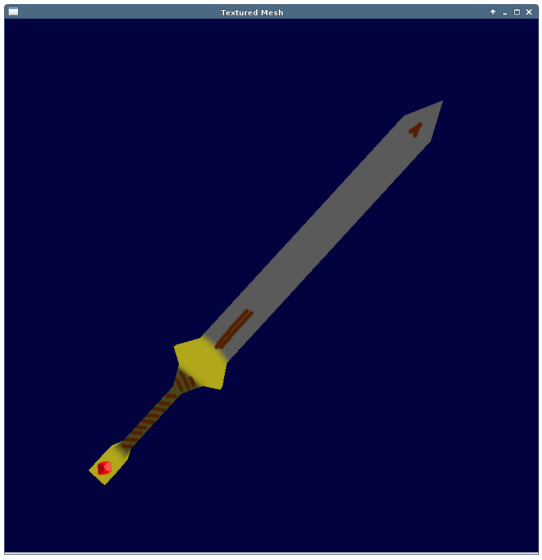
CS4052 Computer Graphics
Assignment 3

A. Gerdelan, Trinity College Dublin

Due date: Midnight, Tue 4 Nov 2014
Course Work %: approx. 10%

1 Outline



- You are required to create your own mesh with texture coordinates, load the mesh and a texture into OpenGL, and display these with **texture sampling**, as pictured above.
- This assignment is strictly individual (no group work).
- Submit your working programme on the due date, and the demonstrators will grade you.
- Submit a report on Blackboard by the due date. The report should include a written description and screen captures.
- If you fail to attend the lab or to submit the report on time, you will receive 0%.
- Demonstrating a project that was not created by you is considered cheating and must be reported as such. Demonstrators will check if you have an understanding of the code that you have written.
- In this assignment you must learn how to use modelling software of your choice. This can be tricky - **reserve some time to follow a modelling tutorial**.
- Loading a mesh from a file requires some loading code. You may use an external library for this. You may also use the Obj loading code that I provided, which does not require any linking, but has limited functionality.

- Loading an image from a file requires a library. I strongly suggest using Sean Barrett's **stb_image.h** for this as it is does not need linking.
- You may, of course, use any 3d meshes or texture images that you create in your game project - be creative!

2 Required Features

Your programme should have the following features. Grades are maximum possible for each feature:

- Successfully loaded an image into a texture, and displayed this in any way. (25%)
- Created your own 3d mesh using Blender, Maya, etc. (25%)
- Successfully imported and displayed a mesh in OpenGL (25%)
- Displayed your custom mesh, with correct texture coordinates, and texture sampling (25%)

3 Notes

I suggest that you **do not attempt to do everything in one go**, but rather make separate, minimal projects, and write a combined demo them after you get each part working.

1. Load up a basic triangle demo
2. Add 2d texture coordinates to your per-vertex data
3. In the fragment shader, use the texture coordinates as your output colour to test that they are working.
4. Load an image using an image loader library. Print the dimensions and number of channels loaded to check if it is as expected.
5. Copy into an OpenGL texture. Be sure that format (number of channels etc.) match the loaded image.
6. Sample the texture in the fragment shader, check if it is the right way up.

1. Load up a basic triangle demo
2. Add an object loader to your project
3. Export a single plane, or simple cube from modelling software
4. Replace the triangle points and point count, with those from the cube
5. Inspect the mesh file in a text editor, and print the first 3 points to make sure that you should see the first triangle in the view area
6. Most common error: viewing your mesh from (0,0,0) which is inside the middle of the mesh. Disable back-face culling to test.

Sean Barrett's stb_image.h is on Github: `https://github.com/nothings/stb`. The instructions for usage are in the top of the file's comments.

3.1 To use my basic Wavefront .obj loader

```
int point_count = 0; // number of vertex points loaded
GLfloat* vp = NULL; // array of vertex points
GLfloat* vn = NULL; // array of vertex normals (not needed for assignment)
GLfloat* vt = NULL; // array of texture coordinates
if (!load_obj_file ("my_mesh_file.obj", vp, vt, vn, point_count)) {
  fprintf (stderr, "ERROR: could not find mesh file...\n");
  // do something
}

// create a vertex points buffer
GLuint points_vbo;
glGenBuffers (1, &points_vbo);
glBindBuffer (GL_ARRAY_BUFFER, points_vbo);
// note: this is dynamically allocated memory, so we can't say "sizeof (vp)"
glBufferData (GL_ARRAY_BUFFER, sizeof (float) * 3 * point_count, vp,
  GL_STATIC_DRAW);

// -----==create a texture coordinates vertex buffer object here==-----

// free allocated memory
free (vp);
free (vn);
free (vt);
```

Note that you must retrieve per-vertex normals, even if you don't ever use them. My loader code only supports **triangulated** meshes that have points, normals, and texture coordinates.

Also note that exporting texture coordinates (UV) and normals are not enabled by default in Blender. When you export in obj format, be sure to check the appropriate boxes: