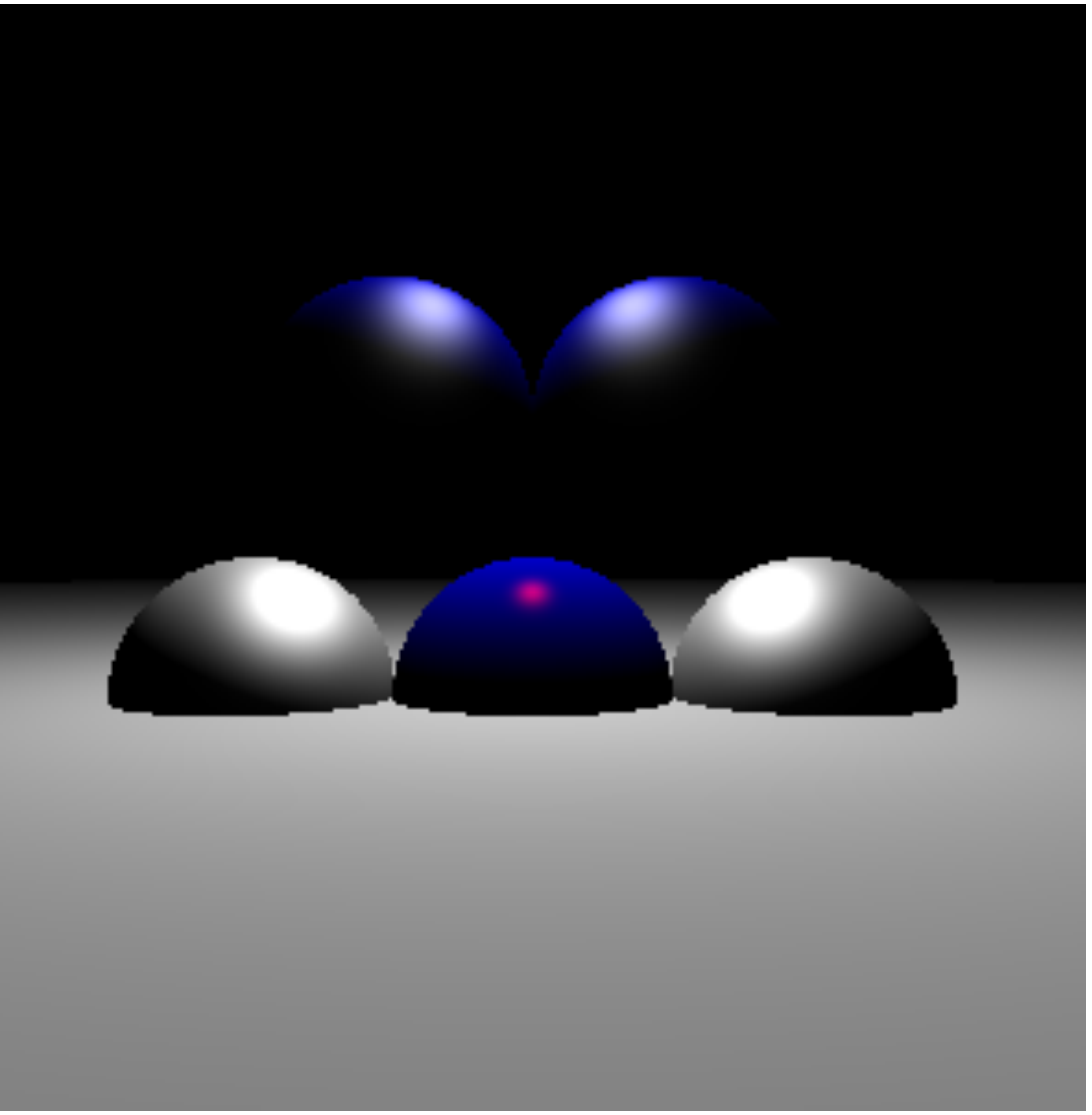
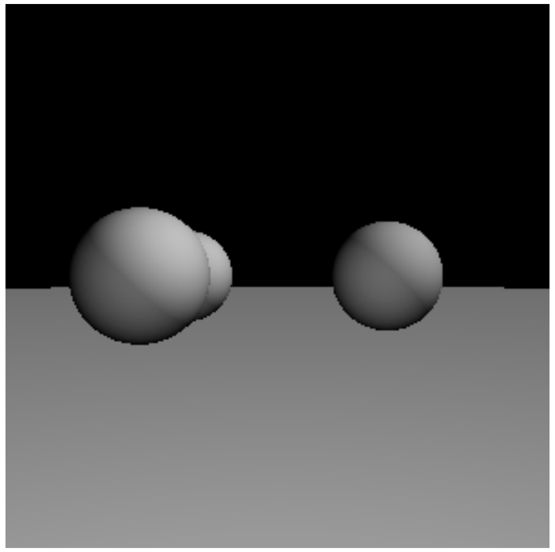
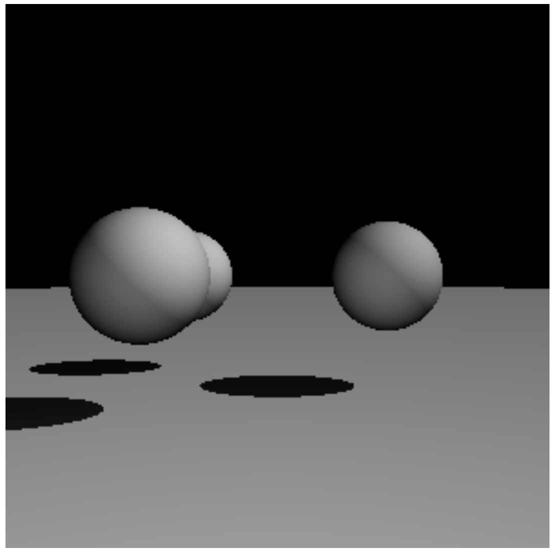
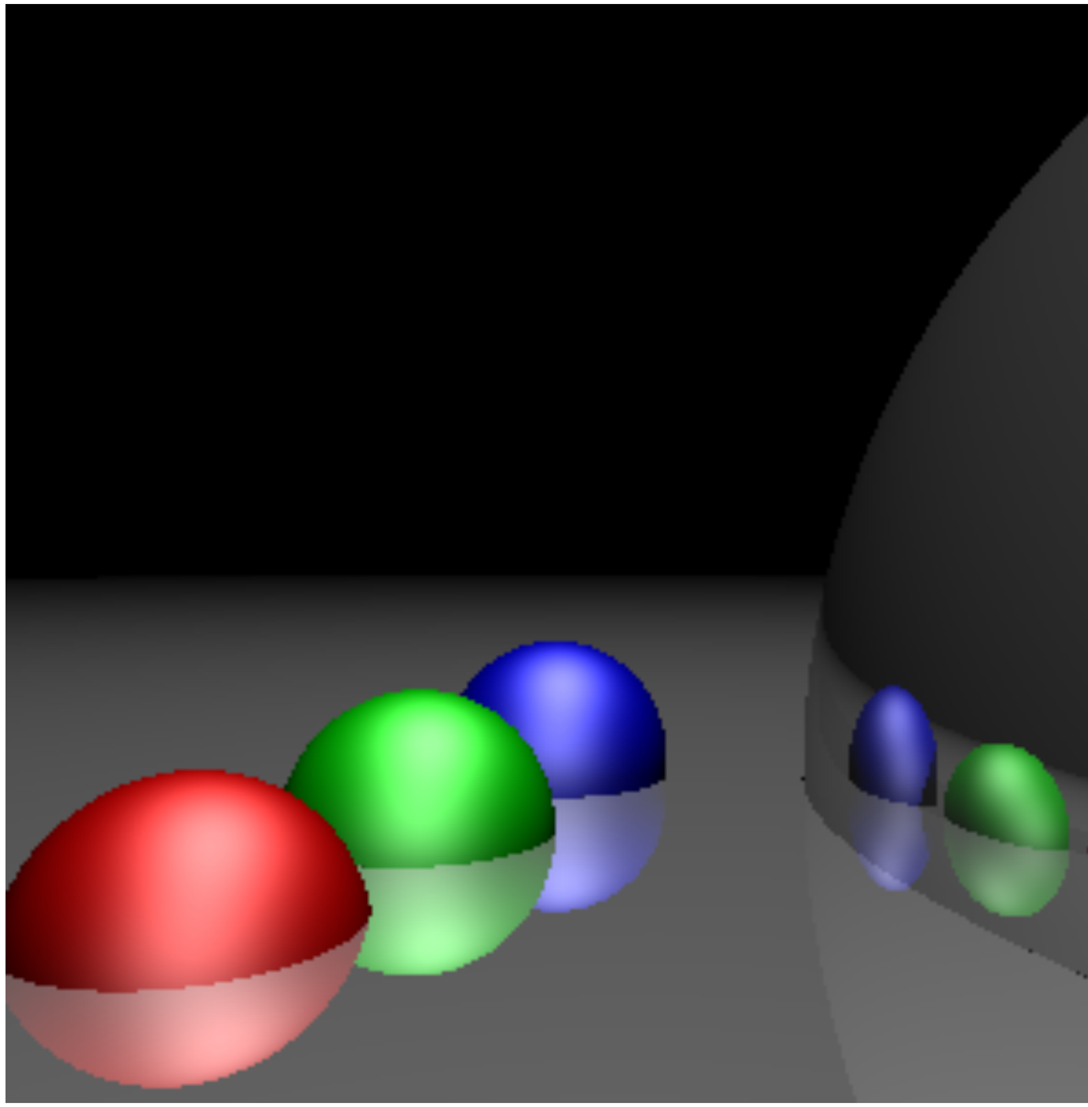








Computer Graphics

Lecture 9

Mirror Reflection

Shadows

Triangles

Announcements

Announcements

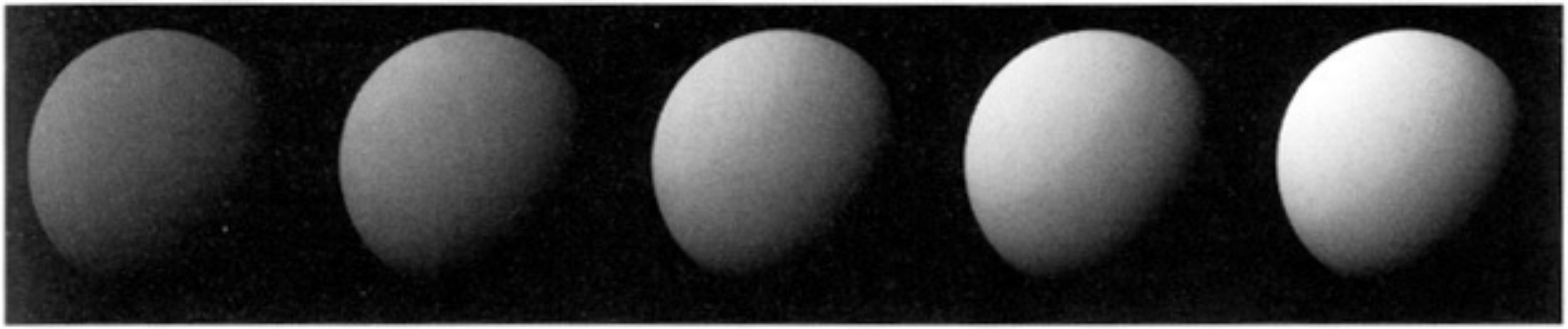
- HW1: If you aren't familiar with latex or some other typesetting system, talk to me.

