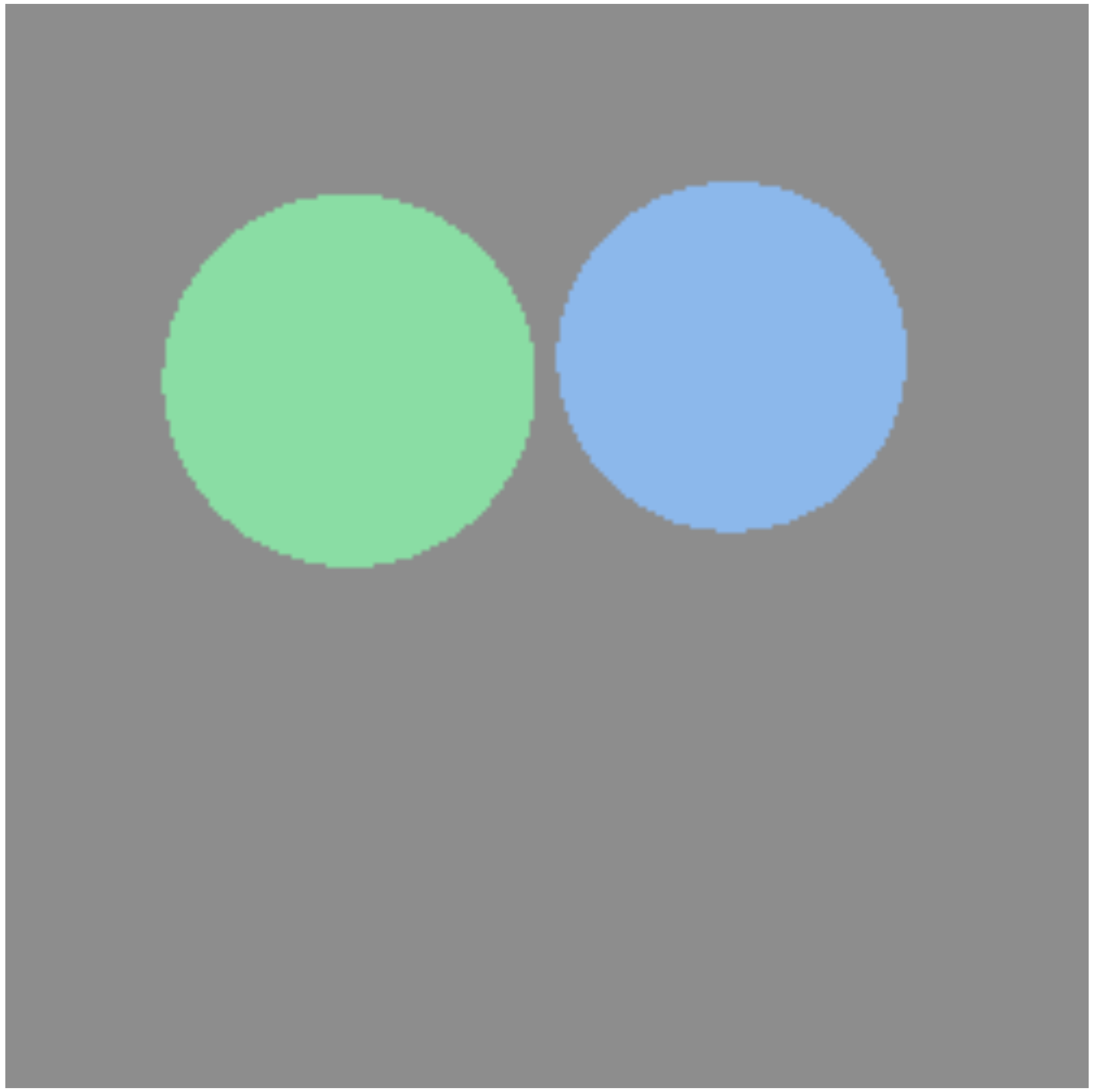
Basic shading in ray tracing

Steve Marschner
CS 4620
Cornell University

Image so far

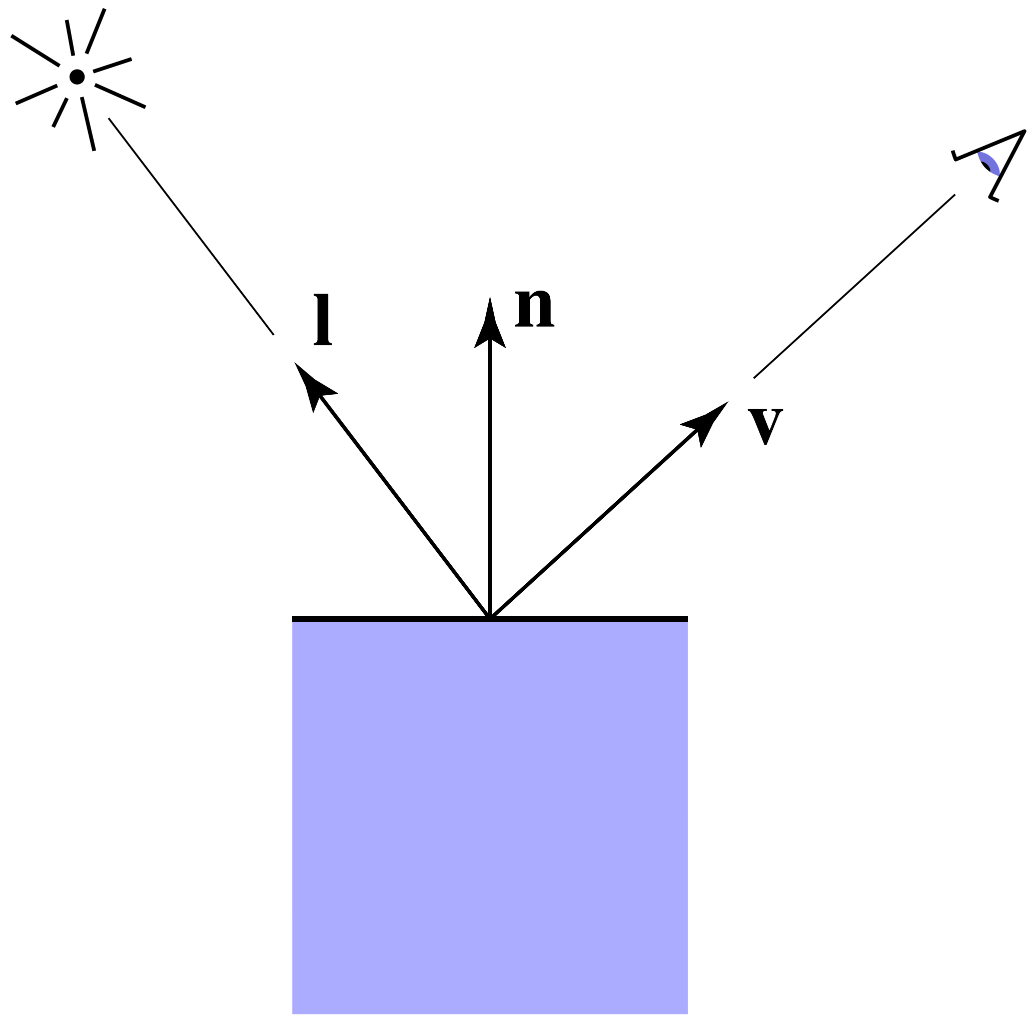
- **With eye ray generation and scene intersection**

```
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        c = scene.trace(ray, 0, +inf);
        image.set(ix, iy, c);
    }

...

Scene.trace(ray, tMin, tMax) {
    surface, t = surfs.intersect(ray, tMin, tMax);
    if (surface != null) return surface.color();
    else return black;
}
```



Shading

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



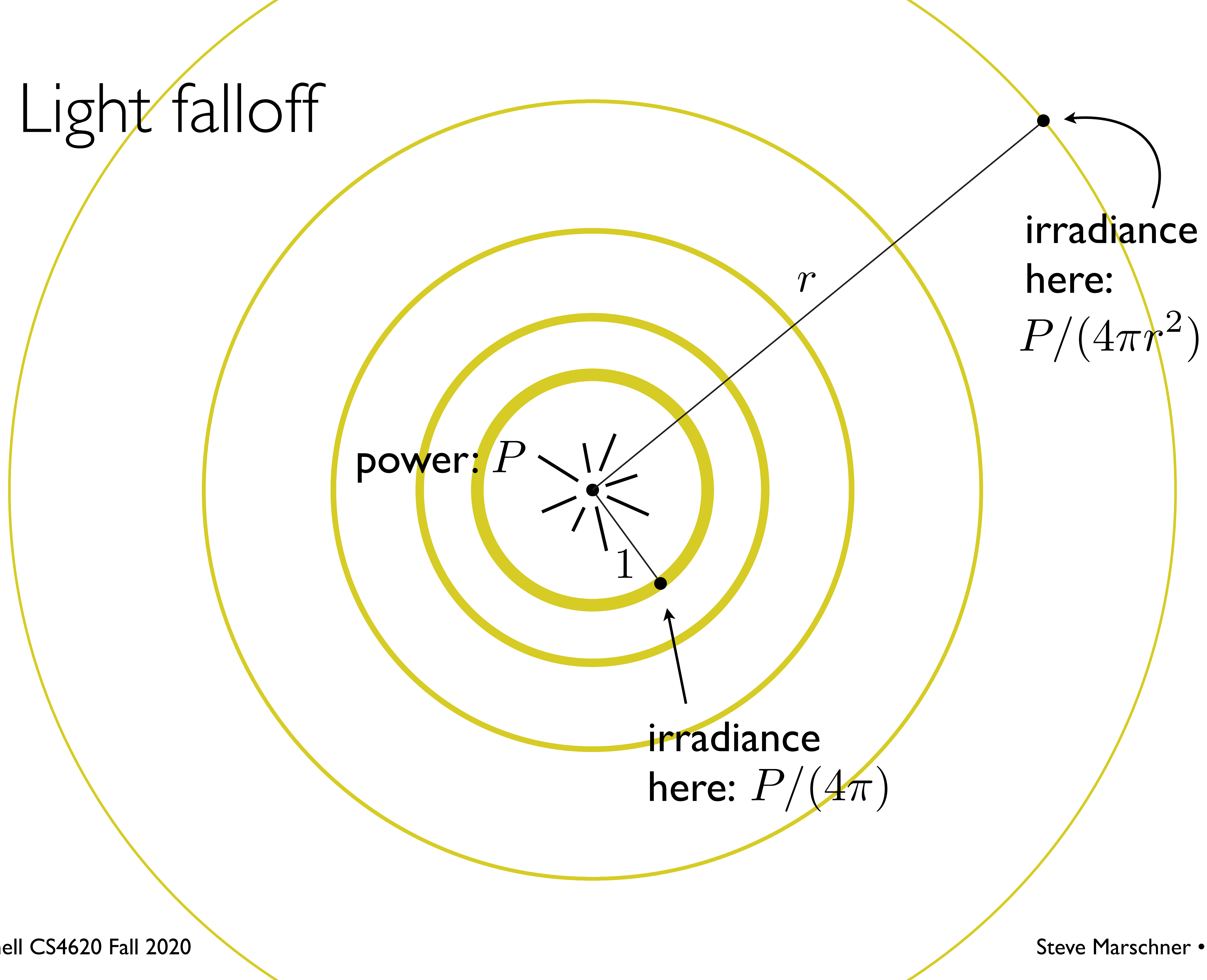
Shading philosophy

- **Goals of shading depend on purpose of image**
  - visualization, CAD: maximize visual clarity
  - visual effects, advertising: maximize resemblance to reality
  - animation, games: somewhere in between
- **Basic starting point: physics of light reflection**
  - a set of useful approximations to real surfaces
  - can remove things for simplicity/clarity
  - can add things for increased accuracy/realism

