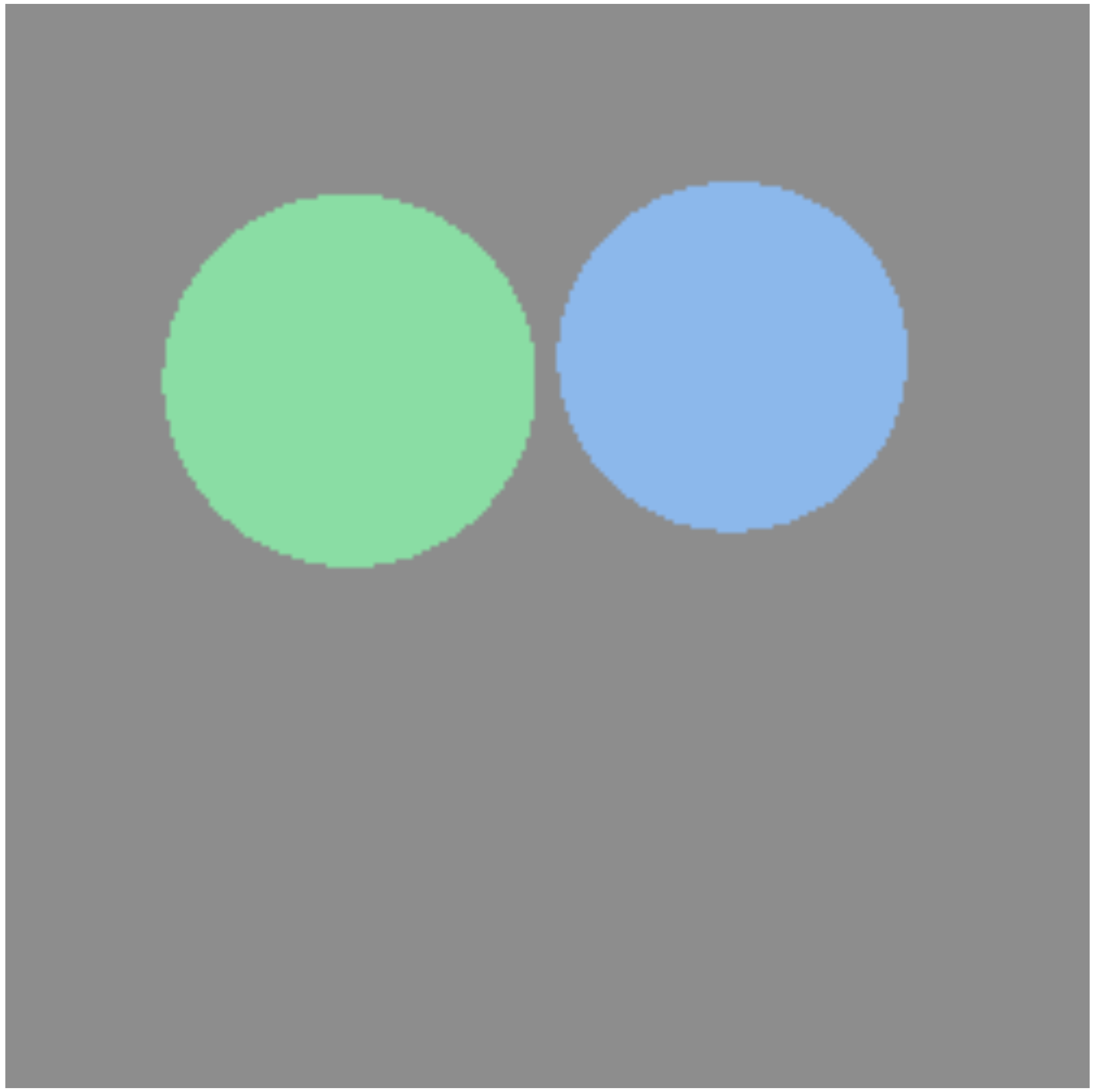
Ray Tracing: shading

CS 4620 Lecture 5

Image so far

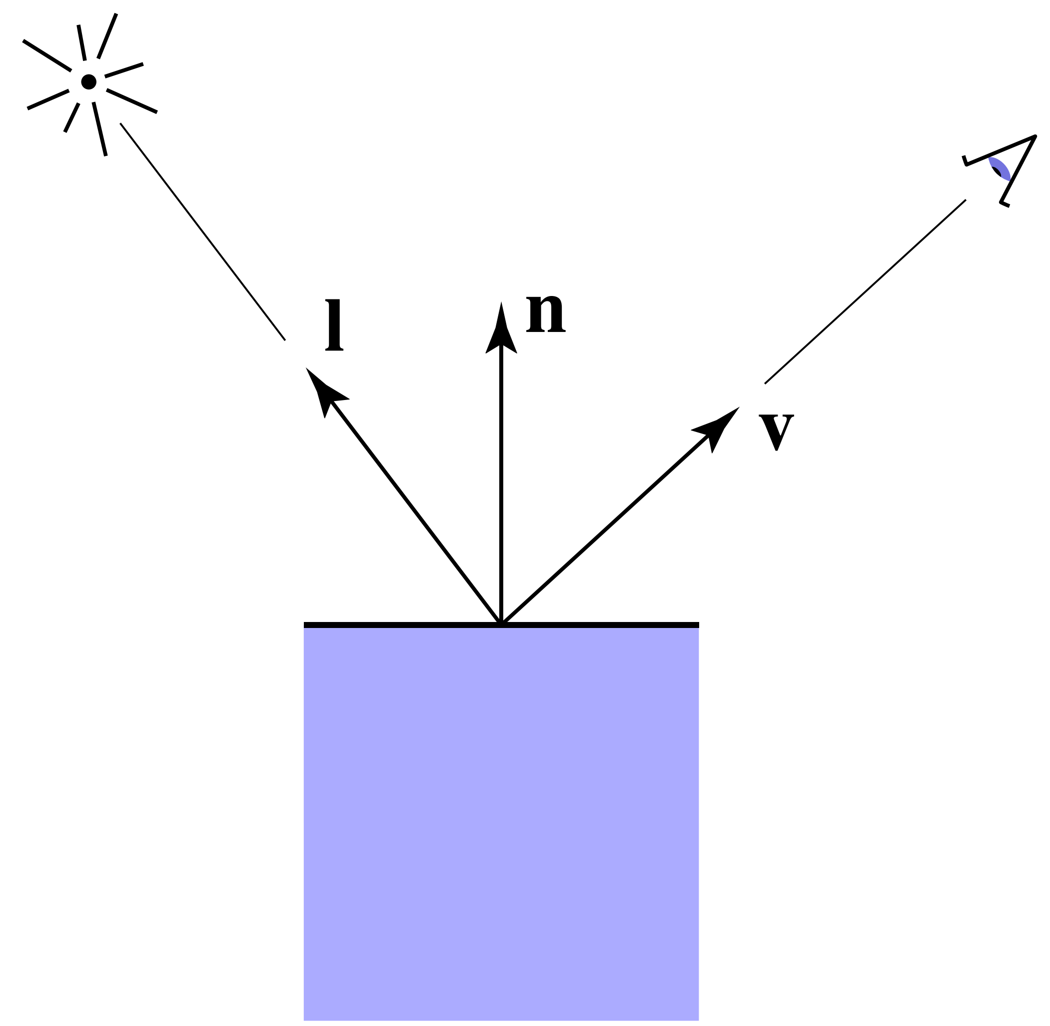
- **With eye ray generation and scene intersection**

```
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;
}
```



