Name: Rowan Birnbaum

Student ID: 100701879

Main.cpp:

```
/************************************************
 *
 *             Example Four
 *
 *  A basic OpenGL program that draws a
 *  triangle on the screen in perspective with
 *  simple control over the eye position.
 *  This program illustrates the construction of
 *  perspective and viewing transformations.
 *
 ************************************************/

#include <Windows.h>
#include <gl/glew.h>
#define GLFW_DLL
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "shaders.h"
#include <stdio.h>
#include "tiny_obj_loader.h"
#include <iostream>

GLuint program;                    // shader programs
GLuint objVAO;              // the data to be displayed
int triangles;
int cx, cy, cz;

float angle = 0.0;
double theta, phi;          // user's position  on a sphere centered on the object
double r;                          // radius of the sphere
GLuint ibuffer;

glm::mat4 projection;       // projection matrix
float eyex, eyey, eyez;     // eye position

/*
 *  The init procedure creates the OpenGL data structures
 *  that contain the triangle geometry, compiles our
 *  shader program and links the shader programs to
 *  the data.
 */

void init() {
       GLuint vbuffer;
       GLuint ibuffer;
       GLint vPosition;
       GLint vNormal;
       int vs;
int fs;
GLfloat* vertices;
GLfloat* normals;
GLushort* indices;
std::vector<tinyobj::shape_t> shapes;
std::vector<tinyobj::material_t> materials;
int nv;
int nn;
int ni;
int i;
float xmin, ymin, zmin;
float xmax, ymax, zmax;

glGenVertexArrays(1, &objVAO);
glBindVertexArray(objVAO);

/*  Load the obj file */

std::string err = tinyobj::LoadObj(shapes, materials, "vase.obj", 0);

if (!err.empty()) {
       std::cerr << err << std::endl;
       return;
}

/*  Retrieve the vertex coordinate data */

nv = shapes[0].mesh.positions.size();
vertices = new GLfloat[nv];
for (i = 0; i < nv; i++) {
       vertices[i] = shapes[0].mesh.positions[i];
}

/*
 *  Find the range of the x, y and z
 *  coordinates.
 */
xmin = ymin = zmin = 1000000.0;
xmax = ymax = zmax = -1000000.0;
for (i = 0; i < nv / 3; i++) {
       if (vertices[3 * i] < xmin)
              xmin = vertices[3 * i];
       if (vertices[3 * i] > xmax)
              xmax = vertices[3 * i];
       if (vertices[3 * i + 1] < ymin)
              ymin = vertices[3 * i + 1];
       if (vertices[3 * i + 1] > ymax)
              ymax = vertices[3 * i + 1];
       if (vertices[3 * i + 2] < zmin)
              zmin = vertices[3 * i + 2];
       if (vertices[3 * i + 2] > zmax)
              zmax = vertices[3 * i + 2];
}
/* compute center and print range */
cx = (xmin + xmax) / 2.0f;
cy = (ymin + ymax) / 2.0f;
cz = (zmin + zmax) / 2.0f;
printf("X range: %f %f\n", xmin, xmax);
```

