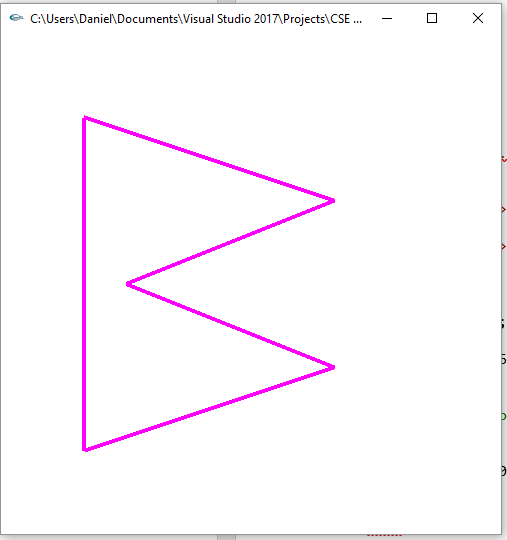
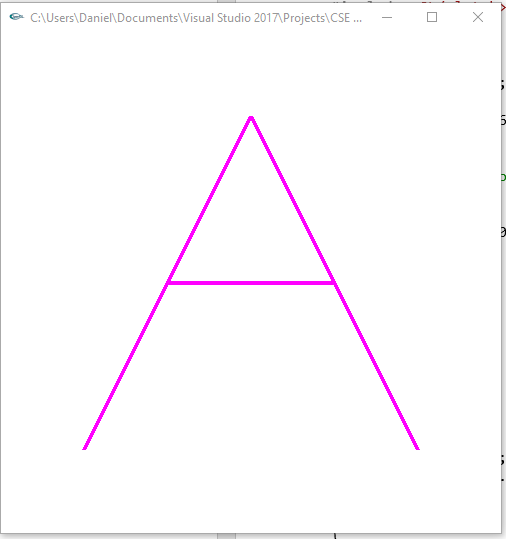
Daniel Meyer

CSE 520-01

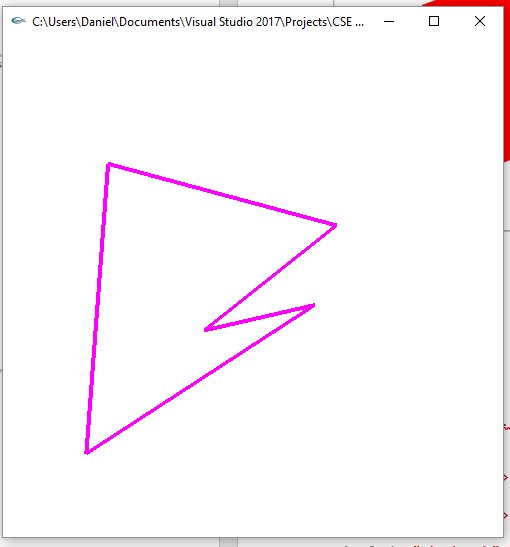
Homework 1

**Homework 1**

**Part 1 (success):**



*Figure “A” Figure “B”*

*Mid morphing between A and B*

**HW1.cpp**

/\*

main.cpp

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/types.h>

#include <time.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

class Point2 {

public:

float x;

float y;

Point2()

{

x = y = 0;

}

Point2(float x0, float y0)

{

x = x0; y = y0;

}

Point2(const Point2 &p)

{

x = p.x;

y = p.y;

}

};

/\*

Global handles for the currently active program object, with its two shader objects

\*/

static GLint win = 0;

Shader shader;

int cLoc; //color

int rLoc; //rotation

int sLoc; //scale

int tLoc; //time

int startLoc;

int endLoc;

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, 1.0, 1.0 };

float anim = 1;

Point2 A[10], B[10];

int init(void)

{

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 \*) "Template.vert", &VertexShaderSource);

//shader.readShaderFile((char \*) "Template.frag", &FragmentShaderSource);

shader.readShaderFile((char \*) "FigAB.vert", &VertexShaderSource);

shader.readShaderFile((char \*) "FigAB.frag", &FragmentShaderSource);