Light

- **Think of light as a flow of particles through space**
  - disregarding wave nature: polarization, interference, diffraction
  - for now disregarding color: only how much light
- **Sources of light**
  - point sources (a flashlight) ← we will stick to this for now.
  - directional sources (the sun)
  - area sources (a fluorescent tube)
  - environment sources (the sky)

Light falloff



Irradiance from isotropic point source

- **A sphere surrounding the source receives all the power**
- **A small, flat surface of area *A* facing the source receives a fraction (area of surface) / (area of sphere) of that power:**

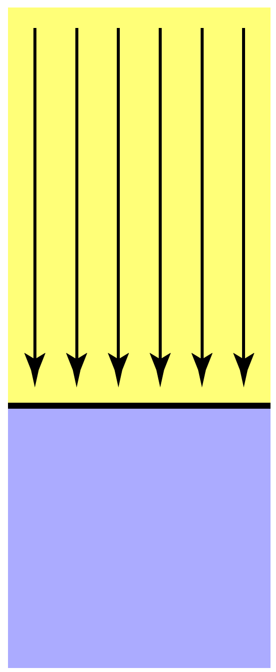
$$P_A = P \frac{A}{4\pi r^2}$$

- **Irradiance is power per unit area:**

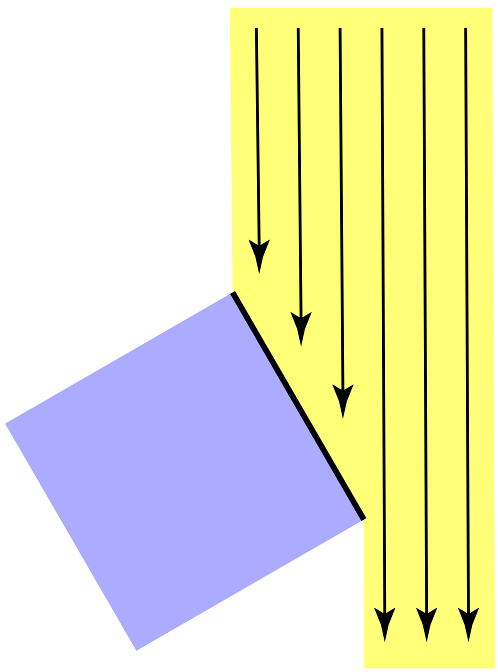
$$E = P_A / A = \frac{P}{4\pi r^2} = \frac{P}{4\pi} \frac{1}{r^2}$$

intensity geometry factor

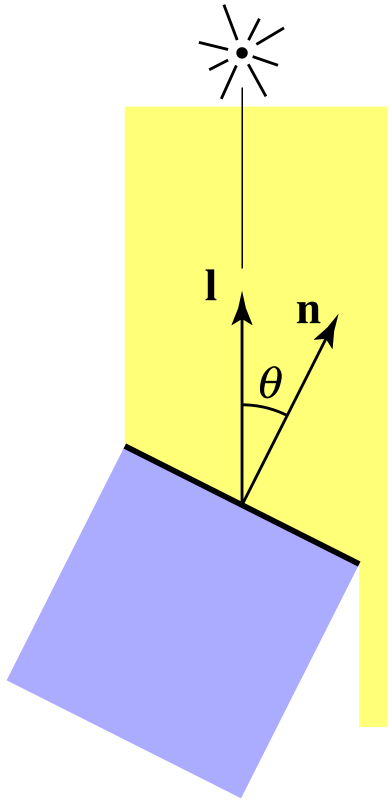
Lambert's cosine law



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

Irradiance from isotropic point source

- **A surface of area *A* facing at an angle to the source receives a factor of $\cos\theta$ less light:**

$$P_A = P\frac{A\cos\theta}{4\pi r^2}$$

- **Irradiance is power per unit area:**

$$E = P_A/A = \underbrace{\frac{P}{4\pi}}_{\text{intensity}}\ \underbrace{\frac{\cos\theta}{r^2}}_{\text{geometry factor}}$$

Diffuse reflection

- **Simplest reflection model**
- **Reflected light is independent of view direction**
- **Reflected light is proportional to irradiance**
  - constant of proportionality is the diffuse reflection coefficient

$$L_d = k_d E$$

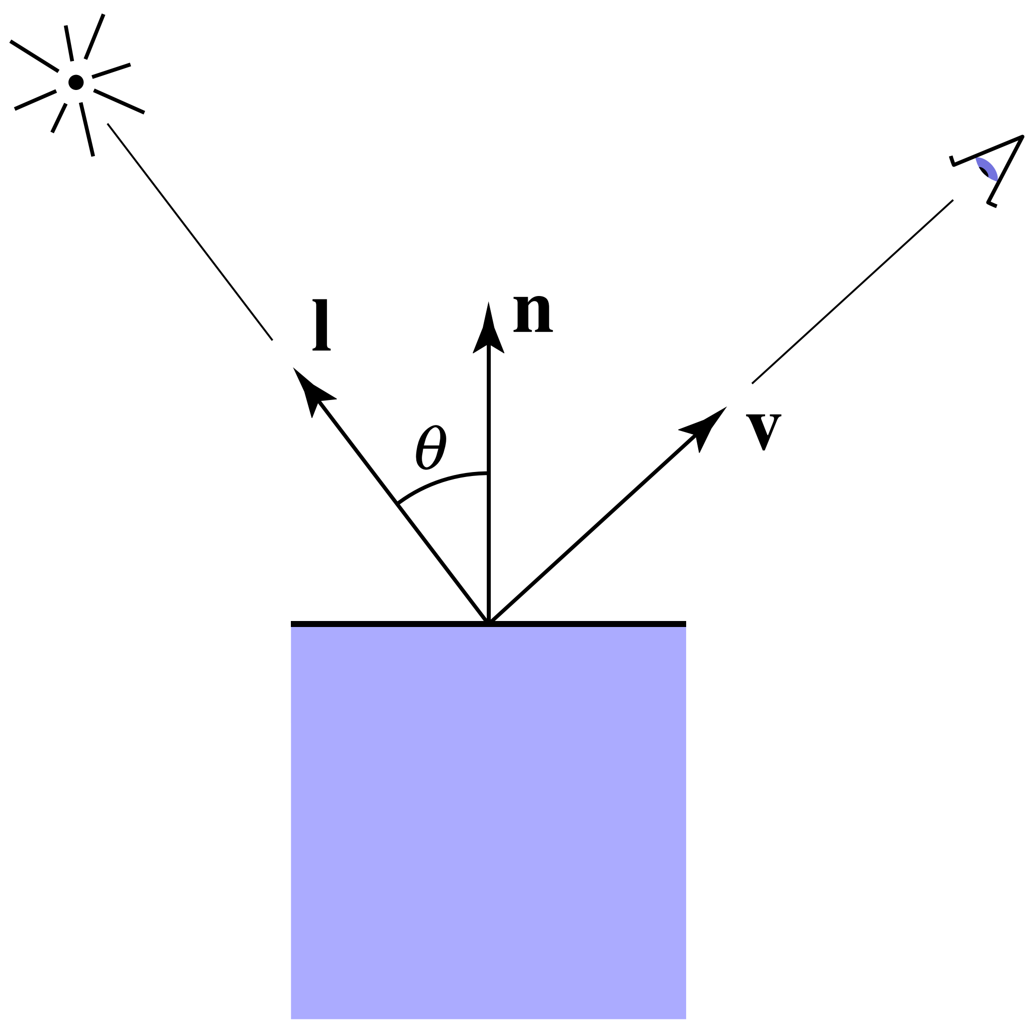
- **More useful to think in terms of reflectance**
  - reflectance is the fraction reflected (between 0 and 1)

$$L_d = \frac{R_d}{\pi} E$$

  - will have to explain the factor of π some other time

Lambertian shading

- **Shading independent of view direction**



irradiance from source

diffuse reflectance

$$L_d = \frac{R}{\pi} \overbrace{\frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I}^{\text{irradiance from source}}$$

diffusely reflected radiance

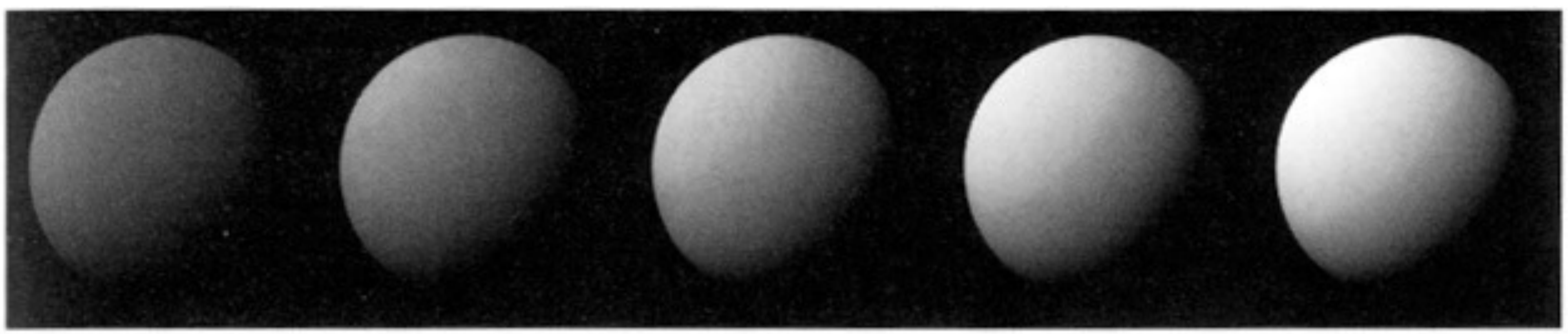
diffuse coefficient

distance to source

intensity of source

Lambertian shading

- **Produces matte appearance**



$k_d \longrightarrow$

[Foley et al.]