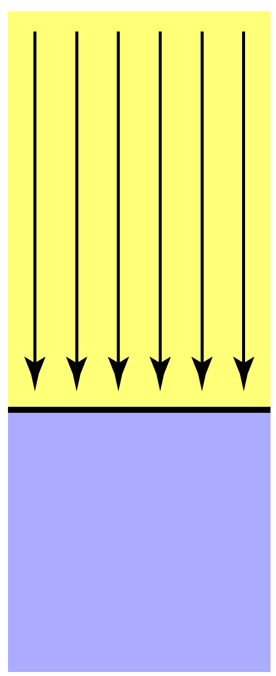
Shading

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

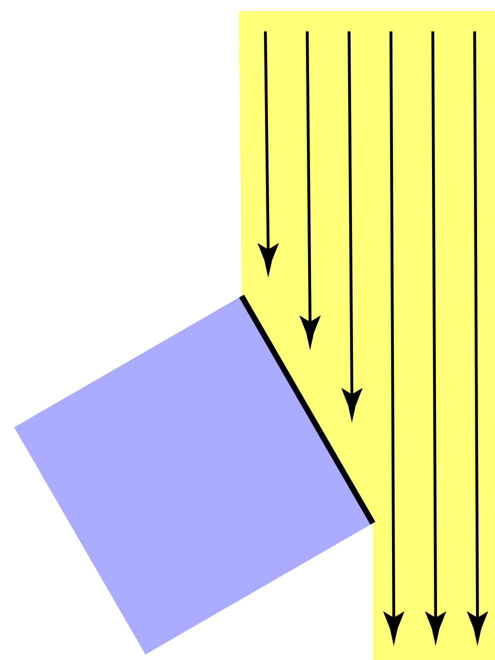


Diffuse reflection

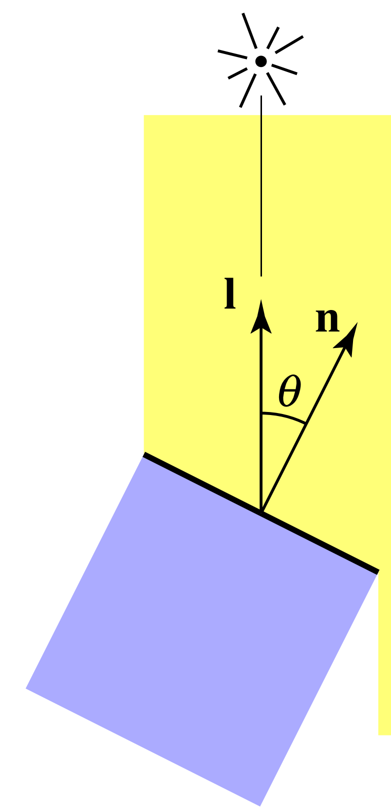
- **Light is scattered uniformly in all directions**
  – the surface color is the same for all viewing directions
- **Lambert's cosine law**



Top face of cube receives a certain amount of light

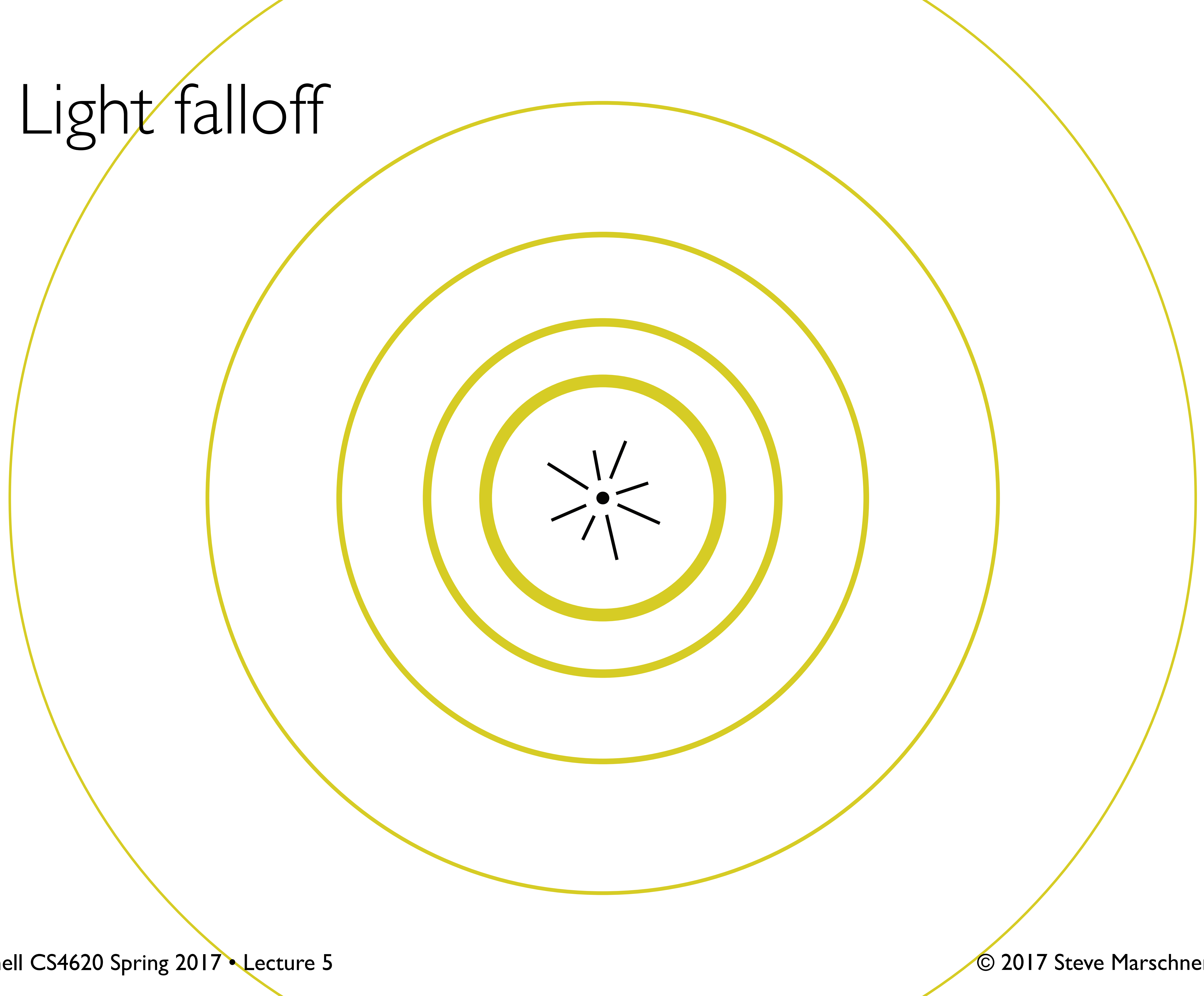


Top face of 60° rotated cube intercepts half the light

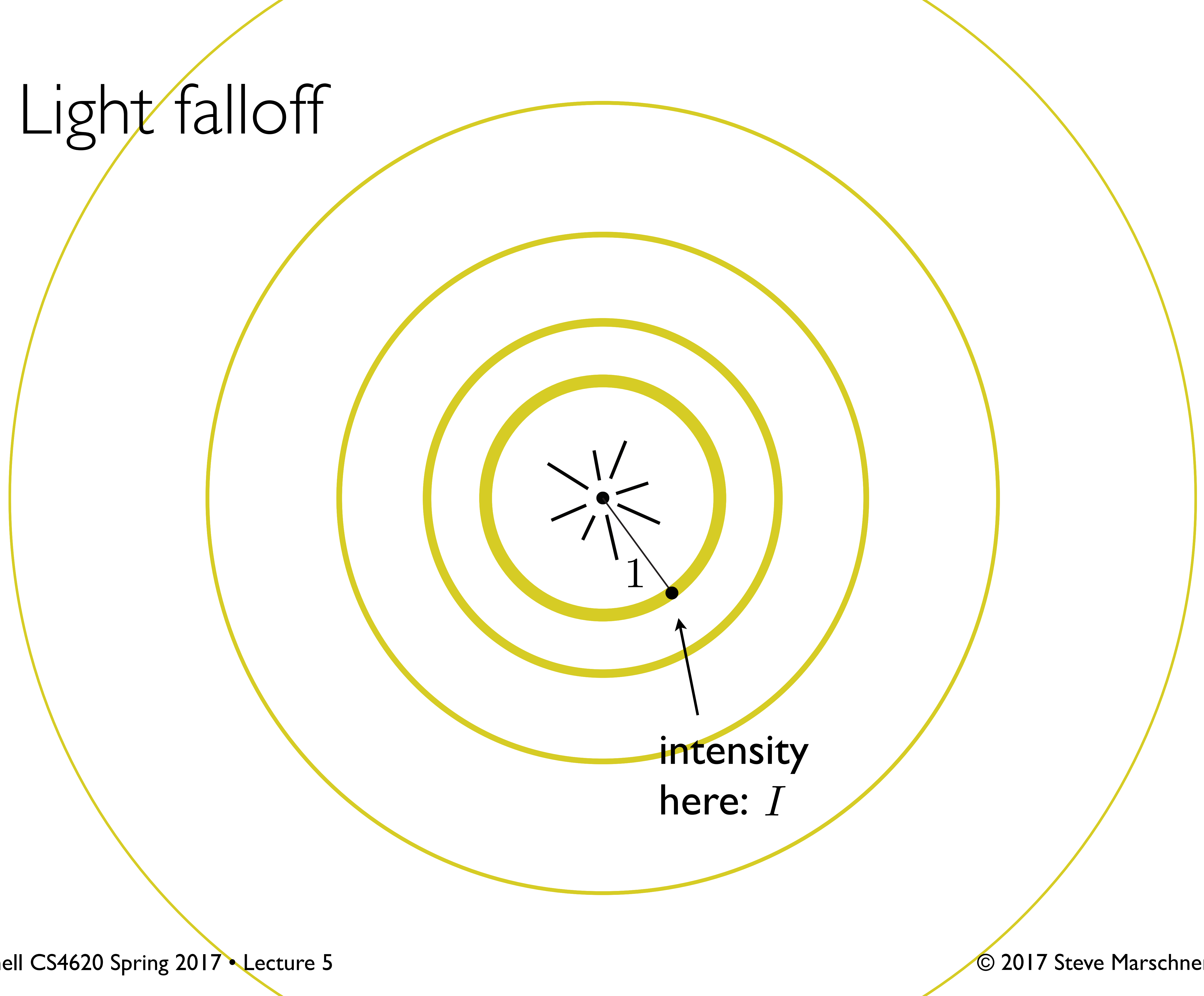


In general, light per unit area is proportional to $\cos\theta = \mathbf{l} \cdot \mathbf{n}$