//shader.readShaderFile((char \*) "Square.vert", &VertexShaderSource);

//shader.readShaderFile((char \*) "Square.frag", &FragmentShaderSource);

vs = new string(VertexShaderSource);

fs = new string(FragmentShaderSource);

loadstatus = shader.createShader(vs, fs);

delete fs;

delete vs;

delete VertexShaderSource;

delete FragmentShaderSource;

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

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

tLoc = glGetUniformLocation(shader.programObject, "time1");

startLoc = glGetAttribLocation(shader.programObject, "start");

endLoc = glGetAttribLocation(shader.programObject, "end");

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)

{

float t = glutGet(GLUT\_ELAPSED\_TIME);

if (anim == 0)

{

while (t > 2000) t -= 2000;

}

else if (anim == 1)

{

t = 0;

}

else

{

t = 2000;

}

glUniform1f(tLoc, t);

glutPostRedisplay();

}

void animate()

{

if (rz < 360)

{

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

}

else

{

rz = 0.0;

}

}

static void Key(unsigned char key, int x, int y)

{

switch (key) {

case 27:

CleanUp();

exit(0);

break;

case 'a':

animate();

break;

case 't':

if (anim == 0)

{

anim = 1;

}

else if(anim == 1)

{

anim = 2;

}

else

{

anim = 0;

}

break;

}

glutPostRedisplay();

}

void makeFigures(Point2 A[], Point2 B[])

{

A[0].x = -2; A[0].y = -2;

A[1].x = -1; A[1].y = 0;

A[2].x = -1; A[2].y = 0;

A[3].x = 1; A[3].y = 0;

A[4].x = 1; A[4].y = 0;

A[5].x = 2; A[5].y = -2;

A[6].x = 2; A[6].y = -2;

A[7].x = 0; A[7].y = 2;

A[8].x = 0; A[8].y = 2;

A[9].x = -2; A[9].y = -2;

B[0].x = -2; B[0].y = -2;

B[1].x = -2; B[1].y = 2;

B[2].x = -2; B[2].y = 2;

B[3].x = 1; B[3].y = 1;

B[4].x = 1; B[4].y = 1;

B[5].x = -1.5; B[5].y = 0;

B[6].x = -1.5; B[6].y = 0;

B[7].x = 1; B[7].y = -1;

B[8].x = 1; B[8].y = -1;

B[9].x = -2; B[9].y = -2;

}

void display(void)

{

makeFigures(A, B);

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

/\*

glLineWidth(4);

glBegin(GL\_LINE\_STRIP); //need GL\_POINTS; "GL\_POINT" doesn't work

for (float x = -3.0; x <= 3.0; x += 0.1)

glVertex3f(x, 0, 0);

glEnd();

\*/

/\*

glVertexAttrib3f(rLoc, rx, ry, rz);

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

glBegin(GL\_QUADS);

glVertex3f(-2.2, -2.2, 0);

glVertex3f(2.2, -2.2, 0);

glVertex3f(2.2, 2.2, 0);

glVertex3f(-2.2, 2.2, 0);

glEnd();

\*/

glLineWidth(4);

glBegin(GL\_LINES);

for(int i = 0; i < 10; i++)

{

glVertex3f(A[i].x, A[i].y, 0);

glVertexAttrib3f(startLoc, A[i].x, A[i].y, 0);

glVertexAttrib3f(endLoc, B[i].x, B[i].y, 0);

i++;

glVertex3f(A[i].x, A[i].y, 0);

//glVertexAttrib3f(startLoc, A[i].x, A[i].y, 0);

//glVertexAttrib3f(endLoc, B[i].x, B[i].y, 0);

}

/\*

//'A'

glVertex3f(-2, -2, 0);

glVertex3f(-1, 0, 0);

glVertex3f(-1, 0, 0);

glVertex3f(1, 0, 0);

glVertex3f(1, 0, 0);

glVertex3f(2, -2, 0);

glVertex3f(2, -2, 0);

glVertex3f(0, 2, 0);

glVertex3f(0, 2, 0);

glVertex3f(-2, -2, 0);

