CG Programming II (VGP 352)

Agenda:

- Outline the course.
- Brief OpenGL review.
  - Lighting and shading.
  - Drawbacks of OpenGL's shading model.
- Phong shading.
  - OpenGL 1.3 features that allow real Phong shading.

Website & Mailing List

- Course website:

http://people.freedesktop.org/~idr/2007-VGP352/

- All assignments and course material will be available there...usually before class.
- There is also a mailing list:

http://lists.paranormal-entertainment.com/mailman/listinfo/aipd-vgp35x

Course Outline

- Advanced shading & lighting.
  - Phong shading and per-pixel lighting
  - Fresnel reflection.
  - Bidirectional reflectance distribution functions
    - Isotropic lighting with BRDFs.
    - Anisotropic lighting with BRDFs.
- Shadows.
  - Shadow map based techniques.
  - Stencil-buffer based techniques.

Course Work – Reading

- Lots of reading!
  - Weekly reading from the book will provide background information on each new topic.
  - Readings from academic papers will provide details of specific algorithms and areas of research.
- Each week someone will present a synopsis of one of the assigned papers.
  - Everyone will present once. There is no escape!

Course Work – Programming

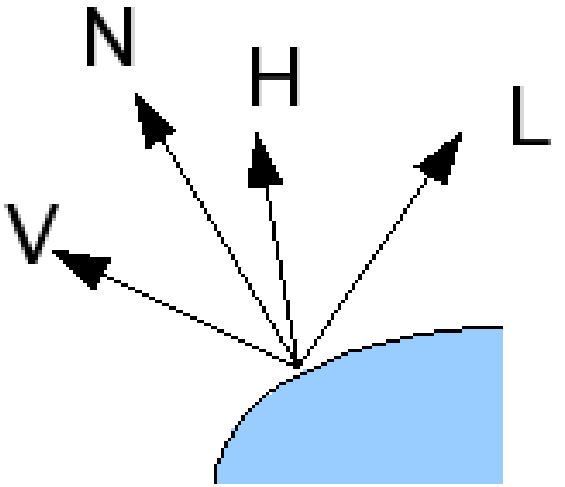
- Lots of coding!
  - Like last term, there will be weekly programming assignments.
    - The grading criteria and assignment requires *will* spelled out more carefully and completely.
    - I'm also toying with the idea of having assignments submitted differently.
  - There will be a term project, but it will structured differently than last term.
    - It will incorporate more of the previous assignments.
    - No collision detection. ;)

Course Work – Exams

- There will be a midterm and a final.
  - Unlike last term, there will be a pre-test so that you know *exactly* what to expect.
  - Unlike last term, the tests will focus more on how different techniques might be applied to achieve a desired result.
  - The tests will still be hard.

Lighting in OpenGL

- Three types of lighting calculations used in OpenGL.
  - Diffuse $I_d = K_d \times L_d \times max(L \cdot N, 0)$
  - Specular $I_s = K_s \times L_s \times max(N \cdot H, 0)^n$
  - Ambient $I_a = K_a \times L_a$
- Calculated per light.



Shading in OpenGL

- When `GL_SMOOTH` shading is used, lighting is calculated per-vertex.
- Calculated lighting values (i.e., colors) are interpolated down each edge of the polygon, then across each scan-line.
  - Also known as Gouraud shading.
- This is fast and easy to implement in the hardware of 1992 when OpenGL 1.0 was born.

What's the problem with this shading model?

- Since lighting is only performed at vertexes, it is easy miss specular highlights...