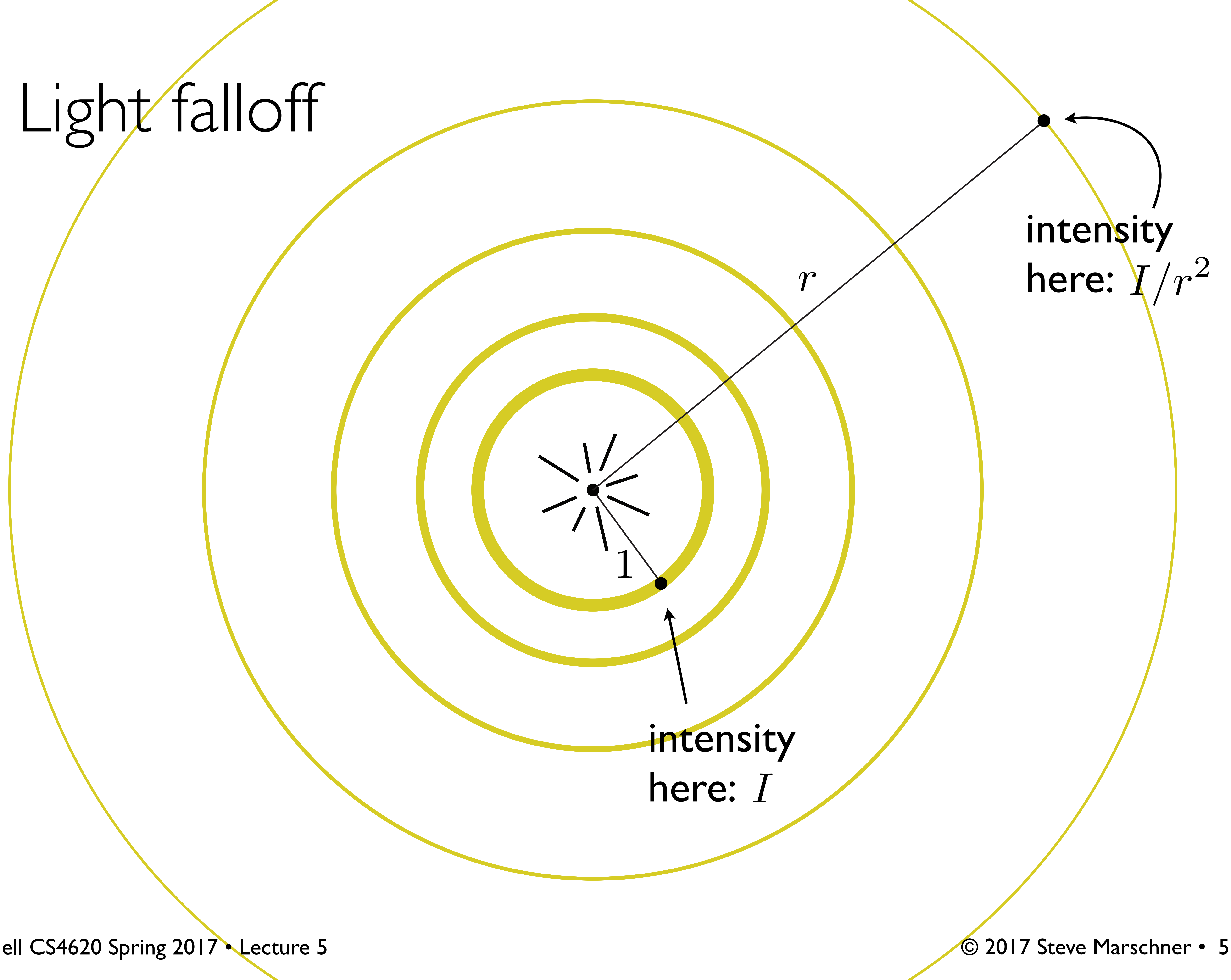
Light falloff



Light falloff

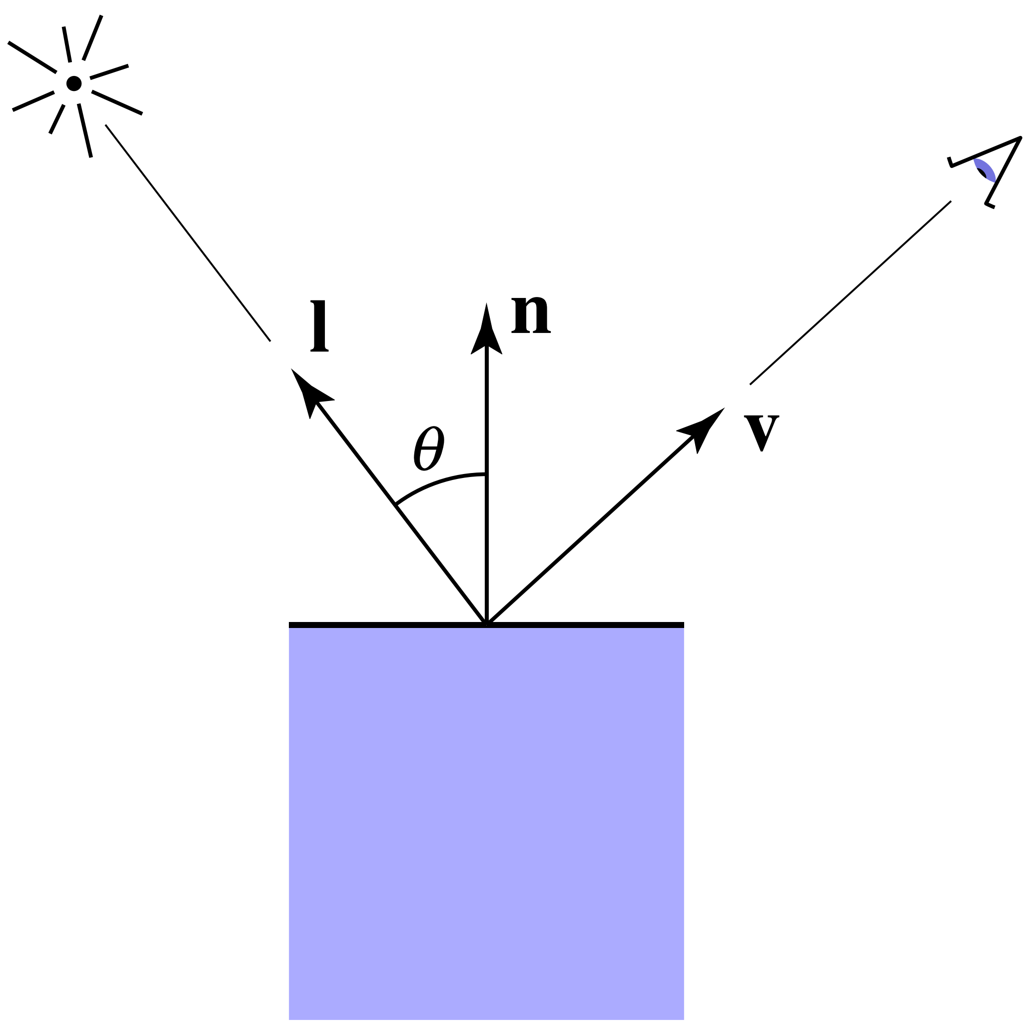


Light falloff



Lambertian shading

- **Shading independent of view direction**



illumination from source

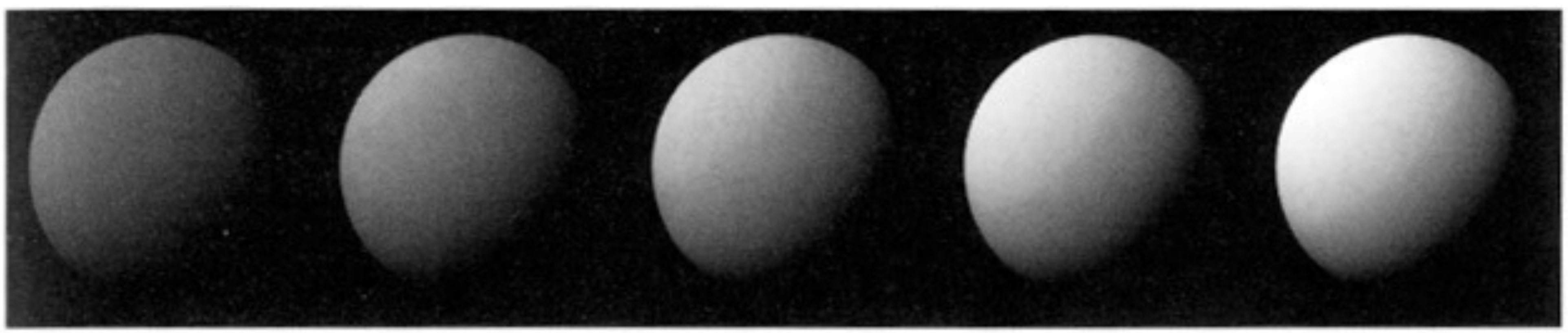
$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

diffuse coefficient

diffusely reflected light

Lambertian shading

- **Produces matte appearance**



$k_d \longrightarrow$

[Foley et al.]

