Global Illumination

Michael Kazhdan

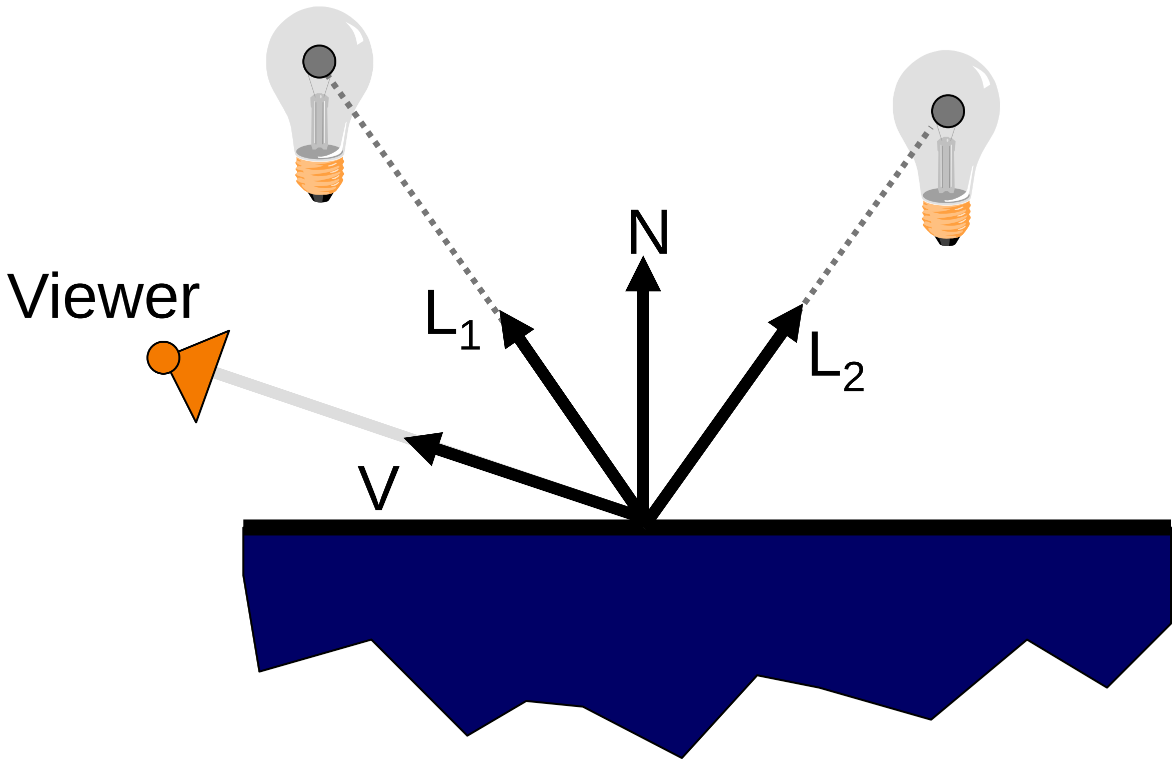
(600.357 / 600.457)

HB Ch. 14.1, 14.2

FvDFH 16.1, 16.2

Surface Illumination Calculation

- Multiple light sources:



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

Overview

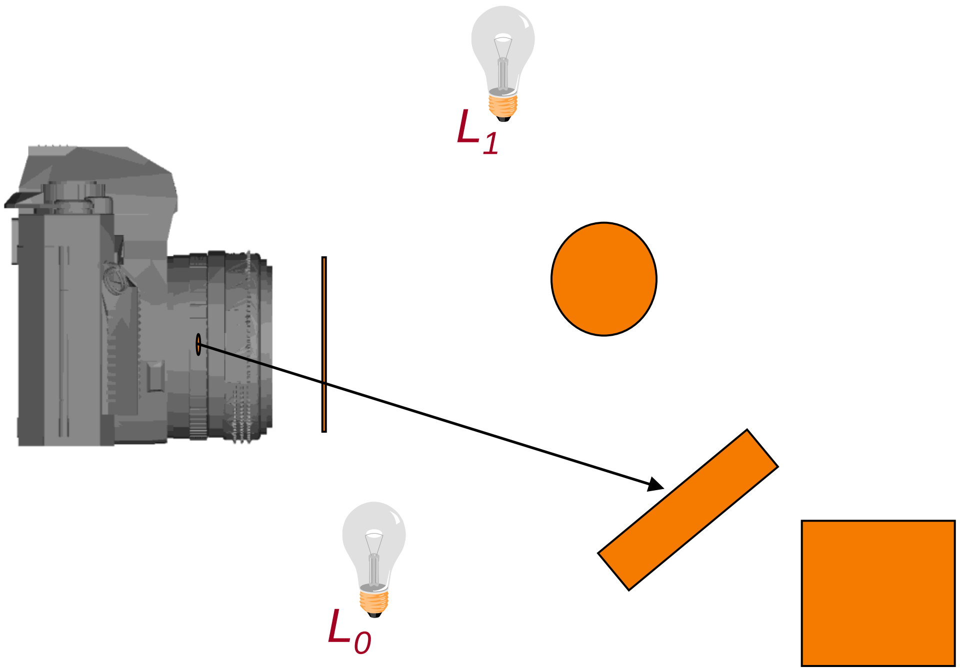
- Direct Illumination
  - Emission at light sources
  - Direct light at surface points
- Global illumination
  - Shadows
  - Transmissions
  - Inter-object reflections

Shadows

- Shadow term tells if light sources are blocked
  - Cast ray towards each light source L_i. If the ray is blocked, do not consider the contribution of the light.

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

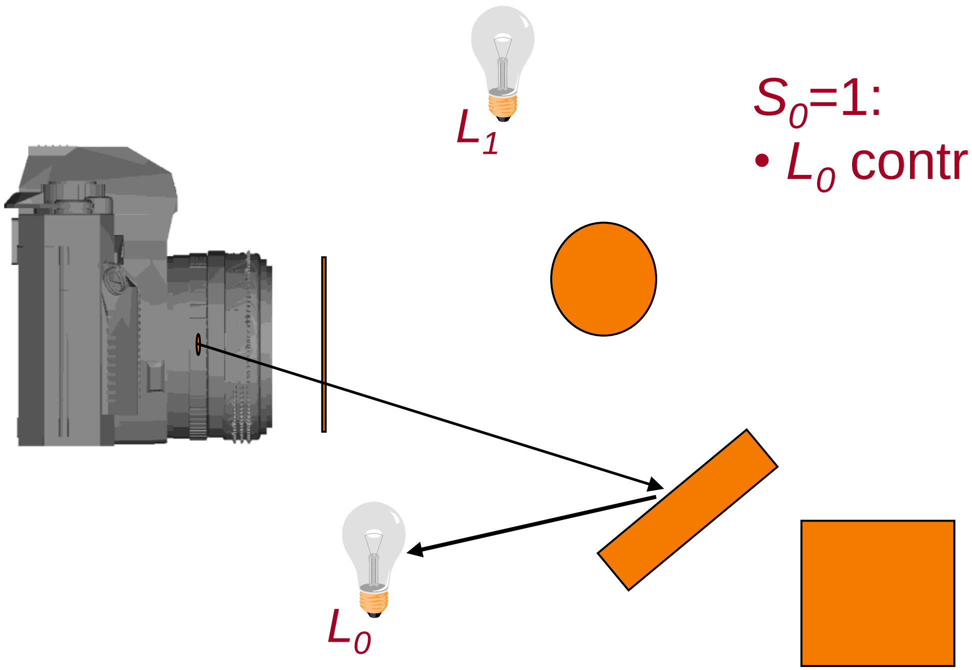


Shadow Term

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

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

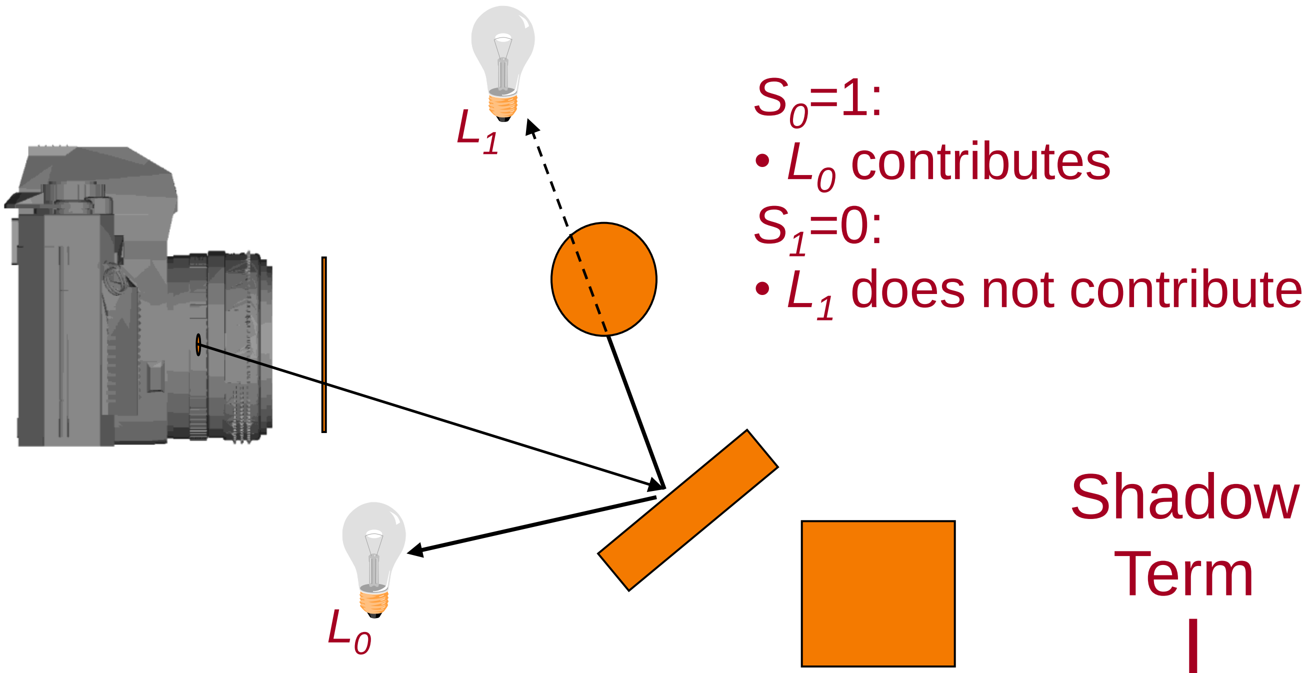


Shadow Term

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

Shadows

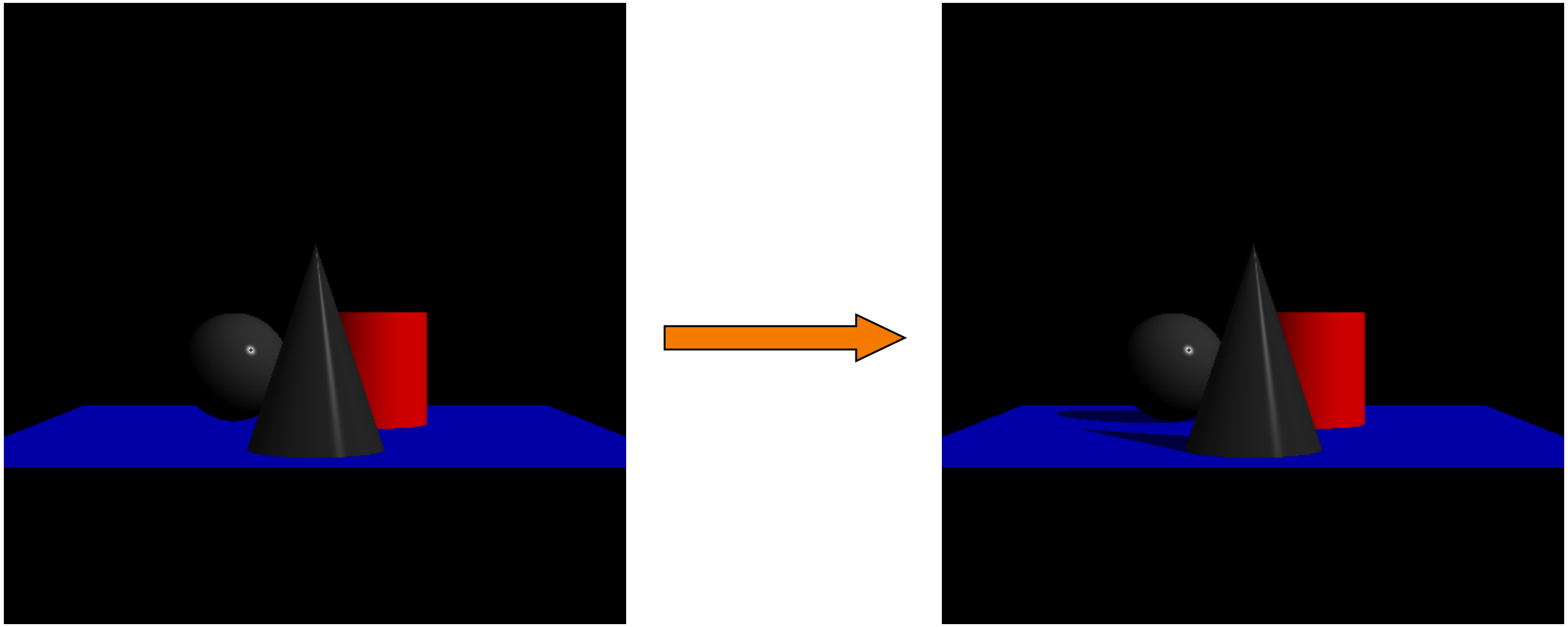
- Shadow term tells if light sources are blocked
  - Cast ray towards each light source L_i
  - $S_i = 0$ if ray is blocked, $S_i = 1$ otherwise



