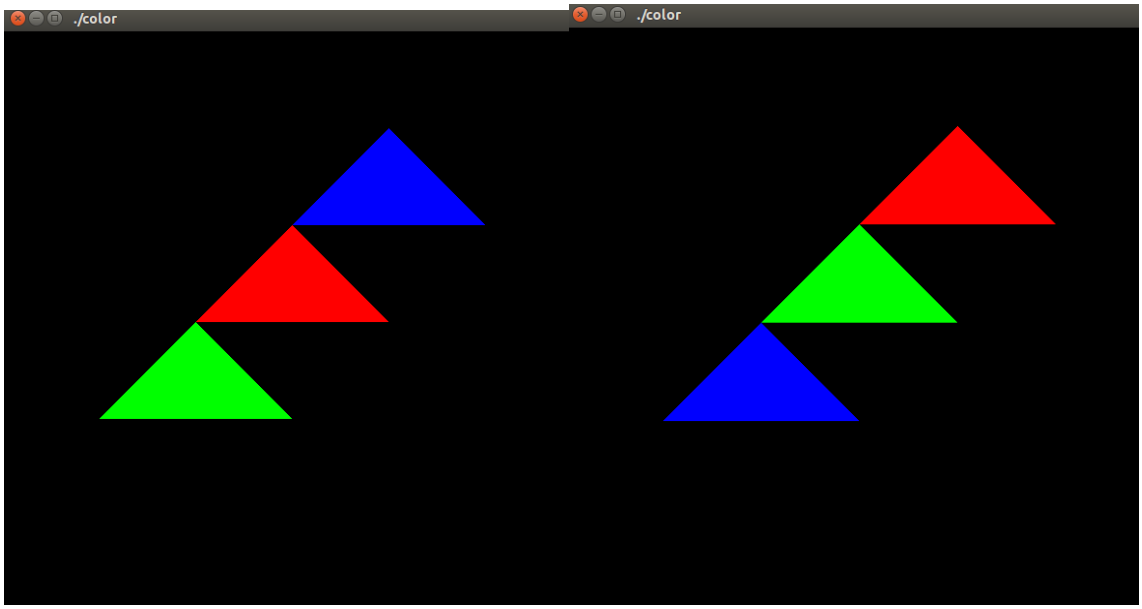
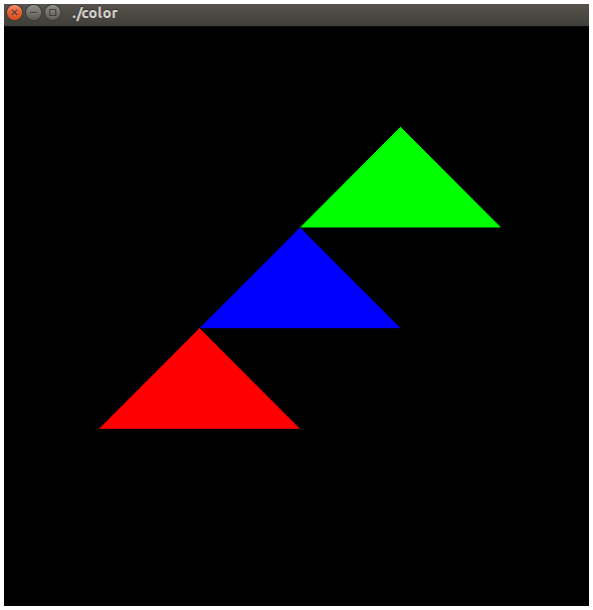
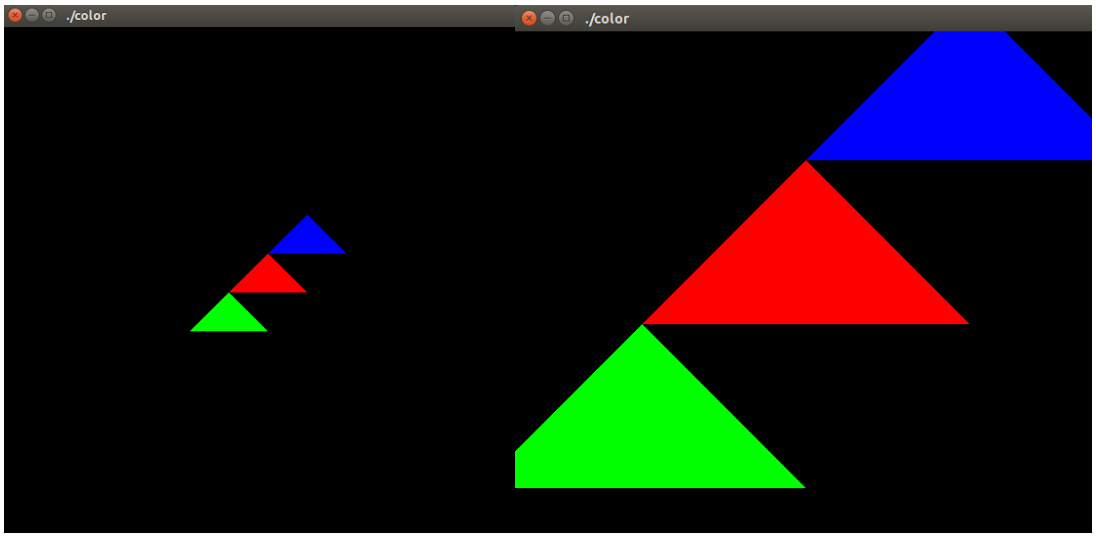
Erik Anchondo
1-29-19
cse 520
lab 3

