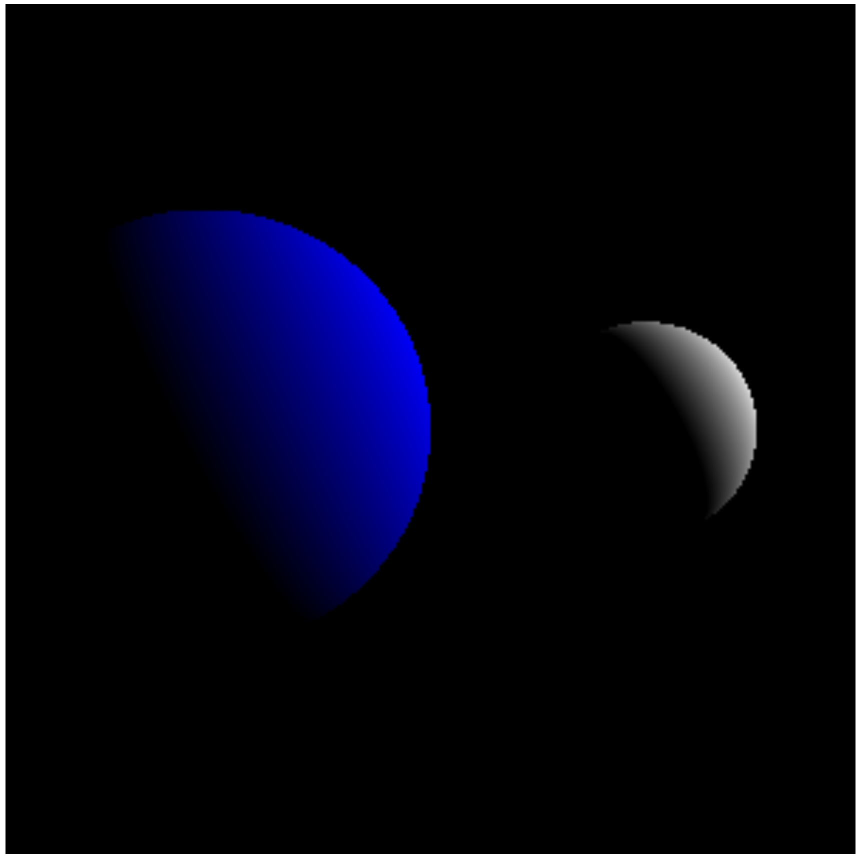
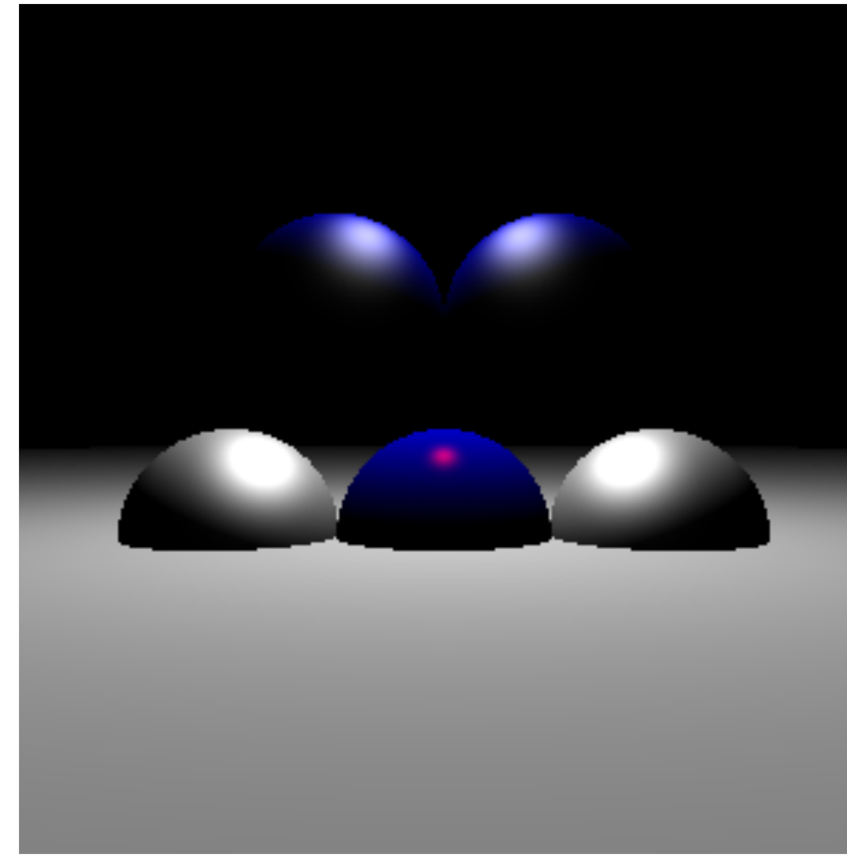
  

# Computer Graphics

Lecture 7

**Ray-Sphere Intersection**

**Lights and Shading**

# Announcements

# Announcements

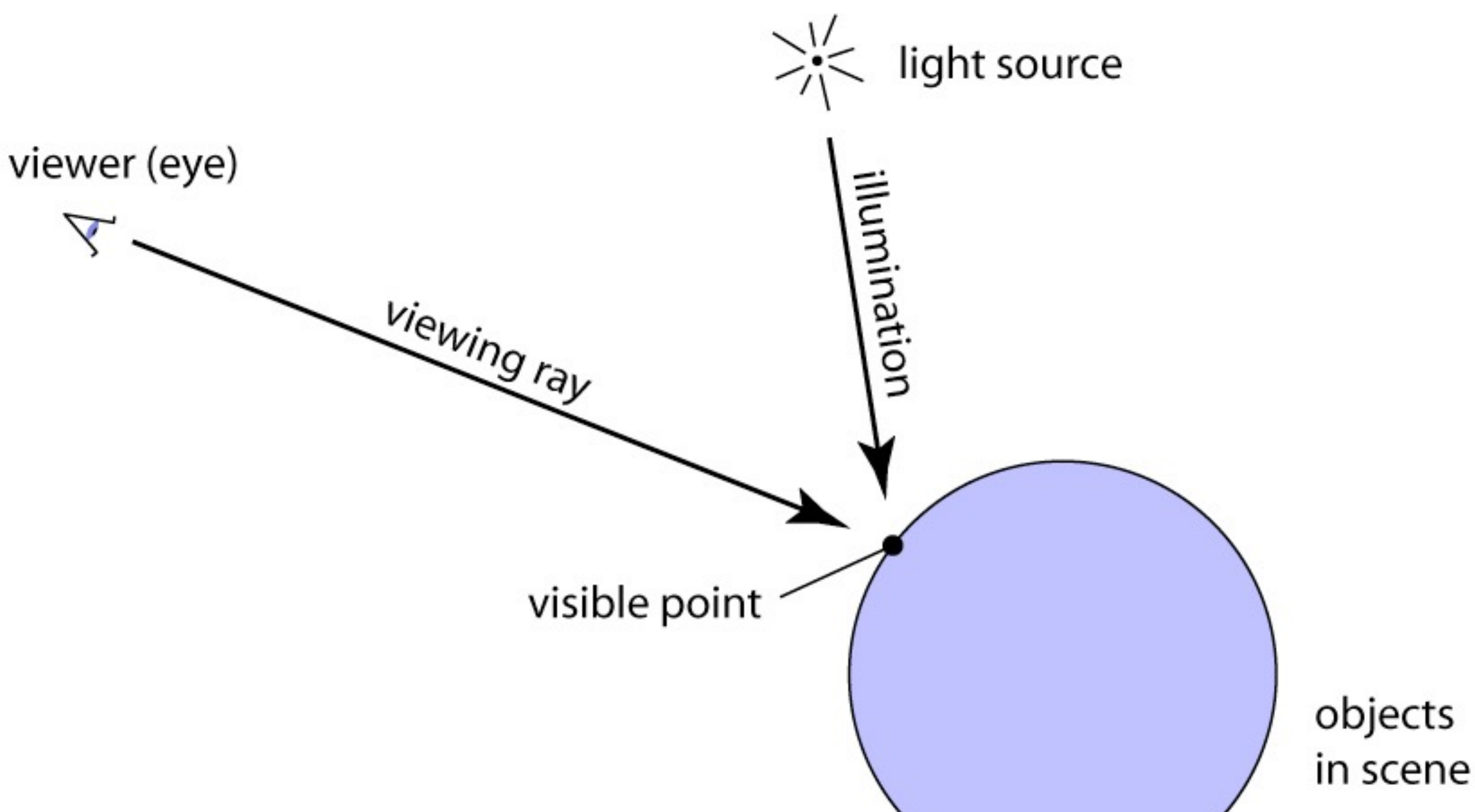
- A2 out later today.

