Computer Graphics

(Shading: adding light to the scene)

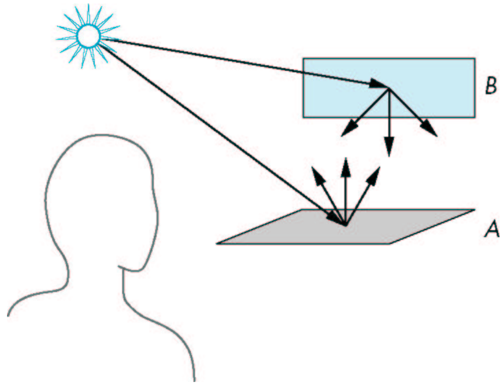
Thilo Kielmann
Fall 2003
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

http://www.cs.vu.nl/˜graphics/

Outline for today

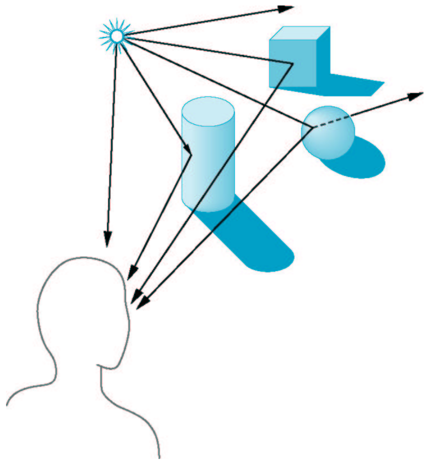
- Light and Matter
- Ray Tracing / Radiosity
- The Phong Reflection Model
- Polygonal Shading
- Light and Matter in OpenGL
- Shading a Sphere

Light and Matter

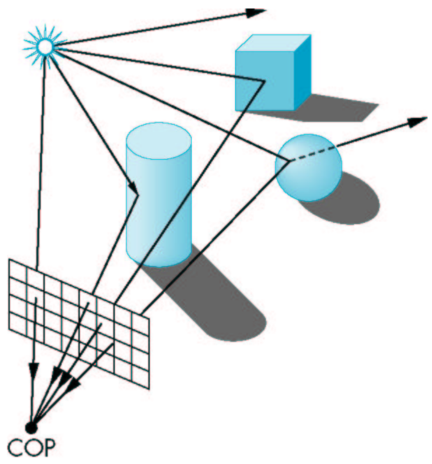


Rendering equation can not be solved in practice.

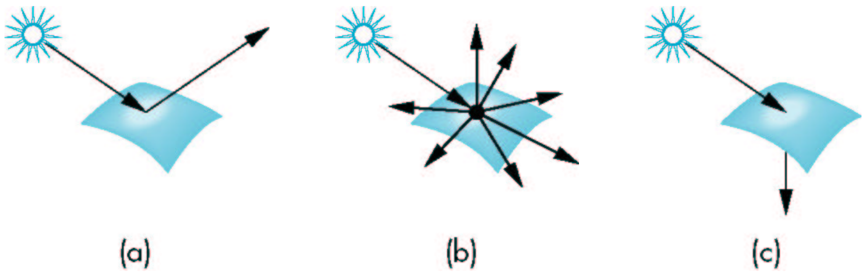
Light and Surfaces



Light, Surfaces, and Computer Imaging

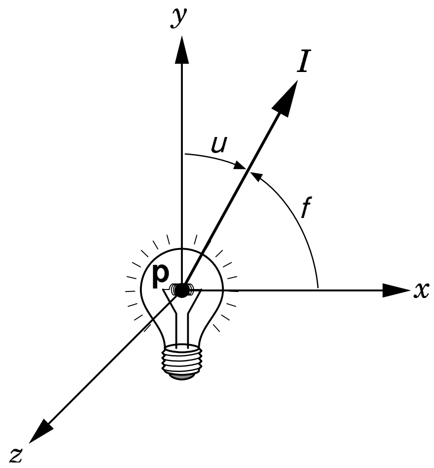


Light-Material Interactions



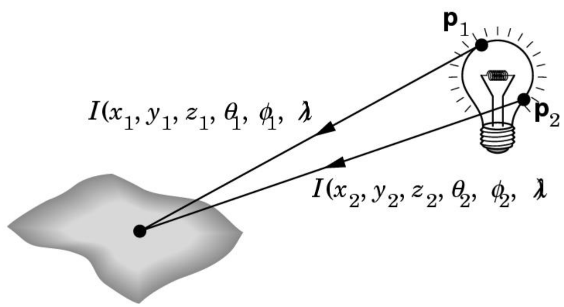
specular diffuse translucent

Light Sources



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

Light Sources (2)



Color sources: $I = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix}$

Light Sources (3)