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

Ray Casting

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

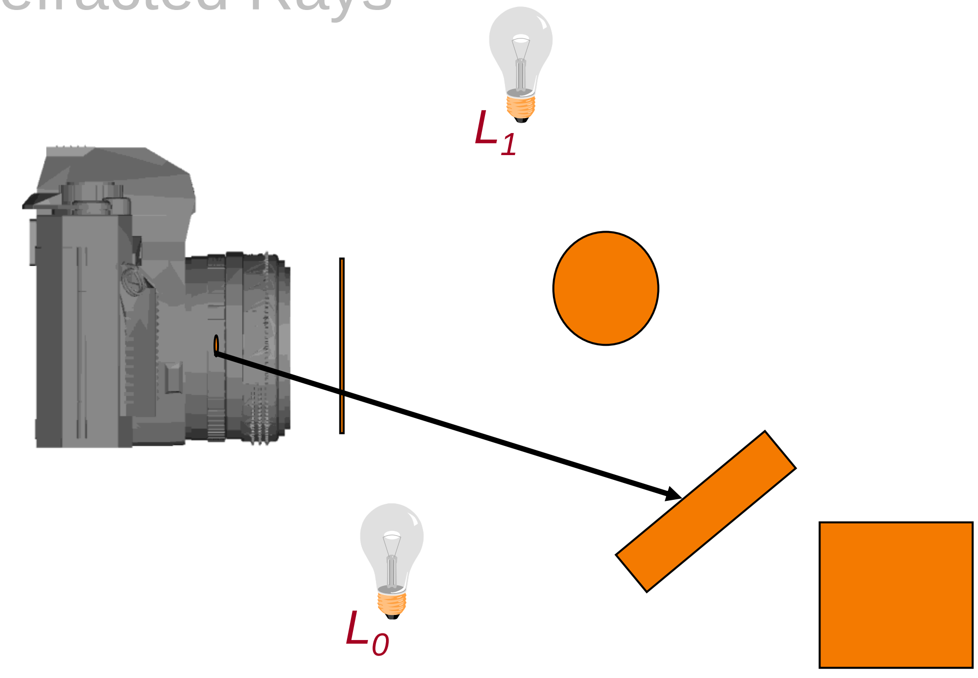


Recursive Ray Tracing

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

Mirror Reflections

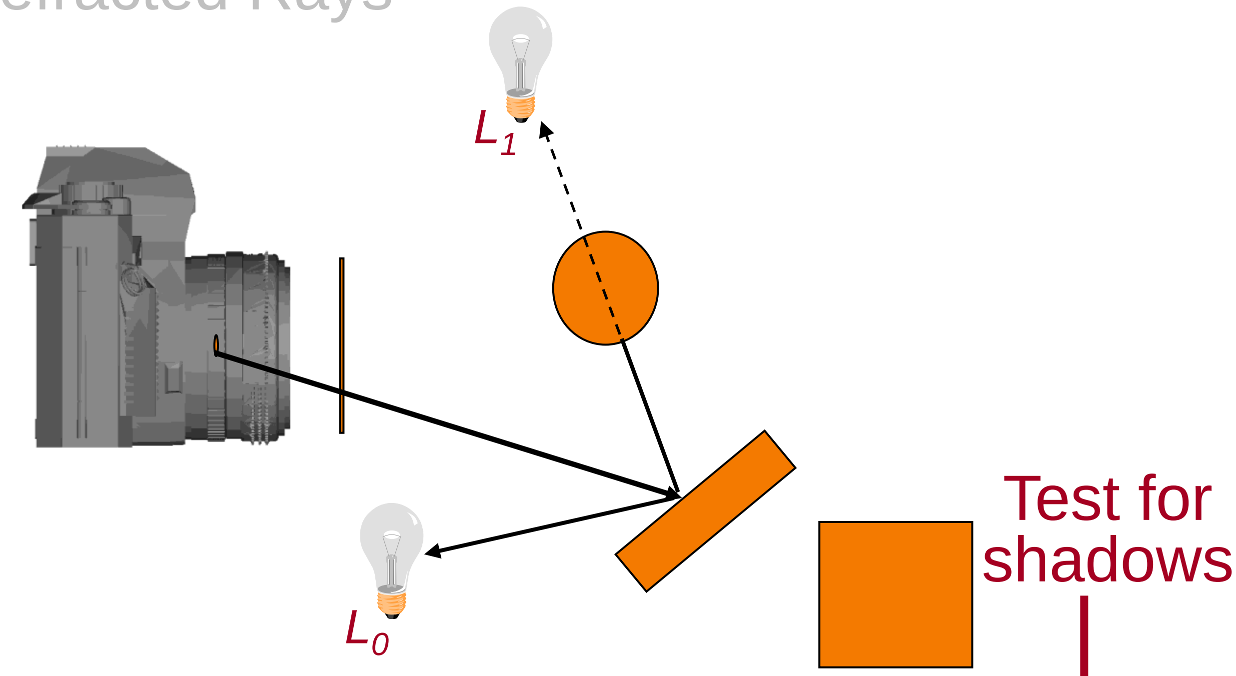
- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



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

Mirror Reflections

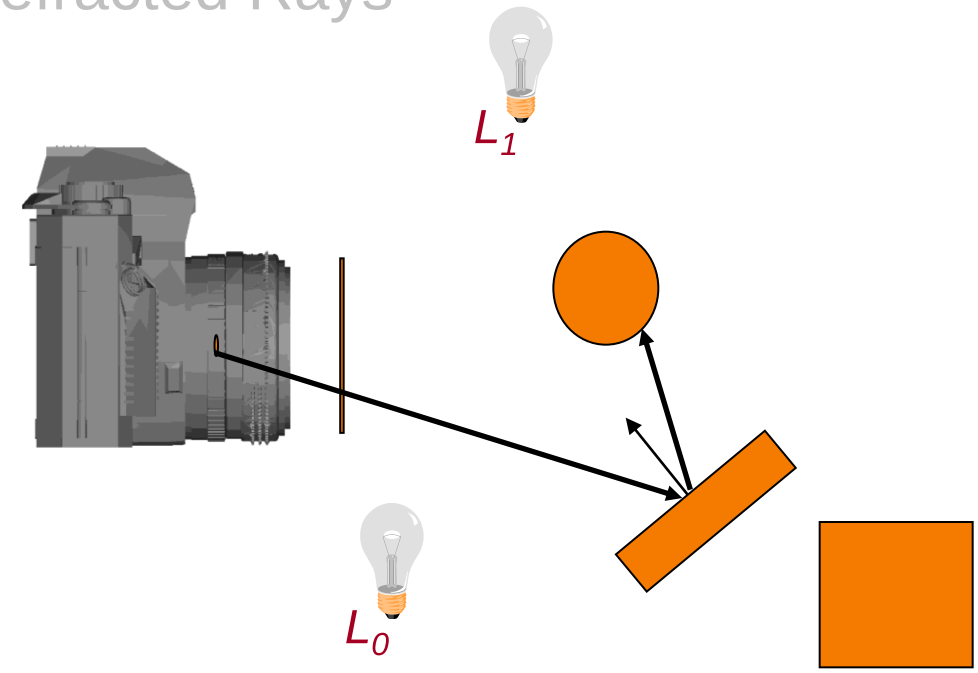
- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



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

Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

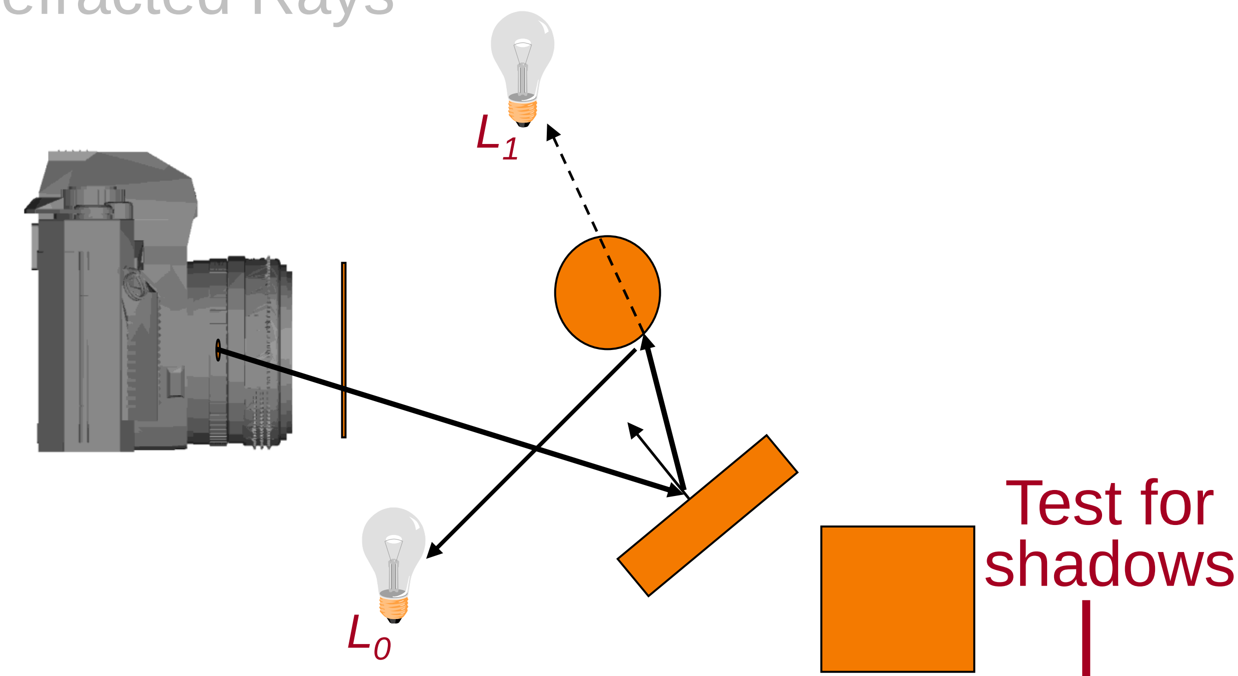


Radiance for mirror reflection ray

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

Mirror Reflections

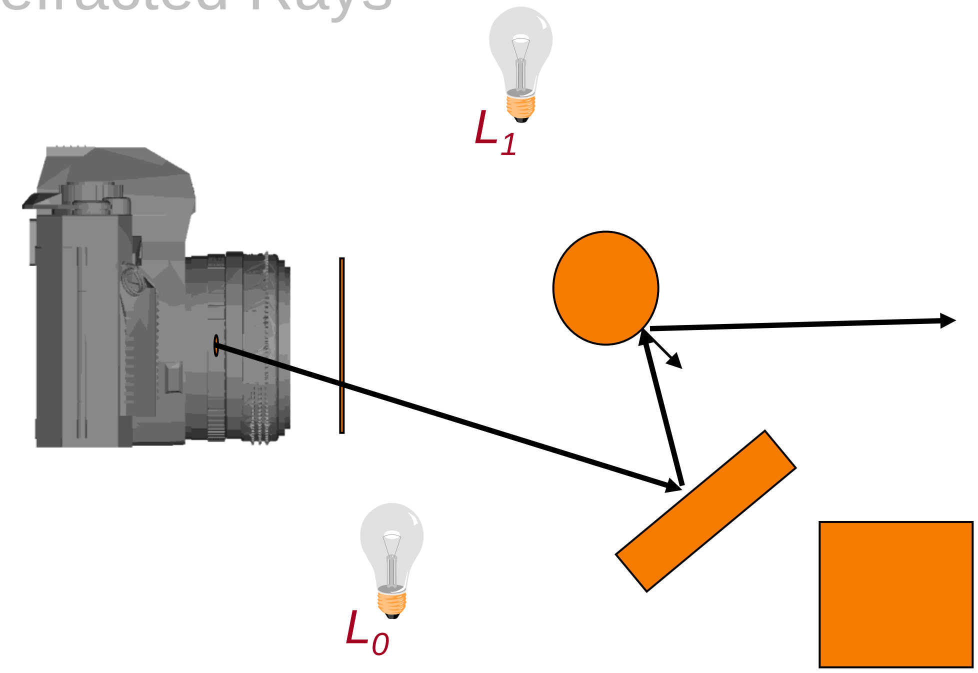
- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