1. Wrote a shader program to display three colored triangles, with one of each triangle being red, green, and blue. The colors change which triangle they are applied to when the user presses the 't' key.

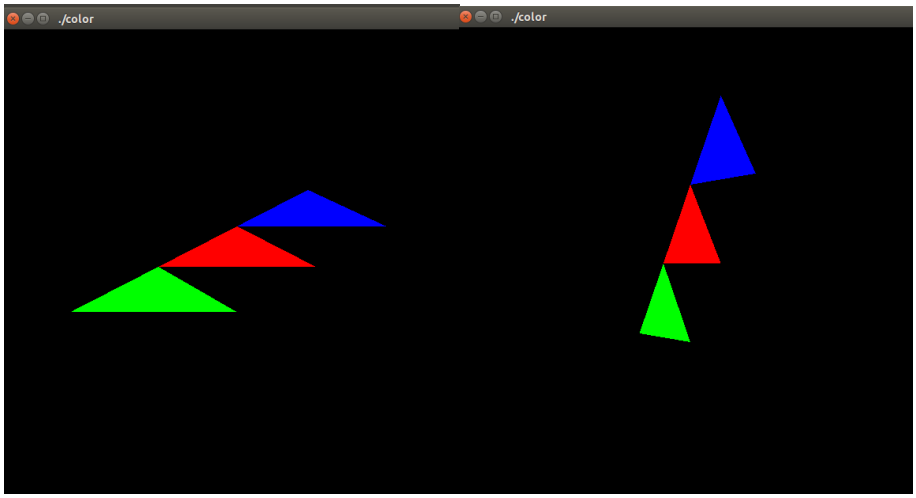


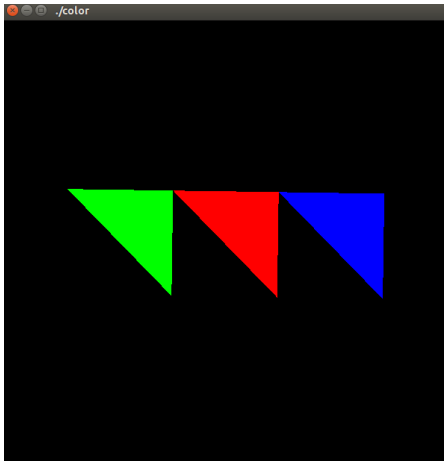


2a. The triangles scale increases (expands) when the 'e' key is pressed and decreases (contracts) when the 'c' key is pressed.



