Daniel Meyer

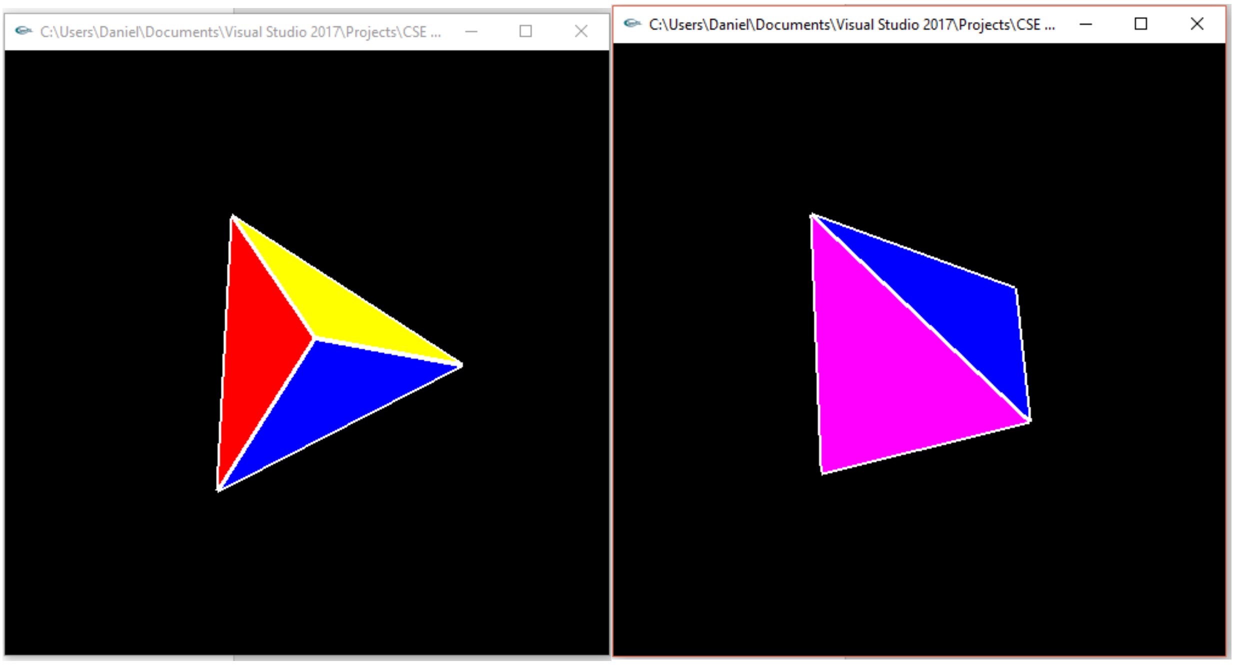
CSE 520

Tong Yu

Color Tetrahedron

Lab 4

Part 1 (success):


C:\Users\Daniel\Documents\Visual Studio 2017\Projects\CSE ...
C:\Users\Daniel\Documents\Visual Studio 2017\Projects\CSE ...

Colored tetrahedron rotated using the mouse

Lab4.cpp

```
/*
   Lab4.cpp
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>

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

```
#include "Shader.h"

using namespace std;

#define PI 3.14159265359

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

bool mouseDown = false;
//For XYZ rotation using different mouse buttons Pt.1
//int mouseDown = 0;

float xrot = 0.0;
float yrot = 0.0;
float zrot = 0.0;

float xdiff = 0.0;
float ydiff = 0.0;
float zdiff = 0.0;

const int screenWidth = 500;
const int screenHeight = 500;
float anglex = 0, angley = 0, anglez = 0;          //rotation angles

int rLoc;
int cLoc;

/*
// Vertex co-ordinate vectors for the tetrahedron.
static float vertices[] =
{
       1.0,  1.0,  1.0, // V0
      -1.0,  1.0, -1.0, // V1
       1.0, -1.0, -1.0, // V2
      -1.0, -1.0,  1.0  // V3
};

// Vertex indices for the four trianglular faces.
static int triangleIndices[4][3] =
{
      {1, 2, 3}, // F0
```