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

Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

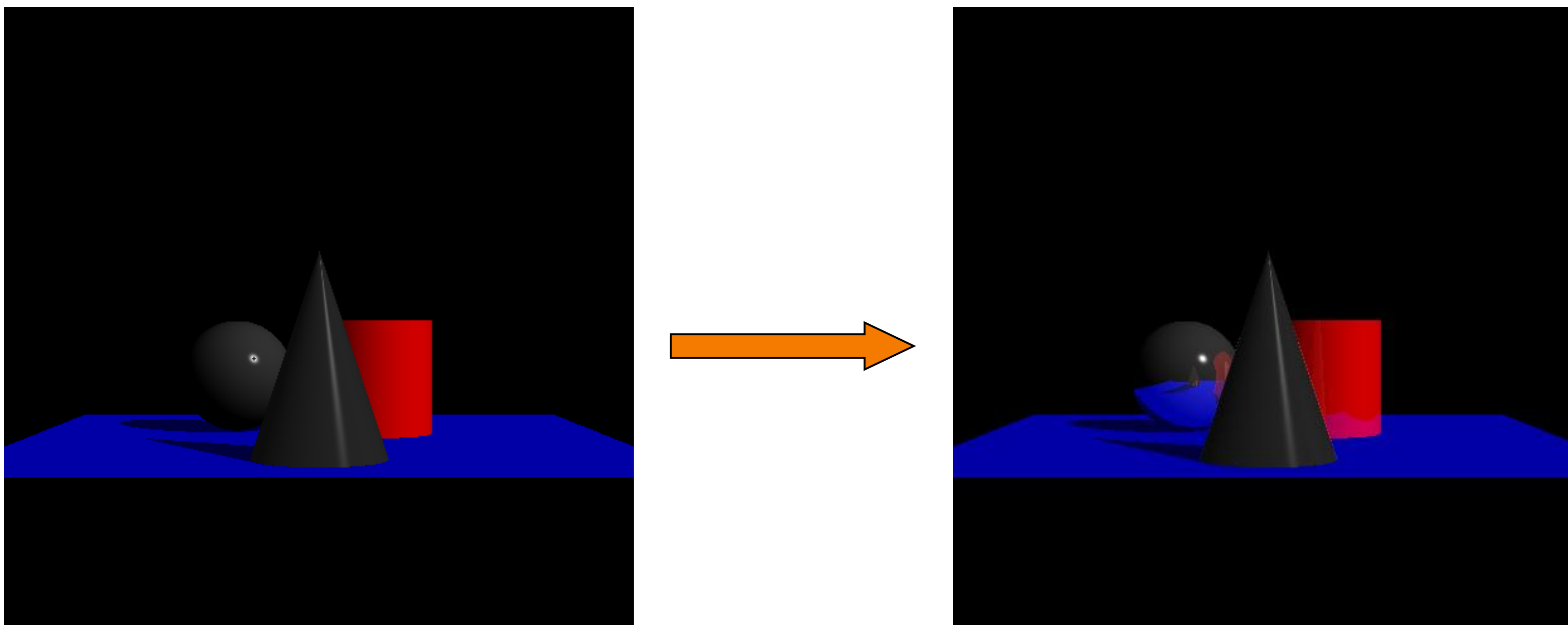


Radiance for mirror reflection ray

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

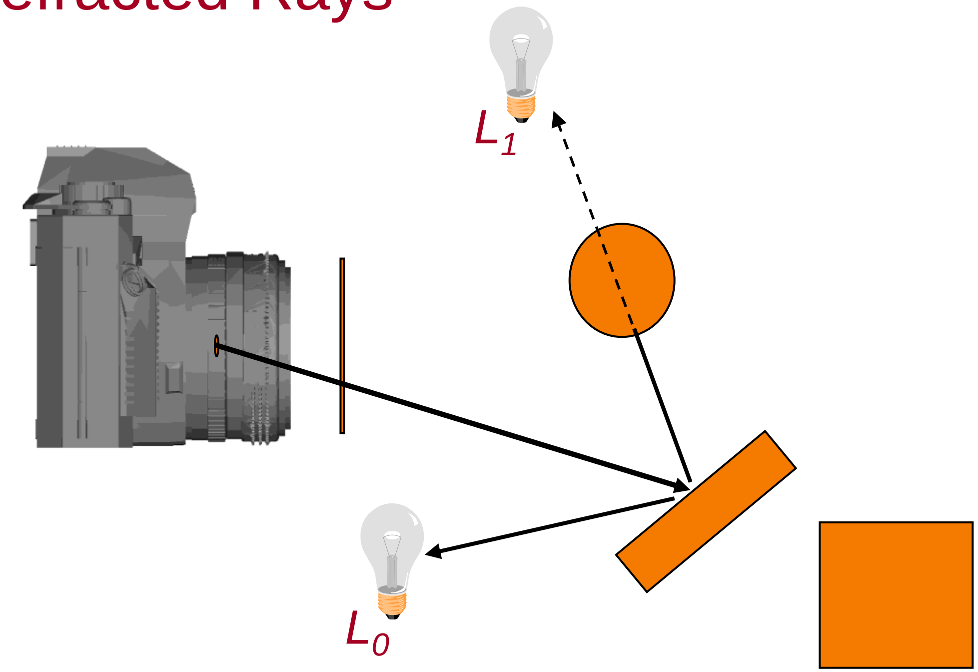
Mirror Reflections

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



Transparent Refraction

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

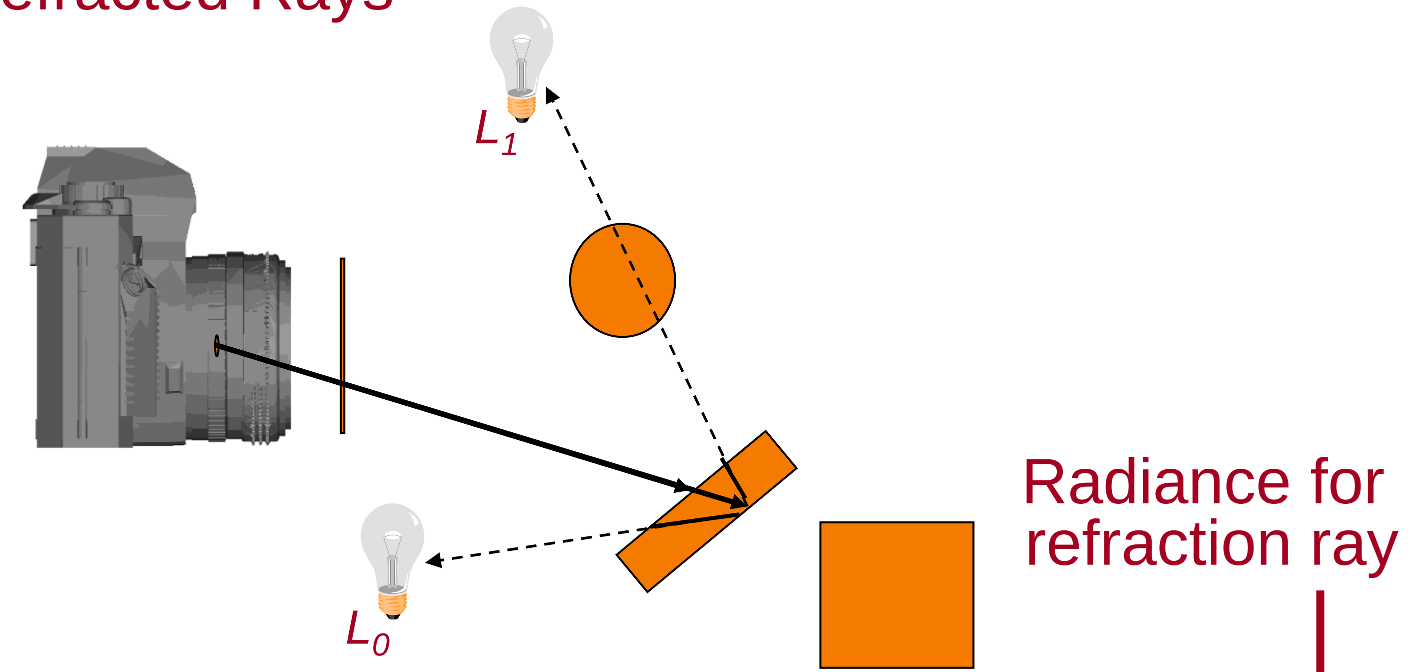


Radiance for refraction ray

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

Transparent Refraction

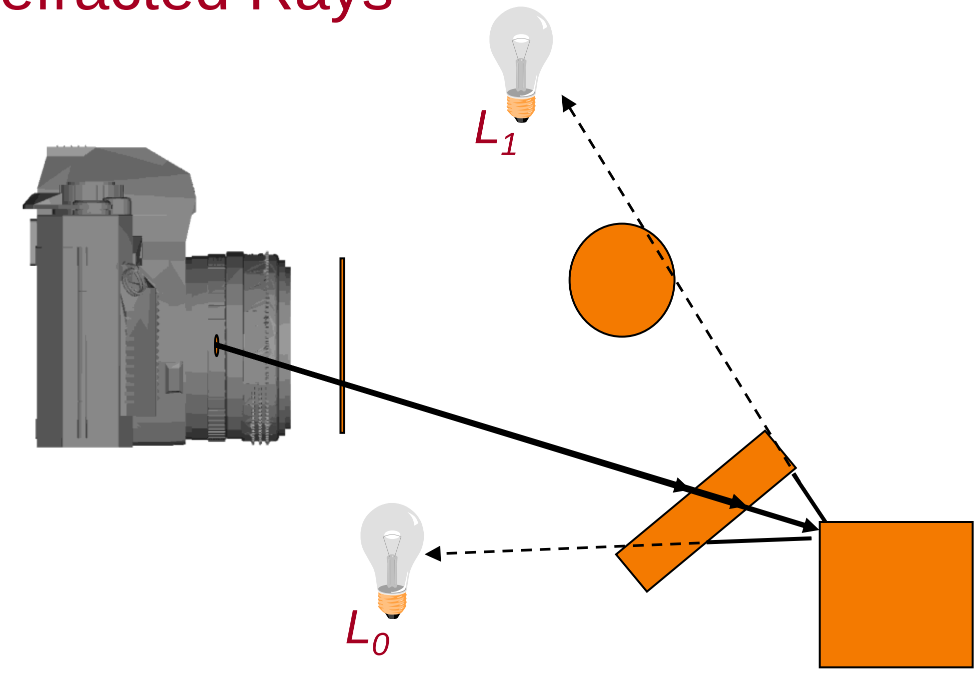
- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



