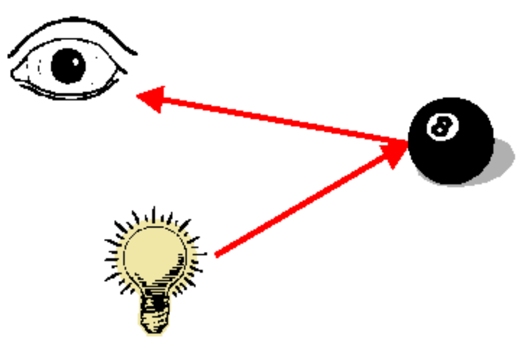
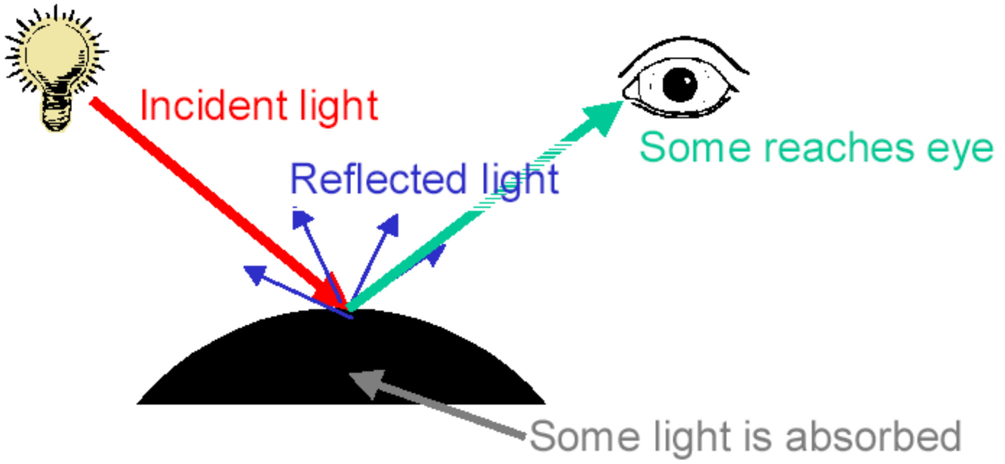
Determining and Object’s Appearance

Ultimately, we’re interested in modeling light transport in scene

- Light is emitted from light sources and interacts with surfaces
- on impact with an object, some is reflected and some is absorbed
- distribution of reflected light determines “finish” (matte, glossy, …)
- composition of light arriving at eye determines what we see

Let’s focus on the local interaction of light with single surface point





Modeling Light Sources

In general, light sources have a very complex structure

- incandescent light bulbs, the sun, CRT monitors, …

To simplify things, we'll focus on point light sources for now

- light source is a single infinitesimal point
- emits light equally in all directions (isotropic illumination)
- outgoing light is set of rays originating at light point

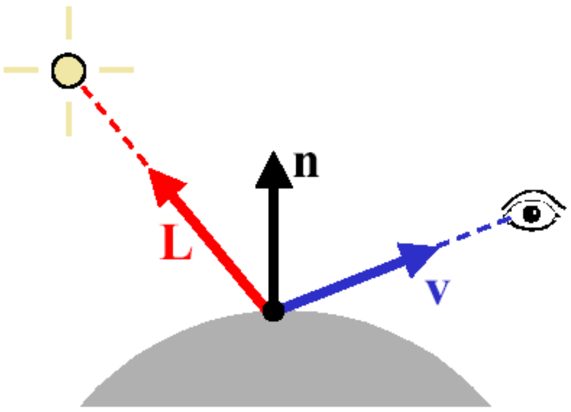
Creating lights in OpenGL

- glEnable(GL_LIGHTING) — turn on lighting of objects
- glEnable(GL_LIGHT0) — turn on specific light
- glLight(…) — specify position, emitted light intensity, …

Basic Local Illumination Model

We're only interested in light that finally arrives at view point

- a function of the light & viewing positions
- and local surface reflectance



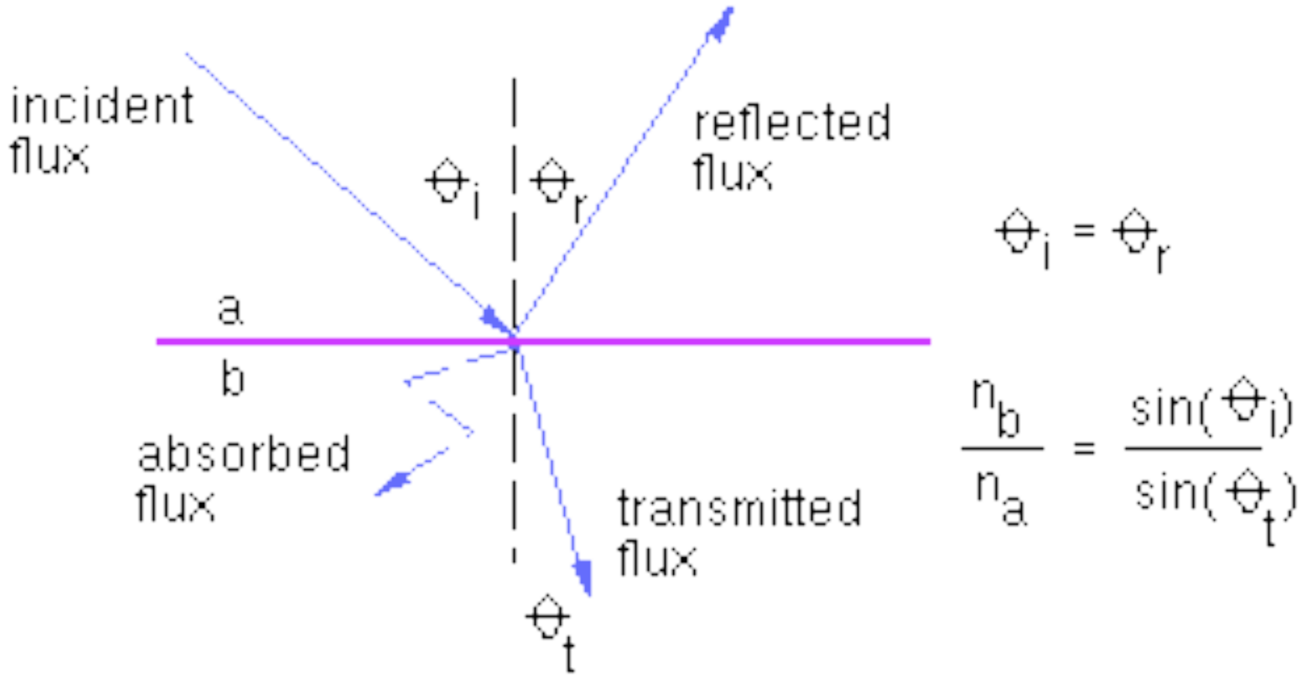
Characterize light using RGB triples

- can operate on each channel separately

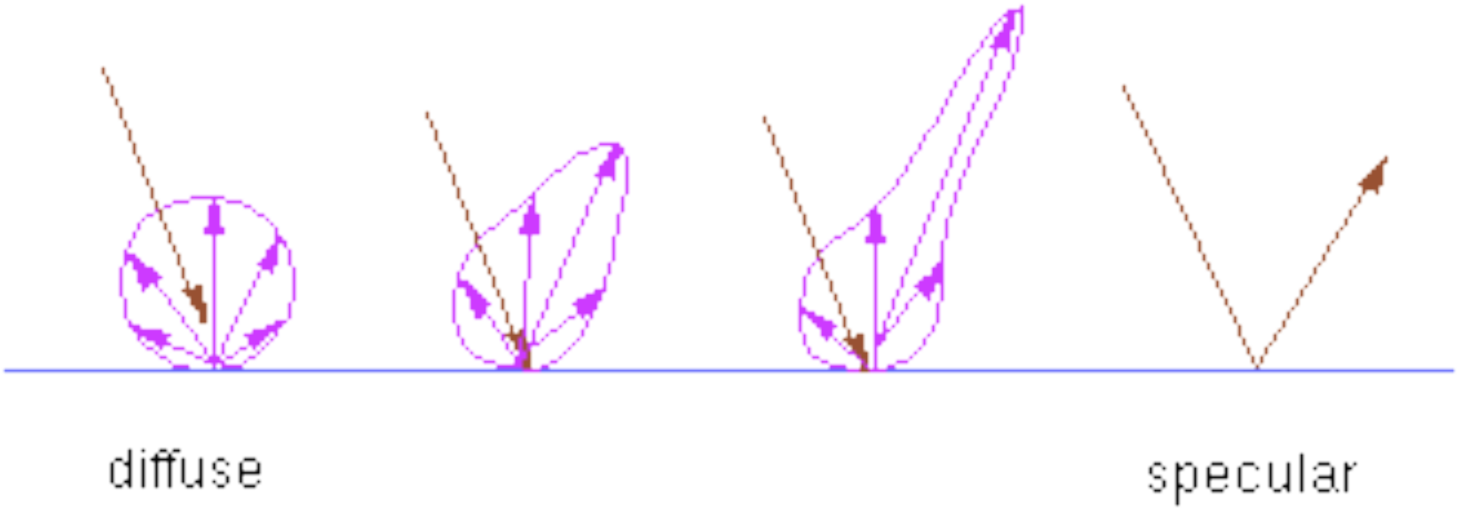
Given a point, compute intensity of reflected light

Local Illumination physics

Law of reflection and Snell's law of refraction



What are we trying to model ?



