CS130 : Computer Graphics
Lighting and Shading

Tamar Shinar
Computer Science & Engineering
UC Riverside
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
Shading

• 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

direct light

reflected light

[COP]
Global Effects

- Shadow
- Multiple reflection
- Translucent surface
Light-material interactions

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

specular  diffuse  translucent
Idealized light sources

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

luminance: \( \mathbf{L} = \begin{bmatrix} L_r \\ L_g \\ L_b \end{bmatrix} \)
Ambient light source

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

$$L_a = \begin{bmatrix} L_{ar} \\ L_{ag} \\ L_{ab} \end{bmatrix}$$

ambient light is the same everywhere but different surfaces will reflect it differently
Point light source

\[ L(p_0) = \begin{bmatrix} L_r(p_0) \\ L_g(p_0) \\ L_b(p_0) \end{bmatrix} \]

illumination intensity at \( p \):

\[ l(p, p_0) = \frac{1}{|p - p_0|^2} L(p_0) \]
Point light source

Most real-world scenes have large light sources

Point light sources alone not realistic
- add ambient light to mitigate high contrast

[Angel and Shreiner]
Most real-world scenes have large light sources

Point light sources alone not 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

\[ \text{Intensity} \]

\[ \cos(\phi) \]

\[ p_s \]

\[ p \]
Spotlights

\[ \cos^e(\phi) \cdot l(p, p_s) \]
Distant light source

characterized by direction
Lambertian Reflection Model
Lambertian Reflection Model

\[ I \propto \cos \theta \]

color intensity

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

\[ I \propto \mathbf{n} \cdot \mathbf{l} \]

color intensity
Lambertian Reflection Model

$I \propto R \cdot n \cdot l$

- **color intensity**
- **reflectance**

**Lambert’s cosine law**

**direct**: maximum light intensity

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

\[ I = LRn \cdot l \]

**illumination**

**direct:** maximum light intensity

**indirect:** reduced light intensity

**color intensity**

**reflectance**

Lambert’s cosine law
Lambertian Reflection Model

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

face points away from the light
Lambertian Reflection Model

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

two-sided lighting
Adding Ambient Reflection

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

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

http://www.andreafisherpottery.com/cgi-bin/artistlnk.cgi?Jody_Folwell
Phong Reflection Model

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

\[ I = I_a + I_d + I_s \]

\[ = R_a L_a + R_d L_d \max(0, 1 \cdot n) + R_s L_s \max(0, \cos \phi) \alpha \]

- **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

ambient reflection coefficient
Diffuse reflection

Ambient + Diffuse + Specular = Phong Reflection
Diffuse reflection

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

**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

$I_s = R_s L_s \max(0, \cos \phi)^\alpha$

\( \alpha = 5..10 \) plastic
\( \alpha = 100..200 \) metal

Phong exponent
Phong Reflection Model

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

Ambient  Diffuse  Specular
Alternative: Blinn-Phong Model

\[
h = \frac{1 + v}{|1 + v|}
\]

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

Ambient  Diffuse  Specular
$\alpha$

10: eggshell
100: shiny
1000: glossy
10000: mirror-like
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

lateral inhibition effect
do the shading calculation once per **vertex**

\[ n = \frac{n_1 + n_2 + n_3 + n_4}{\| n_1 + n_2 + n_3 + n_4 \|} \]
Interpolating Normals

• Must renormalize
Interpolating Normals

• Must renormalize
Interpolating Normals

- Must renormalize

\[ n_0 \quad n_1 \]
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
do the shading calculation once per fragment
Comparison

Flat  Gouraud  Phong
Problems with Interpolated Shading

- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals
Programmable Shading
Fixed-Function Pipeline

User Program \rightarrow \text{Geometry Processing} \rightarrow \text{Pixel Processing}

- Control pipeline through GL state variables
Programmable Pipeline

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

```cpp
void main(void)
{
  vec4 v = gl_modelViewMatrix * gl_Vertex;
  vec3 n = normalize(gl_NormalMatrix * gl_Normal);
  vec3 l = normalize(gl_lightSource[0].position - v);
  vec3 h = normalize(l - normalize(v));

  float p = 16;
  vec4 cr = gl_FrontMaterial.diffuse;
  vec4 cl = fl.LightSource[0].diffuse;
  vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

  vec4 color;
  if (dot(h, n) > 0)
    color = cr * (ca + cl * max(0, dot(n, l))) + cl * pow(dot(h, n), p);
  else
    color = cr * (ca + cl * max(0, dot(n, l)));

  gl_FrontColor = color;
  gl_Position = ftransform();
}
```

Vertex Shader (Gouraud interpolation)

```cpp
varying vec4 v;
varying vec3 n;

void main(void)
{
  vec3 l = normalize(gl_lightSource[0].position - v);
  vec3 h = normalize(l - normalize(v));

  float p = 16;
  vec4 cr = gl_FrontMaterial.diffuse;
  vec4 cl = fl.LightSource[0].diffuse;
  vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

  vec4 color;
  if (dot(h, n) > 0)
    color = cr * (ca + cl * max(0, dot(n, l))) + cl * pow(dot(h, n), p);
  else
    color = cr * (ca + cl * max(0, dot(n, l)));

  gl_FragColor = color;
}
```

Pixel Shader (Phong interpolation)
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

\[ f(p) = 0 \]

\[ \nabla f(p) \]

\[ \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} \]

sphere

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

plane

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

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

**tangent vectors**
\[ \frac{\partial p}{\partial u}, \frac{\partial p}{\partial v} \]

**normal**
\[ \frac{\frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v}}{\left\| \frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v} \right\|} \]