CS562 Project 1: Deferred Shading

Synopsis

Implement deferred shading as a first step toward a complete project. Demonstrate it with many small local lights, and perhaps one bright (possibly shadow casting) global light. You may use **any** class framework or game code (or anything else) as a starting point. If your chosen code has an implementation of a shadow map, keep it in working order for use in the next project.

Instructions

For this project you will implement basic deferred shading. The implementation should consist of the following passes. This ordering allows for **zero**, **one**, or **more** shadow casting lights. If you plan on only one shadow-casting light, this could be simplified slightly.

Looking ahead to further projects, I make this suggestion: The ambient, global light and local-lights should each be implemented in their own shader. If you plan only one shadow casting light, the ambient and global passes may be combined.

- **G-buffer pass:** (Output to the G-buffer render targets)
  - Draw all geometry with with the usual model/viewing/perspective transformations.
  - Vertex shader: Usual preparation for lighting calculation.
  - Pixel shader: Outout all information necessary to perform a later lighting calculation. This includes **world-space-position**, and **world-space-normal**, (or enough information to reconstruct those values) and whatever surface properties are needed for lighting. For instance, both Phong and the usual BRDF lighting calculations require: **diffuse** and **specular** colors, and roughness/gloss/exponent parameter.

- **Lighting pass:** Light your scene with the information contained in the G-buffer in these phases:
  - **Ambient light phase:**
    Output to **screen** or **final** render target with viewport set to its width-height
    Set blending off; depth-testing off; set ambient light amount
    Draw a FSQ (full screen quad) to invoke a pixel shader at all pixels
    Vertex shader: Pass the FSQ's vertex straight through, i.e., no transformation
    Pixel shader: Compute and output ambient light
  - **Global (shadow casting) light phase:**
    For each shadow-casting light:
    Shadow-Map calculation:
    Output to shadow map render target with viewport set to it's width-height
    Set blending off, depth-testing on; output to a shadow-buffer render target
    Create, as usual, a depth-map from the light's point-of-view.
    Lighting calculation:
    Output to **screen** or **final** render target with viewport set to its width-height
    Set blending on; depth-testing off; output to
    Send the depth-map and all light info to the shaders
    Draw a full-screen quad to invoke a pixel processor at all pixels
    Vertex shader: no transformation needed
    Pixel shader:
    Retrieve position/normal and surface values from G-buffer
    Shadow test as normal;
    Output (0,0,0) if the pixel is shadowed.
    Output light's BRDF contribution to pixel.
  - **Many local finite-range lights:**
    Output to **screen** or **final** render target with viewport set to its width-height
    Set blending on; depth-testing off (for now); front or back face culling on
    For each local light:
    Draw geometry to invoke a pixel shader over the light's range
    (Debugging hint: Use a full screen quad first, light's range geometry later.)
    Vertex shader: Usual M/V/P transformations; usual lighting calculations
    Pixel shader:
    Retrieve position/normal and surface values from G-buffer
    If pixel position is out of light's range output (0,0,0); otherwise
    Retrieve all lighting values and output light's BRDF contribution to pixel.

Framework

I don't supply a framework specific to this class. I can, if you wish, provide a very simple interactive graphics framework that was developed for other classes (CS541, CS300, CS251).

What to display

As an aid in debugging (and grading), implement some method of displaying each of the G-buffer values on the screen. This could include displaying each as a texture in small rectangle along the edge of the screen, of as full screen images in response to some user keystroke or other interaction. See images below.

What to hand in

Turn in a **project report** that outlines the project and describes how you implemented, exercised, tested and verified it. The report will naturally contain lots of screen captures from your running program, but **more important** is the text that describes the images and why they are relevant. The more you say about your project, the better, but it should include at least the following:

- The number of render targets, the values stored in them, and their layout.
- The 4 debug images shown below, and how they provide verification that the G-buffer is correct.
- A final image that shows the correct lighting.
- Images showing **lots** of local lights, and a discussion of how many such lights you can use and still experience real-time frame rates.

I require the program code only for completeness sake; I do not intend to compile the code to check for correctness of the project. This is to allow you to implement your project in your game without my having to learn to build and run the game.

Note: Your grade will be determined completely from the project report. It is important to put some quality effort into it. As an aid in this respect, I will provide several great examples from previous semesters.