

EECE 478

Lighting, Materials
and Texture

Learning Objectives

Lighting
- Describe the simple lighting model
- Define point light sources
- Create a lighting structure for a scene

Materials
- Define surface normals for objects
- Model surface materials

Learning Objectives

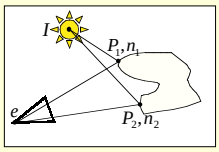
Textures
- Describe how texture maps relate to surface appearance
- Load texture maps into memory
- Define texture coordinates for a surface
- Give reasons to mipmap textures
- Define a mipmap for a texture

Vocabulary

- light source
- specular
- diffuse
- translucent
- illumination function
- ambient light
- spotlight
- lambertian surface
- Phong model
- angle of incidence
- angle of reflection
- flat shading
- Gouraud shading
- texture map
- mipmap

Scene Lighting

- Light source (I)
- Incident points (P_1, P_2)
- Surface normals (n_1, n_2)
- Eye point (e)



Light Sources

Illumination function

- Object surface emits light
- Every surface point emits light in certain intensities dependent on direction

$$I(x,y,z,\theta,\phi,\lambda)$$

- Consider all rays from light source ($\mathbf{p}_0$) to an incident point ($\mathbf{p}_i$) then on to eye (e)
- RGB source: $[I_r, I_g, I_b]$

Ambient Light

Inside rooms or outdoors

- Acts as if constant light coming from all directions
- *Ambient* means *all around*

$$\mathbf{I}_a = \begin{bmatrix} I_{ar} & I_{ag} & I_{ab} \end{bmatrix}^T$$

Point Source

Ideal *point source*

- surface is single point
- light emitted equally in all directions
- intensity reduced by inverse-square law

$$\mathbf{I}(\mathbf{p}_i, \mathbf{p}_0) = \frac{1}{|\mathbf{p}_i - \mathbf{p}_0|^2} \mathbf{I}(\mathbf{p}_0)$$

Distant Source

Ideal *distant source*

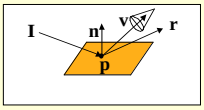
- incident rays are parallel
  - use vector as source point ($\mathbf{p}_0$)
- intensity is constant

$$\mathbf{p}_0 = \begin{bmatrix} x & y & z & 0 \end{bmatrix}^T$$
$$\mathbf{I}(\mathbf{p}_i, \mathbf{p}_0) = \mathbf{I}(\mathbf{p}_0)$$

Phong Model

Color at point **p** is determined by vectors

- **I** – from light source
- **n** – surface normal
- **v** – direction to eye point *e*
- **r** – reflection



Phong Model(2)

Color at point **p** is combination of three components

- *ambient* – from ambient light
- *diffuse* – due to scattered light
- *specular* – due to reflected light

$$I = I_a + I_d + I_s$$
$$= L_a R_a + L_d R_d + L_s R_s$$

Phong: Ambient

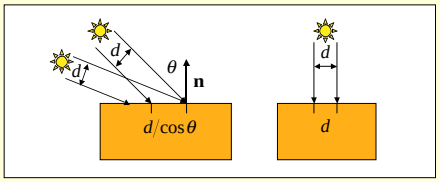
- Intensity constant over surface
- Reflected light is fraction of incident light

$$R_a = k_a$$
$$0 \le k_a \le 1$$
$$I_a = L_a k_a$$

```
GLfloat a[] = {
  1.0, 0.5, 0.5, 1.0
};
glMaterialfv (GL_FRONT,
  GL_AMBIENT, a);
```

Phong: Diffuse(1)

- Based on *Lambertian* surface model
- Reflected light emitted in all directions



Phong: Diffuse(2)

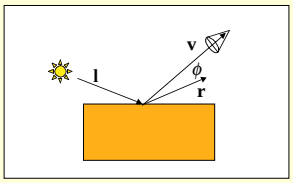
- Intensity proportional to $\cos\theta$
- Cosine derived from dot product
  - normalize **l** and **n**

$$R_d = k_d \cos\theta$$
$$= k_d(\mathbf{l}\cdot\mathbf{n})$$
$$I_d = L_d k_d(\mathbf{l}\cdot\mathbf{n})$$

```
GLfloat d[ ] = {
   1.0, 0.5, 0.5, 1.0
};
glMaterialfv (GL_FRONT,
   GL_DIFFUSE, d);
```