# Announcements

- A2 out later today.
- HW1 out soonish.

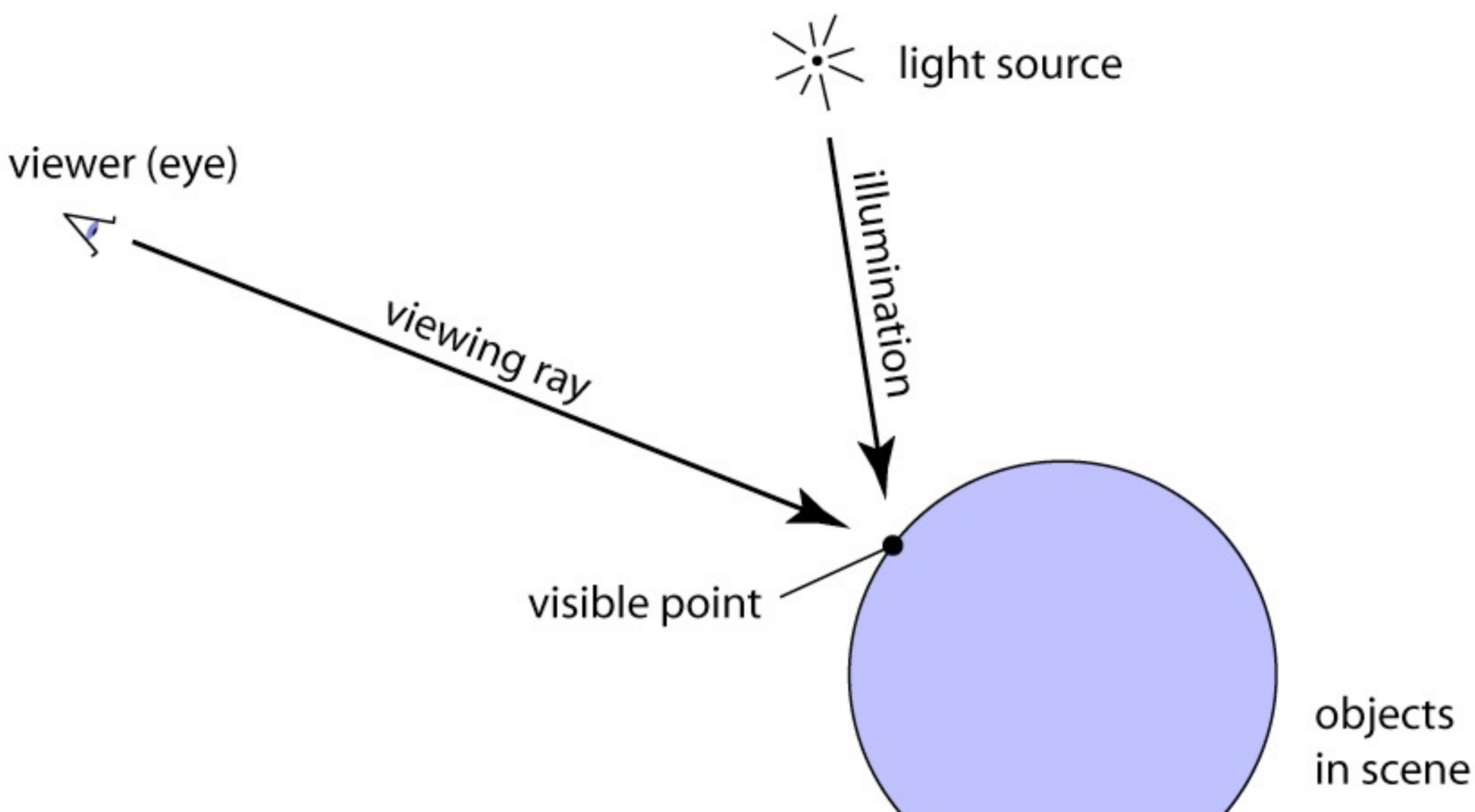
# Ray Tracing: Pseudocode

```
for each pixel:
    generate a viewing ray for the pixel
    find the closest object it intersects
    determine the color of the object
```



# Ray Tracing: Pseudocode

```
for each pixel:
    generate a viewing ray for the pixel
    find the closest object it intersects
    determine the color of the object
```



# Implicit vs Parametric

- Implicit equations: a property true at all points
  - e.g., $ax + by + c = 0$, for a line
- Parametric equations: use a free parameter variable to *generate* all points:
  - e.g., $r(t) = \mathbf{p} + t\mathbf{d}$, for a line
- Intersecting parametric with implicit is usually cleanest.

# Ray-Sphere Intuition: Geometric

- How many times will can ray intersect a sphere?
- For now, consider a unit sphere at the origin.
- What's an implicit equation for a sphere? or: What's true of all points on a sphere?

# Ray-Sphere Intuition: Geometric

- How many times will can ray intersect a sphere?
- For now, consider a unit sphere at the origin.
- What's an implicit equation for a sphere?
  or: What's true of all points on a sphere?

Intuition: LHS gives the point's **signed distance** from sphere.

