



# Illumination

Tom Funkhouser  
Princeton University  
COS 426, Spring 2006

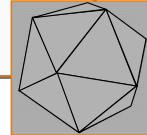


# Ray Casting

```

Image RayCast(Camera camera, Scene scene, int width, int height)
{
  Image image = new Image(width, height);
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
      Ray ray = ConstructRayThroughPixel(camera, i, j);
      Intersection hit = FindIntersection(ray, scene);
      image[i][j] = GetColor(scene, ray, hit);
    }
  }
  return image;
}

```



Wireframe

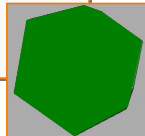


# Ray Casting

```

Image RayCast(Camera camera, Scene scene, int width, int height)
{
  Image image = new Image(width, height);
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
      Ray ray = ConstructRayThroughPixel(camera, i, j);
      Intersection hit = FindIntersection(ray, scene);
      image[i][j] = GetColor(scene, ray, hit);
    }
  }
  return image;
}

```



Without Illumination

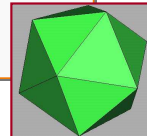


# Ray Casting

```

Image RayCast(Camera camera, Scene scene, int width, int height)
{
  Image image = new Image(width, height);
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
      Ray ray = ConstructRayThroughPixel(camera, i, j);
      Intersection hit = FindIntersection(ray, scene);
      image[i][j] = GetColor(scene, ray, hit);
    }
  }
  return image;
}

```



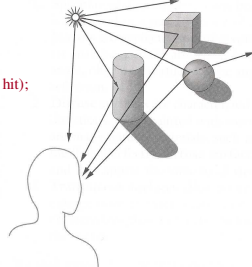
With Illumination



# Illumination

- How do we compute radiance for a sample ray?

```
image[i][j] = GetColor(scene, ray, hit);
```

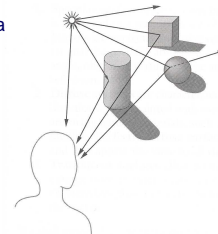


Angel Figure 6.2



# Goal

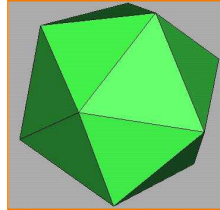
- Must derive computer models for ...
  - Emission at light sources
  - Scattering at surfaces
  - Reception at the camera
- Desirable features ...
  - Concise
  - Efficient to compute
  - "Accurate"



## Overview



- Direct Illumination
  - Emission at light sources
  - Scattering at surfaces
- Global illumination
  - Shadows
  - Refractions
  - Inter-object reflections

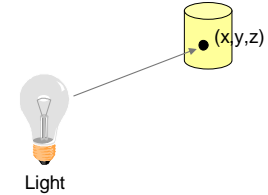


Direct Illumination

## Emission at Light Sources



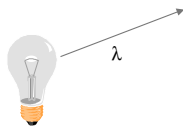
- $I_L(x,y,z,\theta,\phi,\lambda)$  ...
  - describes the intensity of energy,
  - leaving a light source, ...
  - arriving at location  $(x,y,z)$ , ...
  - from direction  $(\theta,\phi)$ , ...
  - with wavelength  $\lambda$



## Empirical Models



- Ideally measure irradiant energy for "all" situations
  - Too much storage
  - Difficult in practice



## OpenGL Light Source Models



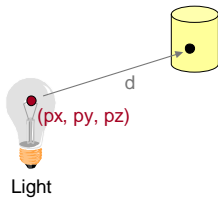
- Simple mathematical models:
  - Point light
  - Directional light
  - Spot light



## Point Light Source



- Models omni-directional point source
  - intensity ( $I_0$ ),
  - position  $(px, py, pz)$ ,
  - factors  $(k_c, k_l, k_q)$  for attenuation with distance ( $d$ )

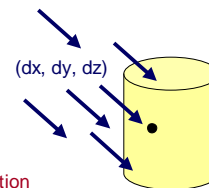


$$I_L = \frac{I_0}{k_c + k_l d + k_q d^2}$$

## Directional Light Source



- Models point light source at infinity
  - intensity ( $I_0$ ),
  - direction  $(dx, dy, dz)$



