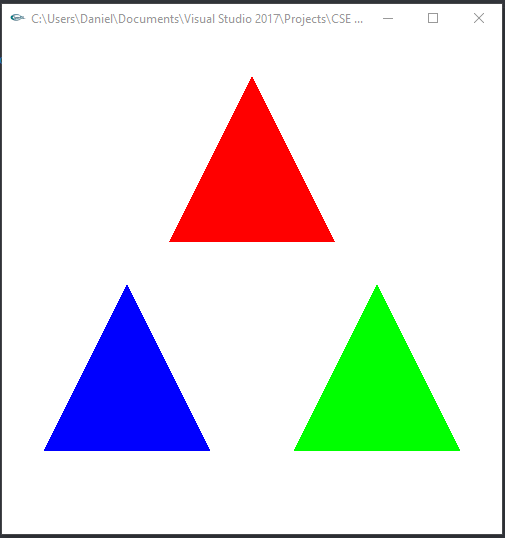
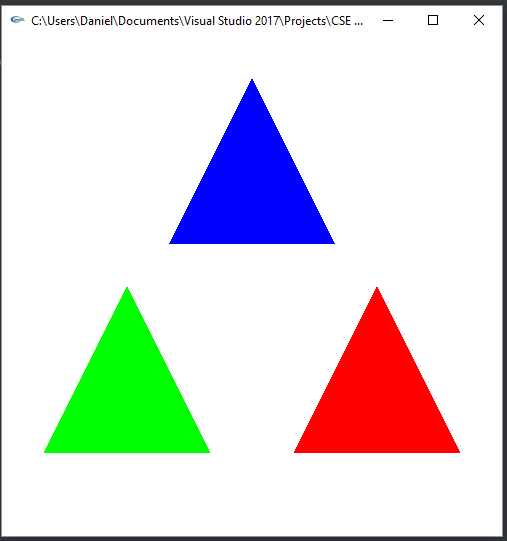
Daniel Meyer

CSE 520-01

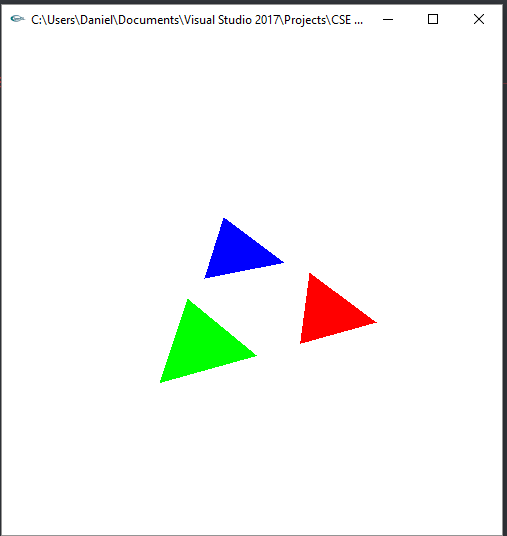
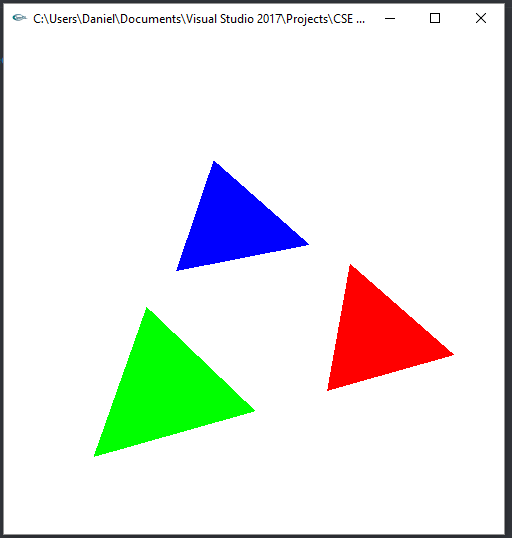
Lab 3

Color Shader

**Lab 3 Report**

*3 Triangles in default position Color rotated using ‘t’*



*Triangles rotated on X, Y, and Z Triangles scaled down*

**Lab3.cpp**

/\*

Lab3.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>

using namespace std;

#define PI 3.14159265359

/\*

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 cLoc;

int rLoc;

int sLoc;

float rx = 0.0;

float ry = 0.0;

float rz = 0.0;

float scale = 1.0;

int colorSelect = 0;

float color[] = { 1.0, 0.0, 0.0, 1.0 };

int readShaderSource(char \*fileName, GLchar \*\*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 = (GLchar \*)malloc(shaderSize);

