DW Normal Mapping Tutorial 2021

My ***indispensable*** video on how to debug shaders:

Part 1 (10 mins): <https://web.microsoftstream.com/video/aac1f9ec-6103-4656-8afa-bca1886de8b0>

Part 2 (3 mins): <https://web.microsoftstream.com/video/73425080-aa1e-4b82-9350-64cc8bd5ffc1>

This is only a rough guide! It assumes you understand how your current project loads and renders a mesh, and also that you understand how your pixel shader and vertex shader works (**if you don’t know this now, or need a refresher, now is the time…**). It also assumes you have lighting in the pixel shader which uses an 'eye' and 'light' direction for diffuse and specular lighting.

QUIZ

1. Make sure you are confident with all the quiz questions in the lecture. If not, ask the tutor
2. Do a little research: what is the difference between an OpenGL normal map and a DirectX normal map? Why, if you found some random normal map on the internet and used it in your code, would this look as though your code isn’t working? What is a really simple fix? Why might the simple fix not be such a good idea in the long run?

CODE

TLDR;

1. Remember on a normal map it is as though we are looking down the negative z axis towards texture (a plan view): so the – x is left / right, y is up / down, z points towards the viewer
2. Load the *normal* map texture in C++ and send to the shader (test it is loaded by temporarily using it as a colour texture)
3. Setup the tangents and binormals for your mesh (in C++) (code included in this tutorial) **IMPORTANT: READ THE IMPORTANT INFORMATION AT THE TOP OF THE SOURCE CODE!**
4. Send these tangents and binormals to the shader
   1. Check they are loaded correctly by debugging shader (videos above)
5. In the shaders:
   1. In vertex shader - create a TBN matrix to convert eye and light positions to tangent space
   2. In pixel shader load normal from normal texture, modify it as per lecture notes
   3. Use the *normal* and *tangent space* eye / light position in your lighting calculations
   4. Test
      1. Visually and by debugging shader
      2. Move the light around does it look correct?
      3. Move the camera around, does it look correct?

**Normal Mapping (the longer version)**

Loading the texture

1. Make a copy of the DX11 Framework.fx - rename it
2. When loading the fx file, load the newly named version
3. Load in the normal map dds file as a separate texture
   1. A normal map file is including with this tutorial, but feel free to use your own
      1. It should be very bumpy to show the technique working
   2. If you don't have any code to load in DDS textures, there are two files in the tutorial folder for this - **you may already have these in your framework**
      1. Use as follows:
      2. CreateDDSTextureFromFile(g\_pd3dDevice, L"Resources\\stone.dds", nullptr, &g\_pTextureRV);
   3. Make changes to the C++ and shader code to load the normal mapping texture
   4. As a test, use this normal map texture instead of the normal colour texture to check it is loading (the stone should show the purpley normal texture)

Loading the mesh

1. The easiest method would be to load a mesh in (if you currently know how to load a pre-built mesh) which has tangents and binormals
2. Note: you can pass in normal, binormal and tangent to the shader, or you can pass in just two and calculate the third in the shader (using the cross product)
3. Alternatively, in the tutorial folder there is some source code which you can use to add tangents and binormals to a mesh
4. Modify your vertex input layout descriptor to take in the tangent and binormal
   1. Look at the D3D11\_INPUT\_ELEMENT\_DESC
   2. Add a new 3 float vector for each new element (i.e. in case of tangent)
      1. { "TANGENT", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, D3D11\_APPEND\_ALIGNED\_ELEMENT, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

Modify the shader code to also include these new elements:

1. Look at the shader code in the lecture notes - make sure you understand it
2. Modify your shader code with the code from the lecture notes
3. test, debug.

Finally:

This is a complex task which should task your knowledge of the meaning of tangent space, normal maps, rendering meshes, shader code and transforming between world and tangent space (and what this means). If you have problems, go through your code carefully and make sure you understand each component - check what **should be** happening and what **is** happening.

RESEARCH

Test your normal is working correctly by doing three tests (one at a time, then together). In each case: does the lighting look correct?

1:

Rotate the cube on axis, one at a time

2:

Rotate the light around each axis.

3:

Rotate the camera around each axis.