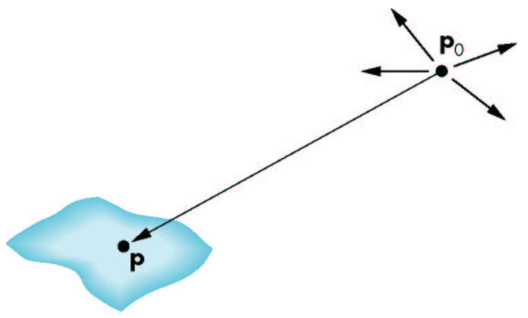
- ambient light
- point sources
- spotlights
- distant light

Ambient Light

- uniform level of light, independent of positions

$$I_a = \begin{bmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

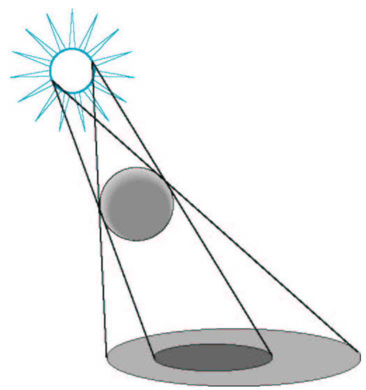
Point Sources



$$d = |p - p_0|$$

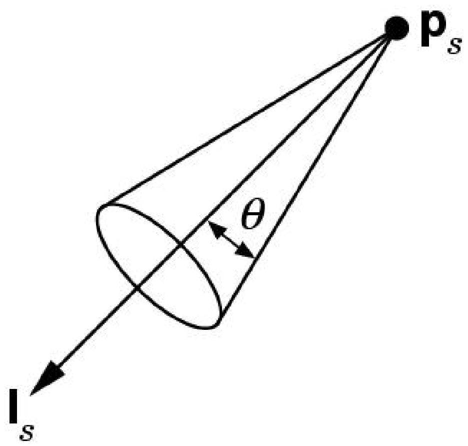
$$I(p, p_0) = \frac{1}{d^2} \begin{bmatrix} I_r(p_0) \\ I_g(p_0) \\ I_b(p_0) \end{bmatrix} \longrightarrow \frac{1}{a+bd+cd^2} \begin{bmatrix} I_r(p_0) \\ I_g(p_0) \\ I_b(p_0) \end{bmatrix}$$

Finite-size Light Sources

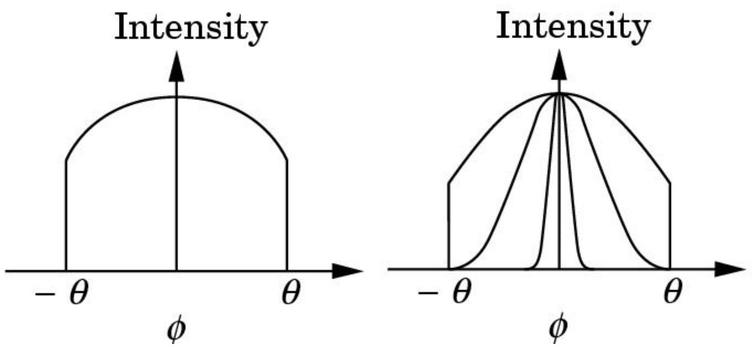


Umbra and penumbra
(not modelled with point sources)

Spotlight



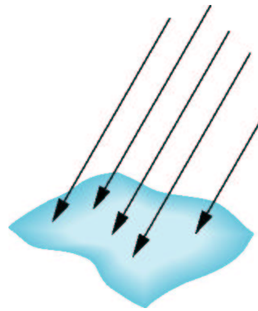
Spotlight Intensity



$\cos^e \phi$

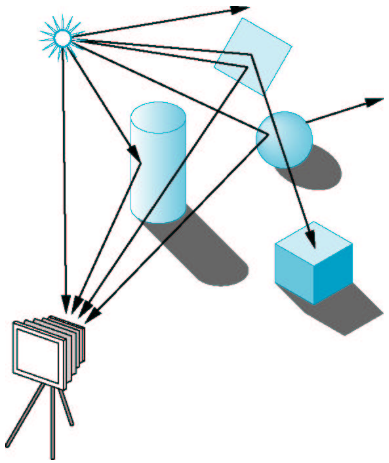
"spotlight exponent" e

Distant Light Sources

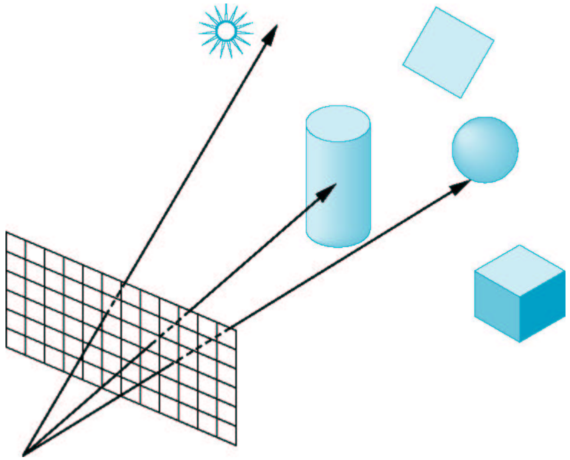


$$p_0 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \longrightarrow p_0 = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

