ASSIGNMENT 3: Terrain renderer using Shaders [20 marks]

In this assignment, you will implement an increasingly complex terrain renderer using each one of the programmable shaders.
You are provided with a working renderer written with OpenGL, GLEW, and GLFW, which draws using a phong shader. It responds to commands to move the camera around the plane (arrow keys) and displays a wireframe version (w).
It is set to render a 3D model of a banana. On line 550, if instead of loading a banana you pass the method an empty string, it will render a plane (check the LoadModel function to see how it does it, it will be useful later) with n_points * n_points forming a regular grid of triangles.
It is textured with a banana texture that is also loaded in (check LoadTextures).
Compilation instructions are available in the readme.txt file provided.

Your tasks are the following:

1: Terrain displacement map – Vertex shader

You are provided with a heightmap texture (mountains_height.bmp). Use it to set the vertex to the correct height in the vertex shader, given a certain scale constant.
For each pixel in the heighmap, you have a 24 bit integer encoded in the red, green, blue channels of your texture. As an example, let's say a pixel has the height 12345678 This will be encoded as such in RGB values according to the significance of that byte:

R = 10111100 = 188 = 0.73725	G = 01100001 = 97 = 0.38039	B = 01001110 = 78 = 0.30588

Meaning you can reconstruct the original value using shift operations!
12320768 + 24832 + 78 = 12345678

Finally, given you have displaced the vertices, the normal vectors that you pass in are now incorrect. You can calculate the correct normal vectors by sampling the values around the current pixel and estimating their differences.

0	3	6
1	4	7
2	5	8

Nx = differences between 0-6, 1-7, 2-8
Ny = fixed value.
Nz = differences between 0-2, 3-5, 6-8

2: Materials and shading – Fragment shader

You are provided with textures to be used as diffuse and specular color. Use them in your phong shader correctly, implemented in your fragment shader.
You have a texture of grass, rock, and snow. Make it so the fragment shader is able to choose which texture to apply according to the height of what it is shading. There should be a smooth transition between different textures being used. Implement tiling on your textures to ensure no obvious repetitions or ugly seams are noticed.

3: Mesh detail – Tessellation Shaders.

Instead of using a higher number of points (n_points) to create a higher quality mesh, you will implement using tessellation shaders to refine our mesh in GPU.
Create a tessellation control shader which will set the tessellation level to a fraction of the resolution of the heightmap, or its total size (if you have the GPU for it). Then create a tessellation evaluation shader which should now process the newly generated vertices and transform them.
You should also work on the C++ side to generate GL_PATCHES instead of GL_TRIANGLES in the LoadModel function, given that these are the primitives expected by the tessellation shaders.

4: Visual detail with billboards- Geometry shader

Finally, add a detail pass where you render vegetation (bushes, trees, flowers) as billboards in a geometry shader. If the height of a vertex being processed (which one should you use?) is in the range where you would texture it with grass, it should now have a random chance of having a billboard with vegetation. This should be done as an additional render pass to the previous one.

Notably, there is no random in GLSL. Here's a version that does the job:

```
float rand(vec2 co){
    return fract(sin(dot(co, vec2(12.9898, 78.233))) * 43758.5453);
}
```

where you can use the UV coordinates as input.

Marking scheme:

[5 marks] Materials
[5 marks] Terrain
[5 marks] Detail
[5 marks] Grass

DISCLAIMER:
Other than codebase you downloaded with this assignment, ALL code must be written by you personally. While discussing solutions with colleagues is allowed, sharing code is forbidden.

You **must** implement all of the functions above using the framework provided, as working within an existing codebase is a valuable skill to master, and will be graded in this coursework.

Your terrain renderer **must** work with the materials provided.

The code **MUST** run on the University's Linux system. You may implement on your own machine, but it will be tested on ours.

PENALTIES:

Poorly structured or badly commented code may be penalised by up to 25% of the marks available. Code that does not compile properly will be assigned a mark of 0, but I will usually give the student one chance to correct this.

DUE DATE: Friday, January 27, 10:00 am