Materials

From Apodaca and Gritz, *Advanced RenderMan*
Lighting

Mies Courtyard House with Curved Elements

Modeling: Stephen Duck; Rendering: Henrik Wann Jensen

Modeling & Simulating Appearance

Reflection models
   Mirror
   Diffuse
   Shiny
Materials
Shading models in OpenGL
Environment mapping
Displacement and normal mapping
Reflection

Reflection from a Surface

Mirror
- Ideal reflection
- Reflection Law

Diffuse
- Matte
- Lambert’s Law

Specular
- Highlights and gloss
- Microfacet model
Mirror: Law of Reflection

\[
\hat{R} + (-\hat{I}) = 2\cos \theta, \hat{N} = -2(\hat{I} \cdot \hat{N})\hat{N}
\]

\[
\hat{R} = \hat{I} - 2(\hat{I} \cdot \hat{N})\hat{N}
\]

Diffuse: Lambert’s Law

Lights only illuminate one side of the surface

Thus the max(0, N.L)
**Specular: Microfacets**

Rough Surface

\[ \mathbf{H} = \frac{\mathbf{L} + \mathbf{E}}{|\mathbf{L} + \mathbf{E}|} \]

Halfway vector

\[ \cos \theta_H = \mathbf{N} \cdot \mathbf{H} \]

Microfacet distribution

\[ (\cos \theta_H)^s = (\mathbf{N} \cdot \mathbf{H})^s \]

Shininess \( s \)

The larger the value \( s \), the narrower the highlight

<table>
<thead>
<tr>
<th>( s = 1 )</th>
<th>( s = 64 )</th>
</tr>
</thead>
</table>

\[ \cos^s \theta \]

CS148 Lecture 16  
Pat Hanrahan, Fall 2011
Simple Materials & Lighting in OpenGL

OpenGL Material Properties

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient</td>
<td>Diffuse</td>
<td>Specular</td>
<td>Final Image</td>
</tr>
<tr>
<td>(Ka)</td>
<td>(Kd)</td>
<td>(Ks, s)</td>
<td></td>
</tr>
</tbody>
</table>

CS148 Lecture 16
Pat Hanrahan, Fall 2011
Setup the Lighting …

```c
//Enable lighting
glEnable(GL_LIGHTING);

//Define the light position
GLfloat light0_position[] = { 1, 1, 1, 0 };

//Define the light diffuse color
GLfloat light0_diffuse[] = { 1, 1, 1, 1 };
GLfloat light0_specular[] = { 1, 1, 1, 1 };

//Set the light position
glLightfv( GL_LIGHT0, GL_POSITION, light0_position );

//Set the light diffuse color
glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );

//Set the light specular color
glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );

//Enable the light
glEnable(GL_LIGHT0);
```

Setup the Material Properties …

```c
//Define material properties
GLfloat mat_ambient[] = { 1, .1, .8, 1 };
GLfloat mat_diffuse[] = { .2f, .2f, .6f, 1 };
GLfloat mat_specular[] = { .8f, .8f, .8f, 1 };
GLfloat mat_shininess[] = { 128 };

//Set material properties
glMaterialfv( GL_FRONT, GL_AMBIENT, mat_ambient );

//Set the diffuse color
glMaterialfv( GL_FRONT, GL_DIFFUSE, mat_diffuse );

//Set the specular color
glMaterialfv( GL_FRONT, GL_SPECULAR, mat_specular );

//Set shininess
glMaterialfv( GL_FRONT, GL_SHININESS, mat_shininess );
```
Changing the Material Properties

```c
// Set up

// when you want to draw something with a new diffuse AND ambient color
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
gColor4f(0.0, 0.1, 1.0);
drawSphere();

// now we just change the diffuse to red
glColorMaterial(GL_FRONT, GL_DIFFUSE);
gColor4f(1.0, 0.0, 0.1);
drawSphere();
```
Normals
Surface Tangents and Surface Normal

\[ \mathbf{N} = \mathbf{T}_1 \times \mathbf{T}_2 \]

Normal at a Point on a Sphere

\[ \mathbf{P} = (x, y, z) \]

\[ x = \sin \theta \cos \phi \]
\[ y = \sin \theta \sin \phi \]
\[ z = \cos \theta \]

\[ \frac{\partial \mathbf{P}}{\partial \theta} = (\cos \theta \cos \phi, \cos \theta \sin \phi, -\sin \theta) \]
\[ \frac{\partial \mathbf{P}}{\partial \phi} = (-\sin \theta \sin \phi, \sin \theta \cos \phi, 0) \]
Normal for a Triangle

Find 2 Tangent Vectors

\[ T_1 = P_1 - P_0 \]
\[ T_2 = P_2 - P_0 \]
Cross-Product Perpendicular to Both Ts

\[ \mathbf{N} = \mathbf{T}_1 \times \mathbf{T}_2 \]

3 Points Define a Plane : Unique Normal

\[ \mathbf{N} \]

\[ \mathbf{P}_0 \rightarrow \mathbf{P}_1 \rightarrow \mathbf{P}_2 \]
Vertex Normal?