\*/

/\*

//'B'

glVertex3f(-2, -2, 0);

glVertex3f(-2, 2, 0);

glVertex3f(-2, 2, 0);

glVertex3f(1, 1, 0);

glVertex3f(1, 1, 0);

glVertex3f(-1.5, 0, 0);

glVertex3f(-1.5, 0, 0);

glVertex3f(1, -1, 0);

glVertex3f(1, -1, 0);

glVertex3f(-2, -2, 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();

if (success)

{

printf("Shaders compiled successfully!\n");

glutMainLoop();

}

else

{

printf("infoLog:: %s\n", shader.infoLog);

}

return 0;

}

**FigAB.vert**

/\*

FigAB.vert

\*/

uniform float time1;

attribute vec3 start;

attribute vec3 end;

void main(void)

{

float s = 2000.0;

float t;

t = time1 / s;

vec4 v4;

v4 = gl\_Vertex;

v4.xy = mix(start.xy, end.xy, t);

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

}

**FigAB.frag**

//FigAB.frag

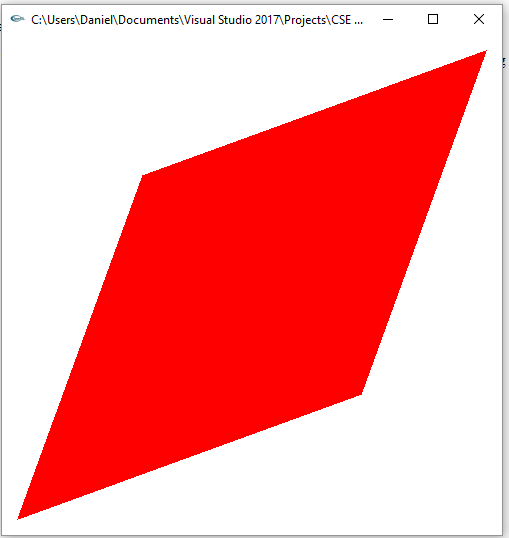
void main(void)

{

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

}

**Part 2 (success):**

*Square rotating about the z-axis by pressing ‘a’*

**HW1.cpp**

/\*

main.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;

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 init(void)

{

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 \*) "Template.vert", &VertexShaderSource);

//shader.readShaderFile((char \*) "Template.frag", &FragmentShaderSource);

//shader.readShaderFile((char \*) "FigAB.vert", &VertexShaderSource);

//shader.readShaderFile((char \*) "FigAB.frag", &FragmentShaderSource);

shader.readShaderFile((char \*) "Square.vert", &VertexShaderSource);

shader.readShaderFile((char \*) "Square.frag", &FragmentShaderSource);

vs = new string(VertexShaderSource);

fs = new string(FragmentShaderSource);

loadstatus = shader.createShader(vs, fs);

delete fs;

delete vs;

delete VertexShaderSource;

delete FragmentShaderSource;

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

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

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();

}

void animate()

{

if (rz < 360)

{

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

}

else

{

rz = 0.0;

}

}

static void Key(unsigned char key, int x, int y)

{

switch (key) {

case 27:

CleanUp();

exit(0);

break;

case 'a':

animate();

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

/\*

glLineWidth(4);

glBegin(GL\_LINE\_STRIP); //need GL\_POINTS; "GL\_POINT" doesn't work

for (float x = -3.0; x <= 3.0; x += 0.1)

glVertex3f(x, 0, 0);

glEnd();

\*/

glVertexAttrib3f(rLoc, rx, ry, rz);

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

glBegin(GL\_QUADS);

glVertex3f(-2.2, -2.2, 0);

glVertex3f(2.2, -2.2, 0);

glVertex3f(2.2, 2.2, 0);

glVertex3f(-2.2, 2.2, 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();

if (success)

glutMainLoop();

else

{

printf("infoLog:: %s\n", shader.infoLog);

}

return 0;

}

**Square.frag**

/\*

Square.frag

\*/

varying vec4 fColor;

void main(void)

{

//make a color with alpha of 1.0

gl\_FragColor = fColor;