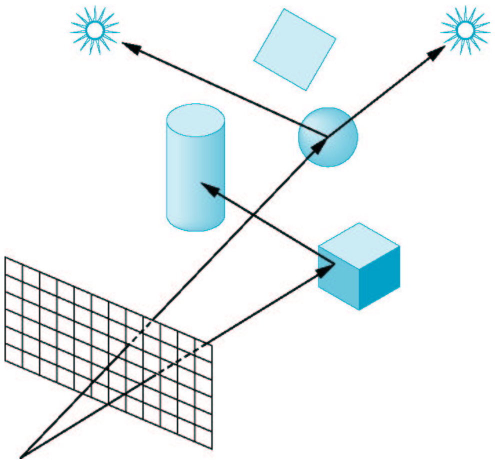
Global Rendering (Ray Tracing)



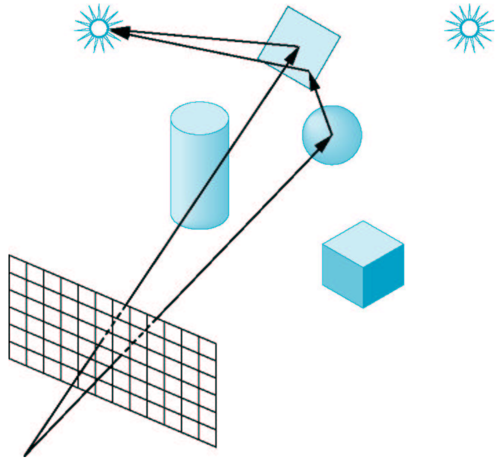
Ray Casting Model



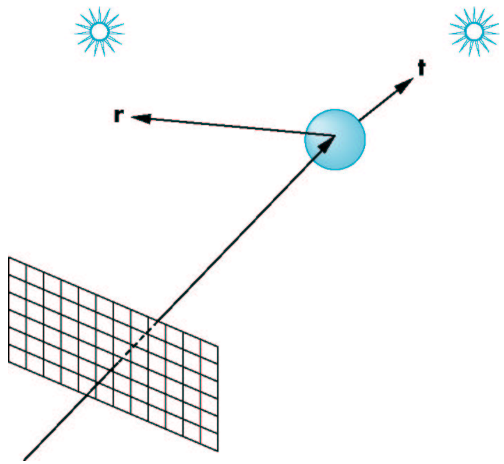
Shadow Rays



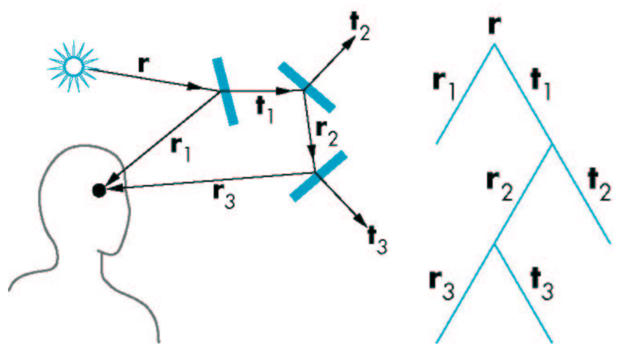
Ray Tracing with Mirrors



Ray Tracing with Reflection and Transmission

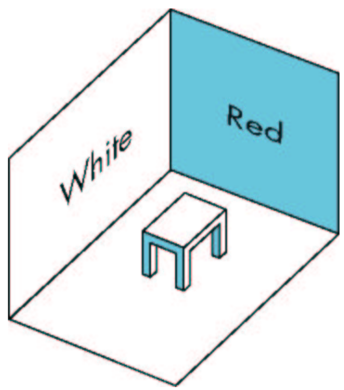


Ray Trees



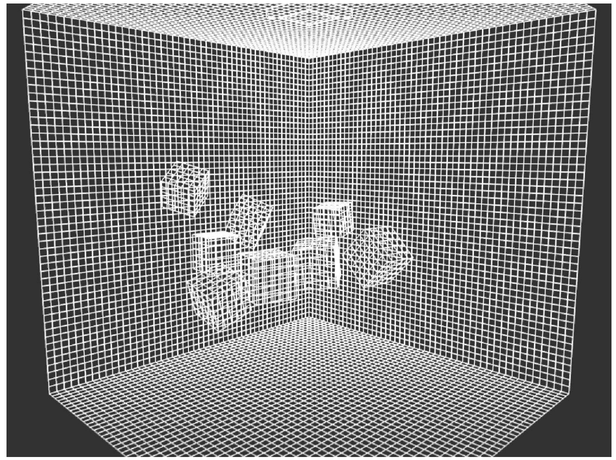
not feasible for diffuse surfaces (too many rays)

