Lighting & Shading  

**Light Sources and Materials**

**Appearance depends on**
- Light sources, locations, properties
- Material (surface) properties
- Viewer position

**Local illumination**
- Compute at material, from light to viewer

**Global illumination (later in course)**
- Ray tracing: from viewer into scene
- Radiosity: between surface patches

**Illumination in the Rendering Pipeline**

**Local illumination**
- Only models light arriving directly from light source
- No interreflections and shadows
  - Can be added through tricks, multiple rendering passes

**Light sources**
- Simple shapes

**Materials**
- Simple, non-physical reflection models

**Light Sources**

**Types of light sources**
- Directional/parallel lights
  - E.g. sun
  - Homogeneous vector
- (Homogeneous) point lights
  - Same intensity in all directions
  - Homogeneous point
- Spot lights
  - Limited set of directions
  - Point+direction+cutoff angle

**Geometry: positions and directions**
- Standard: world coordinate system
  - Effect: lights fixed wrt world geometry
- Alternative: camera coordinate system
  - Effect: lights attached to camera (car headlights)
- Points and directions undergo normal model/view transformation

**Illumination calculations: camera coords**
**Types of Reflection**

- Specular (a.k.a. minor or regular) reflection causes light to propagate without scattering.
- Diffuse reflection sends light in all directions with equal energy.
- Mixed reflection is a weighted combination of specular and diffuse.

**Lambert’s “Law”**

Intuitively, cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.

**Reflectance Distribution Model**

- Most surfaces exhibit complex reflectances
- Vary with incident and reflected directions
- Model with combination
  - specular + glossy + diffuse = reflectance distribution

**Computing Diffuse Reflection**

- Depends on angle of incidence: angle between surface normal and incoming light
  \[ I_{\text{spec}} = k_I I_{\text{inc}} \cos \theta \]
- In practice use vector arithmetic
  \[ I_{\text{spec}} = k_I I_{\text{inc}} (\mathbf{n} \cdot \mathbf{I}) \]
- Always normalizes vectors used in lighting
  - \( \mathbf{n}, \mathbf{I} \) should be unit vectors
- Scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - \( k_I \) diffuse coefficient, surface color
  - \( I_{\text{inc}} \) incoming light intensity
  - \( I_{\text{spec}} \) outgoing light intensity (for diffuse reflection)

**Empirical Approximation**

**Angular falloff**

\[ I_{\text{spec}} = k_s I_{\text{inc}} (\cos \theta)^n \]

- \( k_s \): purely empirical constant, varies with material
- \( I_{\text{spec}} \): specular coefficient, highlight color
- \( n \): no physical basis, works ok in practice

**Phong Lighting**

- Most common lighting model in computer graphics

\[ I_{\text{spec}} = k_s I_{\text{inc}} (\cos \theta)^n \]

- \( k_s \): purely empirical constant, varies with material
- \( I_{\text{spec}} \): specular coefficient, highlight color
- \( n \): no physical basis, works ok in practice
Phong Lighting: The $n_{\text{shiny}}$ Term

- Phong reflectance term drops off with divergence of viewing angle from ideal reflected ray

[Diagram showing viewing angle vs. reflected angle]

**Phong Examples**

- Varying $I$
- Varying $n_{\text{shiny}}$

[Images of spheres with varying properties]

Calculating Phong Lighting

**compute cosine term of Phong lighting with vectors**

$$I_{\text{specular}} = k_s I_{\text{light}} (\mathbf{v} \cdot \mathbf{r})^{n_{\text{shiny}}}$$

- $\mathbf{v}$: unit vector towards viewer/eye
- $\mathbf{r}$: ideal reflection direction (unit vector)
- $k_s$: specular component
  - highlight color
- $I_{\text{light}}$: incoming light intensity

[Diagram showing vectors and calculations]

Lighting in OpenGL

**Light source: amount of RGB light emitted**

- Value represents percentage of full intensity
  - E.g., $I = (1.0, 0.5, 0.5)$
- Every light source emits ambient, diffuse, and specular light

**Materials: amount of RGB light reflected**

- Value represents percentage reflected
  - E.g., $I = (0.0, 1.0, 0.5)$

**Interaction: multiply components**

- Red light $(1.0, 0.0) \times$ green surface $(0.1, 1.0) = \text{black} (0.0, 0.0)$

[Code snippet for OpenGL lighting]

Shading

**CPSC 314**
**Lighting vs. Shading**

**Lighting**
- Process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface

**Shading**
- The process of computing pixel colors

**Applying Illumination**

**Lighting:**
- We now have an illumination model for a point on a surface.
  
  *If surface defined as mesh of polygonal facets, which points should we use?*
  
  - Fairly expensive calculation
  - Several possible answers, each with different implications for visual quality of result

**Applying Illumination**

**Polygonal/triangular models**
- Each facet has a constant surface normal
- If light is directional, diffuse reflectance is constant across the facet.
- Why?

