Day 4&5: Lighting basics

Part 1: Getting ready (adding normals):

* In **vertex** shader:
  + vec3 vCol; to **vec4** vCol;
  + Add “attribute vec4 vNormal;”
  + Change the line with gl\_Position to handle vec4
  + Change the outputs from “vec3 color” to “vec4 fColour”
  + Add “out vec4 fNormal”
  + Change “color = vCol” to “fColour = vCol”
  + Add “fNormal = nNormal”
  + Change all the “attribute” to “in”
* In the **fragment** shader:
  + Change “in vec3 color” to “in vec4 fColour”
  + Add “in vec4 fNormal”
  + Change the vec3 of newColour to vec4
  + Add: “out vec4 pixelColour” to the top
  + Change “gl\_FragColor” to pixelColour;
* Change the vertex layout to include normal
  + Change the vertex structure
    - sVert includes a “float nx,ny,nz”
  + Change the shader, including the normal
    - attribute vec4 vNormal;
  + Change the vertex layout
* Change the “color” output to the vertex shader, also:
  + “color” to “fColour”, and to vec4, also
* Add “fNormal” output to the vertex shader
  + attribute vec4
* Add the normals to the models
  + Check if the model already has normal
  + If not, load into MeshLab, Choose “Filters”, “Normals, Curvatures, and Orientation”, then “Re-Compute Vertex Normals”
  + Export, with “Normal”, *under Vert,* checked
  + Make sure the file now has this header:
    - property float x
    - property float y
    - property float z
    - property float nx
    - property float ny
    - property float nz

    

Part 2: Turing on the lights:

Now that we have the normals loaded, and being passed into the shader, we can use them to “turn on the lights”.

We will be using a variation of the shader used in the “Graphic Shaders: Theory and Practice” book. The chapter we’re using is on FOL (chapter 6).

The complete lighting code is in the “Light\_code.txt”.

First, make sure the normals are being displayed by making the colour almost black (note: don’t set it to black, or the variables will dissaper), then *add* the normal to the colour.

We also need to pass the model/world matrix without any scaling or translation (just rotation). A way to do this is to calculate, then pass the “inverse transpose” of the model matrix.

* Add: “uniform mat4 matModelInvTrans;” to the **vertex** shader
* Add a C/C++ side variable to match, get the uniform location, and set it, in the draw call, where you are setting the matModel
* Add a line to calculate the inverse transpose:
* **matModelInvTran = glm::inverse(glm::transpose(matModel));**

Add the lighting call function (from the “Light\_code.txt”):

* Like C/C++, put the function at the bottom, and place the function signature at the top so you can call it.
* Also add the light structures and variables to the top, too (struct sLight, the “const” variables, and the sLight uniform array.

Add a “eye location” variable to the fragment shader:

* “uniform vec4 eyeLocation;”
* Add the code to the C/C++ side to set this to the location of the camera

Change the “newColour” to “materialDiffuse”

Add another uniform called “vec4 specularColour”

Set the uniform values for the lights…

