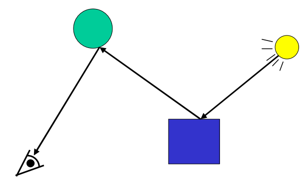
# Lighting & Shading

## Light Transport

- Goal: Model the interaction of light with matter in a way that appears realistic (and is fast)





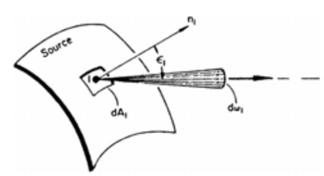
## Light Transport

- Assumptions:
  - Geometrical optics:
    - No diffraction, no polarization, no interference
  - Light travels in a straight line in a vacuum
    - No atmospheric scattering or refraction
    - No gravity effects
  - Discrete-wavelength approximation of color
    - Quantized approximation of dispersion and fluorescence
  - Superposition
    - No non-linear reflecting materials



## Bidirectional Reflectance

- Radiance L  [W/(sr m$^2$)]
  - radiant flux / (unit solid angle * unit area)

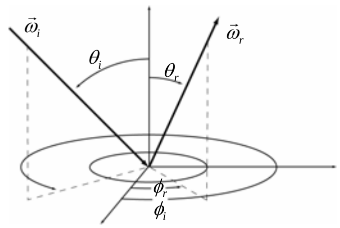


## Bidirectional Reflectance

- Single incoming direction and single outgoing direction

$$L_r(\vec{\omega}_i, \vec{\omega}_r) = f_r(\vec{\omega}_i, \vec{\omega}_r) L_i(\vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i$$

↖ bidirectional reflectance distribution function



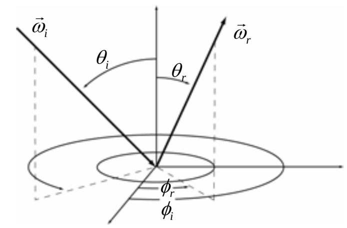



## Bidirectional Reflectance

- Integral over all incoming directions

$$L_r(\vec{\omega}_r) = \int_{\Omega_i} f_r(\vec{\omega}_i, \vec{\omega}_r) L_i(\vec{\omega}_i) \cos\theta_i \, \mathrm{d}\vec{\omega}_i$$

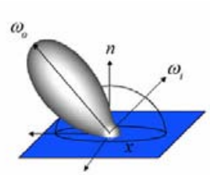
$\Omega_i$ ← incident hemisphere






## Bidirectional Reflectance

- Bidirectional Reflectance Distribution Function (BRDF)
  - anisotropic (4 DOFs) $f_r(\phi_i, \theta_i, \phi_r, \theta_r)$
  - isotropic (3DOFs) $f_r(\phi_i - \phi_r, \theta_i, \theta_r)$






## Bidirectional Reflectance

- Generalizations
  - anisotropic BRDF: 4 DOFs
  - spatially varying: +2
  - spectrally varying: +1
  - Fluorescence: +1
  - Phosphorescence: +1
  - Subsurface Scattering: +2

$\rightarrow$ General BRDF is 11 dimensional function!!

## Subsurface Scattering





## Global Illumination





## Complex Materials and Lighting



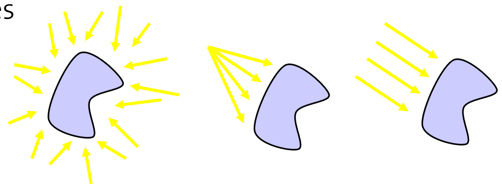


## Real-World vs. OpenGL

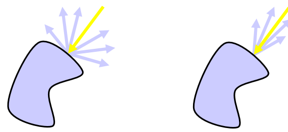
- Real world
  - complex computations
  - see optics textbooks, photorealistic rendering
- OpenGL
  - simplified model
  - ambient, diffuse and specular light sources and reflections
  - easy to tune
  - fast to compute

