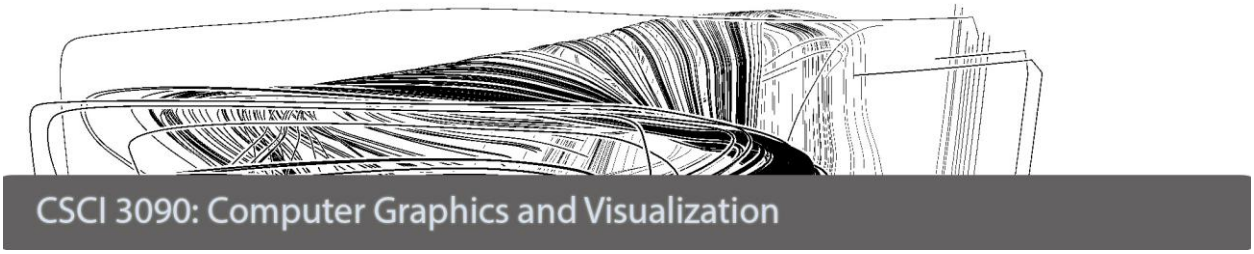


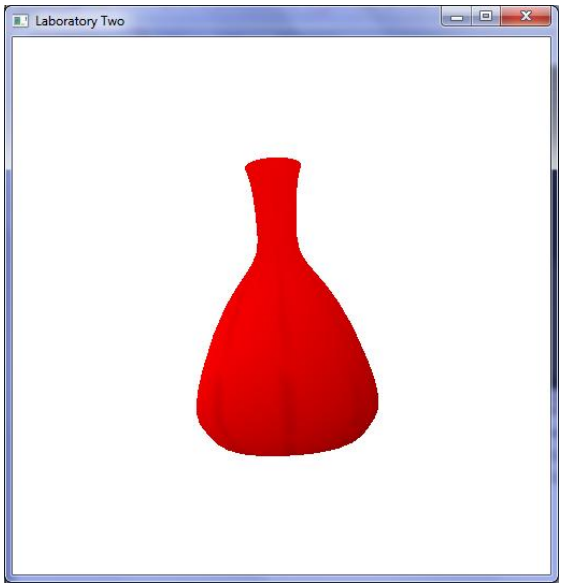
Laboratory Two: OBJ File Loader

Introduction

Entering vertex coordinates and normal vectors by hand quickly becomes tedious and time consuming. For any reasonable object this is not a practical approach. Instead a geometrical modeller is used to construct the geometrical information. This is a program that supports the interactive editing of geometrical and quite often material properties of objects. A well-known open source geometrical modeller and animation program is Blender. These programs use various types of file formats for storing geometrical information and we will explore one of these file formats in this laboratory. The file format that we will use is called OBJ and it is an ASCII file format for geometrical information. This is a fairly sophisticated file format that can store far more information than we can use in OpenGL programs, but it is very widely used so it is worth investigating. You will find many free OBJ files on the web. We will use a simple library for loading OBJ files, called tiny OBJ loader. This library doesn't support all of the OBJ features, but it is very easy to add to our programs and is quite easy to use. There are only two source code files that you need to add to your project.

This laboratory builds on laboratory one, so you should start with the source code from that example. There is a Lab_2.zip file on Blackboard that you must download for this laboratory. This zip file contains tiny_obj_loader.h and tiny_obj_loader.cc that you must add to the project for this laboratory. The lab_2.fs, lab_2.vs and vase.obj file should be placed in the same folder as that of the other source files. Replace the main.cpp file from laboratory one with the new main.cpp file.

You will need to make several modifications to the program, which are described below. When finished you will get the following output:



Modifying the init() procedure

You need to write code to do the following in the init() procedure.

1. Load the OBJ file: The heart of the tiny OBJ loader library is the tinyobj::loadObj procedure. This procedure has four parameters. The first parameter is the list of shapes in the model, and the second parameter is the list of materials. The third parameter is the OBJ file name, and the fourth parameter is the path to the folder where the material files are stored. I have chosen a particularly simple OBJ file that has only one shape and no materials to make this laboratory easier. The first parameter of this procedure needs to be a vector of tiny obj shape structures, where each shape has name and a mesh. Each mesh has positions (the vector coordinates), normal vectors and triangle indices. There is other information stored in the mesh, but we don't need them for this laboratory. The data structures returned by this procedure call are documented in the tiny_obj_loader.h include file.
2. Retrieve the vertex coordinates data from the mesh
3. Compute the centre of the object's bounding box which is then used to construct the viewing transformation in displayFunc()
4. Lastly, retrieve the vertex normals and indices data from the mesh

Other modifications

There are a few other changes that we need to make to the program starting with the include files and declarations at the start of the program. You should modify the start of the program to include the following two include files and the following global declarations:

```
#include "tiny_obj_loader.h"
#include <iostream>

GLuint program;                    // shader programs
GLuint objVAO;                     // the data to be displayed
int triangles;                     // number of triangles
int window;
```

Since this model is much larger than the one in the first laboratory we need to modify our near and far clipping planes in the following way:

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

This is done in the changeSize() procedure. The displayFunc procedure is basically the same as before with the major changes occurring in the call to glm::lookat and the call to glDrawElements. In addition the model matrix has been removed along with the idle procedure that computed the rotation angle. The code for the modified displayFunc procedure is:

```
void displayFunc() {
	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);

	glutSwapBuffers();

}
```

Since the model is much larger we need to change the initial values of eyex, eyey and eyez along with the parameters used in the navigation computations. The modified statements in the main() procedure are:

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

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

Laboratory Report

Submit your source code and a screenshot of your final result in the Blackboard dropbox.