What color do we make these solid objects? Depends on object and light!

How does lighting work?

In the real world..... Light bounces off everything. All objects influence all others.

Global illumination: hard to do - must consider all objects, interactions, interdependence (1 depends on 2 depends...). Good for getting complex lighting effects. An advanced topic.

In the CG world..... Local lighting - decision of how to light a point on an object depends on:
- Surface AT that point
- Eye position
- Lights
LOCAL LIGHTING

Consider only 1 point on 1 object
No shadows
No self shadows
No color spill
No inter-reflection
No area light sources - point sources only
if you want these, add with a hack

3 parts (per light)
specular (direct reflection)
 diffuse (scattering)
ambient (back for indirect)

Lighting is a hack
real lighting is complex
microstructure of materials

get "biggest" features of lighting correct

Phong models are still hacks just get more features right
Lighting vs Shading

What color is a point?

Physics: depends on how light interacts with all objects in scene
- some of the objects reflected light goes off towards eye ⇒

CG: do some computation to determine color Shader

color = Shader(info)

- What info do we give the shader?

Simple Shading:
- reflectance
  - object properties (color)
  - light info (position, color, intensity)
  - eye position
  - local geometry (position, normal)
Diffuse Shading

- Matte objects
- Rough surfaces
  "Micro surface texture" scatters light in all directions

- Chalk, paper, unpolished wood or stone, ...

Lambertian reflects
scatter light in all directions equally

Eye position doesn't matter

Light position DOES matter (relative to surface orientation)

Consider fixed sized object:

\[
\begin{array}{c}
\text{\rotatebox{90}{\text{Fixed size}}} \\
\text{\rotatebox{90}{\text{Object}}} \\
\text{\rotatebox{90}{\text{Light}}} \\
\end{array}
\]

Amount of light that hits is \( \propto \cos \theta \) where \( \theta \) = \( \angle \) between light and normal

\[
D \propto \hat{n} \cdot \hat{l}
\]
One last problem -

What about inter-reflected light -
room isn't totally black.

☐ ← this side of object should have 
    some light

"Ambient" light = indirect light that is just 
bouncing around

Hack is add in a light source that effects all 
objects equally - Ambient lighting
Specular (direct reflection)

Perfect mirror

\[ \begin{align*}
\theta_i & \quad N & \quad \theta_r \\
E \quad & \quad & \quad L
\end{align*} \]

ping-pong ball model

\[ \hat{\theta} = \theta \text{ incidence} = \theta \text{ reflection} \]

Light gets to eye only if things line up exactly

\textbf{Hack} -> if it's close to the eye, that's good enough

Falloff as it gets further away

\[ L \approx \hat{E} \cdot \hat{R} \cdot c_L \]

* need a falloff function

\textbf{Phong Model} \quad L \approx (\hat{E} \cdot \hat{R})^p \cdot c_L \quad \text{specular coefficient}

\textbf{Easier Way} \quad H = \text{half-way angle}

\[ L = (\hat{N} \cdot \hat{H})^p \cdot c_L \]
Hack Lighting Model (CL)

1. Eye Position
2. Object Local Geometry (normals)
3. Each light source has a position (may be at infinity) and a brightness (color) \( I_i \)
4. Ambient light has a brightness (color) \( A \)
5. Surface has a diffuse reflective color \( C_D \), a specular color \( C_S \), a shininess \( s \), and an ambient color (reflective) \( C_A \)

\[
\text{color} = A \cdot C_A + \sum_{i \in \text{lights}} \left( I_i \cdot (C_D \cdot (\hat{n} \cdot \hat{i})) + C_S (\hat{n} \cdot \hat{i})^s \right)
\]
Some improvements:

1. Falloff (brightness depends on distance)
2. More sophisticated ways of finding Co, Cs based on position
3. More complex reflectance function
   \[
   \text{BRDF} = \text{bi-directional reflectance distribution function}
   \]
   given - input direction; output direction \Rightarrow reflectance
How to use this?

Polygons are all the same color (one normal)

- FLAT shading
  - approximation $\hat{\mathbf{L}}$ and $\hat{\mathbf{N}}$ do change, only a little

Problem:

- polygons are an approximation to a smooth surface
- normal per vertex

1. compute color at vertices
   - linearly interpolate color
   - GOURAUD Shading

2. linearly interpolate normals
   - compute lighting per-pixel
   - PHONG SHADING

(do not confuse with Phong LIGHTING)