Diffuse shading

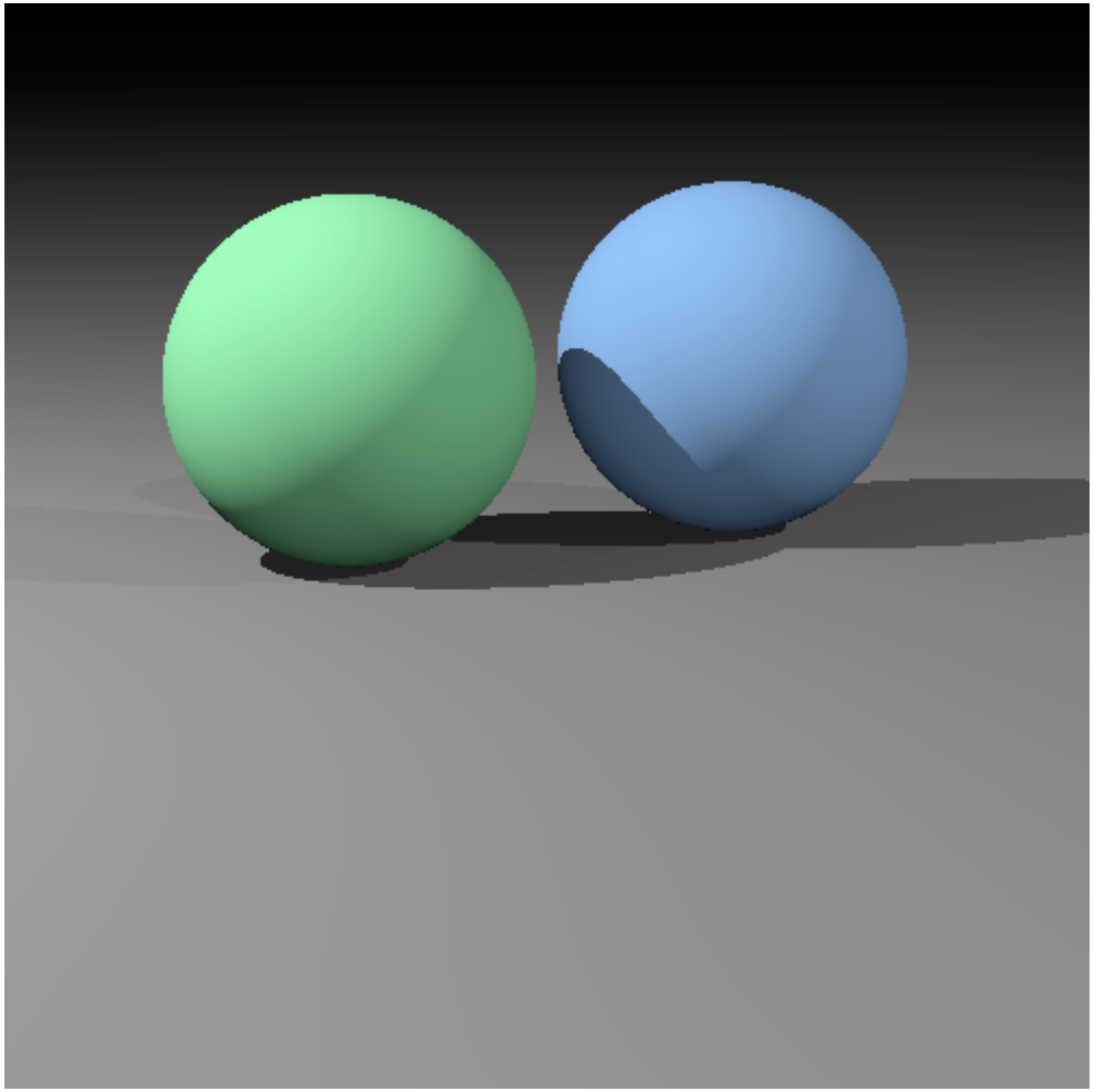
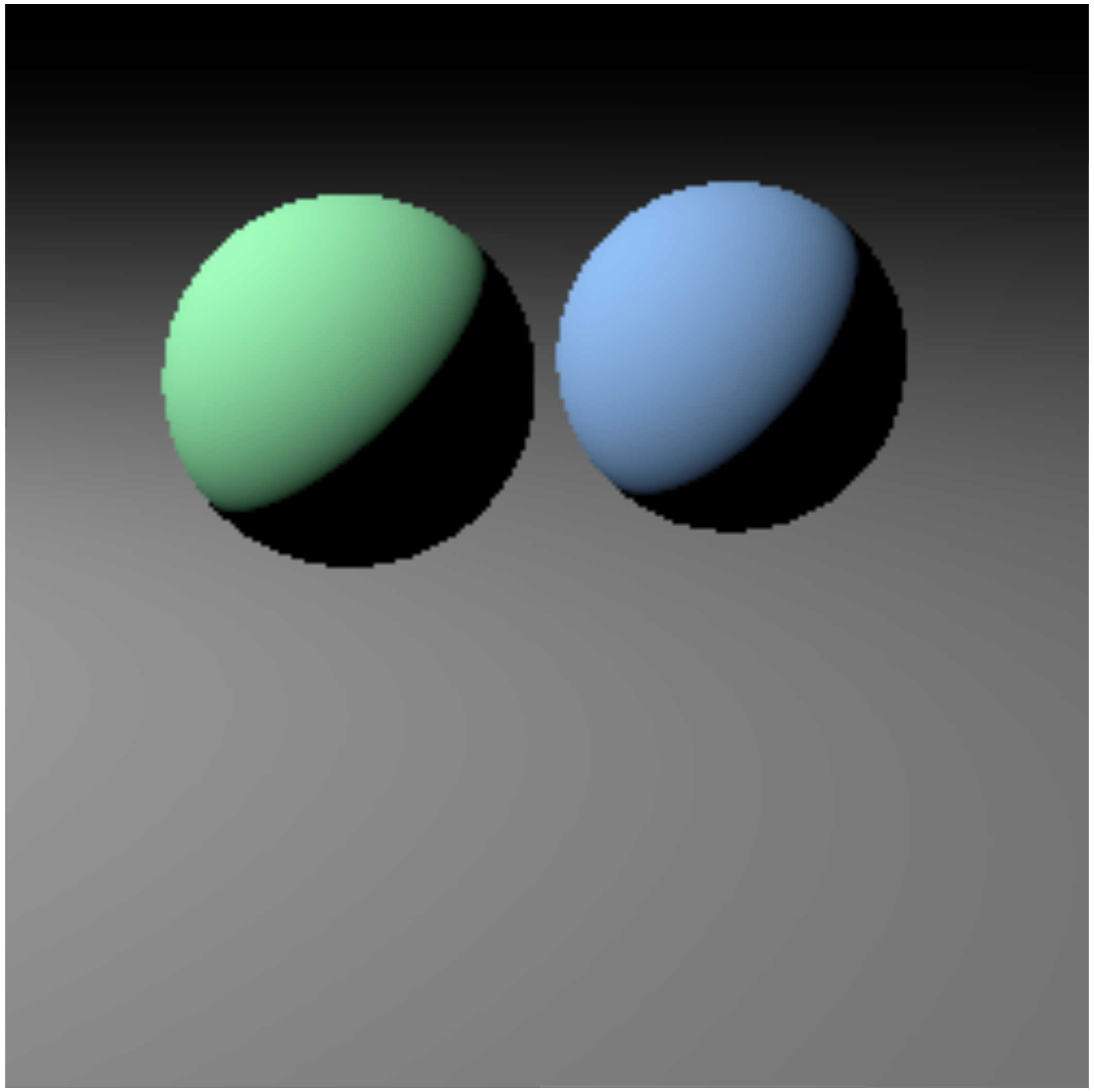