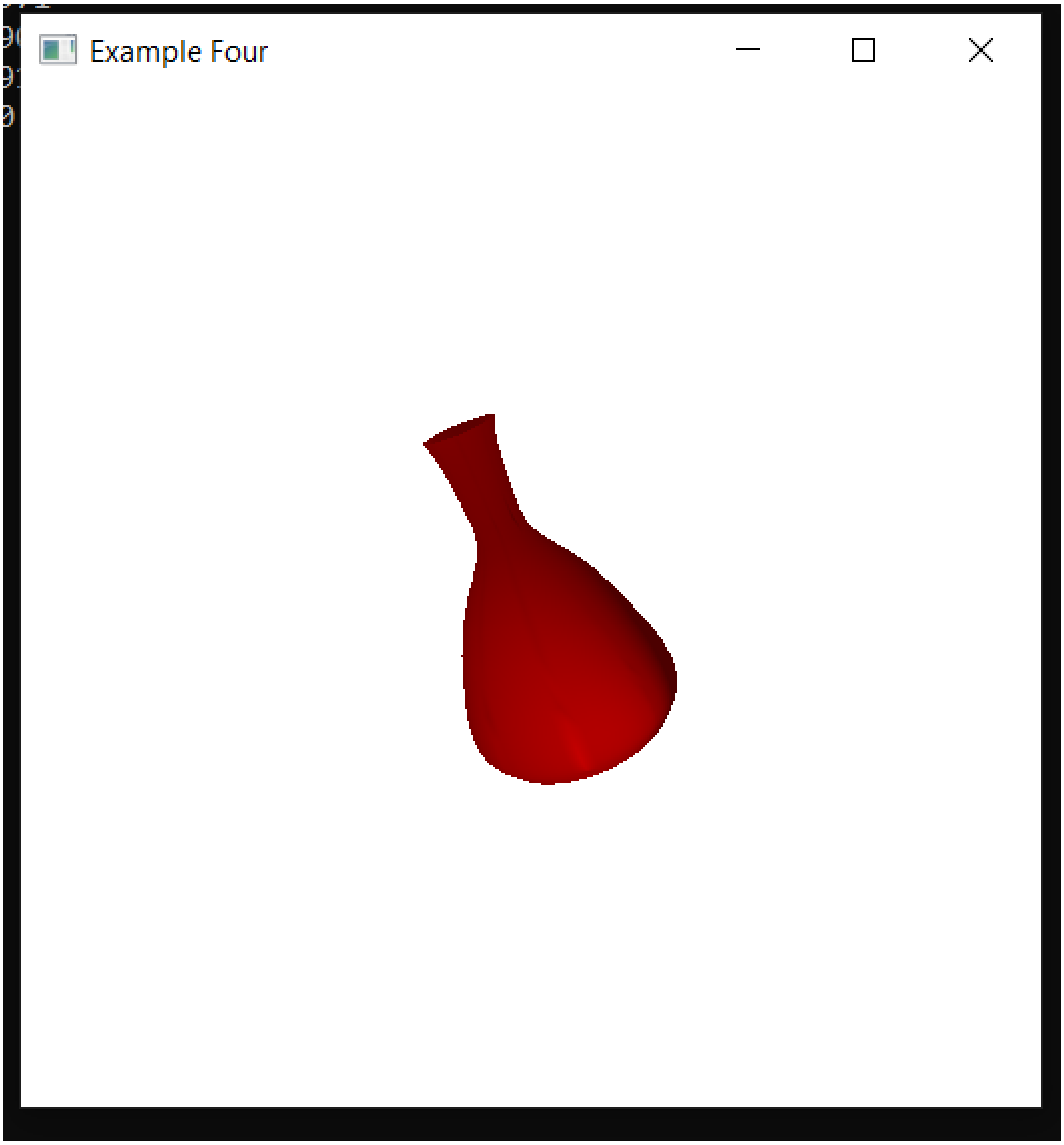
```
        printf("Y range: %f %f\n", ymin, ymax);
        printf("Z range: %f %f\n", zmin, zmax);
        printf("center: %f %f %f\n", cx, cy, cz);

        /*  Retrieve the vertex normals */

        nn = shapes[0].mesh.normals.size();
        normals = new GLfloat[nn];
        for (i = 0; i < nn; i++) {
                normals[i] = shapes[0].mesh.normals[i];
        }

        /*  Retrieve the triangle indices */

        ni = shapes[0].mesh.indices.size();
        triangles = ni / 3;
        indices = new GLushort[ni];
        for (i = 0; i < ni; i++) {
                indices[i] = shapes[0].mesh.indices[i];
        }

        /*
         *  load the vertex coordinate data
         */
        glGenBuffers(1, &vbuffer);
        glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
        glBufferData(GL_ARRAY_BUFFER, (nv + nn) * sizeof(GLfloat), NULL, GL_STATIC_DRAW);
        glBufferSubData(GL_ARRAY_BUFFER, 0, nv * sizeof(GLfloat), vertices);
        glBufferSubData(GL_ARRAY_BUFFER, nv * sizeof(GLfloat), nn * sizeof(GLfloat),
normals);

        /*
         *  load the vertex indexes
         */
        glGenBuffers(1, &ibuffer);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, ni * sizeof(GLushort), indices,
GL_STATIC_DRAW);

        /*
         *  compile and build the shader program
         */
        vs = buildShader(GL_VERTEX_SHADER, "lab2.vs");
        fs = buildShader(GL_FRAGMENT_SHADER, "lab2.fs");
        program = buildProgram(vs, fs, 0);

        /*
         *  link the vertex coordinates to the vPosition
         *  variable in the vertex program.  Do the same
         *  for the normal vectors.
         */
        glUseProgram(program);
        vPosition = glGetAttribLocation(program, "vPosition");
        glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(vPosition);
        vNormal = glGetAttribLocation(program, "vNormal");
        glVertexAttribPointer(vNormal, 3, GL_FLOAT, GL_FALSE, 0, (void*)sizeof(vertices));
        glEnableVertexAttribArray(vNormal);
```

```
}

void framebufferSizeCallback(GLFWwindow *window, int w, int h) {

       // Prevent a divide by zero, when window is too short
       // (you cant make a window of zero width).

       if (h == 0)
              h = 1;

       float ratio = 1.0f * w / h;

       glfwMakeContextCurrent(window);

       glViewport(0, 0, w, h);

       projection = glm::perspective(45.0f, ratio, 1.0f, 800.0f);

}

/*
 *  This procedure is called each time the screen needs
 *  to be redisplayed
 */
void display() {
       glm::mat4 view;
       glm::mat4 modelViewPerspective;
       int modelLoc;
       int normalLoc;
       view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
              glm::vec3(cx, cy, cz),
              glm::vec3(0.0f, 0.0f, 1.0f));
       glm::mat3 normal = glm::transpose(glm::inverse(glm::mat3(view)));