Radiosity



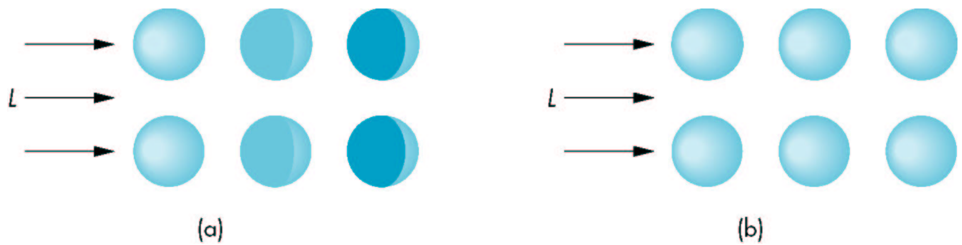
diffuse-diffuse interactions only
approximating the rendering equation

Radiosity (2)



compute interactions between pairs of *patches*

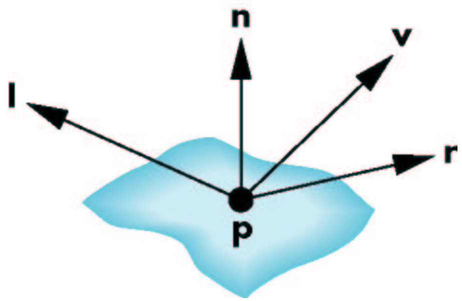
From Global to Local Shading



(global) (local)

Global Shading takes too long.
OpenGL does something simpler (local model)

The Phong Reflection Model



n normal vector at p
v vector to viewer (COP)
l vector to light source
r direction of reflected light

Phong Reflection Model (2)

Three types of interaction: ambient, diffuse, and specular
With the Phong model, light sources have three components: ambient, diffuse, and specular
All kinds of lights, reflections, etc. are computed separately per color (red, green, blue). Simplification:

$$L_i = \begin{bmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{bmatrix} \rightarrow L_i = \begin{bmatrix} L_{ia} \\ L_{id} \\ L_{is} \end{bmatrix}$$

Phong Reflection Model (3)

Reflection:

$$R_i = \begin{bmatrix} R_{ia} \\ R_{id} \\ R_{is} \end{bmatrix}$$

Intensities:

$$\begin{aligned} I &= I_a + I_d + I_s = L_a R_a + L_d R_d + L_s R_s \\ &= \textstyle\sum_i (I_a + I_d + I_s) + I_a \end{aligned}$$

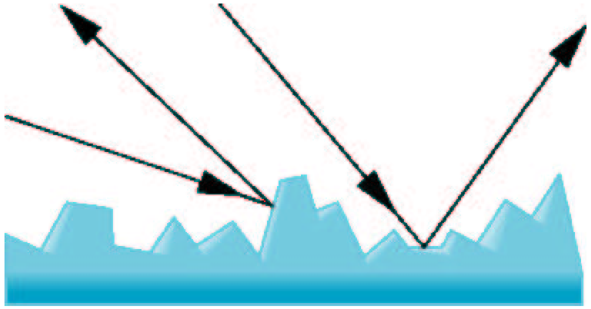
Ambient Reflection

$$R_a = k_a \qquad 0 \leq k_a \leq 1$$

$$I_a = k_a L_a$$

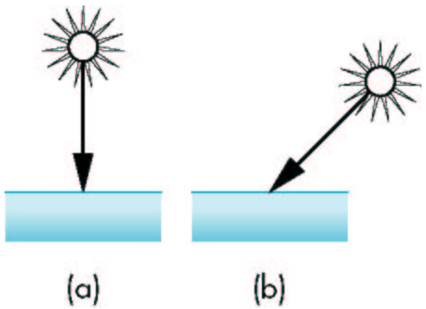
The three ambient coefficients (for red, green, blue) determine the "color" of an object.

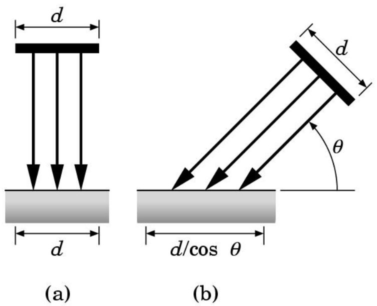
Diffuse Reflection



Illumination depends on angle of incoming light.