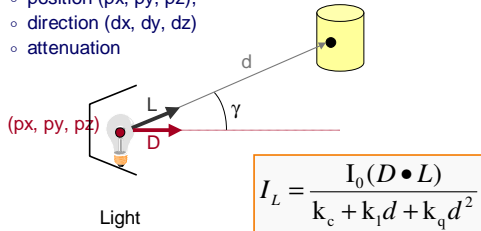
No attenuation with distance

$$I_L = I_0$$

## Spot Light Source

- Models point light source with direction

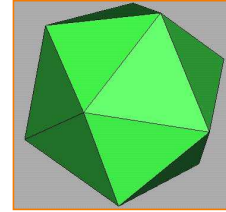
- intensity ( $I_0$ ),
- position ( $px, py, pz$ ),
- direction ( $dx, dy, dz$ )
- attenuation



$$I_L = \frac{I_0(D \cdot L)}{k_c + k_l d + k_q d^2}$$

## Overview

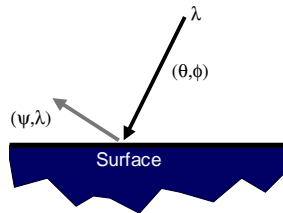
- Direct Illumination
  - Emission at light sources
  - Scattering at surfaces
- Global illumination
  - Shadows
  - Refractions
  - Inter-object reflections



Direct Illumination

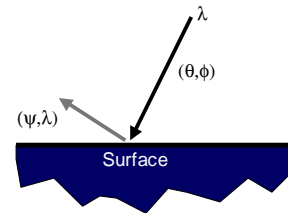
## Scattering at Surfaces

- $R_s(\theta, \phi, \gamma, \psi, \lambda)$  ...
  - describes the amount of incident energy,
  - arriving from direction  $(\theta, \phi)$ , ...
  - leaving in direction  $(\gamma, \psi)$ , ...
  - with wavelength  $\lambda$



## Empirical Models

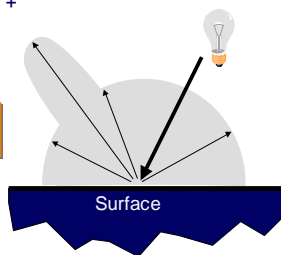
- Ideally measure radiant energy for "all" combinations of incident angles
  - Too much storage
  - Difficult in practice



## OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"

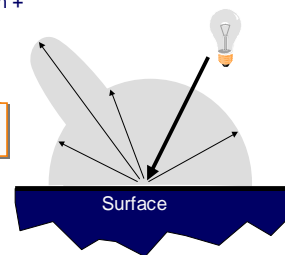
Based on model proposed by Phong



## OpenGL Reflectance Model

- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"

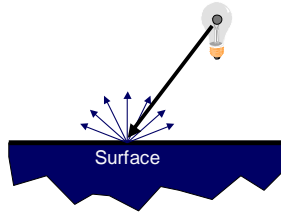
Based on model proposed by Phong



## Diffuse Reflection



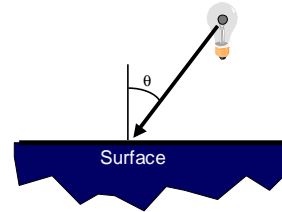
- Assume surface reflects equally in all directions
  - Examples: chalk, clay



## Diffuse Reflection



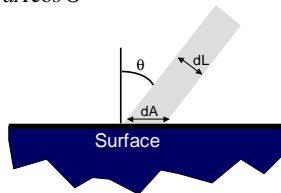
- How much light is reflected?
  - Depends on angle of incident light



## Diffuse Reflection



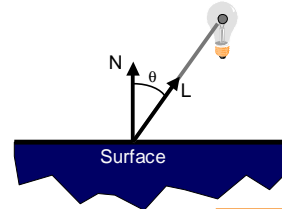
- How much light is reflected?
    - Depends on angle of incident light
- $$dL = dA \cos \Theta$$



## Diffuse Reflection



- Lambertian model
  - cosine law (dot product)

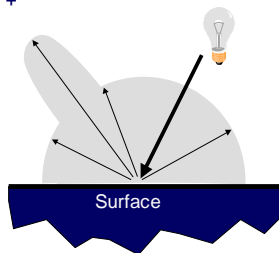


$$I_D = K_D (N \cdot L) I_L$$

## OpenGL Reflectance Model



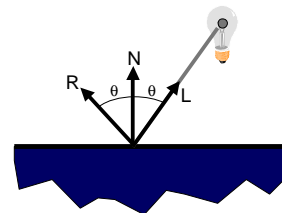
- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"



## Specular Reflection



- Reflection is strongest near mirror angle
  - Examples: mirrors, metals



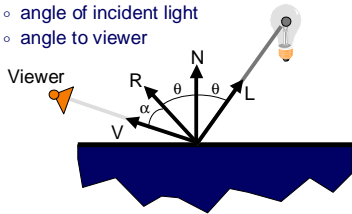
## Specular Reflection



How much light is seen?

Depends on:

- angle of incident light
- angle to viewer



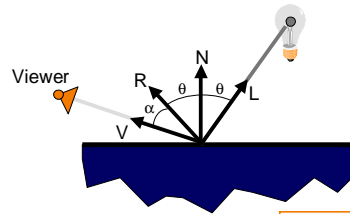
## Specular Reflection



- Phong Model

- $\cos(\alpha)^n$

This is a physically-motivated hack!