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

Transparent Refraction

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

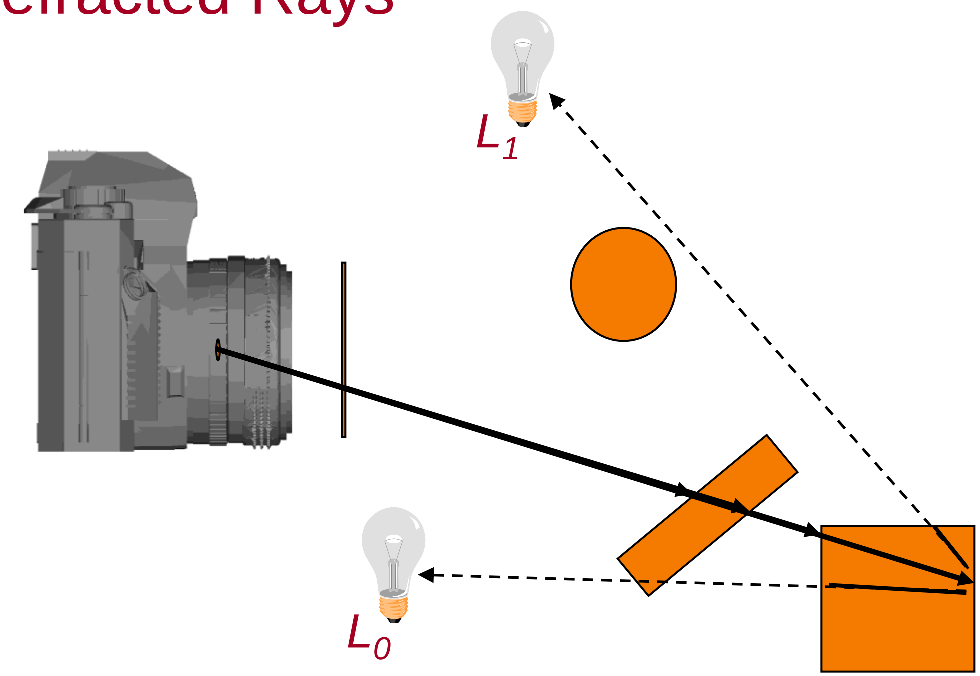


Radiance for refraction ray

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

Transparent Refraction

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays

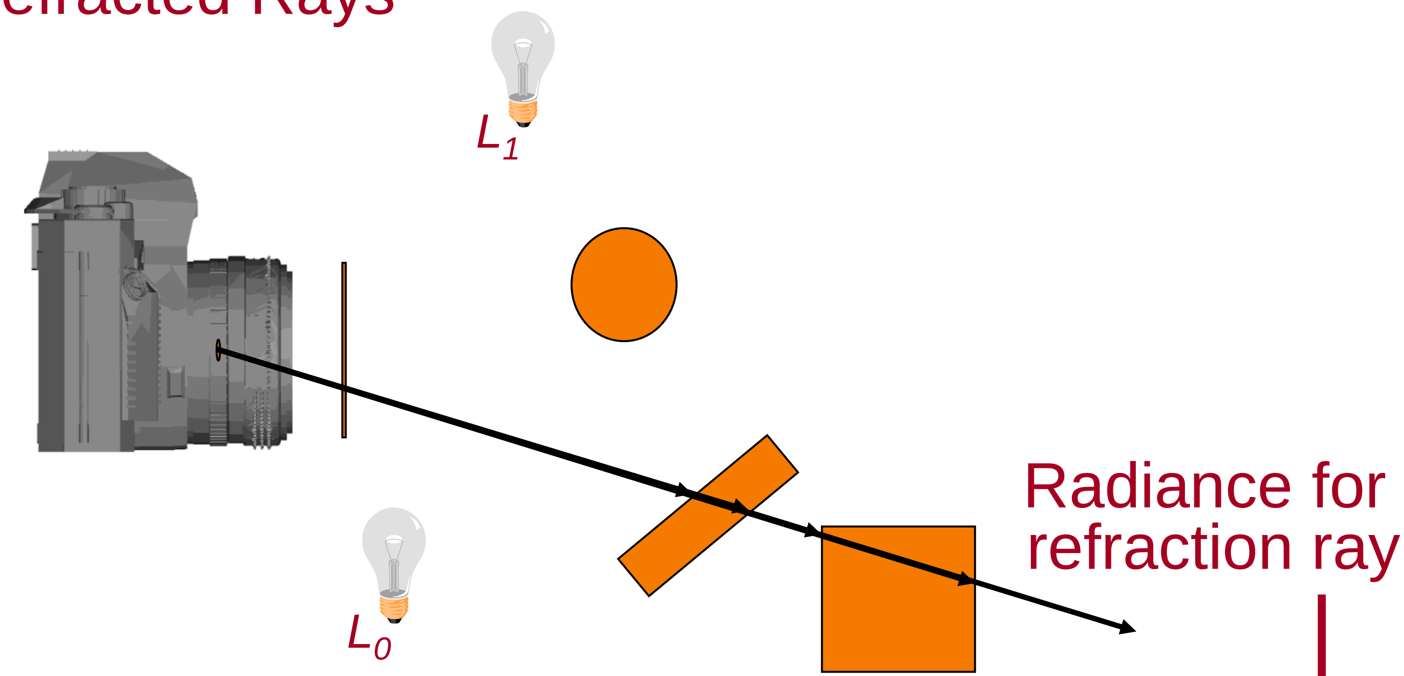


Radiance for refraction ray

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

Transparent Refraction

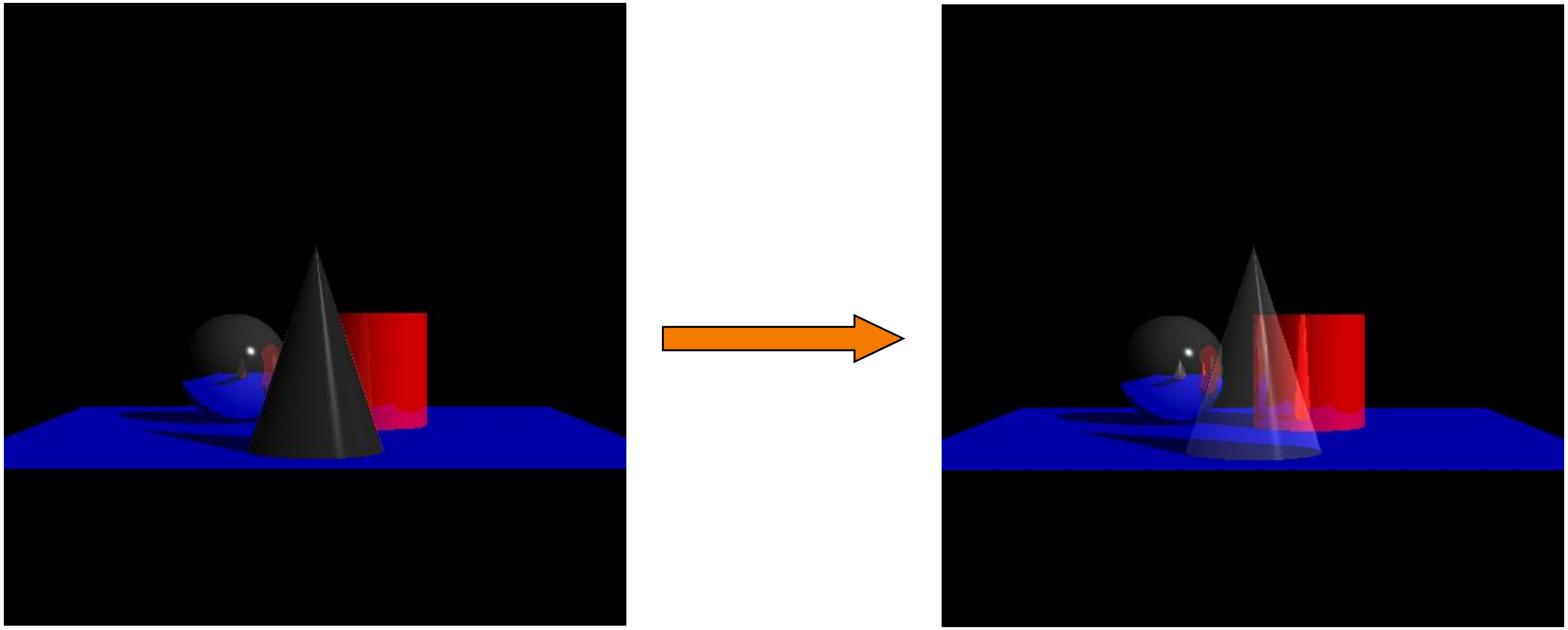
- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



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

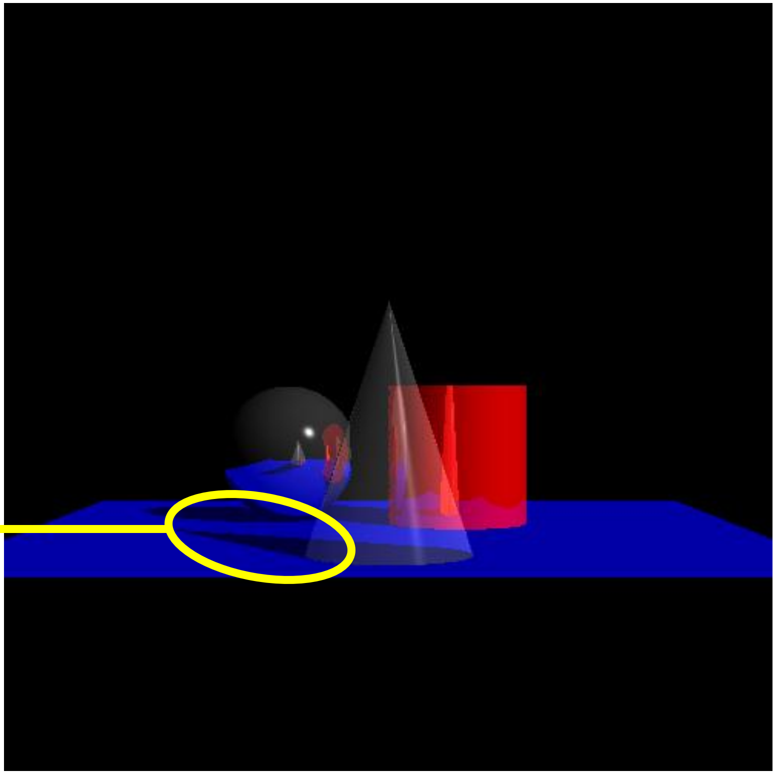
Transparent Refraction

- Also trace secondary rays from hit surfaces
  - Consider contributions from:
    1. Reflected Rays
    2. Refracted Rays



Transparency and Shadow

- Problem:
  - If a surface is transparent, then rays to the light source may pass through the object



Over-shadowing

Transparency and Shadow

- Problem:
  - If a surface is transparent, then rays to the light source may pass through the object
  - Need to modify the shadow term so that instead of representing a binary (0/1) value, it gives the fraction of light passing through.

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

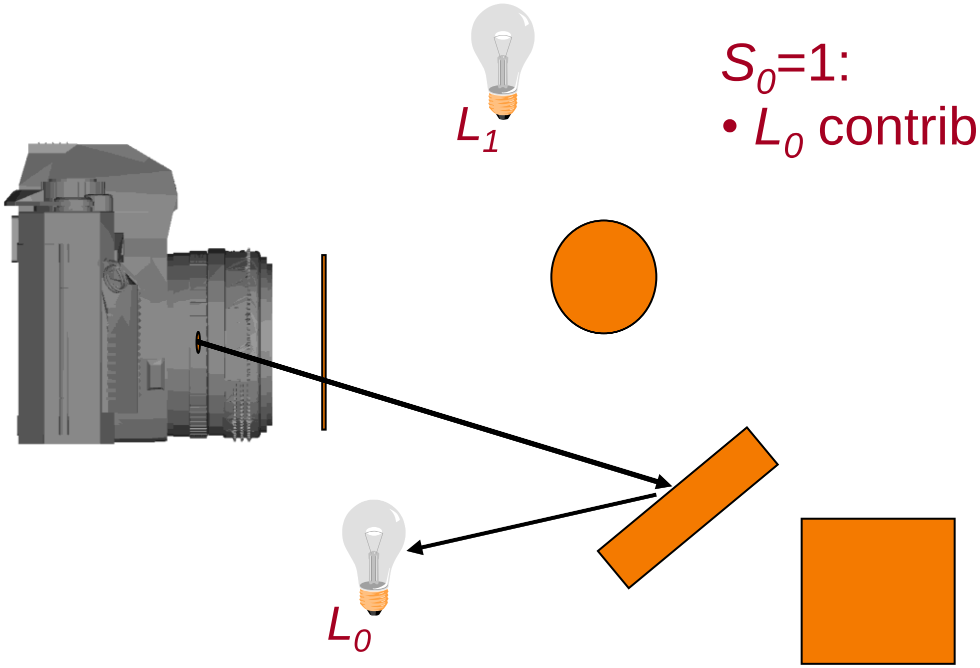
Transparency and Shadow

- Problem:
  - If a surface is transparent, then rays to the light source may pass through the object
  - Need to modify the shadow term so that instead of representing a binary (0/1) value, it gives the fraction of light passing through.
  - Accumulate transparency values as the ray travels to the light source.

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

Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.