Diffuse Reflection

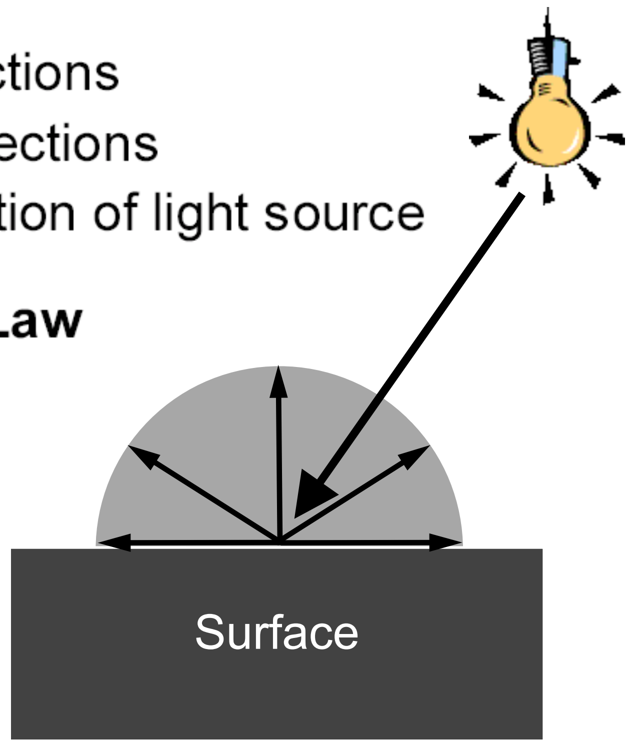
This is the simplest kind of reflection

- also called Lambertian reflection
- models dull, matte surfaces — materials like chalk

Ideal diffuse reflection

- scatters incoming light equally in all directions
- identical appearance from all viewing directions
- reflected intensity depends only on direction of light source

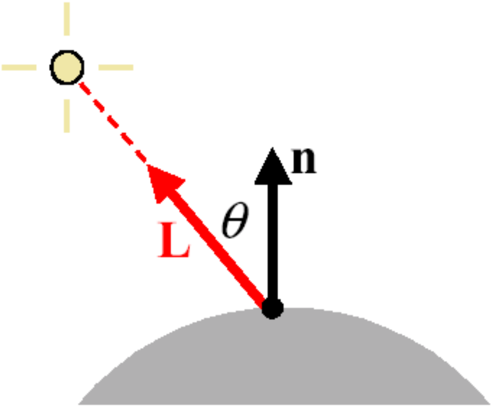
Light is reflected according to Lambert's Law



Lambert's Law for Diffuse Reflection

Purely diffuse object

$$
\boxed{\begin{aligned} I &= I_L k_d \cos\theta \\ &= I_L k_d (\mathbf{n} \cdot \mathbf{L}) \end{aligned}}
$$



I : resulting intensity

I_L : light source intensity

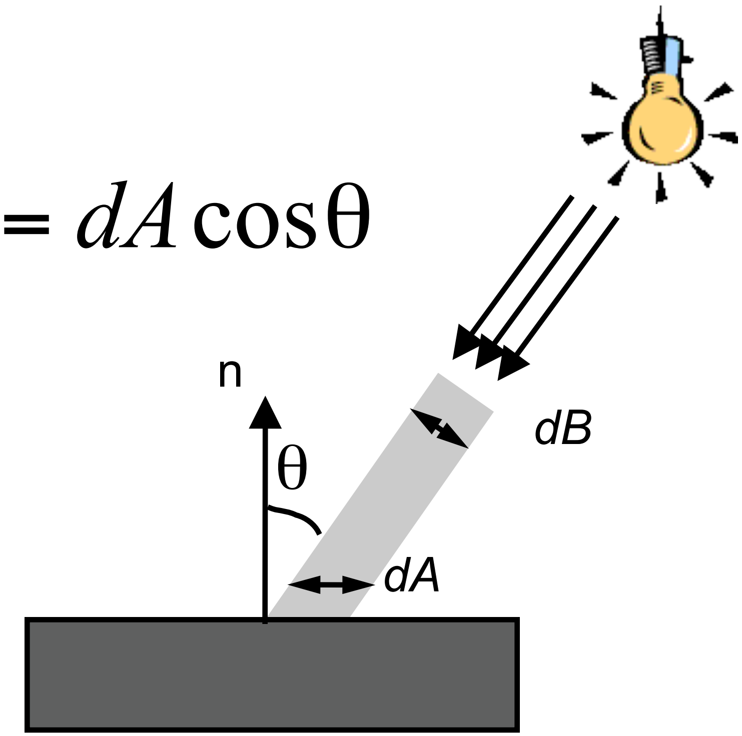
k_d : (diffuse) surface reflectance coefficient

$$k_d \in [0,1]$$

θ : angle between normal & light direction

Proof of Lambert's cosine law

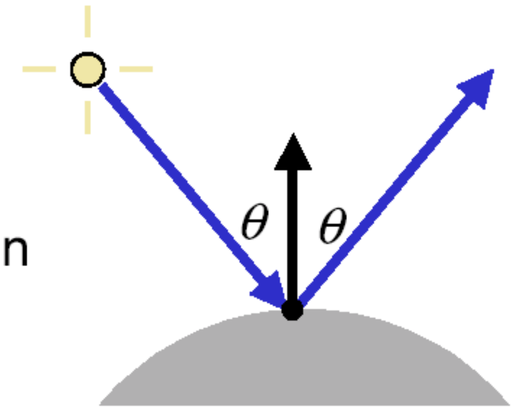
$$dB = dA\cos\theta$$



Specular Reflection

Diffuse reflection is nice, but many surfaces are shiny

- their appearance changes as the viewpoint moves
- they have glossy specular highlights (or specularities)
- because they reflect light coherently, in a preferred direction

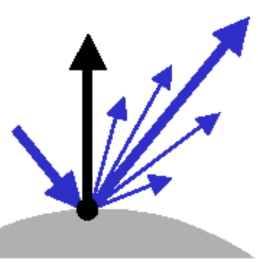


A mirror is a perfect specular reflector

- incoming ray reflected about normal direction
- nothing reflected in any other direction

Most surfaces are imperfect specular reflectors

- reflect rays in cone about perfect reflection direction

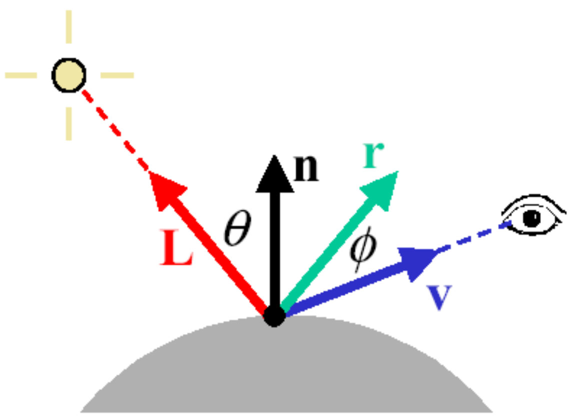


Phong Illumination Model

$$I = I_L k_d \cos\theta + I_L k_s \cos^n \phi$$
$$= I_L k_d (\mathbf{n} \cdot \mathbf{L}) + I_L k_s (\mathbf{r} \cdot \mathbf{v})^n$$

One particular specular reflection model

- quite common in practice
- it is purely empirical
- there's *no physical basis* for it



I : resulting intensity

I_L : light source intensity

k_s : (specular) surface reflectance coefficient

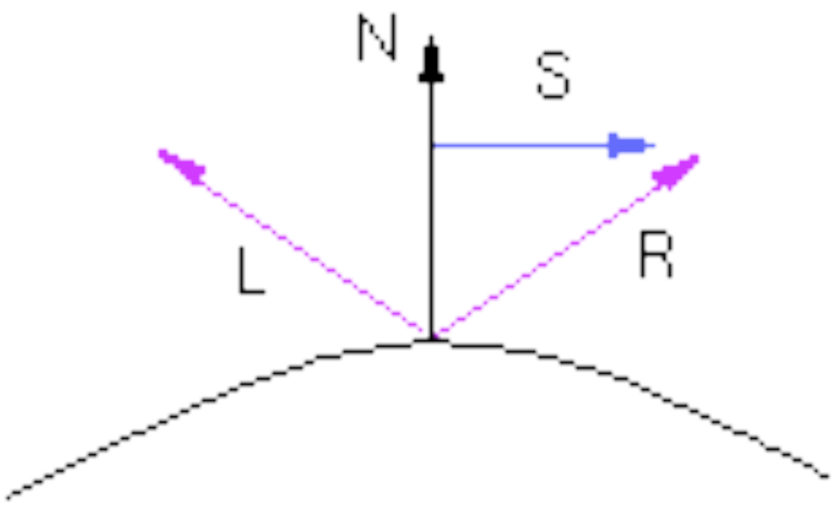
$$k_s \in [0,1]$$

ϕ : angle between viewing & reflection direction

n : "shininess" factor

Computing R

All vectors unit length!!

