Today

- Local Illumination & Shading
  - The BRDF
  - Simple diffuse and specular approximations
  - Shading interpolation: flat, Gouraud, Phong
  - Some miscellaneous tricks
- Normal Vectors
Local Shading

- Local: consider in isolation
  - 1 light
  - 1 surface
  - The viewer
- Recall: lighting is linear
  - Almost always...

Counter example: photochromatic materials

Examples of non-local phenomena
- Shadows
- Reflections
- Refraction
- Indirect lighting
The BRDF

- The Bi-directional Reflectance Distribution Function
  - Given: \( \rho = \rho(\theta_V, \theta_L) = \rho(\mathbf{v}, \mathbf{l}, \mathbf{n}) \)
    - Surface material
    - Incoming light direction
    - Direction of viewer
    - Orientation of surface
  - Return:
    - Fraction of light that reaches the viewer
  - We’ll worry about physical units later...

The BRDF

- Spatial variation capture by “the material”
- Frequency dependent
  - Typically use separate RGB functions
  - Does not work perfectly
  - Better: \( \rho = \rho(\theta_V, \theta_L, \lambda_{in}, \lambda_{out}) \)
Obtaining BRDFs

- Measure from real materials
- Computer simulation
  - Simple model + complex geometry
- Derive model by analysis
- Make something up

Images from Marc Levoy
Beyond BRDFs

- The BRDF model does not capture everything
  - e.g. Subsurface scattering (BSSRDF)

Images from Jensen et. al, SIGGRAPH 2001

Beyond BRDFs

- The BRDF model does not capture everything
  - e.g. Inter-frequency interactions

$$\rho = \rho(\theta_V, \theta_L, \lambda_{in}, \lambda_{out})$$  This version would work....
A Simple Model

- Approximate BRDF as sum of
  - A diffuse component
  - A specular component
  - A “ambient” term

Diffuse Component

- Lambert’s Law
  - Intensity of reflected light proportional to cosine of angle between surface and incoming light direction
  - Applies to “diffuse,” “Lambertian,” or “matte” surfaces
  - Independent of viewing angle
  - Use as a component of non-Lambertian surfaces
Diffuse Component

Comment about two-side lighting in text is wrong...

\[ k_d I (\hat{l} \cdot \hat{n}) \]

\[ \max(k_d I (\hat{l} \cdot \hat{n}), 0) \]

Diffuse Component

- Plot light leaving in a given direction:

- Plot light leaving from each point on surface
Specular Component

- Specular component is a mirror-like reflection
- Phong Illumination Model
  - A reasonable approximation for some surfaces
  - Fairly cheap to compute
- Depends on view direction

\[ k_s I (\hat{r} \cdot \hat{v})^p \]
\[ k_s I \max (\hat{r} \cdot \hat{v}, 0)^p \]
Specular Component

- Computing the reflected direction

\[ \hat{r} = -\hat{l} + 2(\hat{l} \cdot \hat{n})\hat{n} \]

\[ \hat{h} = \frac{\hat{l} + \hat{v}}{||\hat{l} + \hat{v}||} \]

Specular Component

- Plot light leaving in a given direction:

- Plot light leaving from each point on surface
Specular Component

- Specular exponent sometimes called “roughness”

![Diagram of Specular Component with different roughness values](image.png)

Ambient Term

- Really, it's a cheap hack
- Accounts for “ambient, omnidirectional light”
- Without it everything looks like it’s in space
Summing the Parts

\[ R = k_a I + k_d I \max(\hat{l} \cdot \hat{n}, 0) + k_s I \max(\hat{r} \cdot \hat{v}, 0)^P \]

- Recall that the \( k_i \) are by wavelength
  - RGB in practice
  - Sum over all lights
Metal vs Plastic

Metal vs Plastic
Other Color Effects

Images from Gooch et al, 1998
Measured BRDFs

BRDFs for automotive paint

Images from Cornell University Program of Computer Graphics

Measured BRDFs

BRDFs for aerosol spray paint

Images from Cornell University Program of Computer Graphics
Measured BRDFs

Images from Cornell University Program of Computer Graphics

BRDFs for house paint

Images from Cornell University Program of Computer Graphics

Measured BRDFs

Images from Cornell University Program of Computer Graphics

BRDFs for lucite sheet
Details Beget Realism

- The “computer generated” look is often due to a lack of fine/subtle details... a lack of richness.

