Rendering Techniques

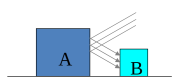
Chapter 8

Rendering

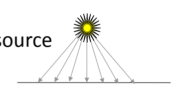
- Rendering: creating the effects of light on surfaces
- Determined from reflected light
  - shape
  - texture
  - color
- Photorealism is currently impossible to duplicate, but very good approximations are possible

Sources of Light

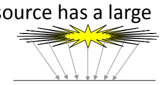
- Emitted light
  - directly from light source



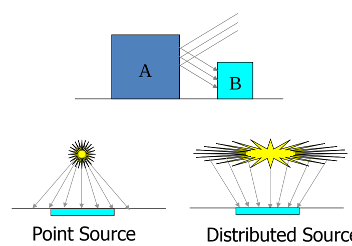
- Ambient (background) light
- Point light source
  - All the light comes from a single source



- Distributed light source
  - Light has many sources, or the source has a large area

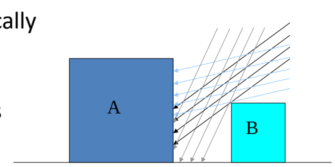


Sources of Light

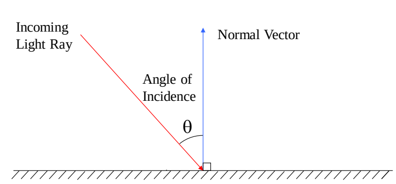


Light Sources

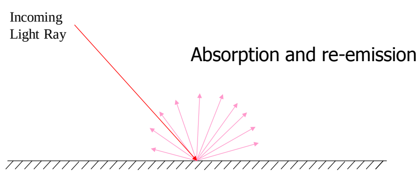
- Point light sources cast distinct shadows
- Distributed light sources (also called "soft" light) produce no distinct shadows
- A scene is typically illuminated by several kinds of light sources



Reflected Light

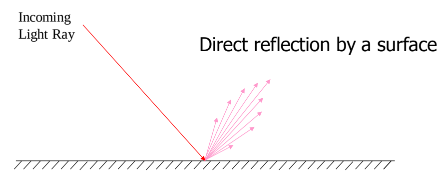


Diffuse Reflection



Diffuse reflection reveals texture and color

Specular Reflection



- Shiny surfaces have bright specular highlights
- Rough surfaces have few or no highlights

Reflected Light Intensity

- Light from a typical surface is the sum of ambient and direct (diffuse and specular) light

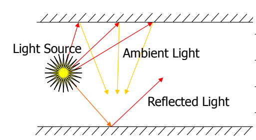
$I = R_a + R_d + R_s$

R_a = reflected ambient light

R_d = direct light reflected diffusely

R_s = specular reflection

Reflected Light Intensity



$I = R_a + R_d + R_s$

R_a = reflected ambient light

R_d = direct light reflected diffusely

R_s = specular reflection

Ambient Light Intensity

$R_a = k_d I_a$

k_d = Coefficient of Reflectivity

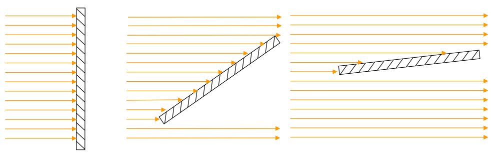
(white = 1, black = 0)

Direct Light Intensity

- Computing the contribution from direct light sources takes a little more work

Lambert's Cosine Law

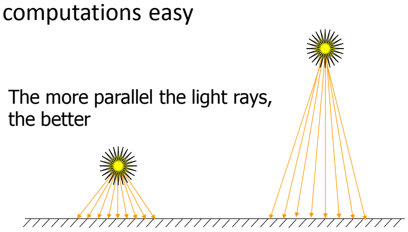
- Intensity of diffuse reflection is directly proportional to the cosine of the angle of incidence



Distant Light Sources

- Light source should be far away to make computations easy

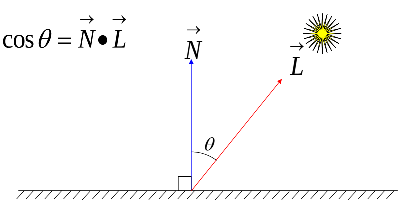
The more parallel the light rays, the better



Intensity Computation

- For distant point sources

$$\cos\theta = \vec{N} \bullet \vec{L}$$



Intensity Computation

$$I_p = \frac{E_p}{d^2}$$

$d =$ distance to point light source

$E_p =$ power of light source

Inverse square law does not give realistic renderings; divide by d instead

Intensity Computation

- Inverse square law does not give realistic renderings; divide by d instead

$$R_d = \frac{K_d E_p}{d + d_0}(\vec{N} \bullet \vec{L})$$

$d_0 =$ a constant to adjust the intensity to make the image look realistic prevent division by zero