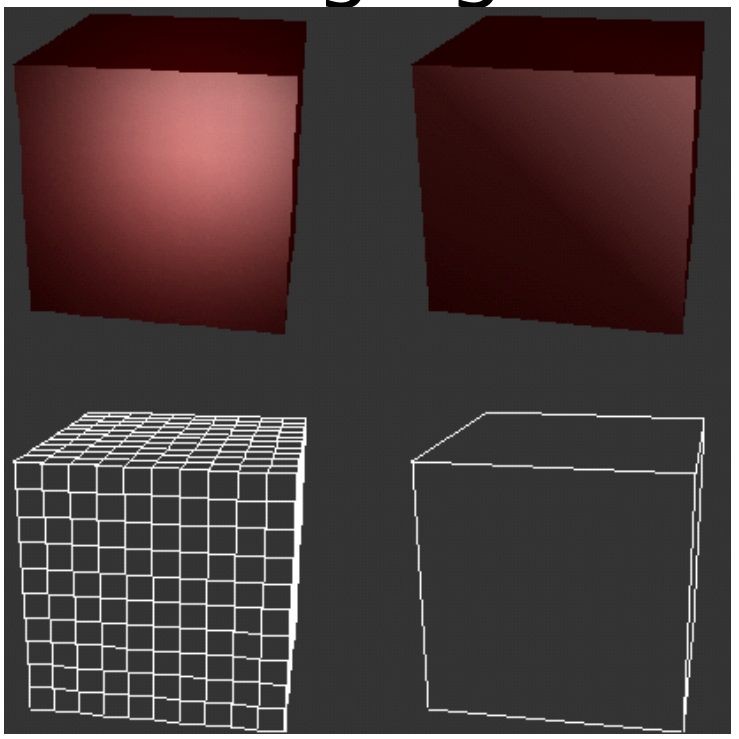


Image from M. Kilgard, “Avoiding 16 Common OpenGL Pitfalls”, 1998.

Phong Shading

- Last term we discussed Phong's *lighting* model, but there is also a Phong shading model.
  - We're going to use Phong shading with Blinn's lighting model.
- Phong shading interpolates normals and performs lighting calculations at each pixel.
  - Much more expensive, but hardware is *really* fast these days.

How can we do this in OpenGL?

- Two problems must be solved:
  - Interpolating surface normals.
  - Performing a per-pixel dot product.
- How can we do these operations in OpenGL?
  - The interpolation step is the easy part.

DOT3 Texture Combine Mode

- The DOT3 texture combine mode can be used to perform per-pixel lighting calculations.
  - Available since OpenGL 1.3 or `ARB_texture_env_combine_dot3`.
  - Pretty much any card since original Radeon or original Geforce supports this in some form.
    - Some old cards may only support `EXT` version, which is *slightly* different.

DOT3 Texture Combine (cont.)

```
glTexEnvi(GL_TEXTURE_ENV,
          GL_TEXTURE_ENV_MODE,
          GL_COMBINE);

/* Store resulting dot product in color
 * and alpha components.
 */
glTexEnvi(GL_TEXTURE_ENV,
          GL_COMBINE_RGB,
          GL_DOT3_RGBA);
```

Putting It Together

- Store light color in texture environment color.
- Store surface normals (in surface space) in a texture.
- Store surface gloss map in a texture.
- Store H vector in per-vertex diffuse color.
  - H *must* be calculated H per-vertex in C code!
- Configure combiners to calculate:

$$(diffuse \cdot texture_0) \times env \times texture_1$$

Putting It Together (cont.)

- This math seems to require one more multiply than we can do in two texture stages.

$$(diffuse \cdot texture_0) \times env \times texture_1$$

- We can get the extra multiply by using the alpha blender.
  - Store the dot product in the alpha.
  - Configure the blender to do `GL_SRC_COLOR * GL_SRC_ALPHA`.
- What is the *range* of color & texture data?

Quick Alpha Example

```
/* Enable alpha blending. */
glEnable(GL_BLEND);

/* Multiply incoming fragment by it's
 * alpha and store in result pixel.
 */
glBlendFunc(GL_SRC_ALPHA, GL_ZERO);
```

Only Specular?

- This only give specular. What about diffuse and ambient?
  - We can get free ambient using `glSecondaryColor` and `glEnable(GL_COLOR_SUM)`.
- Several ways to get diffuse:
  - Use more texture units (if available).
  - Use a second pass.
    - Will slightly different combine and alpha blend logic.

Questions?

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.