```
    {0, 3, 2}, // F1
    {0, 1, 3}, // F2
    {0, 2, 1}  // F3
};

static int colors[] =
{
    1.0, 0.0, 0.0,   //Red
    0.0, 1.0, 0.0,   //Green
    0.0, 0.0, 1.0,   //Blue
    1.0, 1.0, 0.0    //Yellow
};
*/

int init(void)
{
    glEnable(GL_DEPTH_TEST);
    /*
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_COLOR_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3, GL_FLOAT, 0, colors);
    */

    const char *version;
    char *VertexShaderSource, *FragmentShaderSource;
    string *vs, *fs;
    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);
    }
    printf("version=%s\n", version);

    shader.readShaderFile((char *) "Lab4.vert", &VertexShaderSource);
    shader.readShaderFile((char *) "Lab4.frag",
&FragmentShaderSource);

    vs = new string(VertexShaderSource);
    fs = new string(FragmentShaderSource);

    loadstatus = shader.createShader(vs, fs);
    delete fs;
    delete vs;
    delete VertexShaderSource;
    delete FragmentShaderSource;

    rLoc = glGetAttribLocation(shader.programObject, "rotate");
    cLoc = glGetAttribLocation(shader.programObject, "vColor");

    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)
{
    shader.cleanUp();
    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;
    case 'x':
        anglex += 2.0 * (PI / 180);
        break;
    case 'X':
        anglex -= 2.0 * (PI / 180);
        break;
    case 'y':
        angley += 2.0 * (PI / 180);
        break;
    case 'Y':
        angley -= 2.0 * (PI / 180);
        break;
    case 'z':
        anglez += 2.0 * (PI / 180);
        break;
    case 'Z':
        anglez -= 2.0 * (PI / 180);
        break;
    case 'r':
        anglex = angley = anglez = 0;
        break;
    }
    glutPostRedisplay();
}

void display(void)
{
    GLfloat vec[4];

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(0.0, 0.0, 0.0, 0.0);    //get white background
color

    glVertexAttrib3f(rLoc, xrot * 0.05, yrot * 0.05, anglez);

    // Draw tetrahedron.
    glBegin(GL_TRIANGLES);
    //front triangle
    glVertexAttrib3f(cLoc, 1.0, 0.0, 0.0);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glVertexAttrib3f(cLoc, 1.0, 0.0, 0.0);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertexAttrib3f(cLoc, 1.0, 0.0, 0.0);
    glVertex3f(-1.0f, -1.0f, 1.0f);

    //right side triangle
    glVertexAttrib3f(cLoc, 1.0, 1.0, 0.0);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertexAttrib3f(cLoc, 1.0, 1.0, 0.0);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glVertexAttrib3f(cLoc, 1.0, 1.0, 0.0);
```

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

//left side triangle
glVertexAttrib3f(cLoc, 1.0, 0.0, 1.0);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertexAttrib3f(cLoc, 1.0, 0.0, 1.0);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertexAttrib3f(cLoc, 1.0, 0.0, 1.0);
glVertex3f(-1.0f, -1.0f, 1.0f);

//bottom triangle
glVertexAttrib3f(cLoc, 0.0, 0.0, 1.0);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertexAttrib3f(cLoc, 0.0, 0.0, 1.0);
glVertex3f(1.0f, -1.0f, -1.0f);
glVertexAttrib3f(cLoc, 0.0, 0.0, 1.0);
glVertex3f(-1.0f, 1.0f, -1.0f);
glEnd();

// Draw tetrahedron Outline.
glLineWidth(4.0);
glBegin(GL_LINE_STRIP);
//front triangle
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(1.0f, -1.0f, -1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(-1.0f, -1.0f, 1.0f);

//right side triangle
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(1.0f, -1.0f, -1.0f);

//left side triangle
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
glVertex3f(-1.0f, -1.0f, 1.0f);
```

```
    //bottom triangle
    glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glEnd();


    /*
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < 4; ++i)
    {
        glVertexAttrib3f(cLoc, 1.0, 0.0, 0.0);
        glArrayElement(triangleIndices[i][0]);
        glVertexAttrib3f(cLoc, 1.0, 1.0, 0.0);
        glArrayElement(triangleIndices[i][1]);
        glVertexAttrib3f(cLoc, 1.0, 0.0, 1.0);
        glArrayElement(triangleIndices[i][2]);
    }
    glEnd();

    glLineWidth(4);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < 4; ++i)
    {
        glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
        glArrayElement(triangleIndices[i][0]);
        glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
        glArrayElement(triangleIndices[i][1]);
        glVertexAttrib3f(cLoc, 1.0, 1.0, 1.0);
        glArrayElement(triangleIndices[i][2]);
    }
    glEnd();
    */

    glutSwapBuffers();
    glFlush();
}

void movedMouse(int mouseX, int mouseY)
{

    glutPostRedisplay();
}
```

```
void mouse(int button, int state, int x, int y)
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		mouseDown = true;
		xdiff = x - yrot;
		ydiff = -y + xrot;
	}

	/*
	//For XYZ rotation using different mouse buttons Pt.2
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		//mouseDown = true;
		mouseDown = 1;
		xdiff = x - yrot;
		//ydiff = -y + xrot;
	}
	else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
	{
		//mouseDown = true;
		mouseDown = 2;
		ydiff = -y + xrot;
	}
	else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
	{
		mouseDown = 3;
		//zdiff = x - zrot;
		anglez += 5.0 * (PI / 180);
	}
	else
	{
		mouseDown = 0;
		//mouseDown = false;
	}
	*/
}

void mouseMotion(int x, int y)
{
	if (mouseDown == true)
	{
		yrot = x - xdiff * (PI / 180);
		xrot = y + ydiff * (PI / 180);
```

```
        glutPostRedisplay();
    }

    /*
    //For XYZ rotation using different mouse buttons Pt.3
    if (mouseDown == 1)
    {
        yrot = x - xdiff * (PI / 180);
        //xrot = y + ydiff * (PI / 180);

        //glutPostRedisplay();
    }
    else if (mouseDown == 2)
    {
        xrot = y + ydiff * (PI / 180);
    }
    else if (mouseDown == 3)
    {
        zrot = x - zdiff * (PI / 180);
    }
    glutPostRedisplay();
    */
}

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

    glutInit(&argc, argv);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(screenWidth, screenHeight);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    win = glutCreateWindow(argv[0]);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutDisplayFunc(display);
    //glutIdleFunc(Idle);

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

    success = init();
    if (success)
        glutMainLoop();
```

```
      else
      {
            printf("infoLog:: %s\n", shader.infoLog);
      }
      return 0;
}
```

