Lighting and Shading

5 March 2009
CMPT370
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

What's on for today

- Lighting and shading
  - The global rendering equation
  - Light-material interaction
  - Kinds of light sources
- The OpenGL local illumination model
  - Ambient term
  - Diffuse term
  - Specular term
  - Specifying in OpenGL
- See RedBook ch5

Shading for realism

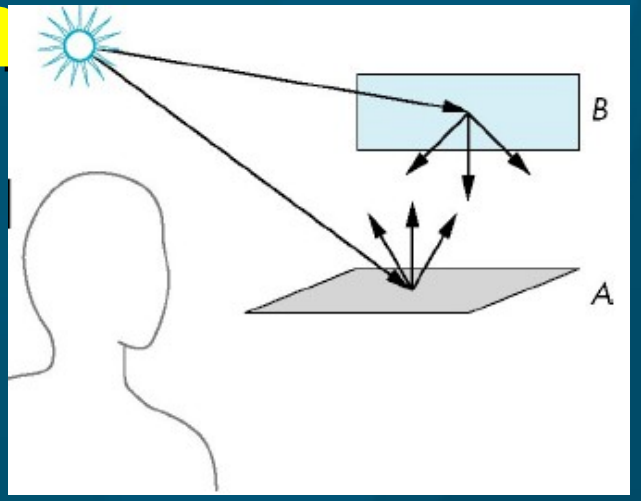
- Colour is part of the OpenGL state
  - Specify glColor (glColor3f, glColor4b, etc.) before adding vertex
- Red ball:
    - glColor3f(1.0, 0.0, 0.0);
    - glVertex3f(...); ...
- Flat-shaded:

- Not realistic!

Factors involved in shading

- What makes the real sphere look like this?
- Interactions between light and material:
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

The rendering equation



- Light originates from light sources
- Each time light strikes a surface:
  - Some absorbed, some scattered

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int \rho(x,x',x'')\, I(x',x'')\, dx''\right]$$

$I(x,x')$: *intensity from x to x'*
$g(x,x')$: *visibility between x, x'*
$\epsilon(x,x')$: *transfer emittance from x to x'*
$\rho(x,x',x'')$: *scattering from x to x' via x''*

- Cannot be solved analytically in general
- Global illumination: all objects, all light sources
- OpenGL pipeline is local: one polygon at a time

Light – material interaction

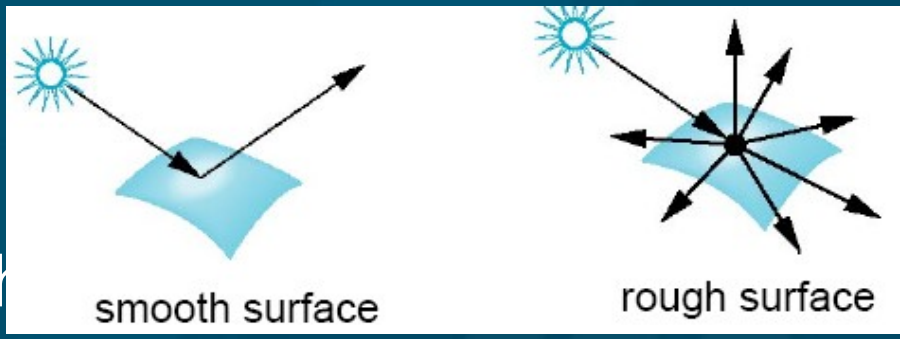
- Light striking a material is
  - Partially absorbed
  - Partially scattered (reflected):
    - Depends on smoothness, orientation of surface
- A surface looks red because it absorbs everything else and reflects the red component of light

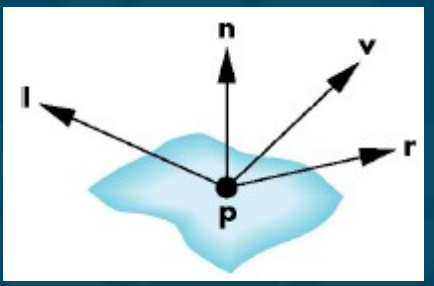
Light sources

- General (area) light sources: must integrate light from all points of light source: hard!
- Simple kinds of light sources:
  - Ambient light: uniform light everywhere
    - Models contribution of many sources
  - Point source: has position and colour
    - Directional light: position is infinitely far away
  - Spotlight: restrict light to a cone
    - Can have falloff at edges of cone