Announcements

- HW1: If you aren't familiar with latex or some other typesetting system, talk to me.
- Use Piazza!

Diffuse (Lambertian) Shading

$$L_d = k_d I \max(0, \vec{n} \cdot \vec{\ell})$$



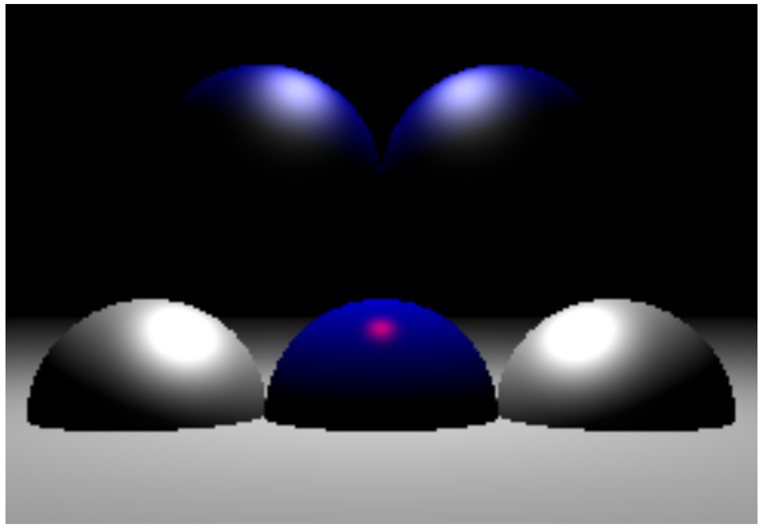
[Foley et al.]

$$k_d \longrightarrow$$

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

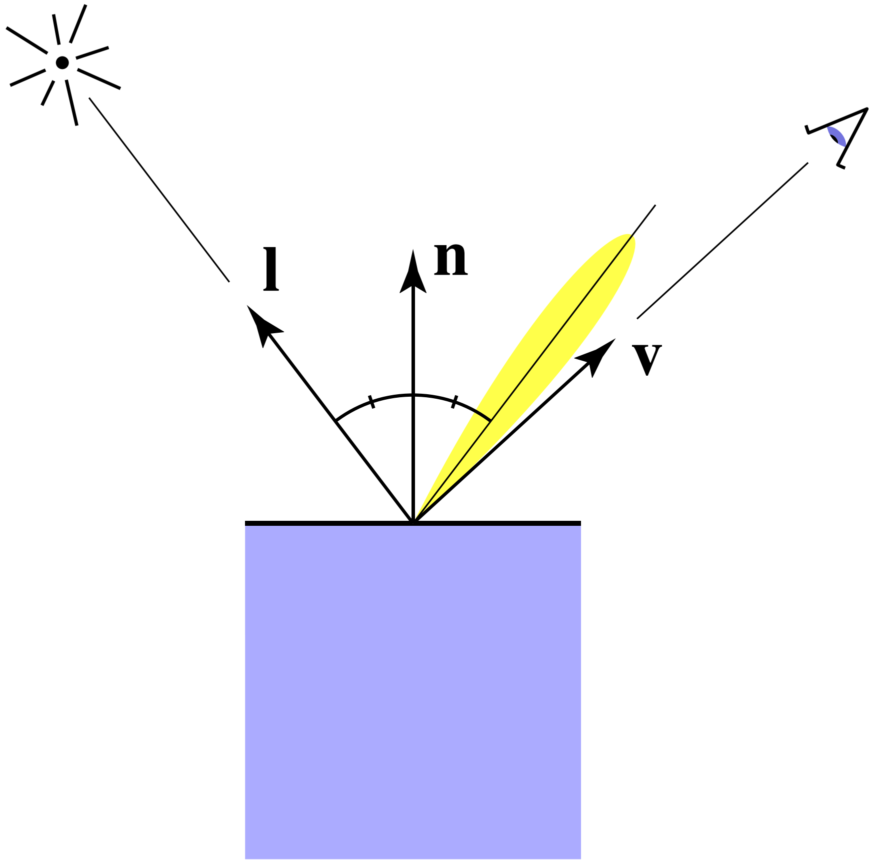
Specular Reflection

- What about non-mirror shiny surfaces?



- They appear brighter *near* "mirror" configuration

- Phong reflection: specular reflection is a function of angle between **r** and **v**.



Specular Reflection

- Blinn-Phong: specular reflection is a function of angle between **half-way vector** between view and light and the **normal**.

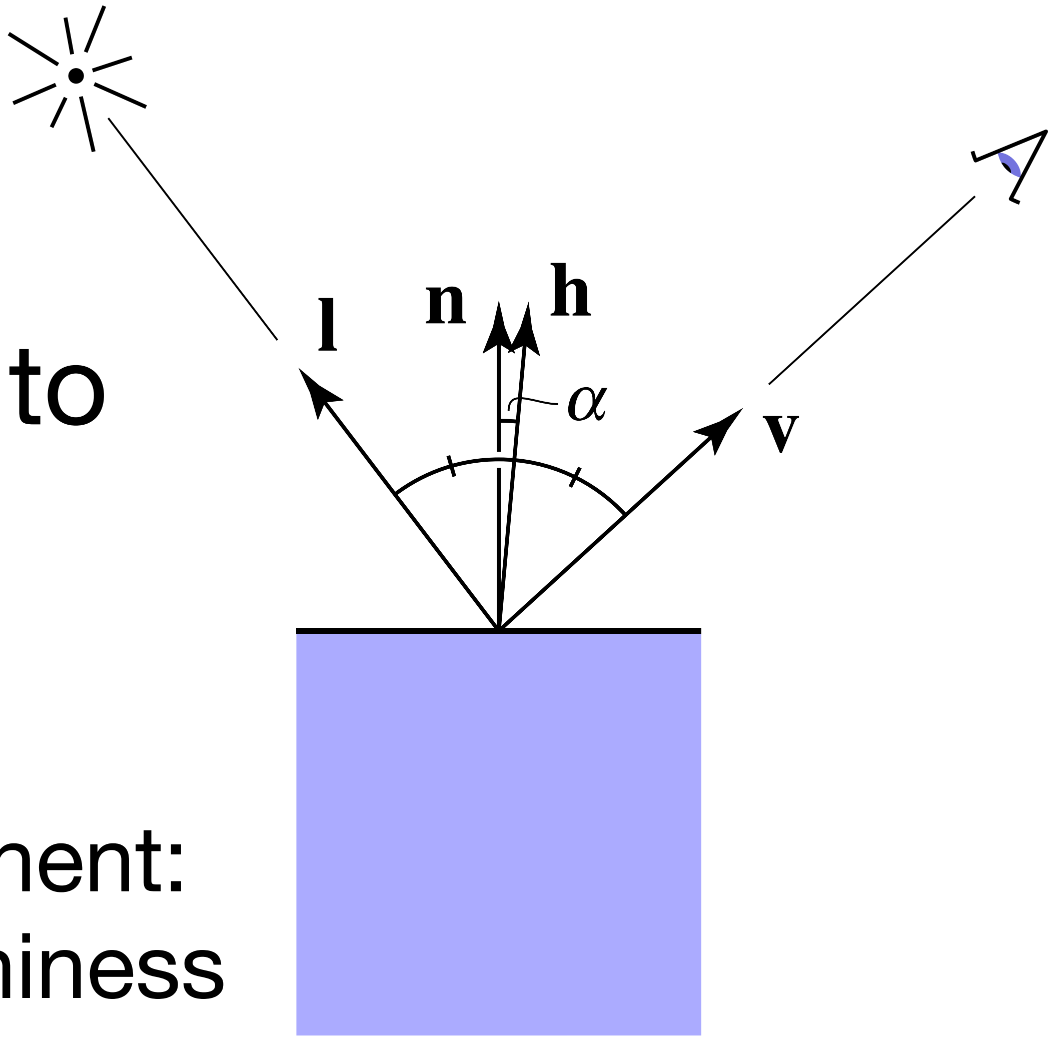
- h = bisector(**v**, **l**)

