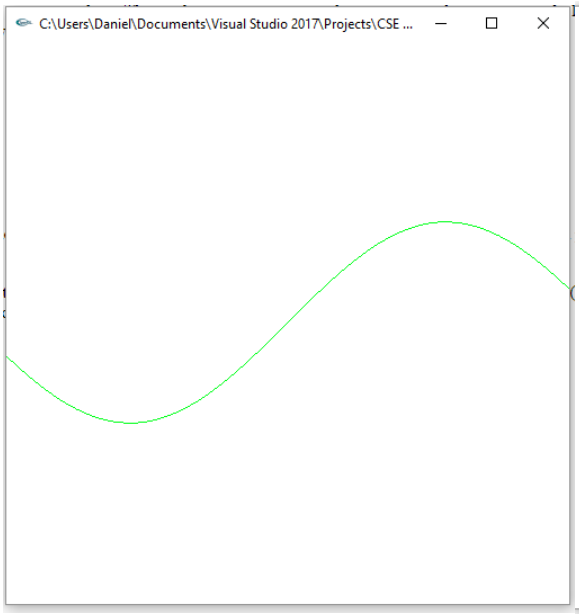
Daniel Meyer

CSE 520-01

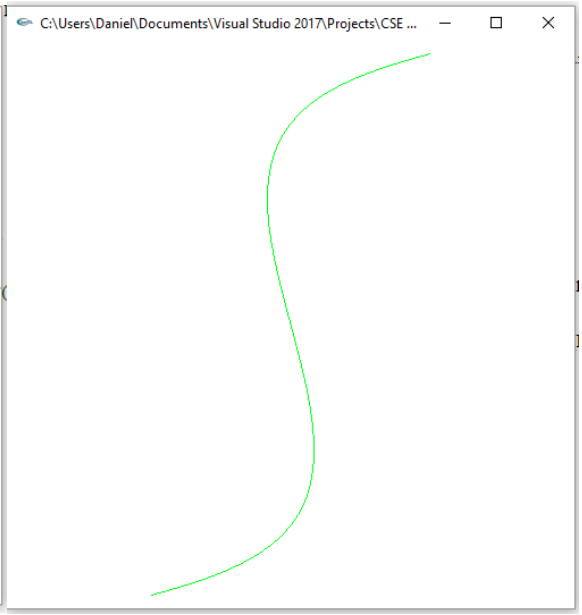
Lab 2

Introduction to Shader Programming

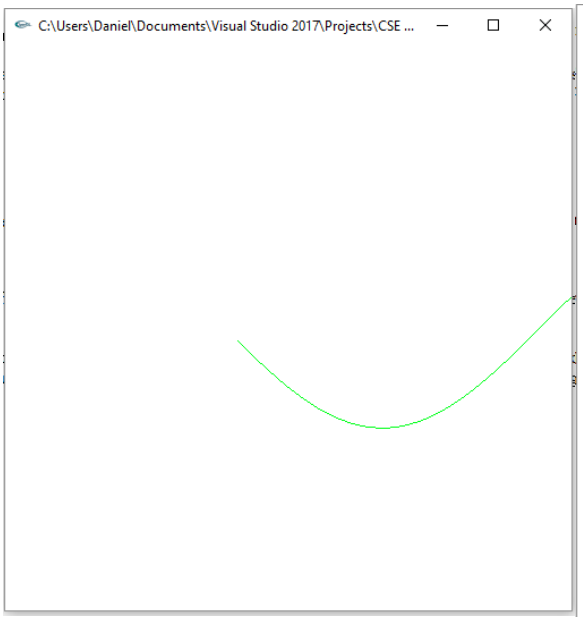
Lab 2 Report



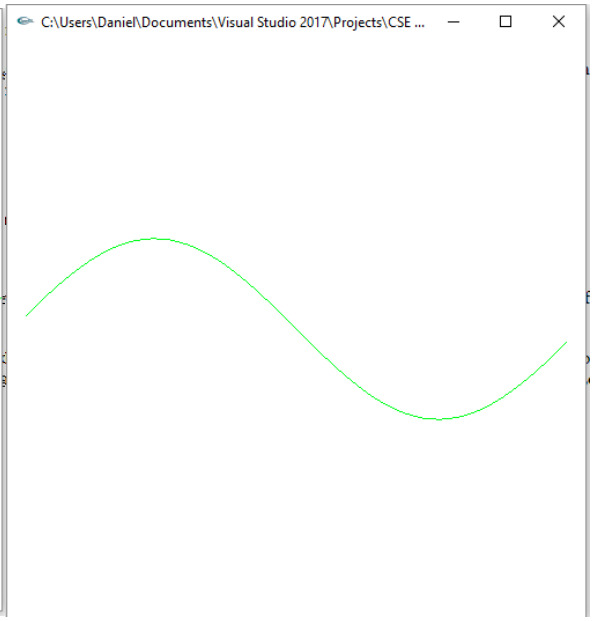
Vertex shader with sin function



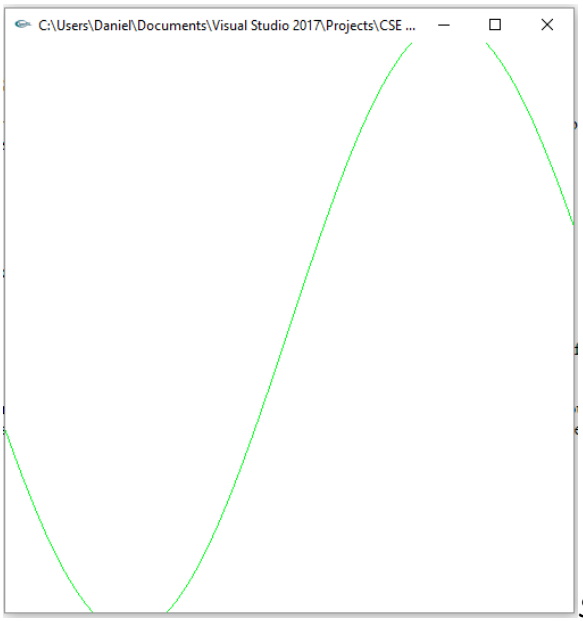
Translate the curve by (1,1)



Translate the curve by (1,1)



Reflect the curve about the y-z plane



Scale in the y direction by a factor of 3

Lab2.cpp

```
/*
  lab2.cpp
  Sample program showing how to write GL shader programs.
  Shader sources are in files "lab2.vert" and "lab2.frag".
  @Author: D.G. Meyer, 2019
*/
#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/glut.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 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 + 1);

	// 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", 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);
	}
	printf("version=%s\n", version);
	readShaderSource((char *) "lab2.vert", &VertexShaderSource);
	readShaderSource((char *) "lab2.frag", &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];

	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
	glPointSize(4);
	glBegin(GL_LINE_STRIP);
	for (float x = -3.0; x <= 3.0; x += 0.1)
		glVertex3f(x, 0, 1);
	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;
}
```

Lab2.vert

```
//lab2.vert
//a minimal vertex shader
void main(void)
{
    vec4 v4;
    v4 = gl_Vertex;
    v4.y = sin(v4.x); //apply sin to y-axis

    mat4 mRotateZ = mat4 ( 0.5, 0.867, 0, 0,//1st col
                        -0.867, 0.5, 0, 0,  //2nd col
                        0, 0, 1, 0,         //3rd col
                        0, 0, 0, 1  );      //4th col
    //v4 = mRotateZ * v4; //90degree about z-axis

    mat4 mTranslate = mat4 ( 1, 0, 0, 0,
                             0, 1, 0, 0,
                             0, 0, 1, 0,
                             2.5, 0, 0, 1);