# Ray-Sphere Intuition: Geometric

- How many times will can ray intersect a sphere?
- An implicit equation for a sphere:

$$x^2 + y^2 + z^2 - 1 = 0$$

Intuition: LHS gives any 3D point's (squared) **signed distance** from sphere's surface.

# Ray-Sphere Intersection: Algebraic

Whiteboard / notes.

# Ray-Sphere intersection

- For now, assume unit sphere centered at the origin. See 4.4.1 for general derivation.

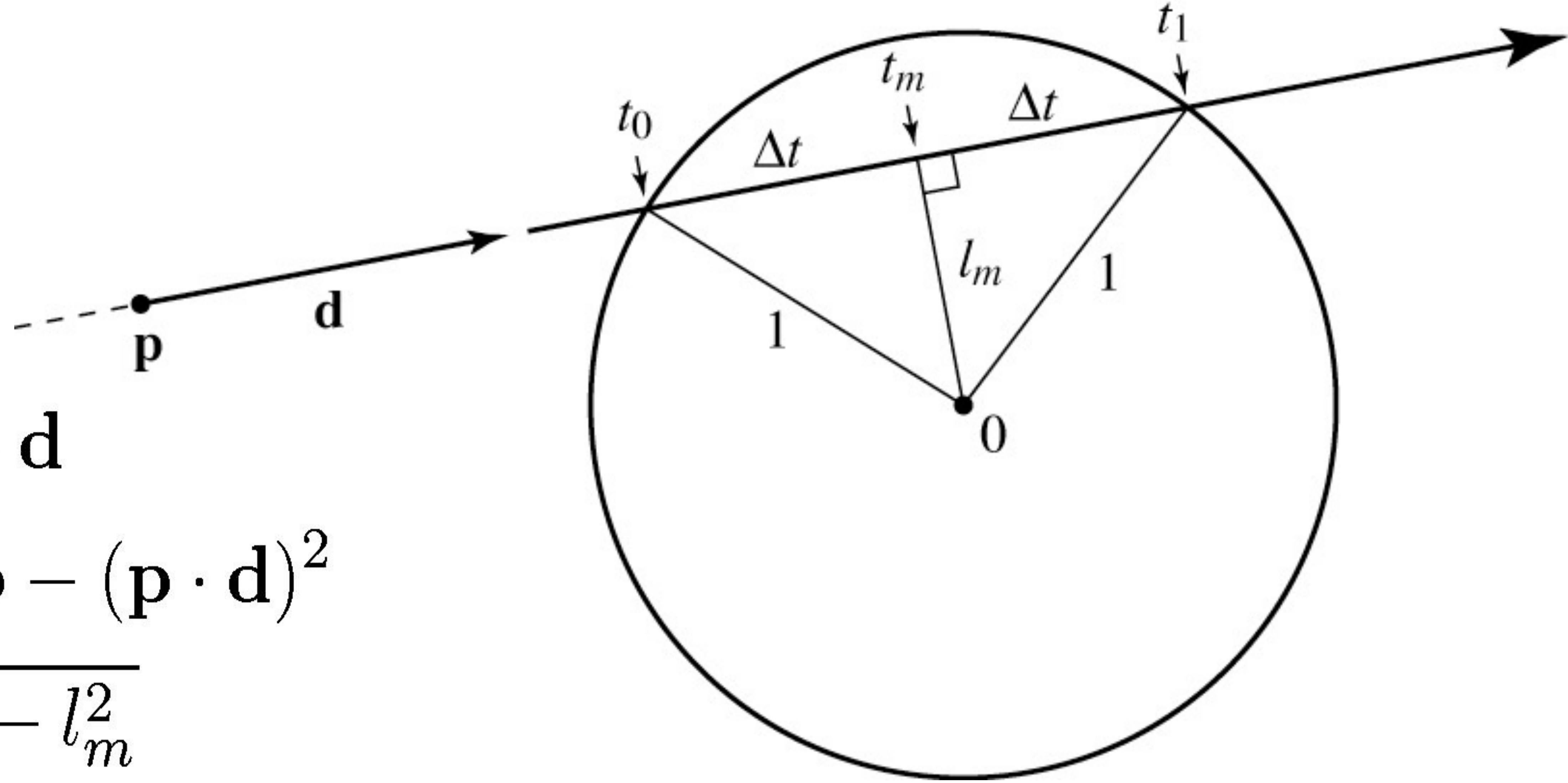
$$t = \frac{-\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - (\mathbf{d} \cdot \mathbf{d})(\mathbf{p} \cdot \mathbf{p} - 1)}}{\mathbf{d} \cdot \mathbf{d}}$$

If **d** is unit-length:

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

# Geometric Intuition

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$




$$t_m = -\mathbf{p} \cdot \mathbf{d}$$

$$l_m^2 = \mathbf{p} \cdot \mathbf{p} - (\mathbf{p} \cdot \mathbf{d})^2$$

$$\Delta t = \sqrt{1 - l_m^2}$$

$$= \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

$$t_{0,1} = t_m \pm \Delta t = -\mathbf{p} \cdot \mathbf{d} \pm \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

# Ray-Sphere: Code Sketch

```
function ray_intersect(ray, sphere, tmin, tmax):
```

- Use above math to find +/- *t*
- If none, return `nothing`
- Otherwise, return closest *t* that lies between `tmin` and `tmax`

# Ray-Scene: Code Sketch

**Brute force:** check all objects.
There are better ways - more on this later.

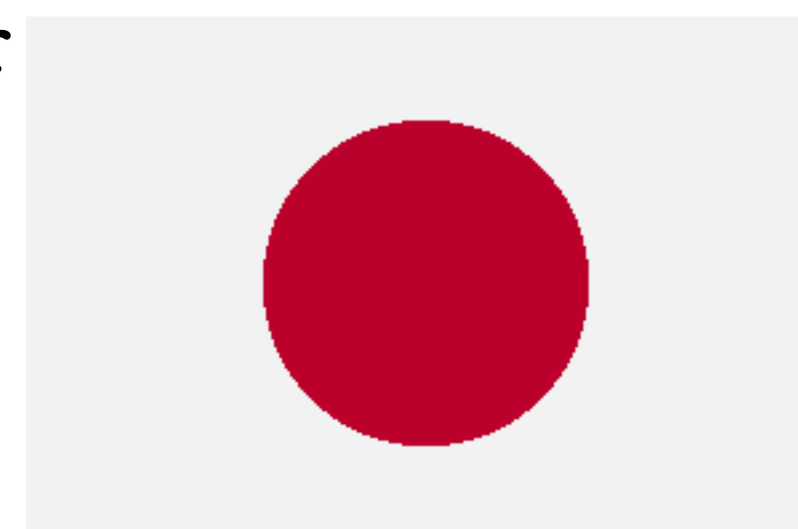
```
find_intersection(ray, scene):
  closest_t = Inf
  closest_obj = nothing
  for obj in scene:
    t = ray_intersect(ray, obj, 1, closest_t)
    if obj != nothing:
      closest_t = t
      closest_obj = surf
  return closest_t, closest_obj
```