- 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



Computing h

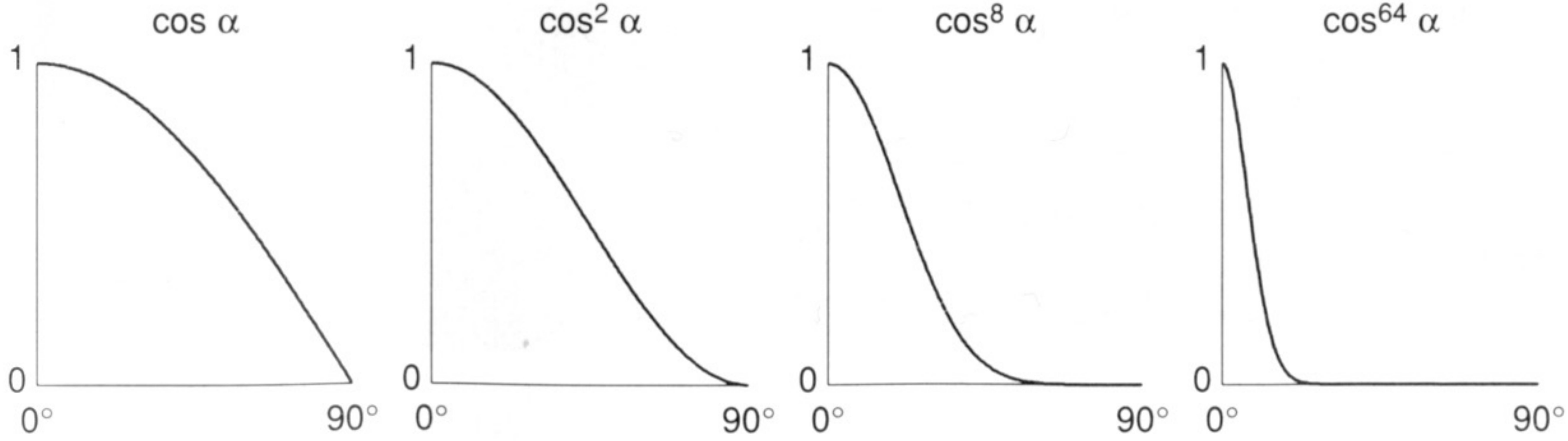
- whiteboard

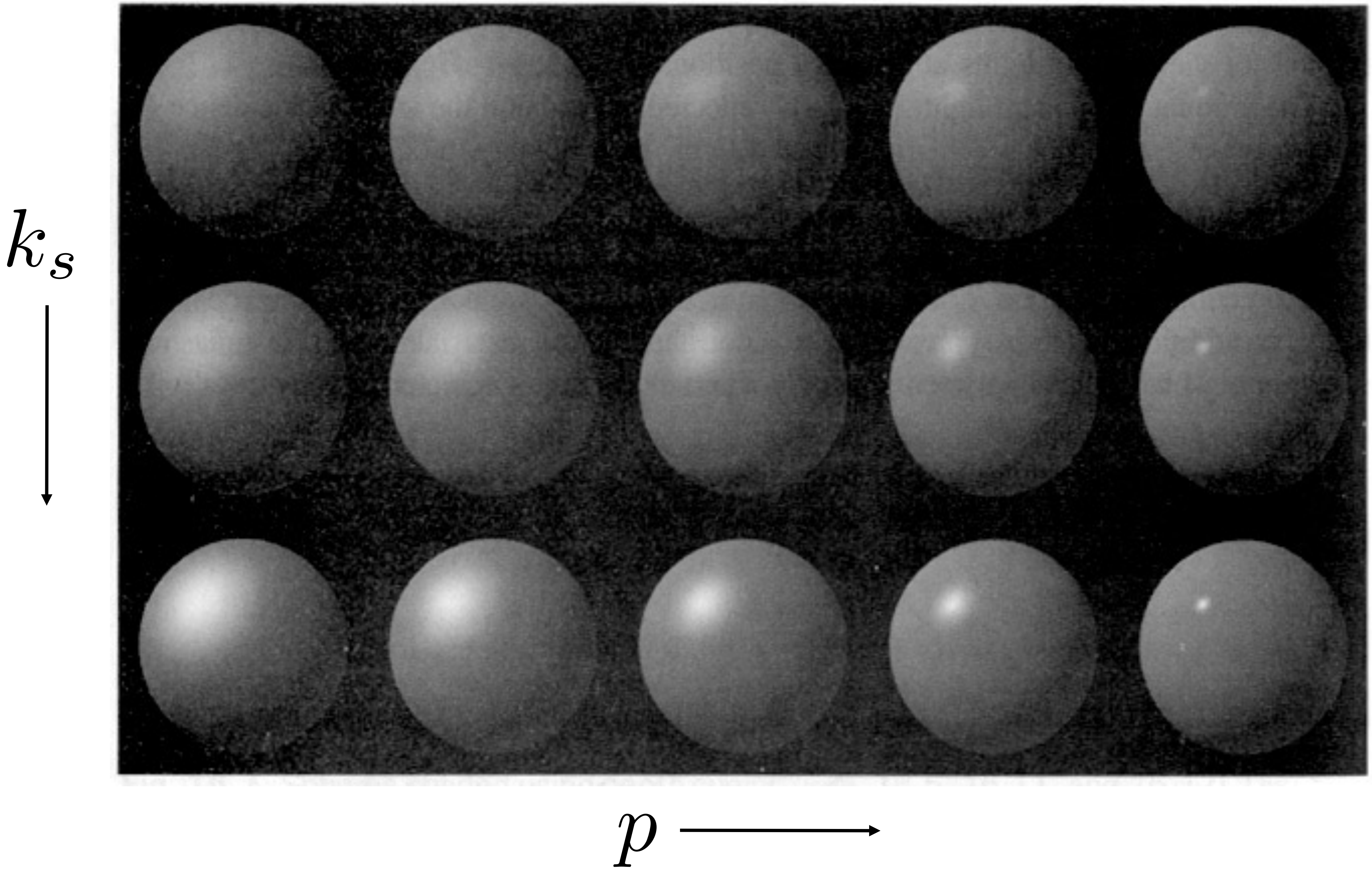
Computing h

- whiteboard

$$\mathrm{bisector}(\vec{v}, \vec{\ell}) = \frac{\vec{v} + \vec{\ell}}{||\vec{v} + \vec{\ell}||}$$

Effect of p




k_s
p

Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse *and* specular components, so we'll combine the two:

$$\begin{aligned} L &= L_d + L_s \\ &= k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p \end{aligned}$$

Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse *and* specular components, so we'll combine the two:

$$
\begin{aligned}
\overset{\text{light reflected}}{L} &= \overset{\substack{\text{diffuse}\\\text{reflection}}}{L_d} + \overset{\substack{\text{specular}\\\text{reflection}}}{L_s} \\
&= k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p
\end{aligned}
$$

Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse *and* specular components, so we'll combine the two:

light reflected

diffuse reflection

specular reflection

$$
\begin{aligned}
L &= L_d + L_s \\
&= k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p
\end{aligned}
$$

diffuse coefficient (surface brightness and color)

light intensity

light direction

normal

Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse *and* specular components, so we'll combine the two:

light reflected

diffuse reflection

specular reflection

$$L = L_d + L_s$$

specular exponent (*sharpness* of specularity)

$$= k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p$$

diffuse coefficient (surface brightness and color)

light intensity

normal

light direction

specular coefficient (*strength* [and color] of specularity)

half-vector between **l** and **v**

Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse *and* specular components, so we'll combine the two:

light reflected

diffuse reflection

specular reflection

$$L = L_d + L_s$$

specular exponent (*sharpness* of specularity)

$$= k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p$$

diffuse coefficient (surface brightness and color)

light intensity

normal

light direction

specular coefficient (*strength* [and color] of specularity)

half-vector between **l** and **v**

In code: `function shade_light(light, hitrec,...)`

What if there are multiple lights?

Light is additive - add them together:

$$L = \sum_{i=1}^{\#\ \mathrm{lights}} k_d I \max(0, \vec{n} \cdot \vec{l_i}) + k_s I \max(0, \vec{n} \cdot \vec{h_i})^p$$

In code:

What if there are multiple lights?

Light is additive - add them together:

$$L = \sum_{i=1}^{\#\ \text{lights}} k_d I \max(0, \vec{n} \cdot \vec{l_i}) + k_s I \max(0, \vec{n} \cdot \vec{h_i})^p$$

In code:

```
function determine_color(hitrec, ray, scene, ...):
```

What if there are multiple lights?

Light is additive - add them together:

$$L = \sum_{i=1}^{\#\text{ lights}} k_d I \max(0, \vec{n} \cdot \vec{l_i}) + k_s I \max(0, \vec{n} \cdot \vec{h_i})^p$$

In code:

```
function determine_color(hitrec, ray, scene, ...):
    color = black
```

What if there are multiple lights?

Light is additive - add them together:

$$L = \sum_{i=1}^{\#\text{ lights}} k_d I \max(0, \vec{n} \cdot \vec{l_i}) + k_s I \max(0, \vec{n} \cdot \vec{h_i})^p$$

In code:

```
function determine_color(hitrec, ray, scene, ...):
    color = black
    for light in scene.lights:
```

What if there are multiple lights?

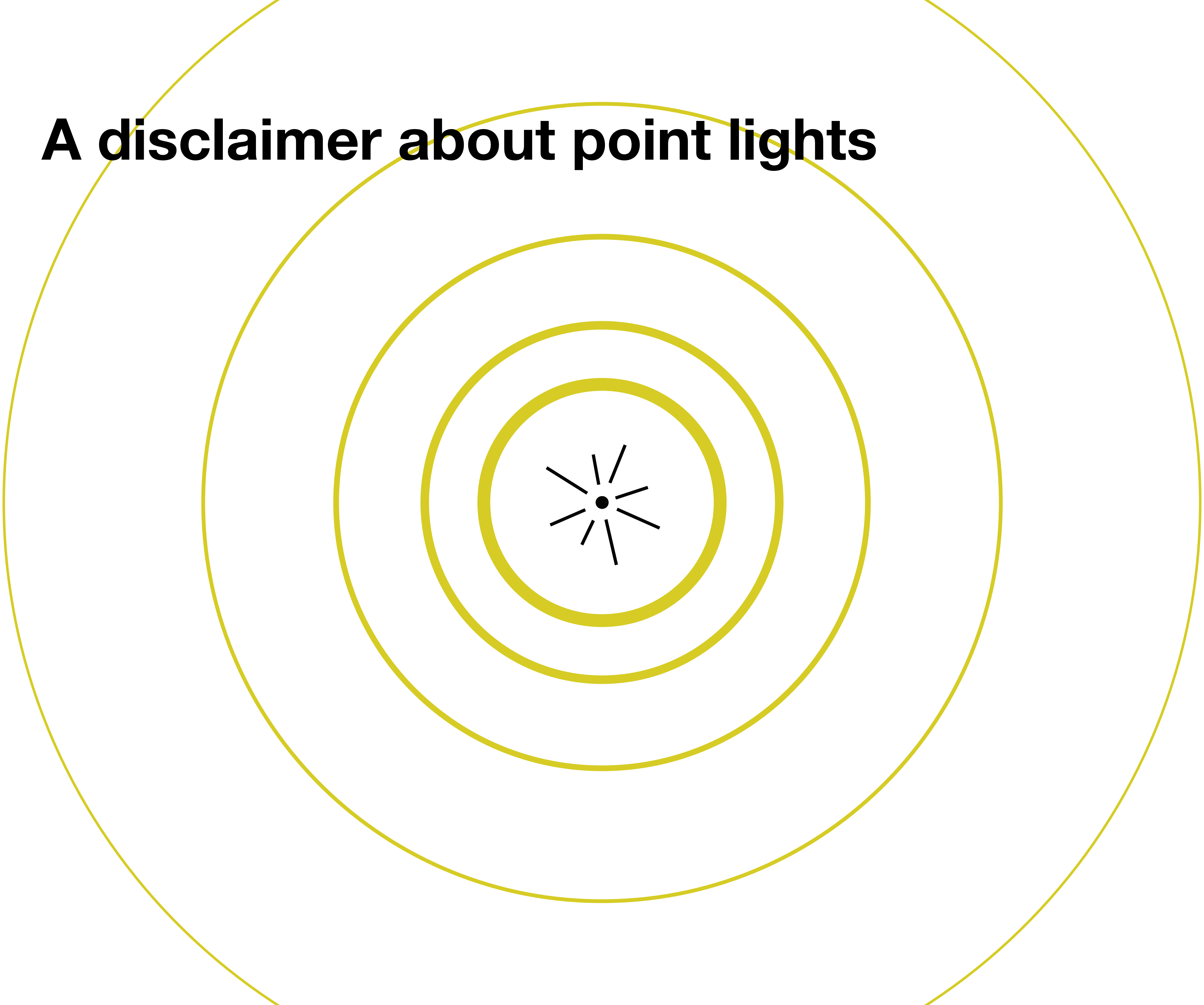
Light is additive - add them together:

$$L = \sum_{i=1}^{\#\text{ lights}} k_d I \max(0, \vec{n} \cdot \vec{l_i}) + k_s I \max(0, \vec{n} \cdot \vec{h_i})^p$$