Diffuse shading

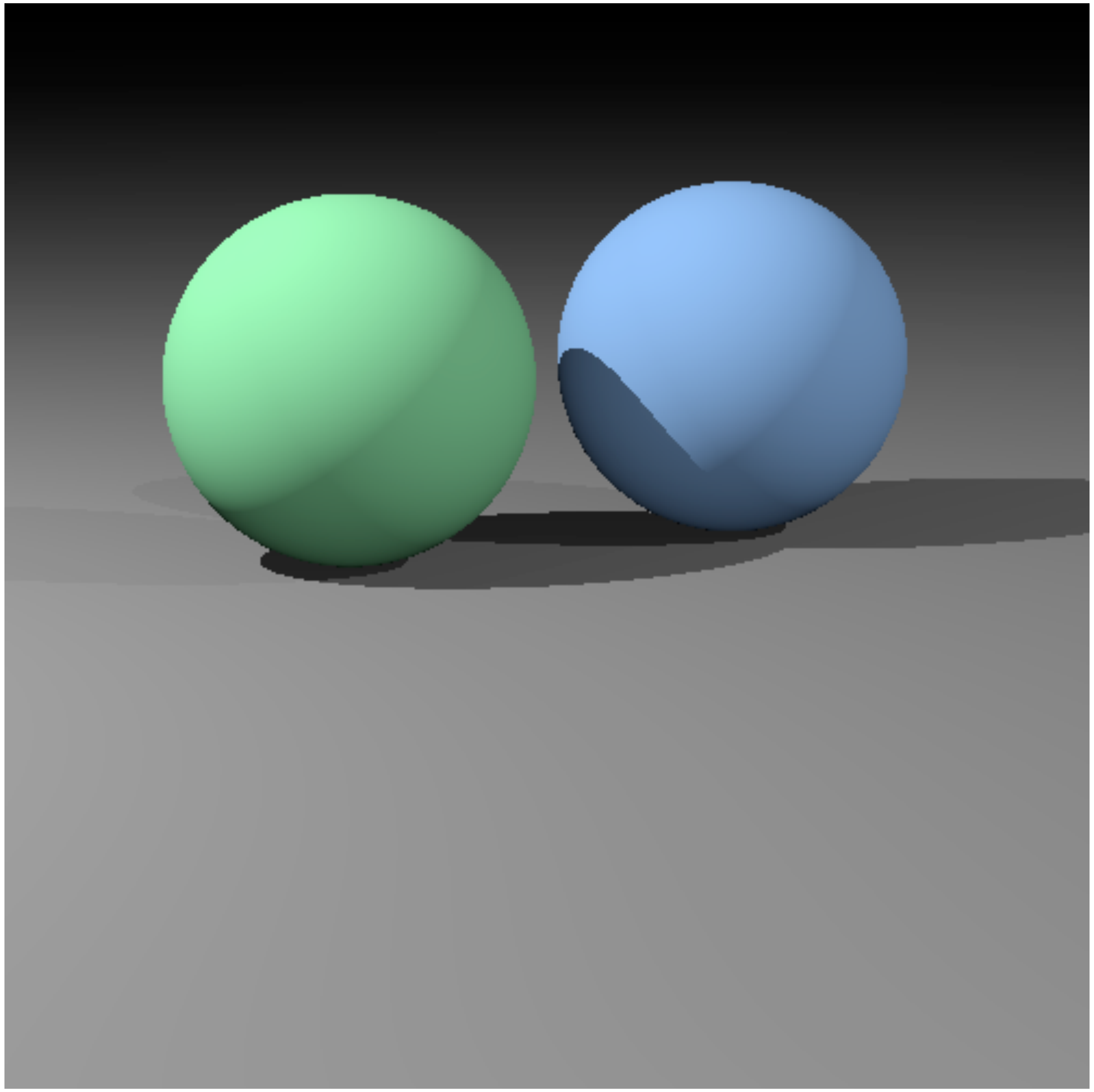
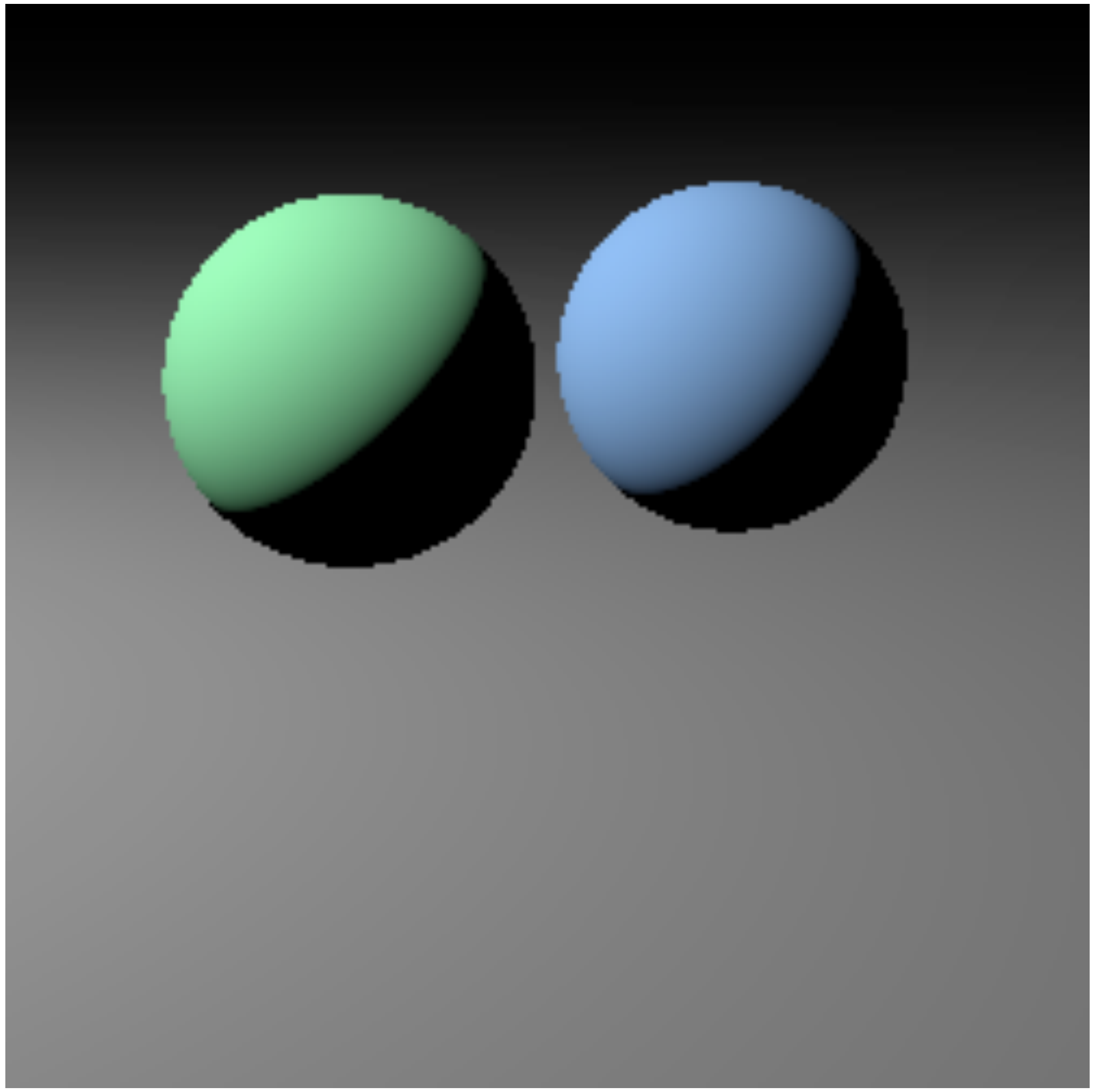