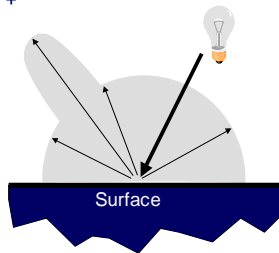
$$I_s = K_s (V \cdot R)^n I_L$$

## OpenGL Reflectance Model



- Simple analytic model:

- diffuse reflection +
- specular reflection +
- emission +
- "ambient"



## Emission



- Represents light emanating directly from polygon

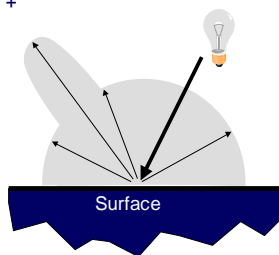


## OpenGL Reflectance Model



- Simple analytic model:

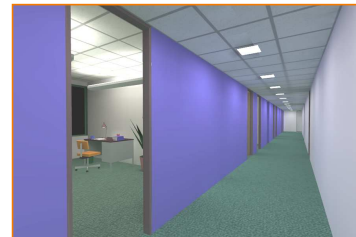
- diffuse reflection +
- specular reflection +
- emission +
- "ambient"



## Ambient Term



- Represents reflection of all indirect illumination

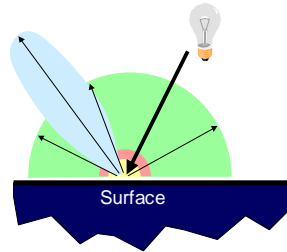


This is a total hack (avoids complexity of global illumination)!

## OpenGL Reflectance Model



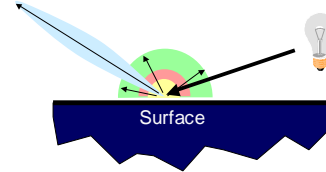
- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"



## OpenGL Reflectance Model



- Simple analytic model:
  - diffuse reflection +
  - specular reflection +
  - emission +
  - "ambient"



## OpenGL Reflectance Model



- Sum diffuse, specular, emission, and ambient

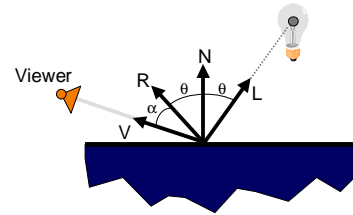
Phong	$\rho_{diffuse}$	$\rho_{specular}$	$\rho_{specular}$	$\rho_{total}$
$\phi = 60^\circ$				
$\phi = 25^\circ$				
$\phi = 0^\circ$				

Leonard McMillan, MIT

## Direct Illumination Calculation



- Single light source:

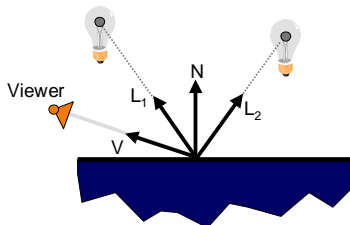


$$I = I_E + K_A I_{AL} + K_D (N \cdot L) I_L + K_S (V \cdot R)^n I_L$$

## Direct Illumination Calculation



- Multiple light sources:



$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i)$$

## Example



Luxo, Jr. (Pixar Animation Studios)

## Overview



- Direct Illumination
  - Emission at light sources
  - Scattering at surfaces
- Global illumination
  - Shadows
  - Transmissions
  - Inter-object reflections



Global Illumination

## Global Illumination

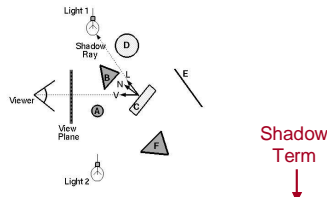


Greg Ward

## Shadows



- Shadow term tells if light sources are blocked
  - Cast ray towards each light source  $L_i$
  - $S_i = 0$  if ray is blocked,  $S_i = 1$  otherwise

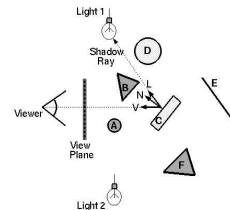


$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L$$

## Ray Casting (last lecture)



- Trace primary rays from camera
  - Direct illumination from unblocked lights only

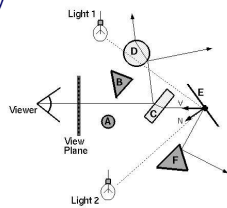


$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L$$

## Recursive Ray Tracing



- Also trace secondary rays from hit surfaces
  - Global illumination from mirror reflection and transparency

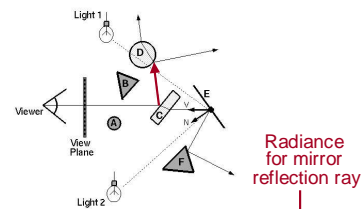


$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

## Mirror reflections



- Trace secondary ray in mirror direction
  - Evaluate radiance along secondary ray and include it into illumination model