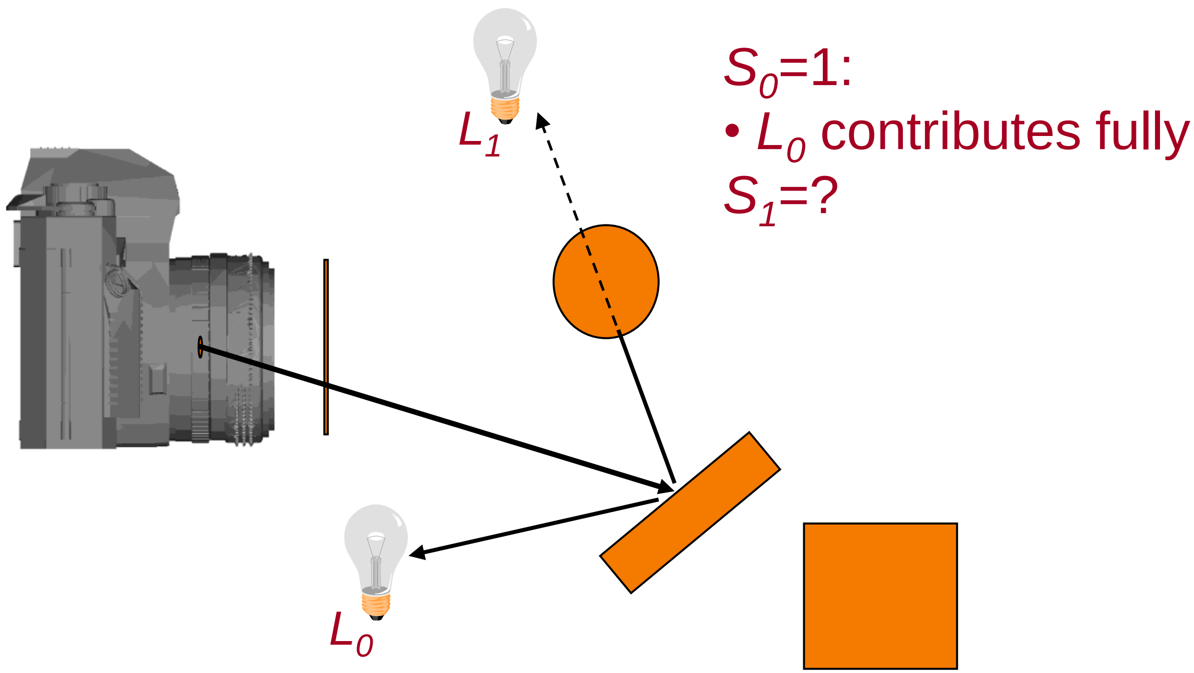


S_0=1:
- L_0 contributes fully

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

Transparency and Shadow

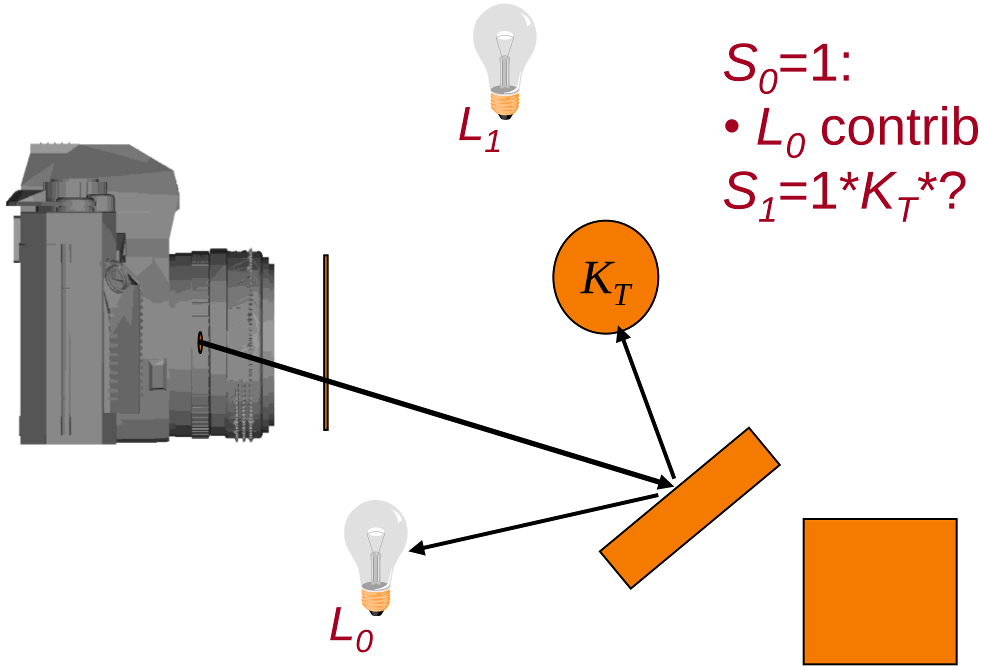
- Accumulate transparency values as the ray travels to the light source.



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

Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.



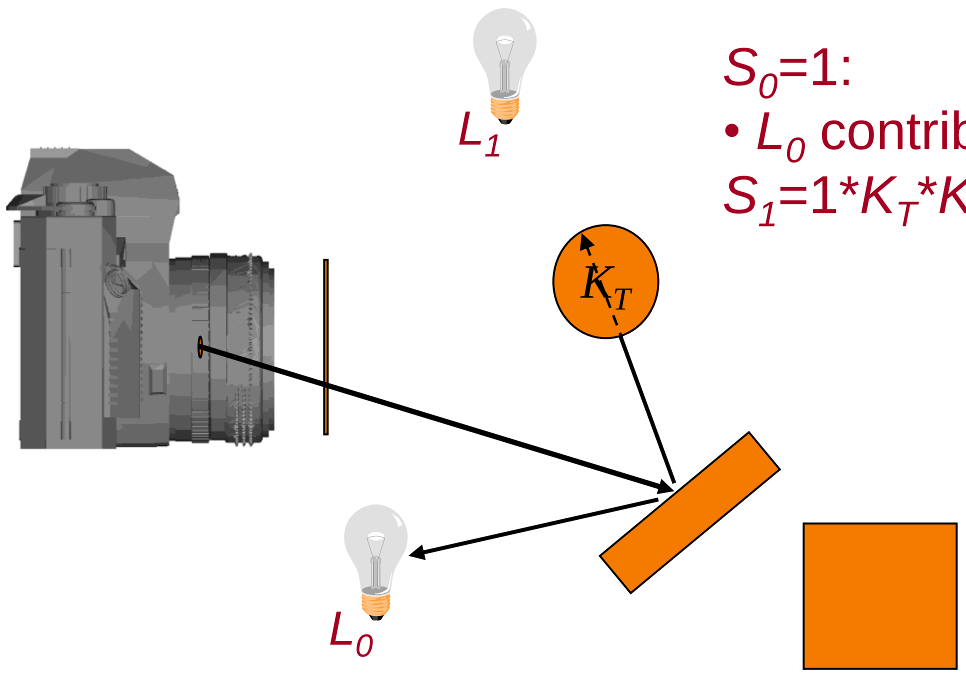
S_0=1:
- L_0 contributes fully

S_1=1*K_T*?

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

Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.



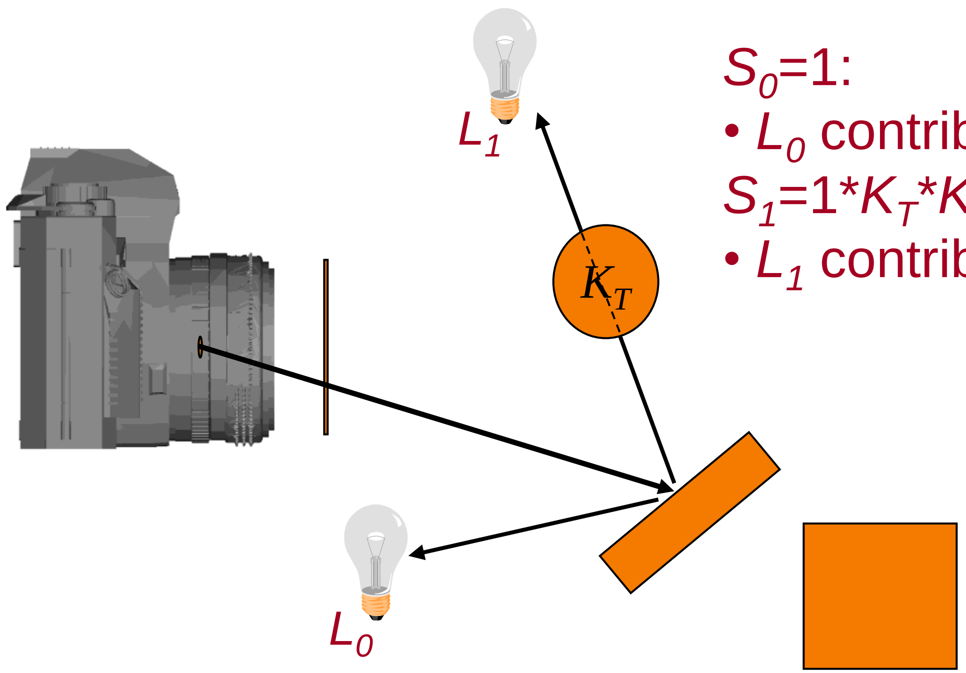
S_0=1:
- L_0 contributes fully

S_1=1*K_T*K_T*?

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

Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.



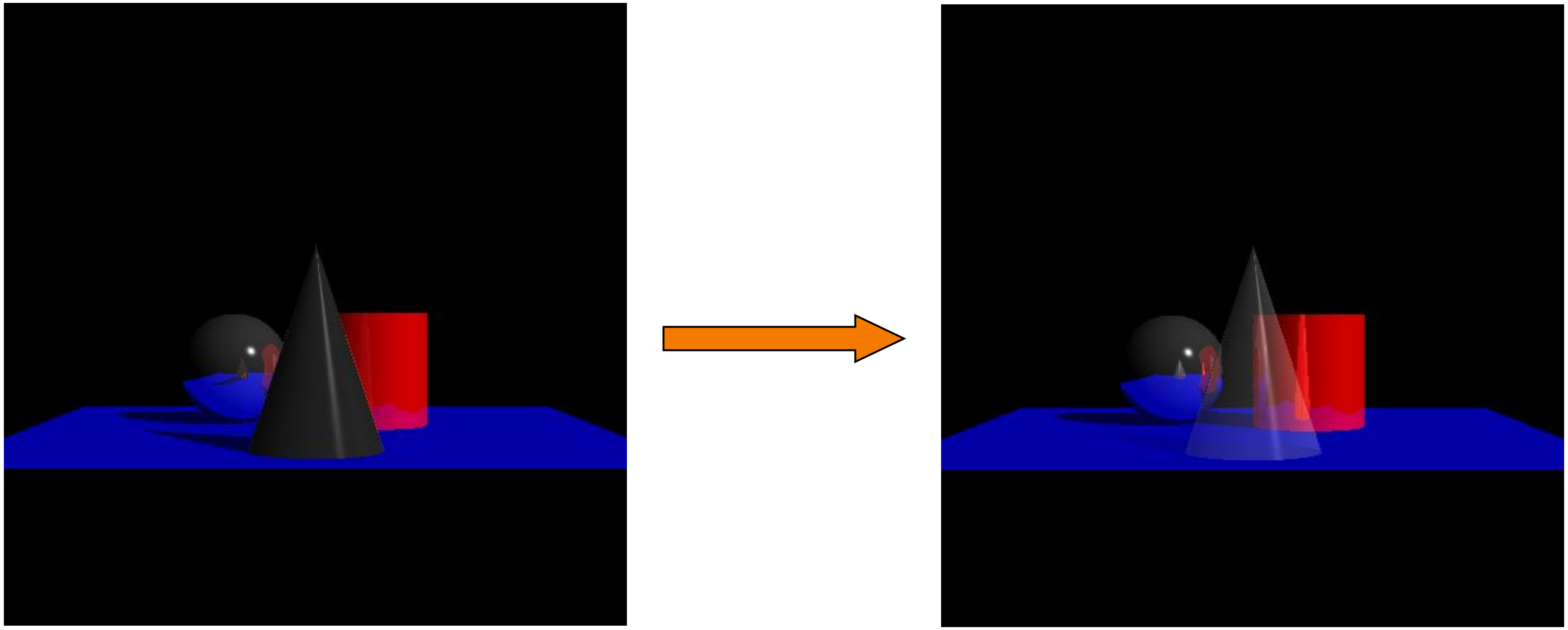
S_0=1:
- L_0 contributes fully

$S_1=1*K_T*K_T$:
- L_1 contributes partially

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

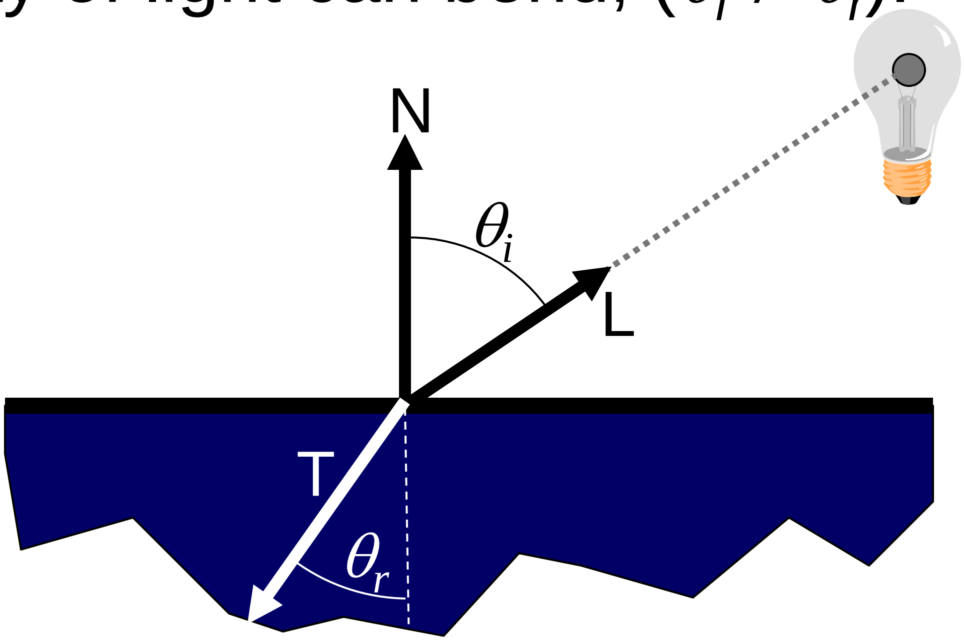
Transparency and Shadow

- Accumulate transparency values as the ray travels to the light source.