Lab4.vert

```
//Lab4.vert
//a minimal vertex shader
attribute vec3 rotate;
attribute vec3 vColor;
varying vec4 fColor;

void main(void)
{
      vec4 v4 = gl_Vertex;

      mat4 mRotateX = mat4 ( 1, 0, 0, 0,//1st col
                        0, cos(rotate.x), sin(rotate.x), 0,  //2nd col
                        0, -sin(rotate.x), cos(rotate.x), 0,
//3rd col
                        0, 0, 0, 1  );      //4th col

      mat4 mRotateY = mat4 ( cos(rotate.y), 0, -sin(rotate.y), 0,//1st
col
                        0, 1, 0, 0,  //2nd col
                        sin(rotate.y), 0, cos(rotate.y), 0,
//3rd col
                        0, 0, 0, 1  );      //4th col

      mat4 mRotateZ = mat4 ( cos(rotate.z), sin(rotate.z), 0, 0,//1st
col
                        sin(rotate.z), cos(rotate.z), 0, 0,  //2nd col
                        0, 0, 1, 0,         //3rd col
                        0, 0, 0, 1  );      //4th col

      v4 = mRotateZ * mRotateY * mRotateX * v4;

      fColor = vec4(vColor, 1.0);
    gl_Position = gl_ProjectionMatrix *  gl_ModelViewMatrix * v4;
}
```

Lab4.frag

```
//Lab4.frag
//a minimal fragment shader
varying vec4 fColor;

void main(void)
{
	gl_FragColor = fColor;
	//gl_FragColor = vec4( 1, 0, 0, 1);   // color
}
```

Shader.h

```
#ifndef _SHADER_H
#define _SHADER_H
#include <string>
#include <GL/glew.h>
#include <GL/glu.h>
#include <GL/glut.h>

using namespace std;

class Shader
{
public:
	int programObject;
	int vertexShaderObject;
	int fragmentShaderObject;
	char *infoLog;
	Shader();
	~Shader();
	int  loadShader(int shaderType, const string *shaderCode);
	bool createShader(const string *vs, const string *fs);
	int  readShaderFile(char *fileName, char **shader);
	void cleanUp();
};

#endif
```

