Lighting and Shading

Slides: Tamar Shinar, Victor Zordon
Why we need shading

• Suppose we build a model of a sphere using many polygons and color each the same color. We get something like

• But we want
Why does the image of a real sphere look like

Light-material interactions cause each point to have a different color or shade

Need to consider
- Light sources
- Material properties
- Location of viewer
- Surface orientation (normal)
General rendering

- The most general approach is based on physics - using principles such as conservation of energy
- A surface either emits light (e.g., light bulb) or reflects light from other illumination sources, or both
- Light interaction with materials is recursive
- The rendering equation is an integral equation describing the limit of this recursive process
Fast local shading models

• the rendering equation can’t be solved analytically

• numerical methods aren’t fast enough for real-time

• for our fast graphics rendering pipeline, we’ll use a local model where shade at a point is independent of other surfaces

• use Phong reflection model

  • shading based on local light-material interactions
Local shading model

[Angel and Shreiner]
Global Effects

shadow

multiple reflection

translucent surface

[Angel and Shreiner]
Light-material interactions

at a surface, light is absorbed, reflected, or transmitted

specular  diffuse  translucent
General light source

Illumination function:

\[ l(x, y, z, \theta, \phi, \lambda) \]

Integrate contributions from all sources to shade the point

[Angel and Shreiner]
Idealized light sources

- Ambient light
- Point light
- Spotlight
- distant (directional) light

luminance: \[ L = \begin{bmatrix} L_r \\ L_g \\ L_b \end{bmatrix} \]
Ambient light source

- achieve a uniform light level
- no black shadows
- ambient light intensity at each point in the scene

\[
L_a = \begin{bmatrix}
L_{ar} \\
L_{ag} \\
L_{ab}
\end{bmatrix} 
\]
Point light source

\[ \mathbf{L}(\mathbf{p}_0) = \begin{bmatrix} L_r(\mathbf{p}_0) \\ L_g(\mathbf{p}_0) \\ L_b(\mathbf{p}_0) \end{bmatrix} \]

illumination intensity at \( \mathbf{p} \):

\[ l(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|^2} \mathbf{L}(\mathbf{p}_0) \]
Point light source

Most real-world scenes have large light sources

Point light sources alone aren't too realistic - add ambient light to mitigate high contrast

[Angel and Shreiner]
Point light source

Most real-world scenes have large light sources

Point light sources alone aren’t too realistic
- drop off intensity more slowly

\[ l(p, p_0) = \frac{1}{d^2} L(p_0) \]

\[ l(p, p_0) = \frac{1}{a + bd + cd^2} L(p_0) \]
Spotlights

Intensity

\[ \cos(\phi) \]
Spotlights

\[ \cos^e(\phi) \cdot l(p, p_s) \]
Distant light source characterized by direction
Lambertian Reflection Model
Lambertian Reflection Model

\[ I \propto \cos \theta \]

- **direct**: maximum light intensity
- **indirect**: reduced light intensity

Lambert's cosine law
Lambertian Reflection Model

\[ I \propto n \cdot l \]

color intensity

Lambert's cosine law

**direct**: maximum light intensity

**indirect**: reduced light intensity
Lambertian Reflection Model

\[ I \propto R \cos \theta \]

**color intensity**  **reflectance**

**Lambert’s cosine law**

**direct**: maximum light intensity  **indirect**: reduced light intensity
Lambertian Reflection Model

Lambert’s cosine law

**direct**: maximum light intensity
**indirect**: reduced light intensity

\[ I = LRn \cdot l \]

- illumination
- color intensity
- reflectance
Lambertian Reflection Model

\[ I = LR \max(0, n \cdot l) \]
Lambertian Reflection Model

\[ I = LR|\mathbf{n} \cdot \mathbf{l}| \]

two-sided lighting
Ambient Reflection

\[ I = LR \max(0, \mathbf{n} \cdot \mathbf{l}) \]

Surfaces facing away from the light will be totally **black**
Ambient Reflection

\[ I = L_a R_a + L_d R_d \max(0, n \cdot l) \]

All surfaces get same amount of ambient light
Phong Reflection Model
Phong Reflection Model