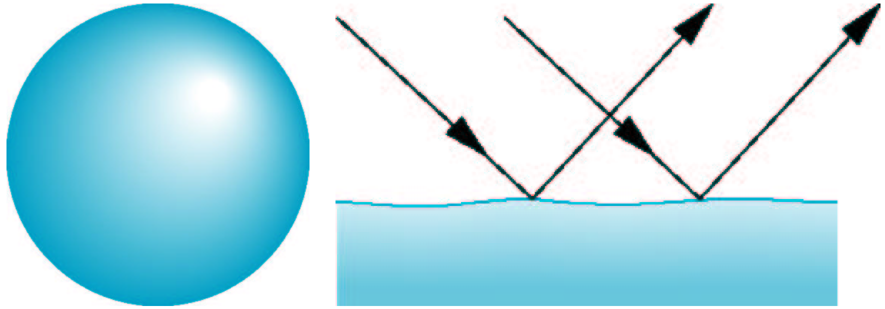
Illumination of Diffuse Surface





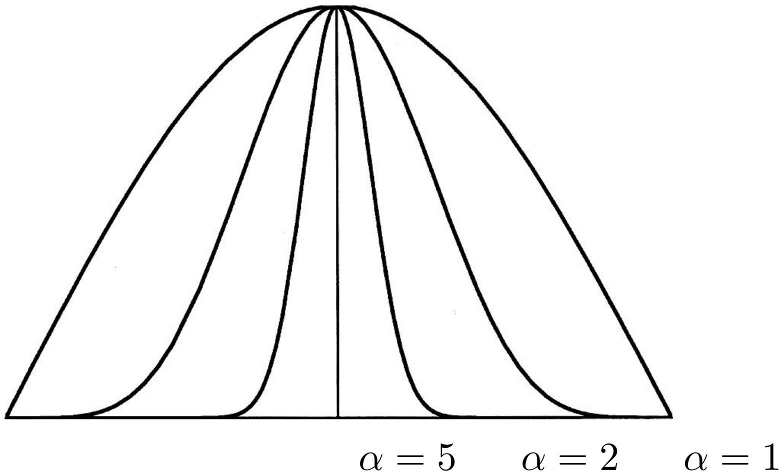
l, n unit length vectors (normalization a.s.a.p.)
$\cos\theta = l \cdot n$
$I_d = k_d(l \cdot n)L_d \qquad I_d = \frac{k_d}{a+bd+cd^2}(l \cdot n)L_d$

Specular Reflection



$I_s = k_s L_s \cos^\alpha \theta$

Shininess Coefficient α



metals: $\alpha \in [100 \ldots 500]$ mirror: $\alpha \to \infty$

Computation of Normal Vectors

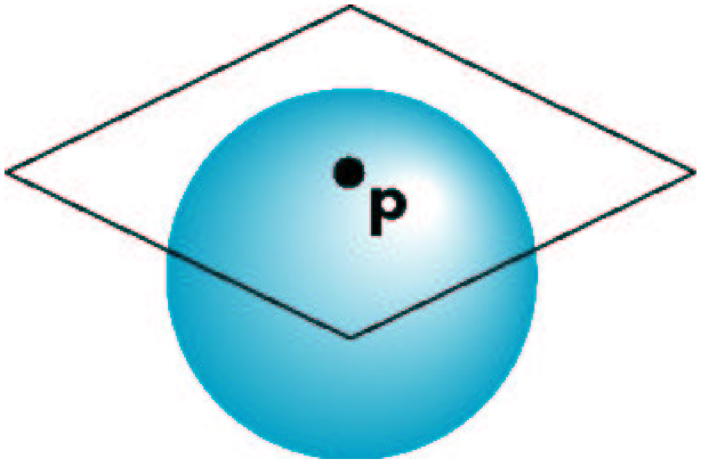
For Polygons, take cross-product of 3 non-collinear points:

$$n = (p_2 - p_0) \times (p_1 - p_0)$$

for convenience of further processing, normalize it:

$$n = \frac{(p_2 - p_0) \times (p_1 - p_0)}{|(p_2 - p_0) \times (p_1 - p_0)|}$$

Normal of a Sphere



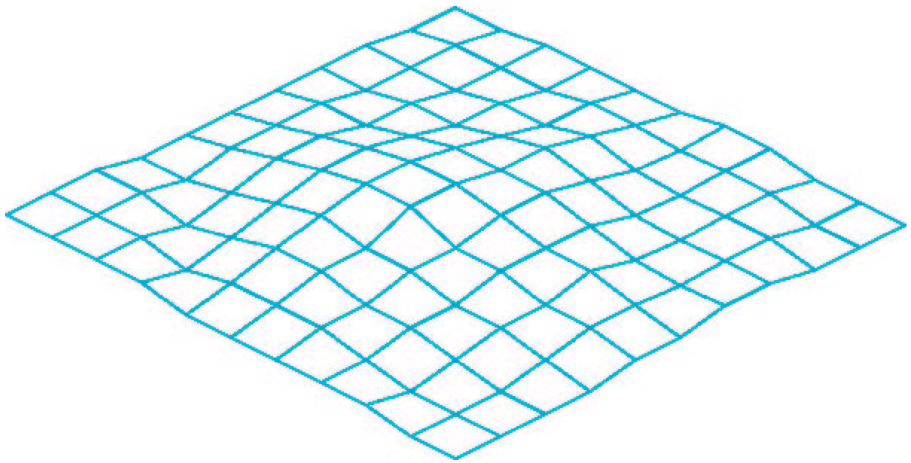
$$n = \frac{p - p_0}{|p - p_0|}$$

Normal Vectors in OpenGL

Normal Vectors have to be computed by the user, and assigned to vertices:

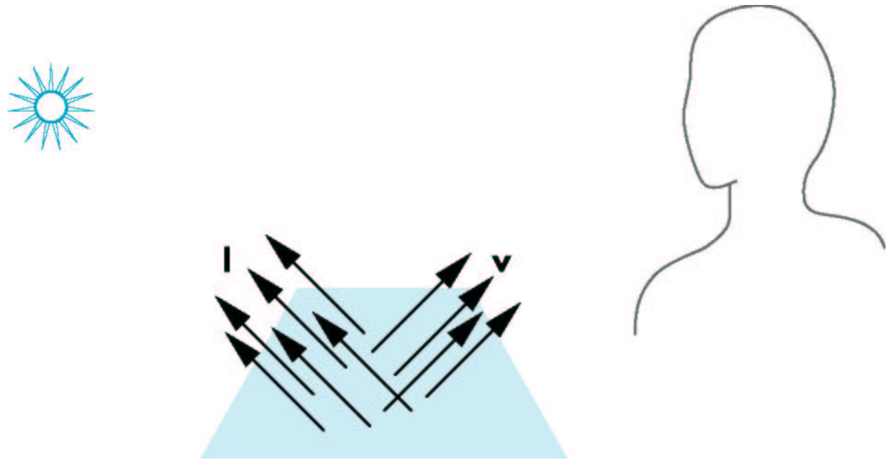
```
glNormal3f(nx, ny, nz);
glNormal3fv(pointer_to_normal):

glVertex3fv( ... );
```


Polygonal Shading



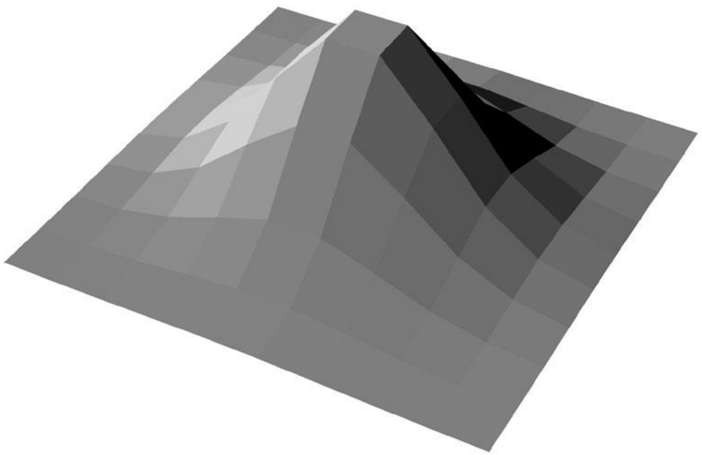
Three ways of shading: flat, interpolative, Phong

Distant Source and Viewer

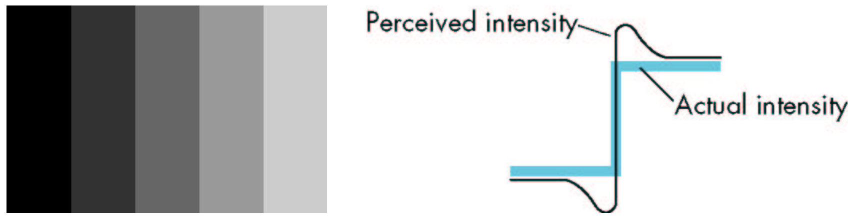



```
glShadeModel(GL_FLAT);
```


Flat Shading



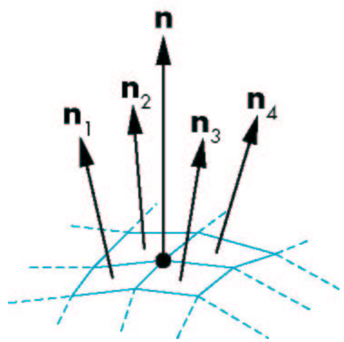
Problem: Mach Bands



The human visual system overemphasizes the intensity steps.

