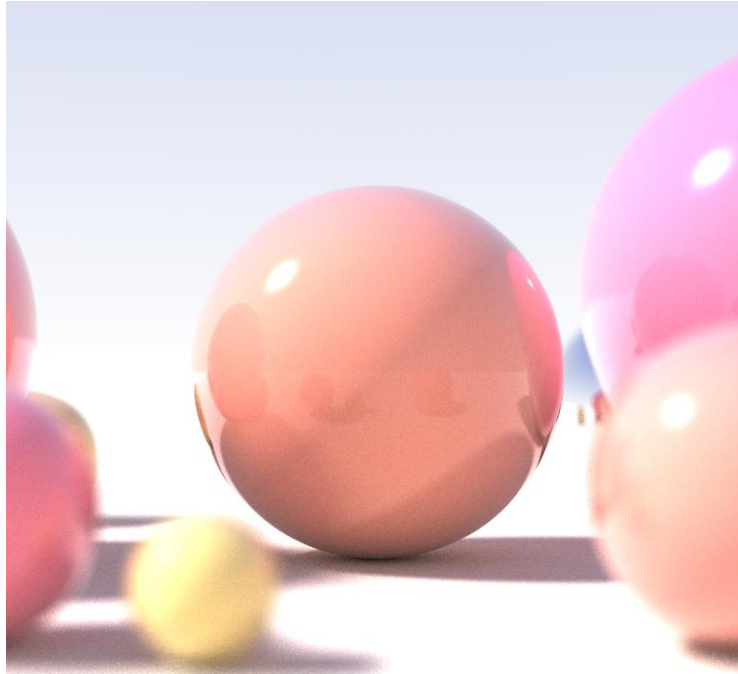


Ray Tracing

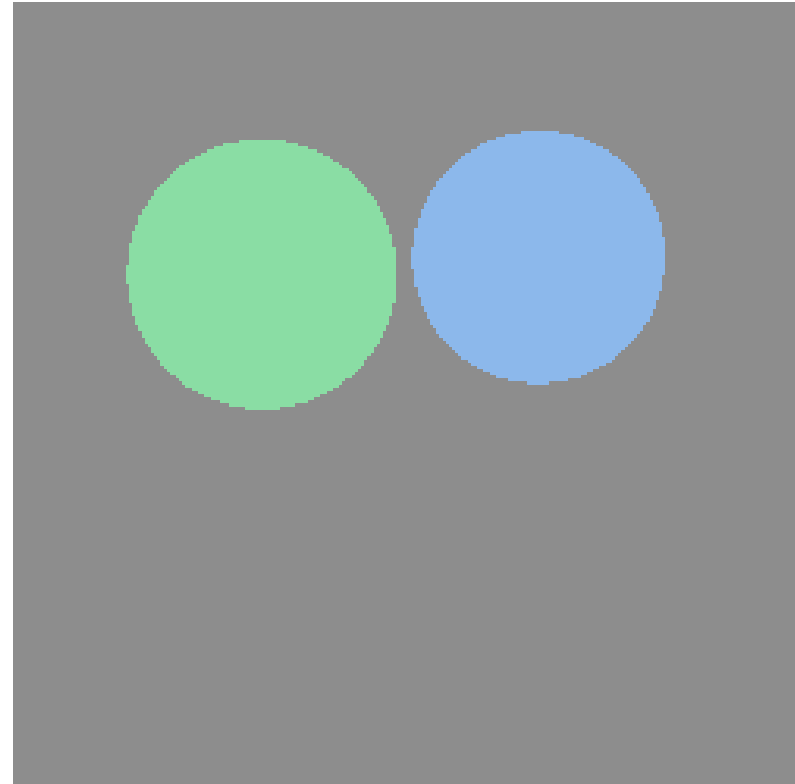


Some Slides/Images adapted from Marschner and Shirley and David Levin



Ray Casting

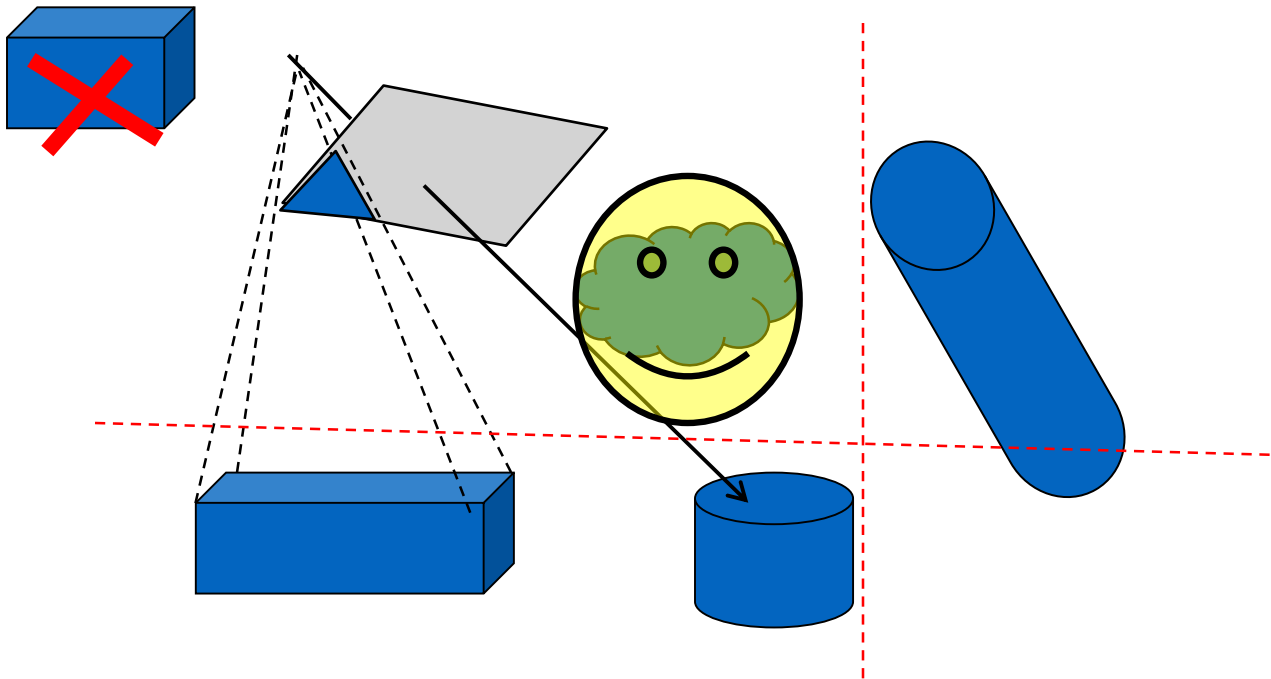
```
for 0 <= iy < ny
  for 0 <= ix < nx
  {
    ray = camera.getRay(ix, iy);
    firstSurface = scene.intersect(result,ray);
    if (firstSurface)
      image.set(ix, iy, firstSurface.color);
    else
      image.set(ix, iy, background.color);
  }
```



Ray Intersection: Efficiency Considerations

Speed-up the intersection process.

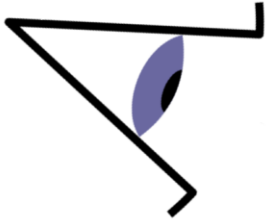
- Ignore object that clearly don't intersect.
- Use proxy geometry.
- Subdivide and structure space hierarchically.
- Project volume onto image to ignore entire sets of rays.