Image so far

```
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
}
```

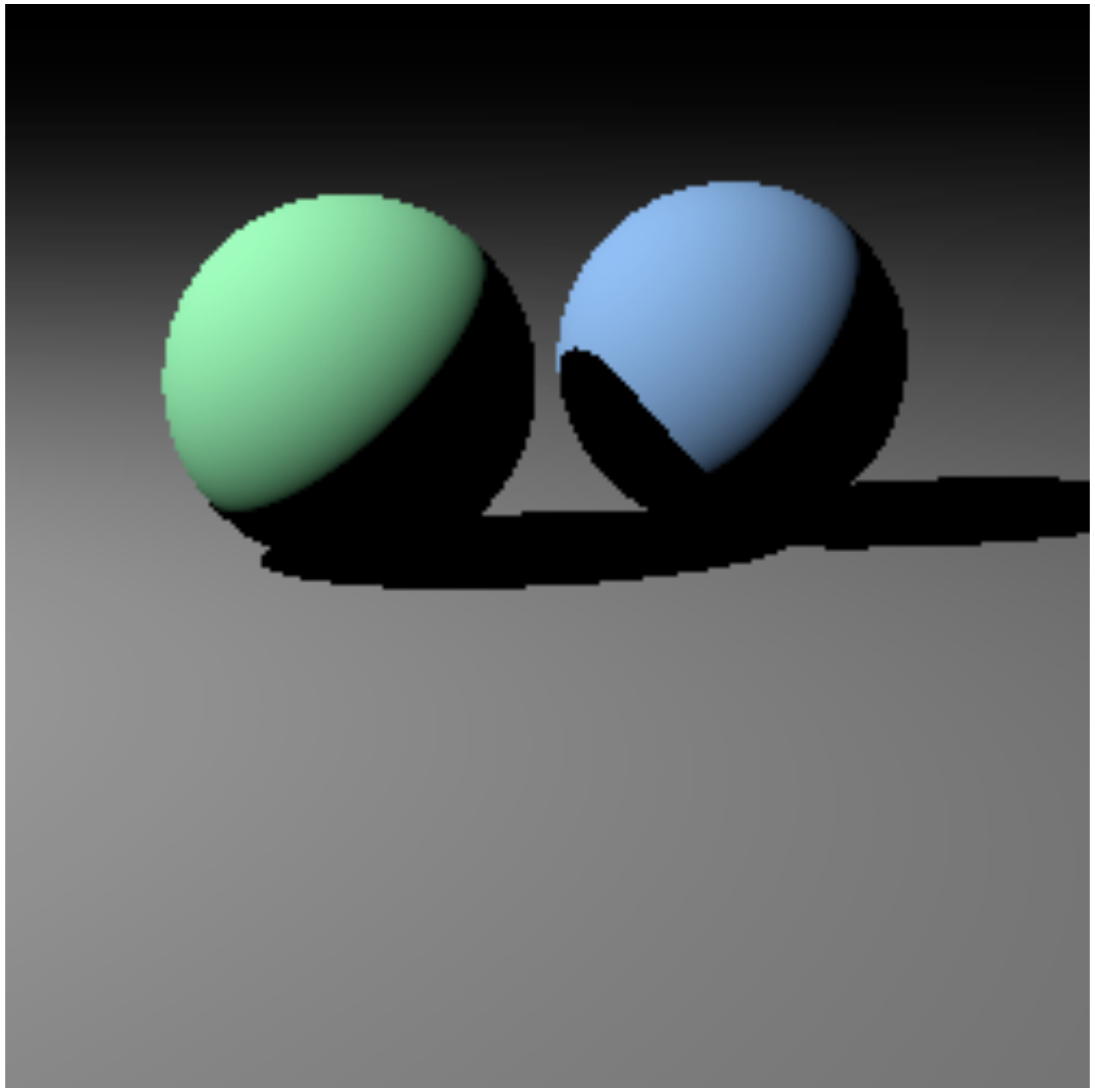


Shadows

- **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!