Image so far

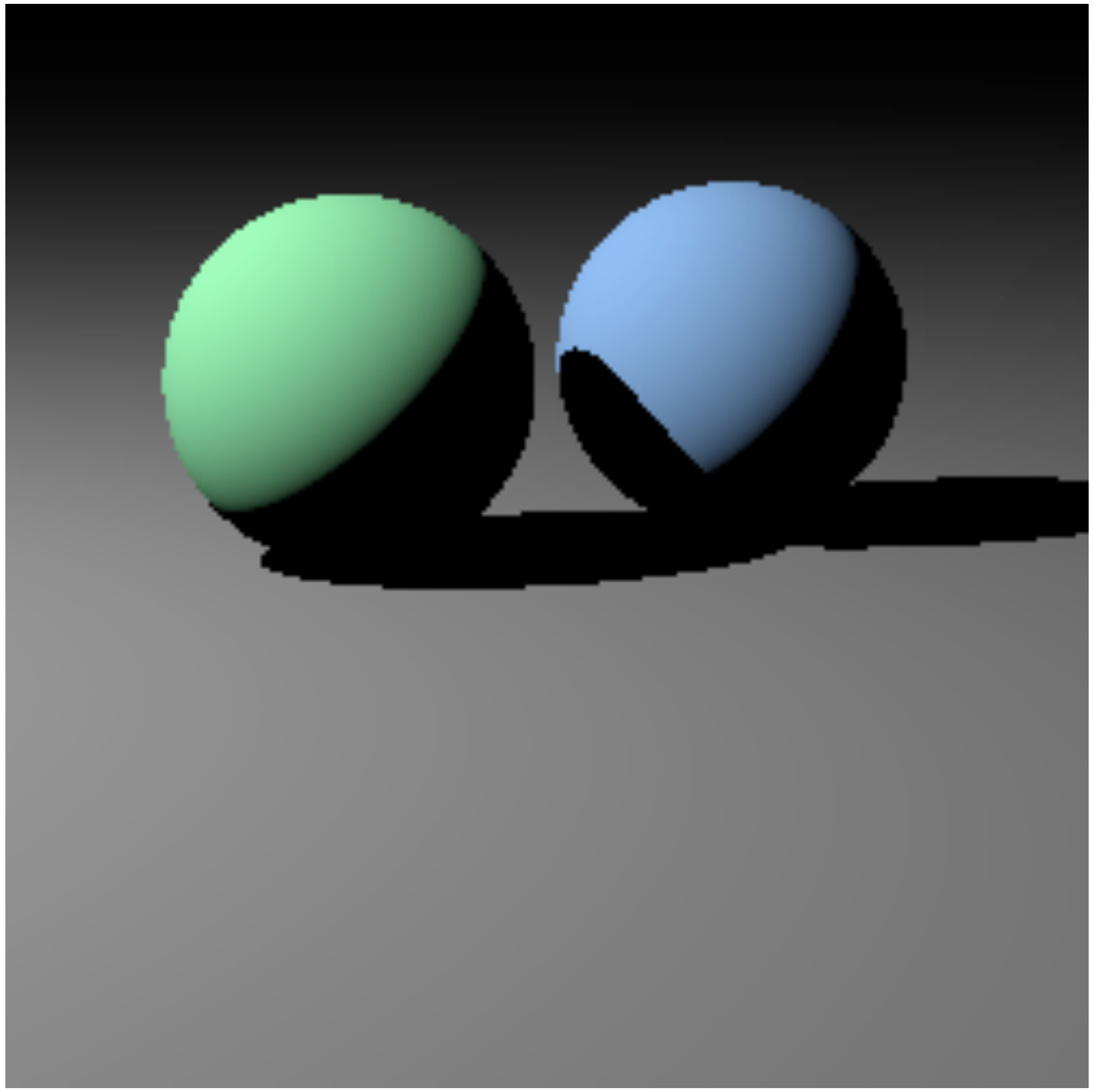
```
Scene.trace(Ray ray, tMin, tMax) {
    surface, t = hit(ray, tMin, tMax);
    if surface is not null {
        point = ray.evaluate(t);
        normal = surface.getNormal(point);
        return surface.shade(ray, point,
            normal, light);
    }
    else return backgroundColor;
}

...

Surface.shade(ray, point, normal, light) {
    v = −normalize(ray.direction);
    l = normalize(light.pos − point);
    // compute shading
}
```



Shadows

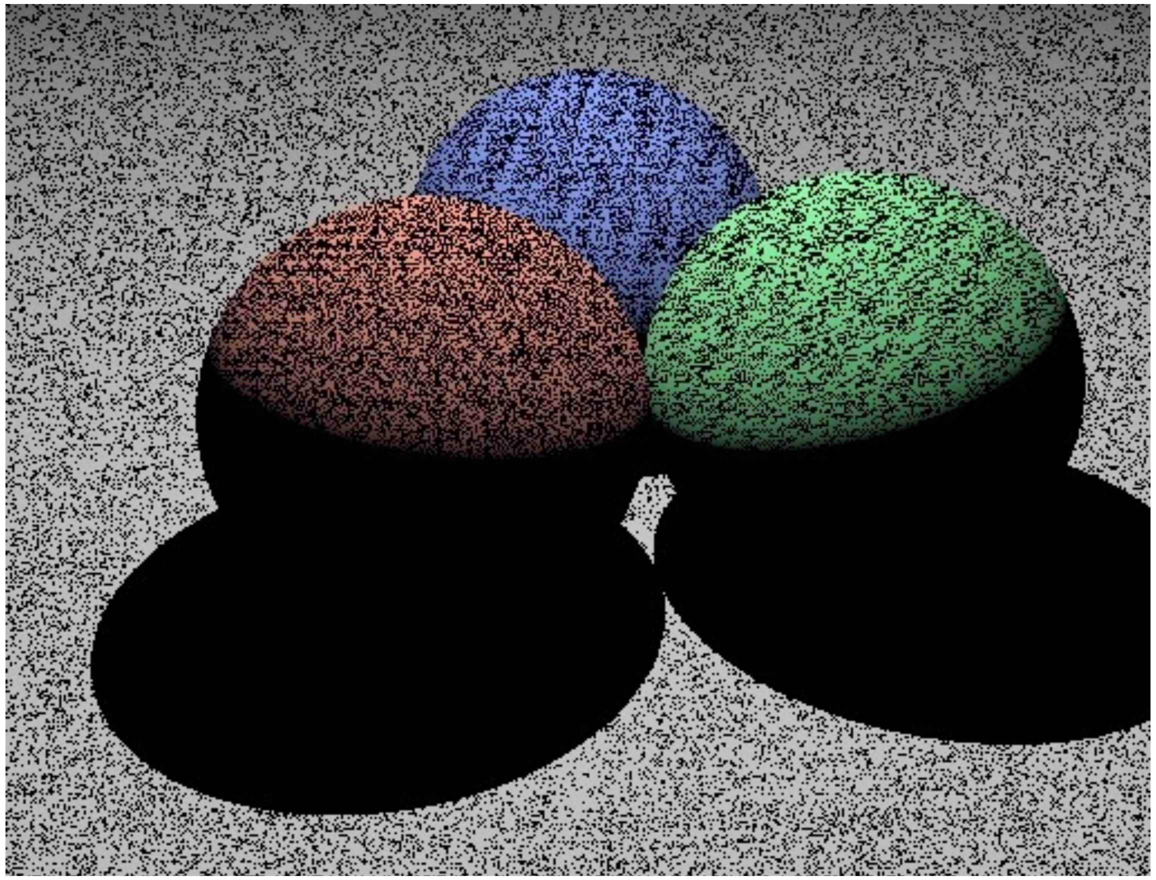
- **Surface is only illuminated if nothing blocks the light**
  – i.e. if the surface can "see" the light
- **With ray tracing it's easy to check**
  – just intersect a ray with the scene!

```
Surface.shade(ray, point, normal, light) {
    shadRay = (point, light.pos – point);
    if (shadRay not blocked) {
        v = –normalize(ray.direction);
        l = normalize(light.pos – point);
        // compute shading
    }
    return black;
}
```



Shadow rounding errors

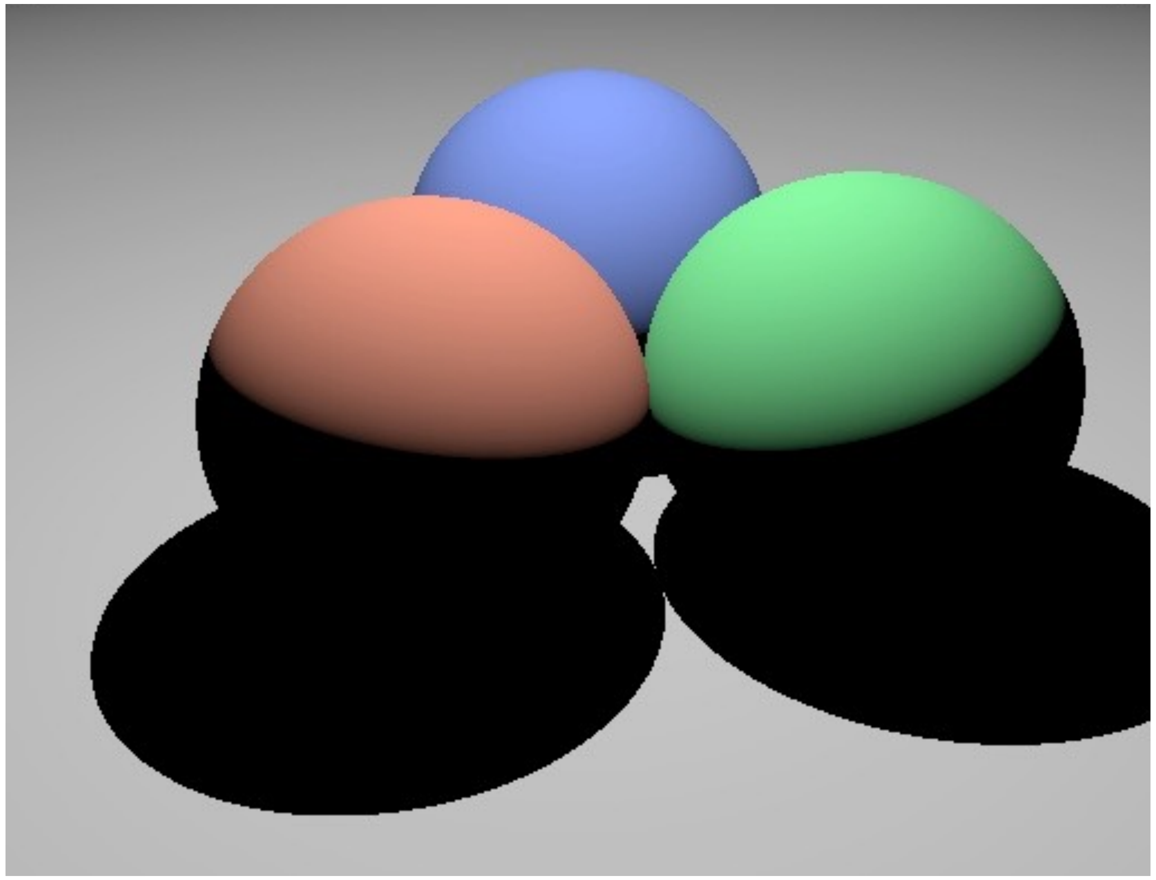
- **Don't fall victim to one of the classic blunders:**



- **What's going on?**
  – hint: at what ***t*** does the shadow ray intersect the surface you're shading?