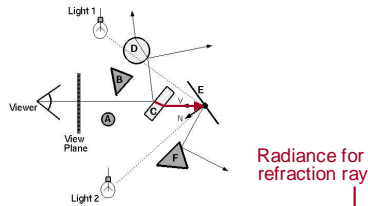


$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

## Transparency



- Trace secondary ray in direction of refraction
  - Evaluate radiance along secondary ray and include it into illumination model



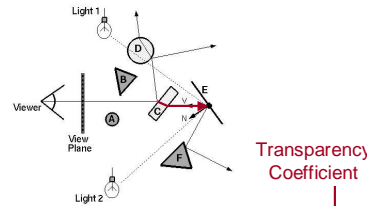
Radiance for refraction ray

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

## Transparency



- Transparency coefficient is fraction transmitted
  - $K_T = 1$  for translucent object,  $K_T = 0$  for opaque
  - $0 < K_T < 1$  for object that is semi-transparent



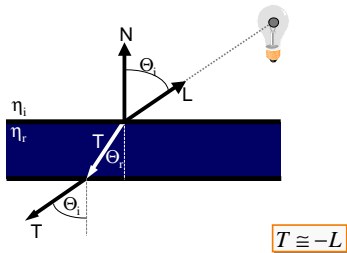
Transparency Coefficient

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

## Refractive Transparency



- For thin surfaces, can ignore change in direction
  - Assume light travels straight through surface



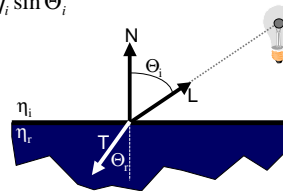
$$T \cong -L$$

## Refractive Transparency



For solid objects, apply Snell's law:

$$\eta_r \sin \Theta_r = \eta_i \sin \Theta_i$$

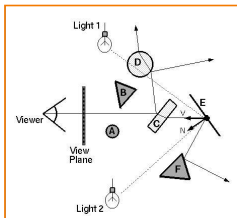


$$T = \left( \frac{\eta_i}{\eta_r} \cos \Theta_i - \cos \Theta_r \right) N - \frac{\eta_i}{\eta_r} L$$

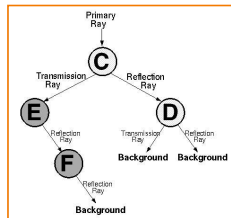
## Recursive Ray Tracing



- Ray tree represents illumination computation



Ray traced through scene



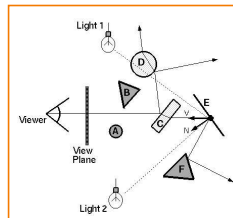
Ray tree

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

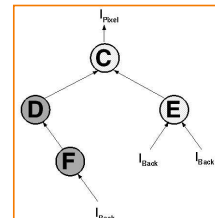
## Recursive Ray Tracing



- Ray tree represents illumination computation



Ray traced through scene



Ray tree

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_S I_R + K_T I_T$$

## Recursive Ray Tracing



- GetColor is called recursively

```
Image RayTrace(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            image[i][j] = GetColor(scene, ray);
        }
    }
    return image;
}
```

## Recursive Ray Tracing



- GetColor is called recursively

```
Color GetColor(Scene scene, Ray ray)
{
    Intersection hit = FindIntersection(ray, scene);
    Ray specular_ray = SpecularRay(ray, hit);
    Ray refractive_ray = RefractiveRay(ray, hit);
    Color color = Phong(scene, ray, hit) +
        Ks * GetColor(scene, specular_ray, hit) +
        Kt * GetColor(scene, refractive_ray, hit);
    return color;
}
```

## Summary



- Ray casting (direct illumination)
  - Usually use simple analytic approximations for light source emission and surface reflectance
- Recursive ray tracing (global illumination)
  - Incorporate shadows, mirror reflections, and pure refractions

All of this is an approximation  
so that it is practical to compute

More on global illumination later!

## Illumination Terminology



- Radiant power [flux] ( $\Phi$ )
  - Rate at which light energy is transmitted (in Watts).
- Radiant Intensity ( $I$ )
  - Power radiated onto a unit solid angle in direction (in Watts/sr)
    - » e.g.: energy distribution of a light source (inverse square law)
- Radiance ( $L$ )
  - Radiant intensity per unit projected surface area (in Watts/m<sup>2</sup>sr)
    - » e.g.: light carried by a single ray (no inverse square law)
- Irradiance ( $E$ )
  - Incident flux density on a locally planar area (in Watts/m<sup>2</sup>)
    - » e.g.: light hitting a surface at a point
- Radiosity ( $B$ )
  - Exitant flux density from a locally planar area (in Watts/m<sup>2</sup>)