**CS409/CS809 Tutorial 13: Displaying OBJ Models with Shaders**

# This lab will show you how to display models using *shaders*, which are short programs that run on the graphics card. You will have to do this if the rest of your program uses shaders (shader and non-shader graphics cannot be mixed). This demonstration only shows how to use the ObjLibrary to display the models. It does not explain how to display other things with shaders. I did not write either of the glm or gl3w libraries used below.

# Setting up an OpenGL Project with the Obj Library of Files

1. Start Microsoft Visual Studio 2015. Select File / New / Project…. Under the list on the left, choose Installed / Templates / Visual C++. Then choose a Win32 Console Application. Change the name to Tutorial13 or some suitable name and change the location to a hard drive located on the computer (typically C or D). You will need to remember this location for steps 3 and 5. Make sure the “Create folder for project” checkbox is not checked and that the “Add to source control” is not checked. Click OK.
	* If the Chose Location window keeps popping up again, you aren’t allowed to save files at the location you chose. Select a different location, such as the desktop.
	* Do not put your program on one of the external drives like H. The connection is slow, and your program may not compile correctly.
2. Click Next> or Application Settings (they go to the same place). Check the “Empty Project” box. Make sure the “Security development lifecycle check” is not checked. Click Finish.
3. Go to <http://www2.cs.uregina.ca/~anima/409/Terms/201810/Tutorials/index.html> and download the [Tutorial13.zip](http://www2.cs.uregina.ca/~anima/409/Terms/201810/Tutorials/Tutorial13.zip) and [ObjLibrary.zip](http://www2.cs.uregina.ca/~anima/409/Terms/201810/Tutorials/ObjLibrary.zip) files to your project directory.
4. Unzip the 2 zip files. You will have a folder named ObjLibrary and 13 other files.
5. In Visual C++, go to Project/Add Existing Item… and select main13.cpp (and holding control, continue to select), Sleep.h, Sleep.cpp, and GetGlut.h.
6. Ensure that Solution Explorer is visible (View / Solution Explorer). In the Solution Explorer, select Tutorial13 (the project name) and right click it. Choose Add / New Filter. A new folder icon should appear in the Solution Explorer. Name it ObjLibrary. (This creates an imaginary folder in the project, but it would not create a Windows folder on the computer).
7. In the Solution Explorer, right click on the ObjLibrary folder icon and select Add / Existing item. Go into the ObjLibrary folder on Windows and select all files. The files should appear in the Solution Explorer inside the ObjLibrary folder. You should get 26 files.
8. Compile and run the original program. You should see 2 windows: a text console and a graphical window labeled “Displaying OBJ Models with Shaders”. There is a spinning bucket in the middle of this second window. Exit the windows. Closing the text window will close the graphical window, and usually vice versa.

# Interfacing with glm

***OpenGL Mathematics (glm)*** *is a library designed to perform calculations for OpenGL programs. We need to use this because shader-based OpenGL removes many of the convenient functions we have been using.*

1. Go to <http://glm.g-truc.net/0.9.8/index.html> and download the newest version of the glm library to your project directory. The link will be in the top left of the window. You should get a zip file named something like glm-0.9.8.5.zip.
2. Unzip the zip file. You should get a folder named glm.
3. Inside the glm folder, there will be folders and files (5 folders and 6 files at time of writing). One of the inner folders will also be named glm. Delete everything except the inner glm folder.
4. Move the inner glm folder to your project directory. If you get a warning message about combining folders with the same name, just say OK.
5. The glm library only contains header files, so you do not have to add them to Visual Studio.
6. Open the ObjLibrary/ObjSettings.h file and uncomment the line that says:
//#define OBJ\_LIBRARY\_GLM\_INTERACTION
You will have to remove read-only-ness from the file. You can do this by hand, or Visual Studio will offer to do it for you when you try to save the file.
7. Recompile everything. In Visual Studio, go to Build/Rebuild Solution. The program should run as before. If you don't do this, your program may not compile. Even if it does compile, you may have errors later.