Transparent Refraction

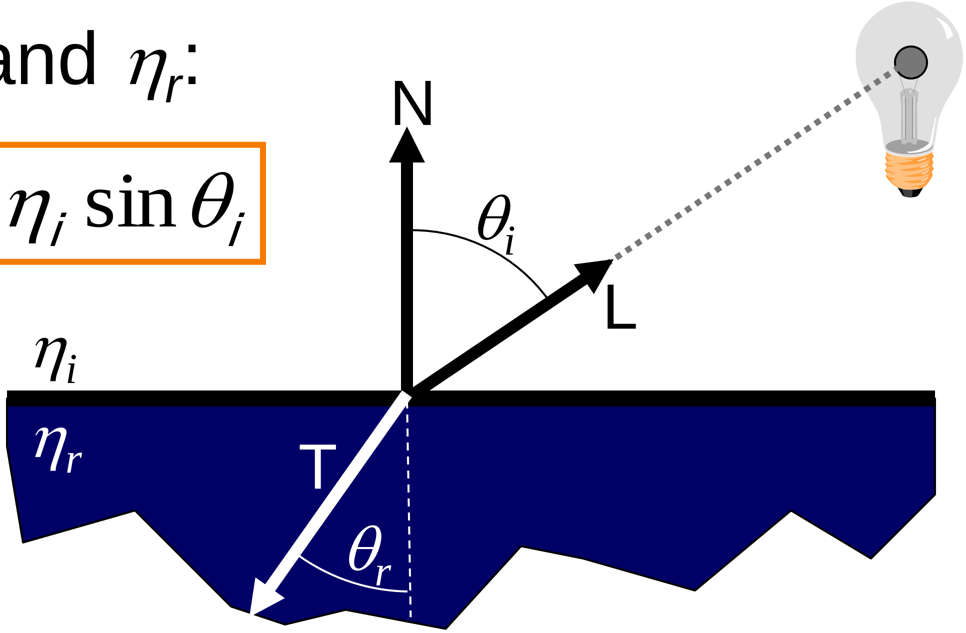
- When a light of light passes through a transparent object, the ray of light can bend, ($\theta_i \neq \theta_r$).



Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$

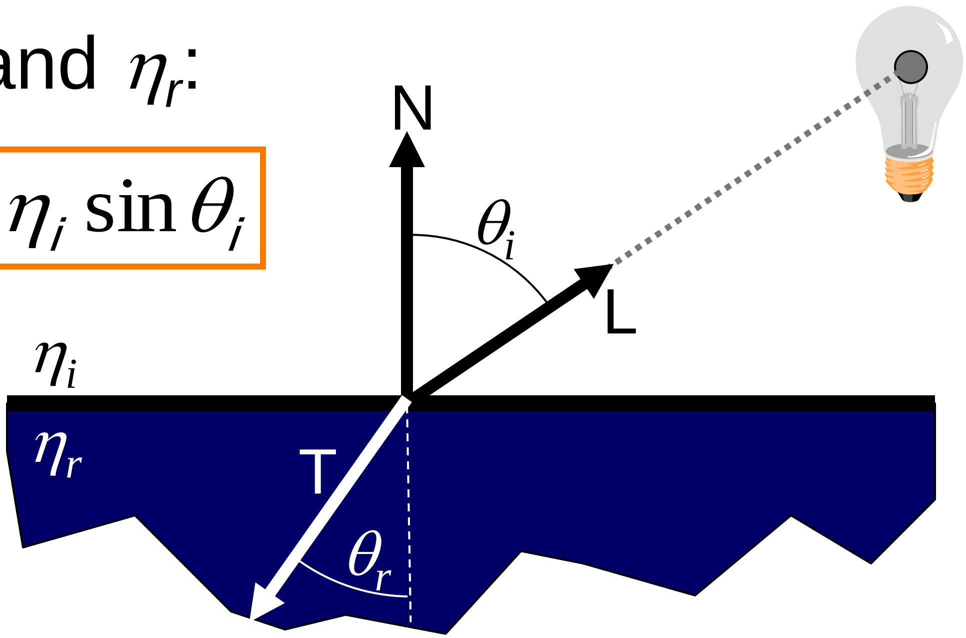


The index of refraction of air is $\eta=1$.

Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$

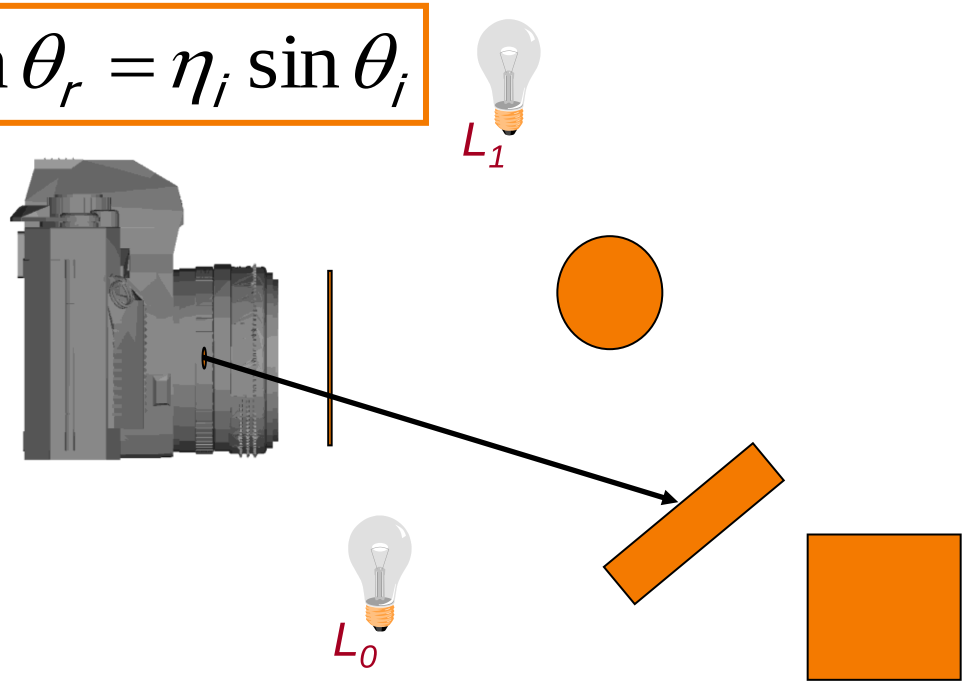


$$\boxed{T = \left(\frac{\eta_i}{\eta_r}\cos\theta_i - \cos\theta_r\right)N - \frac{\eta_i}{\eta_r}L}$$

Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

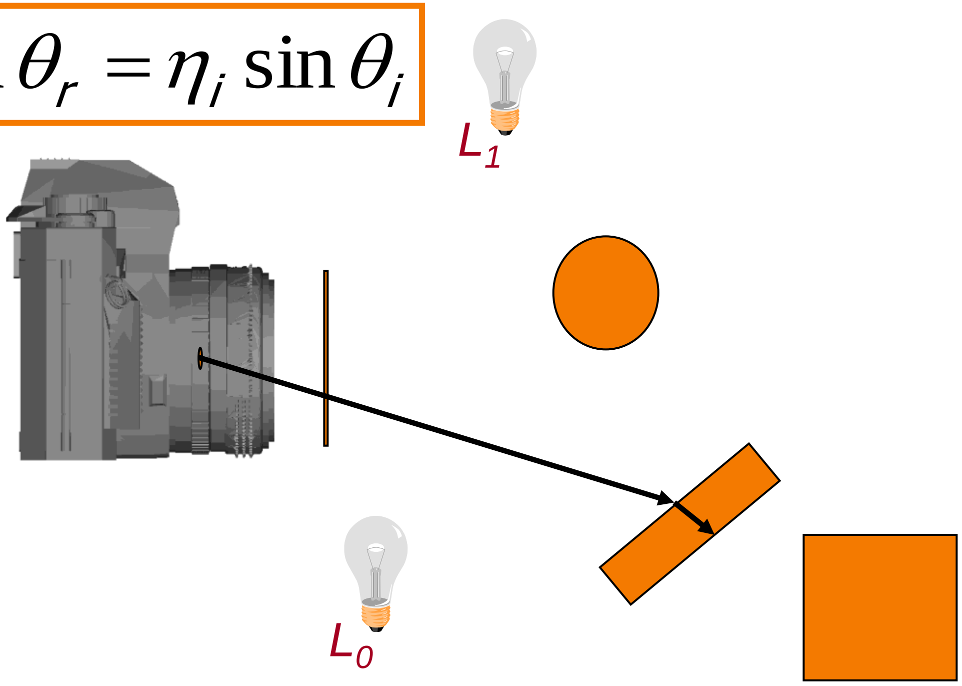
$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$



Snell’s Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

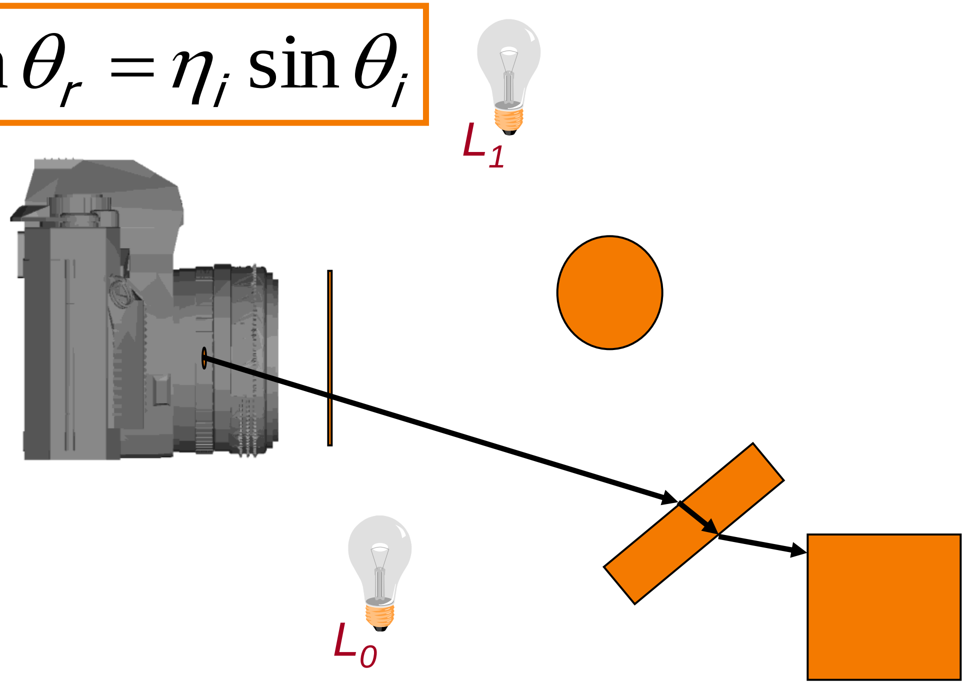
$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$



Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

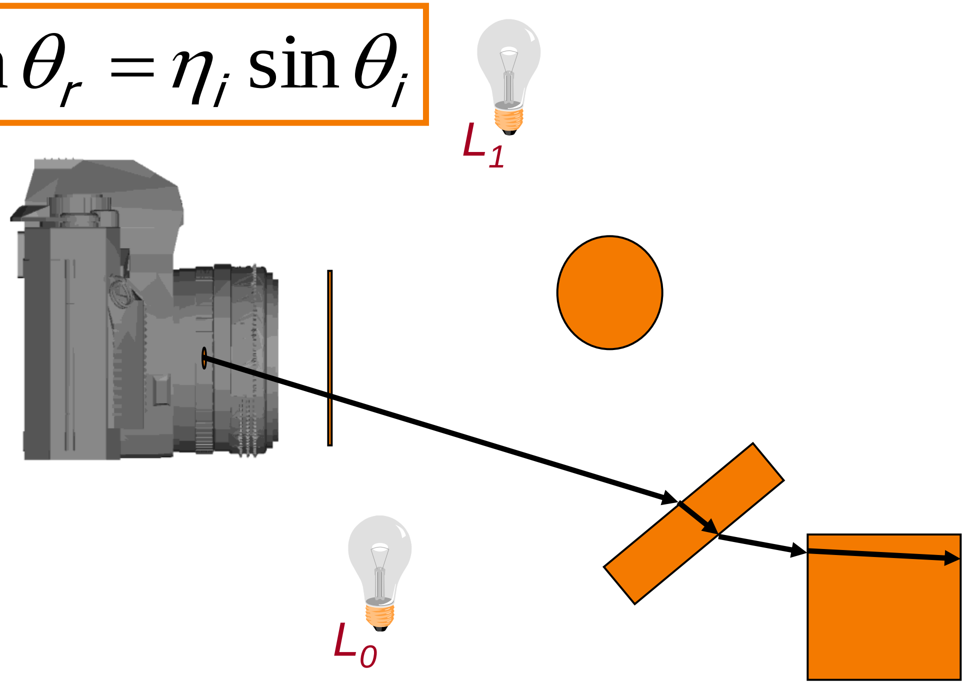
$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$



Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

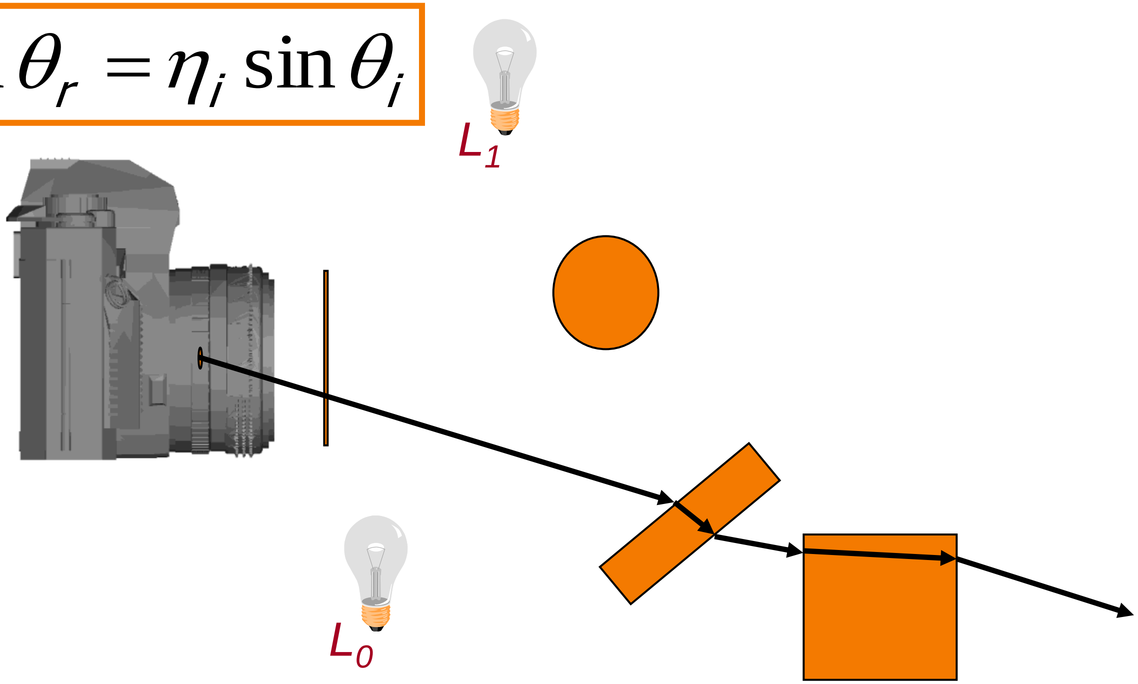
$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$



Snell's Law

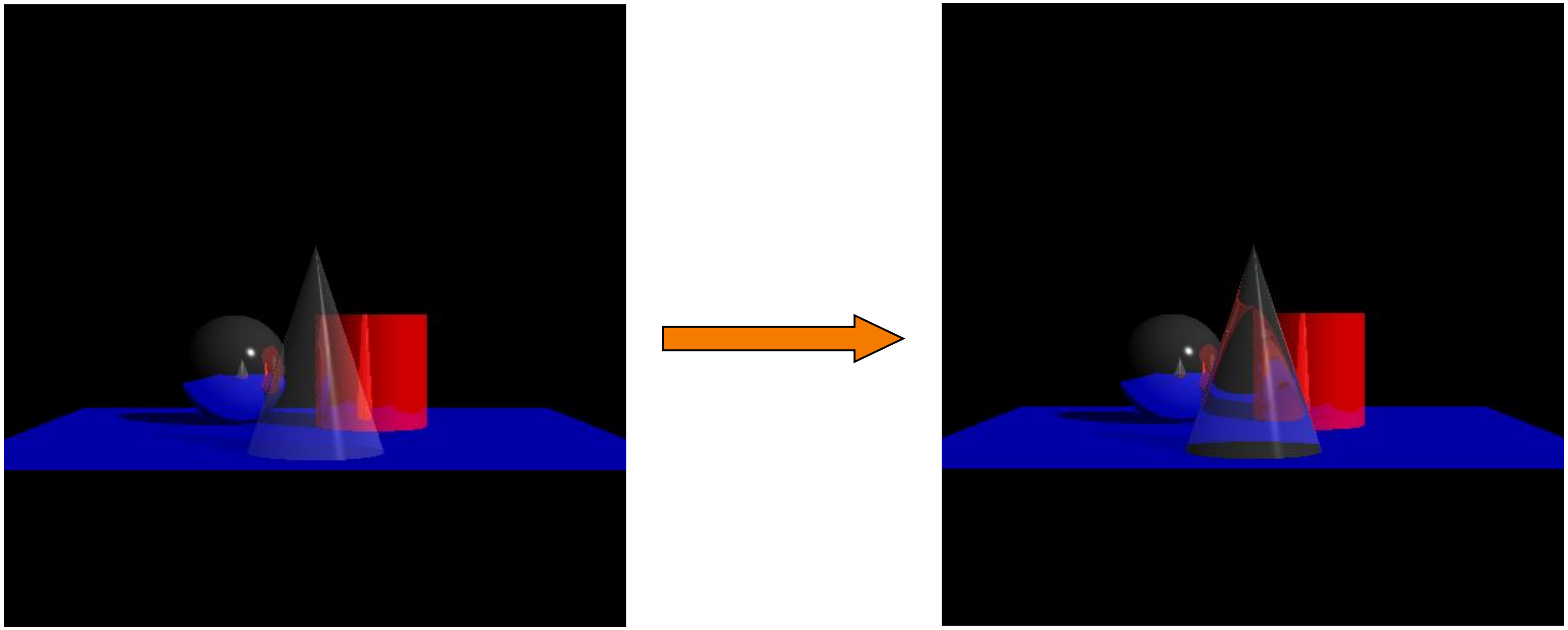
- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:

$$\boxed{\eta_r \sin\theta_r = \eta_i \sin\theta_i}$$



Snell's Law

- The way that light bends is determined by the indices of refraction of the internal and external materials η_i and η_r:



Snell's Law and Shadows

- Problem:
  - If a surface is transparent, then rays to the light source may not travel in a straight line

Snell's Law and Shadows

- Problem:
  - If a surface is transparent, then rays to the light source may not travel in a straight line
  - This is difficult to address with ray-tracing

General Issue

- How do we determine when to stop recursing?

General Issue

- How do we determine when to stop recursing?
  - Depth of iteration
    - » Bounds the number of times a ray will bounce around the scene
  - Cut-off value
    - » Ignores contribution from bounces that contribute very little

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
	Pixel p(0,0,0)
	Ray reflect, refract
	Intersection hit=FindIntersection(ray, scene);
	if ( hit ){
		p += GetSurfaceColor(hit.position);

		reflect.direction = Reflect( ray.direction, hit.normal)
		reflect.position = hit.position + reflect.direction*ε
		if( depth >0 && hit.kSpec>cutOff)
			p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec

		refract.direction = Refract( ray.direction, hit.normal, hit.ir)
		refract.position = hit.position + refract.direction*ε
		if( depth >0 && hit.kTran>cutOff)
			p += GetColor(scene, refract, depth-1, cutOff/hit.kTran) *hit.kTran
	}
	return p
}
```

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if ( hit ){
        p += GetSurfaceColor(hit.position);
    }
    return p
```

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

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
	Pixel p(0,0,0)
	Ray reflect, refract
	Intersection hit=FindIntersection(ray, scene);
	if ( hit ){
		p += GetSurfaceColor(hit.position);

		reflect.direction = Reflect( ray.direction, hit.normal)
		reflect.position = hit.position + reflect.direction *ε
		if( depth >0 && hit.kSpec>cutOff)
			p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec

	}
	return p
```

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

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
    Pixel p(0,0,0)
    Ray reflect, refract
    Intersection hit=FindIntersection(ray, scene);
    if ( hit ){
        p += GetSurfaceColor(hit.position);

        reflect.direction = Reflect( ray.direction, hit.normal)
        reflect.position = hit.position + reflect.direction *ε
        if( depth >0 && hit.kSpec>cutOff)
            p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)*hit.kSpec

        refract.direction = Refract( ray.direction, hit.normal, hit.ir)
        refract.position = hit.position + refract.direction*ε
        if( depth >0 && hit.kTran>cutOff)
            p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)*hit.kTran
    }
    return p
```

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

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
        Pixel p(0,0,0)
        Ray reflect, refract
        Intersecti
        if ( hit ){
                p += GetSurfaceColor(hit.position);

                reflect.direction = Reflect( ray.direction, hit.normal)
                reflect.position = hit.position + reflect.direction *ε
                if( depth >0 && hit.kSpec>cutOff)
                        p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)

                refract.direction = Refract( ray.direction, hit.normal, hit.ir)
                refract.position = hit.position + refract.direction*ε
                if( depth >0 && hit.kTran>cutOff)
                        p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)
        }
        return p
}
```

Why do we need the ε terms?

General Issue

- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
        Pixel p(0,0,0)
        Ray reflect, refract
        Intersecti
        if ( hit ){
                p += GetSurfaceColor(hit.position);

                reflect.direction = Reflect( ray.direction, hit.normal)
                reflect.position = hit.position + reflect.direction *ε
                if( depth >0 && hit.kSpec>cutOff)
                        p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)

                refract.direction = Refract( ray.direction, hit.normal, hit.ir)
                refract.position = hit.position + refract.direction*ε
                if( depth >0 && hit.kTran>cutOff)
                        p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)
        }
        return p
}
```

Why do we need the ε terms?

To ensure that the new ray does not hit its starting location!

General Issue

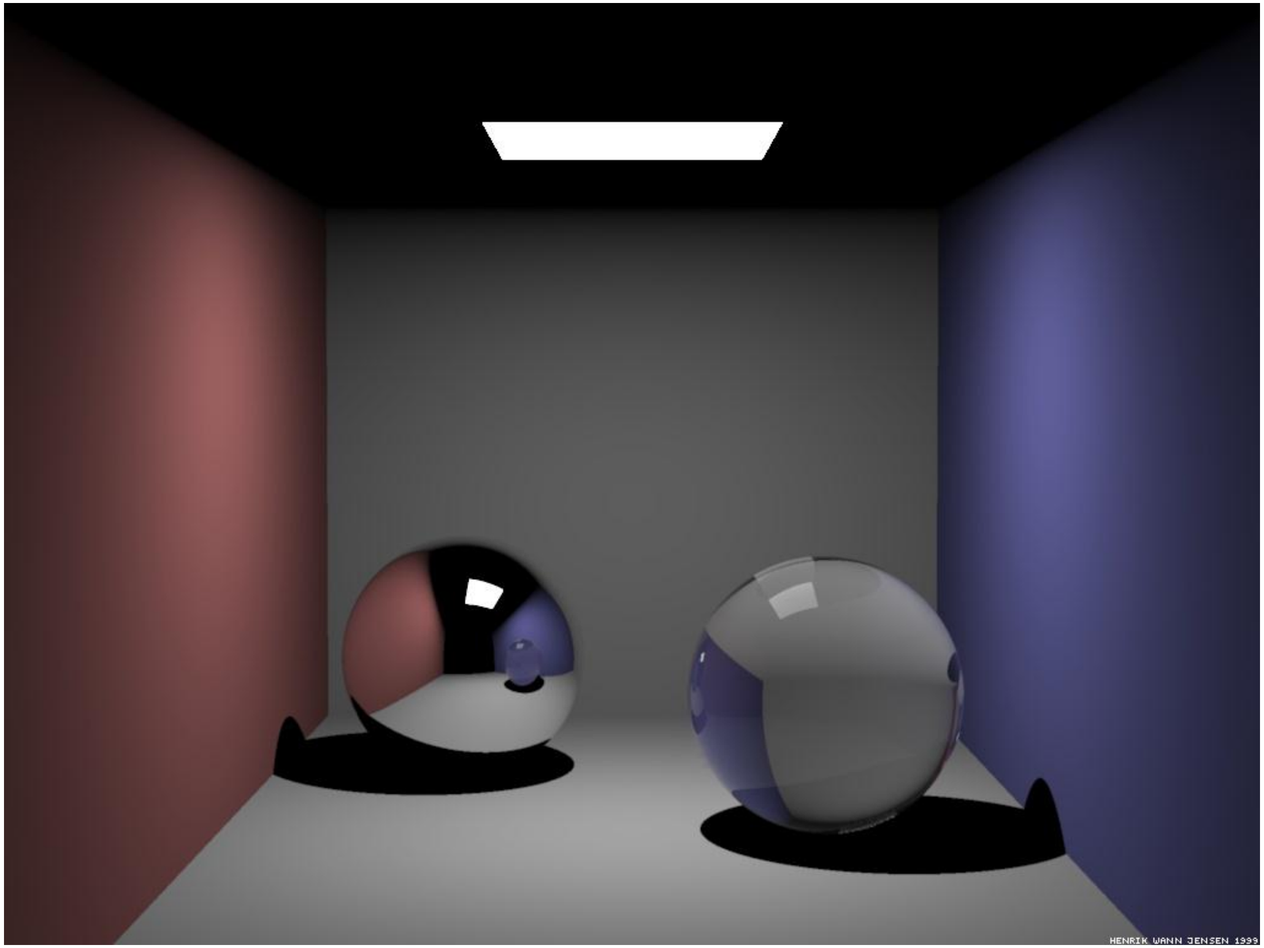
- How do we determine when to stop recursing?

```
Pixel GetColor(scene, ray, depth, cutOff){
	Pixel p(0,0,0)
	Ray reflect, refract
	Intersection hit=FindIntersection(ray, scene);
	if ( hit ){
		p += GetSurfaceColor(hit.position);

		reflect.direction = Reflect( ray.direction, hit.normal)
		reflect.position = hit.position + reflect.direction *ε
		if( depth >0 && hit.kSpec>cutOff)
			p += GetColor(scene, reflect, depth-1, cutOff/hit.kSpec)

		refract.direction = Refract( ray.direction, hit.normal, hit.ir)
		refract.position = hit.position + refract.direction*ε
		if( depth >0 && hit.kTran>cutOff)
			p += GetColor(scene, refract, depth-1, cutOff/hit.kTran)
	}
	return p
}
```

Warning: In practice, cut-off is a scalar while hit.kTran/kSpec are rgb values.

