CS541 Project 2

Synopsis

Implement bump mapping.

Instructions

Implement a bump map using a shader program. Use Phong lighting (with interpolated normals) to light your surface, but first perturb the normals via the bump-map calculation in order to get the "bumped" effect.

You may use the provided images or supply your own (or even compute your own). The two provided for this project contain the surface color in one and the height map in the other:

- **tile_color.raw**: A 256x256 RGB image of a single tile.
- **tile_height.raw**: A 256x256 single channel height image.

Vertex Shader

In addition to the usual attribute variables supplied for each vertex (surface normal, texture coordinate, and material properties), the application also provides a tangent vector called "vertexTangent". This is the P_u value in the equations derived in class. Declare

```
attribute vec3 vertexTangent;
```

in the vertex shader in order to access it. This value (as well as the normal) should be transformed using **gl_NormalMatrix** and then interpolated to each pixel via varying variables.

Pixel (fragment) Shader

To perturb the normal via the bump-map calculation, you'll need:

- N : the normal (interpolated from the vertex shader)
- P_u : the tanent (interpolated from the vertex shader)
- $P_v = N \times P_u$, the other tangent (sometimes called the bi-normal)
- f_u and f_v : Derivatives of the height map, approximated by evaluating the height map map at several points around the texture coordinate g**l_TexCoord[0].**
- s : An extra scale factor to account for the length of the unit normal versus the desired height of the "bumps".

The final calculation of the perturbed normal is: $\bar{N} = N + s(f_v P_u - f_u P_v) \times \frac{N}{\|N\|}$.

This normal is used in the Phong lighting calculation, but be sure to scale it to unit length first.

How to submit

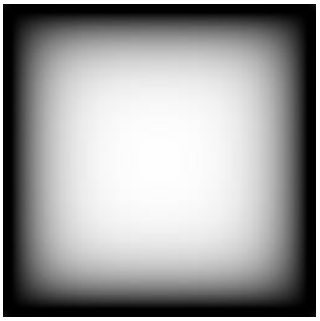
Use Moodle to submit a single zip file of your project.

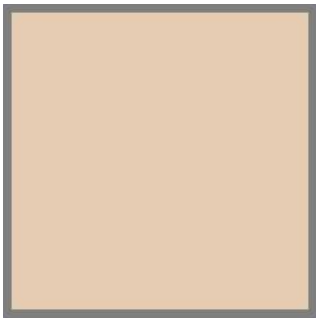
- Create a zip file containing everything I need to compile and run your project. This should include the *.vert, and .frag files, the *.h and *.cpp files, the *.vcxproj file, and any new image files you added to the project.
- Use the Moodle project submission link "CS541 Project 2" to submit the zip file.

Images:

These images are provided in a "raw" format that is trivially easy to read. To read a raw image of width **w,** height **h,** and depth (or channels) **d,** do:

```
unsigned char bytes[w*h*d];
FILE* f = fopen("....raw", "rb");
fread(bytes, w*h, d, f);
fclose(f);
```





Implementation details

See https://faculty.digipen.edu/~gherron/OpenGL/TOC.html for a complete set of Reference Pages describing all the following calls in detail.

Texture coordinates

The framework produces a sphere with texture coordinates exactly matched to the two images (i.e., polar coordinates). Your vertex shader must propagate the texture coordinate to the fragment shader via a built in varying variable like this:

```
gl_TexCoord[0] = gl_MultiTexCoord0;
```

and your pixel shader can access the resulting interpolated texture coordinates with

```
gl_TexCoord[0]
```

Sending texture

To put the texture on the graphics card and associate it with a texture id,do this:

```
int texId;
glGenTextures(1, &texId);
glBindTexture(GL_TEXTURE_2D, texId);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, 256,256, GL_RGB,
                                              GL_BYTE, bytes);
glBindTexture(GL_TEXTURE_2D, 0);
```

Many parameters can be set here. Some affect how the texture is stored in graphics memory, but most affect how the texture is accessed (e.g., bilinear/trilinear blending, mipmap usage, wrap/repeat/clamp modes …). For the single channel height map, replace both **GL_RGB**'s with **GL_LUMINANCE** (or **GL_INTENSITY** or **GL_ALPHA**) which expect one value per pixel instead of the three per pixel expected by **GL_RGB**.

Activate texture

To use a texture in a shader, the texture must be active in a texture unit, and when no longer drawing with a texture, it must be inactive.

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texId); //Activate in unit 0
<<draw stuff here>>>
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, 0); //Deactivate from unit 0
```

Repeat both sections with other textures using **GL_TEXTURE1** and so on for the other texture units.

Shader access

Shaders can access active shaders through samplers:

```
uniform sampler2D tile_color;
uniform sampler2D tile_height;
...
vec4 Kd = texture2D(tile_color, gl_TexCoord[0].st);
```

Both the vertex and pixel shaders can access textures through such a sampler.

Notify shader

The final connection between the shader and the application is to associate a texture ID (like texId above) with the sampler used to access the texture (**samper2D earth** above):

```
int loc = glGetUniformLocation(program, "tile_color");
glUniform1i(loc, 0); // Sampler tile_color will be in texture unit 0
```

This must be done after the shader is bound, but before the shader is executed – meaning before the geometry is drawn. Repeat for other textures in other texture units.