Lighting and Shading

CS148: Intro to CG
Instructor: Dan Morris
TA: Sean Walker
July 14, 2005

Pre-lecture business

- meshes.cpp example from last class
- pp2 is due today
- pp3 goes out today
- start thinking about pp4
- Remote students: email fax #'s

Outline for today

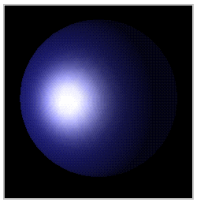
- Lighting
- Shading
- Lighting and shading in OpenGL

What have we done so far?

- We can model objects or load objects that someone else modeled
- We can create a scene with objects and a camera
- We know how our objects get transformed and rasterized

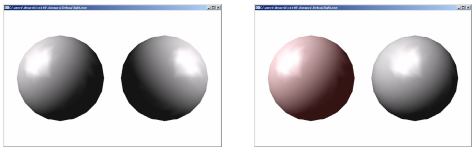
What color is an object?

- So far, we've used glColor3f() to say "my object is blue"
- But what color should a blue sphere's pixels *really* be?



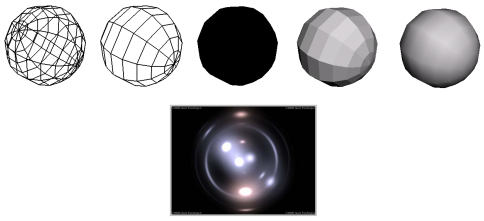
What color is an object? [light.cpp]

- Pixel color – and color in the real world – depend on:
  - Light color and position
  - Material properties
  - Camera position
  - Object geometry, medium



We can't capture all of these things…

- …so we make approximations depending on our compute power, our need for realism, etc.



Illumination Models

- Part of our approximation is our mathematical representation of how light interacts with objects
- For example, one illumination model might be:

  pixel color == object color

What would a sphere look like under this illumination model?

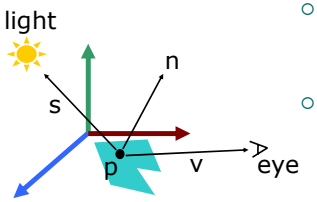
A blue sphere with this illumination model

Illumination Models

- Today we'll look at the individual illumination models that make up the OpenGL lighting system:
  - Diffuse lighting
  - Ambient lighting
  - Specular lighting
- All of our examples will be in grayscale for right now

Lighting framework

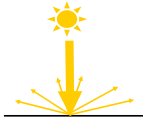
- We want to decide what color some point p on an object should be from a viewer's perspective
- What data do we have to work with?



- Light
  - Position
  - Brightness
- Object
  - Material
  - Surface normal

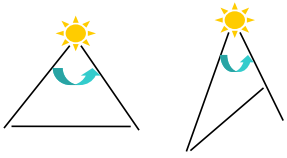
Diffuse Lighting Model

- A fraction of incoming light is reradiated "diffusely" in all directions
- Some of this light will reach the eye
- Since light is radiated equally in all directions, the orientation of the surface relative to the eye doesn't matter



Diffuse Light: Surface orientation

- The orientation of the surface relative to the *light* does matter
- The "brightness" of a surface depends on how many photons hit a unit area
- If the light "sees" more of the object, it will hit it with more photons



When the object faces away from the light, a smaller portion of the light's energy hits a unit area on the object

Diffuse Light: Lambert's Law

- If **s** (vector to light) and **n** (normal) are "more aligned", the object should be brighter
- **What operation tells us "how aligned" two vectors are?**
- Lambert's law:

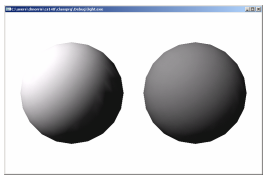
$I_d = L_d(u_s \bullet u_n)$

  - I_d: output (pixel) brightness
  - L_d: light intensity
  - $\mathbf{u}_s$ and $\mathbf{u}_n$: unit versions of **s** and **n**

What's the most important missing piece here?

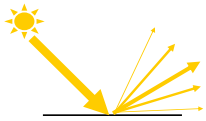
Diffuse Light: Diffuse Coefficient

- r_d: diffuse reflection coefficient
- Captures many complex properties of an object in one number: how much light does it reflect?
- OpenGL convention: 0.0 --> 1.0



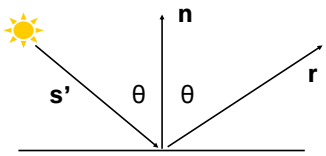
Specular Lighting Model

- Light scatters most strongly in one direction
- Specifically, light scatters most strongly in the same direction a mirror would reflect



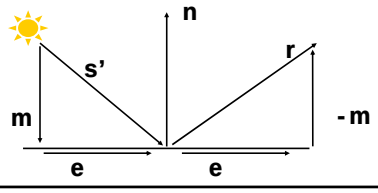
Specular (Mirror-Like) Reflection

- Angle of incidence == angle of reflection
- Clearly **r** will be useful in deciding how bright the object appears…



What is **r** in terms of **s** and **n**?