- efficient, reasonably realistic
- 3 components
- 4 vectors
Phong Reflection Model

\[ I = I_a + I_d + I_s \]
\[ = R_a L_a + R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n}) + R_s L_s \max(0, \cos \phi)^\alpha \]

- Color intensity
- Reflectance
- Illumination
Ambient reflection

\[ I_a = R_a L_a, \quad 0 \leq R_a \leq 1 \]

Different ambient coefficients for different colors
Diffuse reflection

Ambient + Diffuse + Specular = Phong Reflection
Diffuse reflection

\[ I_d = R_d L_d \max(0, 1 \cdot n) \]

diffuse reflection coefficient

Lambert’s cosine law

direct: maximum light intensity

indirect: reduced light intensity
Specular reflection

Ideal reflector

$$\theta_i = \theta_r$$

\( r \) is the mirror reflection direction
Specular reflection is strongest in mirror reflection direction.
Specular reflection

specular reflection drops off with increasing angle $\phi$

$I_s = R_s L_s \cos^\alpha \phi$

specular reflection coefficient

Phong exponent
Specular reflection

\[ I_S = R_S L_S \max(0, \cos \phi)^\alpha \]

\( \alpha = 5 \ldots 10 \) plastic

\( \alpha = 100 \ldots 200 \) metal
Phong Reflection Model

\[ I = I_a + I_d + I_s \]
\[ = R_a L_a + R_d L_d \max(0, \mathbf{l} \cdot \mathbf{n}) + R_s L_s \max(0, \mathbf{v} \cdot \mathbf{r})^\alpha \]

Ambient  +  Diffuse  +  Specular  =  Phong Reflection
Alternative: Blinn-Phong Model

Halfway vector

$$h = \frac{l + v}{|l + v|}$$

$$I = I_a + I_d + I_s$$

$$= R_a L_a + R_d L_d \max(0, l \cdot n) + R_s L_s \max(0, h \cdot n)^\alpha$$

Ambient  Diffuse  Specular
Shading Polygonal Geometry
Smooth surfaces are often approximated by polygons

Shading approaches:

1. Flat
2. Smooth (Gouraud)
3. Phong
Flat Shading

do the shading calculation once per polygon

valid for light at $\infty$ and viewer at $\infty$ and faceted surfaces
Mach Band Effect
Smooth Shading

\[ \mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{||\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4||} \]

do the shading calculation once per \textit{vertex}
Interpolating Normals

- Must renormalize

\[ n_0 \quad \longrightarrow \quad n_1 \]
Interpolating Normals

- Must renormalize
Interpolating Normals

- Must renormalize
We can interpolate attributes using barycentric coordinates

\[ c = \alpha c_0 + \beta c_1 + \gamma c_2 \]

Gouraud shading

(Gouraud, 1971)

http://jtibble.dyndns.org/graphics/eecs487/eecs487.html
Phong Shading

do the shading calculation once per fragment
Comparison

Flat  Gouraud  Phong
Problems with Interpolated Shading

- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals

[Foley, van Dam, Feiner, Hughes]
Programmable Shading
Fixed-Function Pipeline

User Program → Geometry Processing → Pixel Processing

CPU → primitives → GPU

2D screen coordinates

Control pipeline through GL state variables
Supply shader programs to be executed on GPU as part of pipeline
Phong reflectance in vertex and pixel shaders using GLSL

Vertex Shader (Gouraud interpolation)

Pixel Shader (Phong interpolation)
Dawn, NVIDIA

Rusty car shader, NVIDIA

Call of Juarez DX10 Benchmark, ATI

Dawn, NVIDIA
Computing Normal Vectors
Plane Normals

\[ \mathbf{v} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0) \]

\[ \mathbf{n} = \frac{\mathbf{v}}{||\mathbf{v}||} \]
Implicit function normals

\[ f(p) = 0 \]

\[ \nabla f(p) \]

\[ \nabla f = \left( \begin{array}{c} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{array} \right) \]

sphere

\[ p \cdot p - r^2 = 0 \]

plane

\[ n \cdot (p - p_0) = 0 \]
Parametric form

\[ \mathbf{p}(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} \]

tangent vectors

\[ \frac{\partial \mathbf{p}}{\partial u}, \quad \frac{\partial \mathbf{p}}{\partial v} \]

normal

\[ \frac{\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}}{|| \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} ||} \]