2b. The triangles rotate in the positive direction along the x, y, and z axis when the user presses the 'x', 'y', and 'z' keys. The triangles also rotates in the negative direction when the user presses the 'X', 'Y', 'Z' keys.





Here is the full source code to the program:

```
//color.cpp
#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;

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 *) "color.vert", &VertexShaderSource );  //link vert file
   readShaderSource( (char *) "color.frag", &FragmentShaderSource );   //link frag file
   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)
{
  glutPostRedisplay();
}
void change_color()
{
        triangle_1++;
        triangle_2++;
        triangle_3++;
        if (triangle_1 > 3){triangle_1 = 1;}
        if (triangle_2 > 3){triangle_2 = 1;}
        if (triangle_3 > 3){triangle_3 = 1;}
        glutPostRedisplay();
}
void rotate_object(char rotation_axis, float degrees, int rotation_dir)
{
        degrees *= rotation_dir;
        if (rotation_axis == 'x'){x += degrees;}
        if (rotation_axis == 'y'){y += degrees;}
        if (rotation_axis == 'z'){z += degrees;}
        glutPostRedisplay();
}
void scale_object(float scale_size, int scale_dir)
{
        scale_size *= scale_dir;
        s_x += scale_size;
        s_y += scale_size;
        s_z += scale_size;
        glutPostRedisplay();
}
static void Key(unsigned char key, int x, int y)
{
  switch(key)
  {
        case 't':
                change_color();
                break;
        case 'x':
                rotate_object('x', 0.2, 1);
                break;
        case 'y':
                rotate_object('y', 0.2, 1);
                break;
        case 'z':
```

```
                rotate_object('z', 0.2, 1);
                break;
        case 'e':
                scale_object(0.020, 1);
                break;
        case 'c':
                scale_object(0.020, -1);
                break;
        case 'X':
                rotate_object('x', 0.2, -1);
                break;
        case 'Y':
                rotate_object('y', 0.2, -1);
                break;
        case 'Z':
                rotate_object('z', 0.2, -1);
                break;
        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 );
        if (loc == -1)
        {
                printf("No such attribute named %s\n", "vertex_color");
        }

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

        glVertexAttrib1f(loc, triangle_1);              //set color to triangle
                glVertex3f (0.0, 1.0, 0.0);
                glVertex3f (1.0, 0.0, 0.0);
```

```
                glVertex3f (-1.0, 0.0, 0.0);

        glVertexAttrib1f(loc, triangle_2);          //set color to triangle
                glVertex3f (0.0 - 1, 1.0 - 1, 0.0);
                glVertex3f (1.0 - 1, 0.0 - 1, 0.0);
                glVertex3f (-1.0 - 1, 0.0 - 1, 0.0);

        glVertexAttrib1f(loc, triangle_3);          //set color to triangle
                glVertex3f (0.0 + 1, 1.0 + 1, 0.0);
                glVertex3f (1.0 + 1, 0.0 + 1, 0.0);
                glVertex3f (-1.0 + 1, 0.0 + 1, 0.0);
  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);
   glewInit();
  success = init();
  printf("success=%d\n", success );
  if ( success )
  {
        glutMainLoop();
  }
  return 0;
}
```

Here is the vert. shader code:

```
//color.vert
attribute float vertex_color;   //get color from cpp
attribute vec4 a;               //get rotation x,y,z from cpp
attribute vec4 s;               //get scaling x,y,z from cpp
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
	(
		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. shader code:

```
//color.frag
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:

The purpose for this lab was to show how color changing, rotations, and scaling animations work using GLSL shaders. This will be extremely useful for the project.