OpenGL local illumination

- Smooth: reflect concentrated in one direction
  - Rough surfaces scatter light in all directions
- The OpenGL illumination model has 3 parts:
  - Ambient light
  - Diffuse scattering (rough)
  - Specular reflection (smooth
- Uses four vectors:
  - To source (l), To viewer (v)
  - Surface normal (n), Ideal reflection (r)



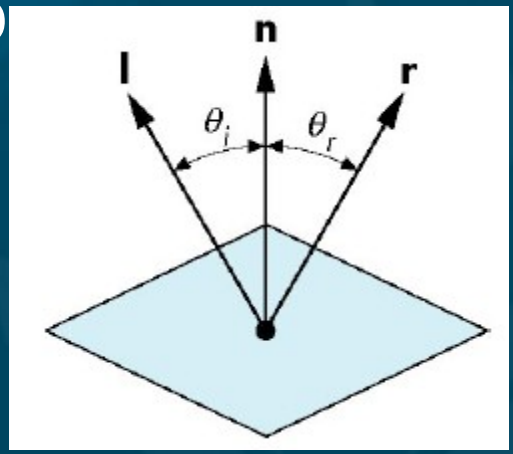


Ambient light

- Simple model of multiple interactions between large area light sources and many objects
  - Shadowless
- Intensity and colour depends on:
  - Colour of ambient light: k_a
  - Reflectivity of surface material with respect to ambient light: I_a
- Ambient term is $k_a I_a$

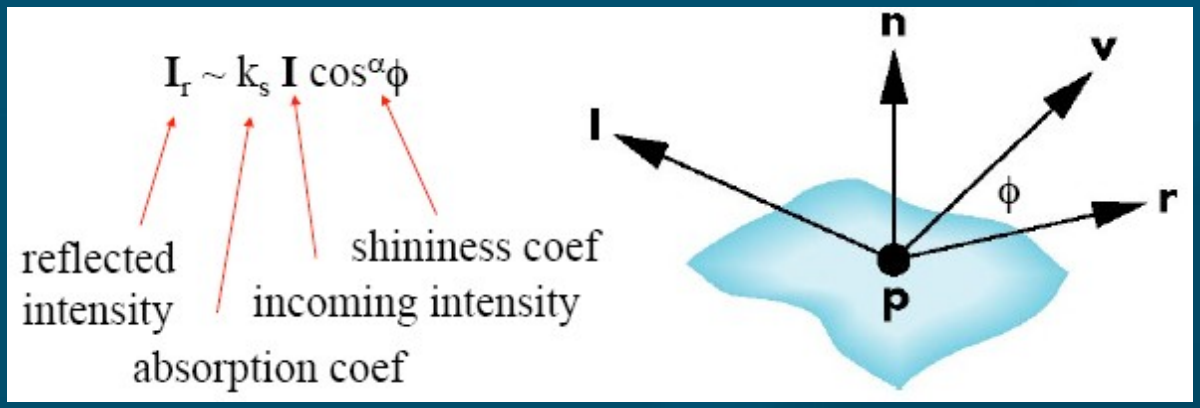
Lambertian surfaces

- An perfectly specular surface reflects all incident light in the direction of reflection
  - Angle of incidence equals angle of reflection
- A perfectly diffuse (Lambertian) surface scatters all incident light equally in all directions
  - Light reflected is proportional to $\cos(\theta_l) = l * n$
  - Diffuse colour of surface k_d also modulates reflected light
- Diffuse term is $k_d I_d (l * n)$