**Flat Shading**

- Simplest approach calculates illumination at a single point for each polygon
  
  - Obviously inaccurate for smooth surfaces

**Flat Shading Approximations**

*If an object really is faceted, is this accurate?*

*No!*

- For point sources, the direction to light varies across the facet
- For specular reflectance, direction to eye varies across the facet
Improving Flat Shading

**What if** evaluate Phong lighting model at each pixel of the polygon?
- Better, but result still clearly faceted

**For smoother-looking surfaces**
  - we introduce **vertex normals at each vertex**
  - Usually different from facet normal
  - Used only for shading
  - Think of as a better approximation of the real surface that the polygons approximate

Vertex Normals

**Vertex normals may be**
- Provided with the model
- Computed from first principles
- Approximated by averaging the normals of the facets that share the vertex

Gouraud Shading

**Most common approach, and what OpenGL does**
- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - Along edges
  - Along scanlines

**Same as Barycentric Coordinates!**

Barycentric Coordinates

**One way to compute them:**

\[
x = \alpha x_1 + \beta x_2 + \gamma x_3 \quad \text{with} \quad \alpha = A_1 / A \quad \beta = A_2 / A \quad \gamma = A_3 / A
\]

Gouraud Shading Artifacts

**Often appears dull, chalky**
**Lacks accurate specular component**
- If included, will be averaged over entire polygon

Convex combination of 3 points

\[
x = \alpha x_1 + \beta x_2 + \gamma x_3
\]

with \( \alpha + \beta + \gamma = 1 \), \( 0 \leq \alpha, \beta, \gamma \leq 1 \)
- \( \alpha, \beta, \gamma \) are called barycentric coordinates

This interior shading missed!
This vertex shading spread over too much area.
**Gouraud Shading Artifacts**

*Mach bands*

- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights

**Perspective transformations**

- Affine combinations only invariant under affine, **not** under perspective transformations
- Thus, perspective projection alters the linear interpolation!

**Phong Shading**

*Linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel*

- Same input as Gouraud shading
- Pro: much smoother results
- Con: considerably more expensive

*Not the same as Phong lighting*

- Common confusion
- Phong lighting: empirical model to calculate illumination at a point on a surface

**Phong Shading**

*Linearly interpolate the vertex normals*

- Compute lighting equations at each pixel
- Can use specular component

\[ I_{\text{pixel}} = k_d I_{\text{ambient}} + \sum_{i=1}^{N_i} I_i (k_l (n \cdot l_i) + k_r (v \cdot r_i)) R_{\text{spec}} \]

\( R_{\text{spec}} \) remember: normals used in diffuse and specular terms

Discontinuity in normal's rate of change harder to detect
Phong Shading Difficulties

- Computationally expensive
  - Per-pixel vector normalization and lighting computation!
  - Floating point operations required
- Lighting after perspective projection
  - Messes up the angles between vectors
  - Have to keep eye-space vectors around
- No direct support in hardware
  - But can be simulated with texture mapping

Shading Artifacts: Silhouettes

- Polygonal silhouettes remain

Shading Artifacts: Orientation

- Interpolation dependent on polygon orientation
  - View dependence!
  
Shading Artifacts: Shared Vertices

- Vertex B shared by two rectangles on the right, but not by the one on the left
  - First portion of the scanline is interpolated between DE and AC
  - Second portion of the scanline is interpolated between BC and GH
  - A large discontinuity could arise

Shading Models Summary

- Flat shading
  - Compute Phong lighting once for entire polygon
- Gouraud shading
  - Compute Phong lighting at the vertices and interpolate lighting values across polygon
- Phong shading
  - Compute averaged vertex normals
  - Interpolate normals across polygon and perform Phong lighting across polygon

The Rendering Pipeline

- Geometry Database
- Model/View Transform.
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Frame-buffer
  - Rasterization
  - Fragment Processing
Coming Up

**Tuesday:**
- Shading (2)
- Quiz 1

**Thursday:**
- Clipping
- A1 due