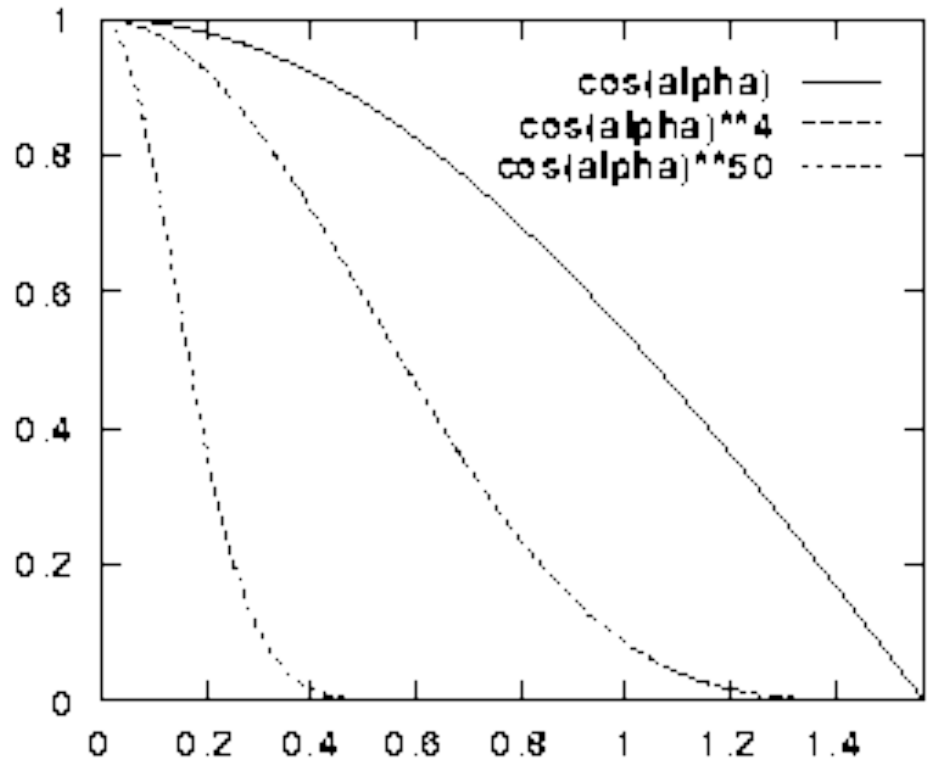


$$R = (N \bullet L)\, N + S$$

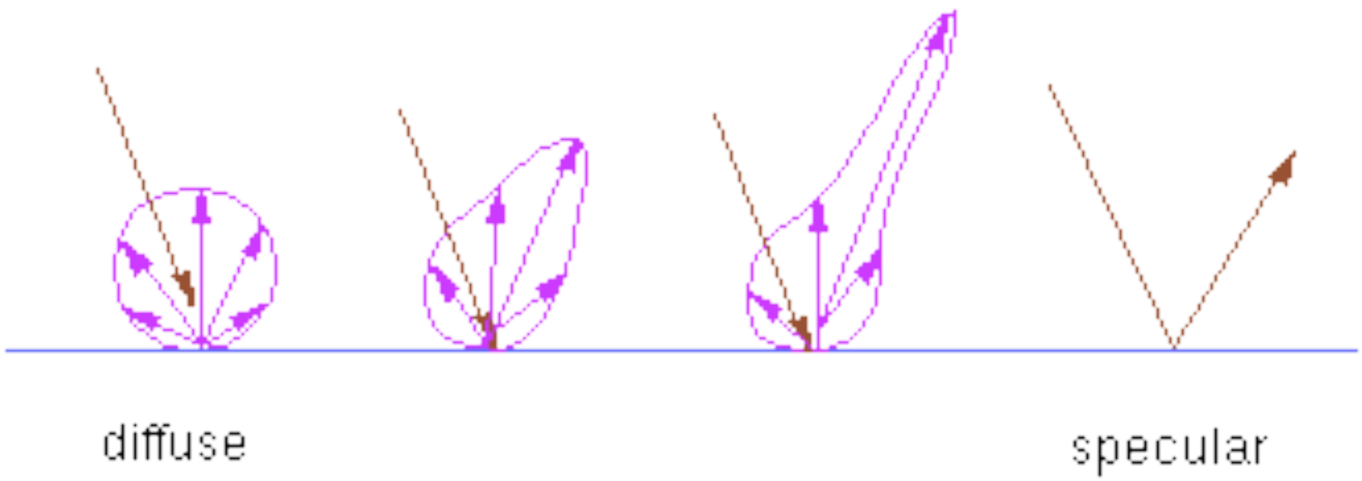
$$S = (N \bullet L)\, N - L$$

$$R = 2N\,(N \bullet L) - L$$

The effect of the exponent *n*

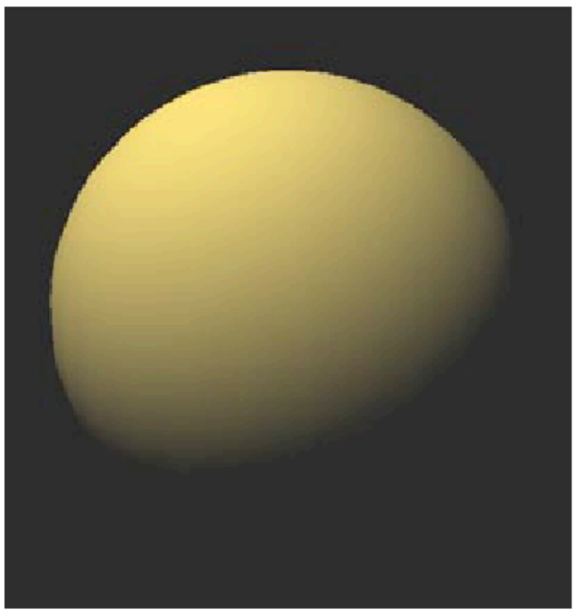


Comparison

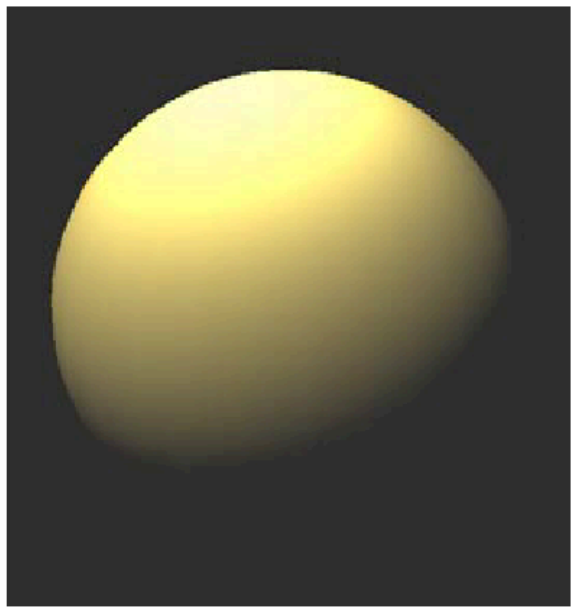


Examples of Phong Specular Model

Diffuse only



Diffuse + Specular (shininess 5)



Diffuse + Specular (shininess 50)

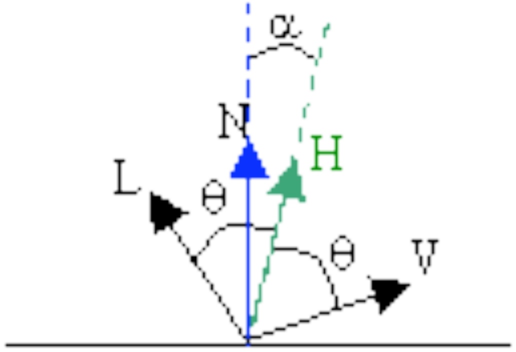
The Blinn-Torrance Specular Model

Agrees better with experimental results

$I_s = I_i K_{spec} (H \cdot V)^n$

Halfway vector H

$$H = \frac{L + V}{\| L + V \|}$$

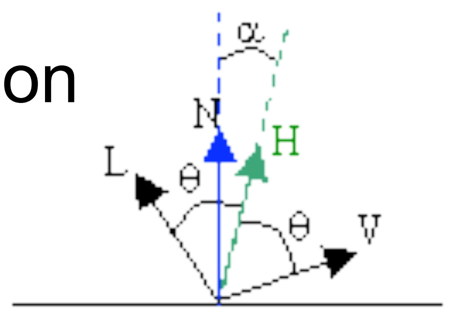


Advantages of the Blinn Specular Model

- Theoritical basis
- No need to compute reflective direction R
- N·H cannot be negative if N·L>0 and N·V>0

$$H = \frac{L + V}{\|L + V\|}$$

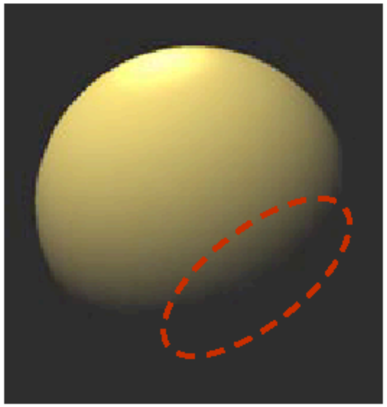
- If the light is directional and we have orthographic projection then N*H constant