Shadow rounding errors

- **Solution: shadow rays start a tiny distance from the surface**

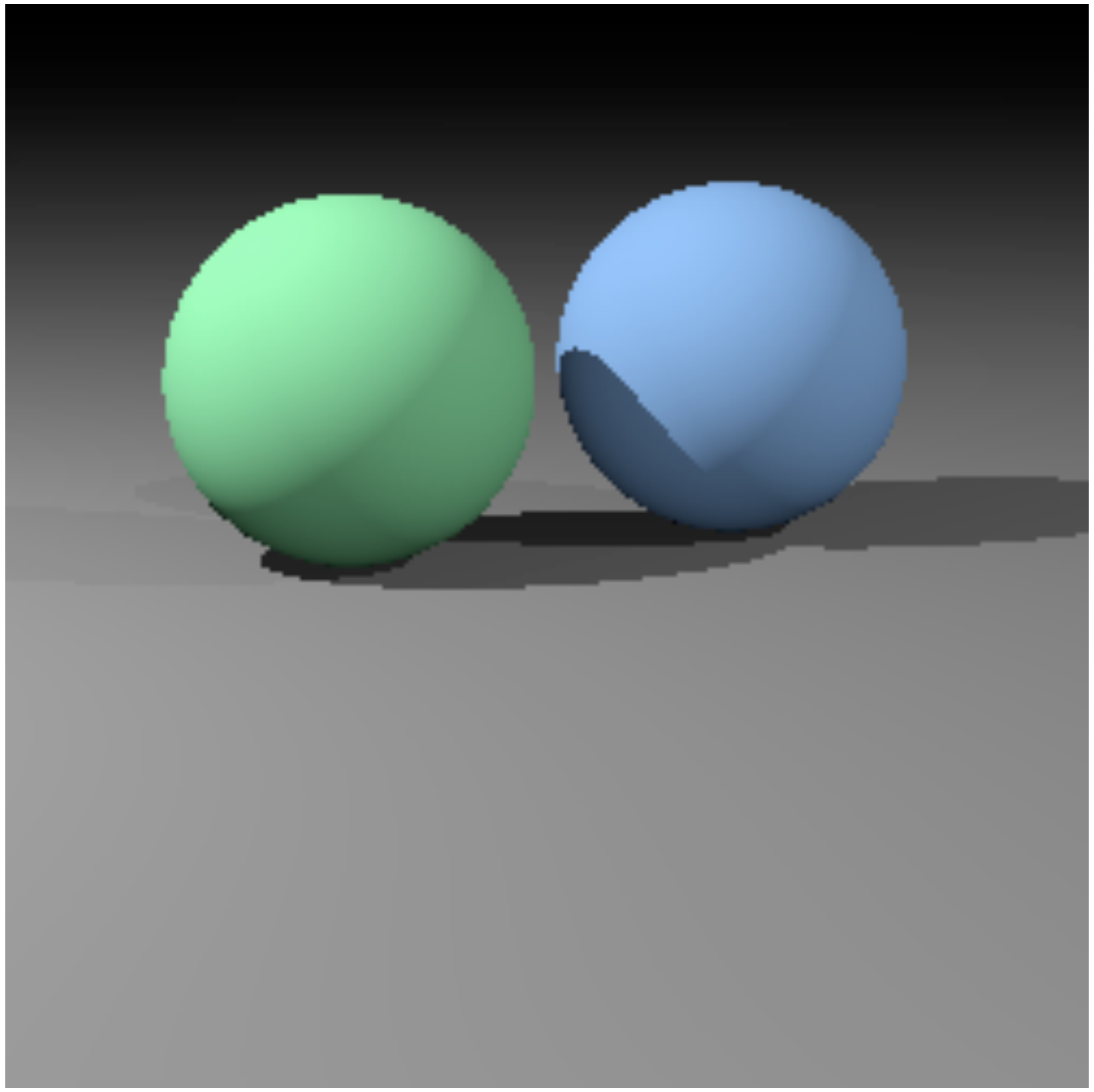


- **Do this by moving the start point, changing the starting *t***

Multiple lights

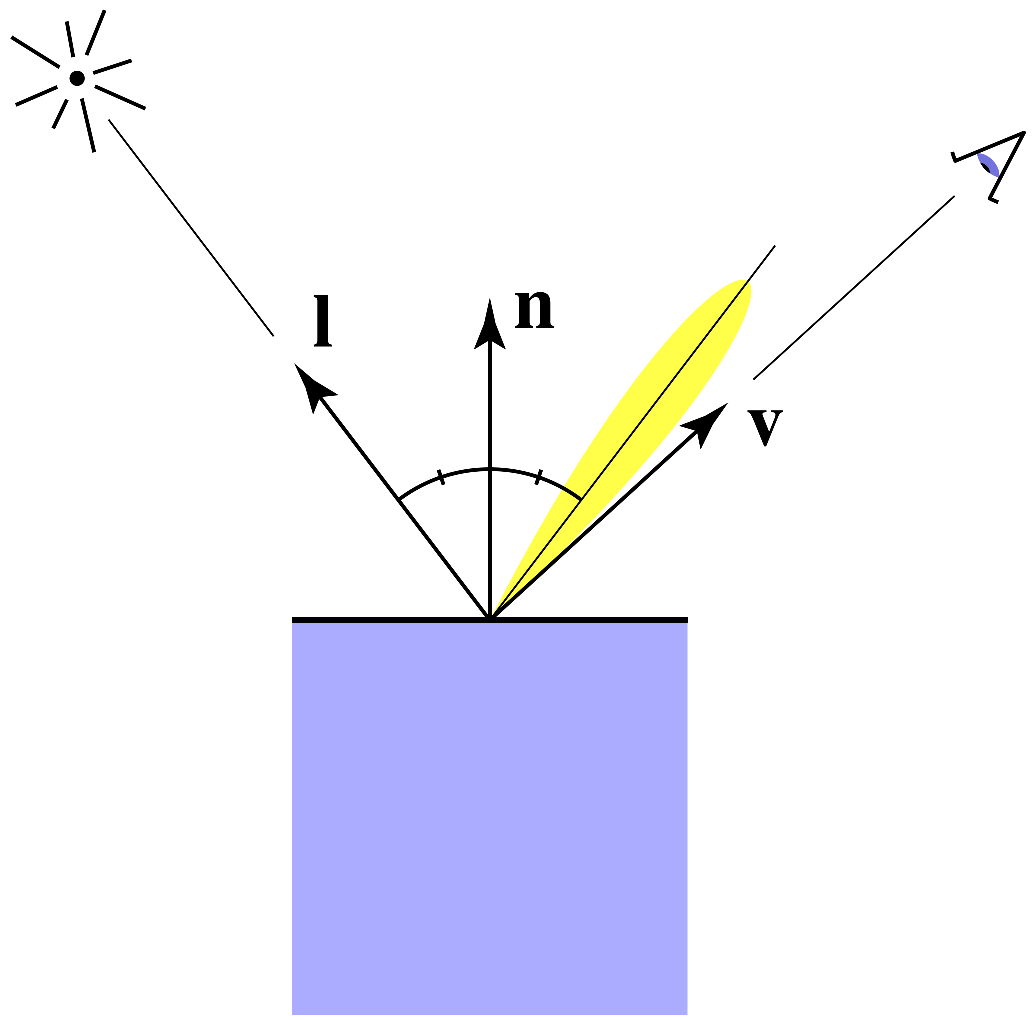
- **Important to fill in black shadows**
- **Just loop over lights, add contributions**

```
shade(ray, point, normal, lights) {
    result = ambient;
    for light in lights {
        if (shadow ray not blocked) {
            result += shading contribution;
        }
    }
    return result;
}
```



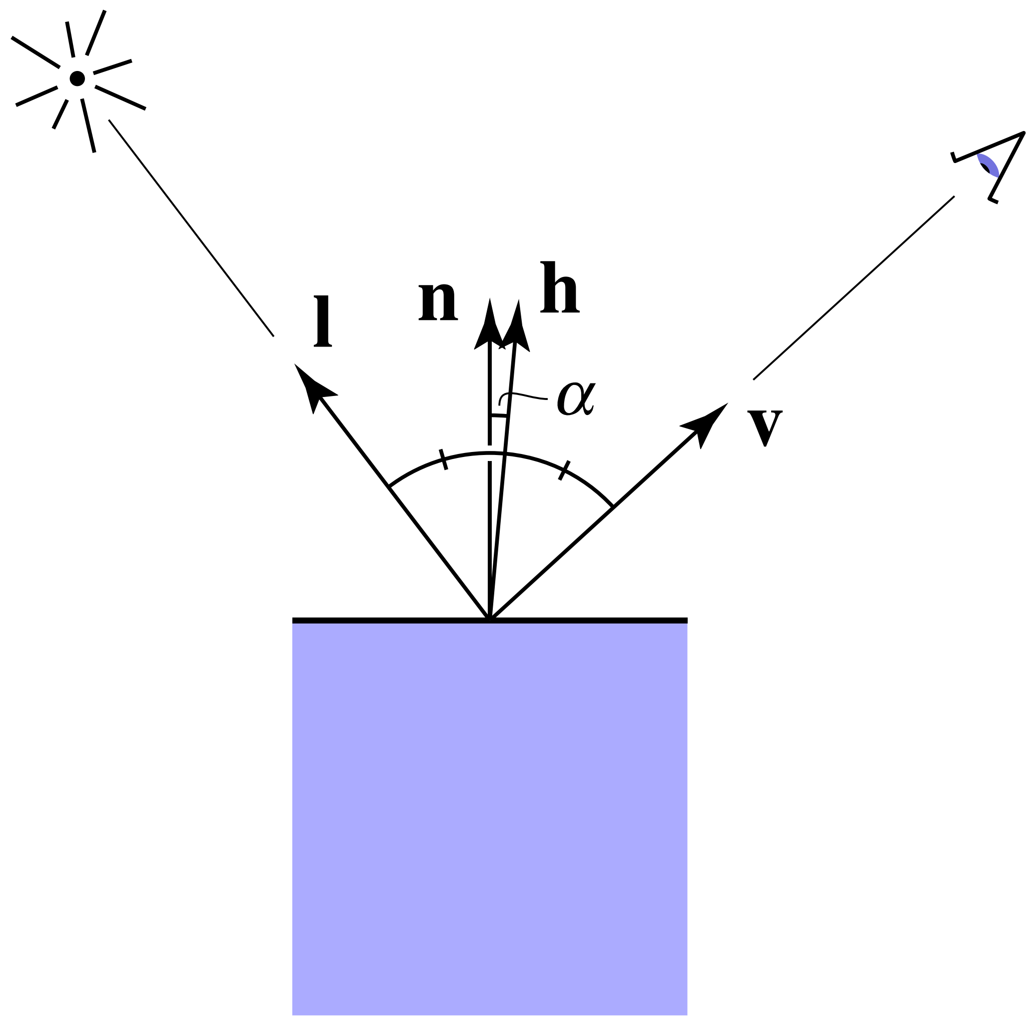
Specular shading

- **Intensity depends on view direction**
  - bright near mirror configuration



Specular shading (Blinn-Phong)