# Lighting

- Light sources



- Light reflection





# Ambient Light

- Scattered by environment
- Coming from all directions
- Reflection independent of
  - Camera position
  - Light position (no light position)
  - Surface orientation
- Reflected intensity: $I = I_a \, k_a$

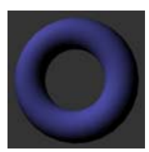
  ($I_a$: light source; $k_a$: material parameter)

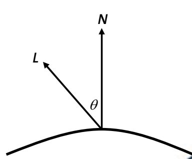


# Diffuse Reflection

- Directed light $I_p$
- Reflection dependent on
  - orientation of surface
  - light source position
- Independent of
  - camera position
    (reflected equally in all directions)
- Reflected intensity: $I = I_p \, k_d \cos\theta$

$$I = I_p \, k_d \, (\mathbf{N} \cdot \mathbf{L})$$



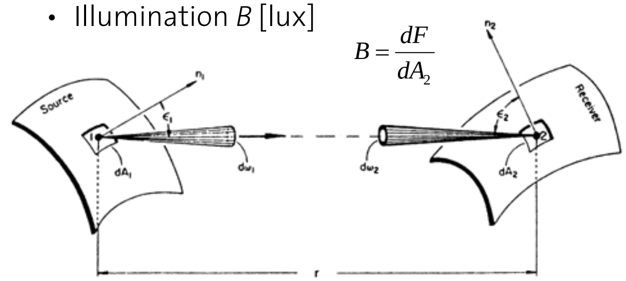





# Two Radiant Surface Patches

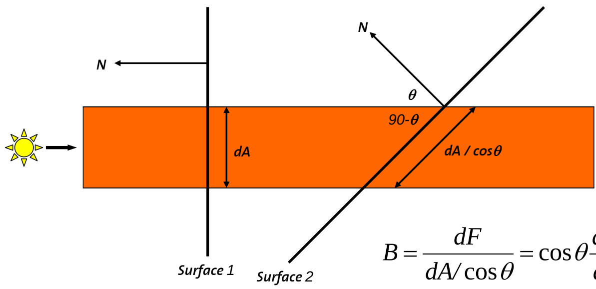
- Illumination $B$ [lux]

$$B = \frac{dF}{dA_2}$$



## Diffuse Reflection

- Diffuse reflection scales with angle




$$B = \frac{dF}{dA/\cos\theta} = \cos\theta \frac{dF}{dA}$$

*illumination*



## A Simple Model

- Sum up ambient light and diffuse reflection:

$$I = I_a\, k_a + I_p\, k_d\, (\mathbf{N} \cdot \mathbf{L})$$



## Attenuation

- Quadratic attenuation due to spatial radiation

$$f_{att} = \frac{1}{d_L^2}$$

- A model often used in Graphics (OpenGL)