# Using the Shader-Based Version of the ObjLibrary

1. Go to <http://www2.cs.uregina.ca/~anima/409/Terms/201810/Tutorials/index.html> and download the [ObjLibraryShaders.zip](http://www2.cs.uregina.ca/~anima/409/Terms/201810/Tutorials/ObjLibraryShaders.zip) file to your project directory.
2. Unzip the zip file. It will add 23 files to the ObjLibrary folder and 4 files to the project directory.
3. In Visual C++, go to Project/Add Existing Item… and select GetGlutWithShaders.h (and holding control, continue to select), gl3w.h, gl3w.cpp, and glcorearb.h.
4. In the Solution Explorer, right click on the ObjLibrary folder icon and select Add / Existing item. Go into the ObjLibrary folder on Windows and select all the files. The new files should appear in the Solution Explorer inside the ObjLibrary folder. Visual Studio will ignore the files you added before. You should now have 41 files.
5. Open the ObjLibrary/ObjSettings.h file and uncomment the line that says:
//#define OBJ\_LIBRARY\_SHADER\_DISPLAY
6. In the main13.cpp file, replace the line that says
#include "GetGlut.h"
with
#include "GetGlutWithShaders.h"
7. Recompile everything. In Visual Studio, go to Build/Rebuild Solution. You will get a list of 12 error messages for the main13.cpp file.

# Replace Removed OpenGL Commands with glm Functions

*Warning: Many glm functions are quite picky about parameters and will not compile if given a double (e.g. 0.1) in place of a float (e.g 0.1f). This happens because glm uses a lot of templates.*

1. Add #includes for glm to main13.cpp after the #include for Sleep.h:
#include "glm/glm.hpp"
#include "glm/gtc/matrix\_transform.hpp"
Do not add a using directive for the glm namespace. It will cause errors.
2. Declare a global variable (before the main function) of the glm::mat4 type to represent the projection matrix:
glm::mat4 projection\_matrix;
3. In the reshape function, remove the 4 lines previously used to calculate the projection matrix:
 ~~glMatrixMode(GL\_PROJECTION);
 glLoadIdentity();
 gluPerspective(60.0, aspect\_ratio, 1.0, 1000.0);
 glMatrixMode(GL\_MODELVIEW);~~Replace them a command to set the projection matrix variable:
 projection\_matrix = glm::perspective(glm::radians(60.0f),
 aspect\_ratio,
 1.0f,
 1000.0f);
4. In the display function, remove the 4 lines previously used to set up the view:
 ~~glLoadIdentity();
 gluLookAt(2.0, 3.0, 4.0,
 0.0, 0.0, 0.0,
 0.0, 1.0, 0.0);~~Replace them a command to calculate the view matrix:
 glm::mat4 view\_matrix = glm::lookAt(glm::vec3(2.0f, 3.0f, 4.0f),
 glm::vec3(0.0f, 0.0f, 0.0f),
 glm::vec3(0.0f, 1.0f, 0.0f));
5. Add a command immediately after to calculate the combined view-projection matrix:
 glm::mat4 view\_projection\_matrix = projection\_matrix \* view\_matrix;
You will use the same view-projection matrix for every model you draw.
6. Add commands immediately before glPushMatrix to calculate the model transformation matrix for the bucket:
 glm::mat4 model\_matrix; // creates identity matrix
 model\_matrix = glm::translate(model\_matrix,
 glm::vec3(0.0f, -1.0f, 0.0f));
 model\_matrix = glm::rotate(model\_matrix,
 glm::radians(bucket\_degrees),
 glm::vec3(0.0f, 1.0f, 0.0f));
 model\_matrix = glm::scale(model\_matrix,
 glm::vec3(0.005f, 0.005f, 0.005f));
You will need a seperate model transformation matrix for each model.
7. Sadly, the program still will not compile. However, it should now only generate 6 error messages.