# Ray Tracing: Code Sketch

```
scene = model_scene()
for each pixel (i,j):
    ray = get_view_ray(i, j)
    t, obj = find_intersection(ray, scene)
    if obj != nothing:
      canvas[i,j] = obj.color
    else:
      canvas[i,j] = scene.bgcolor
```

# Ray Tracing: Code Sketch

```
scene = model_scene()
for each pixel (i,j):
    ray = get_view_ray(i, j)
    t, obj = find_intersection(ray, scene)
    if obj != nothing:
        canvas[i,j] = obj.color
    else:
        canvas[i,j] = scene.bgcolor
```



# Ray Tracing: Code Sketch

```
scene = model_scene()
for each pixel (i,j):
    ray = get_view_ray(i, j)
    t, obj = find_intersection(ray, scene)
    if obj != nothing:
        canvas[i,j] = obj.color
    else:
        canvas[i,j] = scene.bgcolor
```

**Let's work on this.**

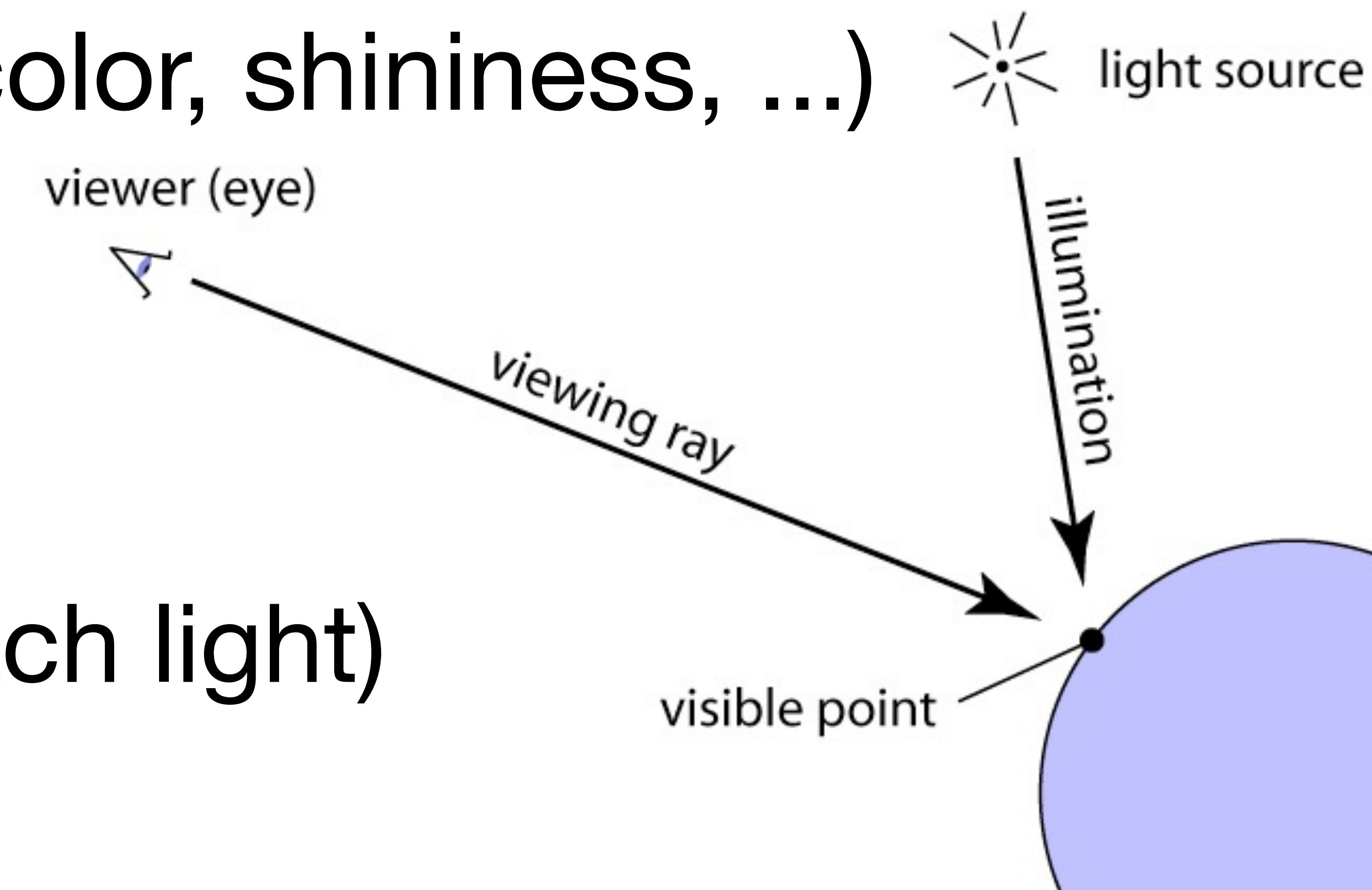
# Shading

- What does the color of a pixel depend on?

# Shading

What does the color of a pixel depend on?

- surface normal
- surface properties (color, shininess, ...)
- eye direction
- light direction (for each light)



# Shading

What does the color of a pixel depend on?