- Decompose s':
  - $\mathbf{m} = (\mathbf{s}' \bullet \mathbf{u}_n)\mathbf{u}_n$ (parallel to **n**)
  - $\mathbf{e} = \mathbf{s}' - \mathbf{m}$ (perpendicular to **n**)
- What is **r** in terms of **e** and **m**?
  - $\mathbf{r} = \mathbf{e} + (-\mathbf{m})$
  - $\mathbf{r} = \mathbf{e} + (-\mathbf{m}) = (\mathbf{s}' - \mathbf{m}) - \mathbf{m} = \mathbf{s}' - 2\mathbf{m}$
  - $\mathbf{r} = \mathbf{s}' - 2\,(\mathbf{s}' \bullet \mathbf{u}_n)\,\mathbf{u}_n$



Specular Illumination

- More light goes out along the direction **r**
- The more the view axis v lines up with r, the brighter the object should appear
- $I_s = L_s r_s(u_r \bullet u_v)$

What does each term mean?

- Doesn't capture the fact that some materials are "more specular" than others
- r_s tells us what color the material reflects, but now how that color falls off with angle

Specular Illumination: Phong Model

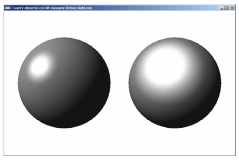
- Add a parameter *f* that tells us "how specular" a material is:

  $I_s = L_s r_s (u_r \bullet u_v)^f$

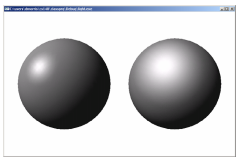
- Higher f = more specular; light falls off faster as **v** moves away from **r**
- In OpenGL terminology, *f* is "shininess"

Specular Illumination: Examples

Which sphere is shinier?



Same shininess as above spheres, both have reduced r_s



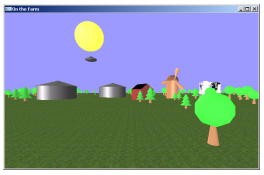
Ambient Lighting Model

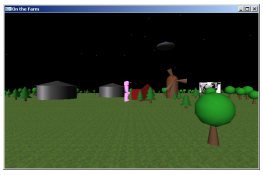
- Stuff pointing away from lights isn't really pitch black
- But we can't model all the reflections in a real scene...
- So we just add "ambient lighting" that doesn't depend on orientation:

  $I_a = L_a r_a$

Ambient Lighting: Examples

- Too much ambient light: everything gets washed out
- Too little ambient light: very deep shadows





Putting it All Together

- In our illumination model, the light at a point is equal to:

  $I = L_a r_a + L_d r_d (\mathbf{u}_s \bullet \mathbf{u}_n) + L_s r_s (\mathbf{u}_r \bullet \mathbf{u}_v)^f$

- We can do this computation for all lights in a scene and add the results together

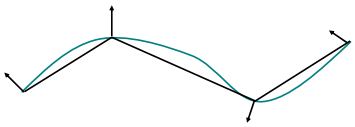
What happens if the surface normal points *away* from the light?

Outline for today

- Lighting
- Shading
- Lighting and shading in OpenGL

An aside: Normals in OpenGL

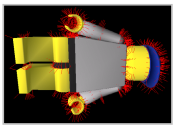
- Normals in OpenGL are associated with *vertices*, not faces
- Why? Objects in 3D graphics are usually discrete approximations of continuous real-world objects
- Surface normal actually changes over the surface of the "real" object
- Vertex normals are "samples" of the real normal



An aside: Normals in OpenGL

- Before I call glVertex(), I usually call:
- glNormal3f(x,y,z);

  To tell GL what the current surface normal is.
- Normals are transformed through the whole pipeline, so they're specified in *object* coordinates
- Mesh files usually specify normals



Limitations of our lighting model

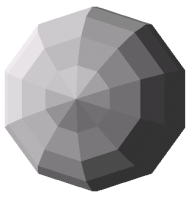
- Now we know how to compute the pixel color for a given point
- This took a few multiplications, so it would be expensive to do this for every single pixel
- Plus, we don't usually *have* exact surface normals everywhere, since we often approximate curved objects with flat polygons