The Ambient Glow

So far, areas not directly illuminated by any light appear black

- this tends to look rather unnatural
- in the real world, there's lots of ambient light



To compensate, we invent new light source

- assume there is a constant ambient "glow"
- this ambient glow is *purely fictitious*

Just add in another term to our illumination equation

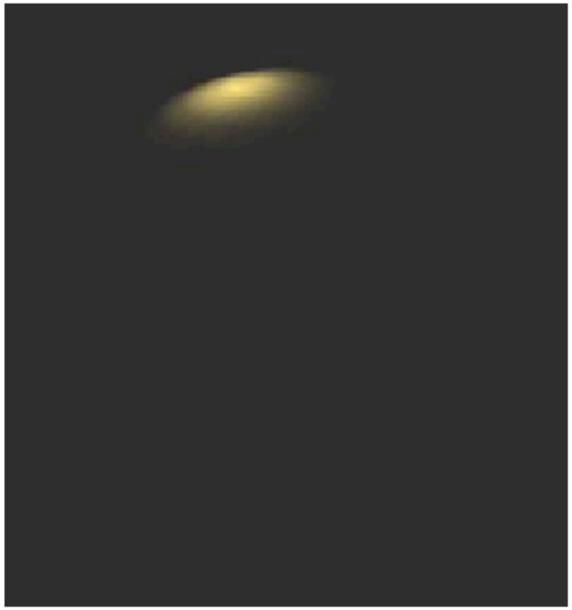
$$I = I_L k_d \cos\theta + I_L k_s \cos^n \phi + I_a k_a$$

I_a : ambient light intensity

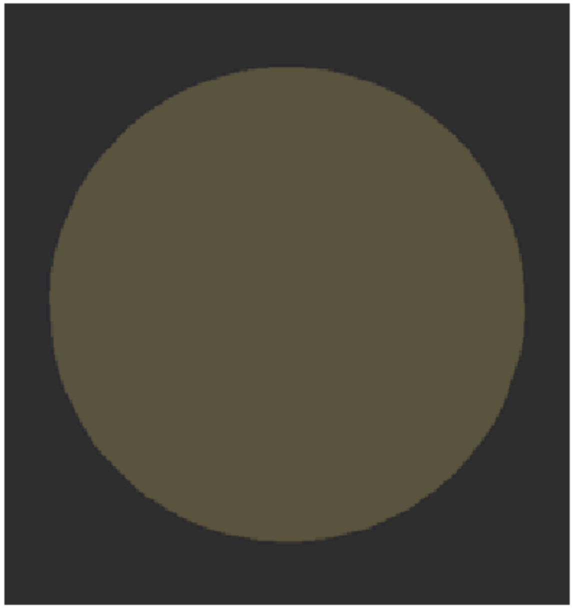
k_a : (ambient) surface reflectance coefficient

Our Three Basic Components of Illumination

Diffuse

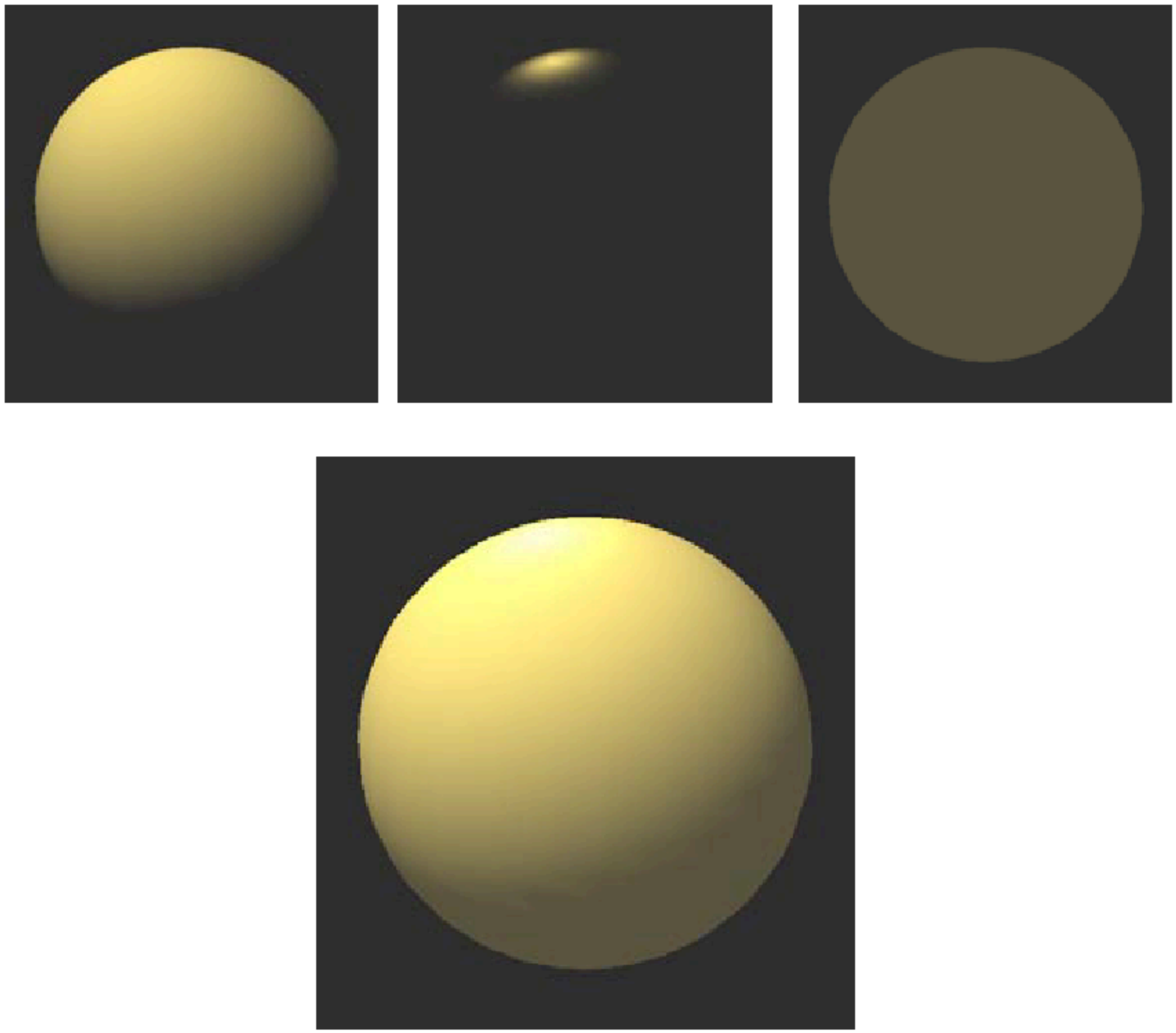


Specular



Ambient

Combined for the Final Result



Lights and materials

$ObjectColor_r = I_r = I_{a\_r} K_{a\_r} + I_{i\_r} K_{diff\_r} (N \cdot L) + I_{i\_r} K_{spec\_r} (R \cdot V)^n$

$ObjectColor_g = I_g = I_{a\_g} K_{a\_g} + I_{i\_g} K_{diff\_g} (N \cdot L) + I_{i\_g} K_{spec\_g} (R \cdot V)^n$

$ObjectColor_b = I_b = I_{a\_b} K_{a\_b} + I_{i\_b} K_{diff\_b} (N \cdot L) + I_{i\_b} K_{spec\_b} (R \cdot V)^n$

Material properties:

$K_a, K_{diff}, K_{spec}, n$

Light properties

I_a, I_{diff}, I_{spec}

Questions

If you shine red light (1,0,0) to a white object what color does the object appear to have?

What if you shine red light (1,0,0) to a green object (0,1,0) ?

What is the color of the highlight?

Special cases

$$I_r = I_{a\_r}K_{a\_r} + I_{i\_r}K_{diff\_r}(N \times L) + I_{i\_r}K_{spec\_r}(R \times V)^n$$

$$I_g = I_{a\_g}K_{a\_g} + I_{i\_g}K_{diff\_g}(N \times L) + I_{i\_g}K_{spec\_g}(R \times V)^n$$

$$I_b = I_{a\_b}K_{a\_b} + I_{i\_b}K_{diff\_b}(N \times L) + I_{i\_b}K_{spec\_b}(R \times V)^n$$

- What should be done if I >1?
  Clamp the value of I to one.
- What should be done if N*L < 0?
  Clamp the value of I to zero or flip the normal.
- How can we handle multiple light sources?
  Sum the intensity of the individual contributions.