Direction -vs- Point Lights

- For a point light, the light direction changes over the surface
- For “distant” light, the direction is constant
- Similar for orthographic/perspective viewer
**Falloff**

- Physically correct: $1/r^2$ light intensify falloff
  - Tends to look bad (why?)
  - Not used in practice
- Sometimes compromise of $1/r$ used

---

**Spot and Other Lights**

- Other calculations for useful effects
  - Spot light
  - Only light certain objects
  - Negative lights
  - etc.
Surface Normals

- The normal vector at a point on a surface is perpendicular to all surface tangent vectors.

- For triangles normal given by right-handed cross product.

Flat Shading

- Use constant normal for each triangle (polygon).
  - Polygon objects don’t look smooth.
  - Faceted appearance very noticeable, especially at specular highlights.
  - Recall mach bands...
Smooth Shading

- Compute “average” normal at vertices
- Interpolate across polygons
- Use threshold for “sharp” edges
  - Vertex may have different normals for each face

Gouraud Shading

- Compute shading at each vertex
  - Interpolate colors from vertices
  - Pros: fast and easy, looks smooth
  - Cons: terrible for specular reflections

Note: Gouraud was hardware rendered...
Phong Shading

- Compute shading at each pixel
  - Interpolate *normals* from vertices
  - Pros: looks smooth, better speculars
  - Cons: expensive

![Gouraud and Phong shading](image)

Note: Gouraud was hardware rendered...

Transforming Normals

- Normals are directions
  - But they must maintain underlying geometric property
    \[ \hat{n} \cdot (a - b) = 0 \]
    \[ \hat{n}^T (a - b) = 0 \]
  - A bit of manipulation
    \[ \hat{n}_{\text{new}}^T (Ta - Tb) = 0 \]
    \[ \hat{n}_{\text{new}}^T T(a - b) = 0 \]
    \[ \hat{n}_{\text{new}}^T T^{-1} T(a - b) = 0 \]
    \[ \hat{n}_{\text{new}} = T^{-T} \hat{n} \]
Transforming Normals

- Special transformation rule for normals
  \[ \hat{n}_{\text{new}} = T^{-T} \hat{n} \]
- Note: A rotation is its own inverse transpose

Depth Distortion

- Recall depth distortion from perspective
  - Interpolating in screen space different than in world
  - Ok, for shading (mostly)
  - Bad for texture
We know the $S_i$, $P_i$, and $b_i$, but not the $a_i$. 
Depth Distortion

\[ S_1 = \frac{P_1}{h_1} \]

\[ S_2 = \frac{P_2}{h_2} \]

\[ S_3 = \frac{P_3}{h_3} \]

\[ S_4 = \frac{P_4}{h_4} \]

\[ X = \sum \limits_i S_i b_i \]

\[ X = \frac{Q}{h} = \left( \sum \limits_i P_i a_i \right) / \left( \sum \limits_j h_j a_j \right) \]

\[ Q = \sum \limits_i P_i a_i \]

\[ \sum \limits_i S_i b_i = \left( \sum \limits_i P_i a_i \right) / \left( \sum \limits_j h_j a_j \right) \]
Depth Distortion

\[ S_1 = \frac{P_1}{h_1} \]

\[ S_2 = \frac{P_2}{h_2} \]

\[ S_3 = \frac{P_3}{h_3} \]

\[ S_4 = \frac{P_4}{h_4} \]

\[ X = \sum_i S_i b_i \]

\[ \sum_i P_i b_i / h_i = \left( \sum_i P_i a_i \right) / \left( \sum_j h_j a_j \right) \]

\[ Q = \sum_i P_i a_i \]

\[ b_i / h_i = a_i / \left( \sum_j h_j a_j \right) \quad \forall i \]

Independent of given vertex locations.
Depth Distortion

Linear equations in the $a_i$.

$$\sum_j h_j a_j b_i / h_i - a_i = 0 \quad \forall i$$

Not invertible so add some extra constraints.

$$\sum_i a_i = \sum_i b_i = 1$$
Depth Distortion

For a line:  \[ a_1 = \frac{h_2 b_1}{b_1 h_2 + h_1 b_2} \]

For a triangle:  \[ a_1 = \frac{h_2 h_3 b_1}{h_2 h_3 b_1 + h_1 h_3 b_2 + h_1 h_2 b_3} \]

Obvious Permutations for other coefficients.