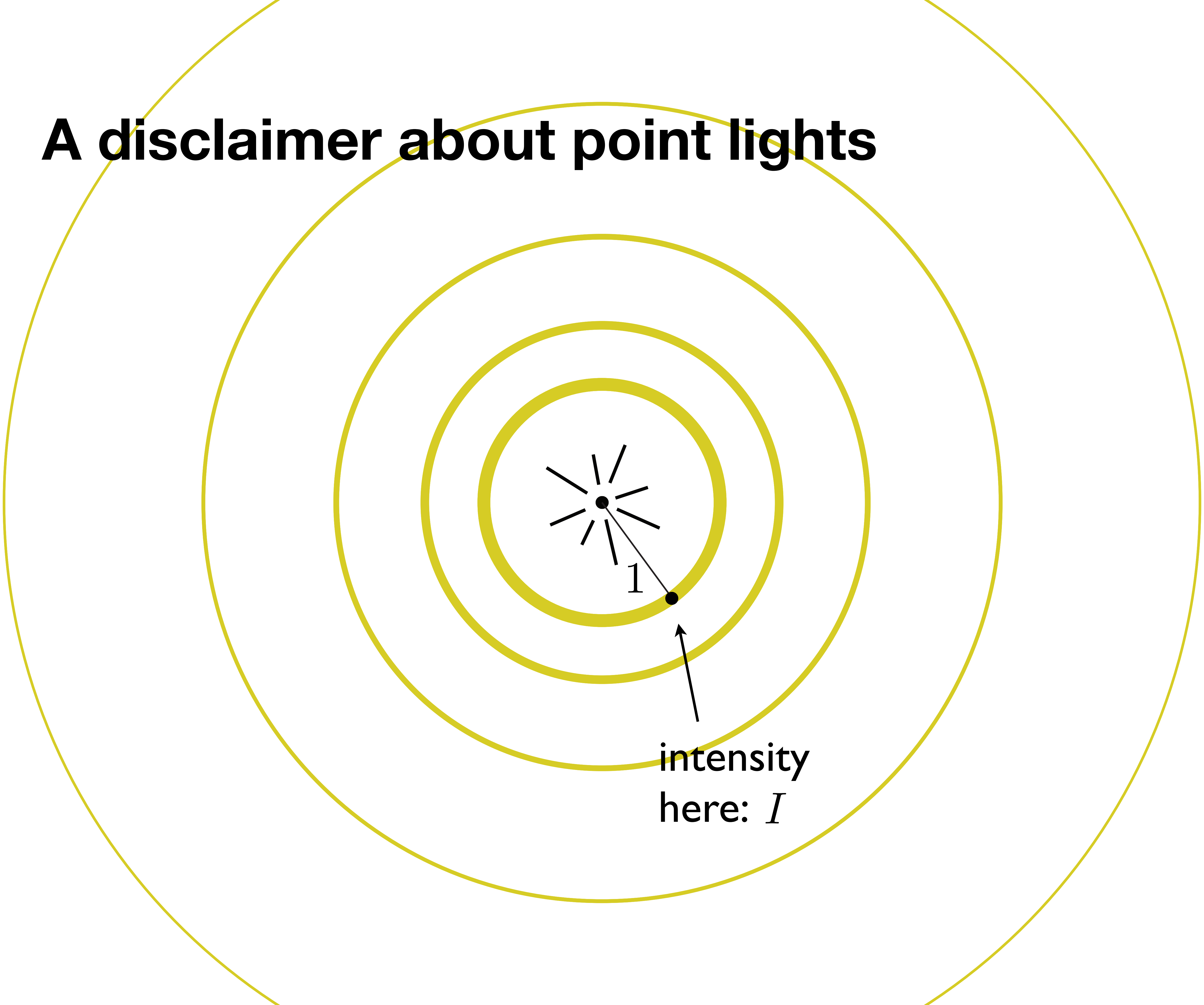
In code:

```
function determine_color(hitrec, ray, scene, ...):
    color = black
    for light in scene.lights:
        color += shade_light(light, hitrec, ...)
```

