObject = glCreateProgram();
    glAttachShader( programObject, vertexShaderObject);
    glAttachShader( programObject, fragmentShaderObject);
    glLinkProgram(programObject);
    glGetProgramiv(programObject, GL_LINK_STATUS, &linked);
    if (!linked)
    {
       return 0;
    }
    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);
}
readShaderSource( (char *) "tetra.vert", &VertexShaderSource );
readShaderSource( (char *) "tetra.frag", &FragmentShaderSource );
loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);
 glUseProgram(programObject);

 //colors
 GLchar names[][20] = {"red", "green", "blue"};
 GLint loc[10];
 for ( int i = 0; i < 3; ++i )
 {
       loc[i] =  glGetUniformLocation(programObject, names[i]);
       if (loc[i] == -1)
       {
       printf("No such uniform named %s\n", names[i]);
       }
 }
 glUniform3f(loc[0], 1.0, 0.0, 0.0);
 glUniform3f(loc[1], 0.0, 1.0, 0.0);
 glUniform3f(loc[2], 0.0, 0.0, 1.0);

 //scaling
 GLchar names2[][20] = {"sx","sy","sz"};
 for ( int i = 0; i < 3; i++ )
 {
       thesLoc[i] = glGetAttribLocation ( programObject, names2[i] );
       if ( thesLoc[i] < 0 )
       {
                printf("No such name %s\n", names2[i]);
       }
 }

 //rotations
 char names1[][20] = {"ax", "ay", "az"};
 for ( int i = 0; i < 3; i++ )
 {
       thetaLoc[i] = glGetAttribLocation ( programObject, names1[i] );
       if ( thetaLoc[i] < 0 )
       {
```

```
                printf("No such name %s\n", names1[i]);
        }
    }

  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)
{
        if (timeCounter < time_interval)
        {
                timeCounter++;
        }
        else
        {
                timeCounter = 0;
                if (rot_state == 1)
                {
                        x += animation_speed;
                }
                if (rot_state == 2)
                {
                        y += animation_speed;
                }
                if (rot_state == 3)
                {
                        z += animation_speed;
```

```
                }
                if (rot_state >= 4)
                {
                        rot_state = 0;
                }
        }
        glutPostRedisplay();
}

void myMouse( int button, int state, int x, int y )
{
        if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN )
        {
                rot_state++;
        }
        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( 0.0, 0.0, 0.0, 0.0 );
  glColor3f ( 1, 0, 0 );
  loc = glGetAttribLocation(programObject, "vertex_color");
  aLoc = glGetAttribLocation(programObject, "a");
  sLoc = glGetAttribLocation(programObject, "s");

  glBegin ( GL_TRIANGLES );
        glVertexAttrib3f (aLoc, x, y, z);               //rotate
        glVertexAttrib3f (sLoc, s_x, s_y, s_z);                 //scale

        glVertexAttrib1f(loc, triangle_1);              //set color to triangle
```

```
                glVertex3f (0.7, 0.7, 0.3);
                glVertex3f (-0.9, 0.2, 0.3);
                glVertex3f (0.2, -0.9, 0.3);

        glVertexAttrib1f(loc, triangle_2);          //set color to triangle
                glVertex3f (0.7, 0.7, 0.3);
                glVertex3f (-0.9, 0.2, 0.3);
                glVertex3f (0.0, 0.0, -1.0);

        glVertexAttrib1f(loc, triangle_3);          //set color to triangle
                glVertex3f (-0.9, 0.2, 0.3);
                glVertex3f (0.2, -0.9, 0.3);
                glVertex3f (0.0, 0.0, -1.0);

        glVertexAttrib1f(loc, triangle_1);          //set color to triangle
                glVertex3f (0.2, -0.9, 0.3);
                glVertex3f (0.7, 0.7, 0.3);
                glVertex3f (0.0, 0.0, -1.0);
  glEnd();

  glutSwapBuffers();
  glFlush();
}

int main(int argc, char *argv[])
{
  int success = 0;
  glutInit(&argc, argv);
  glutInitWindowPosition( 0, 0);
  glutInitWindowSize(600, 600);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  win = glutCreateWindow(argv[0]);
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(Key);
  glutDisplayFunc(display);
  glutIdleFunc(Idle);
  glutMouseFunc( myMouse );
   glewInit();
  success = init();
  printf("success=%d\n", success );
  if ( success )
  {
        glutMainLoop();
  }
  return 0;
}
```

Here is the vert file:

```
attribute float vertex_color;    //get color from cpp
attribute vec4 a;                //get rotation x,y,z from cpp
attribute vec4 s;
varying float color_select;      //pass to frag shader

void main(void)
{
        vec4 v4;
        v4 = gl_Vertex;
        color_select = vertex_color;

        mat4 m0 = mat4                  //x rotation
        (
                1, 0, 0, 0,
                0, cos(a.x), -sin(a.x), 0,
                0, sin(a.x), cos(a.x), 0,
                0, 0, 0, 1
        );

        mat4 m1 = mat4                  //y rotation
        (
                cos(a.y), 0, sin(a.y), 0,
                0, 1, 0, 0,
                -sin(a.y), 0, cos(a.y), 0,
                0, 0, 0, 1
        );

        mat4 m2 = mat4                  //z rotation
        (
                cos(a.z), -sin(a.z), 0, 0,
                sin(a.z), cos(a.z), 0, 0,
                0, 0, 1, 0,
                0, 0, 0, 1
        );

        mat4 m3 = mat4                  //scale
        (
                s.x, 0, 0, 0,
                0, s.y, 0, 0,
                0, 0, s.z, 0,
                0, 0, 0, 1
        );

        v4 = m0 * m1 * m2 * m3 * v4;
```

```
	gl_Position = gl_ProjectionMatrix * gl_ModelViewMatrix * v4;
}
```

Here is the frag file:

```
uniform vec3 red;
uniform vec3 green;
uniform vec3 blue;
varying float color_select;
void main(void)
{
	vec3 color;
	if (color_select < 1.1)			//red
	{
		color = red;
	}
	else if (color_select > 2.9)		//green
	{
		color = blue;
	}
	else					//blue
	{
		color = green;
	}
	gl_FragColor = vec4(color, 1.0);
}
```

summary:

This lab shows how to rotate na object along the x, y, and z axis use glsl shaders. This also shows how to get mouse input from the user. I have followed all the instructions for this lab and I believe that I should receive the full 20 points for this assignment.