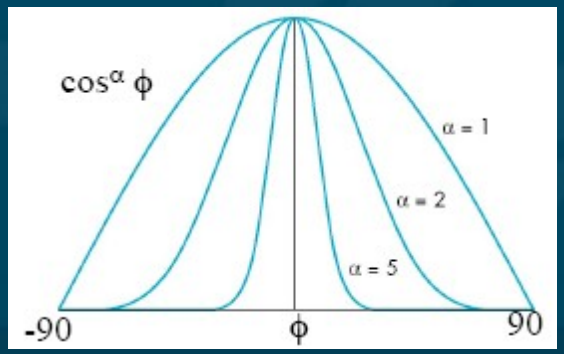


Shininess

- If viewer is looking along the reflection vector, we see the specular highlight
- Phong added falloff if viewer is slightly off from ideal reflection vector: shininess coefficient



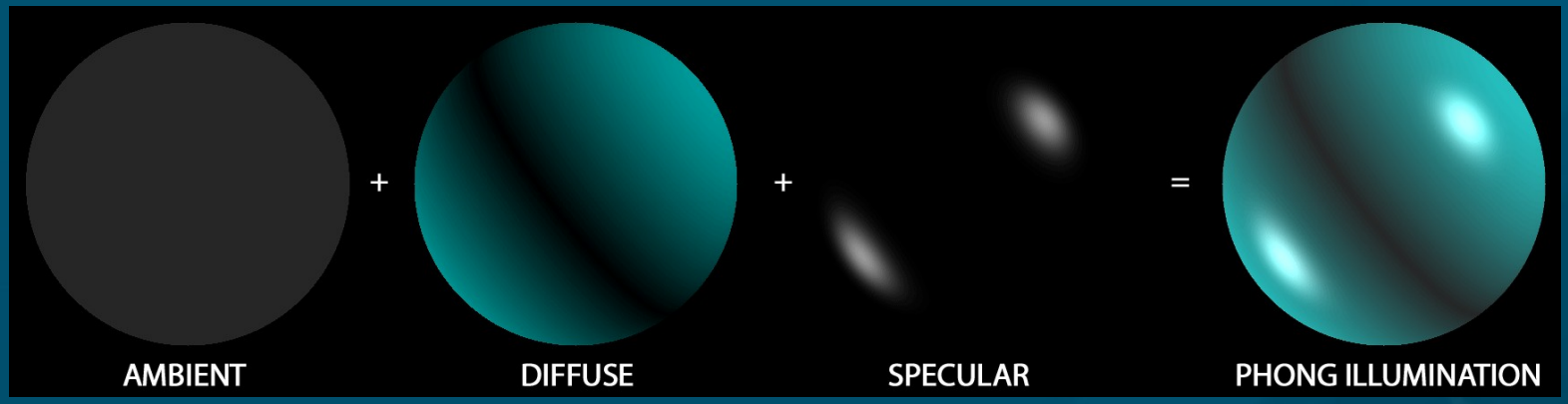
- cos(φ) is dotproduct of view vector v and reflection r
- Specular term is: $k_s I_s (v * r)^{\alpha}$



Putting it together

- Intensity of a surface patch from our view is

$$I = k_a I_a + k_d I_d (l * n) + k_s I_s (v * r)^{\alpha}$$



Light properties (9):

- Ambient colour I_a
- Diffuse colour I_d
- Specular colour I_s

Material properties (10):

- Absorption coefficients: Ambient k_a, diffuse k_d, specular k_s
- Shininess coefficient α

Doing this in OpenGL

- Enable shading and select shading model
- Specify lights
- Specify material properties
- Specify geometry and normals

Selecting lighting model

- Enable lighting (otherwise only flat-shading):
  - glEnable(GL_LIGHTING);
  - Also have to enable each light source:
    - glEnable(GL_LIGHT0);
    - glEnable(GL_LIGHT1);
    - Up to GL_MAX_LIGHTS lights (at least 8)
- Set lighting model parameters:
  - Set global ambient light colour
    - glLightModelif(GL_LIGHT_MODEL_AMBIENT, r, g, b)
    - Other params: GL_LIGHT_MODEL_LOCAL_VIEWER, GL_LIGHT_MODEL_TWO_SIDED