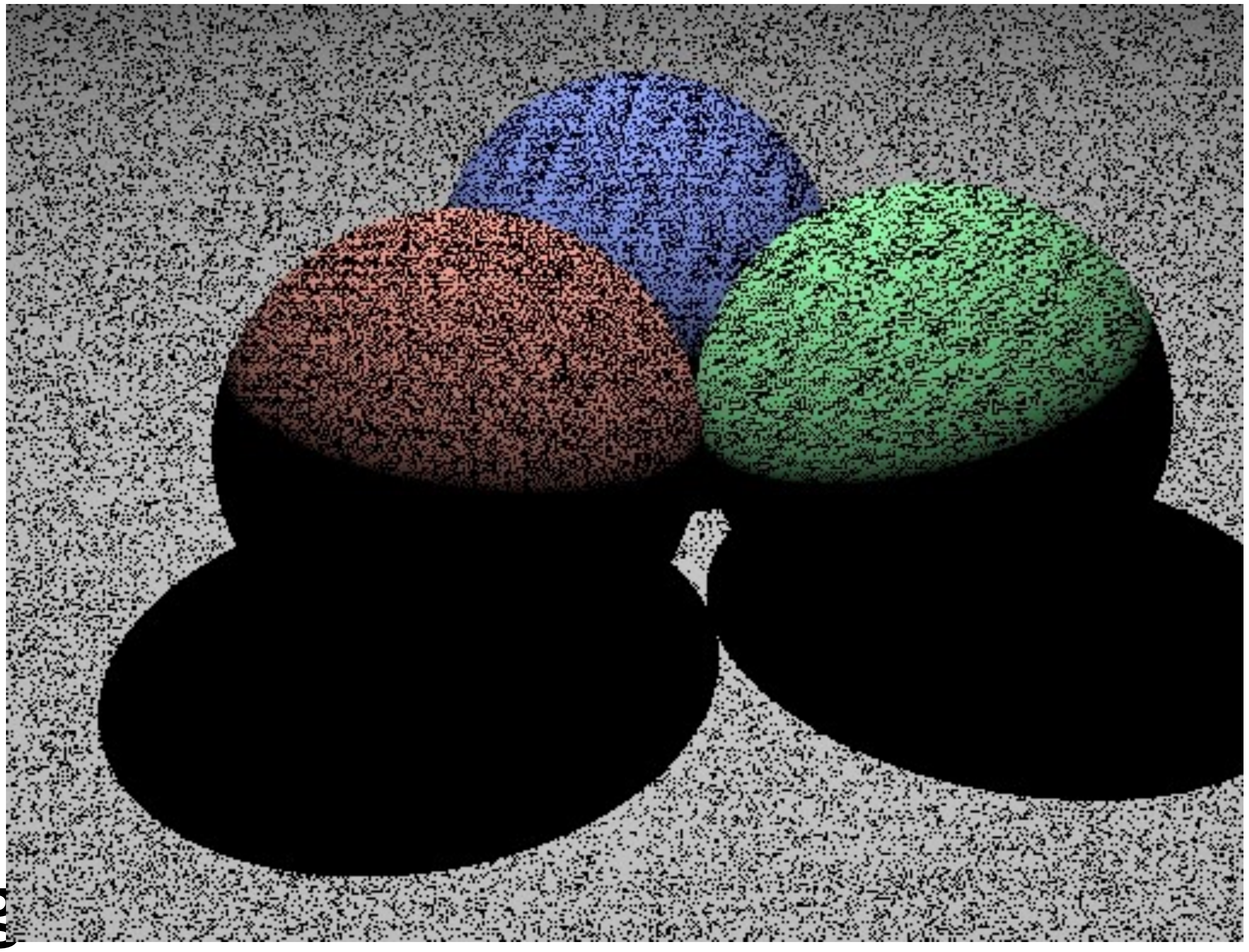
Image so far

```
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

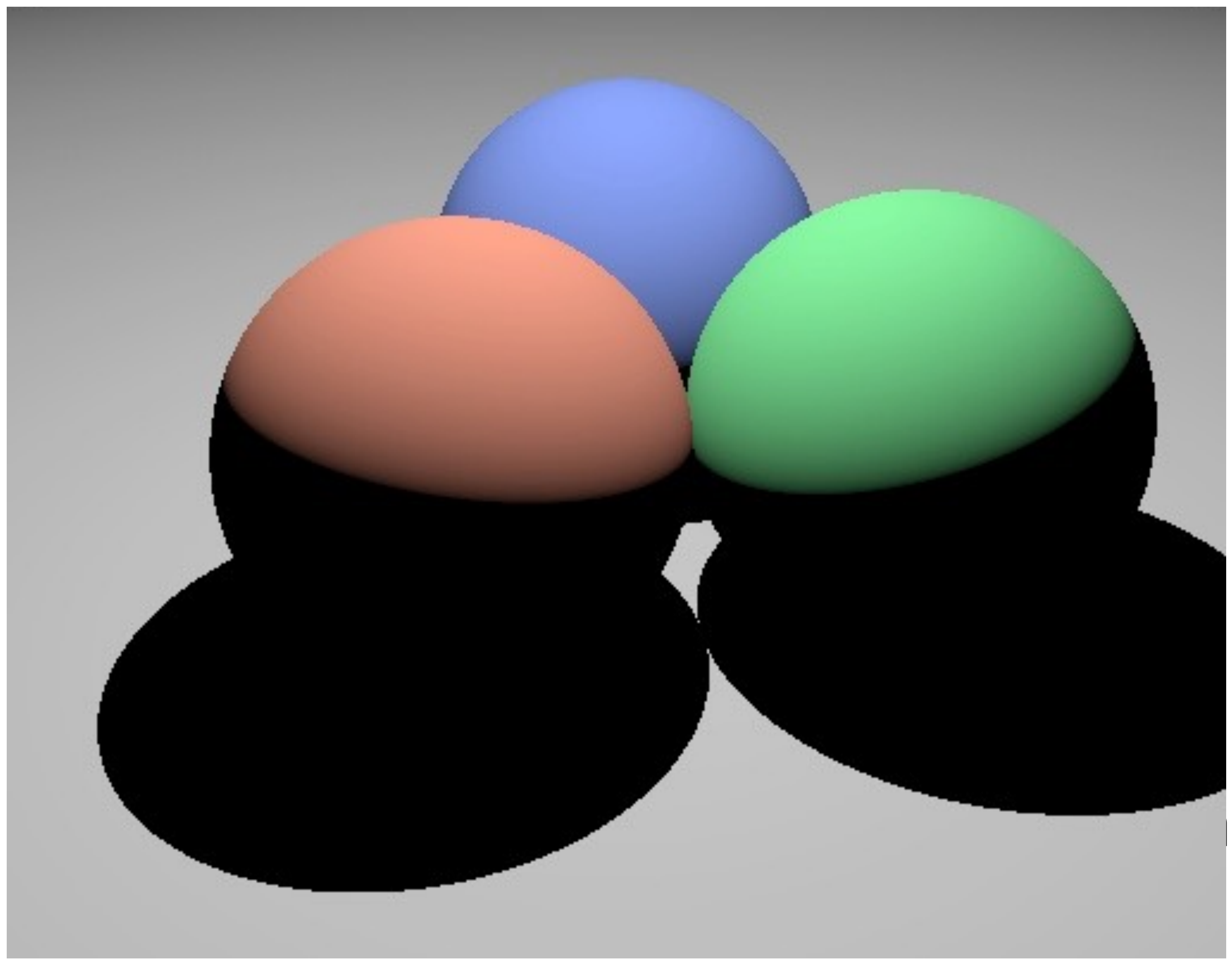
- **Don’t fall victim to one of the classic blunders:**



- **What’s going**
  - 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**



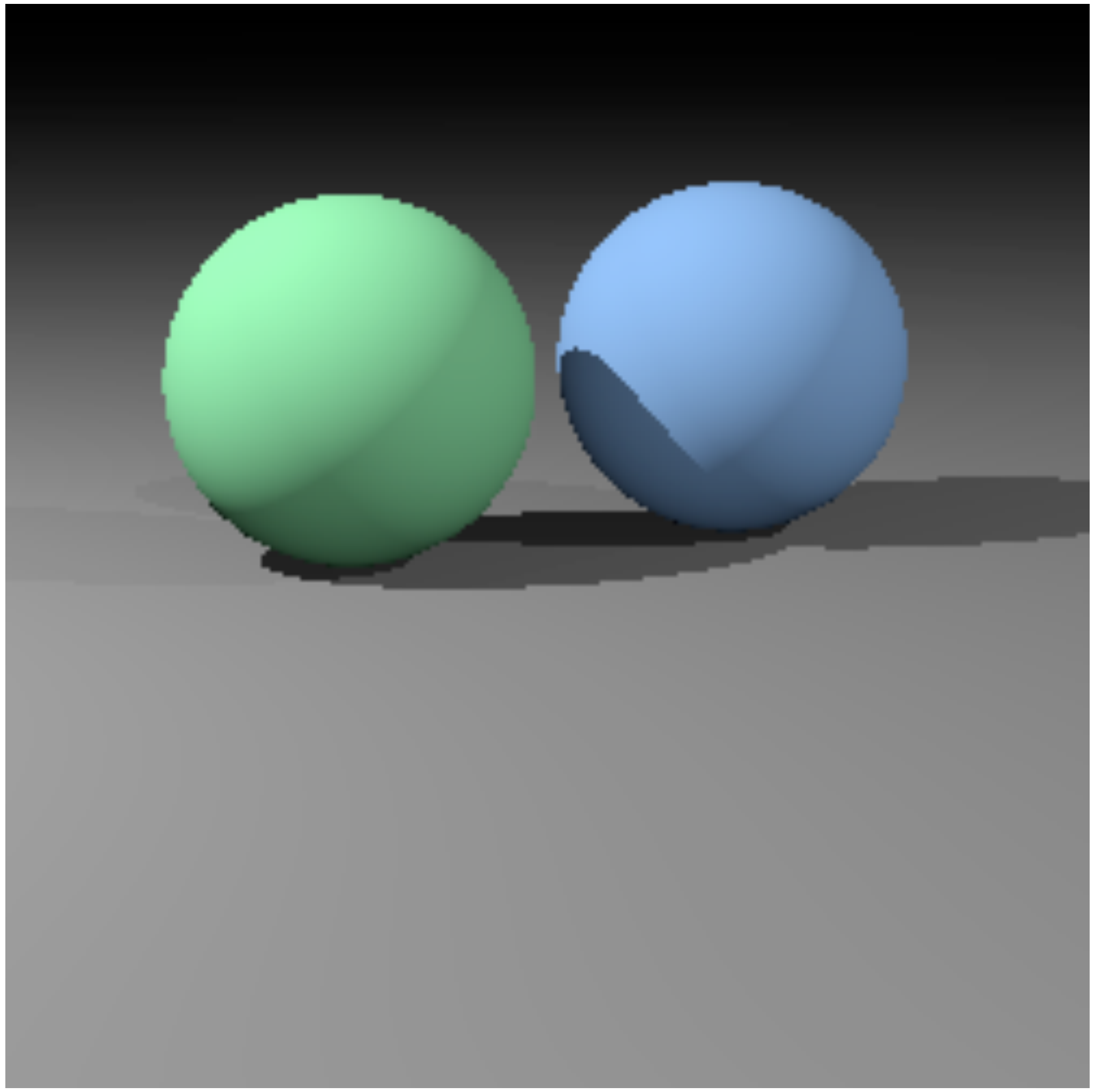
- **Do this by** … **g the *t* range**

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…

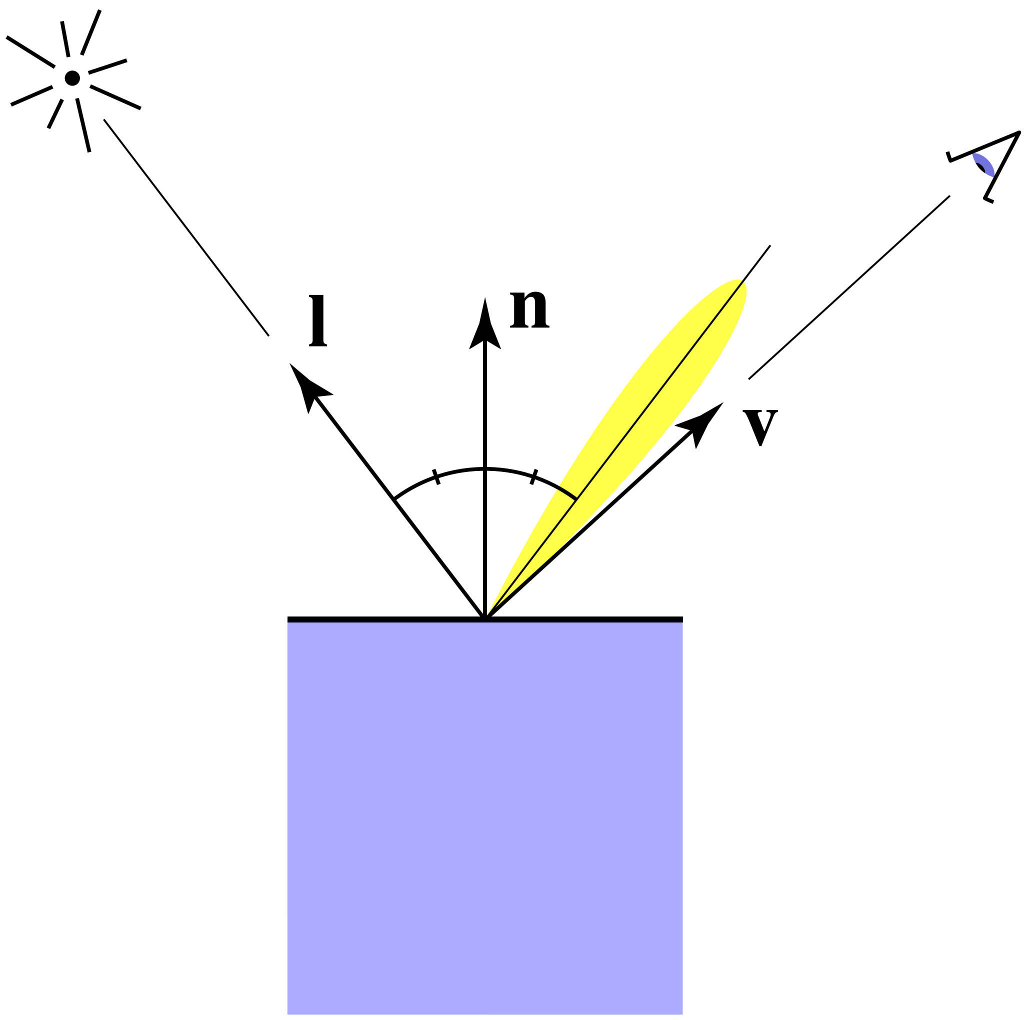
Image so far