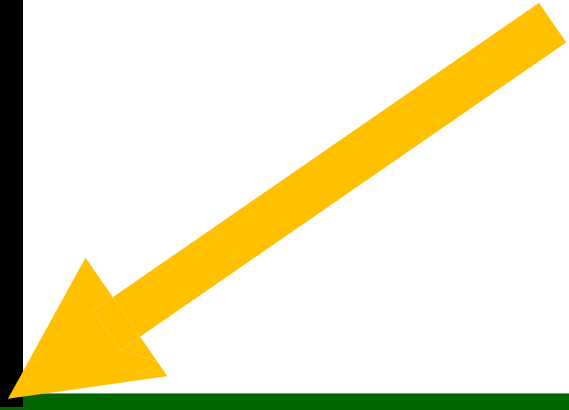
Light and Surfaces



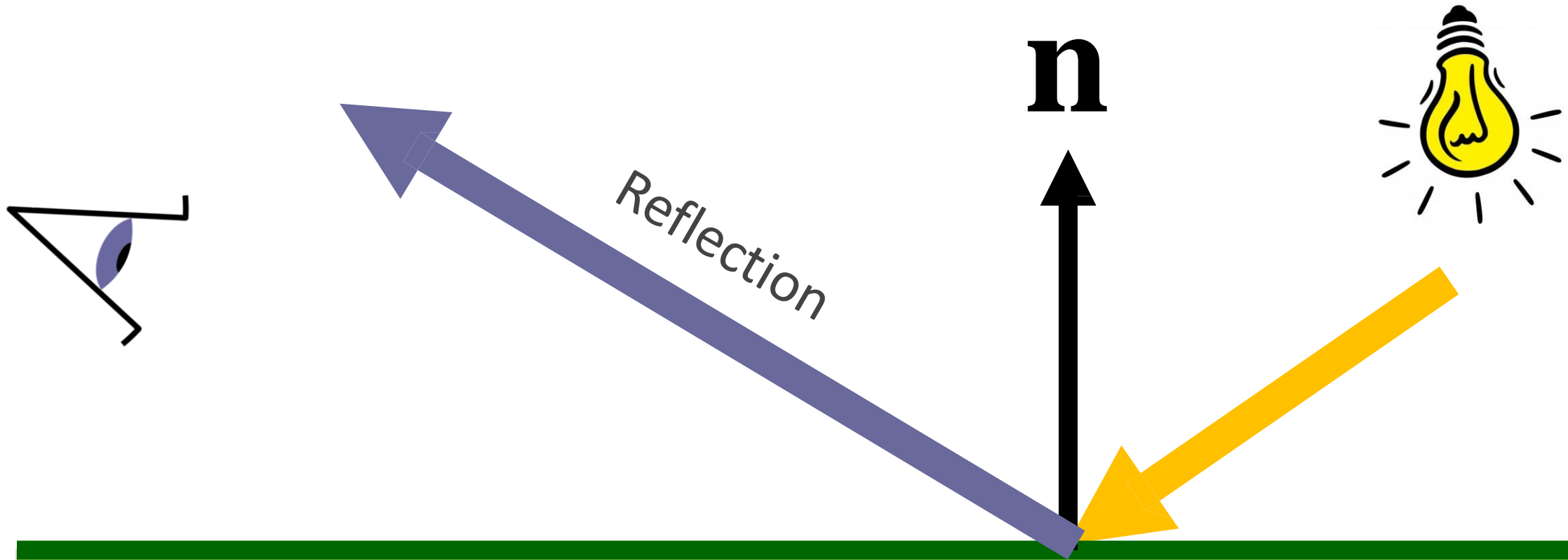
Light and Surfaces



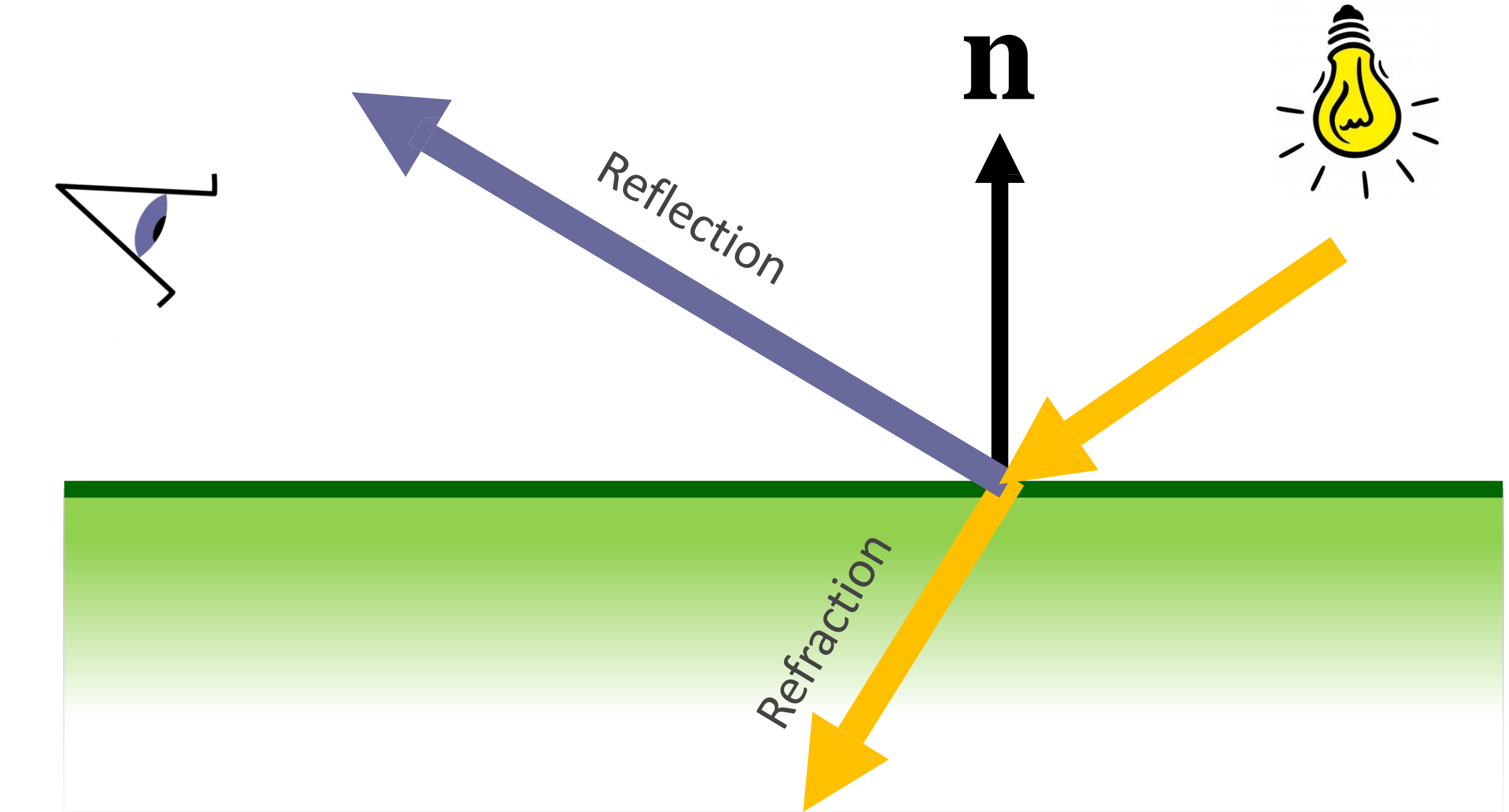
n



Light and Surfaces

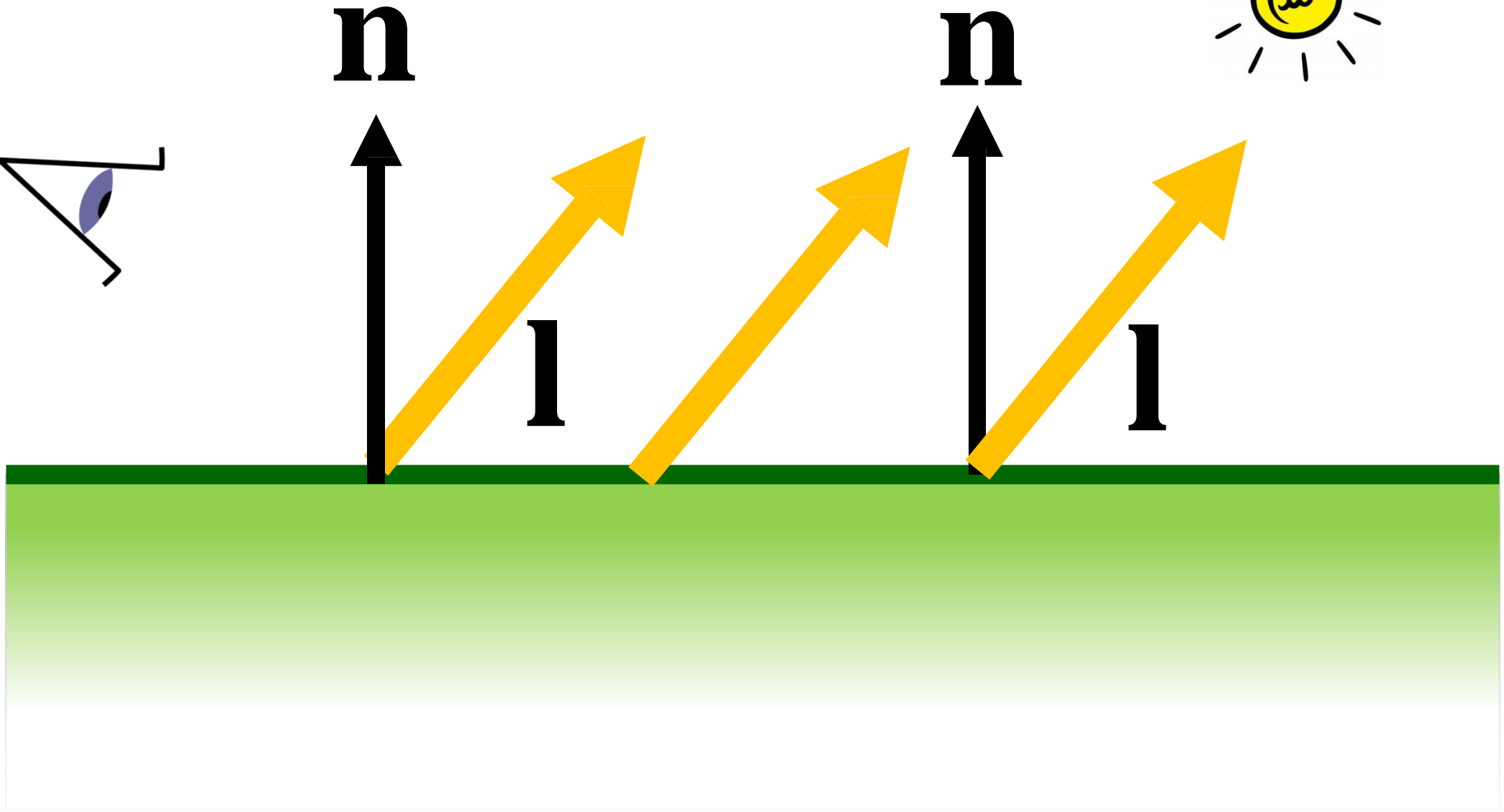
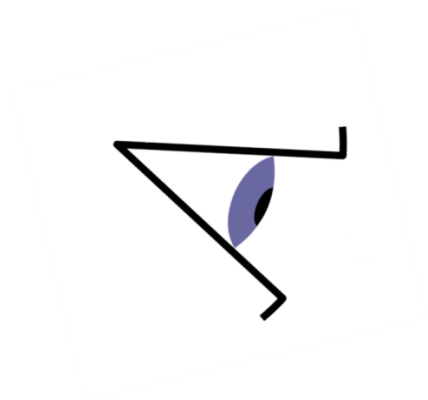


Light and Surfaces



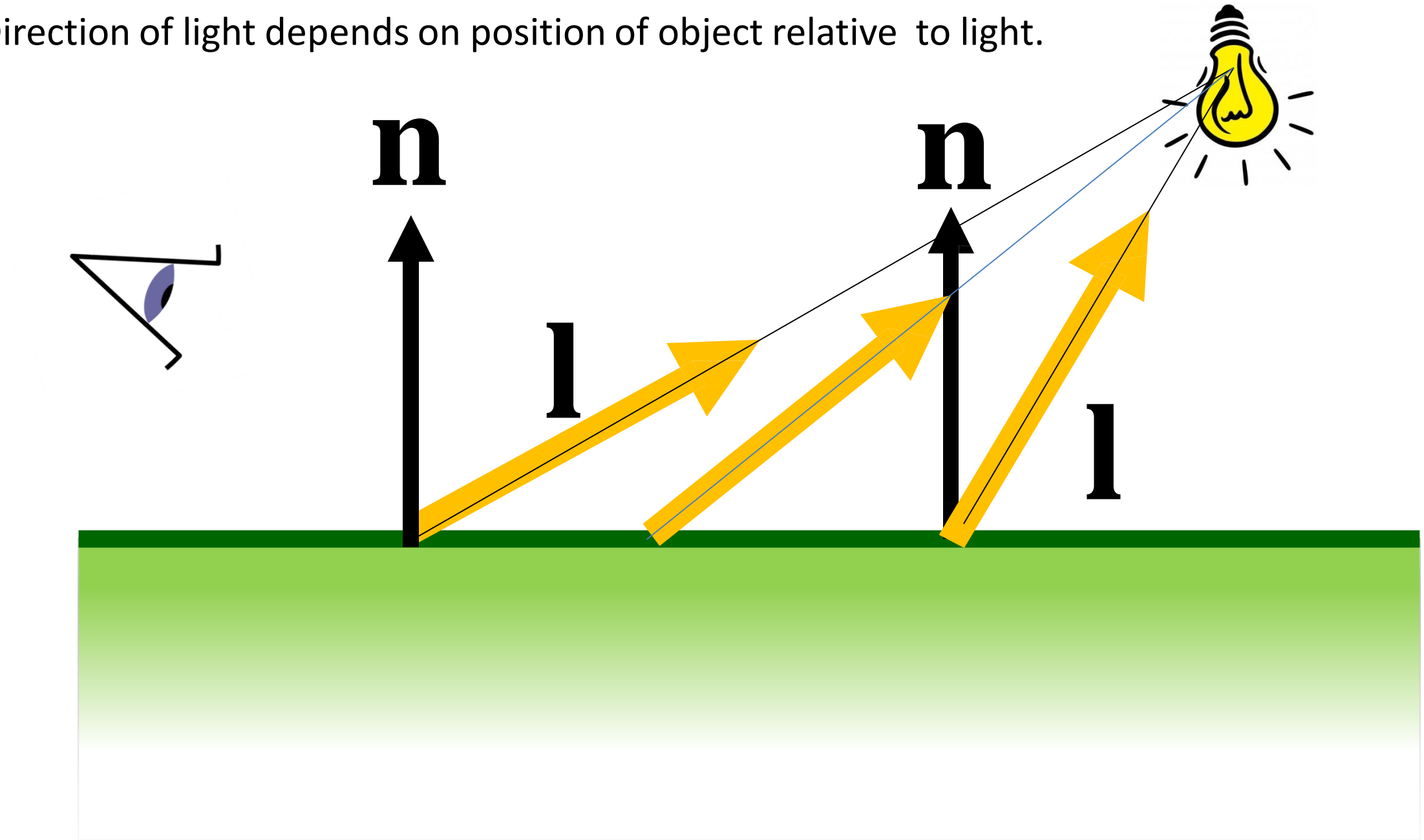
Directional Light

Direction of light is independent of the object. **Light is very far away**



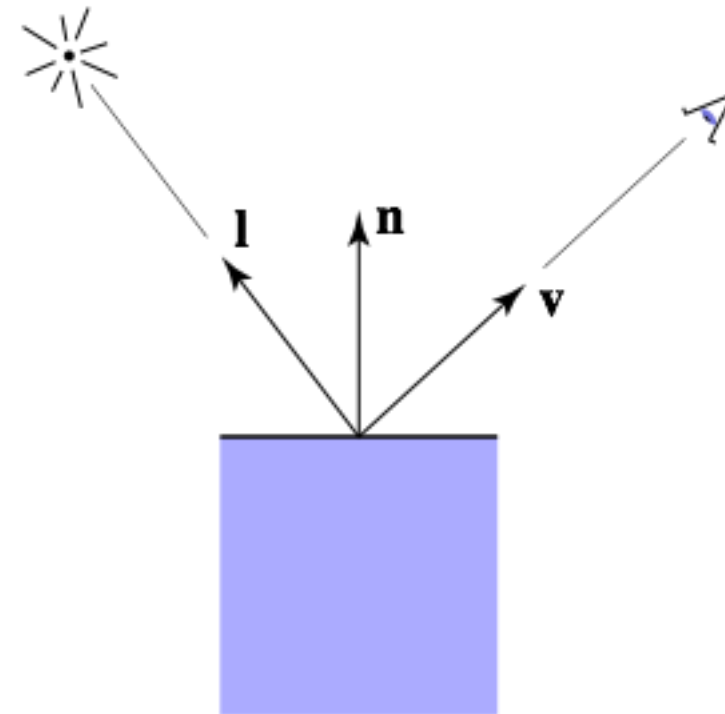
Point Light

Direction of light depends on position of object relative to light.



Shading

- Compute light reflected toward camera
- Inputs:
 - eye direction
 - light direction
(for each of many lights)
 - surface normal
 - surface parameters
(color, shininess, ...)



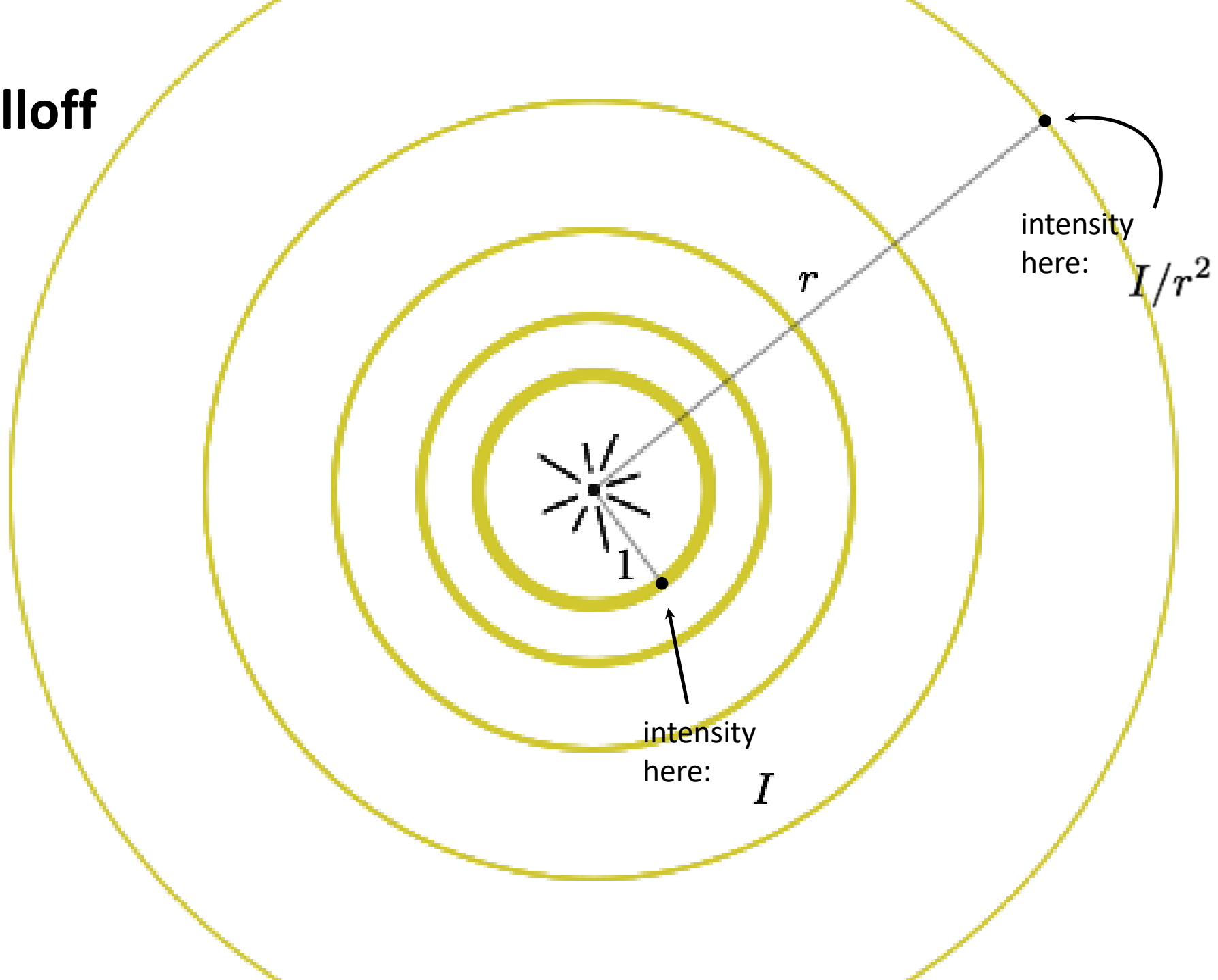
Computing the Normal at a Hit Point

- Polygon normal: cross product of two non-collinear edges.
- Implicit surface normal $f(p)=0$:
 $\text{gradient}(f)(p)$.
- Explicit parametric surface $f(a,b)$:

$$\delta f(s,b)/\delta s \times \delta f(a,t)/\delta t.$$

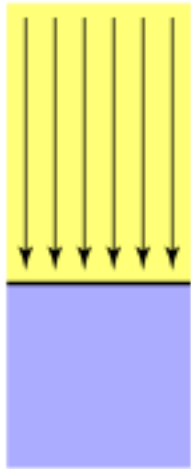


Light falloff

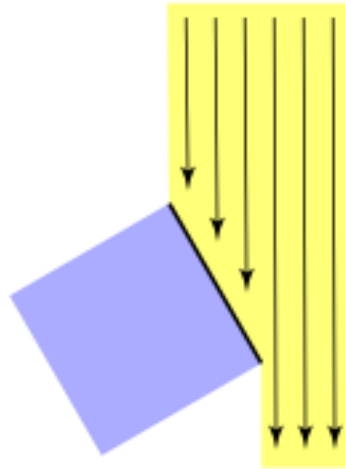


Diffuse reflection

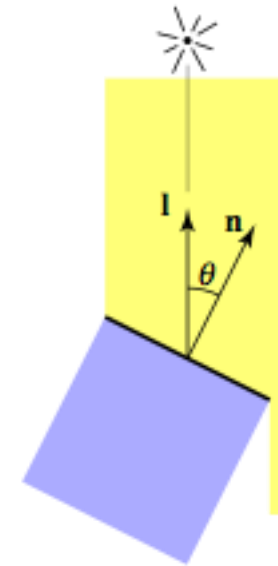
- Light is scattered uniformly in all directions
 - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube
receives a certain
amount of light



Top face of
60° rotated cube
intercepts half the light



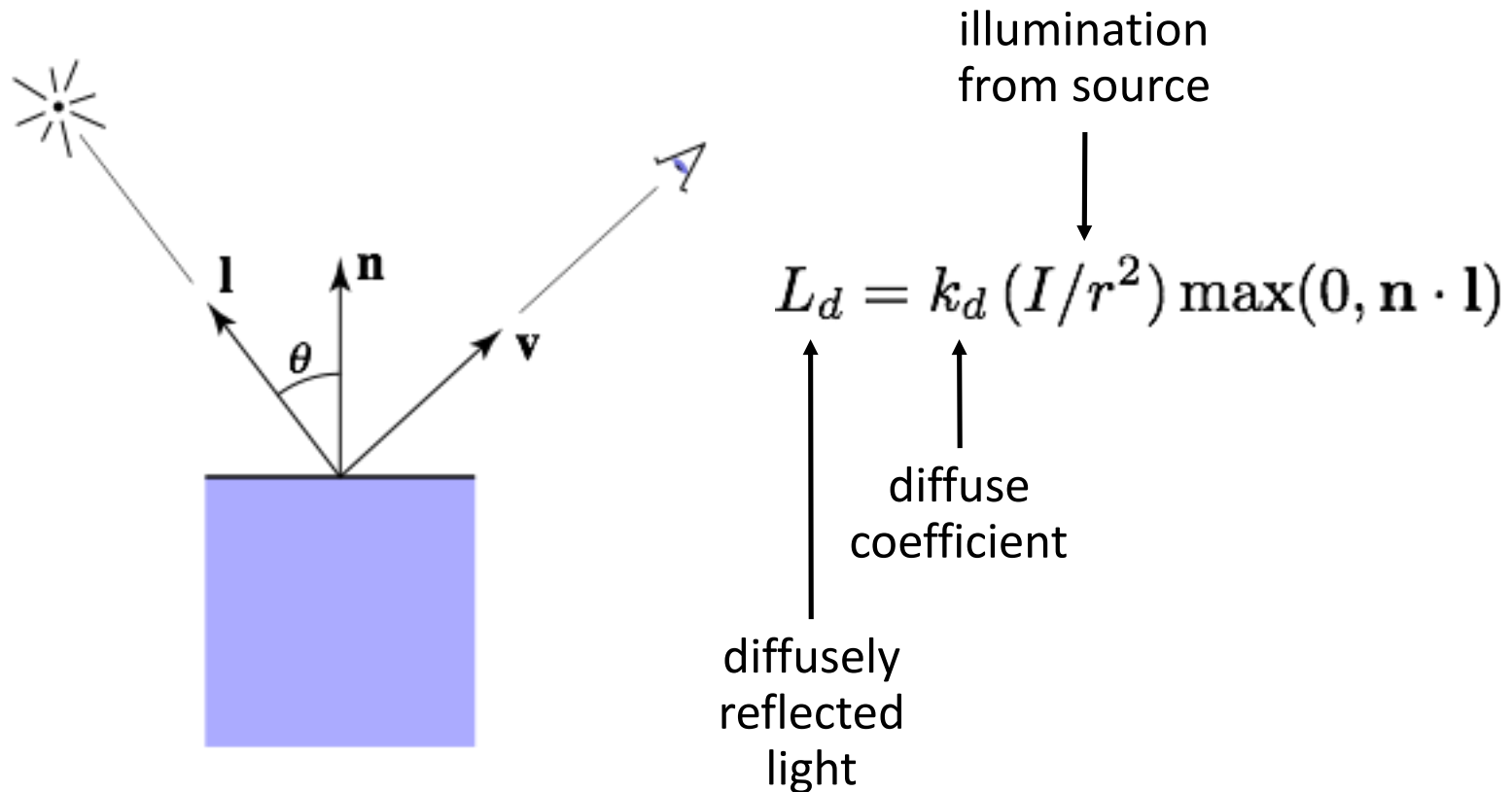
In general, light per unit
area is proportional to