Average of Face Normals

Sometimes weight triangle normals by angle, area, etc.
OpenGL: Must Specify Normals

```gl
glBegin(GL_TRIANGLES);
glNormal(1.f,1.f,1.f);
glVertex3f(0.f,0.f,1.f);
glNormal(1.f,1.f,1.f);
glVertex3f(1.f,0.f,1.f);
glNormal(1.f,1.f,1.f);
glVertex3f(1.f,1.f,1.f);
glEnd();
```

N.B. Normal may be different at each vertex

OpenGL Coordinate Systems

```
ModelView
\downarrow
Object or Model
\downarrow
World
\downarrow
View or Camera
\downarrow
Window (2D)
```

- `glRotate`
- `glTranslate`
- `gluLookat`
- `glPerspective`
Transformations in OpenGL

```c
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glFrustum(-1., 1., -1., 1., -1., 1.);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookat(from, to, up);

glTranslate(...);
glRotate(...);
```

Physics of Shading

Physical laws depend on distances, lengths, and angles; e.g. Law of Diffuse Reflection

\[ E = \max(0, \hat{N} \cdot \hat{L}) \]

Unit vectors, angle between vectors

 Transforms may change lengths and angles
  - Rigid body (isometric) transforms preserve
  - Perspective changes both distance and angles
Recall: Perspective Frustum

$$\text{glFrustum}(l, r, b, t, n, f)$$

Effect of Perspective Transform

Perspective transform maps the viewing pyramid to a unit cube

Doesn't preserve angles and distances!
Therefore, shading calculations must be done before the perspective transformation
Transforming Normals

Idea is to preserve dot products $\hat{N} \cdot \hat{V}$

To preserve dot product, insert identity matrix

$$N \cdot V = N^T I V = N^T M^{-1} M V = N' \cdot V'$$

Where, $V' = M \cdot V$ and $N' = M^{-T} N$

$M^{-T}$ is the inverse transpose

---

GLSL Lighting Calculation

```
vec3 N, L, V, H; float NdotL, NdotH;

vec4 Cd = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
vec4 Ca = gl_FrontMaterial.ambient * gl_LightSource[0].ambient;
vec4 Cs = vec4(0.0);

P = gl_ModelViewMatrix * gl_Vertex
N = normalize(gl_NormalMatrix * gl_Normal);
L = normalize(vec3(gl_LightSource[0].position - P));
NdotL = max(dot(N, L), 0.0);
if (NdotL > 0.0) {
   Cs = gl_FrontMaterial.specular * gl_LightSource[0].specular;
   H = normalize( gl_LightSource[0].halfVector.xyz );
   NdotH = max(dot(N, H), 0.0);
   Cs *= pow(NdotH, gl_FrontMaterial.shininess);
}
gl_FrontColor = Ca + NdotL * Cd + Cs;
```
Flat Shading

1. Shade once per triangle
2. Same color everywhere inside the triangle

Gouraud Shading – Smooth Shading

1. Shade at the vertices
2. Interpolate the resulting colors
Phong Shading – Higher Shading Rate

1. Interpolate the vertex normals
2. Shade each fragment

Environment Maps
Gazing Ball (Light Probe)

Miller and Hoffman, 1984

Gazing Ball (Light Probe)

Miller and Hoffman, 1984

Photograph of mirror ball
Environment reflected in the ball

$\mathbf{N} = (n_x, n_y, \sqrt{1 - n_x^2 + n_y^2})$

Sideways view

$n_x = 2(u - 1)$

$n_y = 2(v - 1)$
Gazing Ball (Light Probe)

Miller and Hoffman, 1984

- To shade, index the ball texture map using (nx, ny)

Environment Maps

*Interface, Chou and Williams (ca. 1985)*
Displacement Maps

Displacement/Bump Mapping

\[ P(u, v) \]
\[ S(u, v) = \frac{\partial P(u, v)}{\partial u} \quad T(u, v) = \frac{\partial P(u, v)}{\partial v} \]
\[ N(u, v) = S \times T \]

- **Displacement**
  \[ P'(u, v) = P(u, v) + h(u, v)N(u, v) \]

- **Perturbed normal**
  \[ N'(u, v) = P'_u \times P'_v \]
  \[ = N + h_u(T \times N) + h_v(S \times N) \]

From Blinn 1976
Normal Maps

(nx, ny, nz) = (r, g, b)

http://members.shaw.ca/jimht03/normal.html

Translucent Materials
Opaque vs. Translucent

Subsurface Scattering
Skin: Subsurface

Modeled by Stephen Stahlberg

CS148 Lecture 16
Pat Hanrahan, Fall 2011

Skin: Surface Only

Modeled by Stephen Stahlberg

CS148 Lecture 16
Pat Hanrahan, Fall 2011
Would you Drink this Milk?

Skim Milk
Whole Milk

Diffuse Milk – Paint?
Things to Remember

Basic reflection models
- Mirror reflection
  - Angle of incidence equals angle of reflection
- Diffuse reflection
  - Reflection proportional to the energy falling on the surface
- Specular reflection
  - Microfacet models

Materials: Combination of ambient, diffuse and specular

OpenGL
- Why GL_PROJECTION and GL_MODELVIEW
- Computing and transforming normals

Environment maps
Displacement and normal maps