Shader.cpp

```
/*  Shader.cpp
 *  Compile: g++ -c Shader.cpp
 */
#include <string>
#include <GL/glew.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include "Shader.h"
#include <stdio.h>
#include <malloc.h>
#include <iostream>

using namespace std;

Shader::Shader()
{
     infoLog = NULL;
}

Shader::~Shader()
{
     if (infoLog != NULL)
          delete infoLog;
}
// Create a Shader object
// User provides vertex shader code (vs) and fragement shader code
(fs)
bool  Shader::createShader(const string *vs, const string *fs)
{
     // create empty OpenGL   Program,load,attach,and link shaders
     programObject = glCreateProgram();
     if (vs != NULL) {
          vertexShaderObject = loadShader(GL_VERTEX_SHADER, vs);
          // add the vertex shader to program
          glAttachShader(programObject, vertexShaderObject);
     }
     if (fs != NULL) {
          fragmentShaderObject = loadShader(GL_FRAGMENT_SHADER, fs);
          // add the fragment shader to program
          glAttachShader(programObject, fragmentShaderObject);
     }
     glLinkProgram(programObject); // creates   program executables
     int linked;
     glGetProgramiv(programObject, GL_LINK_STATUS, &linked);

     if (!linked) {
          printf("Shader not linked!\n");
          return false;
     }

     glUseProgram(programObject); // use shader program

     return true;
}

int Shader::loadShader(int shaderType, const string *shaderCode)
{

     // create a vertex shader type ( GL_VERTEX_SHADER)
     // or a fragment shader type ( GL_FRAGMENT_SHADER)
     int shader = glCreateShader(shaderType);

     // pass source code to the shader and compile it
     char *strPointer = (char *)shaderCode->c_str();
     glShaderSource(shader, 1, &strPointer, NULL);
     glCompileShader(shader);
     int compiled;
     glGetShaderiv(shader, GL_COMPILE_STATUS, &compiled);
     printf("Shader type=%d\n", shaderType );
     if (!compiled)
          printf("Compiling %d failed!\n", shaderType);
     else
     {
          printf("------------\n");
          printf("%s", strPointer);
          printf("\n--------------\n");
     }

     int maxLength;
     glGetShaderiv(shaderType, GL_INFO_LOG_LENGTH, &maxLength);

     // maxLength includes NULL character
     infoLog = (char *)malloc(sizeof(char) * maxLength);
     glGetShaderInfoLog(vertexShaderObject, maxLength, &maxLength,
infoLog);

     //cout << infoLog << endl;

     return shader;
}

void Shader::cleanUp()
```

```
{
    glDeleteProgram(programObject);
    glDeleteShader(vertexShaderObject);
    glDeleteShader(fragmentShaderObject);
}

int Shader::readShaderFile(char *fileName, char **shader)
{
    // Allocate memory to hold the source of our shaders.
    FILE *fp;
    int count, pos, shaderSize;

    fp = fopen(fileName, "r");
    if (!fp)
        return 0;

    pos = (int)ftell(fp);
    fseek(fp, 0, SEEK_END);                //move to end
    shaderSize = (int)ftell(fp) - pos;     //calculates file size
    fseek(fp, 0, SEEK_SET);                //rewind to beginning

    if (shaderSize <= 0) {
        printf("Shader %s empty\n", fileName);
        return 0;
    }

    *shader = (char *)malloc(shaderSize + 1);

    if (*shader == NULL)
        printf("memory allocation error\n");
    // Read the source code
    count = (int)fread(*shader, 1, shaderSize, fp);
    (*shader)[count] = '\0';

    if (ferror(fp))
        count = 0;

    fclose(fp);

    return 1;
}
```

Summary:

For this assignment we had to create a tetrahedron and color its different surfaces. Each surface had to be a different solid color using the fragment shader. The next party of the assignment was to add functionality for rotating the tetrahedron using the mouse. For this I created two different methods of rotating the object. The first method (most user-friendly) is currently implemented allows the user to hold down either left or right mouse button and rotate the object. The other implementation (commented out with respective tags; 3 parts) allows user to rotate using the left mouse button (x-axis), right mouse button (y-axis), and middle mouse button (z-axis). Initially there were some difficulties using vertex arrays to draw the object and then coloring the fragment shader which led to each vertex being colored and the fragment shader filling in using interpolation. My solution to this was to manually enter each vertex and color each one to ensure solid colors instead of using glArrayElement and the vertex array. Overall, the program compiles successfully and runs as outlined by the assignment and fell I deserve the full 20 points.