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;

}

// public

int installShaders(const GLchar \*vertex, const GLchar \*fragment)

{

GLint vertCompiled, fragCompiled; // status values

GLint linked;

printf("------------\n");

printf("%s", vertex);

printf("\n--------------\n");

printf("------------\n");

printf("%s", fragment);

printf("\n--------------\n");

// 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);

}

readShaderSource((char \*) "Lab3.vert", &VertexShaderSource);

readShaderSource((char \*) "Lab3.frag", &FragmentShaderSource);

loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);

cLoc = glGetAttribLocation(programObject, "vColor");

rLoc = glGetAttribLocation(programObject, "rotate");

sLoc = glGetAttribLocation(programObject, "VertexScale");

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;

case 't':

if (colorSelect == 0)

{

color[0] = 1.0;

color[1] = 0.0;

color[2] = 0.0;

color[3] = 1.0;

colorSelect++;

break;

}

else if (colorSelect == 1)

{

color[0] = 0.0;

color[1] = 1.0;

color[2] = 0.0;

color[3] = 1.0;

colorSelect++;

break;

}

else if (colorSelect == 2)

{

color[0] = 0.0;

color[1] = 0.0;

color[2] = 1.0;

color[3] = 1.0;

colorSelect++;

break;

}

else

{

colorSelect = 0;

break;

}

case 'e':

scale += 0.1;

break;

case 'c':

scale -= 0.1;

break;

case 'x':

rx -= 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

case 'X':

rx += 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

case 'y':

ry -= 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

case 'Y':

ry += 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

case 'z':

rz -= 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

case 'Z':

rz += 2.0 \* (PI / 180); //need to convert degree to radians for GLSL

break;

}

glutPostRedisplay();

}

void display(void)

{

GLfloat vec[4];

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glClearColor(1.0, 1.0, 1.0, 0.0); //get white background color

glColor3f(1, 0, 0); //red, this will have no effect if shader is loaded

glVertexAttrib3f(rLoc, rx, ry, rz);

glVertexAttrib1f(sLoc, scale);

glBegin(GL\_TRIANGLES);

glVertexAttrib4f(cLoc, color[0], color[1], color[2], color[3]);

glVertex3f(-1.0, 0.5, 0.0);

glVertex3f(1.0, 0.5, 0.0);

glVertex3f(0.0, 2.5, 0.0);

glVertexAttrib4f(cLoc, color[1], color[2], color[0], color[3]);

glVertex3f(-2.5, -2.0, 0.0);

glVertex3f(-0.5, -2.0, 0.0);

glVertex3f(-1.5, 0.0, 0.0);

glVertexAttrib4f(cLoc, color[2], color[0], color[1], color[3]);

glVertex3f(0.5, -2.0, 0.0);

glVertex3f(2.5, -2.0, 0.0);

glVertex3f(1.5, 0.0, 0.0);

glEnd();

glutSwapBuffers();

glFlush();

}

int main(int argc, char \*argv[])

{

int success = 0;

glutInit(&argc, argv);

glutInitWindowPosition(0, 0);

glutInitWindowSize(500, 500);

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;

}

**Lab3.frag**

/\*

Lab3.frag

\*/

varying vec4 fColor;

void main(void)

{

//make a color with alpha of 1.0

//gl\_FragColor = vec4(color, 1.0);

gl\_FragColor = fColor;

//gl\_FragColor = vec4(1, 0, 0, 1);

}

**Lab3.vert**

//Lab3.vert

attribute vec3 rotate;

attribute float VertexScale;

attribute vec4 vColor;

varying vec4 fColor;

void main(void)

{

vec4 v4;

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

mat4 mScale = mat4 (VertexScale, 0, 0, 0,

0, VertexScale, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1);

v4 = mScale \* mRotateZ \* mRotateY \* mRotateX \* v4;

fColor = vColor;

gl\_Position = gl\_ProjectionMatrix \* gl\_ModelViewMatrix \* v4;

}

**Summary:**

For this assignment we were tasked with was to create aa shader program that displayed 3 triangles that were red, blue, and green. Then we were tasked with adding the functionality to rotate the color of each triangle to the next one using *‘t’* as well as expand and collapse the triangles using *‘e’* and *‘c*’ respectively. Finally, functionality for rotating the triangles using *x/X, y/Y,* and *z/Z* for each axis. I performed this task and the program compiles and runs correctly, as such I believe I have earned the full, 20-point credit for the assignment.