Normal Mapping

Description:
You will need to implement normal mapping using the OpenGL Shader Language (GLSL). There are three tasks for this lab: (1) set up your compile environment, (2) implement normal mapping, and (3) draw the light source as a yellow sphere.

Normal mapping is a common graphics trick to make a 2D object appear to be 3D. The idea is that an object will be given a “normal map” in the form of an image/texture. For every pixel, the normal of the surface at that pixel is the rgb value of the pixel \((r = x, g = y, \text{and } b = z)\). Given the normal (and a light vector), you can compute the shading based on Phong shading. The result of the shading should then be applied to the color of the pixel (in this case, to modify the brick color of the pixel).

Your Task:
- You will write the fragment shader that implements bump mapping.
  - Note: the vertex shader is implemented for you
  - The main.cpp has most of the set up ready for you, except:
    - You will need to pass the light vector from the CPU to the GPU, which you should do in:
      - drawLightSource()
    - To paint the sphere yellow, you might need to pass additional information from the CPU to the GPU.
  - Setup GLEW
    - GLEW is a library that makes programming shaders easy.
    - If you haven’t done this already in Lab0, you will need to set up your computer to use GLEW if commenting out \#include<GL/glew.h> doesn’t immediately work for you.
  - Implement the fragment shader
    - Hint: it can be done in between 5-10 lines of code.
Files Given:
main.cpp – Main
shaderManager.cpp and .h – This is a new file that shows how to load a file, and bind it to our program.
Ppm.cpp and .h – This is how you would load texture files
bump.frag and bump.vert – These are the shader files you’ll be working with

Data files given:
You are given two images:

Figure 1: A regular texture                 Figure 2: A normal map
(Image sources: http://www.bricksntiles.com/textures/)

The normal map is just like any image. The normal map has red, green, and blue (rgb) components. Each component together makes a normal that we can use to determine how much light each pixel receives.

GLSL Refresher – Keywords functions:

normalize (vec*) – normalizes a vector
dot(vec*, vec*) – Takes in two vectors a and b and returns a scalar
max(a, b) – Find the max between two scalar values
texture2D(sampler2D, takes in a texture coordinate).rgb – Retrieve colors from texture.

Example: texture2D(bump_image,gl_TexCoord[0].st).rgb
gl_TexCoord[0].st
gl_FragColor – Sets the final color for an individual fragment(pixel) in this shader
vec3, vec4 – Useful datatypes for creating floating point vectors

Going Further:
Did you enjoy this in class assignment?
• Add onto the shaderManager.cpp so you can bind and unbind different shaders to different objects.
• How can you control the depth of the bumps? Implement this in the interface
• Experiment! Try to implement a toon shader
• Load in the bunny.ply, texture it, and then add a normal map!

References
GLEE (GL Easy Extensions) http://sourceforge.net/projects/glee/?source=navbar