    //v4 = mTranslate * v4; //translate by (1,1)

    mat4 mReflect = mat4 ( 1, 0, 0, 0,      //1st col
                        0, -1, 0, 0,        //2nd col
                        0, 0, -1, 0,        //3rd col
                        0, 0, 0, 1  );      //4th col

    //v4 = mReflect * v4; //180 X-Roll or reflect about the y-z place

    mat4 mScale = mat4 (1, 0, 0, 0,
                        0, 3, 0, 0,
                        0, 0, 1, 0,
                        0, 0, 0, 1);
```

```
    v4 = mScale * v4; //scale in y by 3

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

Lab2.frag

```
//lab2.frag
//a minimal fragment shader

void main(void)
{
  //gl_FragColor = vec4( 1, 0, 0, 1); //red color

    vec4 c1 = vec4 ( 1, 0.1, 0.1, 1 );
  vec4 c2 = vec4 ( 0, 1, 0.1, 1 );
//  float r = 0.5;
  float r = 1;
  vec4 color = mix ( c1, c2, r );
  //gl_FragColor = vec4( 0, 0, 0, 1); // color
  gl_FragColor = color;    // color
}
```

Summary:

This assignment was the first using GLSL fragment and vertex shaders and was relatively simple. I initially had some difficulty setting up the Linker in Visual Studio to recognize GLEW, but after fixing the Linker I was able to get the program to run and compile correctly. As per the above screenshots I modified the y vertex value by applying the sin function followed by performing a 90-degree rotation about the z-axis, translating the curve by (1,1), reflecting the curve about the y-z plane, and finally scaling it in the y direction by a factor of 3. I also changed the program from its default color to green using the fragment shader. Something I did note was that there wasn't any built-in commands for Affine Transformations and instead had to perform the math on my own and provide the matrices myself and then multiply those with the vertex. The program compiled and ran without errors according to the assignment and as such believe I have earned the full 20 points.