Shading Polygons: Flat Shading

Illumination equations are evaluated at surface locations

- so where do we apply them?

We could just do it once per polygon

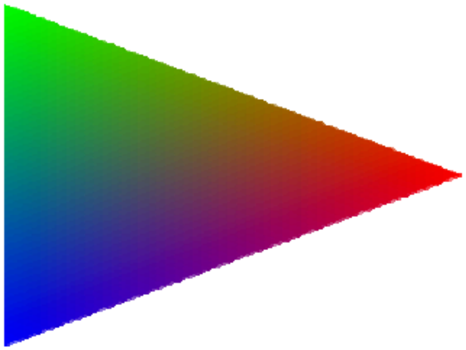
- fill every pixel covered by polygon with the resulting color

OpenGL — glShadeModel(GL_FLAT)

Shading Polygons: Gouraud Shading

Alternatively, we could evaluate at every vertex

- compute color for each covered pixel
- linearly interpolate colors over polygon



Misses details that don't fall on vertex

- specular highlights, for instance

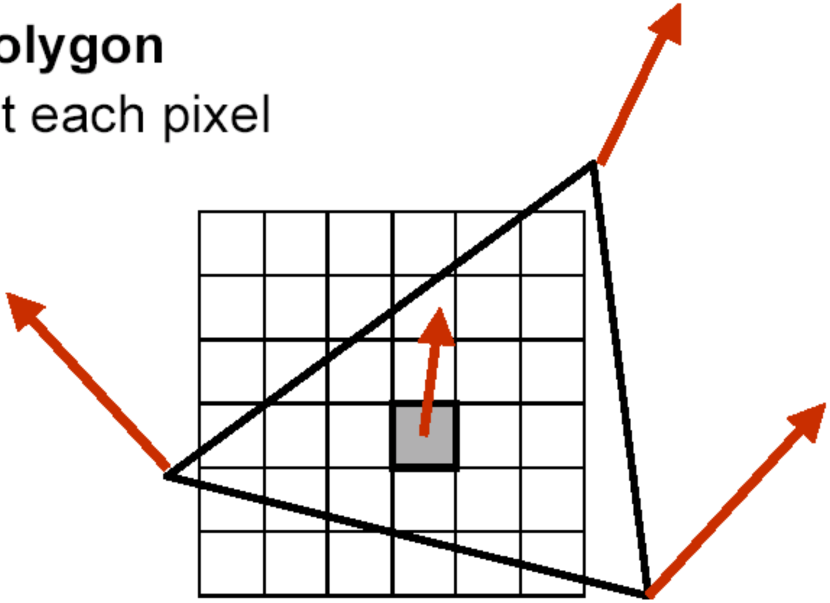
OpenGL — glShadeModel(GL_SMOOTH)

Shading Polygons: Phong Shading

Don't just interpolate colors over polygons

Interpolate surface normal over polygon

- evaluate illumination equation at each pixel



OpenGL — not supported

Defining Materials in OpenGL

Just like everything else, there is a current material

- specifies the reflectances of the objects being drawn
- reflectances (e.g., k_d) are RGB triples

Set current values with glMaterial(…)

```
GLfloat tan[] = {0.8, 0.7, 0.3, 1.0};
GLfloat tan2[] = {0.4, 0.35, 0.15, 1.0};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, tan);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, tan);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tan2);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0);
```

Defining Lights in OpenGL

A fixed set of lights are available (at least 8)

- turn them on with glEnable(GL_LIGHTx)
- set their values with glLight(…)

```
GLfloat white[] = {1.0, 1.0, 1.0, 1.0}
GLfloat p[] = {-2.0, -3.0, 10.0, 1.0}; // w=0 for directional light

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

glLightfv(GL_LIGHT0, GL_POSITION, p);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white); // can be different

glEnable(GL_NORMALIZE); // guarantee unit normals
```

Tricky Point about light position in OpenGL

The light position is specified in world coordinates, transformed with the current modelview matrix and stored in EYE coordinates.

- What does that mean?
- It means that if you change the position of the eye after the light position is set
  GLfloat pos[4] = {0,0,0,1} ;
  glLightfv(GL_LIGHT0, GL_POSITION, pos) ;
  gluLookAt(…..) ;
  The light will maintain its position with the respect to the new eye! i.e it will move with the camera.

Example1:

Where is the light with respect to the eye?

```
GLfloat pos[4] = {0,0,0,1} ;
GLfloat eye[3] = {0,0,10} ;
GLfloat ref[3] = {0,0,0} ;
GLfloat up[3] = {0,1,0} ;

glMatrixMode(GL_MODELVIEW) ;
glLoadIdentity() ;
glLightfv(GL_LIGHT0, GL_POSITION, pos) ;
gluLookAt(eye,ref,up) ;
```

World?

Example1:

Where is the light with respect to the eye?

```
GLfloat pos[4] = {0,0,0,1} ;
GLfloat eye[3] = {0,0,10} ;
GLfloat ref[3] = {0,0,0} ;
GLfloat up[3] = {0,1,0} ;

glMatrixMode(GL_MODELVIEW) ;
glLoadIdentity() ;   // that means camera matrix identity as well
glLightfv(GL_LIGHT0, GL_POSITION, pos) ; // 0  with respect to
                                         // current camera
gluLookAt(eye,ref,up) ;                  // 0 with respect to new
                                         // camera
```

World?

(0,0,10)

Example2:

Where is the light with respect to the eye?

```
GLfloat pos[4] = {0,0,0,1} ;
GLfloat eye[3] = {0,0,10} ;
GLfloat ref[3] = {0,0,0} ;
GLfloat up[3] = {0,1,0} ;

glMatrixMode(GL_MODELVIEW) ;
glLoadIdentity() ;
glTranslatef(0,0,-10) ;
glLightfv(GL_LIGHT0, GL_POSITION, pos) ;
gluLookAt(eye,ref,up) ;
```

World?

Example3:

Where is the light with respect to the eye?

```
GLfloat pos[4] = {0,0,0,1} ;
GLfloat eye[3] = {0,0,10} ;
GLfloat ref[3] = {0,0,0} ;
GLfloat up[3] = {0,1,0} ;

glMatrixMode(GL_MODELVIEW) ;
glLoadIdentity() ;
gluLookAt(eye,ref,up) ;
glLightfv(GL_LIGHT0, GL_POSITION, pos) ;
glutSwapBuffers() ;
```

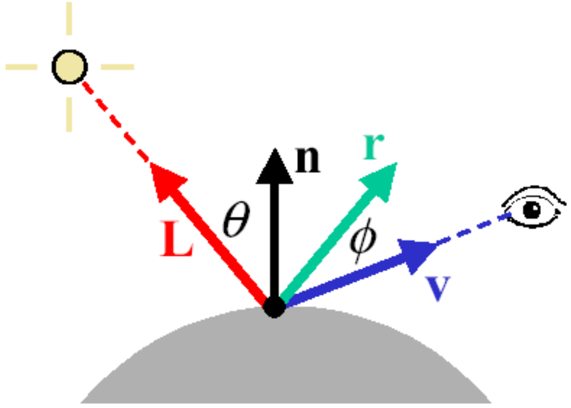
World?

Summarizing the Shading Model

We describe local appearance with illumination equations

- consists of a sum of set of components — light is additive
- treat each wavelength independently
- currently: diffuse, specular, and ambient terms