$$\cos \theta = \mathbf{l} \cdot \mathbf{n}$$



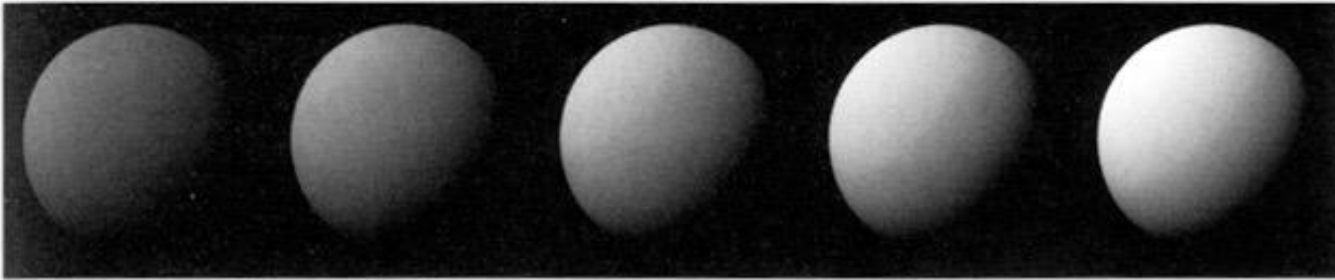
Lambertian shading

Shading independent of view direction



Lambertian shading

Produces a matte appearance



$k_d \longrightarrow$

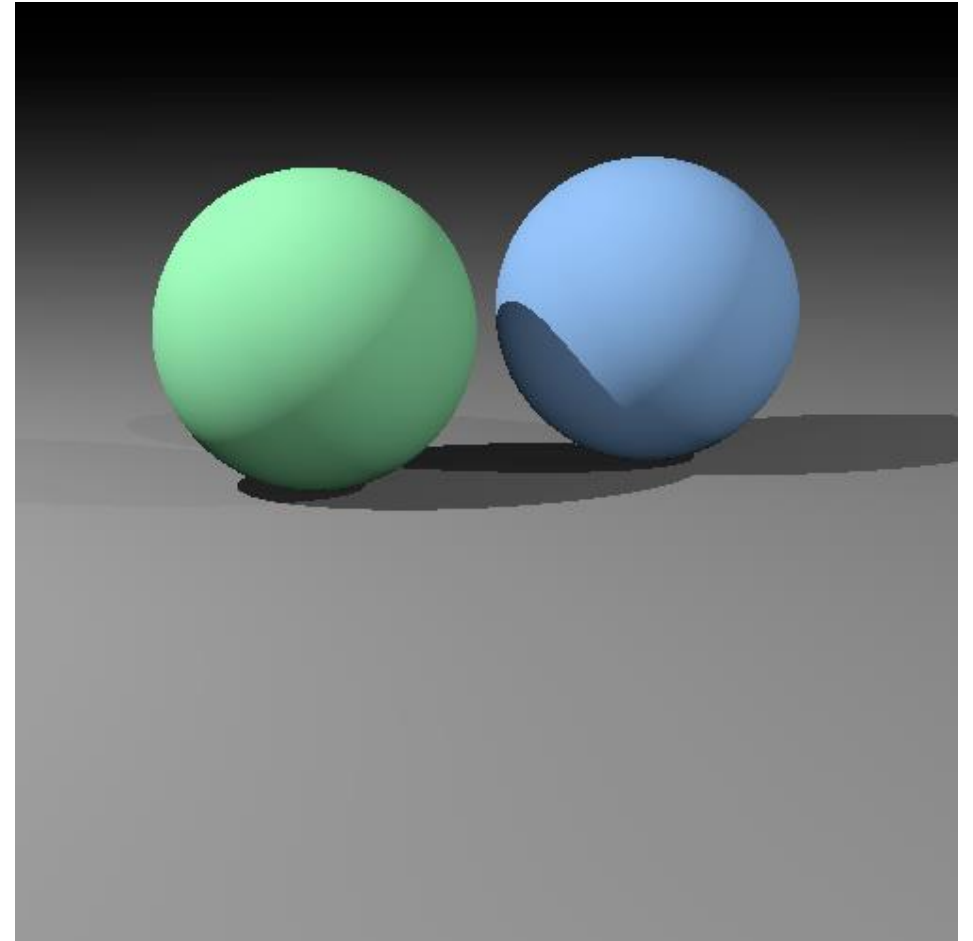


Image without shading

```
for 0 <= iy < ny
  for 0 <= ix < nx
  {
    ray = camera.getRay(ix, iy);
    firstSurface = scene.intersect(result,ray);
    if (firstSurface)
      image.set(ix, iy, firstSurface.color);
    else
      image.set(ix, iy, background.color);
  }
```

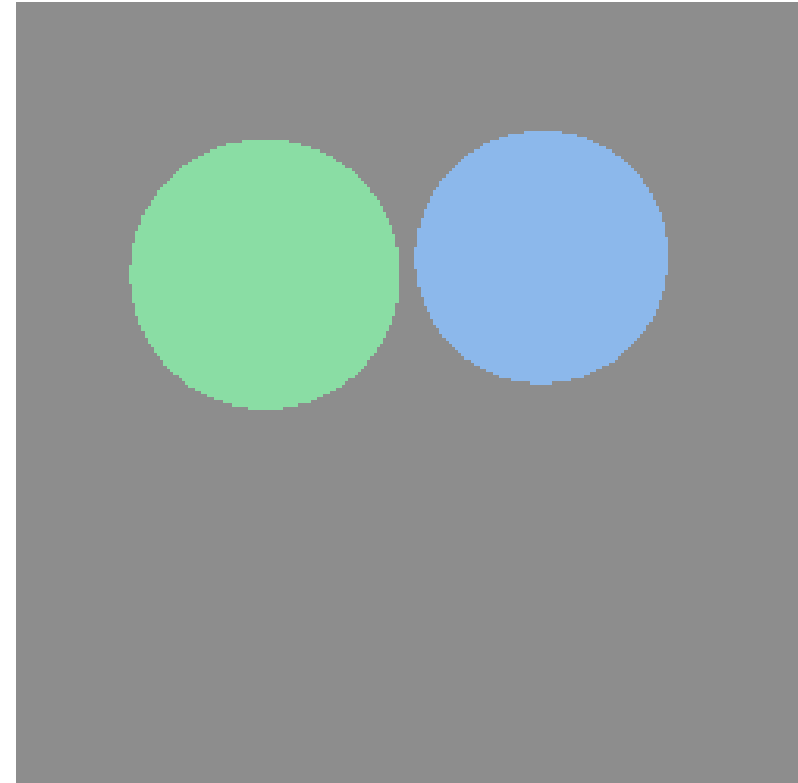
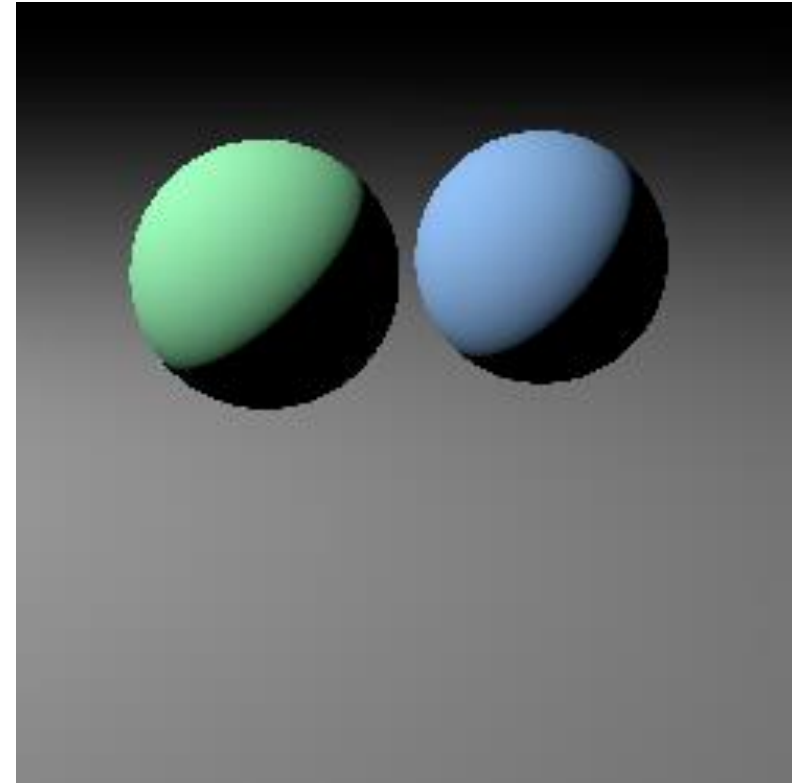


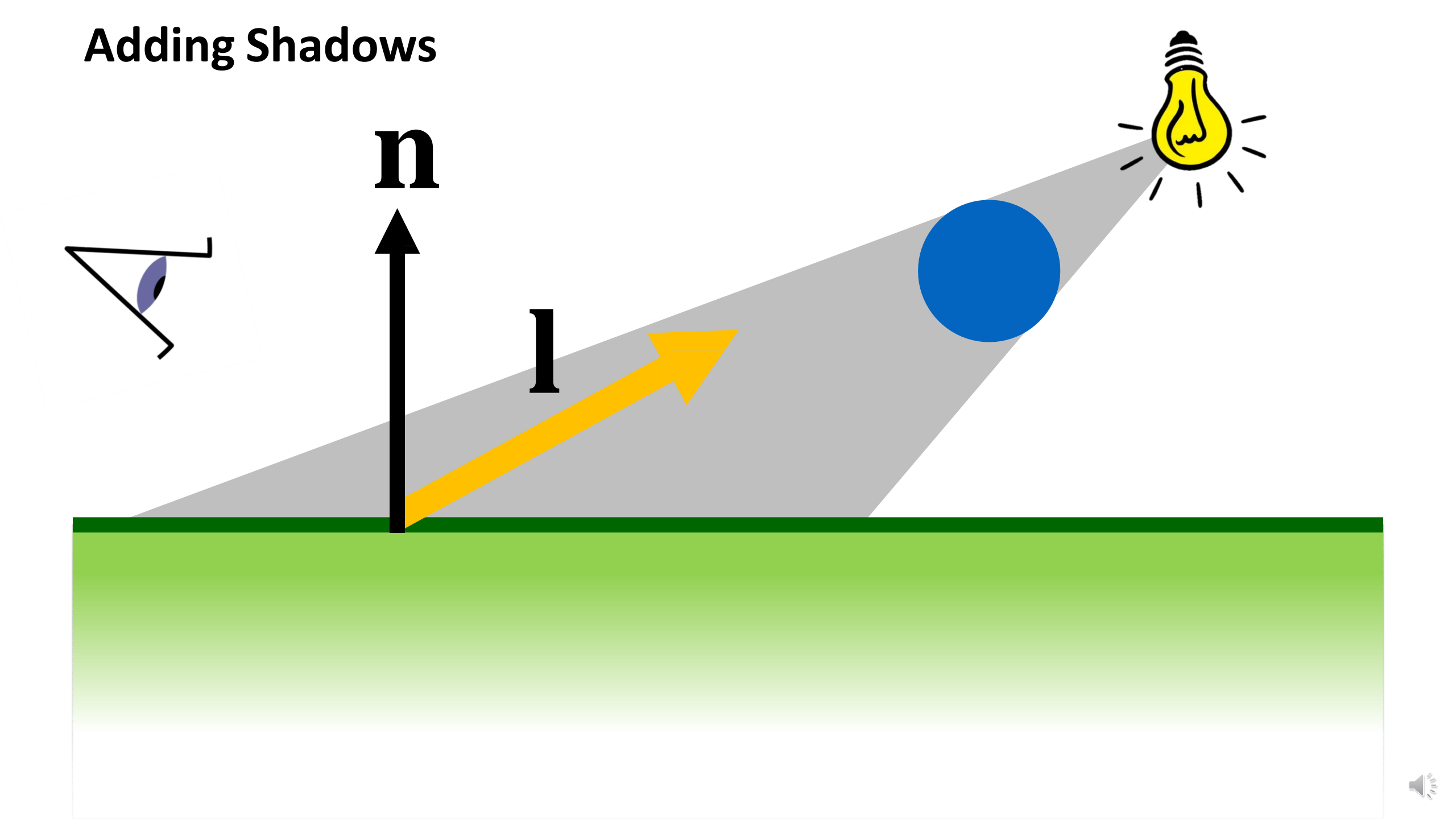
Image with shading

```
for 0 <= iy < ny
  for 0 <= ix < nx
  {
    ray = camera.getRay(ix, iy);
    firstSurface = scene.intersect(result,ray);
    image.set(ix, iy,
              firstSurface.shade(ray,light,result.point,
                                result.normal);
    else
      image.set(ix, iy, background.color);
  }

Surface.shade(ray,light,point,normal) {
  l=light.pos-position;
  it= surface.k*light.intensity*max(0,normal.l);
  return surface.color*it;
}
```



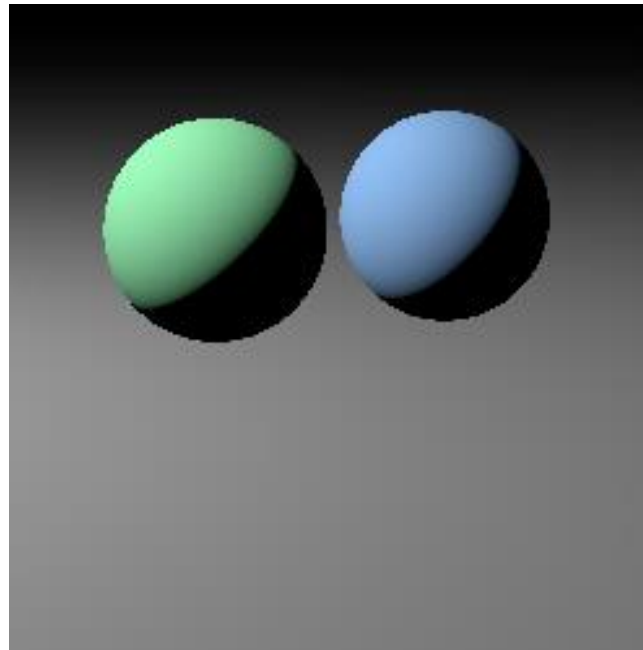
Adding Shadows



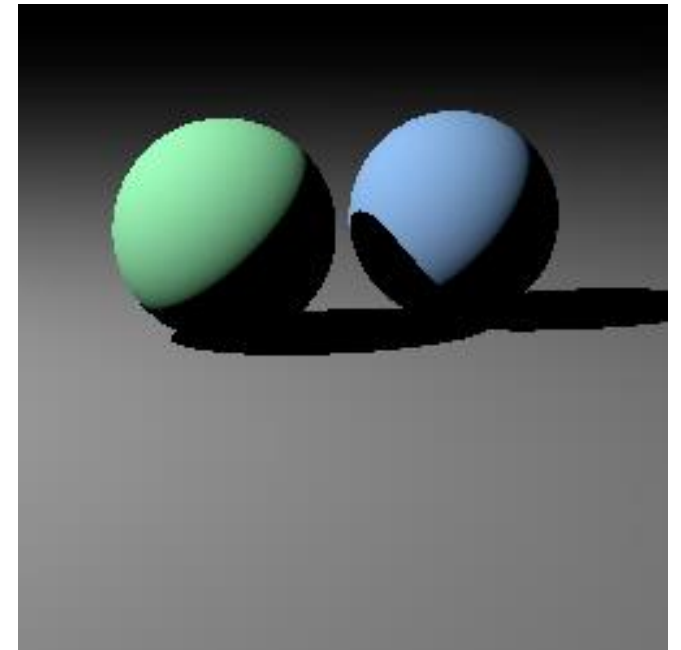
Shadows

- Surface is only illuminated if nothing blocks its view of the light.
- With ray tracing it's easy to check if a point in the scene is in shadow.
just shoot a ray from the point to the light and intersect it with the scene!

```
Surface.shade(ray,light,point,normal) {  
    l=light.pos-position;  
    shadowray=(point,l);  
    if !scene.intersect(result,shadowray)  
    {  
        it= surface.k*light.intensity*  
            max(0,normal.l);  
        return surface.color*it;  
    }  
    else  
        return black;  
}
```