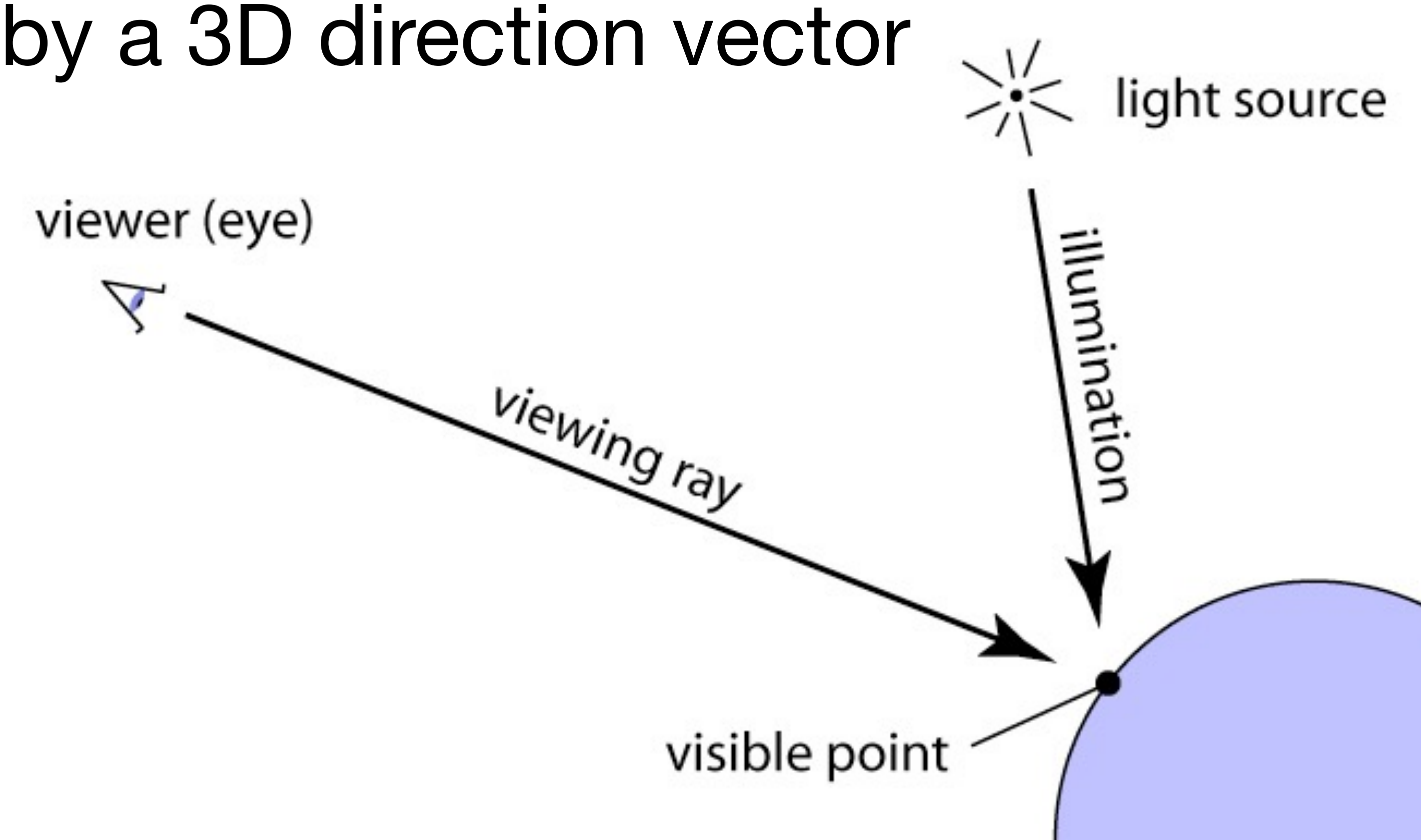
- surface normal *stored in or calculated from object*
- surface properties (color, shininess, ...) *stored in object*
- eye direction *calculated from viewing ray and intersection point*
- light direction (for each light) *calculated from light and intersection point*

# Eye Direction: Exercise

Given a ray ($\mathbf{p} + t\mathbf{d}$) and the $t$ at which it intersects a surface, find a unit vector giving the direction from the surface towards the viewer.

# Light Sources

- Where does light come from?
- Two simple kinds of sources:
  - point source: defined by a 3D position
  - directional source: defined by a 3D direction vector



# Light Sources: Exercise

Given a ray (**p** + *t***d**) and the *t* at which it intersects a surface, calculate a unit vector giving the direction from the surface towards:

- a point light source at position $\vec{s}$
- a directional light source with direction $\vec{\ell}$

# Diffuse (Lambertian) Reflection

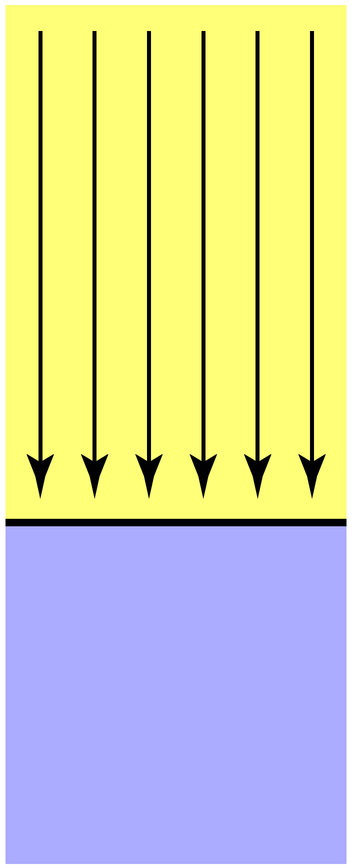
- On a *diffuse* surface, light scatters uniformly in all directions.
- No dependence on view direction.
- Many surfaces are approximately diffuse:
  - matte painted surfaces, projector screens,
  - anything that doesn't look "shiny"

# Diffuse (Lambertian) Reflection

- whiteboard

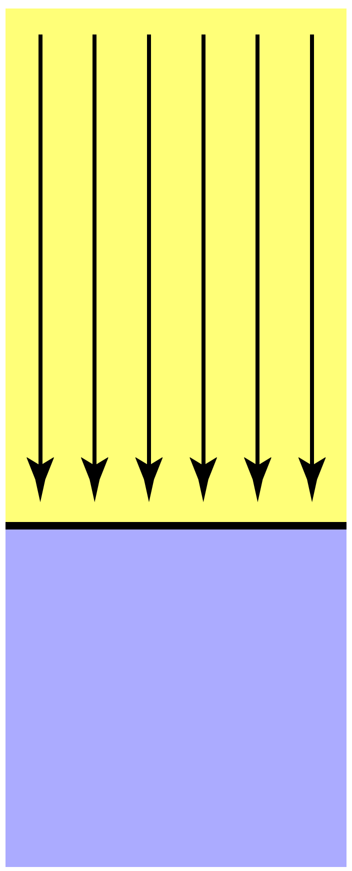
# Diffuse (Lambertian) Reflection

# Diffuse (Lambertian) Reflection

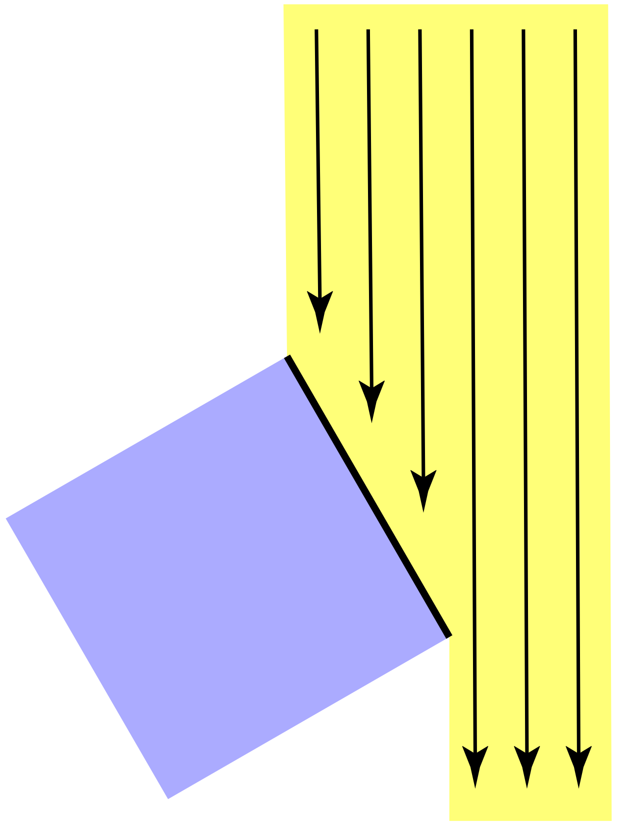


The top face of a cube receives some amount of light.