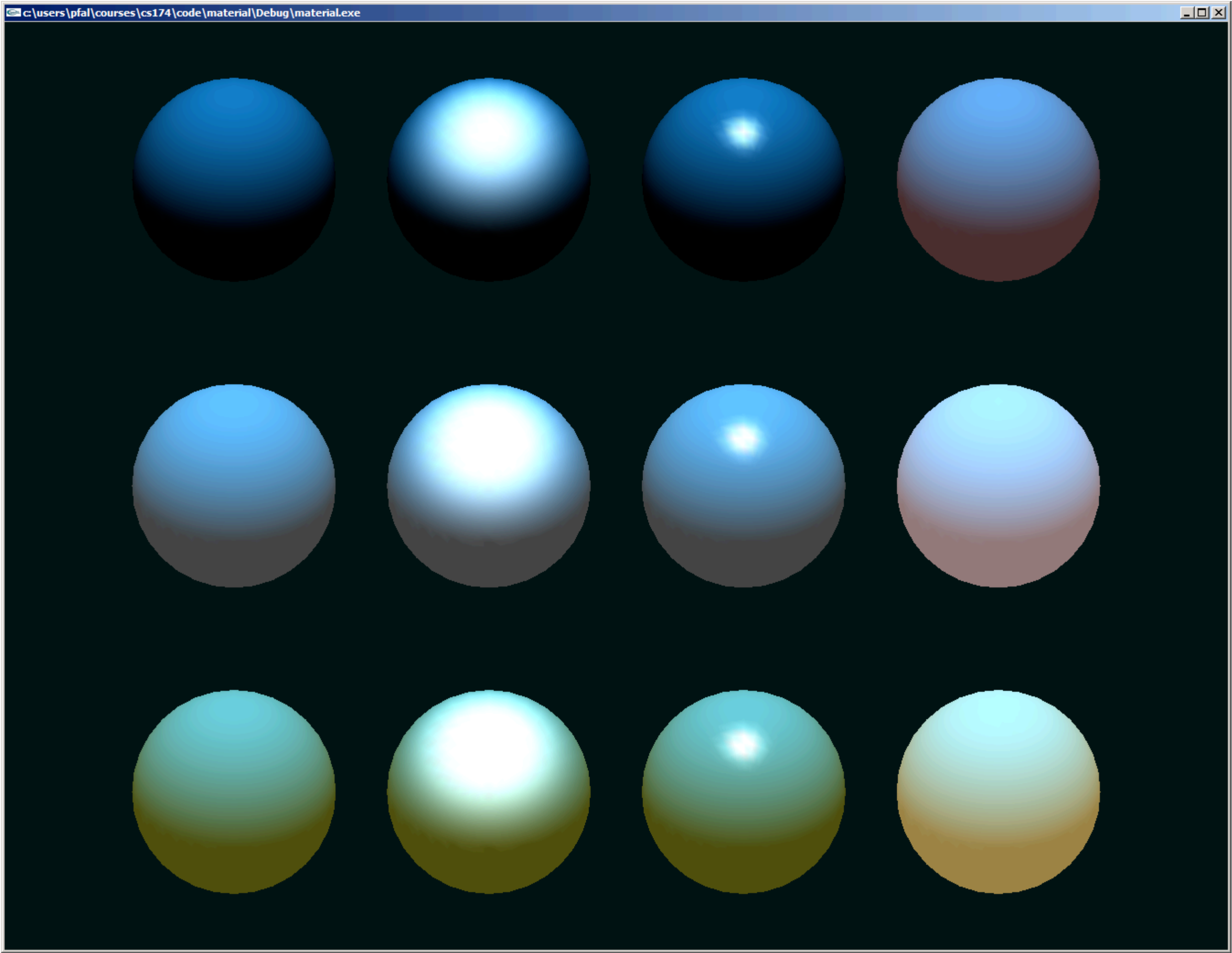
$$I = I_L k_d \cos\theta + I_L k_s \cos^n \phi + I_a k_a$$



Must shade every pixel covered by polygon

- flat shading: constant color
- Gouraud shading: interpolate corner colors
- Phong shading: interpolate corner normals

Examples


c:\users\pfal\courses\cs174\code\material\Debug\material.exe

Problems with shading algorithms

Orientation dependence

Silhouettes

Perspective distortion

- It happens at screen space so need to use hyperbolic interpolation

T-vertices

- If you do not have smooth normals color changes if polygon order changes

Generation of vertex normals

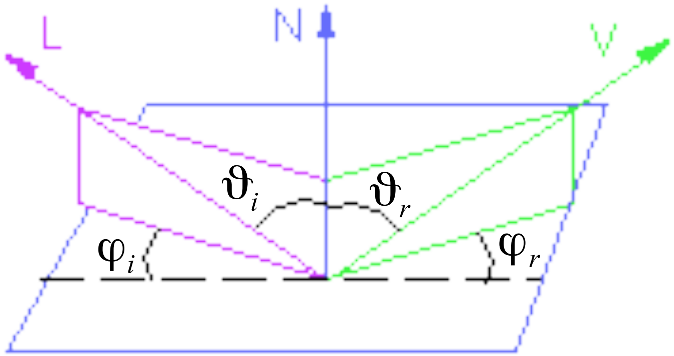
Advanced concepts

Physics-based illumination models

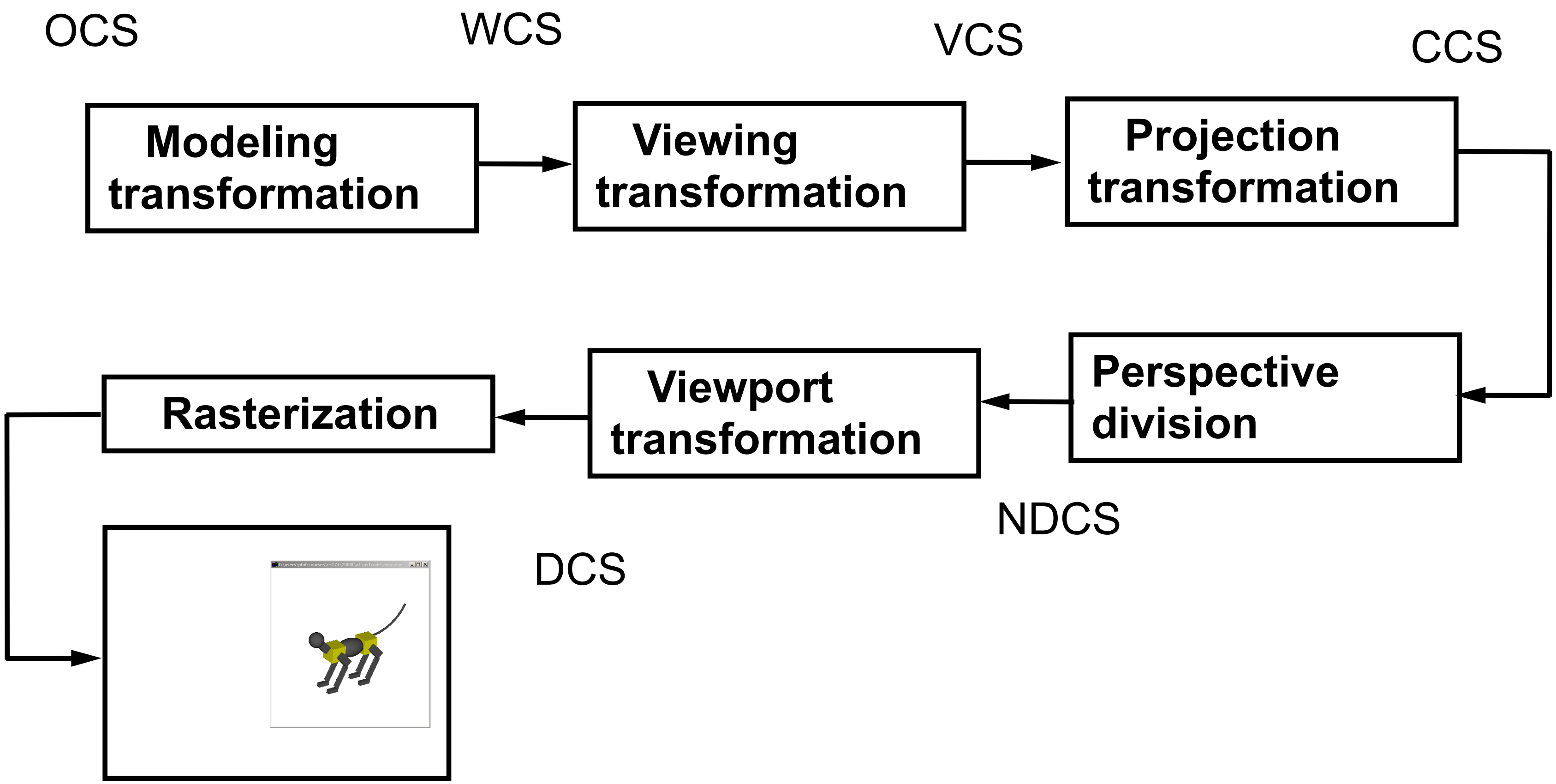
BRDF: Bidirectional reflectance function