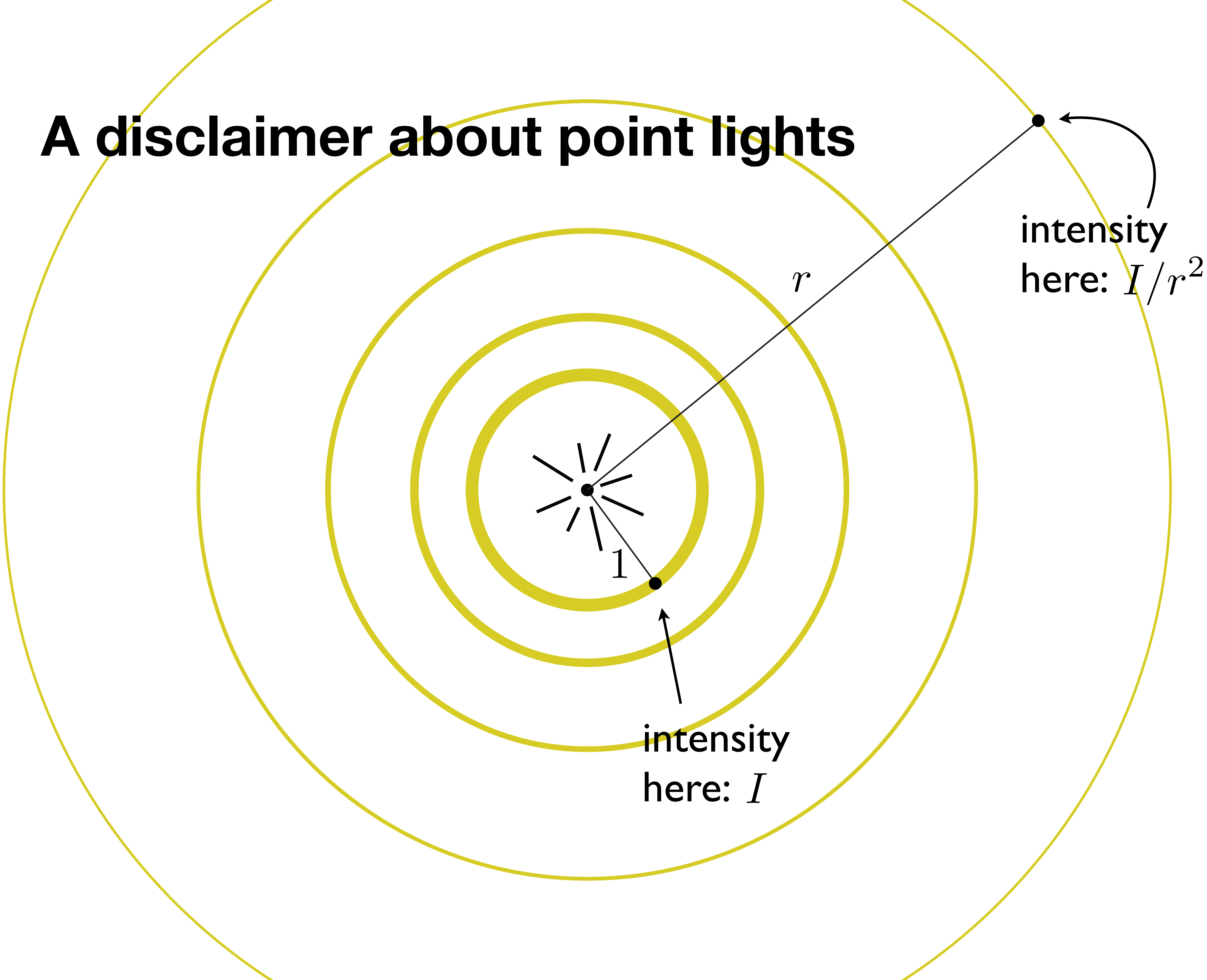
A disclaimer about point lights



A disclaimer about point lights



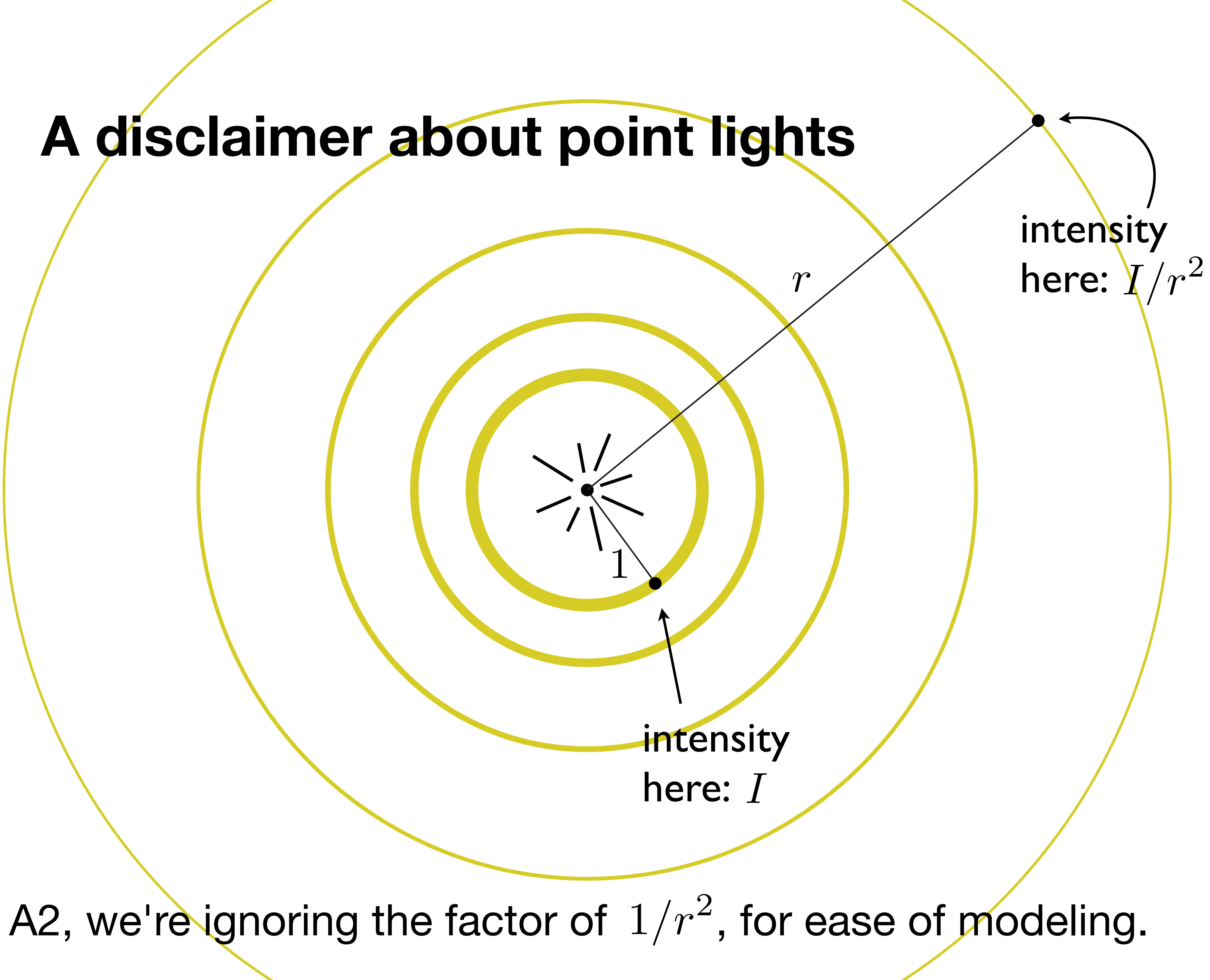
A disclaimer about point lights



intensity
here: I/r^2

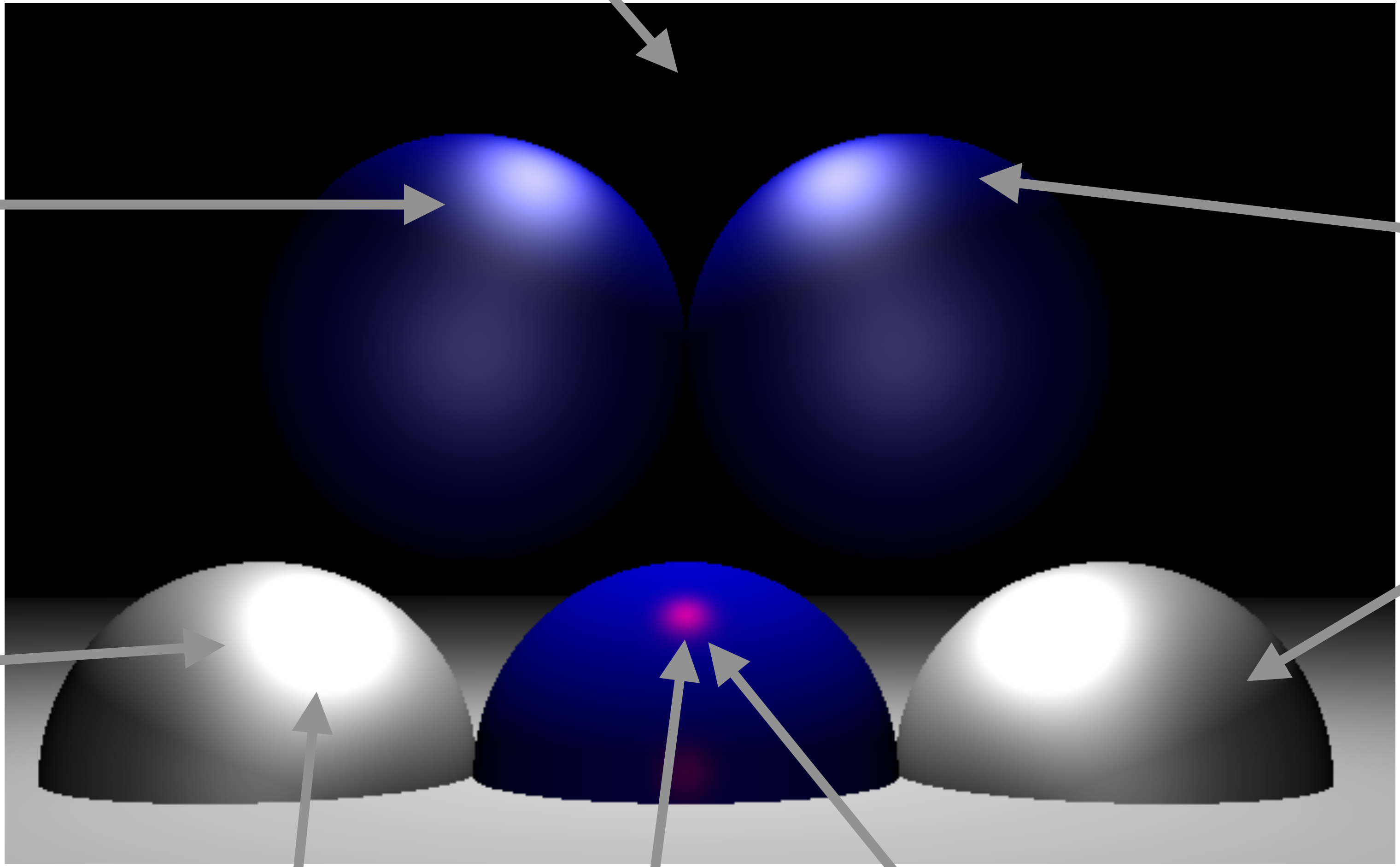
intensity
here: I

A disclaimer about point lights



In A2, we're ignoring the factor of $1/r^2$, for ease of modeling.

Our images so far:



Point light

smaller k_s

blue k_d

gray k_d

larger k_s

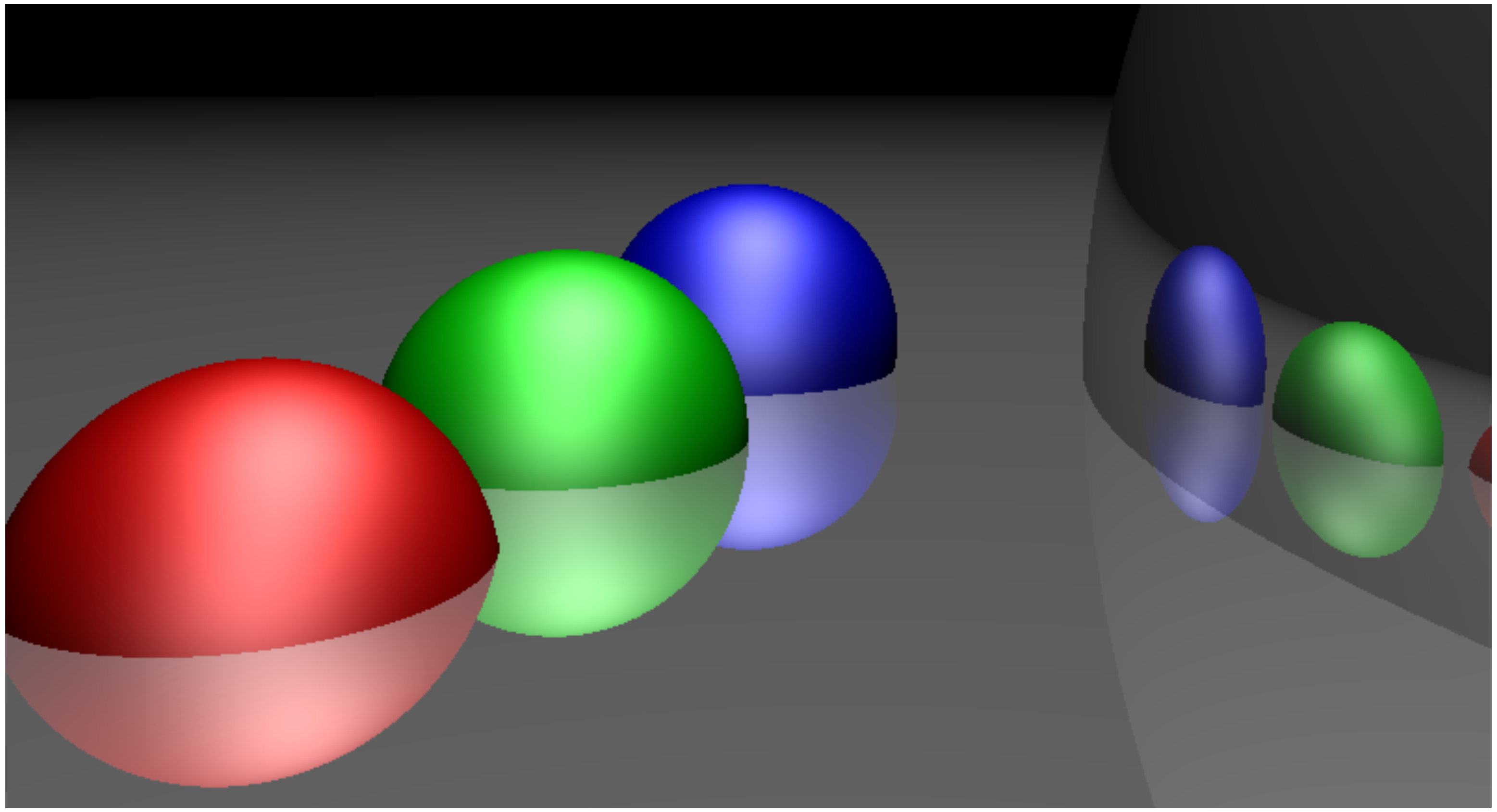
smaller p

larger p

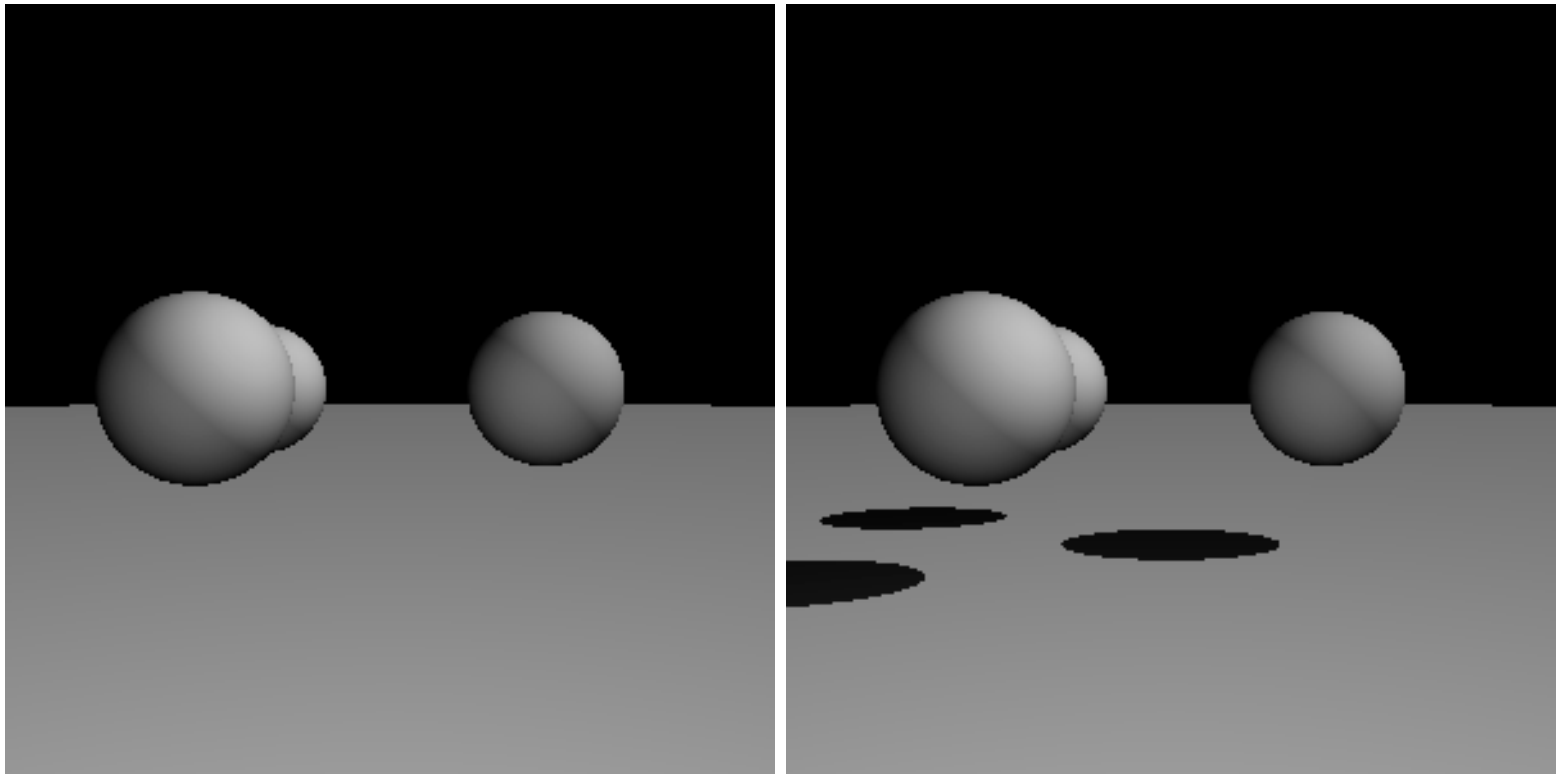
red k_s

What's next?

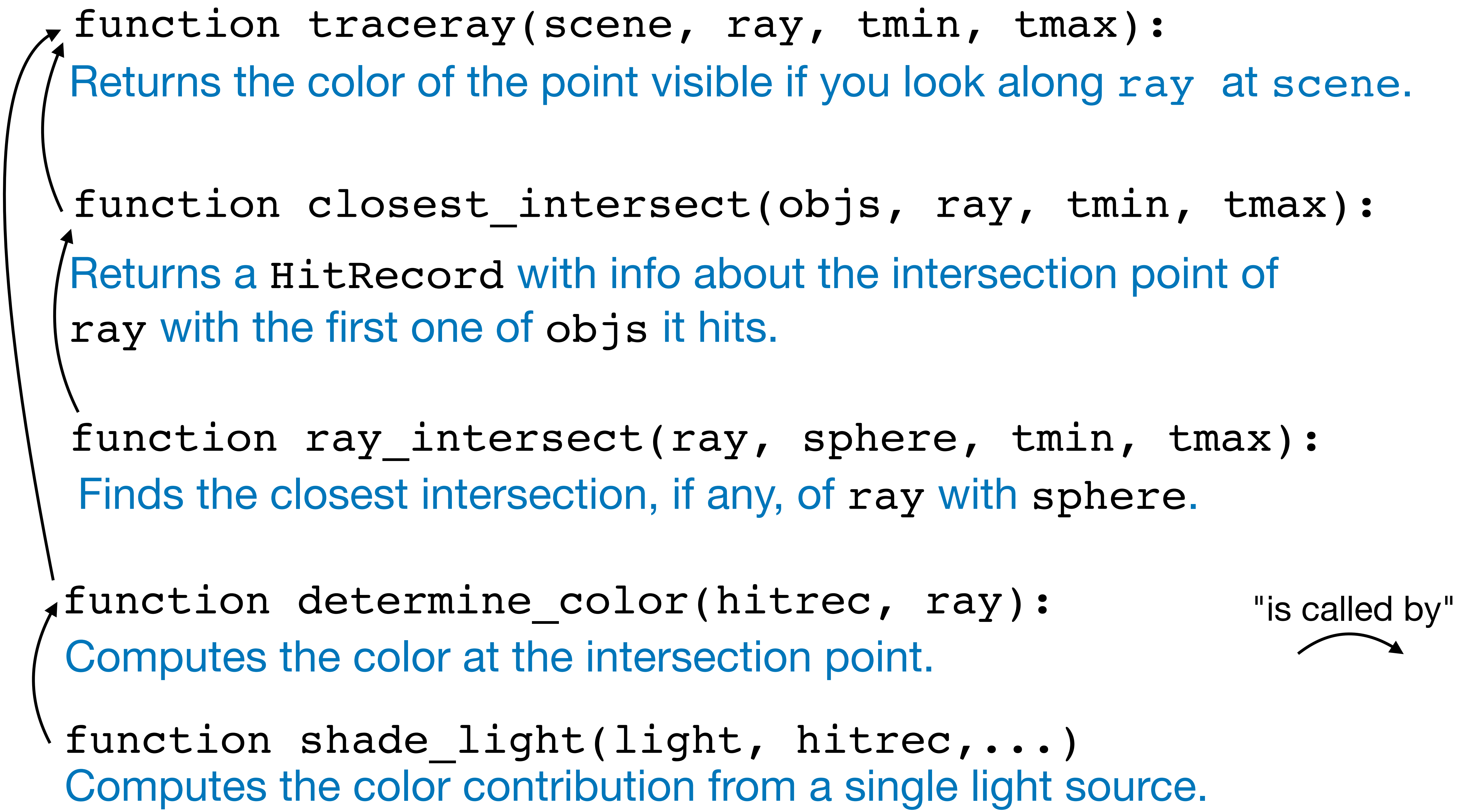
Mirror-reflective surfaces



Shadows



A2: Code Inventory



`function traceray(scene, ray, tmin, tmax):`

Returns the color of the point visible if you look along `ray` at `scene`.

`function closest_intersect(objs, ray, tmin, tmax):`

Returns a `HitRecord` with info about the intersection point of `ray` with the first one of `objs` it hits.

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

Finds the closest intersection, if any, of `ray` with `sphere`.

`function determine_color(hitrec, ray):`

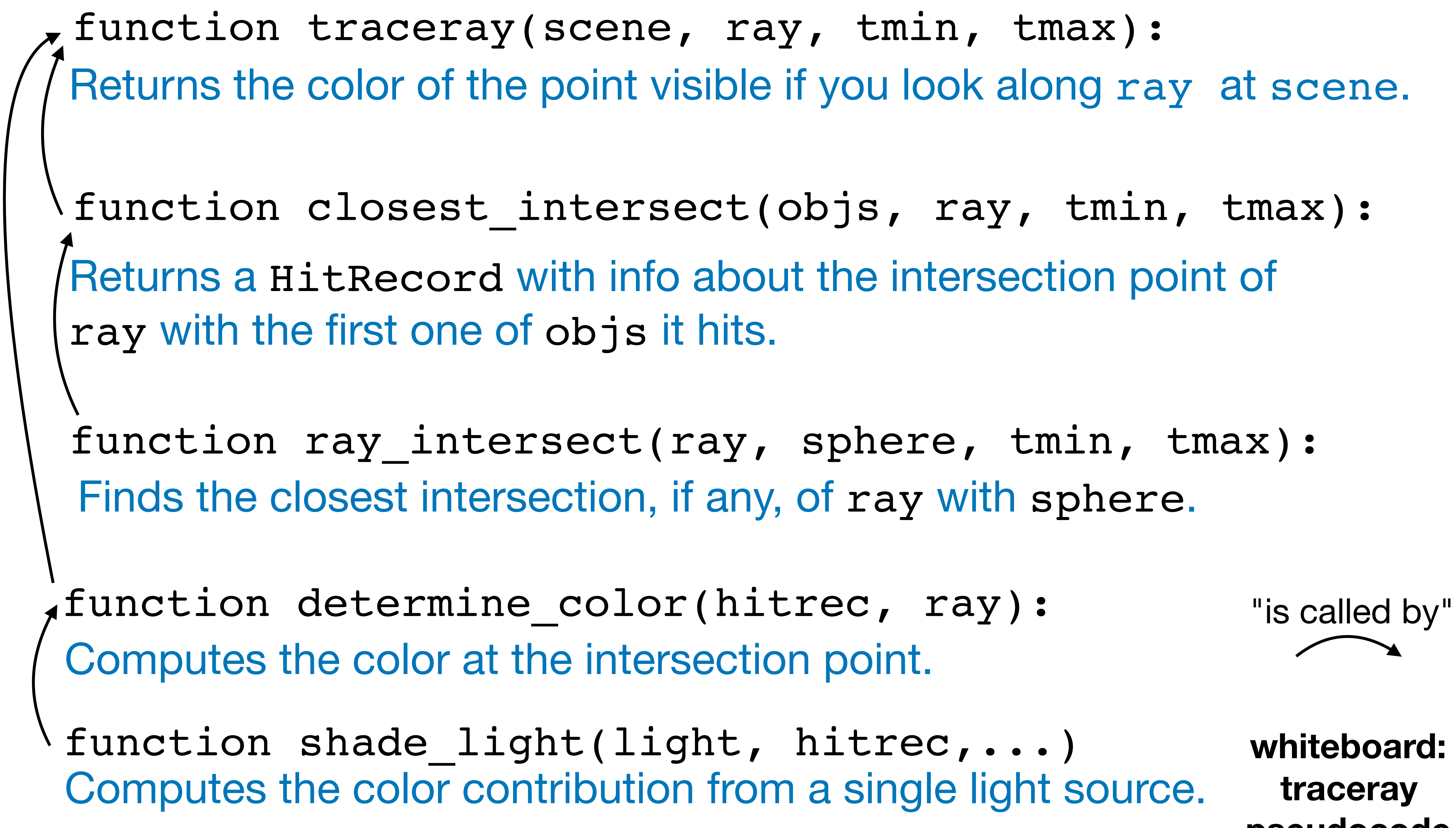
Computes the color at the intersection point.