# Diffuse (Lambertian) Reflection

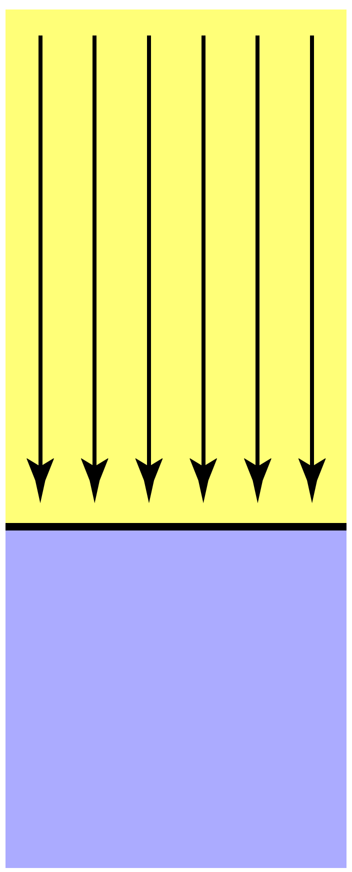


The top face of a cube receives some amount of light.

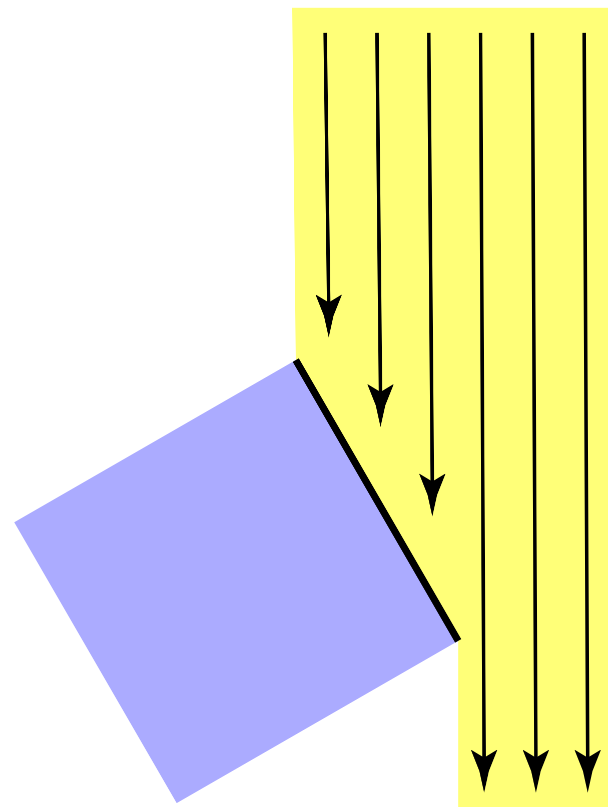


Rotated 60º, the same face receives half the light.

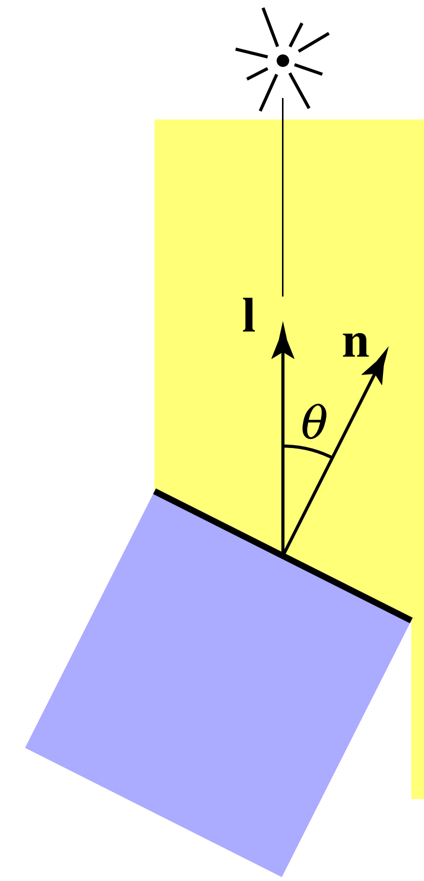
# Diffuse (Lambertian) Reflection



The top face of a cube receives some amount of light.



Rotated 60°, the same face receives half the light.




Light per unit area is proportional to $\cos\theta = \vec{n} \cdot \vec{\ell}$

# Diffuse (Lambertian) Shading

- The full model:

$$L_d = k_d I \max(\vec{n} \cdot \vec{\ell})$$

diffuse
coefficient

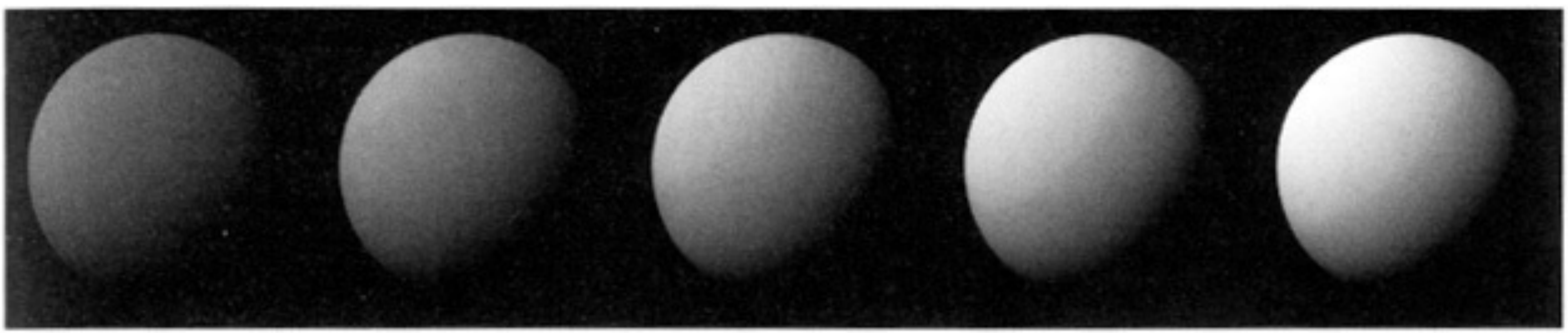
why max?