# Adapting the Program to use Shaders

1. Add code to the main function to initialize shader-based OpenGL. It should be after the glutCreateWindow function:
 int gl3w\_err = gl3wInit();
 if(gl3w\_err != 0)
 {
 // Problem: gl3wInit failed, something is seriously wrong.
 cerr << "gl3wInit failed, aborting." << endl;
 exit(1);
 }
2. Add an #include for ObjShader.h immediately before the #include for ObjModel.h:
#include "ObjLibrary/ObjShader.h"
3. Add a line to the init function immediately before loading the bucket model to initialize the ObjLibrary shaders:
 ObjShader::load();
4. Replace the #include for DisplayList with an #include for ModelWithShader.h:
#include "ObjLibrary/ModelWithShader.h"
This class is very different internally, but it is used in the same way.
5. Replace the bucket\_list global variable with a variable of the ModelWithShader type:
ModelWithShader bucket\_with\_shader;
6. In the init function, replace the command to initialize the old bucket\_list variable with one to initialize the new bucket\_with\_shader variable:
 bucket\_with\_shader = bucket.getModelWithShader();
7. Add a command to the display function immediately before glPushMatrix to draw the bucket:
 bucket\_with\_shader.draw(model\_matrix, view\_projection\_matrix);
Remove everything from glPushMatrix to glPopMatrix.
8. Compile and run everything. The program should produce the same output as at the beginning. However, it is now using shaders to display the bucket.

# Instanced Drawing

***Instanced drawing*** *refers to drawing many instances using a single call to the graphics hardware. This is considerably faster when you need to draw very many (100+) of the same model, each with its own transformation matrix. Instanced drawing is only available when using shaders.*

1. Use a double for loop to draw a 100 x 100 grid of buckets centered on the origin.
 for(unsigned int z = 0; z < 100; z++)
 for(unsigned int x = 0; x < 100; x++)
 {
 glm::mat4 model\_matrix; // creates identity matrix
 model\_matrix = glm::translate(model\_matrix,
 glm::vec3(x - 50.0f, -1.0f, z - 50.0f));
 model\_matrix = glm::rotate(model\_matrix,
 glm::radians(bucket\_degrees),
 glm::vec3(0.0f, 1.0f, 0.0f));
 model\_matrix = glm::scale(model\_matrix,
 glm::vec3(0.005f, 0.005f, 0.005f));
 bucket\_with\_shader.draw(model\_matrix, view\_projection\_matrix);
 }
2. You should see a large grid of buckets, each turning indepenantly. Your program will also run very slowly. (Mine was about 2 frames per second.)
3. Replace the model\_matrix variable declared inside the double loop with an array of 10000 elements displayed outside the loop:
 glm::mat4 model\_matrix[10000]; // one matrix per bucket
4. Inside the loop, calculate an index into the matrix array. Then initialize the matrix with that index:
 unsigned int index = z \* 100 + x;
 model\_matrix[index] = glm::translate(model\_matrix[index],
 glm::vec3(x - 50.0f, -1.0f, z - 50.0f));
 model\_matrix[index] = glm::rotate(model\_matrix[index],
 glm::radians(bucket\_degrees),
 glm::vec3(0.0f, 1.0f, 0.0f));
 model\_matrix[index] = glm::scale(model\_matrix[index],
 glm::vec3(0.005f, 0.005f, 0.005f));
5. Move the draw call into its own separate loop through the array:
 for(unsigned int i = 0; i < 10000; i++)
 {
 bucket\_with\_shader.draw(model\_matrix[i], view\_projection\_matrix);
 }
6. The big, slow grid of buckets should display the same as before. However, all the buckets are now being draw using model matrices in the same array.
7. Replace the loop that draws the buckets with a single instanced draw call:
 bucket\_with\_shader.drawInstanced(model\_matrix, 10000, view\_projection\_matrix);
8. The buckets should now display much faster.