       modelViewPerspective = projection * view;
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glUseProgram(program);
       modelLoc = glGetUniformLocation(program, "model");
       glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(modelViewPerspective));
       normalLoc = glGetUniformLocation(program, "normalMat");
       glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
       glBindVertexArray(objVAO);
       glDrawElements(GL_TRIANGLES, 3 * triangles, GL_UNSIGNED_SHORT, NULL);
}


/*
 *  Called each time a key is pressed on
 *  the keyboard.
 */

static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
       if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
              glfwSetWindowShouldClose(window, GLFW_TRUE);

       if (key == GLFW_KEY_A && action == GLFW_PRESS)
              phi -= 0.1;
```

```
        if (key == GLFW_KEY_D && action == GLFW_PRESS)
                phi += 0.1;
        if (key == GLFW_KEY_W && action == GLFW_PRESS)
                theta += 0.1;
        if (key == GLFW_KEY_S && action == GLFW_PRESS)
                theta -= 0.1;

        eyex = (float)(r*sin(theta)*cos(phi));
        eyey = (float)(r*sin(theta)*sin(phi));
        eyez = (float)(r*cos(theta));

}

void error_callback(int error, const char* description)
{
        fprintf(stderr, "Error: %s\n", description);
}


int main(int argc, char **argv) {
        GLFWwindow *window;

        // start by setting error callback in case something goes wrong

        glfwSetErrorCallback(error_callback);

        // initialize glfw

        if (!glfwInit()) {
                fprintf(stderr, "can't initialize GLFW\n");
        }

        // create the window used by our application

        window = glfwCreateWindow(512, 512, "Example Four", NULL, NULL);

        if (!window)
        {
                glfwTerminate();
                exit(EXIT_FAILURE);
        }

        // establish framebuffer size change and input callbacks

        glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);

        glfwSetKeyCallback(window, key_callback);

        /*
         *  initialize glew
         */
        glfwMakeContextCurrent(window);
        GLenum error = glewInit();
        if(error != GLEW_OK) {
                printf("Error starting GLEW: %s\n",glewGetErrorString(error));
                exit(0);
        }
```

```
	glEnable(GL_DEPTH_TEST);
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glViewport(0, 0, 512, 512);

	projection = glm::perspective(45.0f, 1.0f, 1.0f, 800.0f);

	init();

	eyex = 0.0;
	eyez = 500.0;
	eyey = 0.0;

	theta = 1.5;
	phi = 1.5;
	r = 500.0;

	glfwSwapInterval(1);

	// GLFW main loop, display model, swapbuffer and check for input

	while (!glfwWindowShouldClose(window)) {
		display();
		glfwSwapBuffers(window);
		glfwPollEvents();
	}

	glfwTerminate();

}
```

Output:


Example Four