diffusely
reflected light

light intensity

# Diffuse (Lambertian) Shading

$$L_d = k_d I \max(\vec{n} \cdot \vec{\ell})$$



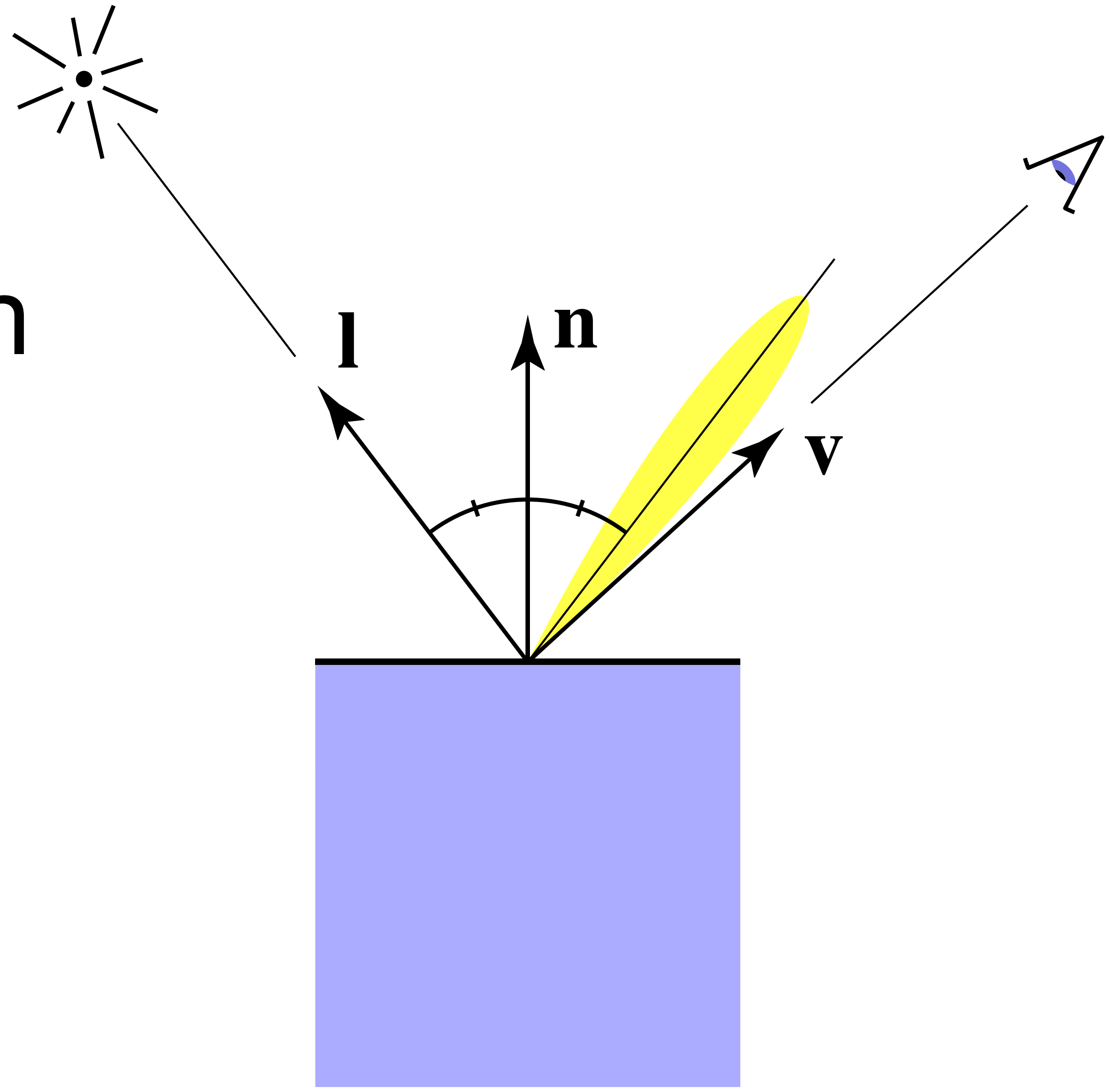
[Foley et al.]

$k_d$ ⟶

For colored objects, $k_d$ is a 3-vector of R, G, and B reflectances.

# Specular Reflection

- What about shiny surfaces?
- They appear brighter near "mirror" configuration



# Specular Reflection

- Approximation:

  half-way vector between view and light is close to the normal.

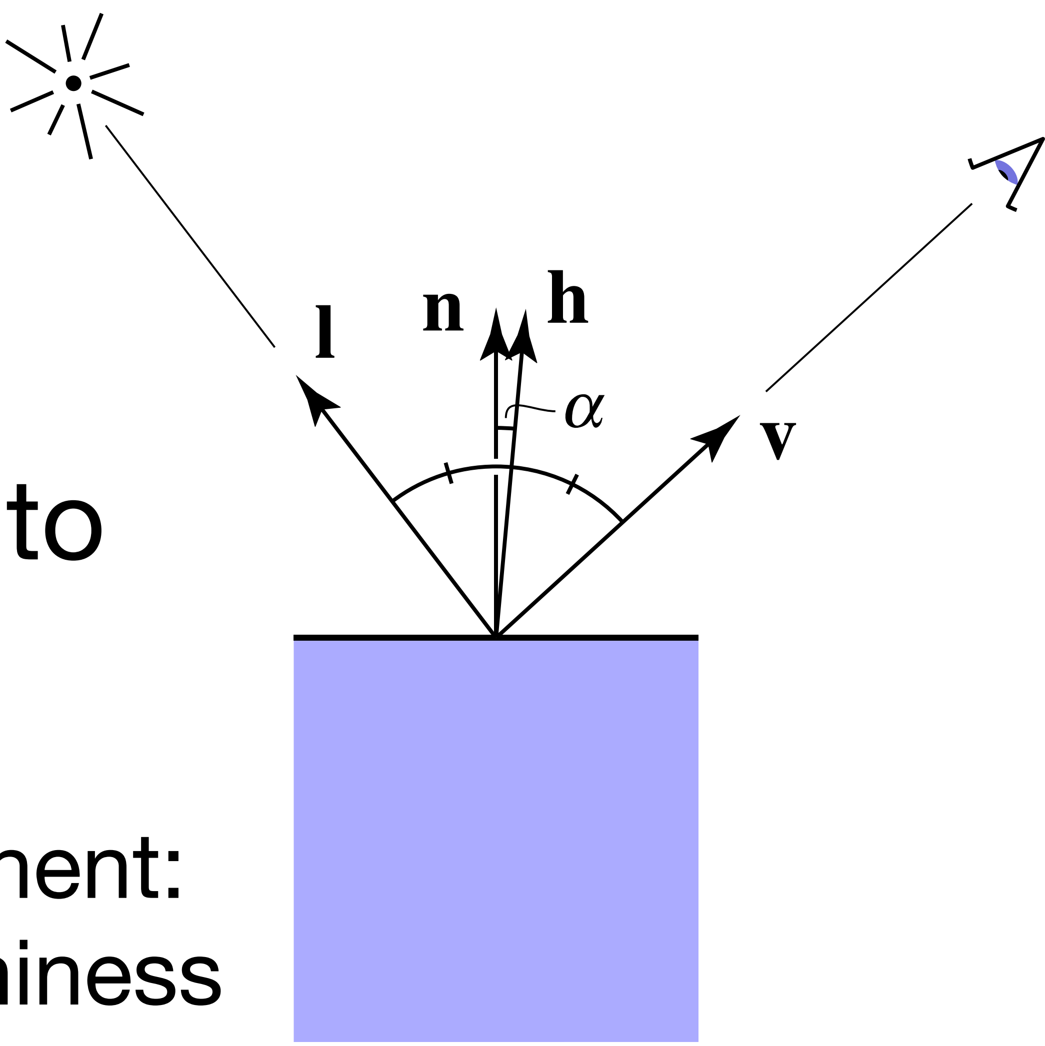
- h = bisector(**v**, **l**)