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



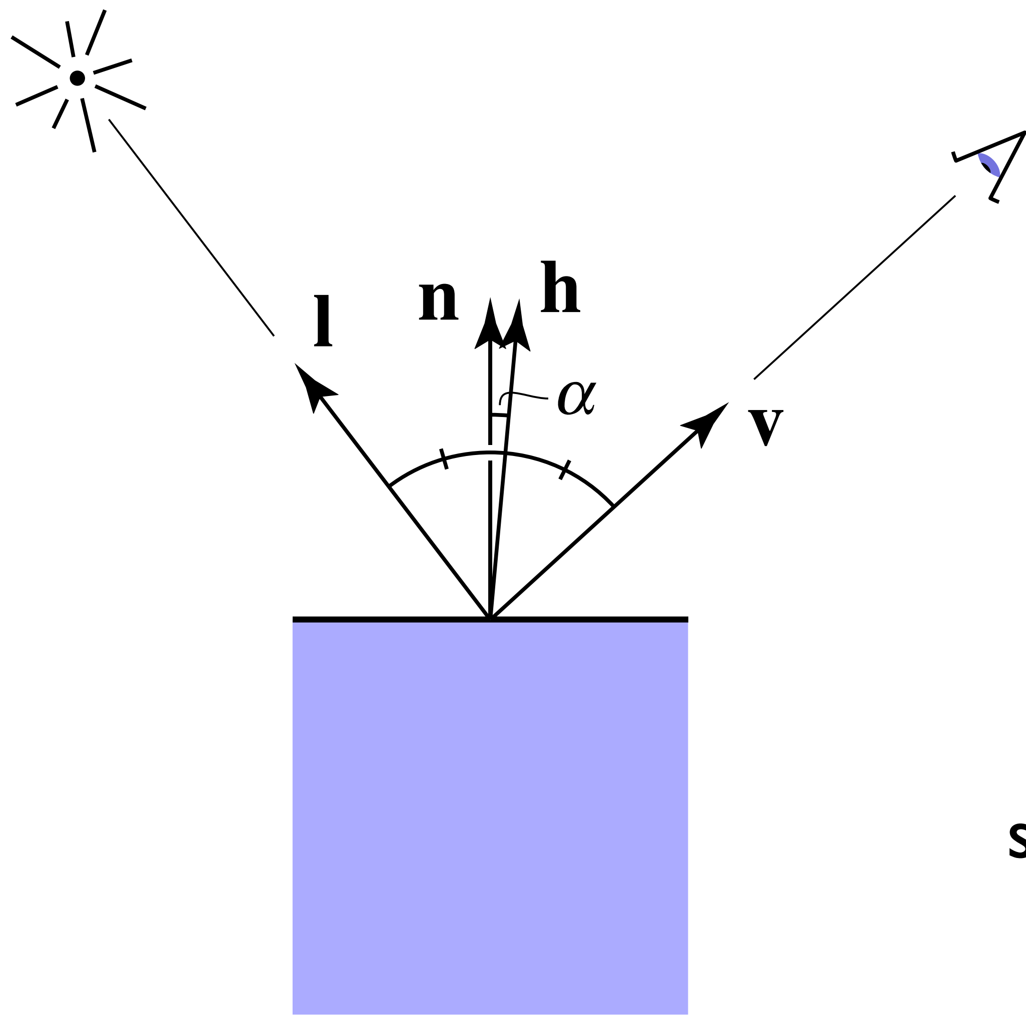
Specular shading (Blinn-Phong)

- **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



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

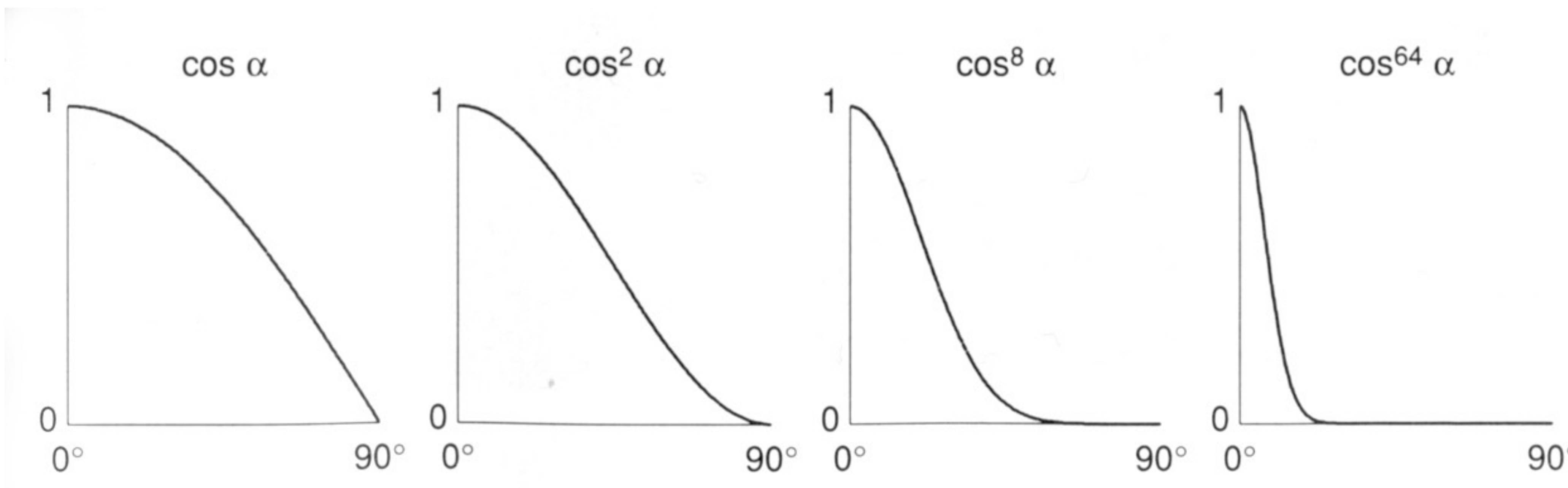
$$L_s = k_s\,(I/r^2)\max(0, \cos\alpha)^p$$
$$= k_s\,(I/r^2)\max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light

specular coefficient

Phong model—plots

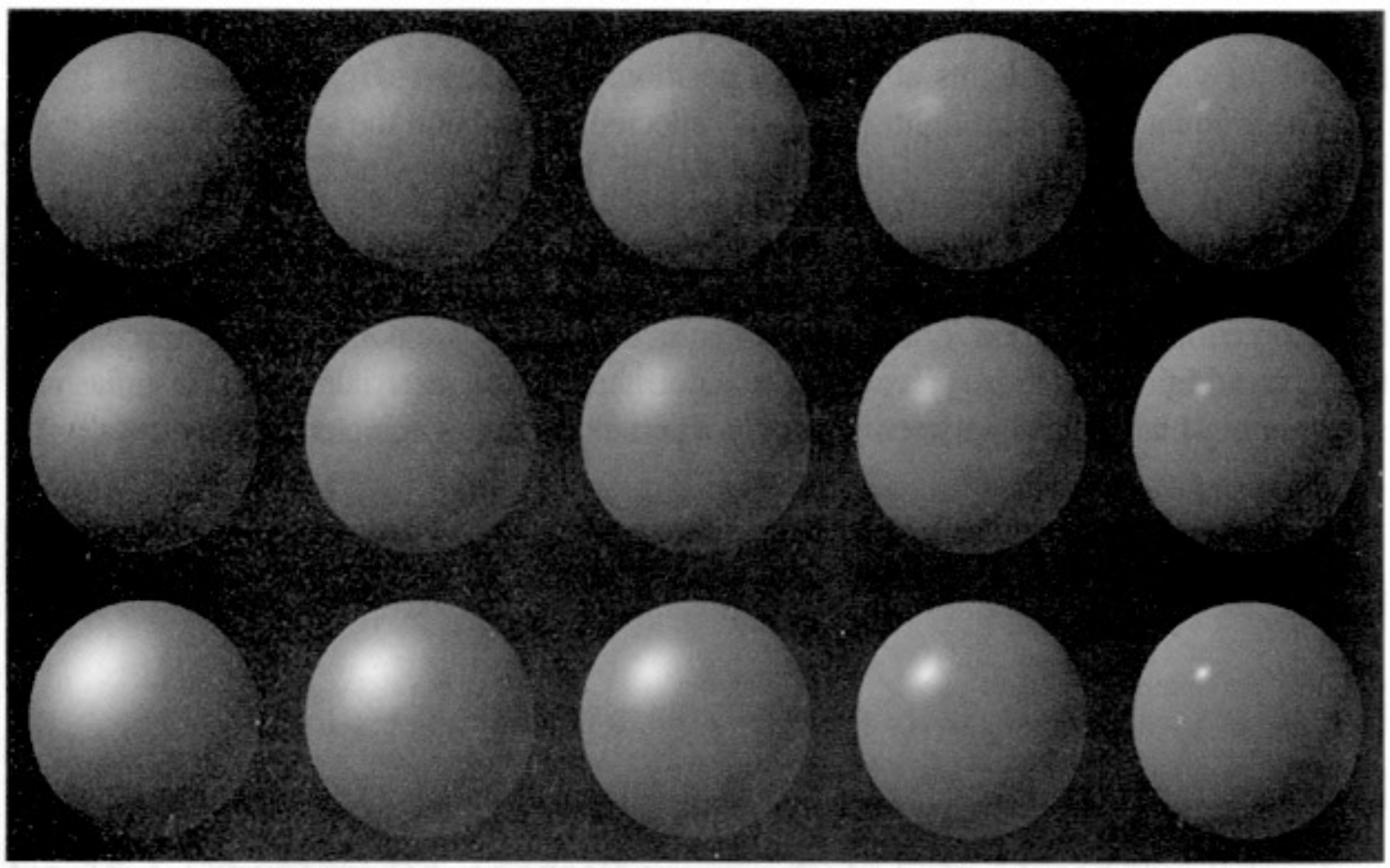