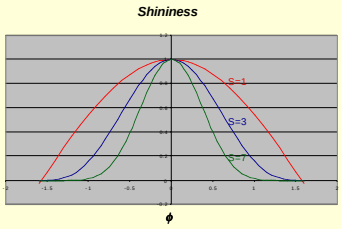

Phong: Specular(1)

- Based on *Phong* surface model
- Reflected light emitted by reflection



Phong: Shininess

Shininess specified by α in to $(\cos\phi)^{\alpha}$



Phong: Specular(2)

Shininess such that

- Larger values are shinier surfaces
- Values >500 are metallic
- Values >100 are “shiny” (e.g. plastic)

$$R_s = k_s \cos^n \phi$$
$$= k_s(\mathbf{v}\cdot\mathbf{r})^{\alpha}$$
$$I_s = L_s k_s(\mathbf{v}\cdot\mathbf{r})^{\alpha}$$

```
GLfloat s[] = {
  1.0, 0.5, 0.5, 1.0
};
glMaterialfv (GL_FRONT,
  GL_SPECULAR, s);
```


Phong: Complete

Total intensity of surface point

- *ambient + diffuse + specular*
- optionally attenuate by distance (I_d,I_s)

$$I = I_a + I_d + I_s$$
$$= L_a k_a + \mathrm{A}(d)\left(L_d k_d(\mathbf{l}\cdot\mathbf{n}) + L_s k_s(\mathbf{v}\cdot\mathbf{r})^{\alpha}\right)$$
$$= L_a k_a + \frac{1}{a+bd+cd^2}\left(L_d k_d(\mathbf{l}\cdot\mathbf{n}) + L_s k_s(\mathbf{v}\cdot\mathbf{r})^{\alpha}\right)$$

Phong: Vectors

Four vectors determine shading

- Light vector ($\mathbf{l}$) *given*
- View vector ($\mathbf{v}$) *given*
- Normal vector ($\mathbf{n}$) *computed*
- Reflection vector ($\mathbf{r}$) *computed*

Phong: Normals

Normal to surface (*collection of triangles*)

- Consider surface as plane
- Compute normal of triangle ($\mathbf{p}_1$,$\mathbf{p}_2$,$\mathbf{p}_3$)

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$

Given three non-collinear points

$$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$$

Phong: Reflection

Light ray reflected off surface

- angle of incidence = angle of reflection
- reflected *about* normal vector
- $\mathbf{l}$, $\mathbf{n}$, $\mathbf{r}$ all in same plane

$$\mathbf{l} \cdot \mathbf{n} = \mathbf{n} \cdot \mathbf{r}$$

$$\mathbf{r} = \alpha\mathbf{l} + \beta\mathbf{n}$$

assuming all are unit vectors

$$\mathbf{r} = 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}$$

OpenGL: Normals

Normal specified as *attribute (state)*

- applies to *all* following vertices
- user responsible for normalizing

```
/* Flat shaded triangle. */
glShadeModel (GL_FLAT);
glBegin (GL_TRIANGLES);
  glNormal3f (0.577, -0.577, 0.577);
  glVertex3f (1.0, 0.0, 0.0);
  glVertex3f (0.0, 1.0, 0.0);
  glVertex3f (0.0, 0.0, 1.0);
glEnd ();
```


OpenGL: Normals

Per-vertex normals are interpolated

- *Gouraud* shading

```
/* Smooth shaded triangle. */
glShadeModel (GL_SMOOTH);
glBegin (GL_TRIANGLES);
  glNormal3f (0.0, 0.707, 0.707);
  glVertex3f (1.0, 0.0, 0.0);
  glNormal3f (0.707, 0.0, 0.707);
  glVertex3f (0.0, 1.0, 0.0);
  glNormal3f (0.707, 0.707, 0.0);
  glVertex3f (0.0, 0.0, 1.0);
glEnd ();
```


OpenGL: Light Source

Values of L_x in Phong model

- Small number of distinct sources
- Indexed by `GL_LIGHT`*n*
- `glEnable` turns on

```
/* Define simple point light source. */
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
glLightfv (GL_LIGHT0, GL_POSITION, p0);
glLightfv (GL_LIGHT0, GL_AMBIENT, a0);
glLightfv (GL_LIGHT0, GL_DIFFUSE, d0);
glLightfv (GL_LIGHT0, GL_SPECULAR, s0);
```