- Reflected light proportional to

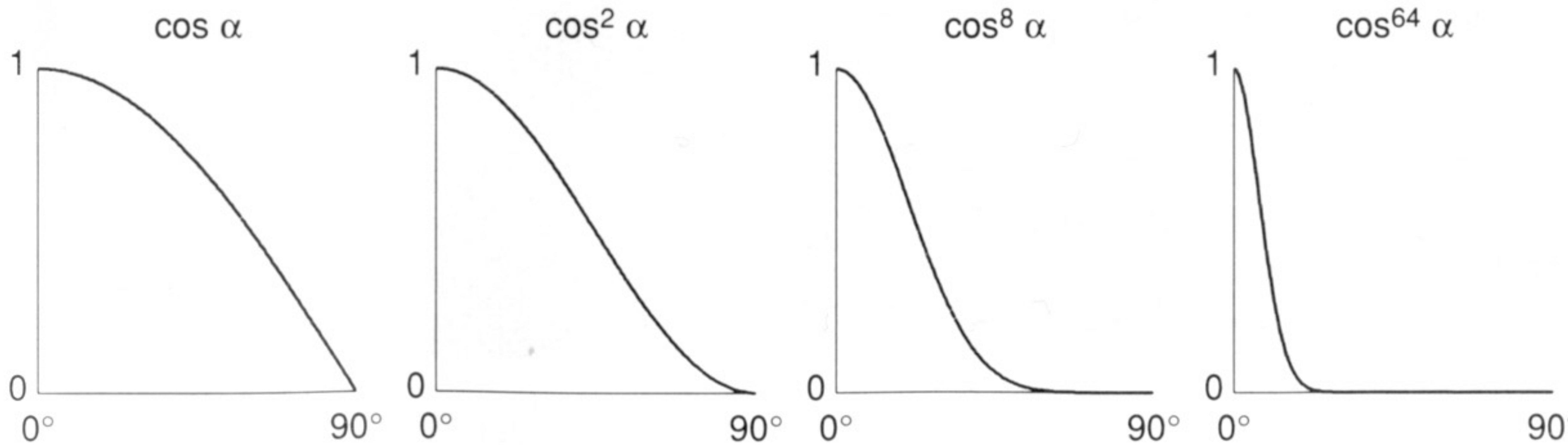
$$k_s \max(0, \vec{n} \cdot \vec{h})^p$$

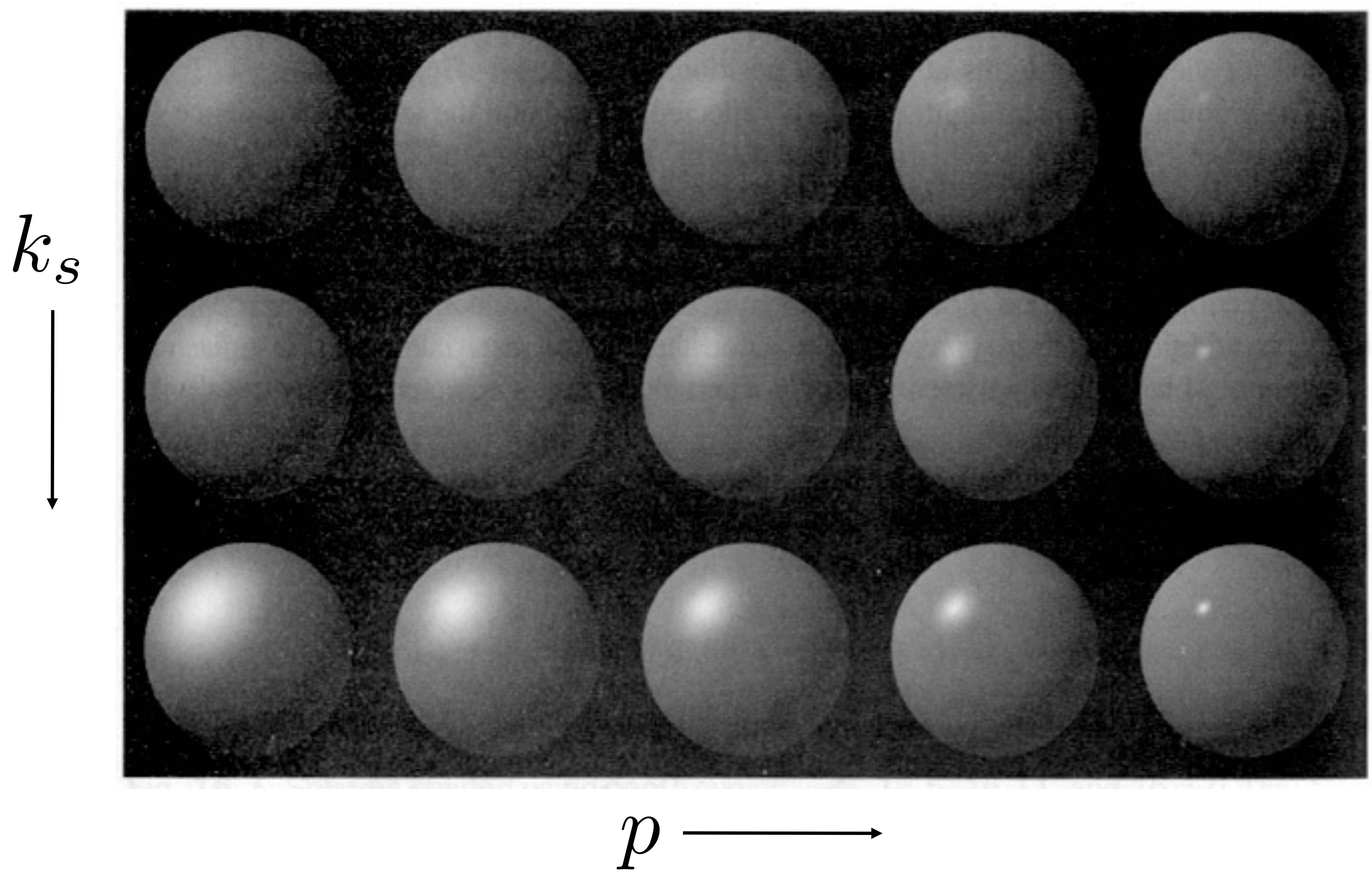
specular coefficient: determines strength of specularity term

specular exponent: determines shininess



# Effect of *p*




$k_s$
$p$