}

**Square.vert**

attribute vec3 rotate;

attribute vec4 vColor;

varying vec4 fColor;

void main(void)

{

vec4 v4;

v4 = gl\_Vertex;

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

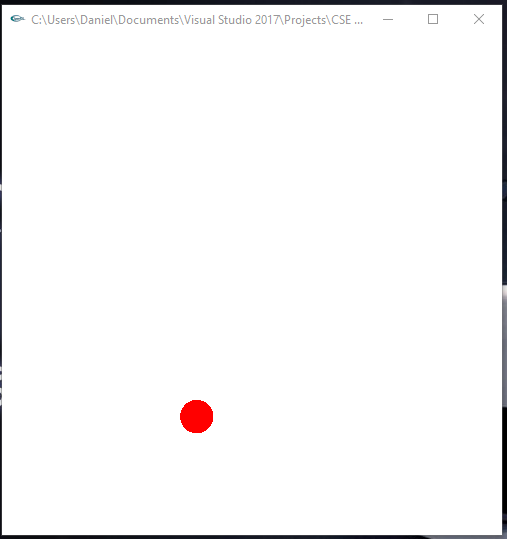
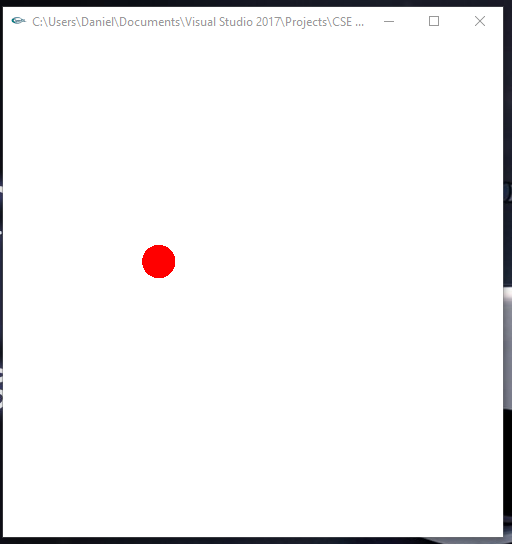
v4 = mRotateZ \* v4;

fColor = vColor;

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

}

**Part 3 (Partial Success):**



*Ball Rising Ball Falling*

**HW1.cpp**

/\*

main.cpp

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/types.h>

#include <time.h>

#include <math.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

#define X .525731112119133606

#define Z .850650808352039932

/\*

Global handles for the currently active program object, with its two shader objects

\*/

static GLint win = 0;

Shader shader;

int pLoc; //position

int tLoc; //time

int vLoc; //velocity

float iVelocityY = 7.0;

float iVelocityX = 1.0;

float iVelocityZ = 0.0;

float iPositionX = -2.0;

float iPositionY = -2.0;

float iPositionZ = 0.0;

int init(void)

{

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 \*) "Ball.vert", &VertexShaderSource);

shader.readShaderFile((char \*) "Ball.frag", &FragmentShaderSource);

vs = new string(VertexShaderSource);

fs = new string(FragmentShaderSource);

loadstatus = shader.createShader(vs, fs);

vLoc = glGetUniformLocation(shader.programObject, "iVel");

pLoc = glGetUniformLocation(shader.programObject, "iPos");

tLoc = glGetUniformLocation(shader.programObject, "time1");

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)

{

float t = glutGet(GLUT\_ELAPSED\_TIME);

while (t > 5000) t -= 5000;

glUniform1f(tLoc, t);

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(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

glUniform3f(pLoc, iPositionX, iPositionY, iPositionZ);

glUniform3f(vLoc, iVelocityX, iVelocityY, iVelocityZ);

glutSolidSphere(0.2, 50, 50);

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();

if (success)

{

printf("Shaders compiled successfully!\n");

glutMainLoop();

}

else

{

printf("infoLog:: %s\n", shader.infoLog);

}

return 0;

}

**Ball.vert**

//Ball.vert

uniform vec3 iVel;

uniform vec3 iPos;

uniform float time1;

varying float prevY;

void main(void)