$$\rho(\vartheta_i, \varphi_i, \vartheta_r, \varphi_r, \lambda)$$

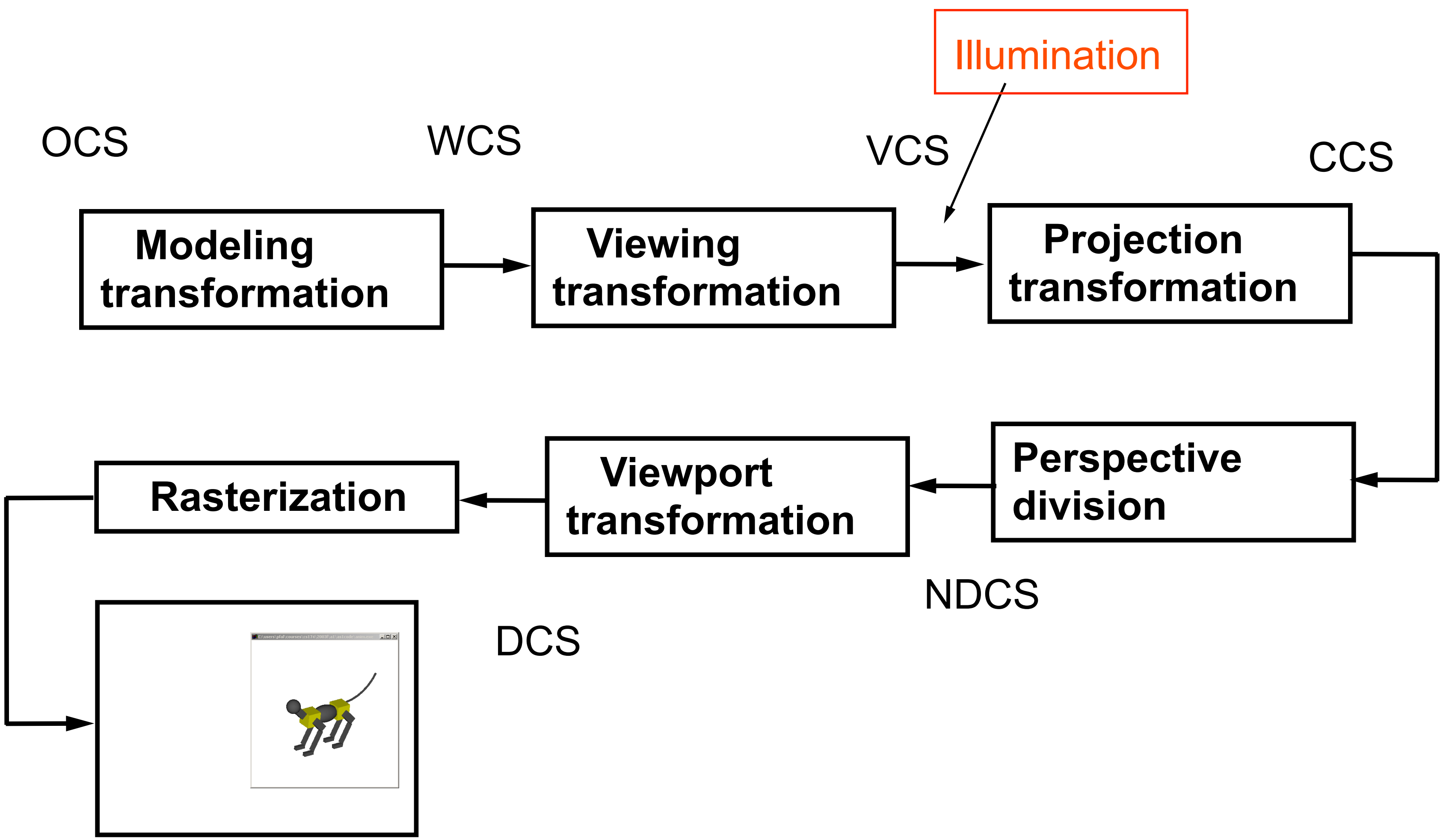
λ : light wavelength



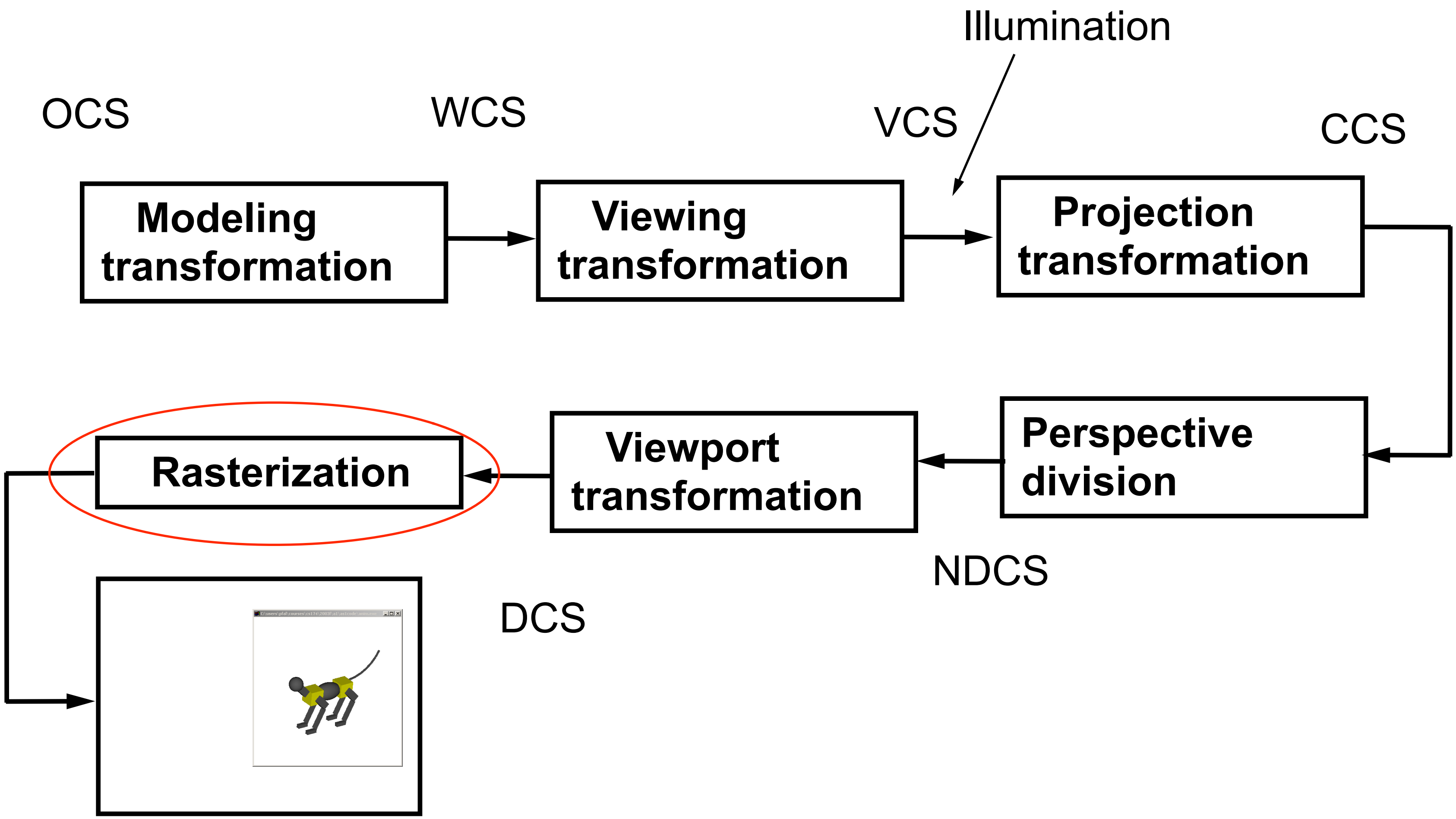
Illumination in Graphics Pipeline



Illumination in Graphics Pipeline

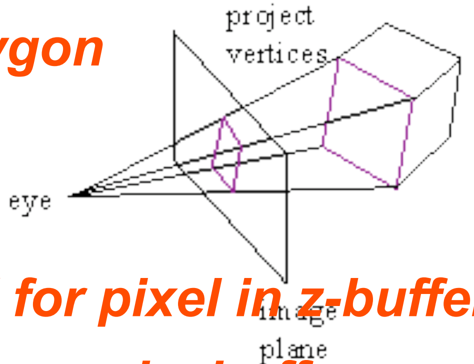


Z-buffer Graphics Pipeline

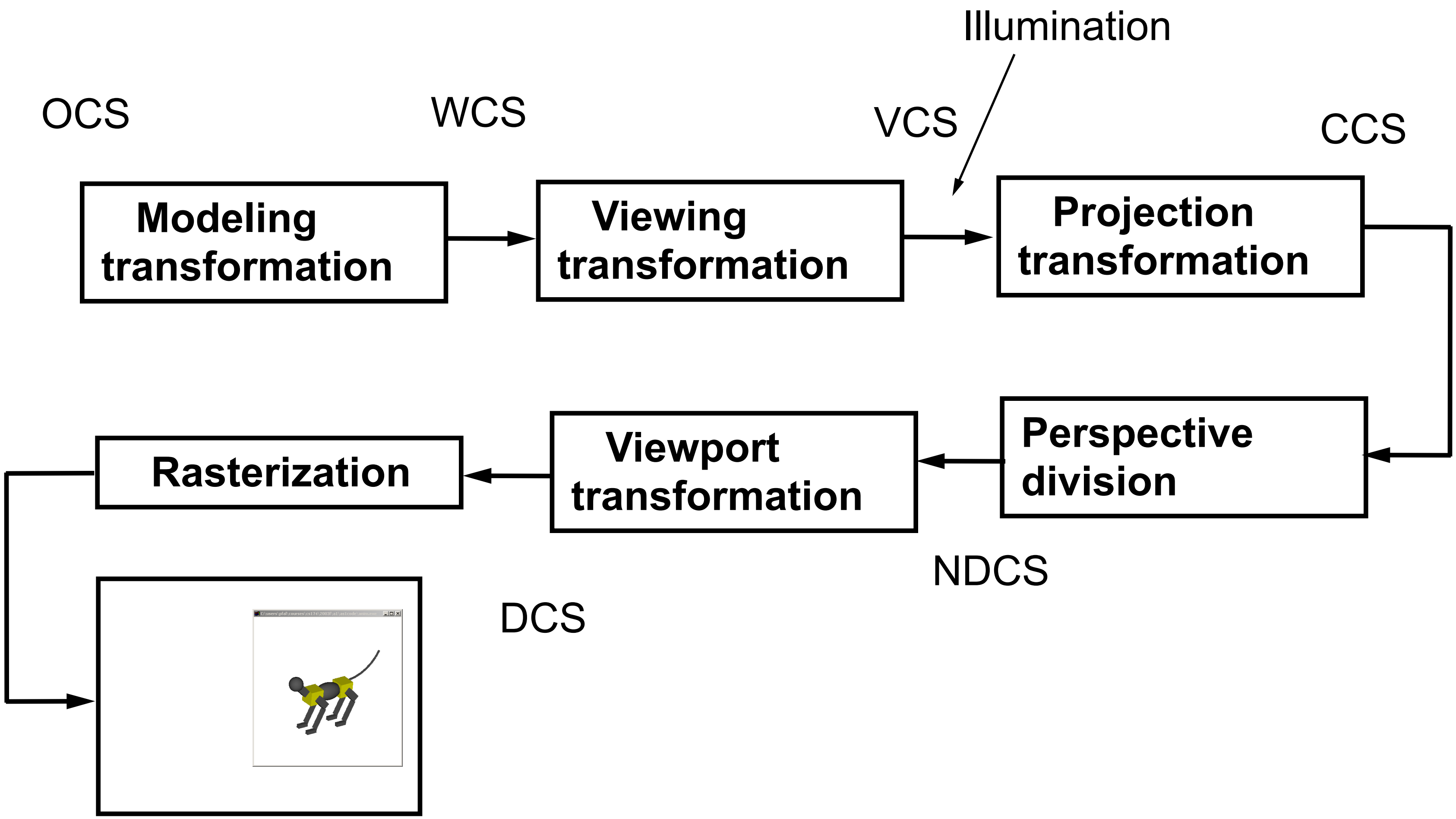


Z-buffer algorithm

for each polygon in model
  project vertices of polygon onto viewing plane
  for each pixel inside the projected polygon
    calculate pixel colour
    calculate pixel z-value
    compare pixel z-value to value stored for pixel in z-buffer
    if pixel is closer, draw it in frame-buffer and z-buffer
  end
end



COMPLETION OF Z-buffer Graphics Pipeline



What Have We Ignored?

Some local phenomena

- shadows — every point is illuminated by every light source
- attenuation — intensity falls off with square of distance to light
- transparent objects — light can be transmitted through surface

Global illumination

- reflections of objects in other objects
- indirect diffuse light — ambient term is just a hack

Realistic surface detail

- can make an orange sphere
- but it doesn’t have the texture of the real fruit

Realistic light sources

Global Illumination

Computing light interface between all surfaces

Radiosity

Ray tracing

Courtesy of Henrik Wann Jensen

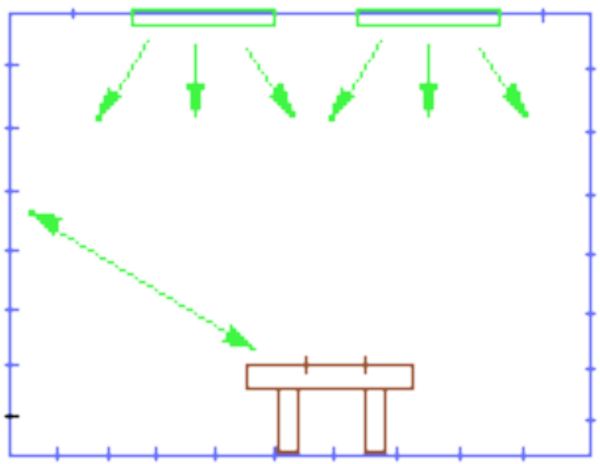
Radiosity (Hill: not covered. Foley & van Dam: Ch 16.13, p. 793-806)