Without shadows

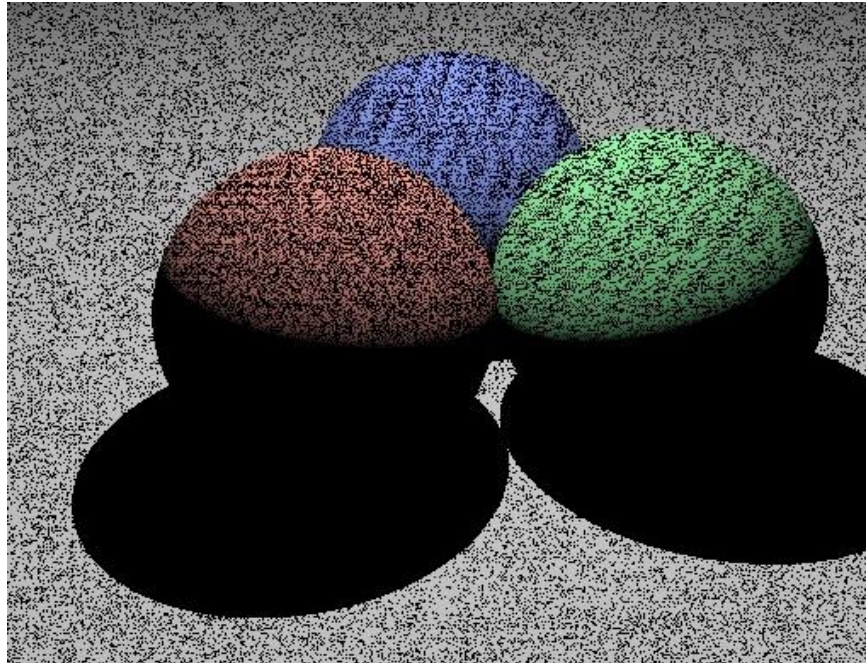


With shadows



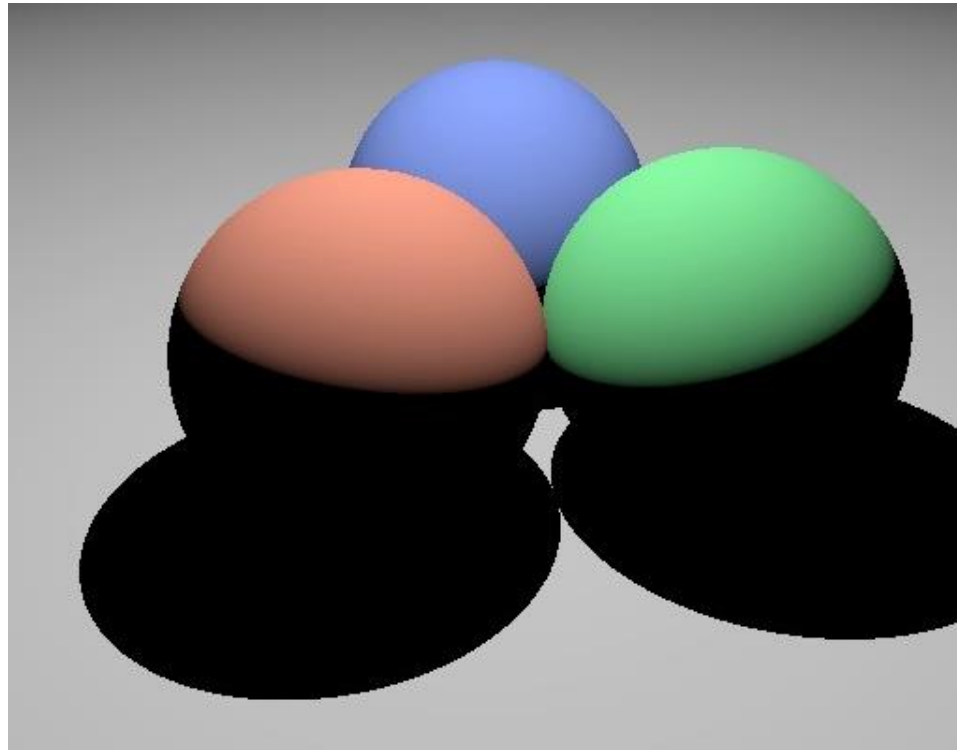
Classic shadow error

What's going on?



Classic shadow error

Start shadow rays just outside surface



Multiple lights

- Important to fill in black shadows
- Just loop over lights, add contributions
- Ambient shading
 - black shadows are not really right
 - one solution: dim light at camera
 - alternative: add a constant “ambient” color to the shading...



Ambient shading

Shading that does not depend on anything

- add constant color to account for disregarded illumination and fill in black shadows

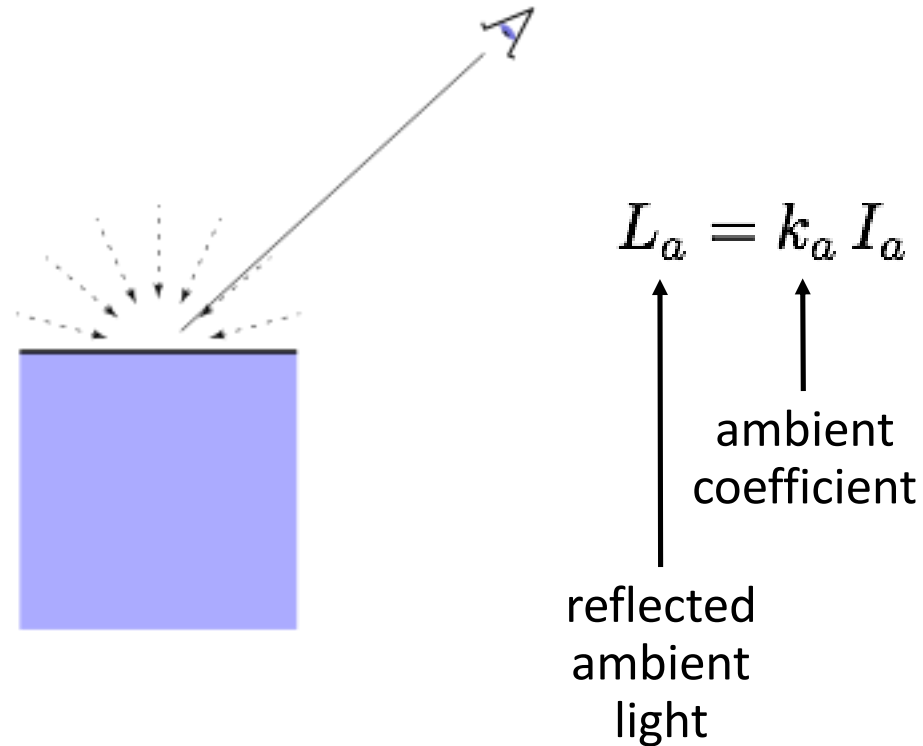
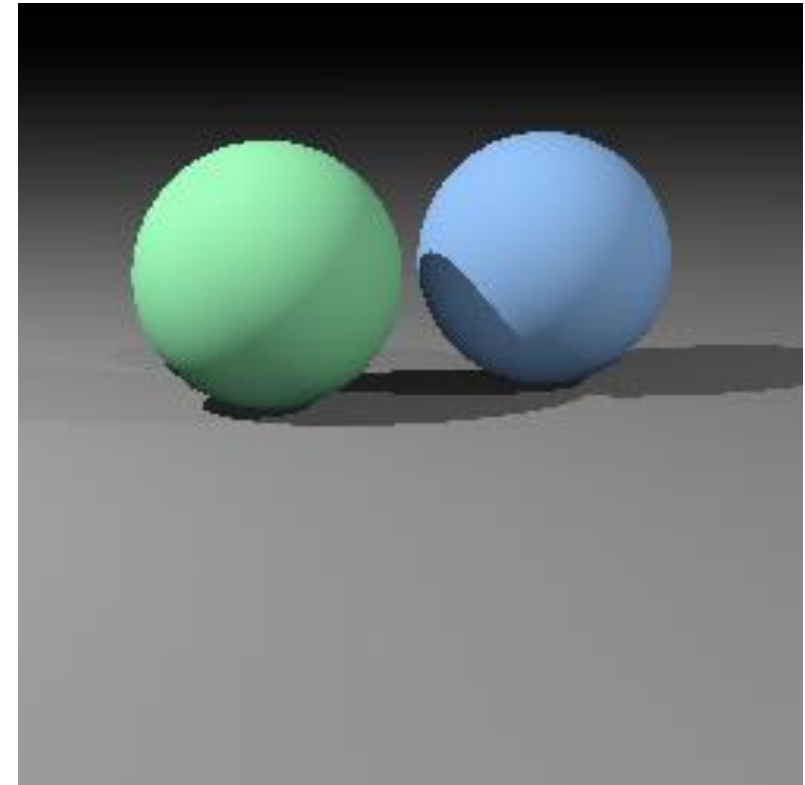


Image with multiple lights

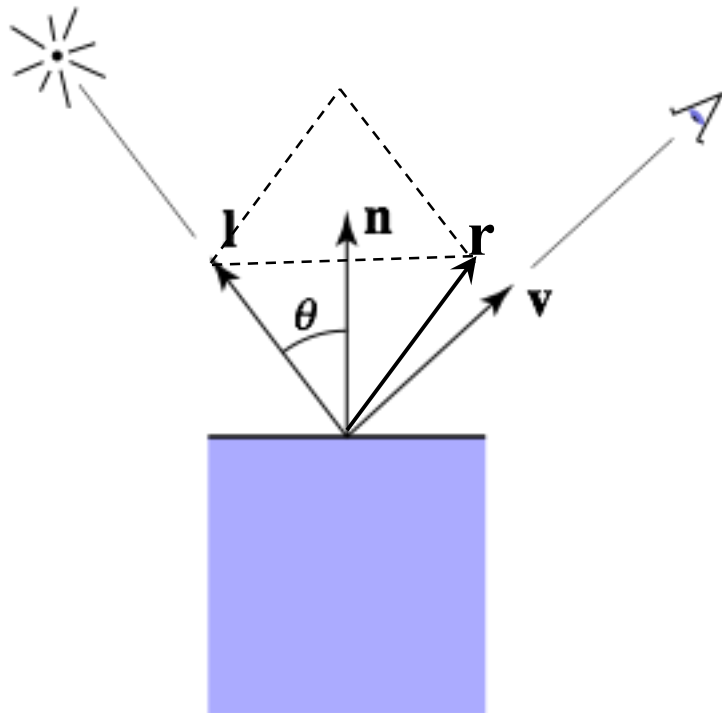
```
shade(ray, lights, point, normal) {  
    result = ambient;  
    for light in lights {  
        l=light.pos-position;  
        shadowray=(point+ $\epsilon$ *normal,l);  
        if !scene.intersect(result,shadowray)  
        {  
            it= surface.k*light.intensity*max(0,normal.l);  
            result+= surface.color*it;  
        }  
    }  
    return result;  
}
```



Mirror reflection

Intensity depends on view direction

- reflects incident light from mirror direction



$$\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$$

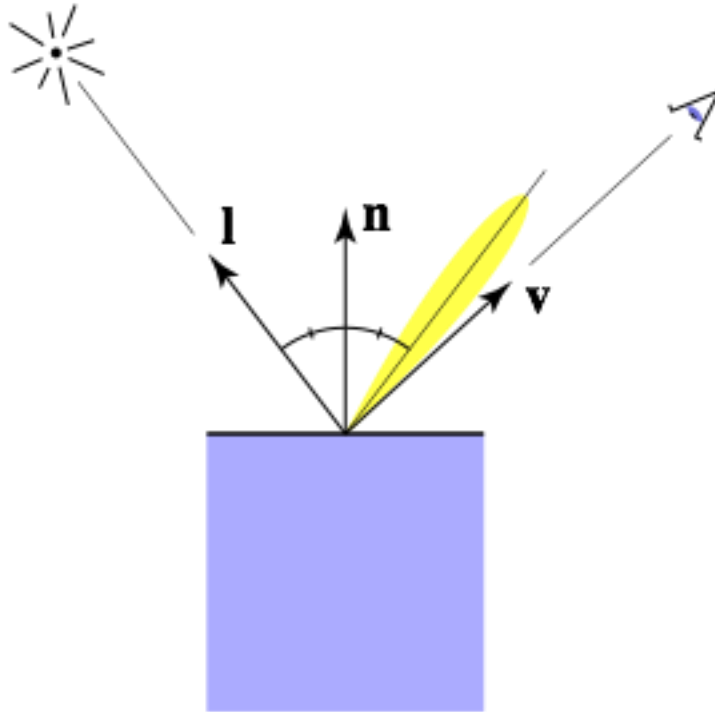


Specular shading (Phong)

Intensity depends on view direction

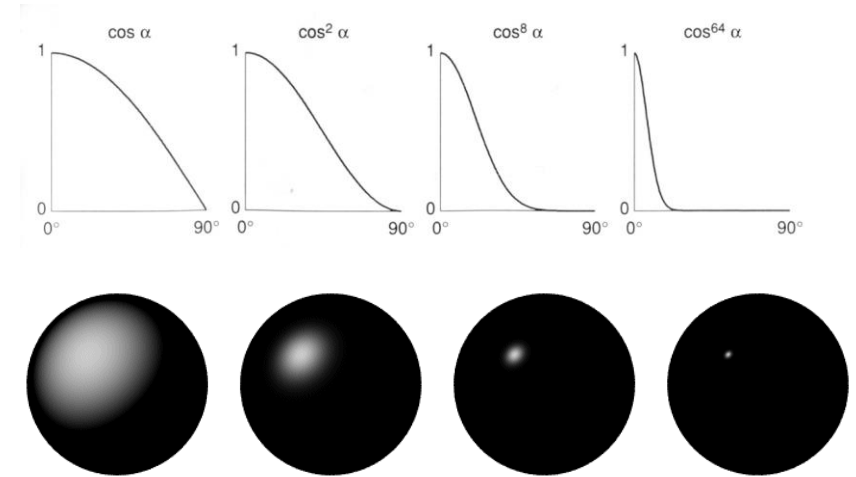
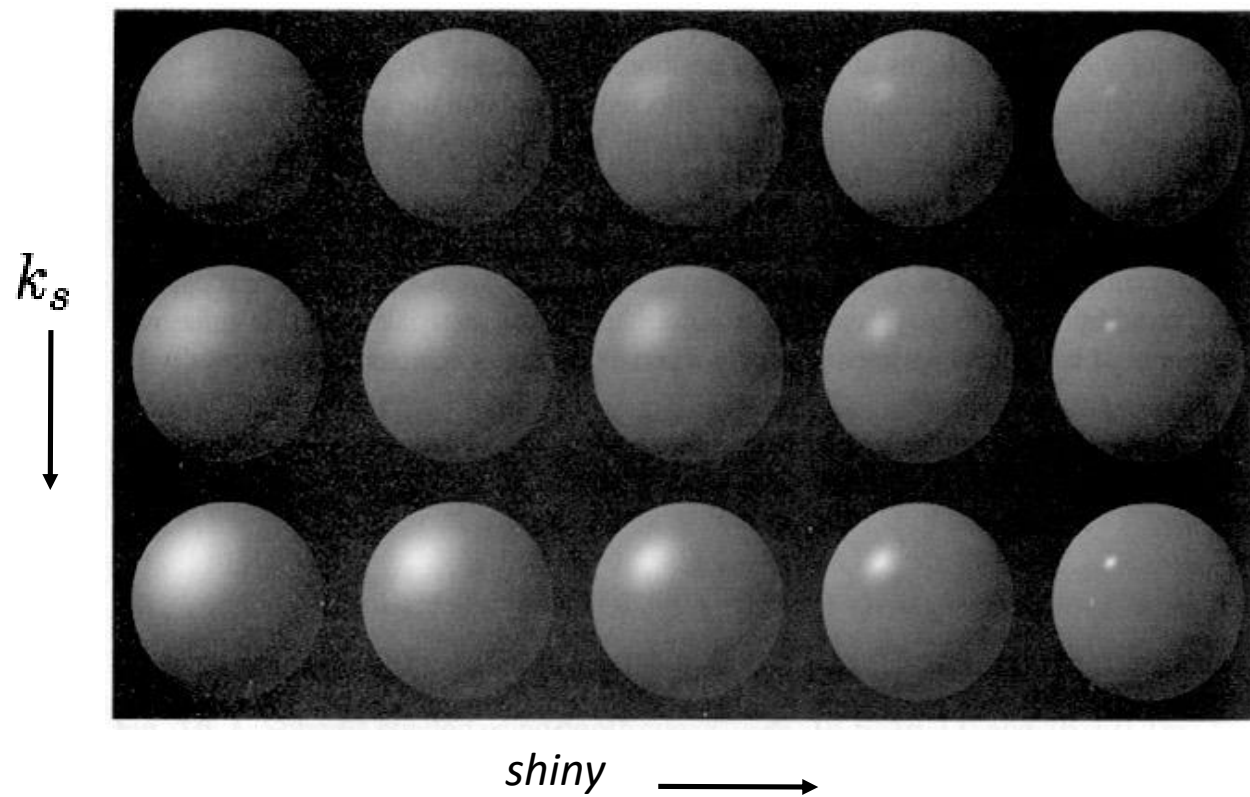
– bright near mirror configuration