We need smoother shading.

Interpolative/Gouraud Shading



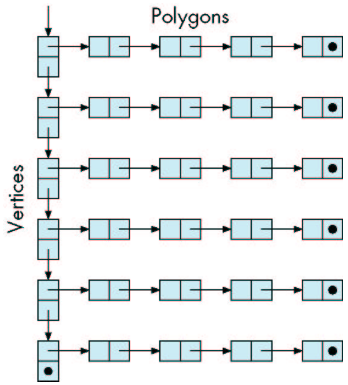
assign averaged normal vector to vertex:

$$n = \frac{n_1+n_2+n_3+n_4}{|n_1+n_2+n_3+n_4|}$$

```
glShadeModel(GL_SMOOTH)
```

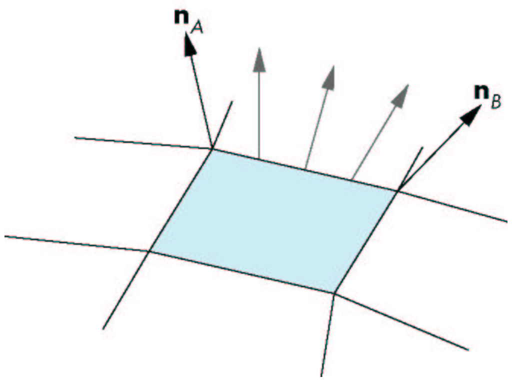
and interpolate colors between vertices

How do we know adjacent polygons?



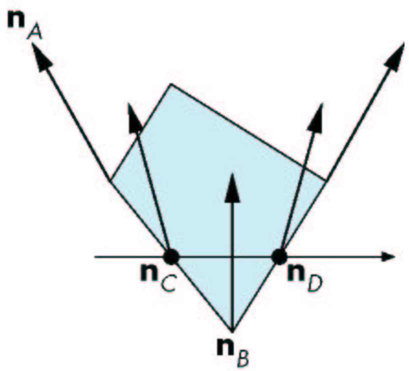
Application has to do the "bookkeeping."

Phong Shading



Interpolating edge normals: $n(\alpha) = (1 - \alpha)n_A + \alpha n_B$

Bilinear Interpolation



$n(\alpha, \beta) = (1 - \beta)n_C + \beta n_D$

Phong shading takes time – done offline!

Light Sources in OpenGL

. . . implement the Phong reflection model.

```
glLightfv(source, parameter, pointer_to_array);
glLightf(source, parameter, value);

GLfloat light0_pos[] = {1.0, 2.0, 3.0, 1.0};
GLfloat light0_dir[] = {1.0, 2.0, 3.0, 0.0};

GLfloat diffuse0[]  = {1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[]  = {1.0, 0.0, 0.0, 1.0};
GLfloat specular0[] = {1.0, 1.0, 1.0, 1.0};
```

Light Sources in OpenGL (2)

OpenGL standard requires 8 sources
GL_LIGHT0 . . . GL_LIGHT7.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

glLightfv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT,  ambient0);
glLightfv(GL_LIGHT0, GL_DIFFUSE,  diffuse0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```


Light Sources in OpenGL (3)

Attenuation: $f(d) = \frac{1}{a+bd+cd^2}$

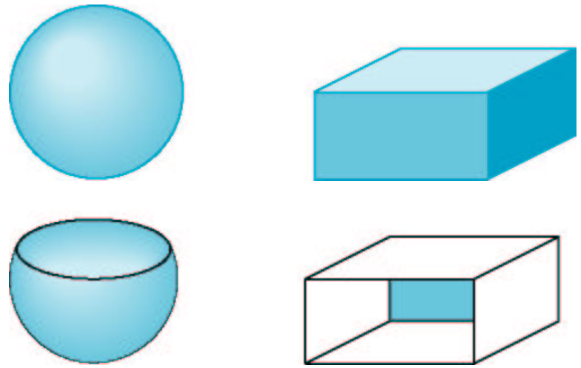
```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a);
```