Physics-based (heat transfer and illumination engineering)

Suited for Diffuse reflection

Infinite reflections

Soft shadows



Radiosity algorithm

Break scene into small patches

Assume uniform reflection and emission per patch

Energy balance for all patches:

Light leaving surface=emitted light + reflected light

Notation

- Flux: energy per unit time (W)
- Radiosity B: exiting flux density (W/m^2)
- E: exiting flux density for light sources
- Reflectivity R: fraction of incoming light reflected (unitless)
- Form factor Fij: fraction of energy leaving Ai and arriving at Aj determined by the geometry of polygons i and j

Energy balance

light leaving surface = emitted light + reflected light

$$B_i A_i = E_i A_i + R_i \sum_j B_j F_{ji} A_j$$

$$B_i = E_i + R_i \sum_j B_j F_{ji} \frac{A_j}{A_i}$$

form-factor reciprocity:

$$F_{ji} A_j = F_{ij} A_i$$

$$B_i = E_i + R_i \sum_j B_j F_{ij}$$

Linear system

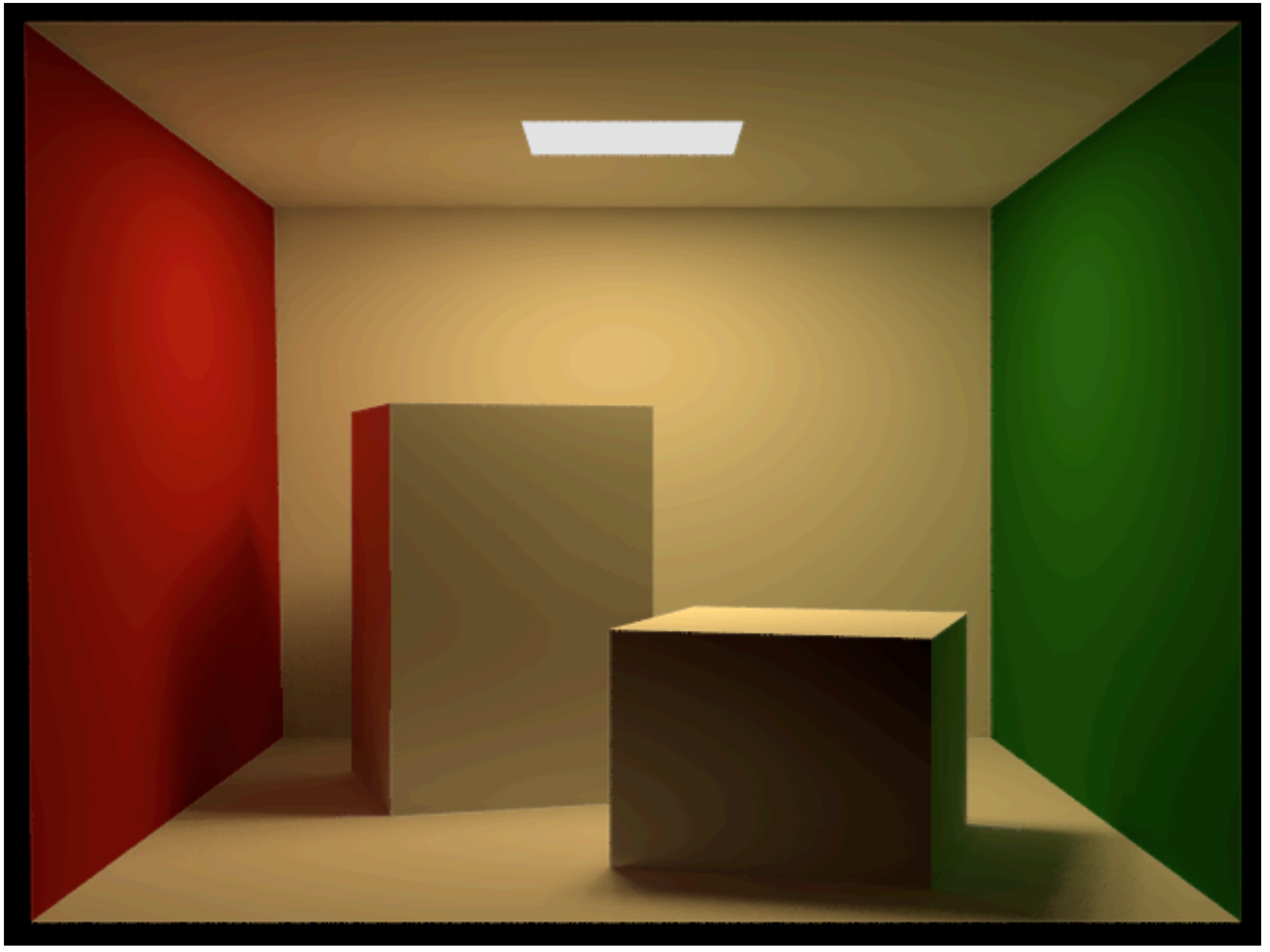
$$\begin{bmatrix} E_1 \\ E_2 \\ \cdots \\ E_n \end{bmatrix} = \begin{bmatrix} 1 - R_1 F_{11} & \cdots & -R_1 F_{1n} \\ -R_2 F_{21} & \cdots & -R_2 F_{2n} \\ \cdots & \cdots & \cdots \\ -R_n F_{n1} & \cdots & 1 - R_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdots \\ B_n \end{bmatrix}$$

Matrix o(n^2)

Form-factor computing

Constant radiosity patches

Example: The Cornell scene



Radiosity summary

Object space algorithm

Suited for diffuse reflections

Nice soft-shadows