{

float s = 1000.0;

float g = -10.0;

float t;

vec3 minimum = vec3(0.0, -2.0, 0.0);

float cor = 0.752; //coefficient of resitution for a hand ball

t = time1 / s;

float t1; //time at jump

float t2; //time at peak

float t3; //time when at bottom

float dt; //change in time

//float prevY;

float dy = 0.0; //change in y

vec4 objPos;

objPos = gl\_Vertex;

objPos.xyz = objPos.xyz + iPos.xyz;

t1 = time1 / s; //initial time

objPos.x = objPos.x + iVel.x \* t;

objPos.y = objPos.y + iVel.y \* t + g / (2.0) \* t \* t;

objPos.z = objPos.z + iVel.z \* t;

dy = objPos.y - iPos.y;

if(objPos.y < dy) //ball is now falling, record peak time

{

t2 = t;

dt = t2 - t1;

}

t3 = t; //record time as ball falls

if((t3 - dt) == dt) //if change in falling time = change in rising time, ball is at initial position / floor, bounce

{

objPos.y = reflect(-objPos.y, normalize(iPos.y)) \* cor;

}

//dy = objPos.y - prevY;

//prevY = objPos.y;

/\*

if(objPos.y < iPos.y)

{

//objPos.y = reflect(-objPos.y, normalize(iPos.y)) \* cor;

}

\*/

//objPos.y = reflect(-objPos.y, normalize(iPos.y)) \* cor;

//objPos.x = reflect(objPos.x, normalize(iPos.x));

gl\_Position = gl\_ModelViewProjectionMatrix \* objPos;

prevY = objPos.y;

}

**Ball.frag**

//Ball.frag

//a minimal fragment shader

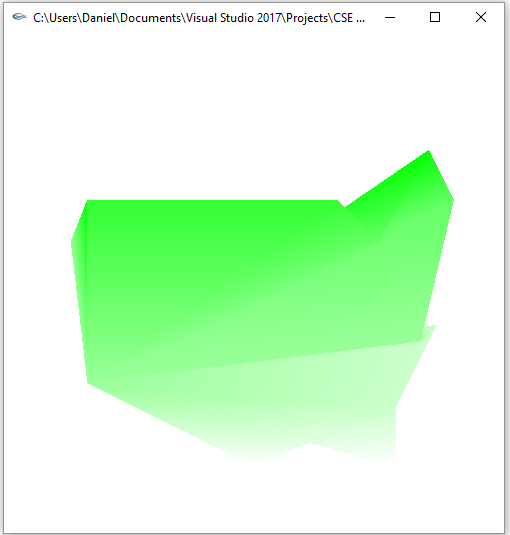
void main(void)

{

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

}

**Part 4 (success):**

*Green = coldest, white = hottest*

**HW1.cpp**

/\*

main.cpp

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <sys/types.h>

#include <time.h>

#include <math.h>

#define GLEW\_STATIC 1

#include <GL/glew.h>

#include <GL/glu.h>

#include <GL/glut.h>

#include "Shader.h"

using namespace std;

/\*

Global handles for the currently active program object, with its two shader objects

\*/

static GLint win = 0;

Shader shader;

int init(void)

{

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 \*) "US.vert", &VertexShaderSource);

shader.readShaderFile((char \*) "US.frag", &FragmentShaderSource);

vs = new string(VertexShaderSource);

fs = new string(FragmentShaderSource);

loadstatus = shader.createShader(vs, fs);

//Set up initial uniform values

GLchar names[][20] = { "CoolestColor", "HottestColor", "CoolestTemp",

"TempRange" };

GLint loc[10];

for (int i = 0; i < 4; ++i) {

loc[i] = glGetUniformLocation(shader.programObject, names[i]);

if (loc[i] == -1)

printf("No such uniform named %s\n", names[i]);

}

glUniform3f(loc[0], 0.0, 1.0, 0.0);

glUniform3f(loc[1], 1.0, 1.0, 1.0);

glUniform1f(loc[2], 0.0);

glUniform1f(loc[3], 1.0);

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)