Defining spotlights: GL_SPOT_DIRECTION, GL_SPOT_EXPONENT, GL_SPOT_CUTOFF

distant/local viewer:

```
glLightModel(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```


glLightModel(GL_LIGHT_MODEL_TWO_SIDED, GL_TRUE)



Materials in OpenGL

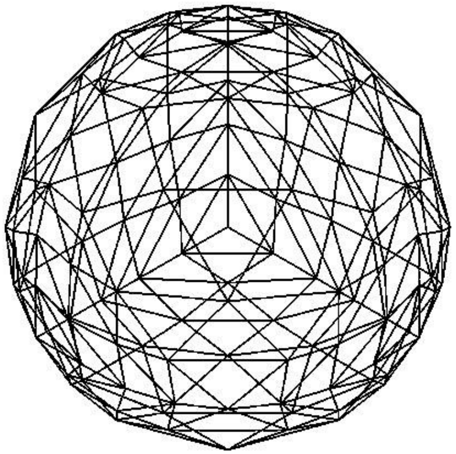
modal parameters (to be set before drawing vertices):

```
glMaterialfv(face, type, pointer_to_array);
glMaterialf(face, type, value);

GLfloat ambient[]  = {1.0, 0.0, 0.0, 1.0};
GLfloat diffuse[]  = {1.0, 0.0, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
```

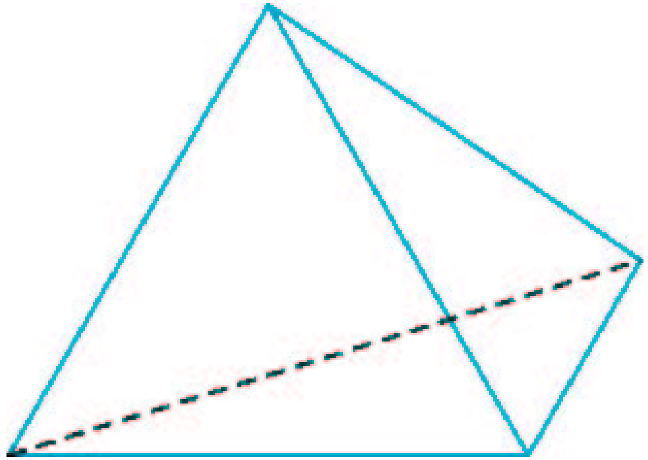
Materials in OpenGL(2)

```
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);

glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0);

GLfloat emission[]={0.0, 0.3, 0.3, 1.0};
glMaterialf(GL_FRONT_AND_BACK, GL_EMISSION, emission);
```

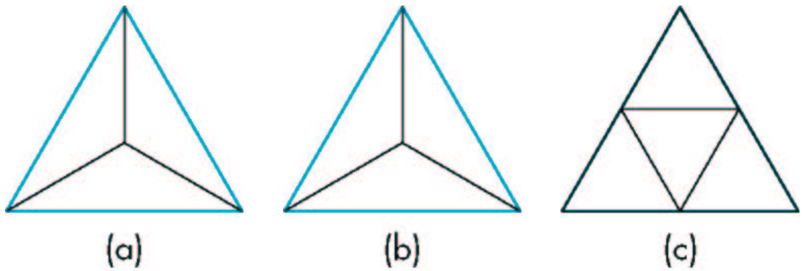
Shading a Sphere



Start from a Tetrahedron

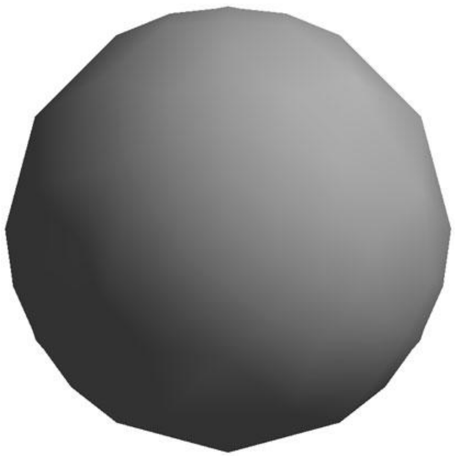


Recursive Subdivision



Flat-shaded Sphere

Shading with True Normal Vectors



Drawing Sphere-Triangles

```
void triangle( point a, point b, point c)

/* display one triangle using a line loop for wire frame, a single
normal for constant shading, 3 normals for interpolative shading */
{
    if (mode==0) glBegin(GL_LINE_LOOP); /* wire frame */
    else glBegin(GL_POLYGON);
       if(mode==1) glNormal3fv(a);      /* flat shading */
       if(mode==2) glNormal3fv(a);      /* interpolative shading */
       glVertex3fv(a);
       if(mode==2) glNormal3fv(b);
       glVertex3fv(b);
       if(mode==2) glNormal3fv(c);
       glVertex3fv(c);
    glEnd();
}
```

Summary

- Light and Matter
- Ray Tracing / Radiosity
- The Phong Reflection Model
- Polygonal Shading
- Light and Matter in OpenGL
- Next week: Discrete techniques (texture etc.)