$$k_s * I_s * (\mathbf{v} \cdot \mathbf{r})^{shiny}$$

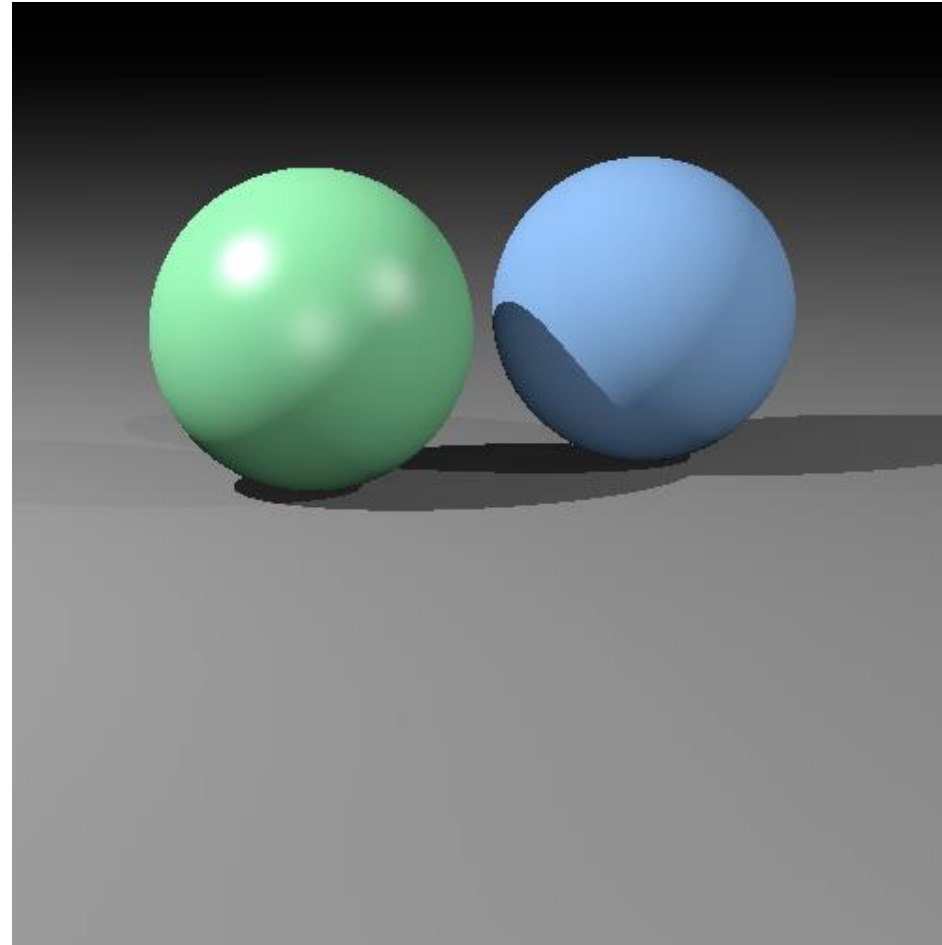


Phong model

Increasing *shiny* narrows the lobe

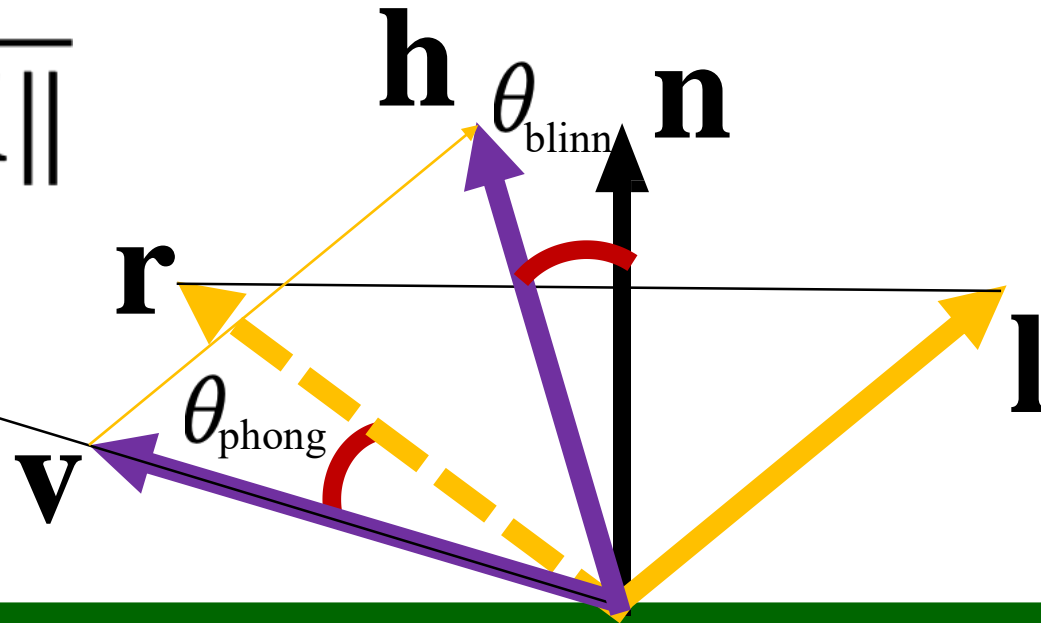
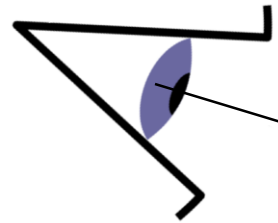


Diffuse + Specular (Phong) shading

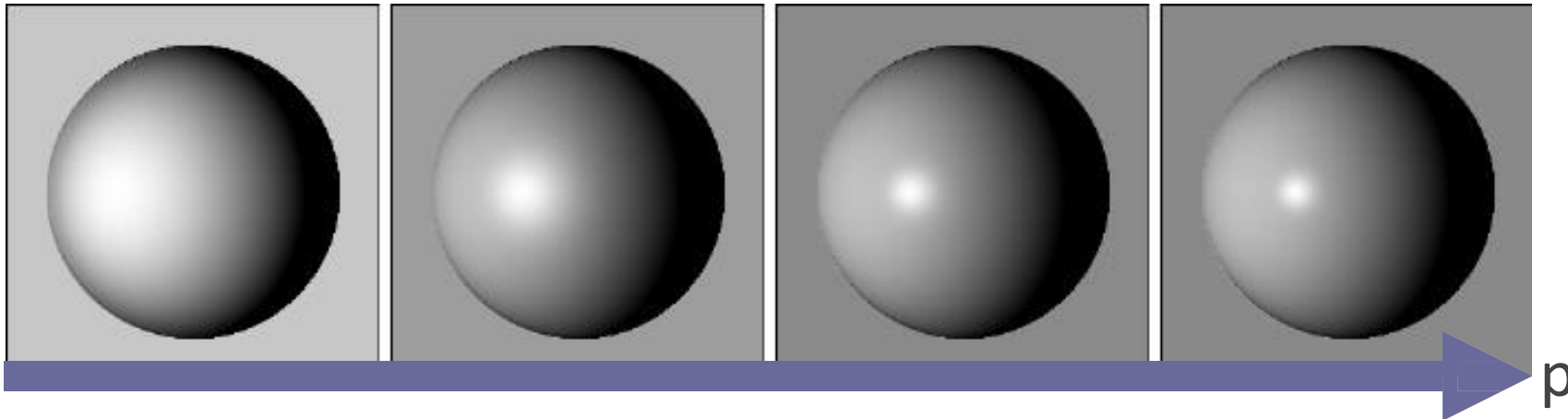
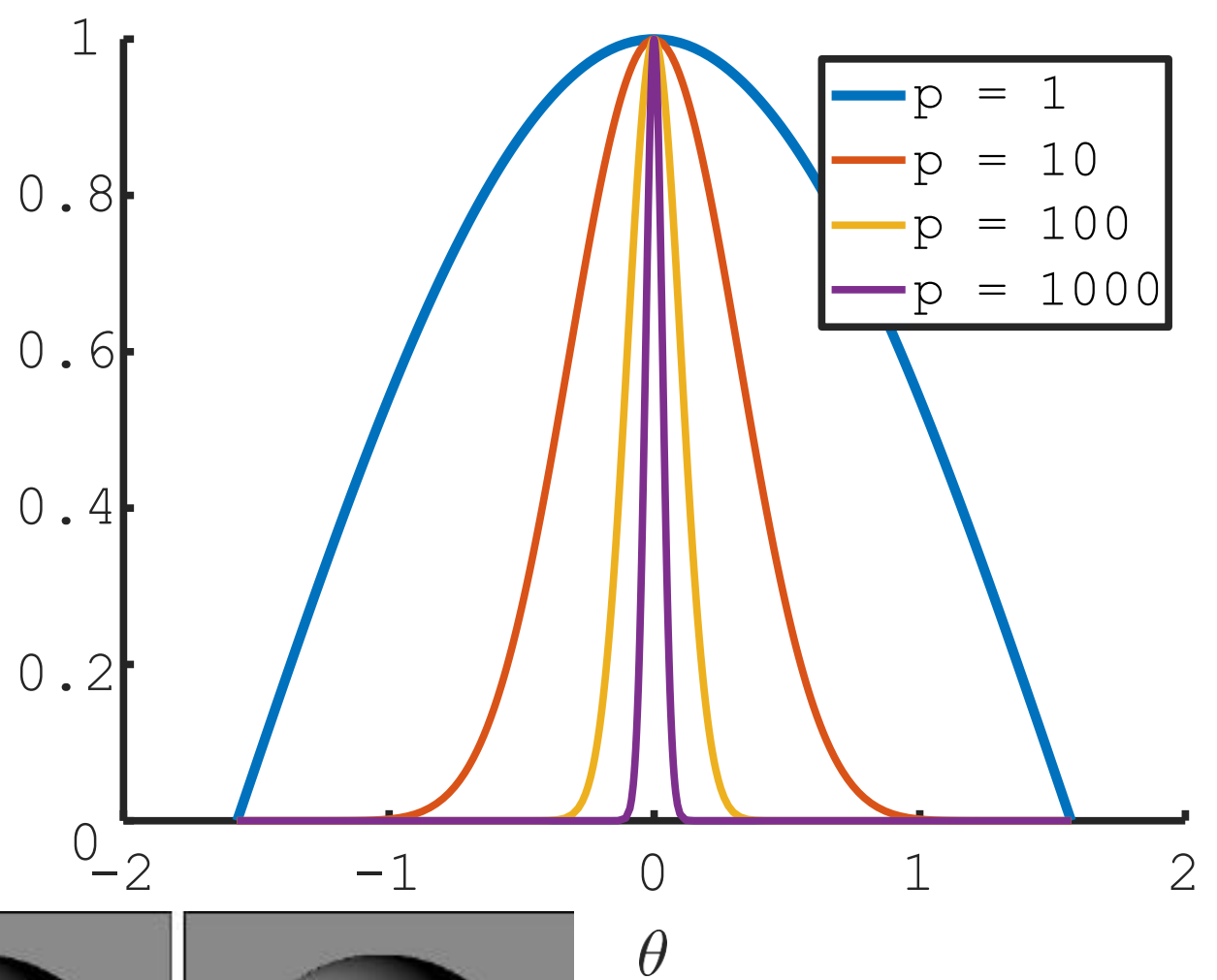


Specular Shading (Blinn)

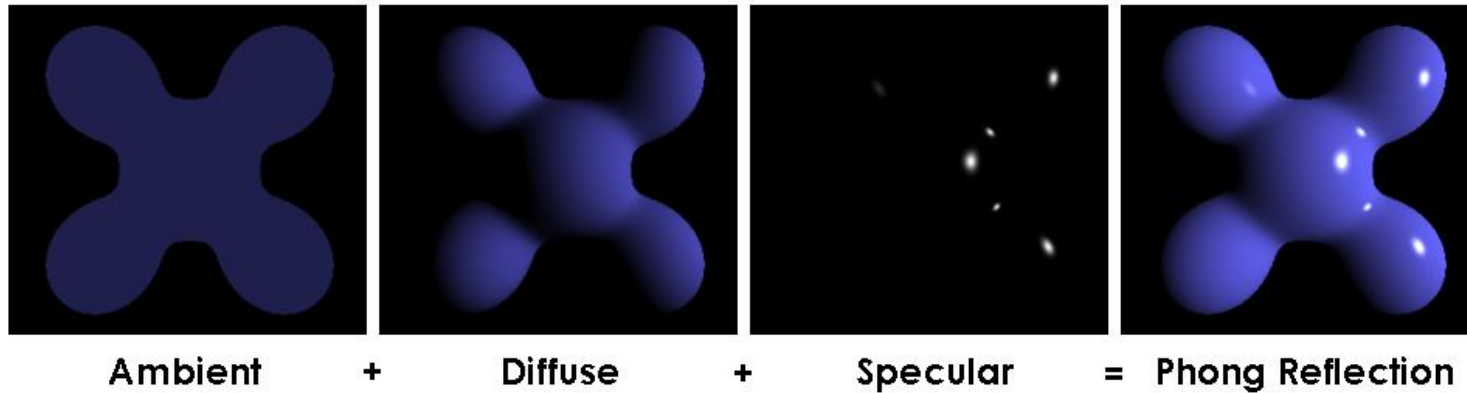
$$\mathbf{h} = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$



$$L = k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^p$$



Local Illumination



- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p \end{aligned}$$

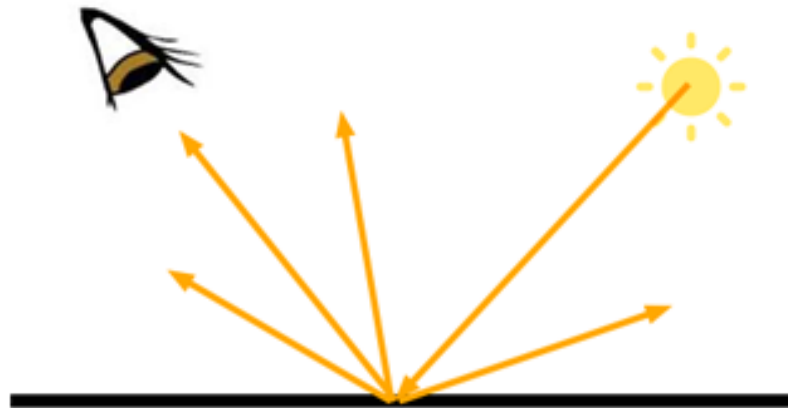
- The final result is the sum over many lights

$$\begin{aligned} L &= L_a + \sum_{i=1}^N [(L_d)_i + (L_s)_i] \\ L &= k_a I_a + \sum_{i=1}^N \left[k_d (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \right. \\ &\quad \left. k_s (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p \right] \end{aligned}$$