- **Increasing *p* narrows the lobe**



[Foley et al.]

Specular shading

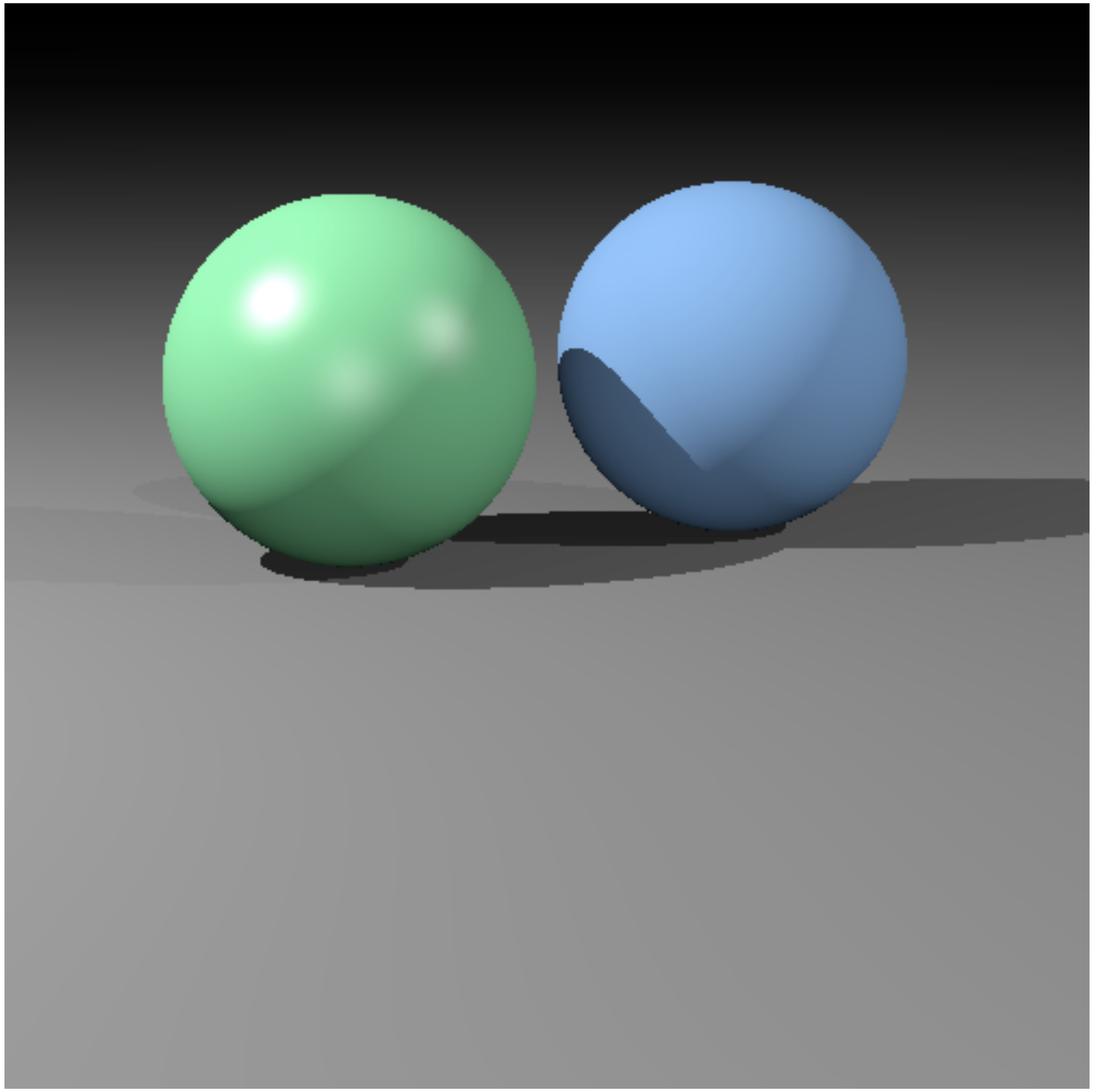
k_s



p

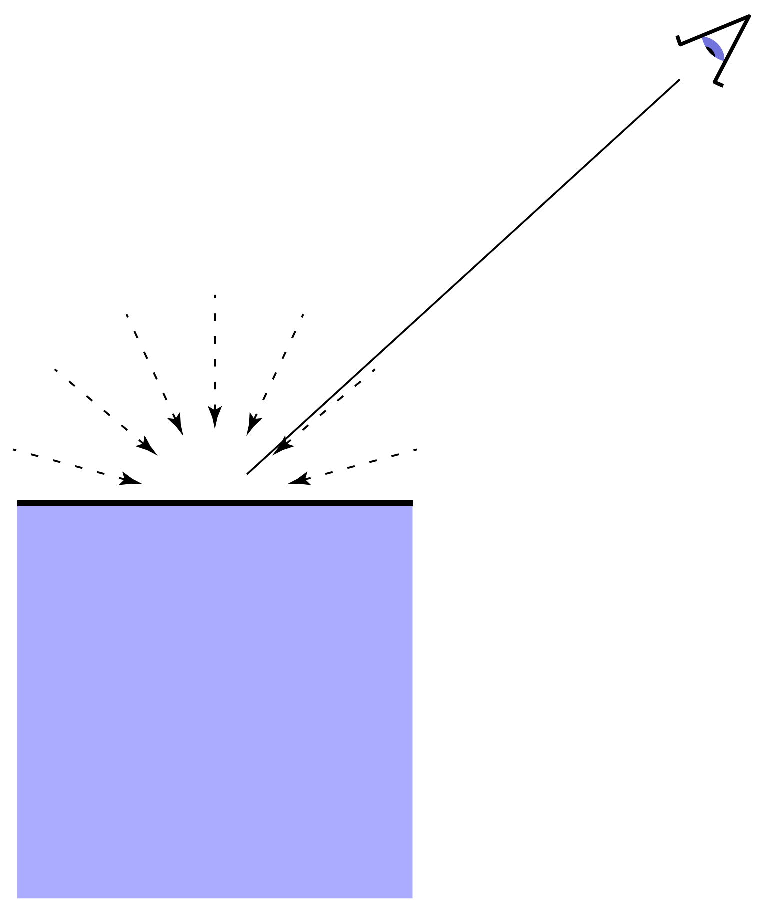
[Foley et al.]

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



$$L_a = k_a \, I_a$$

ambient coefficient

reflected ambient light

Putting it together

- **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**

$$
L = L_a + \sum_{i=1}^{N} \left[(L_d)_i + (L_s)_i \right]
$$

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