Defining lights

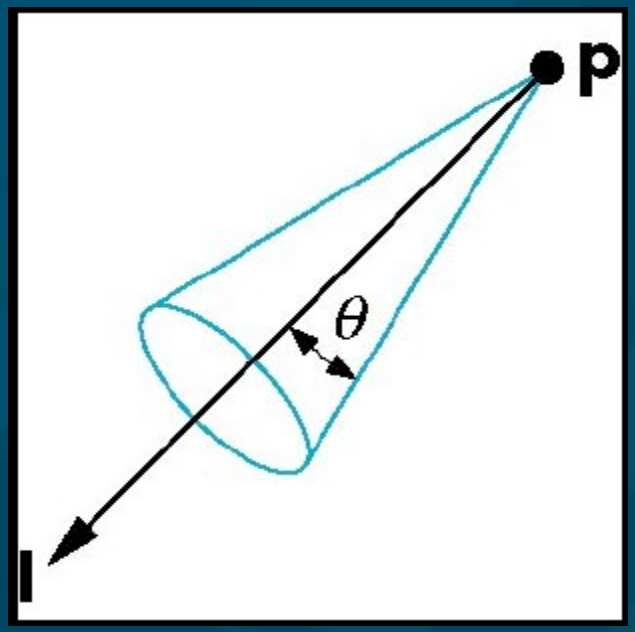
- Point source: position, colours (amb, diff, spec)
  - `GLfloat diffuse0[] = {1.0, 0.0, 0.0, 1.0};` `// RGBA`
  - `GLfloat ambient0[] = {1.0, 0.0, 0.0, 1.0};`
  - `GLfloat specular0[] = {1.0, 0.0, 0.0, 1.0};`
  - `Glfloat light0_pos[] = {1.0, 2.0, 3,0, 1.0};`
    `// homogeneous coords`
  - `glEnable( GL_LIGHTING );`
  - `glEnable( GL_LIGHT0 );`
  - `glLightv( GL_LIGHT0, GL_POSITION, light0_pos );`
  - `glLightv( GL_LIGHT0, GL_AMBIENT, ambient0 );`
  - `glLightv( GL_LIGHT0, GL_DIFFUSE, diffuse0 );`
  - `glLightv( GL_LIGHT0, GL_SPECULAR, specular0 );`

Directional light source

- The position of a point source is specified in homogeneous coordinates:
  - w=1.0: light is a point source
    - (x,y,z) give coordinates of position
  - w=0.0: light is a directional source
    - (x,y,z) give vector
    - **Glfloat light0_pos[] = {1.0, 2.0, 3,0, 1.0};**
    - **glLightv(GL_LIGHT0, GL_POSITION, light0_pos);**
- Light sources are geometric objects, too:
  - Affected by current model-view matrix

Spotlights

- Spotlights have:
  - RGBA colour (amb, diff, spec)
  - Position
  - Direction
  - Cutoff distance
  - Attenuation exponent α
    - Falloff is proportional to $\cos^{\alpha}\varphi$

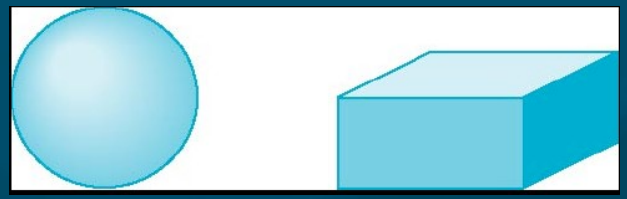


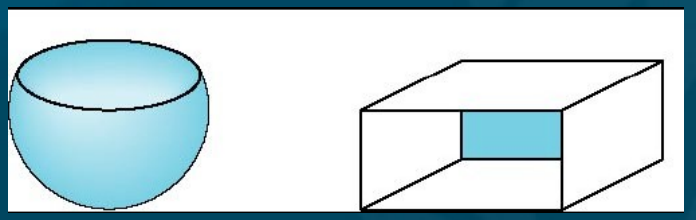
Material properties

- Part of the OpenGL state: specify before the vertices/polygon to which they apply
  - **GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};**
  - **GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};**
  - **GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};**
  - **GLfloat shine = 100.0;**
  - **glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);**
  - **glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);**
  - **glMaterialfv(GL_FRONT, GL_SPECULAR, specular);**
  - **glMaterialfv(GL_FRONT, GL_SHININESS, shine);**