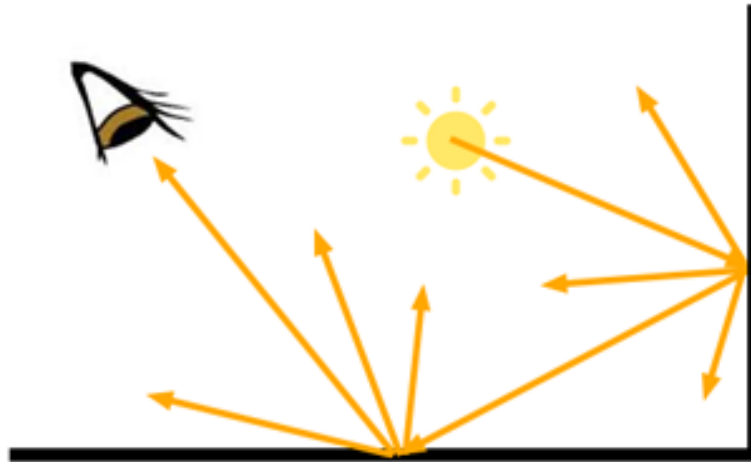


Direct Illumination

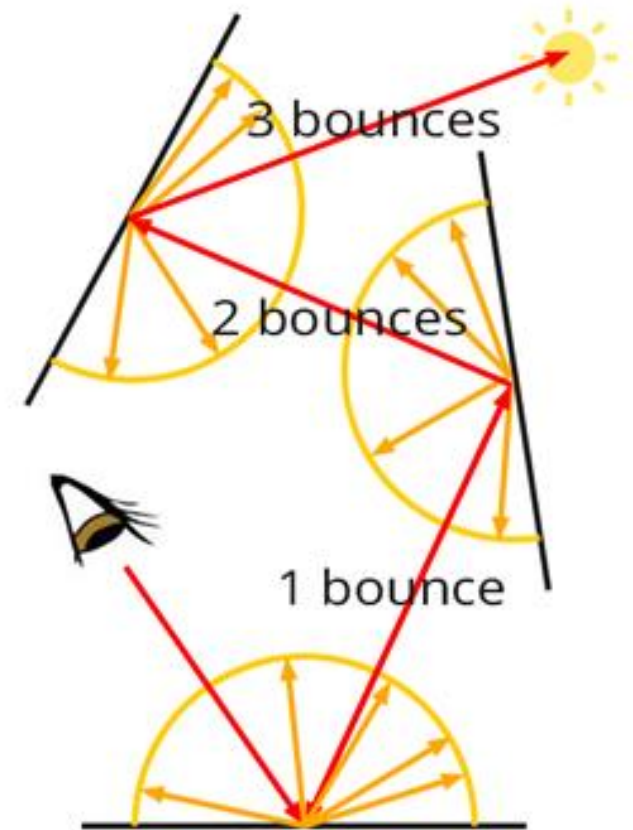
...no Global Effects so far



direct illumination



indirect illumination

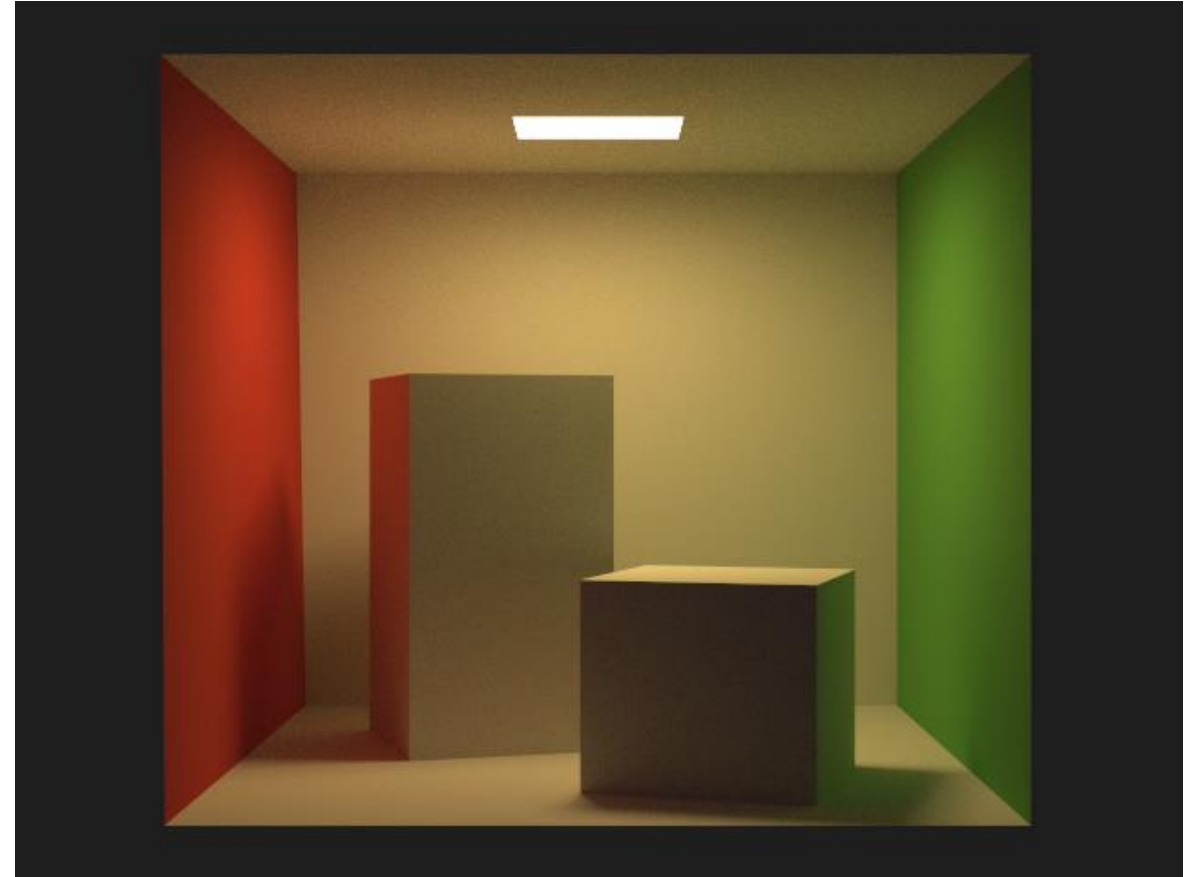
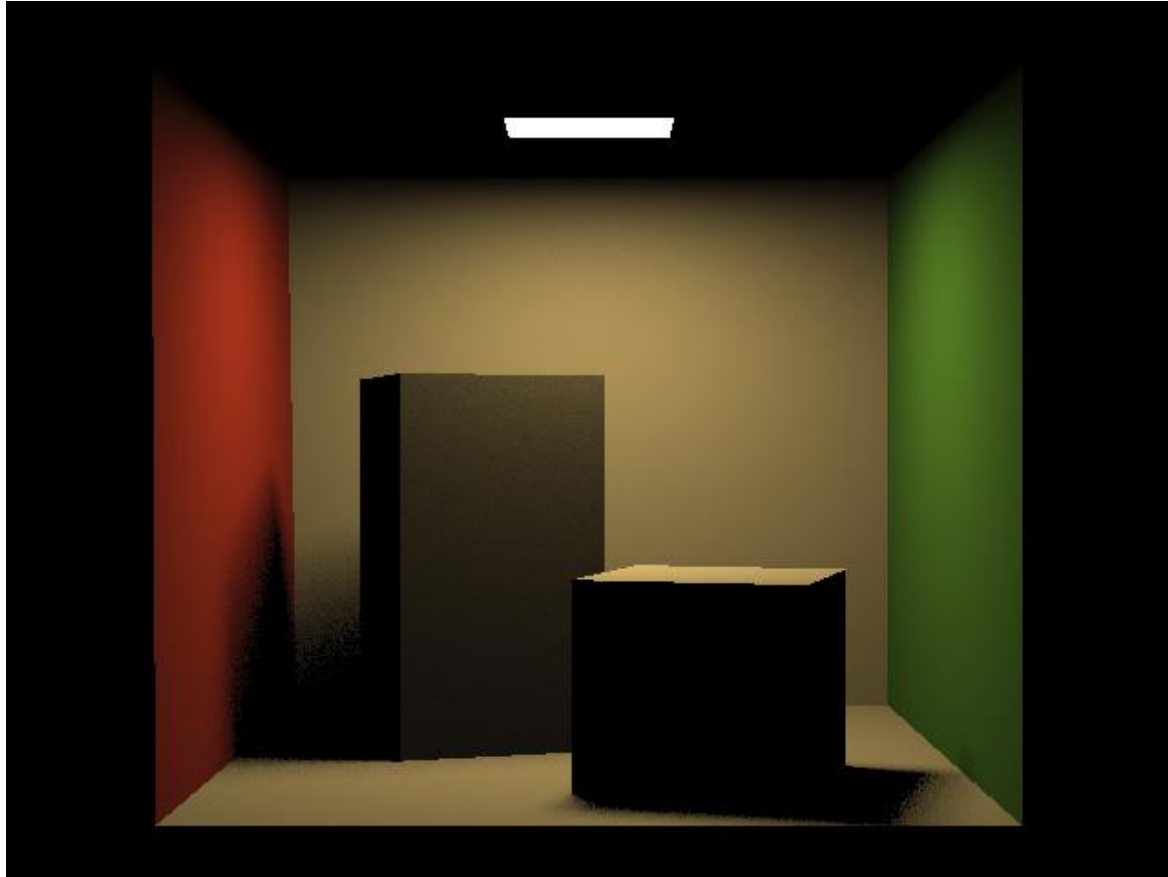


© www.scratchapixel.com

© www.scratchapixel.com



Direct vs. Indirect Illumination



<http://www.deluxerender.com/2017/01/the-cornell-box-a-renderers-rite-of-pathage/>
https://en.wikipedia.org/wiki/Cornell_box



Local vs. Global Illumination

Local Illumination Models

- e.g. Phong, Blinn.
- Model source from a light reflected once off a surface towards the eye.
- Indirect light is included with an ad hoc “ambient” term which is normally constant across the scene.

Global Illumination Models

- e.g. recursive ray tracing (incomplete model).
- Try to measure light propagation in the scene.
- Model interaction between objects, other objects, and their environment

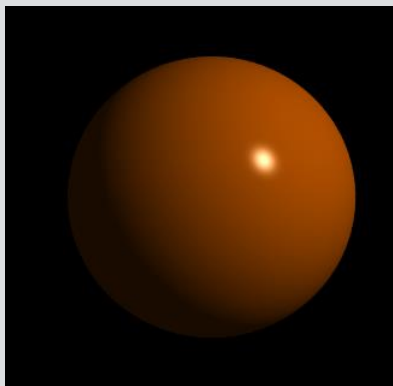


Path Tracing

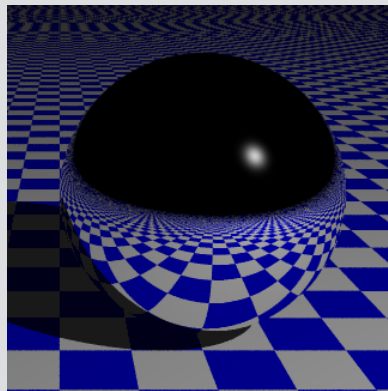
- A ray from a light L can bounce of any number of specular S and diffuse objects D before entering the eye E. The paths from E to L for eg. can be
E-D-L, E-S-D-D-S-S-D-S-L, E-D-D-S-D-S-L...
- Rays are infinitely thin
- Don't disperse
- Ray Tracing model shiny objects exhibiting multiple reflections, i.e. paths of the form
E - S* - D+ - L.



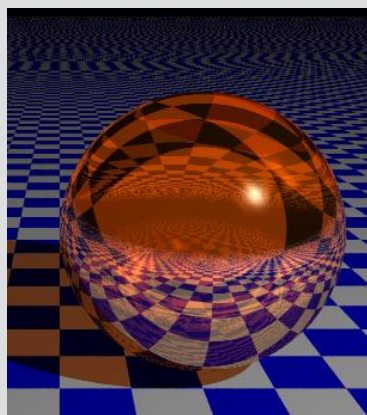
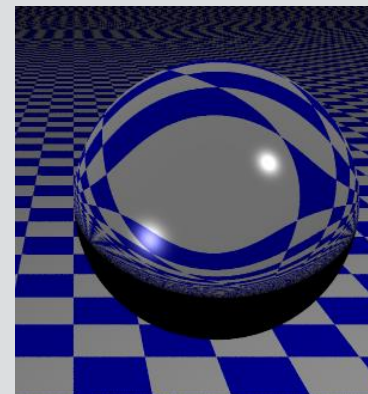
local illumination



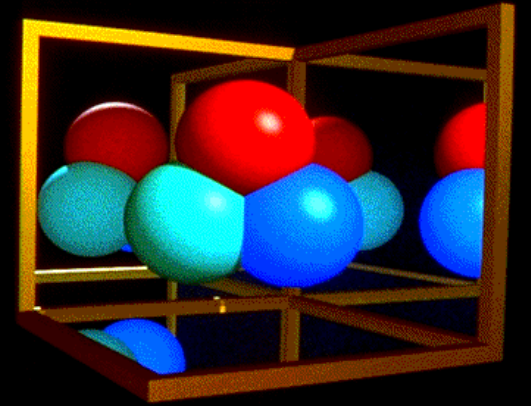
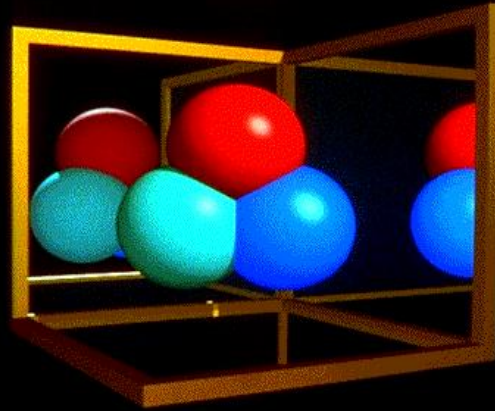
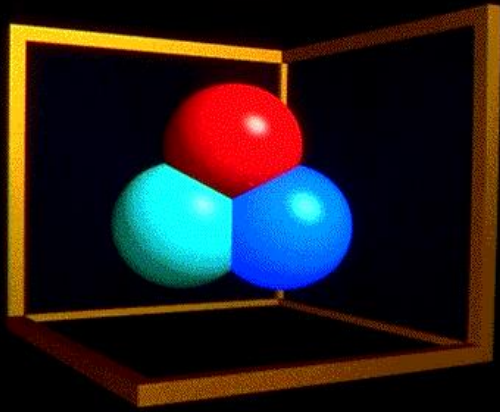
reflection



refraction

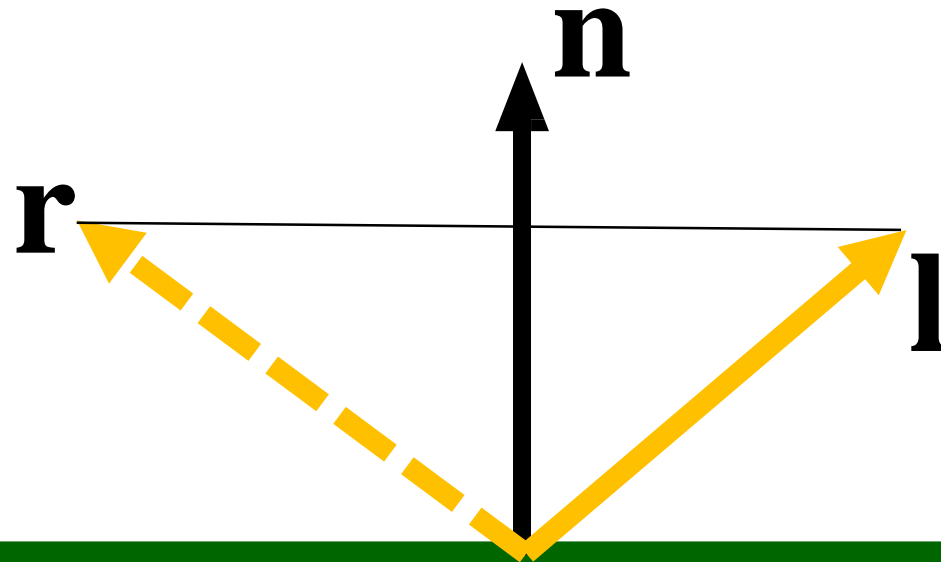


Ray Tracing recursion



Reminder: reflected ray

$$\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$$



Ray Tracing

```
for each pixel in the image {  
    pixel colour = rayTrace(viewRay, 0)  
}
```