$$f_{att} = \min\left(\frac{1}{c_1 + c_2\, d_L + c_3\, d_L^2}, 1\right)$$

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, c1);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, c2);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c3);
```

- Include attenuation

$$I = I_a\, k_a + f_{att}\, I_p\, k_d\, (\mathbf{N} \cdot \mathbf{L})$$



## Wavelength Dependency

- Colored light and reflection as functions of wavelength $\lambda$

$$I_\lambda = I_{a_\lambda}\, k_a\, O_{d_\lambda} + f_{att}\, I_{p_\lambda}\, k_d\, O_{d_\lambda}\, (\mathbf{N} \cdot \mathbf{L})$$

- Restriction to RGB (OpenGL)

$$I_R = I_{a_R}\, k_{a_R} + f_{att}\, I_{p_R}\, k_{d_R}\, (\mathbf{N} \cdot \mathbf{L})$$
$$I_G = I_{a_G}\, k_{a_R} + f_{att}\, I_{p_G}\, k_{d_G}\, (\mathbf{N} \cdot \mathbf{L})$$
$$I_B = I_{a_B}\, k_{a_R} + f_{att}\, I_{p_B}\, k_{d_B}\, (\mathbf{N} \cdot \mathbf{L})$$

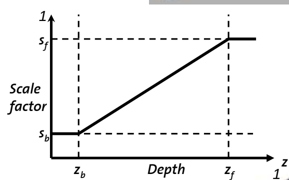
***Restriction of spectral sampling to RGB can lead to substantial color artifacts***

## Depth Cueing (Fog)

- Linear depth cue by blending with the color of the participating medium

$$I_\lambda = sI_\lambda + (1-s)I_{dc_\lambda}$$

- s is scaling factor



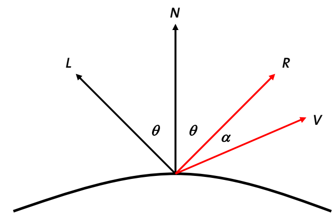

$$s = s_b + \frac{(z-z_b)(s_f-s_b)}{z_f-z_b}$$

for $z_b \le z \le z_f$



## Specular Reflection

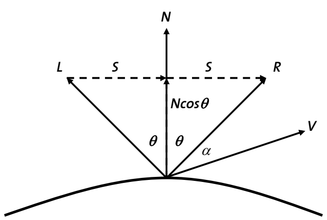
- Depends on angle between reflection and viewing ray






## Reflected Ray

- Computed using simple vector algebra




$$\mathbf{R} = \mathbf{N}\cos\theta + \mathbf{S}$$

$$\mathbf{R} = 2\mathbf{N}\cos\theta - \mathbf{L} = 2\mathbf{N}(\mathbf{N}\cdot\mathbf{L}) - \mathbf{L}$$

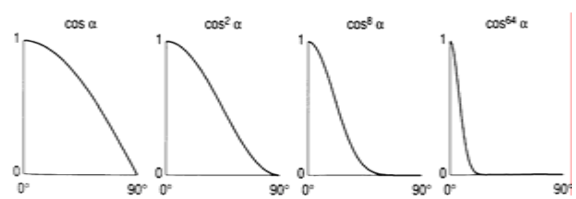
$$\cos\alpha = \mathbf{R}\cdot\mathbf{V} = (2\mathbf{N}(\mathbf{N}\cdot\mathbf{L}) - \mathbf{L})\cdot\mathbf{V}$$



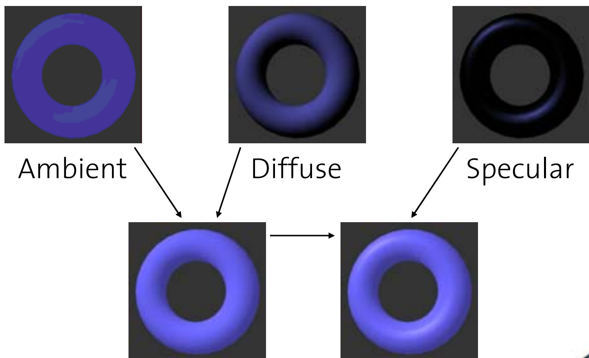
## Phong Illumination Model

- Approximates specular reflection by cosine powers

$$I_\lambda = I_{a_?}\, k_a O_{d_\lambda} + f_{att} I_{p_\lambda}[k_d O_{d_\lambda}(\mathbf{N}\cdot\mathbf{L}) + k_s(\mathbf{R}\cdot\mathbf{V})^n]$$



## Ambient + Diffuse + Specular





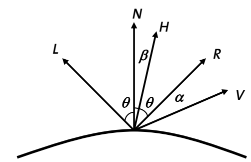

## Extensions

- Specular colors

$$I_\lambda = I_{a_\lambda} k_a O_{d_\lambda} + f_{att} I_{p_\lambda} [k_d O_{d_\lambda} (\mathbf{N} \cdot \mathbf{L}) + k_s O_{s_\lambda} (\mathbf{R} \cdot \mathbf{V})^n]$$

- Halfway-Vector (faster)

$$\cos^n \beta = (\mathbf{N} \cdot \mathbf{H})^n \quad \mathbf{H} = \frac{\mathbf{L}+\mathbf{V}}{|\mathbf{L}+\mathbf{V}|}$$



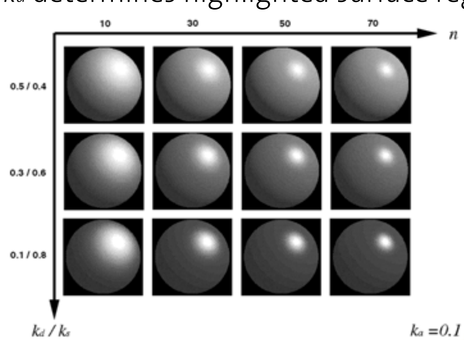

- Multiple Light Sources

$$I_\lambda = I_{a_\lambda} k_a O_{d_\lambda} + \sum_{1 \le i \le m} f_{att_i} I_{p_{\lambda i}} [k_d O_{d_\lambda} (\mathbf{N} \cdot \mathbf{L}_i) + k_s O_{s_\lambda} (\mathbf{R}_i \cdot \mathbf{V})^n]$$



## Highlights

- $k_s/k_d$ determines highlighted surface regions






## OpenGL

- Light source:

```
GLfloat light_pos[] = {x,y,z, 1};
GLfloat light_Ia[] = {IaR, IaG, IaB, IaA};
GLfloat light_Id[] = {IdR, IdG, IdB, IdA};
GLfloat light_Is[] = {IsR, IsG, IsB, IsA};
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ia);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Id);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Is);
```

- Reflection (material):

```
GLfloat material_Ka[] = {kaR, kaG, kaB, kaA};
GLfloat material_Kd[] = {kdR, kdG, kdB, kdA};
GLfloat material_Ks[] = {ksR, ksG, ksB, ksA};
glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_SHININESS, Se);
```

## Light Position

- Position in world-coordinates
- Light stationary

```
gluLookAt(eyeX, eyeY, eyeZ,
          centerX, centerY, centerZ,
          upX, upY, upZ);
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

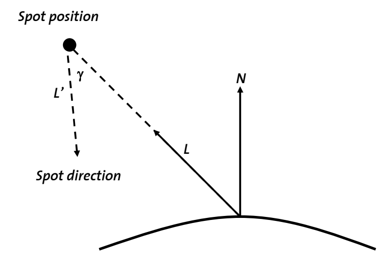
- Position in camera-coordinates
- Light moving with camera (headlight)

```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
gluLookAt(eyeX, eyeY, eyeZ,
          centerX, centerY, centerZ,
          upX, upY, upZ);
```



## Light Sources

- Directed Spotlights




$$I_{L'_\lambda} \cos^p \lambda$$

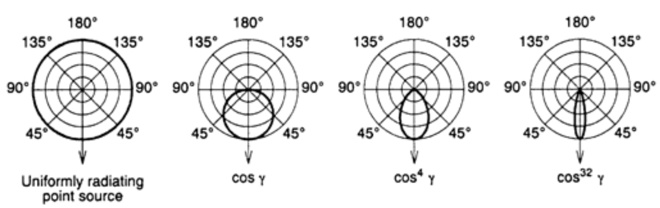
$$I_{L'_\lambda} (-L \cdot L')^p$$

```
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, L');
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, p);
```



## Intensity Diagrams

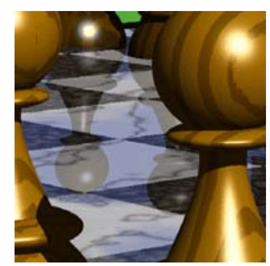
- Implemented in most modern graphics APIs






## Shading

- Shading requires many evaluations of a lighting model
- Questions:
  - where to evaluate ?
  - when to evaluate ?