`function shade_light(light, hitrec,...)`

Computes the color contribution from a single light source.

"is called by"

A2: Code Inventory



```
function traceray(scene, ray, tmin, tmax):
```

Returns the color of the point visible if you look along `ray` at `scene`.

```
function closest_intersect(objs, ray, tmin, tmax):
```

Returns a `HitRecord` with info about the intersection point of `ray` with the first one of `objs` it hits.

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

Finds the closest intersection, if any, of `ray` with `sphere`.

```
function determine_color(hitrec, ray):
```

Computes the color at the intersection point.

```
function shade_light(light, hitrec,...)
```

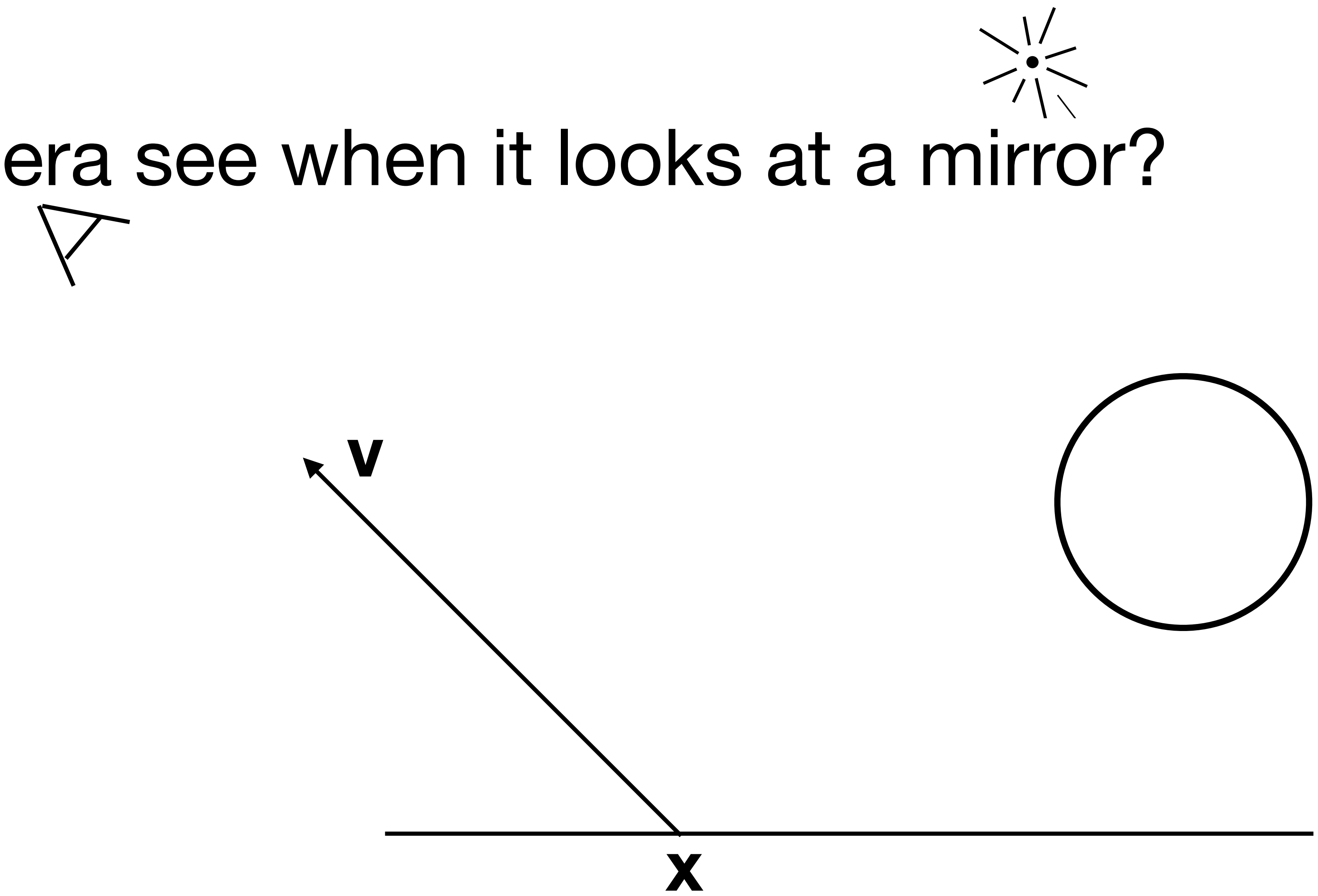
Computes the color contribution from a single light source.

"is called by"

whiteboard:
traceray
pseudocode

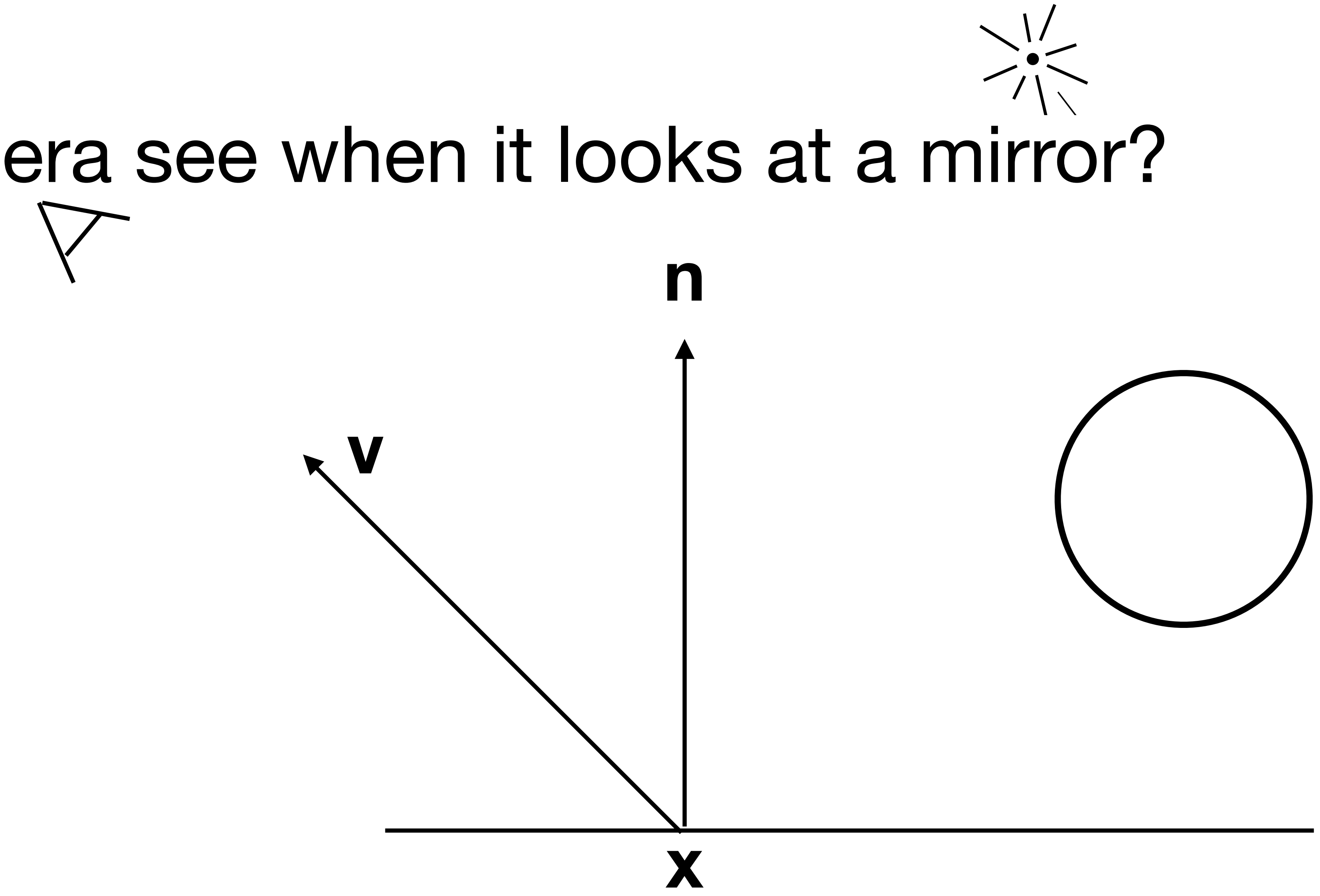
Mirror Reflection

What does a camera see when it looks at a mirror?

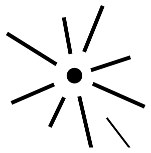


Mirror Reflection

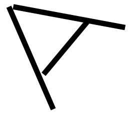
What does a camera see when it looks at a mirror?