- **Close to mirror ⇔ half vector near normal**
  - Measure "near" by dot product of unit vectors



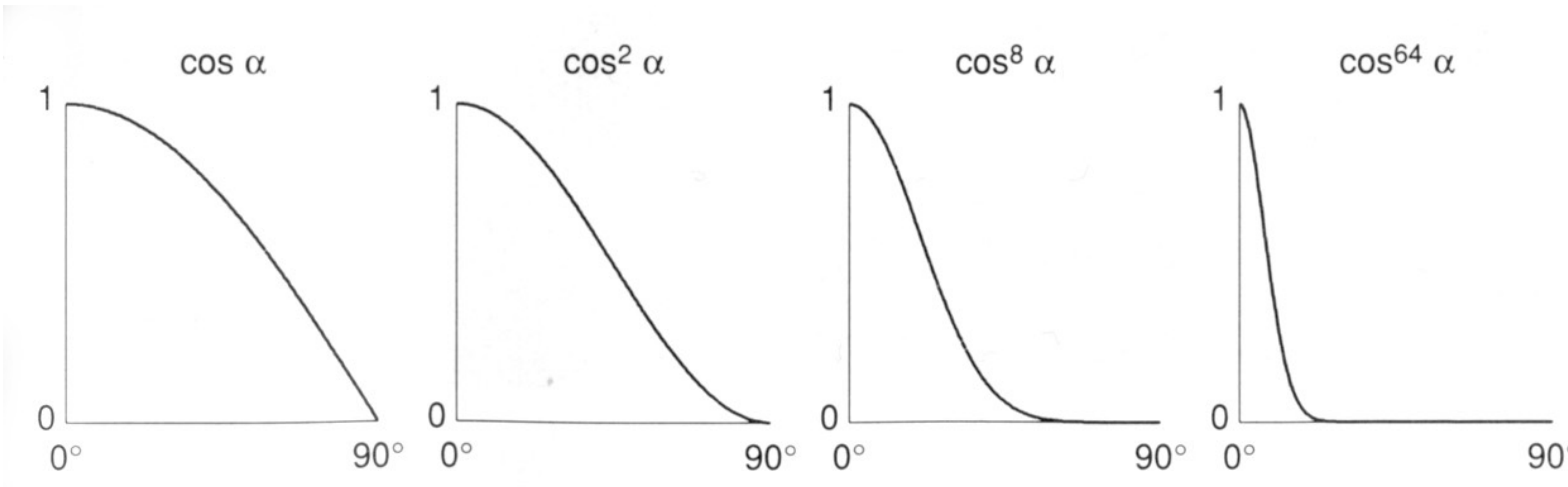
$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$
$$= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

let's work with the expression:

$$(\cos \alpha)^p$$
$$= (\mathbf{n} \cdot \mathbf{h})^p$$

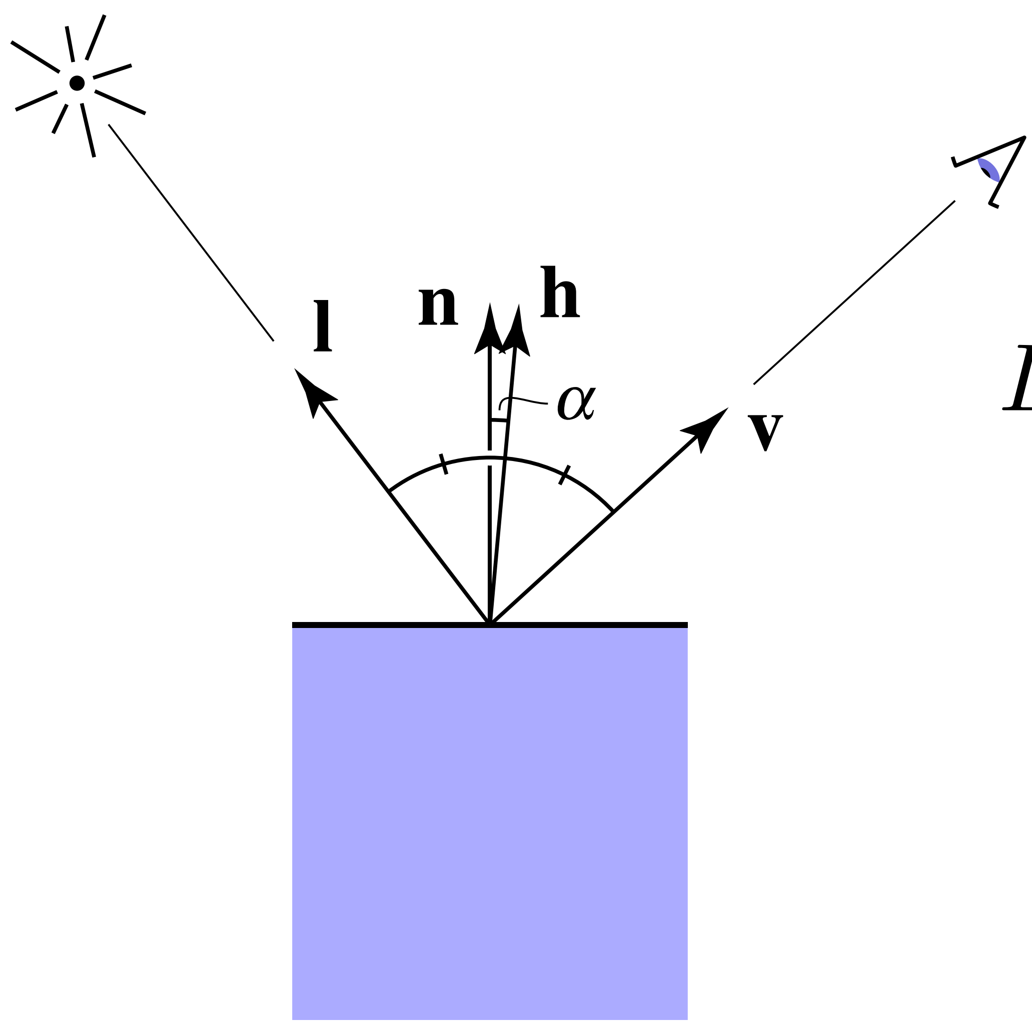
Phong model—plots

- **Increasing *p* narrows the peak**
  – corresponds to increasing "shininess"



[Foley et al.]

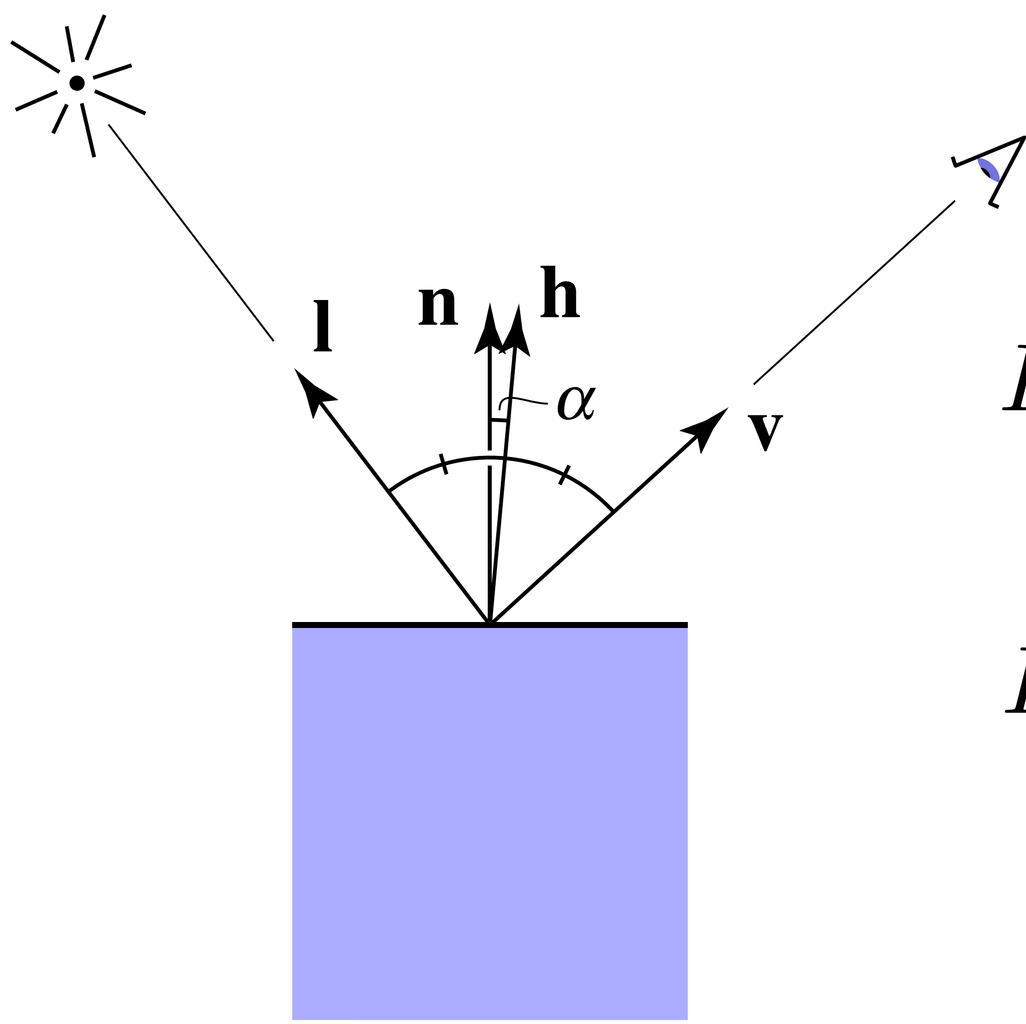
Specular shading (Blinn-Phong)



$$L_d = \frac{R}{\pi} \frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I$$

generalize this

Specular shading (Blinn-Phong)