## *Shading Models*

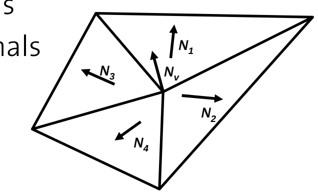
- ***Flat shading*** (constant shading)
  - one color per primitive
- ***Gouraud shading***
  - linear interpolation of vertex intensities
- ***Phong shading***
  - linear interpolation of vertex normals
- Gouraud & Phong shading need vertex normals



## *Gouraud Shading* – Procedure

1. Calculate face normals
2. Calculate vertex normals

$$N_v = \frac{\sum_{i=1}^{n} N_i}{\left|\sum_{i=1}^{n} N_i\right|}$$

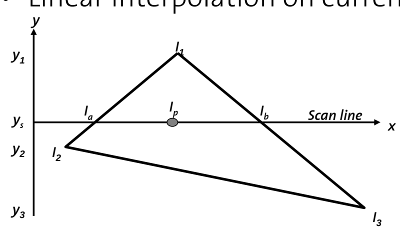



3. Evaluate lighting model for each vertex
4. Interpolate vertex intensities across primitive (bilinear interpolation)



## *Gouraud Shading* – Interpolation

- Linear interpolation on current scan line




$$I_a = I_1 - (I_1 - I_2)\,\frac{(y_1 - y_s)}{(y_1 - y_2)}$$

$$I_b = I_1 - (I_1 - I_3)\,\frac{(y_1 - y_s)}{(y_1 - y_3)}$$

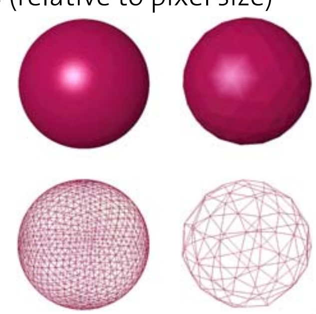
$$I_p = I_b - (I_b - I_a)\,\frac{(x_b - x_p)}{(x_b - x_a)}$$

- Done during scan conversion
- Hardware acceleration possible



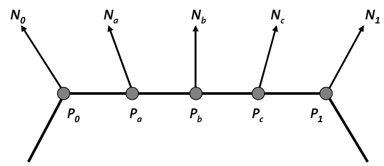
## *Gouraud Shading* – Quality

- Shading quality depends on size of projected primitives (relative to pixel size)



## *Phong Shading*

- Linear interpolation of normals (not intensities) on current scan line




- Evaluation of lighting model at each pixel
- More accurate but slower than Gouraud



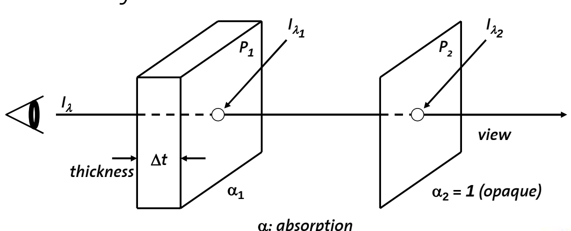
## *Shading Methods* – Overview

- Shading in image space:
  - ***Flat shading*** (*Constant shading*)
  - ***Gouraud shading***
- Shading in object / screen space:
  - ***Phong shading***
- Impact on graphics hardware...



## *Transparency*

- Transparency done with $\alpha$-blending...
- Linearizing exponential attenuation of intensity in media






## *Transparency* – Basic Laws

- P1 filters intensity according to $I'_{\lambda_2} = I_{\lambda_2} e^{-\alpha_1 \Delta t}$
- Linearization yields $I'_{\lambda_2} = I_{\lambda_2}(1-\alpha_1 \Delta t)$
- Emission of $P_1$ $I'_{\lambda_1} = I_{\lambda_1} \alpha_1 \Delta t$
- Summing up (for unit length)

$$I_\lambda = I'_{\lambda_1} + I'_{\lambda_2} = I_{\lambda_1}\alpha_1 + I_{\lambda_2}(1-\alpha_1)$$

- For N polygons (***$\alpha$-blending***)

$$I_\lambda = \sum_{i=1}^{N} \alpha_i I_{\lambda_i} \cdot \prod_{b=1}^{i-1} (1-\alpha_b)$$