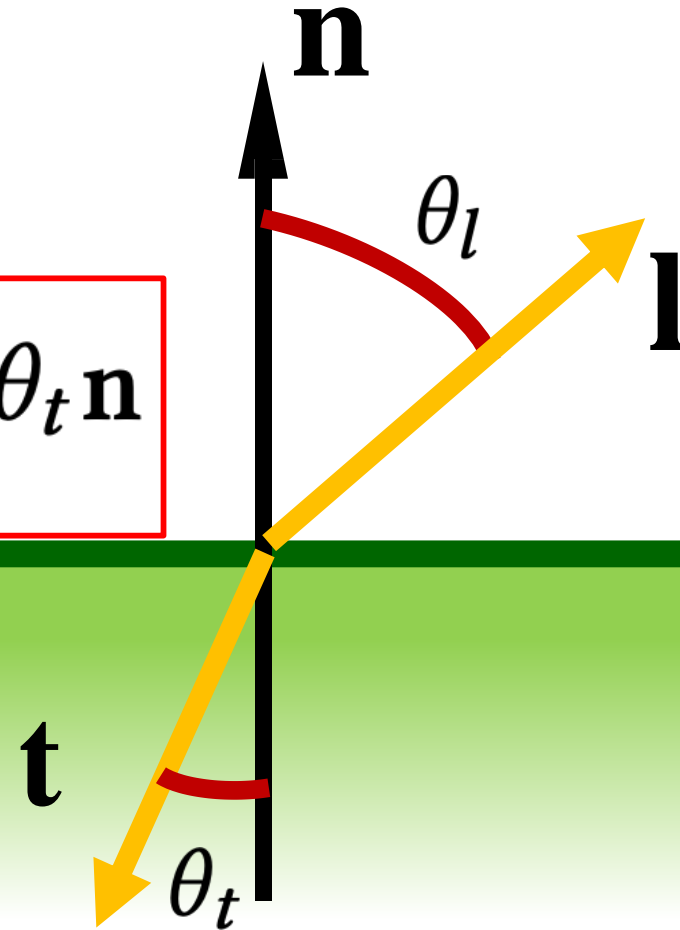
```
colour rayTrace(Ray, depth) {  
    for each object in the scene {  
        if(Intersect ray with object) {  
            colour = shading model  
            if(depth < maxDepth)  
                colour += rayTrace(reflectedRay, depth+1)  
        }  
    }  
    return colour  
}
```



Refraction (Snell's Law)

$$c_l \sin(\theta_l) = c_t \sin(\theta_t)$$

$$\mathbf{t} = -\frac{c_l}{c_t} \mathbf{l} + \frac{c_l}{c_t} \cos(\theta_l) \mathbf{n} - \cos \theta_t \mathbf{n}$$

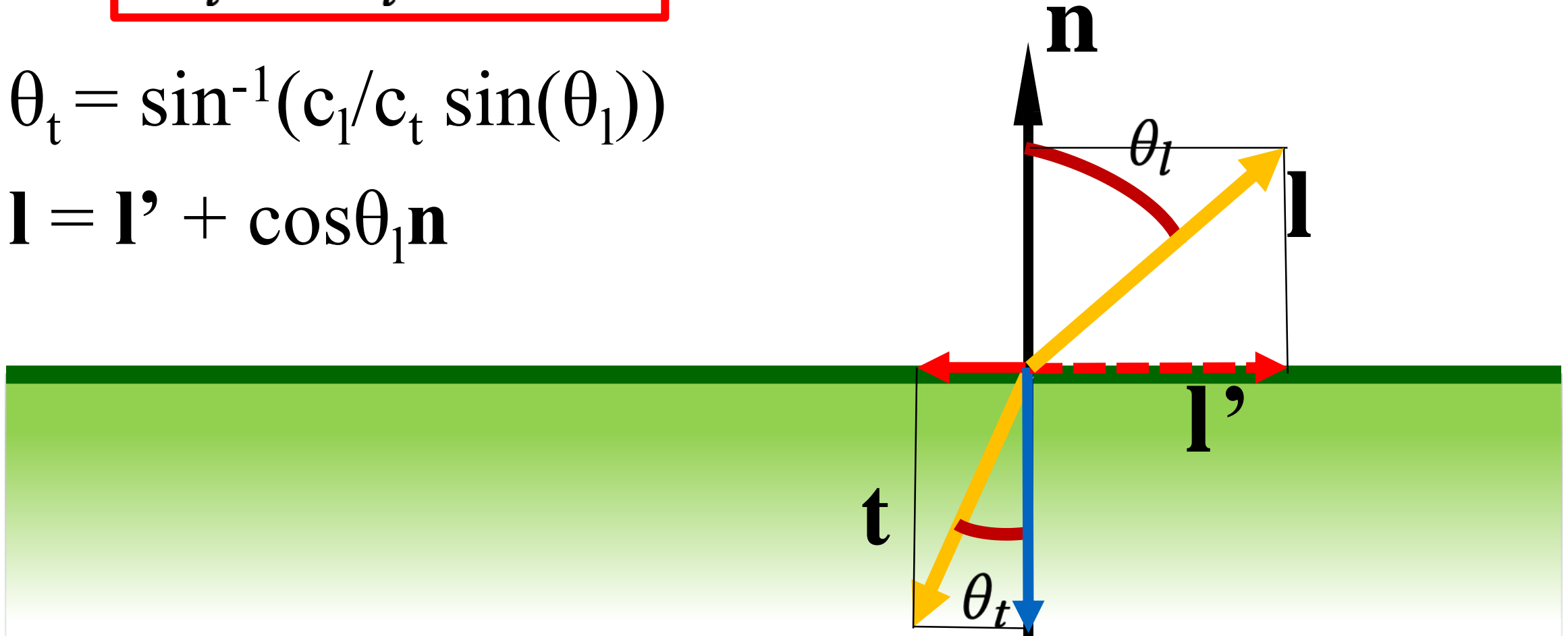


Refraction (Snell's Law)

$$\mathbf{t} = -\frac{c_l}{c_t} \mathbf{l} + \frac{c_l}{c_t} \cos(\theta_l) \mathbf{n} - \cos\theta_t \mathbf{n}$$

$$\theta_t = \sin^{-1}(c_l/c_t \sin(\theta_l))$$

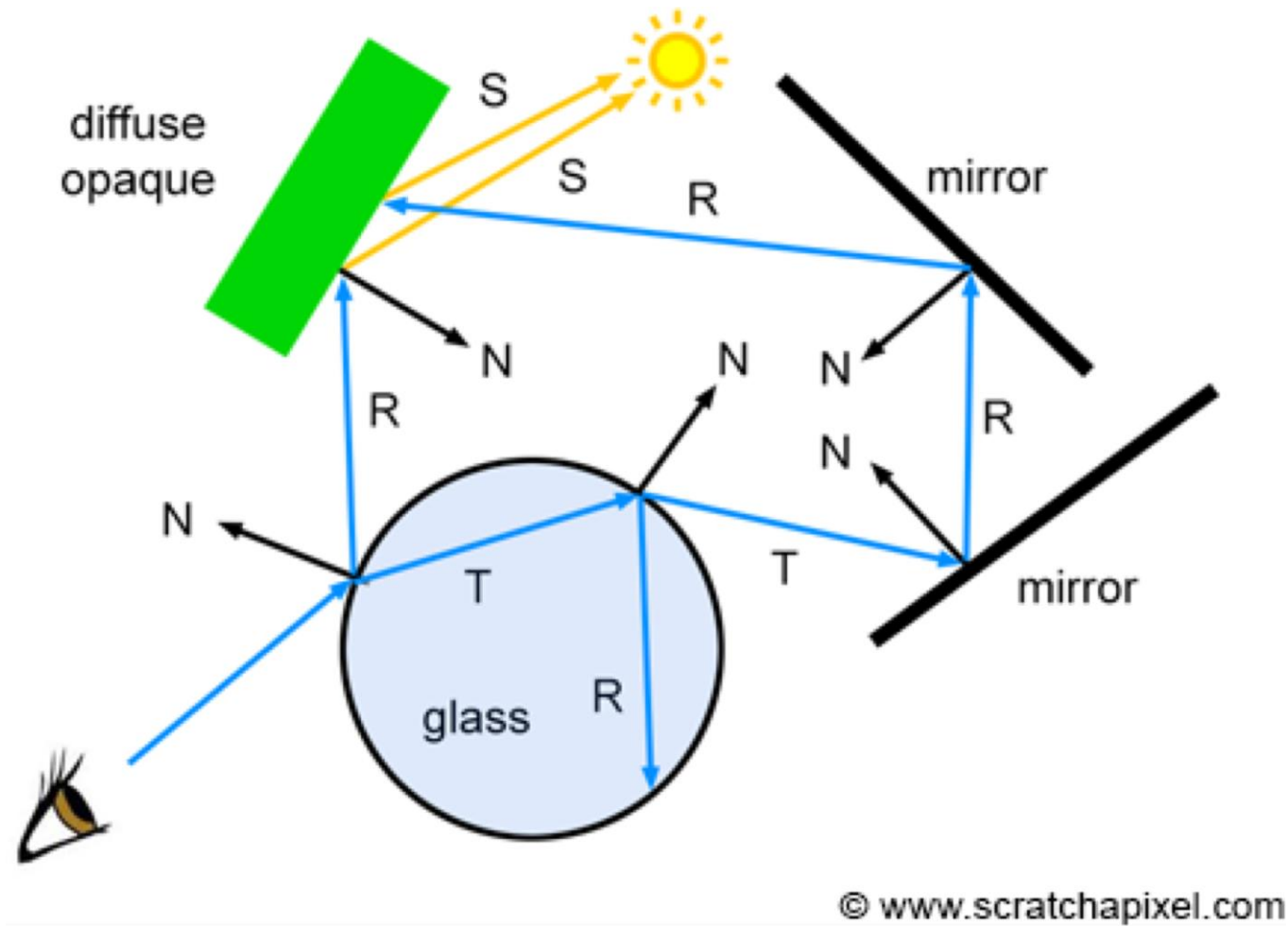
$$\mathbf{l} = \mathbf{l}' + \cos\theta_l \mathbf{n}$$



```
colour rayTrace(Ray, depth) {  
    for each object in the scene {  
        if(Intersect ray with object) {  
            colour = shading model  
            if(depth < maxDepth) {  
                colour +=  
                    rayTrace(reflectedRay, depth+1)  
                colour +=  
                    rayTrace(refractedRay, depth+1)  
            }  
        }  
    }  
    return colour  
}
```



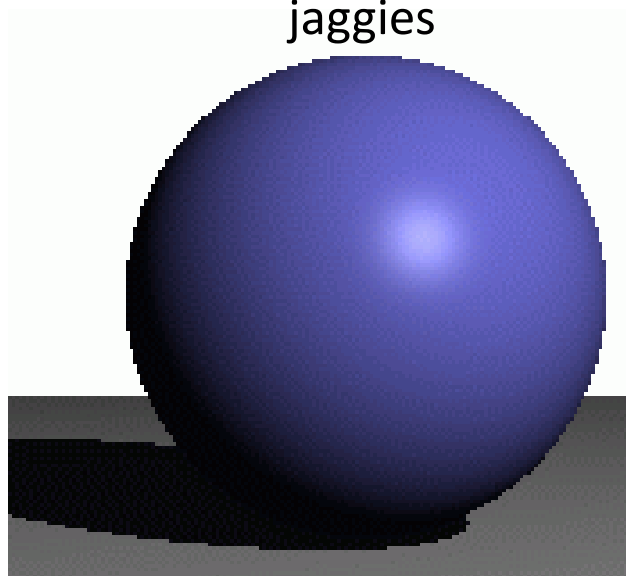
Ray Spawning



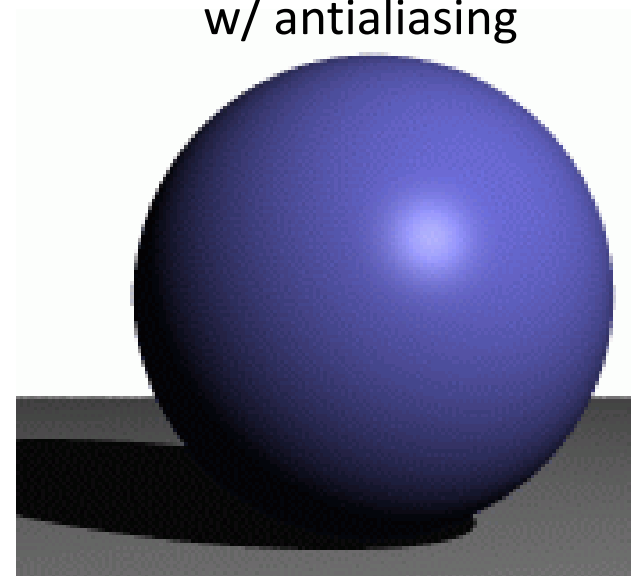
Ray Tracing supersampling

point light

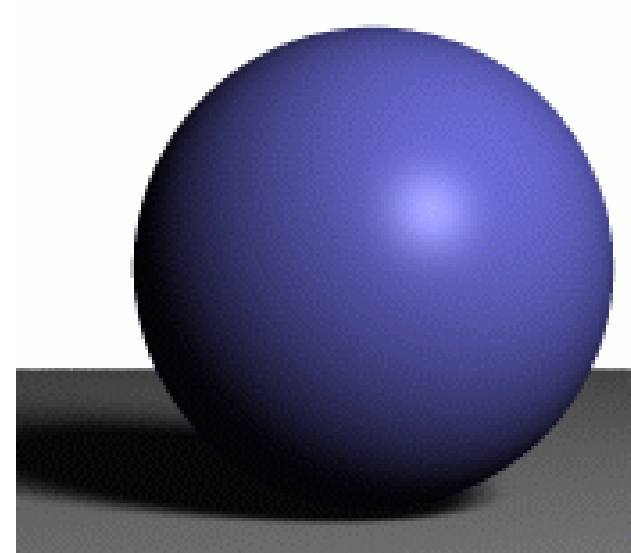
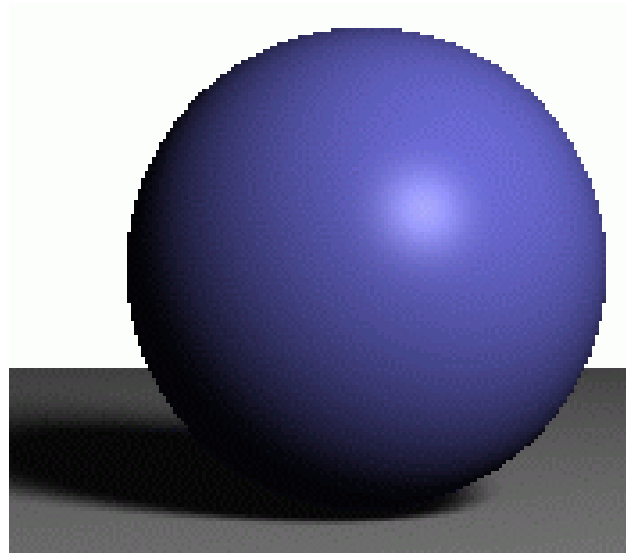
jaggies



w/ antialiasing



area light



Ray Tracing

- Unifies in one framework
 - Hidden surface removal
 - Shadow computation
 - Reflection of light
 - Refraction of light
 - Global **specular** interaction



Ray Tracing Deficiencies

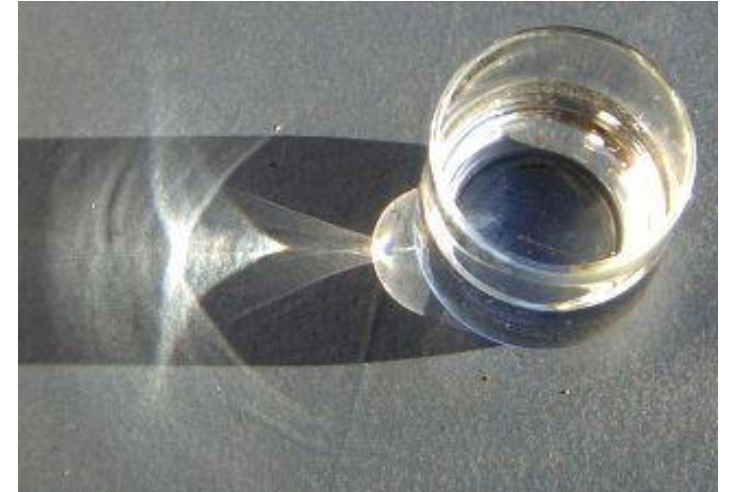
- Intersection computation time can be long (solution: **bounding volumes**).
- Recursive algorithm can lead to exponential complexity (solution: stochastic sampling).
- Ignores light transport mechanisms involving diffuse surfaces.