{

float t = glutGet(GLUT\_ELAPSED\_TIME);

while (t > 3000) t -= 3000;

glUniform1f(tLoc, t);

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(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

loc = glGetAttribLocation(shader.programObject, "VertexTemp");

glBegin(GL\_POLYGON);

if (loc == -1)

printf("No such attribute named %s\n", "VertexTemp");

//0.0 = coldest, 1.0 = hottest

glVertexAttrib1f(loc, 0.6);

glVertex3f(-2.0, -1.2, 0);

glVertexAttrib1f(loc, 0.8);

glVertex3f(-1.0, -1.5, 0);

glVertexAttrib1f(loc, 1.0);

glVertex3f(0.0, -2.2, 0);

glVertexAttrib1f(loc, 0.8);

glVertex3f(1.2, -1.7, 0);

glVertexAttrib1f(loc, 1.0);

glVertex3f(1.7, -2.2, 0);

glVertexAttrib1f(loc, 0.8);

glVertex3f(1.7, -1.5, 0);

glVertexAttrib1f(loc, 0.8);

glVertex3f(2.2, -0.5, 0);

glVertexAttrib1f(loc, 0.6);

glVertex3f(2.0, -0.7, 0);

glVertexAttrib1f(loc, 0.4);

glVertex3f(2.4, 1.0, 0);

glVertexAttrib1f(loc, 0.0);

glVertex3f(2.1, 1.6, 0);

glVertexAttrib1f(loc, 0.4);

glVertex3f(1.5, 0.5, 0);

glVertexAttrib1f(loc, 0.2);

glVertex3f(1.0, 1.0, 0);

glVertexAttrib1f(loc, 0.2);

glVertex3f(-2.0, 1.0, 0);

glVertexAttrib1f(loc, 0.0);

glVertex3f(-2.0, 1.0, 0);

glVertexAttrib1f(loc, 0.6);

glVertex3f(-2.2, 0.5, 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();

if (success)

{

printf("Shaders compiled successfully!\n");

glutMainLoop();

}

else

{

printf("infoLog:: %s\n", shader.infoLog);

}

return 0;

}

**US.vert**

//US.vert

//uniform qualified variables are changed at most once per primitives

uniform float CoolestTemp;

uniform float TempRange;

//attribute qualified variables are typically changed per vertex

attribute float VertexTemp;

//varying qualified variables communicate from the vertex shader to

//the fragment shader

varying float Temperature;

void main(void)

{

//compute a temperature to be interpolated per fragment

// in the range [0.0, 1.0]

Temperature = ( VertexTemp - CoolestTemp ) / TempRange;

//Temperature = ( 1.0 - CoolestTemp ) / TempRange;

gl\_Position = ftransform();

/\*

Same as:

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

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

\*/

}

**US.frag**

/\*

US.frag:

uniform qualified variables are changed at most once per primitive

by the application, and vec3 declares a vector of three

floating-point numbers.

\*/

uniform vec3 CoolestColor;

uniform vec3 HottestColor;

varying float Temperature;

void main(void)

{

//get a color between coolest and hottest colors, using the

// mix() built-in function

vec3 color = mix ( CoolestColor, HottestColor, Temperature );

//make a color with alpha of 1.0

gl\_FragColor = vec4(color, 1.0);

}

**Summary:**

For this assignment we were given tasks to create 4 different shaders: one that morphs ‘A’ to ‘B’ and back, one that animates the rotation of a square about the z-axis, one that animates a ball bouncing with given initial velocity and position, and finally a shader that shows the temperature distribution across a polygon of the US. Most of these compiled and ran as intended, however the bouncing ball shader proved to be a problem. I was able to get the ball to bounce with its initial velocity and position, however difficulty appeared when trying to get it to bounce again. I attempted recording previous y positions to determine the time to reach peak and then use that time to determine when the ball had reached the floor to begin another bounce. I attempted over a dozen permutations and was still unsuccessful with the code provided being my most successful attempt. Overall, I feel I deserve full credit for parts 1,2 and 4 and partial credit for part 3 of the assignment leaving a final score of 50 points.