Advanced Computer Graphics Assignment 3: Deferred Shading

Due date: check nestor

1 Assignment and grading

The complete assignment will be graded on a 0-100 scale, and the score will be broken down as follows

- 20 points: Questions
- 20 points: Multiple render targets
- 20 points: Attribute blending
- 20 points: Backface culling
- 20 points: Phong shading

It is important that you understand what you are doing, so we want you to answer some questions that are interleaved in the assignments. Answers should not be too long, just convey the basic idea. The answers to the questions should be saved to a file called `questions3.txt` (Note: `questions3.txt` is supplied, and has the questions written out). You will want to answer them as you complete the assignment, since many of them require experimentation.

What to hand in: Please hand in C and Cg sources for your code for each of the assignments. Also, after you filled in the answers to the questions in file `questions3.txt`, archive everything and submit as per the instructions on Nestor.

2 Reading material

- The course slides (can be found on nestor)
- Cg User's Manual and Reference Manual
- OpenGL reference pages `http://www.opengl.org/sdk/docs/man/`, for documentation on GL extensions (such as GL_EXT_framebuffer_object) see the extension registry `http://www.opengl.org/registry/`.
- Optional: the book Point-Based Graphics, by Markus Gross

3 Getting started

It is advised to start from the code of the previous assignment, as Deferred Shading is a logical extension to Gouraud Shading (in the context of point-based rendering). But do not overwrite main.cpp just yet, it contains some code you will need (so you should merge it with whatever you had).

As rendering only spheres gets boring, three point-based shape files have been provided for this assigment: a sphere, a bunny and a lion (Figure 1).

The `pts` files are in a very simple binary format that can be directly used for rendering. It contains the number of points (`int numpoints`), followed by the actual surfel data:

```
struct Surfel
{
    float pos[3];
    float color[3]; // Changed from 4 to save space
    float uvec[3];
    float vvec[3];
};
```

Code for reading these files has been provided.

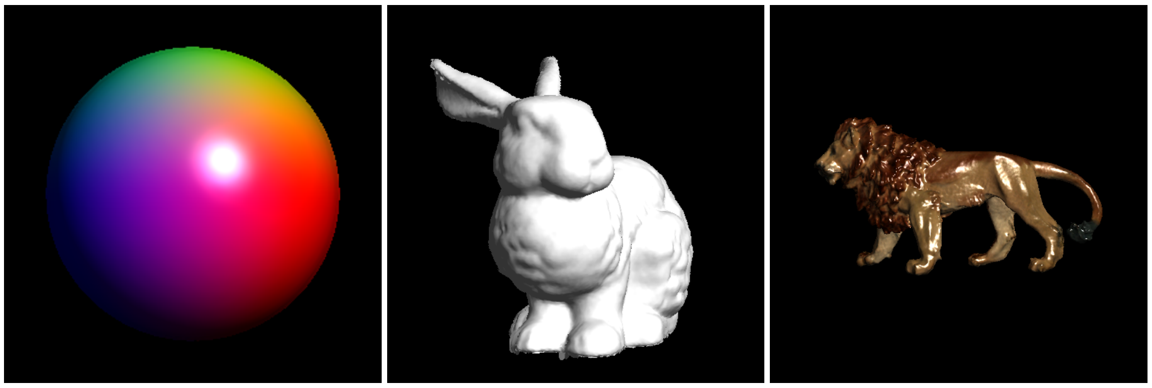

Figure 1: *Some example point sets:* `sphere.pts`*,* `bunny.pts`*,* `lion.pts`*. Light position:* $(2,2,0)$ *in eye space. Note that for the sphere a resolution of* 300×300 *was used to avoid running into hardware/driver-dependent limits to point sizes.*

4 Deferred Shading

The goal of the assignment will be to be able to render the different example objects using high quality point-based rendering. The technique that will be used is called deferred shading, in which surfel attributes are accumulated on a per-pixel basis, and the actual lighting and shading is done, along with normalization, in a post-processing pass.

4.1 Multiple render targets

With modern hardware, it is possible to render to at least 4 separate render targets simultaneously, all consisting of 4 floating point channels (RGBA). But for bandwidth reasons we would like to minimize the amount of attributes that are stored per pixel, restricting ourself to 2 RGBA16 render targets (8 floating point attributes).

Question 1: What attributes do we absolutely need to store in the render targets for diffuse and specular per-pixel Phong shading, in addition to the accumulated kernel weight (1 float)?

Add the necessary GL infrastructure for rendering to multiple render targets (textures), using FBOs with multiple color texture attachments. See the slides on how to do this.

4.2 Attribute blending

Change your point-splatting fragment shader to output the per-fragment attributes that you decided were necessary in last section.

In the previous lab session we were using a separate blend function for the alpha component as for the red/green/blue components, so that multiplication with the kernel happened automatically. This is no longer desired, as we want to be able to use the alpha component of the second texture for a generic attribute. Multiply your attributes with the kernel manually, and use cumulative blending for all components.

```
glBlendFunc(GL_ONE, GL_ONE);
```

For debugging purposes, you can show the contents of any of the attribute channels instead of the color in the normalization pass.

4.3 Backface culling

Up to now, we have blended in the attributes of all splats within a certain depth range, including back-facing ones. Blending the attributes of back-facing splats will result in artifacts at contours and with thin objects. Implement backface culling in your vertex shader: if a surfel doesn't face the viewer, set the homogeneous coordinate of the output position to -1 to cull the point.

Question 2: Instead of doing backface culling we can also set the `epsilon` parameter to a very small value to ignore the back-facing layers. Why is this not a good solution?

4.4 Phong shading

The full-screen postprocessing pass, that only normalizes the color in Gouraud shading, now has to be revisised to take the two intermediate textures as input, normalize all the attributes, and do Phong shading using the represented attributes.

Implement Phong shading with diffuse and specular components. One *positional point-light* suffices (in the first lab session this was done for a *directional light* with non-deferred shading, but you can probably re-use some of the code). Pass the light position in view space as a uniform parameter to the fragment shader.

Some hints:

- You can use the same transformation from screen position `WPOS` to a position on the near plane $\mathbf{q}_n$ as used in the ray-casting of splats, to determine the position of the pixel in view-space for doing lighting. $\mathbf{q}_n$ has to be multiplied with λ to get position $\mathbf{q}$.
- To make the blending nice and smooth might require some experimentation with the `epsilon` parameter (the depth offset, for the visibility splatting pass).

Question 3: Instead of storing explicit normals as attributes, the normal could also be calculated from the depth attribute by taking directional derivatives. Name one advantage and a disadvantage of doing this.

Your final submission should be able to render the three example objects smoothly without serious artifacts.