Intensity Computation

- Total diffuse reflection is the sum of diffusely reflected direct and ambient light

$$R_{td} = k_d I_a + \frac{k_d E_p}{d + d_0}(N \bullet \vec{L})$$

- Both terms depend on K_d

Color

- Color is determined mostly from diffuse reflection
- The formula for red:

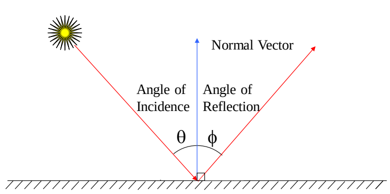
$$R_{td_r} = k_{d_r} I_{a_r} + \frac{k_{d_r} E_{p_r}}{d + d_0} (\vec{N} \bullet \vec{L})$$

- Replace the *r* subscript with *g* and *b* for the green and blue versions

Specular Reflection

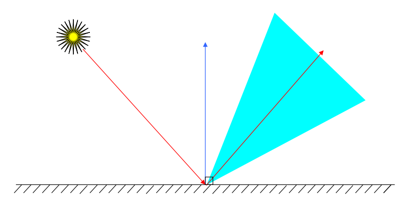
- While diffuse reflection determines color, shine and shape is determined largely from specular highlights
- Smooth surfaces have specular highlights; rough surfaces do not
- Flat surfaces usually have larger specular highlights than curved surfaces

Dull vs. Shiny Surfaces

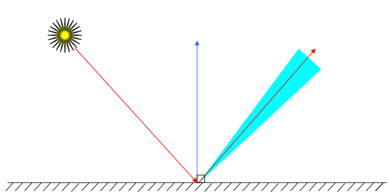


$$\theta = \phi$$

Dull Surface

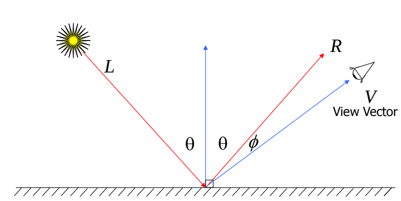


Shiny Surface



- The more narrow the area of reflection, the closer the surface is to a mirror

View Vector



- V is the view vector
- Specular reflection is visible when V is close to R

Phong Model

- In the Phong model, intensity of light along a vector at angle ϕ from angle of relection is proportional to $\cos^n \phi$ where n is the shininess of the surface
- $\phi = 0$ implies $\cos^n \phi = 1$
  thus the specular reflection is greatest at the angle of reflection

Phong Model

- n = 1 implies intensity decreases gradually as $\phi = \pm \frac{\pi}{2}$
- Surfaces with n = 1 are dull surfaces

Phong Model

- Metallic surfaces have large n values (150+)
- Intensity of specular reflection drops off rapidly as the line of sight moves away from the angle of reflection
- Shiny surfaces have smaller, more intense highlights

Specular Reflection

$$R_s = \frac{E_p}{d + d_0} w(\theta, \lambda) \cos^n \phi$$

$w(\theta, \lambda)$ may be replaced by a constant K_s

$\cos \phi = \vec{V} \bullet \vec{R}$ (if $\vec{V}$ and $\vec{R}$ are normalized)

$$R_s = \frac{E_p}{d + d_0} K_s (\vec{V} \bullet \vec{R})^n$$

Comprehensive Formula

$$I_r = I_{a_r} K_{d_r} + \underbrace{I_{p_r} (K_{d_r} \frac{\cos \theta}{d} + K_s \cos^n \phi)} + \ldots$$

I_r = Intensity of red component of reflected light

I_{a_r} = Intensity of red component of ambient light

K_{d_r} = Diffuse reflectivity constant for red

Comprehensive Formula

$$I_r = I_{a_r} K_{d_r} + \underbrace{I_{p_r} (K_{d_r} \frac{\cos \theta}{d} + K_s \cos^n \phi)} + \ldots$$

θ = Angle of incidence (between normal and light source

d = Distance of point source from surface

K_s = Specular reflection coefficient (ratio of specularly reflected light to incident light

Comprehensive Formula

$$I_r = I_{a_r} K_{d_r} + \underbrace{I_{p_r}(K_{d_r}\frac{\cos\theta}{d} + K_s \cos^n \phi)}+ \ldots$$

$n =$ Phong's shininess constant for a surface

$\phi =$ Angle in which the direction of reflected light deviates from the precise angle of reflection

Comprehensive Formula

$$I_r = I_{a_r} K_{d_r} + \underbrace{I_{p_r}(K_{d_r}\frac{\cos\theta}{d} + K_s \cos^n \phi)}+ \ldots$$