Shading

- *Shading* is the process of filling polygons with color based on the illumination at some points on the polygon
- Usually we evaluate I (illumination) at vertices, and use that data to shade the polygon

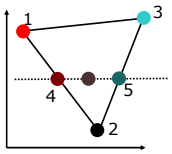
Flat Shading

- Simplest, fastest shading algorithm:
  - Pick a point on the polygon
  - Compute illumination at that point
  - Fill the whole polygon with that color

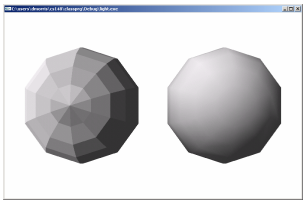


Gouraud Shading

- Most realistic shading algorithm supported by OpenGL
  - Compute illumination at all vertices
  - Interpolate illumination values when rasterizing the polygon



Flat Shading vs. Gouraud Shading



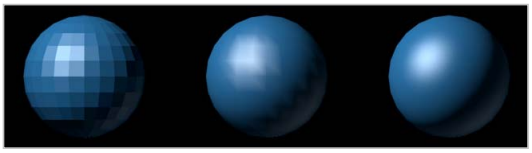
When might I *want* flat shading?

Phong Shading

- Instead of interpolating color, interpolate *normals* and re-compute color at each pixel
- Also called "per-pixel" lighting

Phong Shading

- More realistic images without additional geometry
- Allows specular highlights *within* a face
- Not supported by OpenGL, but starting to be supported by hardware



Outline for today

- Lighting
- Shading
- Lighting and shading in OpenGL

OpenGL is state-based

- **Name an OpenGL command that actually results in a *change* to the framebuffer.**
- Most commands just set up state that will affect what happens when you send the next vertex
- Lighting and materials work this way…

GL Lighting: Globals

- Set up a shading model:
  glShadeModel(GL_SMOOTH);
  glShadeModel(GL_FLAT);
- Enable lighting:
  glEnable(GL_LIGHTING);

These things generally happen once in your program…

GL Lighting: Lights

- Turn on one or more lights:
  glEnable(GL_LIGHT0);
- Specify light properties:

```
float pos[4] = { 1.0 , 1.0 , 1.0 , 1.0 };
float ld[4] = { 1.0 , 1.0 , 1.0 , 1.0 };
…
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glLightfv(GL_LIGHT0, GL_DIFFUSE, ld);
glLightfv(GL_LIGHT0, GL_AMBIENT, la);
glLightfv(GL_LIGHT0, GL_SPECULAR, ls);
```

- These things generally happen once per frame…

GL Lighting: Lights

- Lights are transformed by the modelview matrix but not the projection matrix

In what coordinate system does GL do its lighting calculations?

- Sometimes I want a light to "move with the camera", sometimes I want it to stay "fixed in the scene"

How do I specify a light that moves with the camera without knowing the camera position?

How do I specify a light that stays fixed in the scene?

GL Lighting: Materials

- Specify material properties:

```
float shininess[1] = { 50.0 };
float diff[4] = { 1.0 , 0.0 , 0.0 , 1.0 };

glMaterialfv(GL_FRONT, GL_DIFFUSE, diff);
glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
glMaterialfv(GL_FRONT, GL_SHININESS,
shininess);
```

- These things generally happen once or a few times per object…

GL Lighting: Geometry

- Do just what we've always done to render primitives, but add a normal to each vertex:

```
glBegin(GL_TRIANGLES);
glNormal3f(1.0,0.0,0.0);
glVertex3f(2.0,4.0,5.0);
glNormal3f(1.0,0.0,0.0);
glVertex3f(2.0,4.0,7.0);
…
```

- Normals are generally specified for each vertex

GL Lighting: Geometry

- Normals are transformed through the whole pipeline along with their vertices

What coordinate system do we specify normals in?

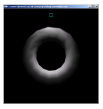
- Normals must be of unit length, unless you call glEnable(GL_NORMALIZE);

Why do we almost *never* enable GL_NORMALIZE?

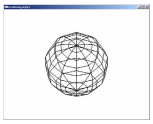
What type of transformation will badly mess up your lighting?

GL Lighting: Examples

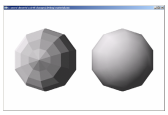
[movelight.cpp]



[rendering.cpp]



[materials.cpp]



Next Time

- Texture-mapping