CS541 Project 3

Synopsis

Implement shadows using a two-pass shadow map algorithm.

Instructions

This project comes with a framework which implements a two-pass rendering algorithm, with an FBO containing a texture map that serves as output of the first pass and input to the second pass. As distributed, the shaders for pass 1 write nothing useful to that texture map, and the shaders for pass 2 do nothing with its content. The pass 2 shaders do implement an extremely simple “dot-product” lighting. The finished project will include:

- Pass 1 GLSL shaders which produce a shadow map (depth) texture
- Pass 2 GLSL shaders which will use that shadow map texture to determine if a pixel is shadowed or not, and then calculate the appropriate Phong lighting.
- Some C++ code in DrawScene to produce the shadow-mapping transformation.

Details for each piece follow.

Removing shadow-acne

Once you get basic shadow-mapping working, experiment with removing the shadow-acne. Several such methods are:

- **Front face culling while generating the shadow map:** Enable front-face culling by surrounding the drawing call with glEnable/glDisable:

  glEnable(GL_CULL_FACE)
  glCullFace(GL_FRONT)
  glCallList(scene.sceneDL)
  glDisable(GL_CULL_FACE))

  You may wish to enable/disable this via a boolean controlled by the keyboard.
- **Shadow depth offset:** Instead of comparing two depths for equality (always folly with floating point numbers), add a small offset to compensate for slight roundoff errors.

Project Report

Submit a project report in a file named **report.txt,** (or report.doc or report.odt, …) the both describes you implementation, and more importantly, describes your experiments and result with removing shadow-acne.

Grading Basis

Grading will be on a 100 point basis, with the following distribution:

- **Accuracy:** 70%, judgment based on looking at the results on the screen.
- **Removing acne**: 20%, based mostly evidence in your project report of what you tried and your judgment of how well it worked.
- **Code:** 10%, judgment based on looking at the code for proper implementation and reasonable commenting practices.

Pass 1

Pass 1 must produce, as output at each pixel, the depth of light penetration into the scene. The depth must be calculated in the vertex shader, interpolated to each pixel via a varying variable, and written out by the fragment shader.

Between passes

The transformation which converts from screen coordinates to light coordinates must be produced and made available to the pass2 shaders. This transformation is the product $\text{ShadowTransformation} = B P_L V_L V^{-1}$. You can find code to produce each of those matrices in procedure **DrawScene** in file **scene.cpp**, and you can use the graphics card machinery to do the multiplication, and make the result available to the pass2 shaders.

Pass 2

Pass two, computes each pixel's coordinate in the light's coordinate system. Call this quantity shadowCoord. ShadowCoord's (X,Y)/W is used to lookup a depth from the shadow map which is compared to shadowCoord's depth to determine shadow/not-shadow. The efficient way to calculate shadowCoord, is to calculate it in the vertex shader, and let a varying variable interpolate it to each pixel.

Vertex Shader

Shadow coordinate is calculated into a varying variable as:

shadowCoord = ShadowTransformation * gl_ModelViewMatrix * gl_Vertex

Fragment Shader

Lookup the shadow texture using projection (i.e., homogeneous) coordinates. Either

texture2D(depthTexture, shadowCoord.xy/ shadowCoord.w)

or let the texture lookup do the h-division:

texture2DProj(depthTexture, shadowCoord).

You should first check that the lookup coordinate falls within the texture coordinate bounds ($0 \le x/w \le 1$, and $0 \le y/w \le 1$), and do something appropriate if not.

The depth comparison between the light's depth (from the texture lookup) and the pixel's depth (from shadowCoord) determines shadow/no-shadow, and the Phong lighting calculation can proceed accordingly.