Comprehensive Formula

$$I_r = I_{a_r} K_{d_r} + \underbrace{I_{p_r}(K_{d_r}\frac{\cos\theta}{d} + K_s \cos^n \phi)}+ \ldots$$

- The second term can be duplicated for additional point sources or distributed sources
- I_r could add up to > 1; must adjust

Reflection and Transparency

- Light striking a surface can be
  - reflected
  - absorbed
  - transmitted
- Usually some combination of these three activities takes place

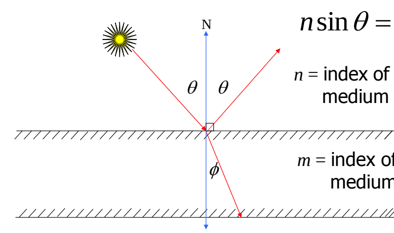
Reflection and Transparency

- Opaque materials completely absorb and/or reflect all light which strikes them
- Transparent objects allow light to pass through
  - The light will dim as it passes through

Snell's Law

- Snell's Law of refraction

$$n \sin\theta = m \sin\phi$$



n = index of refraction in medium 1

m = index of refraction in medium 2

Transmitted Light

$$I = tI_f + (1-t)I_b$$

$0 \le t \le 1$

$t =$ Transparency of front object

$0 =$ Total transparency

$1 =$ Total opacity

Ray Tracing

- Also called *recursive ray tracing*
- Good for complicated reflections, refractions, even dull surfaces
- Very versatile--used frequently in solid modeling

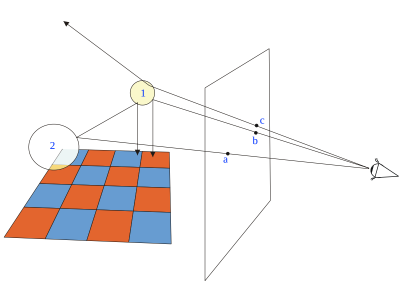
Ray Tracing

- Color is a ray of light entering our eyes from an initial source
- The ray may have a complicated history of reflections and refractions

Ray Tracing

- Ray tracing follows a ray of light backwards from our eye to the initial light source (or as far backwards as is practical)
- Each pixel on the screen corresponds to a light ray

Ray Tracing



Ray Tracing Algorithm

1. For a given pixel (x_s, y_s) follow the ray from the center of projection through this pixel.
2. Check all objects in the scene to see if they intersect with this ray

   2.1 If there are intersections, choose the one closest when travelling along the ray. This intersection is the "hit," and the object encountered is the "hit object."

Algorithm (cont.)

2.2 If there are no intersections (no "hit"), then the object is the background

3. If the hit object is opaque, set (x_s, y_s) to the color of the object at this point (include production of shadows). Consider next pixel, go to Step 1.

Algorithm (cont.)

4. If the hit object is shiny, compute the reflection vector and follow the ray from the hit point along the reflection vector; go to Step 2.
   If the hit object is transparent, compute the refraction vector and follow the ray from the hit point along this vector; go to Step 2.
   If there are too many reflections in succession, set pixel to background; go to Step 1.

Ray Tracing Pseudocode

```
RayTrace() {
    Select center of projection c;
    Select viewport into the virtual scene;
    for ( each scan line s in image ) {
        for ( each pixel x in s ) {
            r ← ray from c passing through x;
            x.color ← trace(r, 1);
        }
    }
}
```


Ray Tracing Pseudocode

```
trace(ray, depth) {
    obj ← closest object that intersects ray;
    if ( obj exists ) {
        pt ← point of intersection;
        N ← normal at intersection;
        return shade(obj, ray, pt, N, depth);
    } else {
        return backgroundColor;
    }
}
```


Ray Tracing Pseudocode

```
shade(obj, ray, pt, N, depth) {
    color ← ambientColor;
    for ( each light lt ) {
        sRay ← ray to lt from pt;   L ← vector to lt;
        if ( N • L > 0 ) {
            Compute how much light is blocked from opaque and
            transparent surfaces, and use to scale diffuse and
            specular terms before adding them to color;
        }
    }
    // Continued
```


Ray Tracing Pseudocode

```
    if ( depth < MAX_DEPTH ) {
        if ( obj is reflective ) {
            rRay ← ray in reflection direction from pt;
            color ← color + trace(rRay, depth + 1) × K_s ;
        }
        if ( obj is transparent ) {
            tRay ← ray in refraction direction from pt;
            if ( total internal reflection does not occur )
                color ← color + trace(tRay, depth + 1) × K_t ;
        }
    }
    return color;
}
```