Illumination Examples

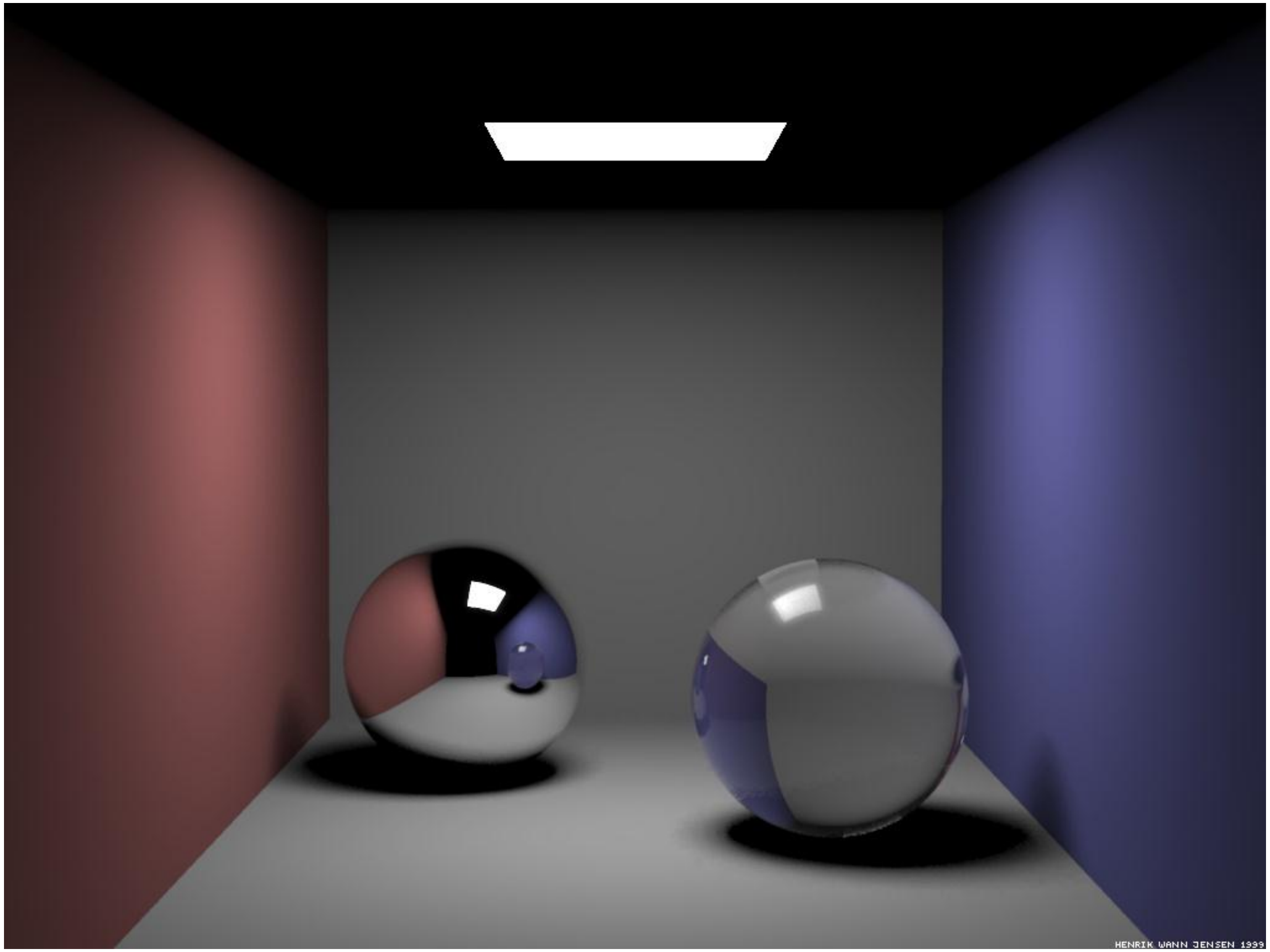
- Ray tracing



Courtesy Henrik Wann Jensen

Illumination Examples

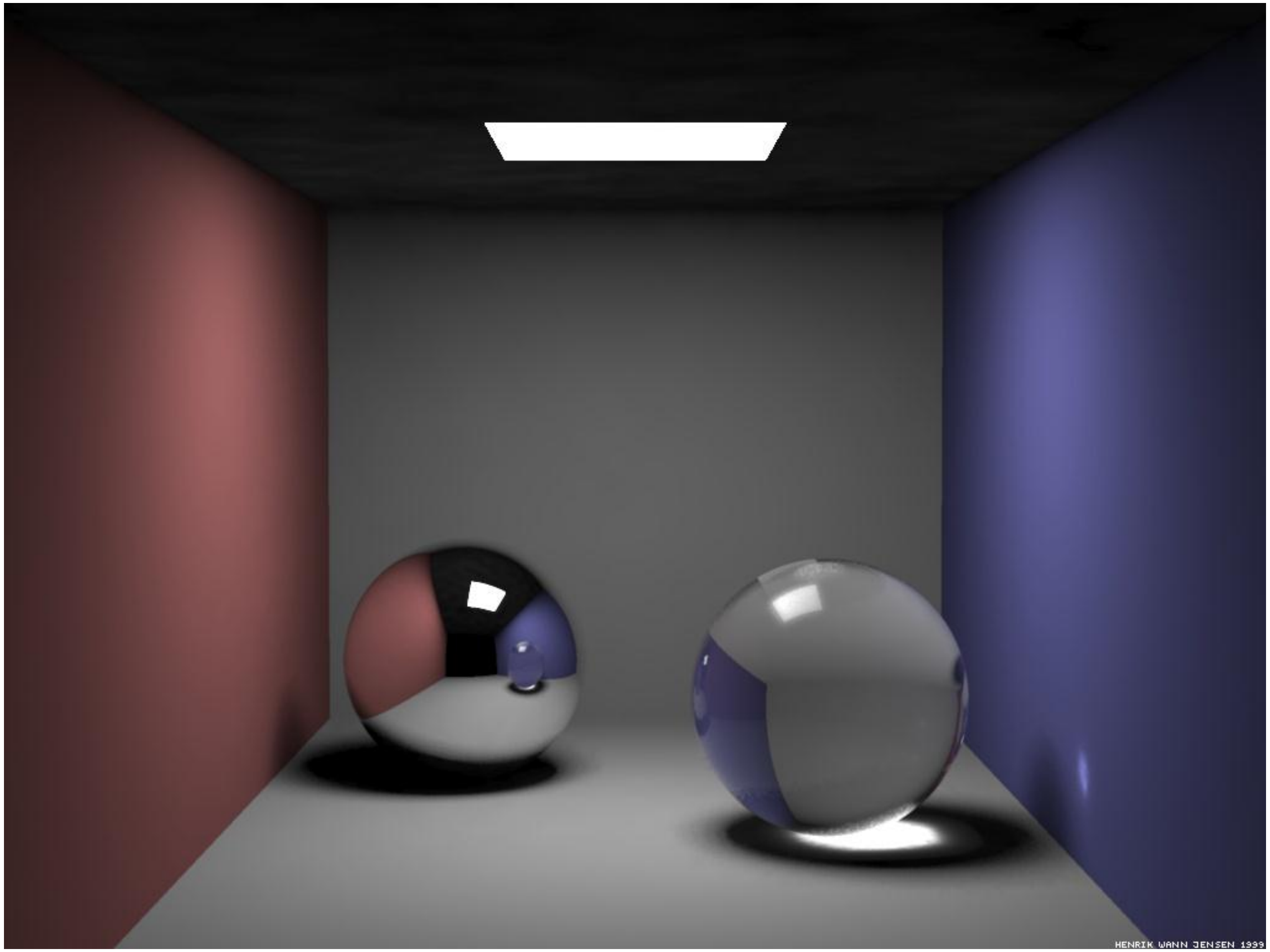
- Soft Shadows



Courtesy Henrik Wann Jensen

Illumination Examples

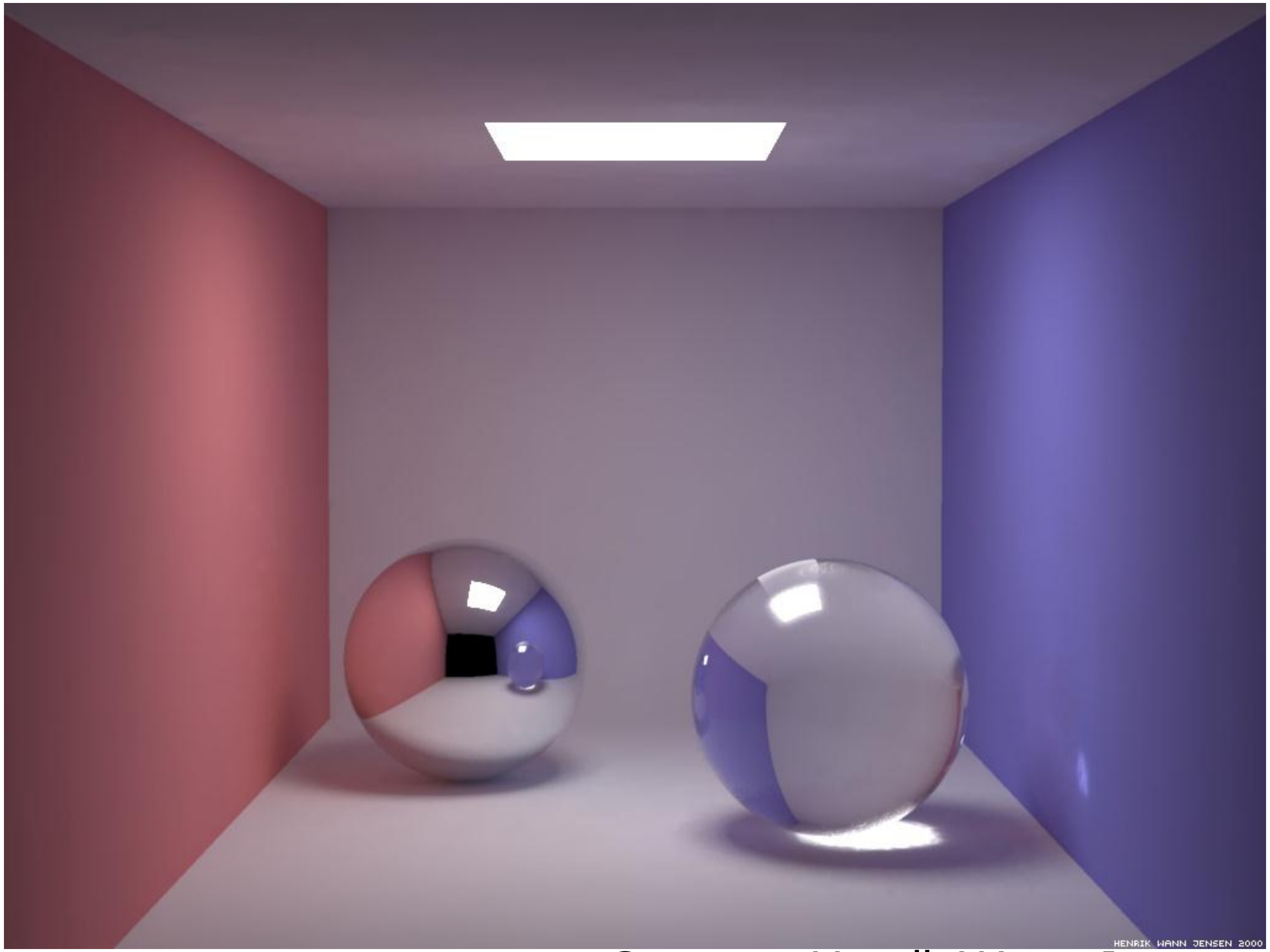
- Caustics



Courtesy Henrik Wann Jensen

Illumination Examples

- Full Global Illumination



Courtesy Henrik Wann Jensen

Recursive Ray Tracing

- GetColor calls RayTrace recursively

```
Image RayTrace(Camera camera, Scene scene, int width, int height
                    int depth, float cutOff){
    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, depth, cutOff);
        }
    }
    return image;
}
```

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!