Front and back face materials

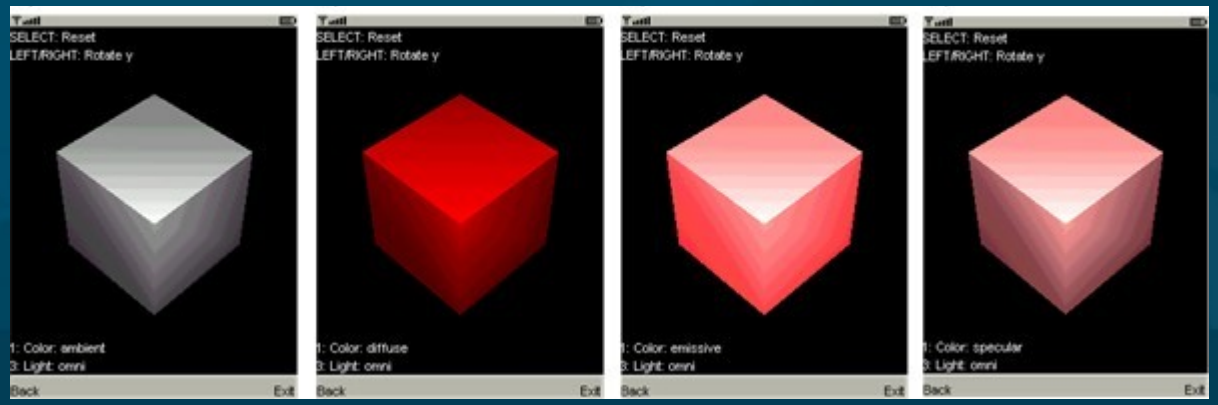
- Recall back-face culling: don't render faces which point away from camera ($v * n > 0$)
- Two-sided lighting disables back-face culling
  - Front and back faces can get different material properties
  - Use GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK in glMaterialf()





Emissive light

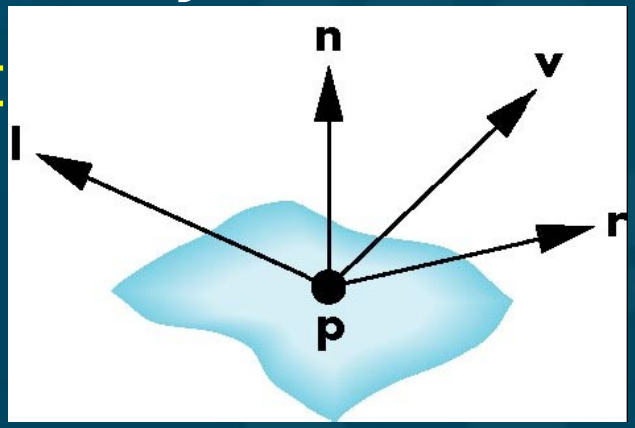
- An extra feature OpenGL throws in is the emissive term:
  - **GLfloat emission[] = 0.0, 0.3, 0.3, 1.0);**
  - **glMaterialf(GL_FRONT, GL_EMISSION, emission);**
  - Extra light added to the shading equation:
    - I = (amb) + (diffuse) + (specular) + (emissive)
  - Simulates glowing object
  - Does not shine light on other objects



(a) ambient
(b) diffuse
(c) emissive
(d) specular

Shading needs normal vectors

- The illumination model relies on four vectors:
  - To light (l): specified by the model/scene
  - To viewer (v): specified by model-view matrix
  - Surface normal (n)
  - Reflection (r): compute from l, n
- Computing normals is not always easy
  - Depends on how we represent the surface
  - OpenGL leaves this up to us (in our application)



Shading polygons

- We specify in our model for each vertex:
  - Vertex coordinates
  - Vertex colours
  - Vertex normal

- Use lighting model to calculate vertex shades
- Smooth shading: vertex shades are interpolated across the polygon
  - glShadeModel(GL_SMOOTH);
- Flat-shading uses the colour of the first vertex:
  - glShadeModel(GL_FLAT);

Calculating normals

- OpenGL expects us to find the normals
- Some shapes can be done analytically:
  - Sphere: normal points from centre
  - No vertex normals: flat-shaded
  - With vertex normals: smooth-shaded
    - Note silhouette edge
- How to find normals for general surfaces?