Project 4 Discussion

Robb T. Koether

Hampden-Sydney College

Mon, Nov 2, 2009
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
Introduction

- This *program* will implement the lighting model in the CPU.
- You will draw the same object that you drew in Project 3.
The user will have the choice between GPU lighting and CPU lighting.

A pop-up menu will let the user choose between a point light source and directional lighting and between a local viewer and an infinite viewer.

Each choice will be reported in the text window.

If CPU lighting is working correctly and the user switches between CPU and GPU lighting, there should be no apparent change in the rendered scene (except that it takes longer).
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
### The User Interface

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press +</td>
<td>Zoom in</td>
</tr>
<tr>
<td>Press −</td>
<td>Zoom out</td>
</tr>
<tr>
<td>Press ←</td>
<td>Rotate the camera left</td>
</tr>
<tr>
<td>Press →</td>
<td>Rotate the camera right</td>
</tr>
<tr>
<td>Press ↑</td>
<td>Rotate the camera up</td>
</tr>
<tr>
<td>Press ↓</td>
<td>Rotate the camera down</td>
</tr>
<tr>
<td>Press &gt;</td>
<td>Increase fineness of mesh</td>
</tr>
<tr>
<td>Press &lt;</td>
<td>Decrease fineness of mesh</td>
</tr>
<tr>
<td>Press G</td>
<td>GPU lighting</td>
</tr>
<tr>
<td>Press C</td>
<td>CPU lighting</td>
</tr>
<tr>
<td>Right-click</td>
<td>Pop-up menu</td>
</tr>
</tbody>
</table>
The User Interface

- The pop-up menu will contain four items:
  - Point light source
  - Directional light source
  - Local viewer
  - Infinite viewer
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
Recall the Phong lighting model from Lecture 15:

\[(s_a + L_a)m_a + L_d m_d (l \cdot n) + L_s m_s (h \cdot n)\alpha.\]

Most of this information (the non-geometric information) we will obtain by querying the GPU.

The geometric information (eye point, look point, light position) we will have to pass as function parameters.
The geometric variables in the equation are $l$, $n$, and $h$.

However, $l$ is determined by the vertex and the light position.

And $h$ is determined by $l$ and $v$, which is determined by the vertex and the eye point.
The Lighting Model

Therefore, the geometric information required is:

- The vertex
- The normal
- The light position
- The eye point
- The look point (for infinite viewer)

The vertex and the normal are stored in the Mesh object, so we need to pass the eye point, the look point, and the light position as parameters.
1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The `CPUdraw()` Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
The `CPUdraw()` Function

- We will add a `CPUdraw()` function to the `Mesh` class.
- The prototype is
  
  ```cpp
  void Mesh::CPUdraw(Point3D& eye, Point3D& look, Point3D& light);
  ```

- This function will use the equation of the lighting model to calculate the shade of each vertex.
- The shade will then be set by the `glColor()` function and the vertex will be rendered.
Outline

1 Introduction
2 User Interface
3 Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4 Design
5 Due Dates
Every OpenGL state variable stored in the GPU may be obtained by using the various OpenGL query functions.

For example,

- `glGetInteger()`
- `glGetFloat()`
- `glGetLight()`
- `glGetMaterial()`, etc.
For example, to find the ambient light level of Light 0, use the `glGetLight()` function:

```c
float la[4];
glGetLightfv(GL_LIGHT0, GL_AMBIENT, la);
```

To find the ambient material reflection, use the `glGetMaterial()` function:

```c
float ma[4];
glGetMaterialfv(GL_FRONT, GL_AMBIENT, ma);
```

See Appendix B of the Red Book for more details.
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The CPUdraw() Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
The Light Position

- We will pass the light position as a parameter, even though it can be obtained by the `glGetLight()` function.
- This is because the light position is stored in eye coordinates, not world coordinates.
- The same would be true for the eye point and the look point, except that they are not stored!
- Why not?
The Light Position

On the other hand, we will need the homogeneous coordinate of the light position to tell whether the light source is directional.

That coordinate is the same in eye and world coordinates.
Your design should answer the following questions.

- Which query functions will be needed to obtain the values required by the lighting model?
- How will you handle the choice between a positional light and a directional light?
- How will you handle the choice between a local viewer and an infinite viewer?
Outline

1. Introduction
2. User Interface
3. Discussion
   - The Lighting Model
   - The `CPUdraw()` Function
   - Querying the GPU
   - The Light Position
4. Design
5. Due Dates
## The Due Dates

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Mon, Nov 9</td>
</tr>
<tr>
<td>Program</td>
<td>Mon, Nov 16</td>
</tr>
<tr>
<td>Corrections</td>
<td>Return date + one week</td>
</tr>
</tbody>
</table>