OpenGL: Material

Values of R_x in Phong model
- Can specify front, back or both
- Emission is *object as light source*

```
/* Define Phong material. */
glMaterialfv  (GL_FRONT, GL_EMISSION, e);
glMaterialfv  (GL_FRONT, GL_AMBIENT, a);
glMaterialfv  (GL_FRONT, GL_DIFFUSE, d);
glMaterialfv  (GL_FRONT, GL_SPECULAR, s);
glMaterialf   (GL_FRONT, GL_SHININESS, sh)
```


Pipeline Buffers

- Color buffers
  - front and back for animation
  - left and right for stereo
- Depth buffer
  - hidden surface removal
- Accumulation buffer
  - multi-pass methods
- Stencil buffer
  - stencil screening

Surface Texture

Can vary color/material on vertex-by-vertex basis using `glColor` or `glMaterial`
- Maximum resolution is # of polygons
- Expensive to change

Alternative: Define *texture* image
- Maximum resolution is image resolution
- Inexpensive with hardware support

Texture Coordinates

Need way to define connection between:
- Points on image (2D)
- Vertices on surface of object

Define *texture coordinates* on texture image
- s is horizontal coordinate of image [0,1]
- t is vertical coordinate of image [0,1]

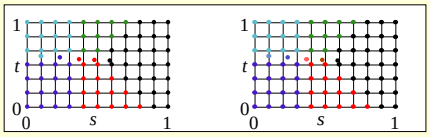
Texture Maps

1. Enable texture
2. Define texture image
3. For each vertex of object:
   a. Set texture coordinates
   b. Define other vertex properties
   c. Bind vertex with `glVertex*`

Texture image is wrapped onto object

Effect of Texture Map

Texture is interpolated into polygon
- Texture coords interpolated bilinearly
- Inter-pixel values may be interpolated by `GL_NEAREST` or `GL_LINEAR`



Texture & Shading

May have both texture map and shading
- Must define combination
- Texture determines surface color

Decal: Unaffected by shading

Modulate: Filtered by shading equations

Use `glTexEnv` to change mode

OpenGL: Texture Definition

`glTexImage2D` copies image from program memory to *texture memory*
- define contents of current texture

Parameters for:
- image size and format
- mipmap level and border properties
- location of image data

OpenGL: glTexImage2D

Details:
- Components defines how texture will be stored in texture memory
- Image format may be converted

```
/* Define texture image in tarray */
glTexImage2D (GL_TEXTURE_2D, 0, //level 0
 3,                 // components
 512, 512, 0,       // size and border width
 GL_RGB,            // tarray is RGB
 GL_UNSIGNED_BYTE, // with 0-255 bytes
 tarray);           // memory image is here
```

OpenGL: Variations

In addition to `glTexImage2D` we have:

- `glTexSubImage2D`: Define part of larger texture
- `glTexImage1D`: Define 1D texture (t coordinate is 0)
- `glTexImage3D`: For volumetric texture
- `glCopyTexSubImage2D`: Copy memory from buffer into texture image

Texture Objects

Keep many textures in texture memory

- Define each texture once
- Easily change which texture being mapped

Texture object is *name* of defined texture

- `glGenTextures` creates new *names* (ints)
- `glBindTexture` make name *current*

Texel Magnification

Texture $(s,t) \Rightarrow$ 3D vertex $\Rightarrow$ screen

- Texture pixel (*texel*) becomes quadrilateral on screen
- May be *magnified* or *minified* if it is >1 pixel or <1 pixel on screen
- Need to define linear or nearest for both

Texture Foreshortening

- Texture mapped object may be at range of distances
  - Magnification and minifaction varies across image or with distance
  - Minification may be expensive
  - Minification may also cause moiré effects

Solved by *mipmapping*

Mipmaps

Mipmap = Texture pyramid
- Level 0 is base texture image
- Level n+1 is 2x2 downsample of level n
- Fill pyramid down to level with 1 pixel

Interpolate in 3D into pyramid
- Choose level where texel size ≈ screen pixel
- Can use nearest or linear interpolation