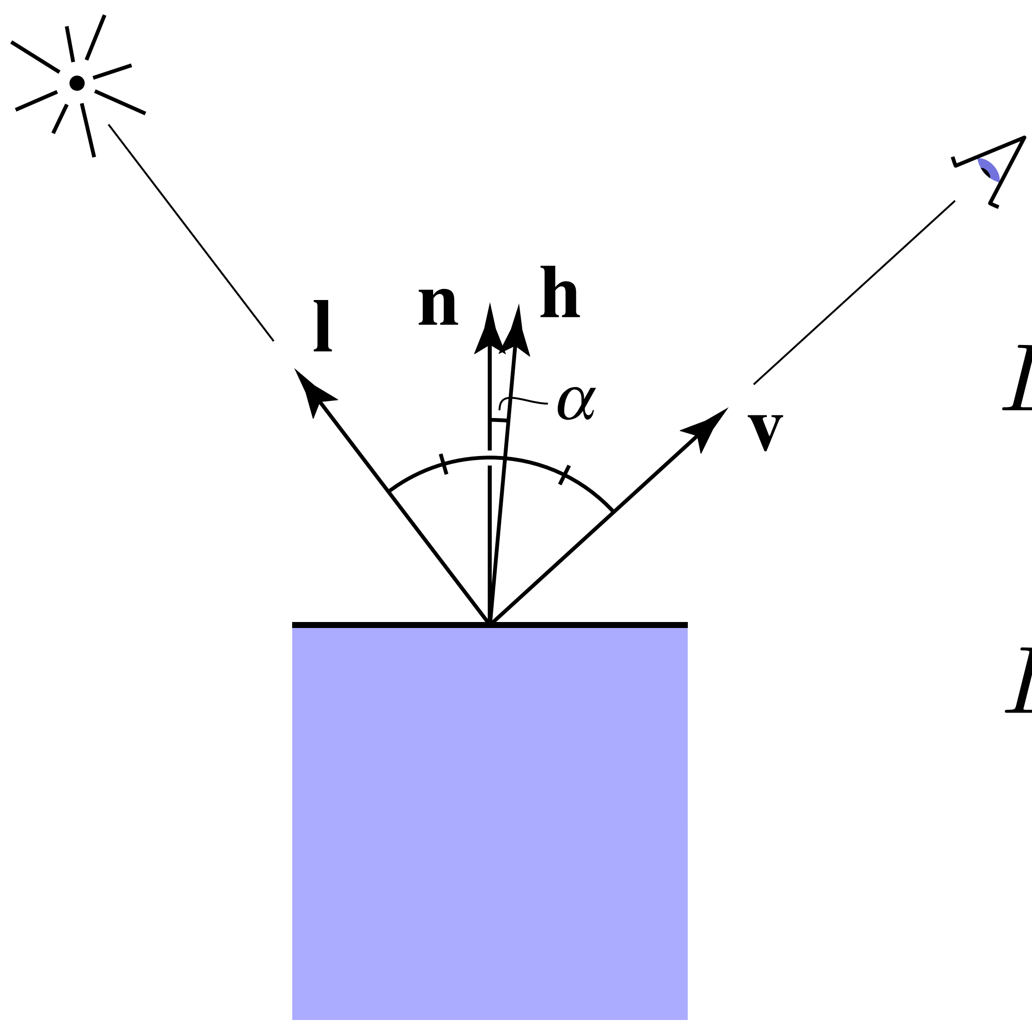


$$L_d = \frac{R}{\pi} \, \frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} \, I$$

$$L_r = \left(\frac{R}{\pi} + k_s (\mathbf{n} \cdot \mathbf{h})^p \right) \, \frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} \, I$$

Specular shading (Blinn-Phong)

note: this model is officially called "modified Blinn-Phong."



$$L_d = \frac{R}{\pi} \frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I$$

$$L_r = \left(\frac{R}{\pi} + k_s (\mathbf{n} \cdot \mathbf{h})^p \right) \frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I$$