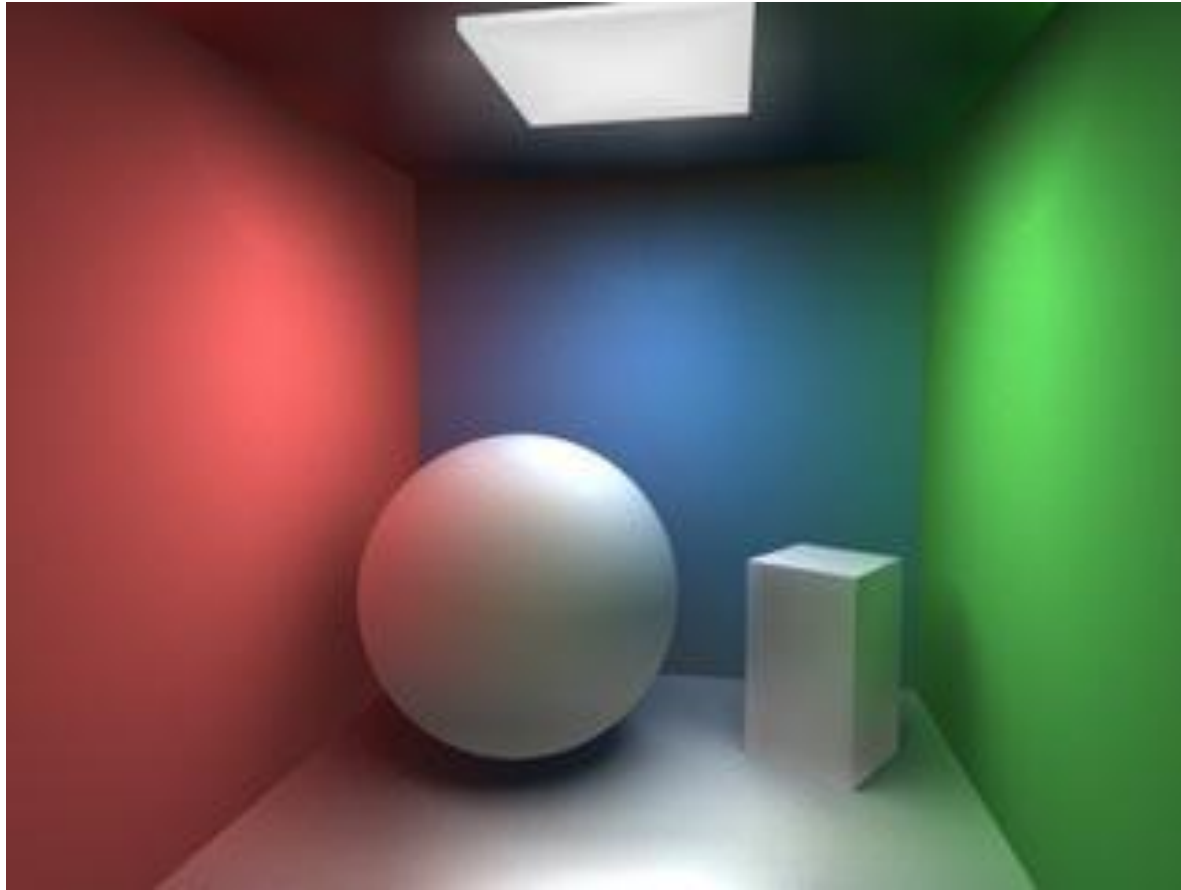


Ray Tracing Improvements: Caustics

- Transport $E - S - S - S - D - S - S - S - L$
- Trace from the light to the surfaces and then from the eye to the surfaces
- “shower” scene with light and then collect it
- “Where does light go?” vs “Where does light come from?”
- Good for caustics



Radiosity: E – D – D – D - L



The Rendering Equation



$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}' \cdot \vec{n}) d\vec{w}'$$

scarier version found here: https://en.wikipedia.org/wiki/Rendering_equation

research paper found here: <https://dl.acm.org/doi/10.1145/15886.15902>



The Rendering Equation

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}' \cdot \vec{n}) d\vec{w}'$$

outgoing light at position \mathbf{x} and
direction \mathbf{w}

emitted light at position \mathbf{x} and
direction \mathbf{w}

and

reflected light at position \mathbf{x} and
direction \mathbf{w}'



The Rendering Equation

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}' \cdot \vec{n}) d\vec{w}'$$

the reflected light at position x and direction \vec{w}

is the integral over all possible directions \vec{w}'

the incoming light from all directions

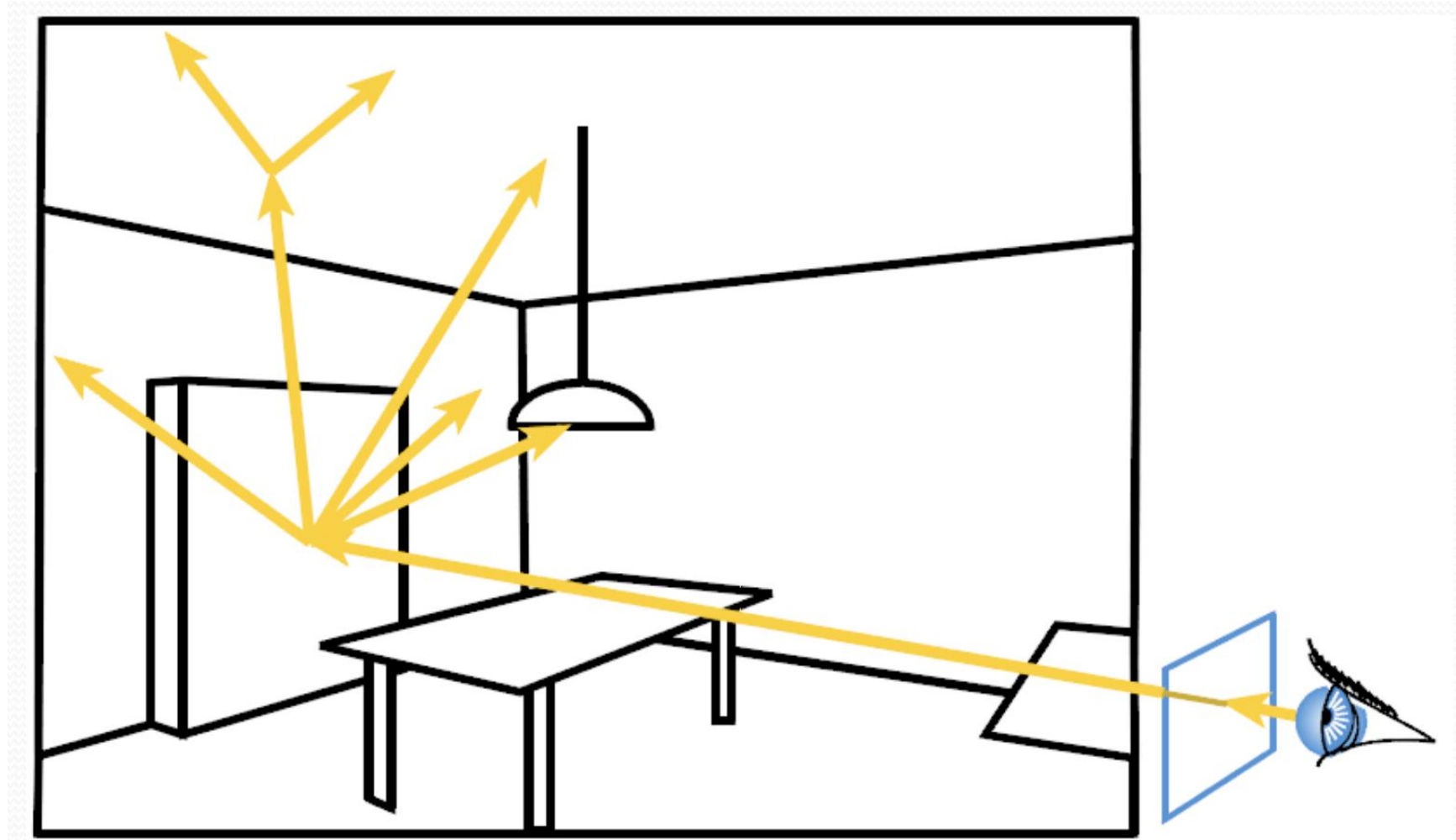
times

BRDF:
a function describing how light is reflected at an opaque surface



Monte Carlo Methods

Rely on random sampling to “solve” rendering equation



Area Light

Hard v soft shadows



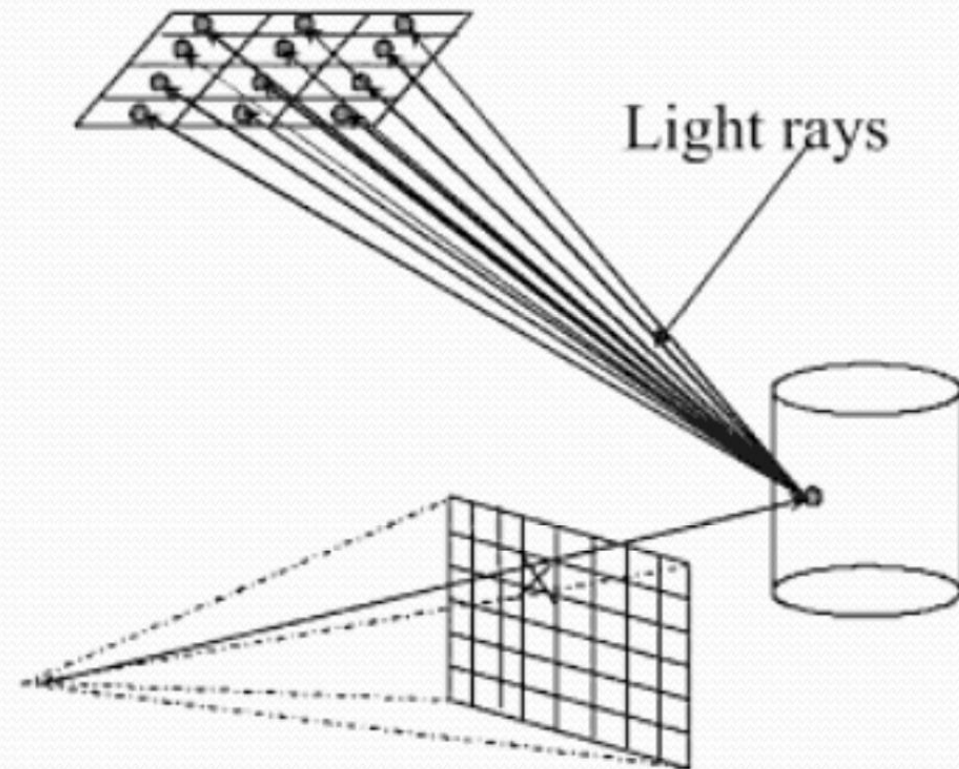
Hard shadow

More realistic soft shadows



Area Light

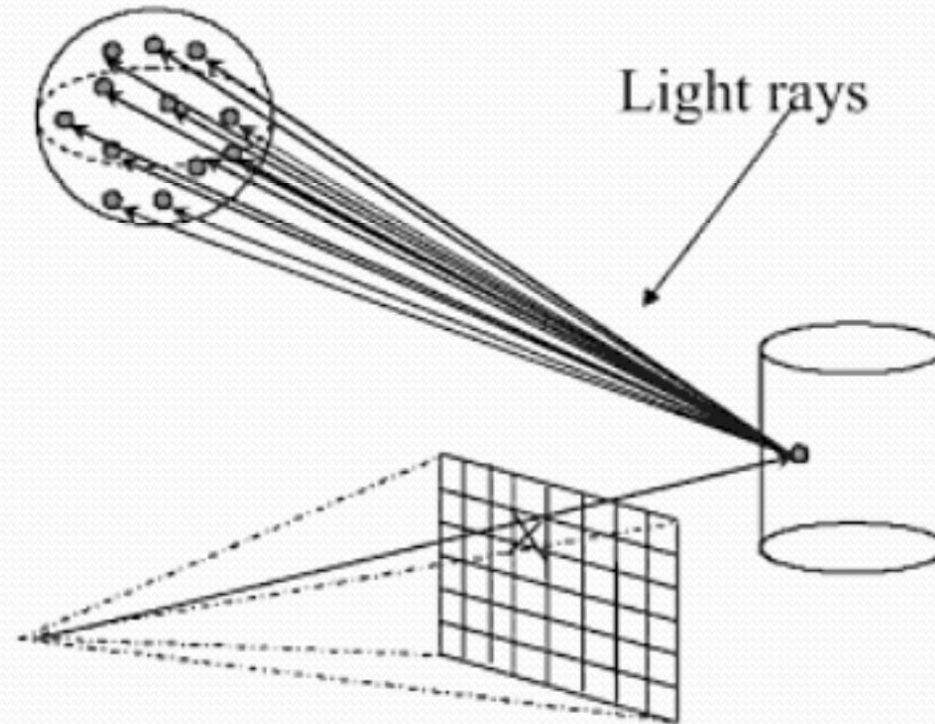
- Disadvantages of the simple uniform method:
 - Very time consuming
 - If the grid resolution is low, artifacts appear in the shadows.



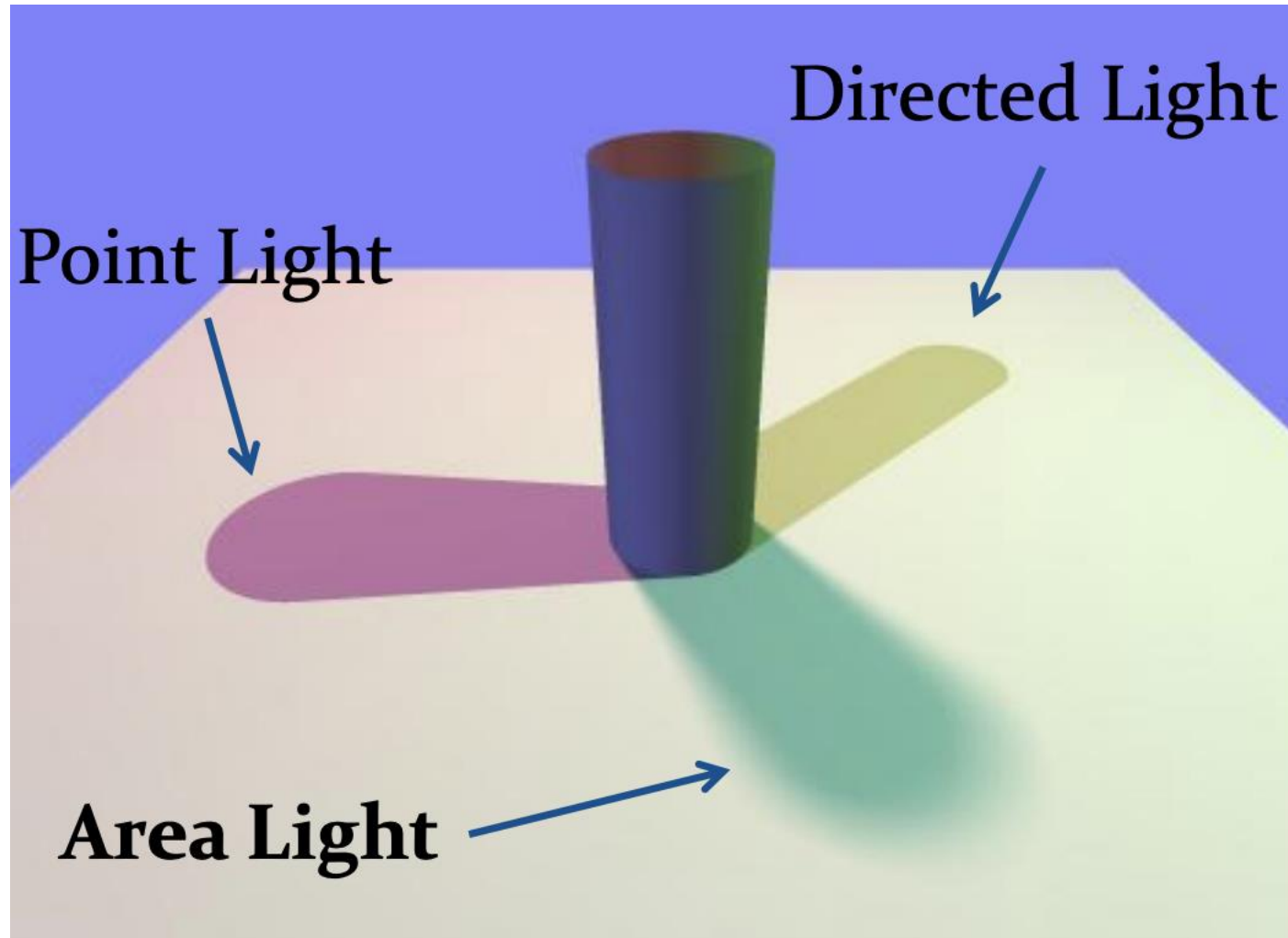
Area Light

Monte-Carlo Area light

- Light is modeled as a sphere
- Highest intensity in the middle. Gradually fade out.
- Shoot n rays to random points in the sphere
- Average their value.



Area Light



Faster Intersections for Ray Tracing