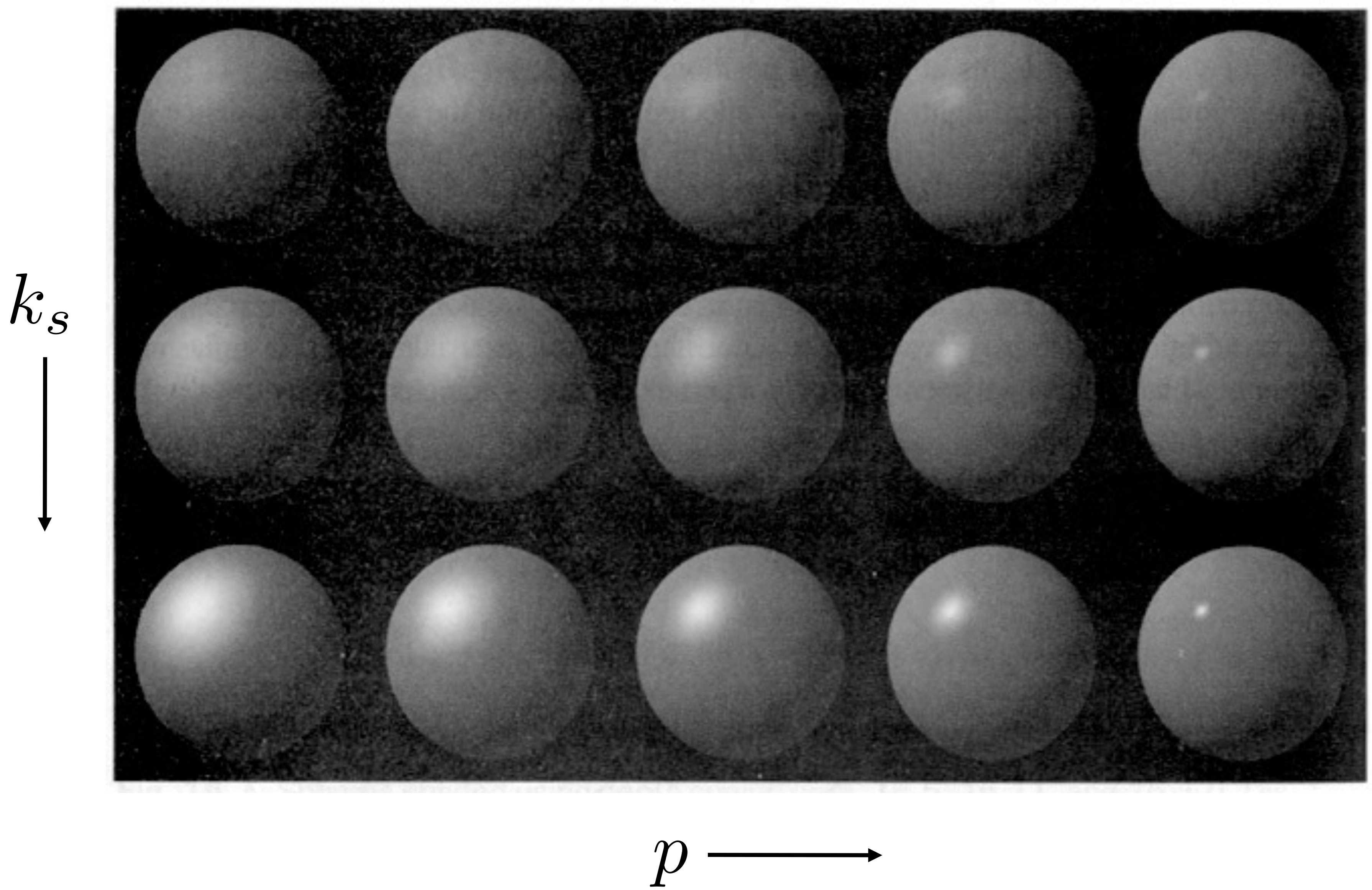
diffuse coefficient

specular coefficient

specular term

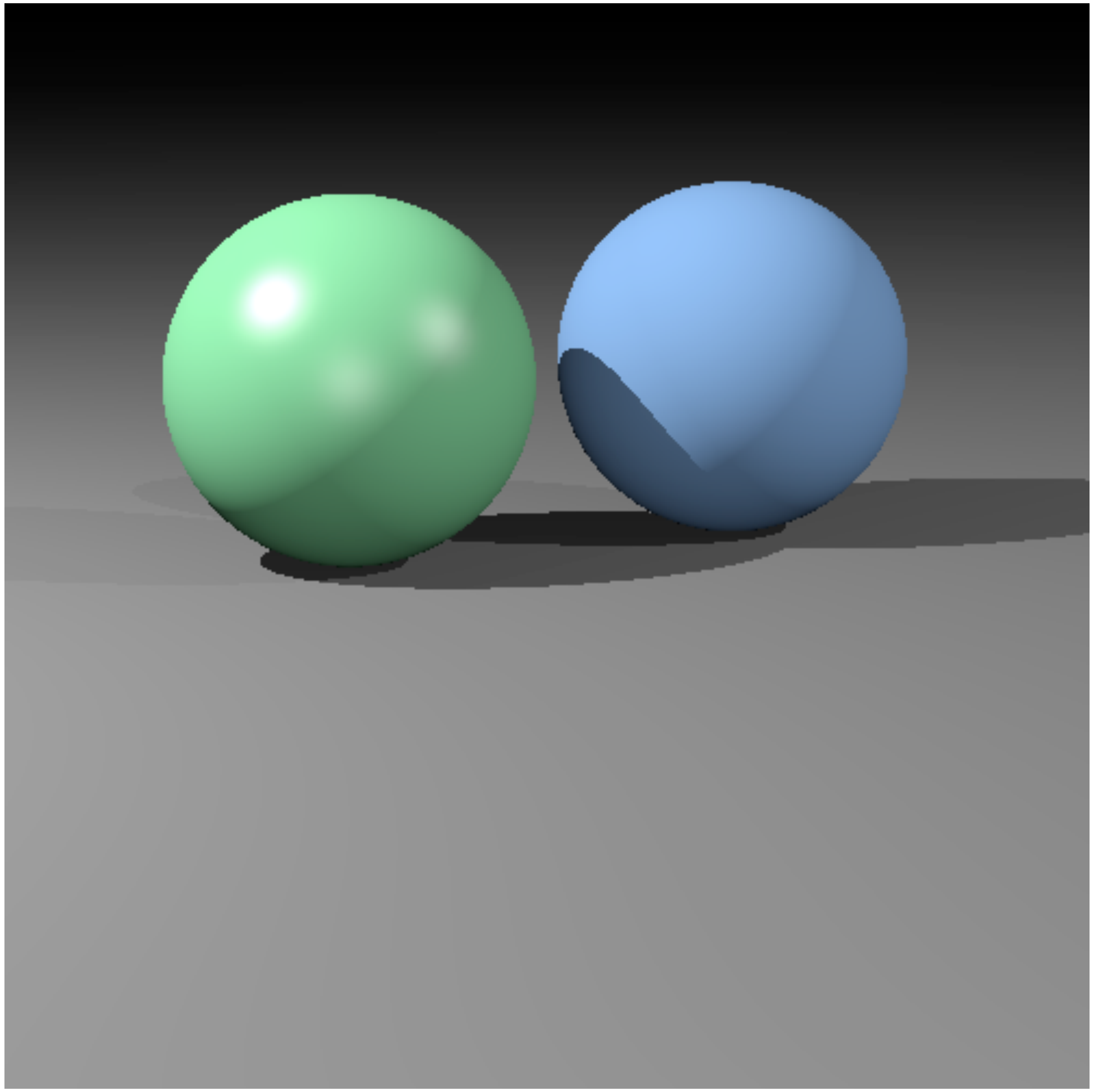
Specular shading

- **Blinn-Phong**



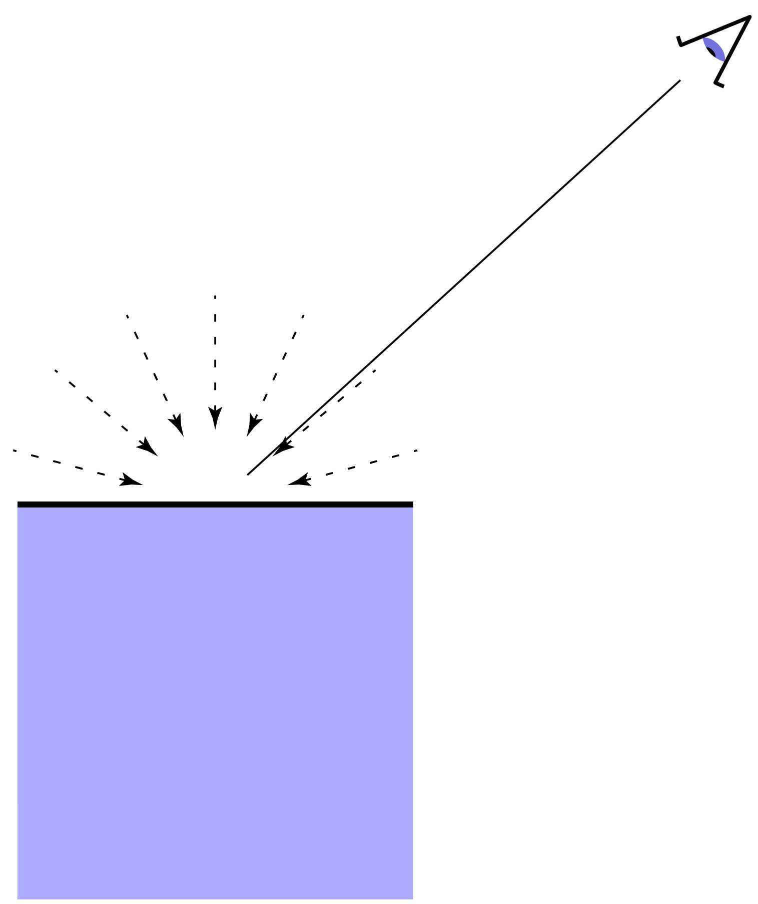
[Foley et al.]

Diffuse + Phong shading



Ambient shading

- **Shading that does not depend on anything**
  - add constant color to account for disregarded illumination and fill in black shadows



$$L_a = k_a \, I_a$$

reflected ambient light

ambient coefficient

Mirror reflection

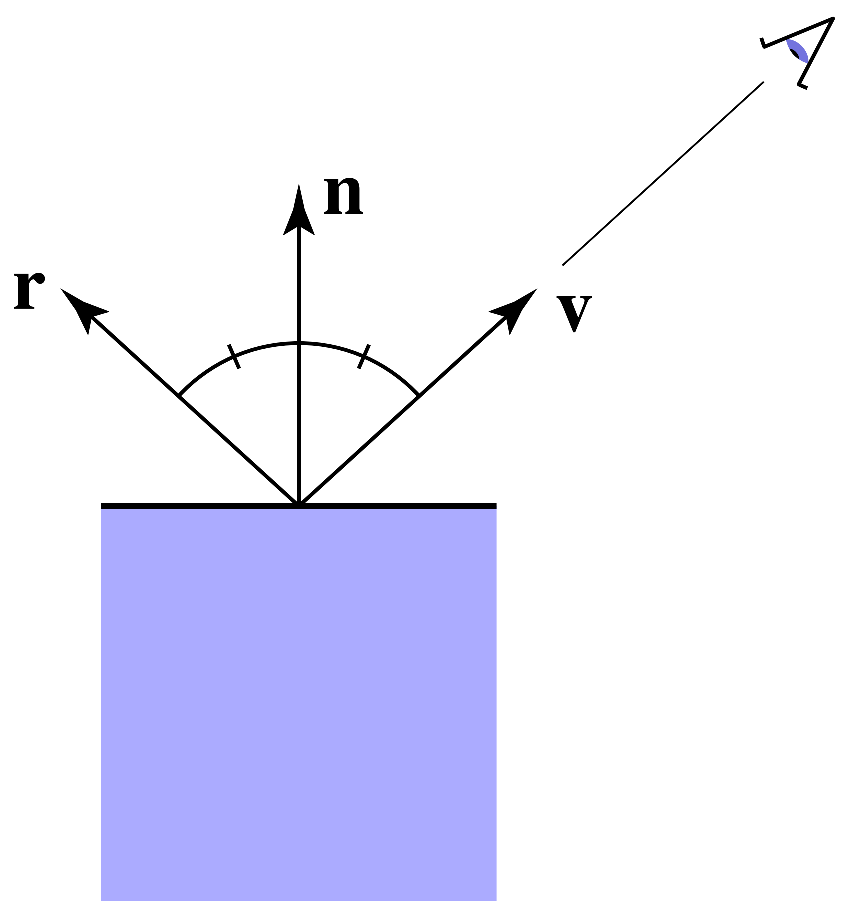
- **Consider perfectly shiny surface**
  - there isn't a highlight
  - instead there's a reflection of other objects
- **Can render this using recursive ray tracing**
  - to find out mirror reflection color, ask what color is seen from surface point in reflection direction
- **"Glazed" material has mirror reflection plus specular/diffuse**

$$L = L_a + L_r + L_m$$

  - where L_m is evaluated by tracing a new ray

Mirror reflection

- **Intensity depends on view direction**
  - reflects incident light from mirror direction



$$\begin{aligned}\mathbf{r} &= \mathbf{v} + 2((\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}) \\ &= 2(\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}\end{aligned}$$

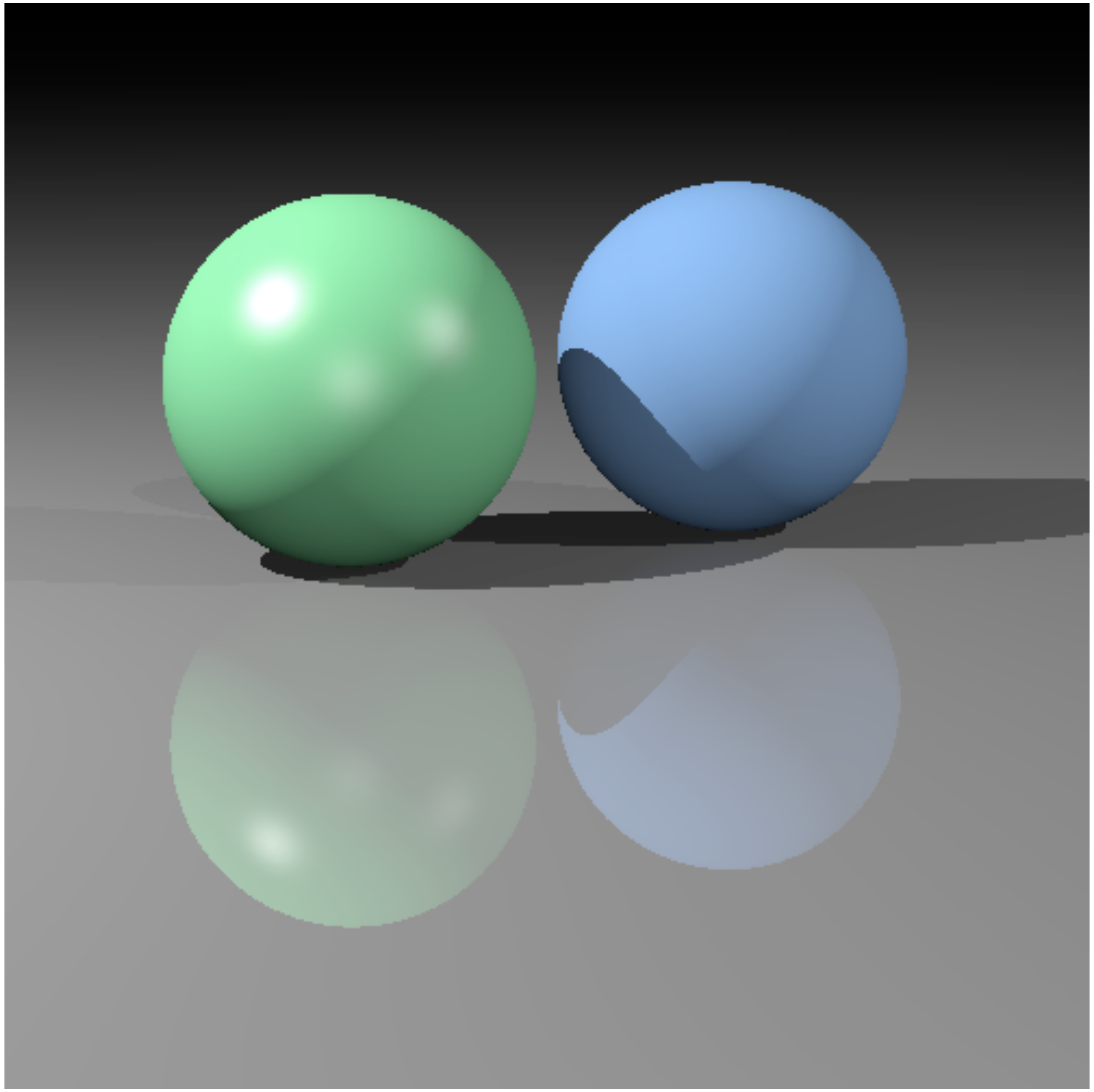
$$L_m = k_m\, L(\mathbf{r})$$

mirror-reflected light

mirror coefficient

result of tracing ray in direction **r**

Diffuse + mirror reflection (glazed)



(glazed material on floor)