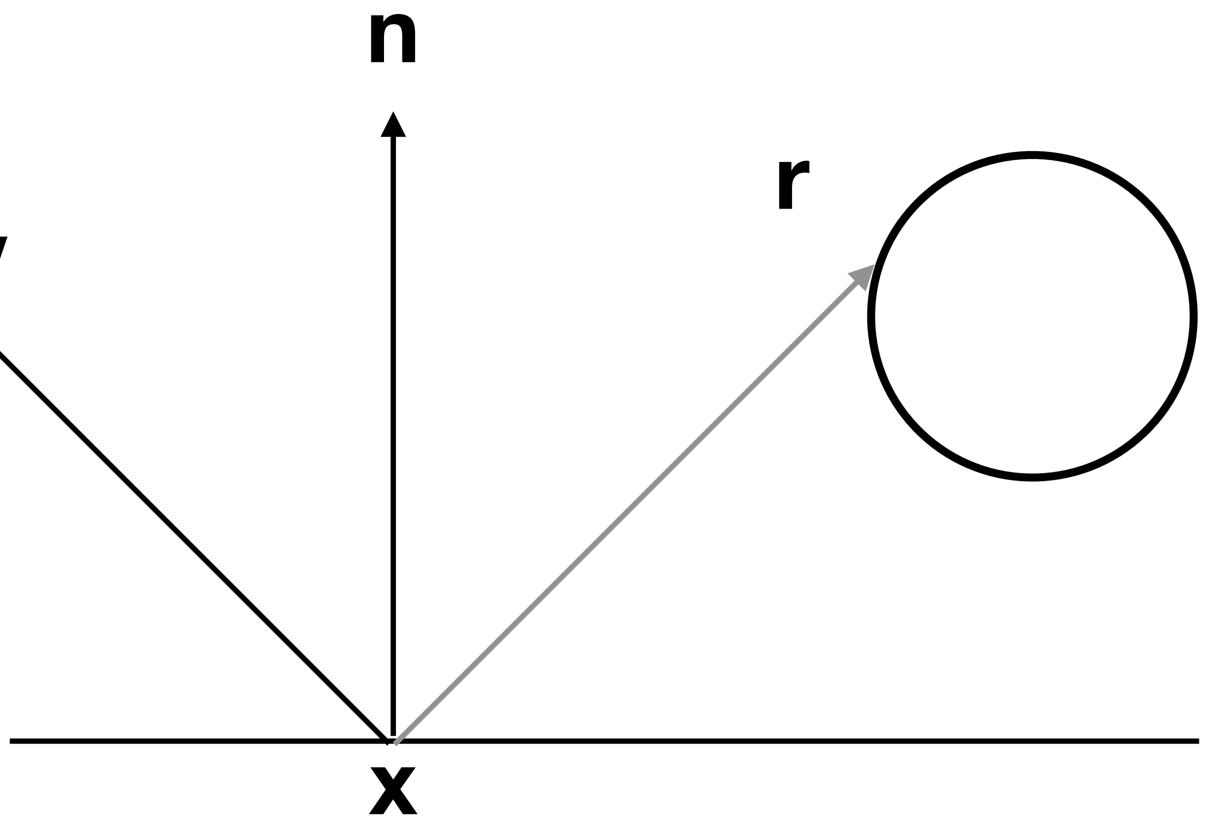


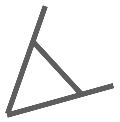
Mirror Reflection



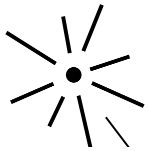
What does a camera see when it looks at a mirror?



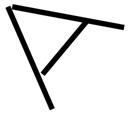


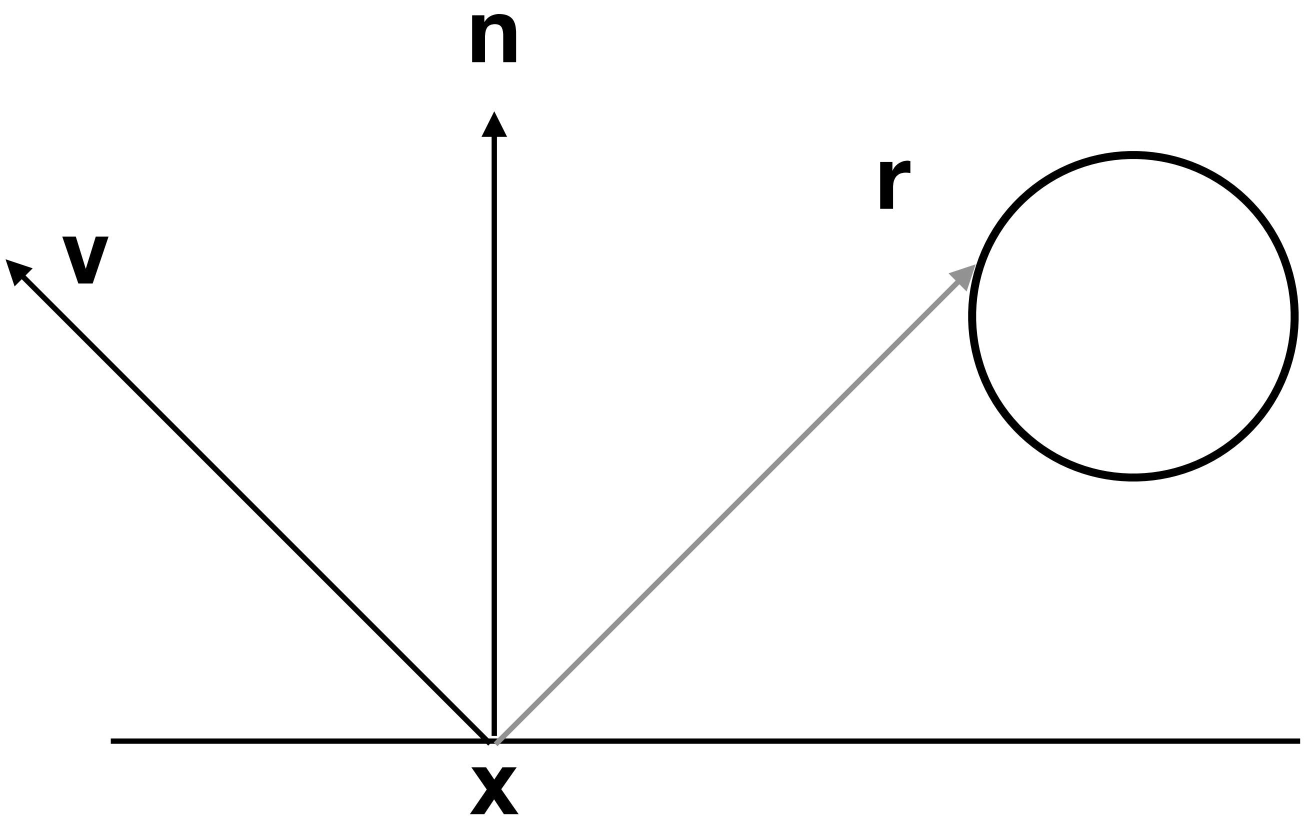
Hint: 

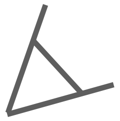
Mirror Reflection



What does a camera see when it looks at a mirror?

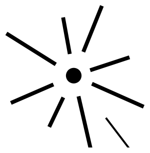




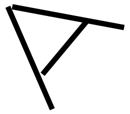
Hint: 

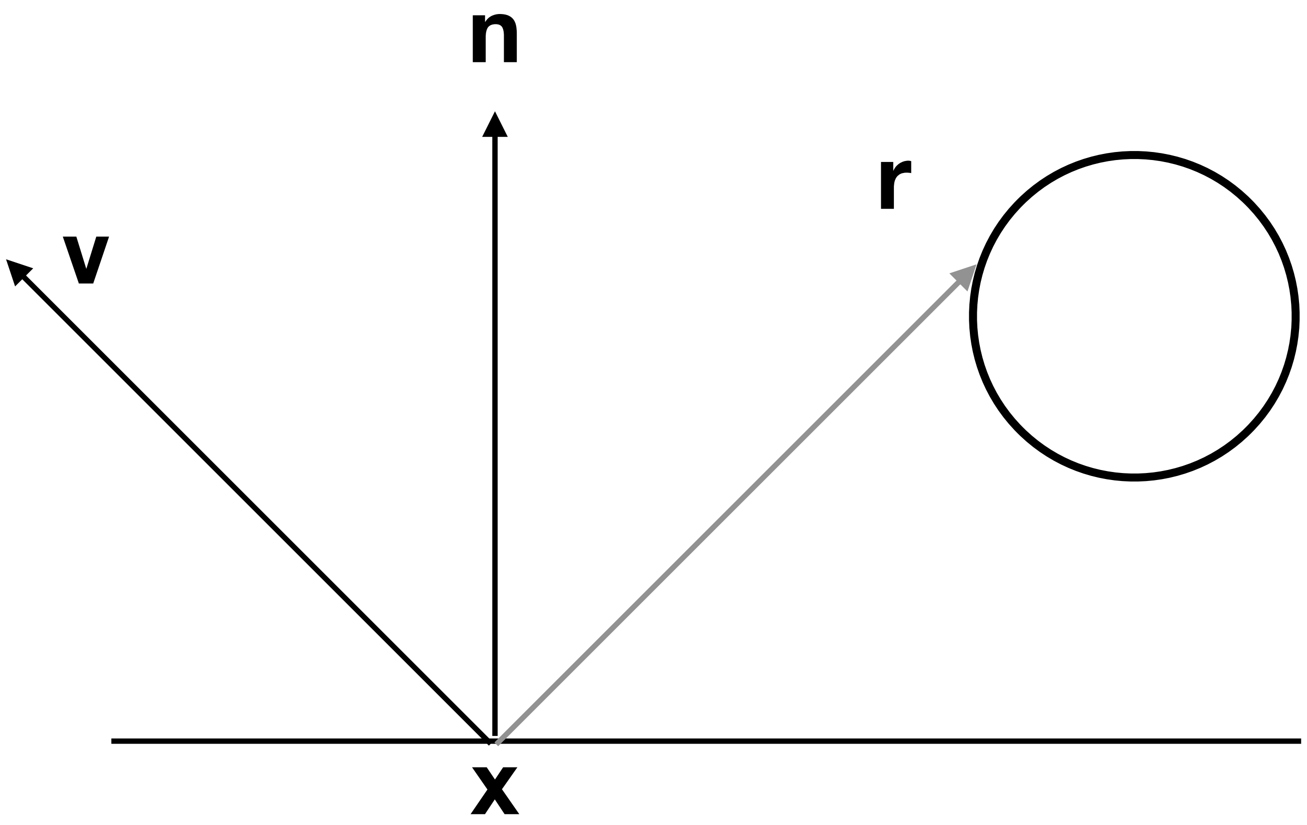
Can we do this using the tools we already have?

Mirror Reflection



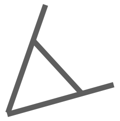
What does a camera see when it looks at a mirror?





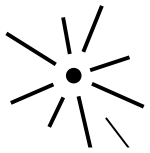
From last time:

$$\vec{r} = -\vec{v} + 2(\vec{v} \cdot \vec{n})\vec{n}$$

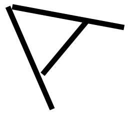
Hint: 

Can we do this using the tools we already have?

Mirror Reflection

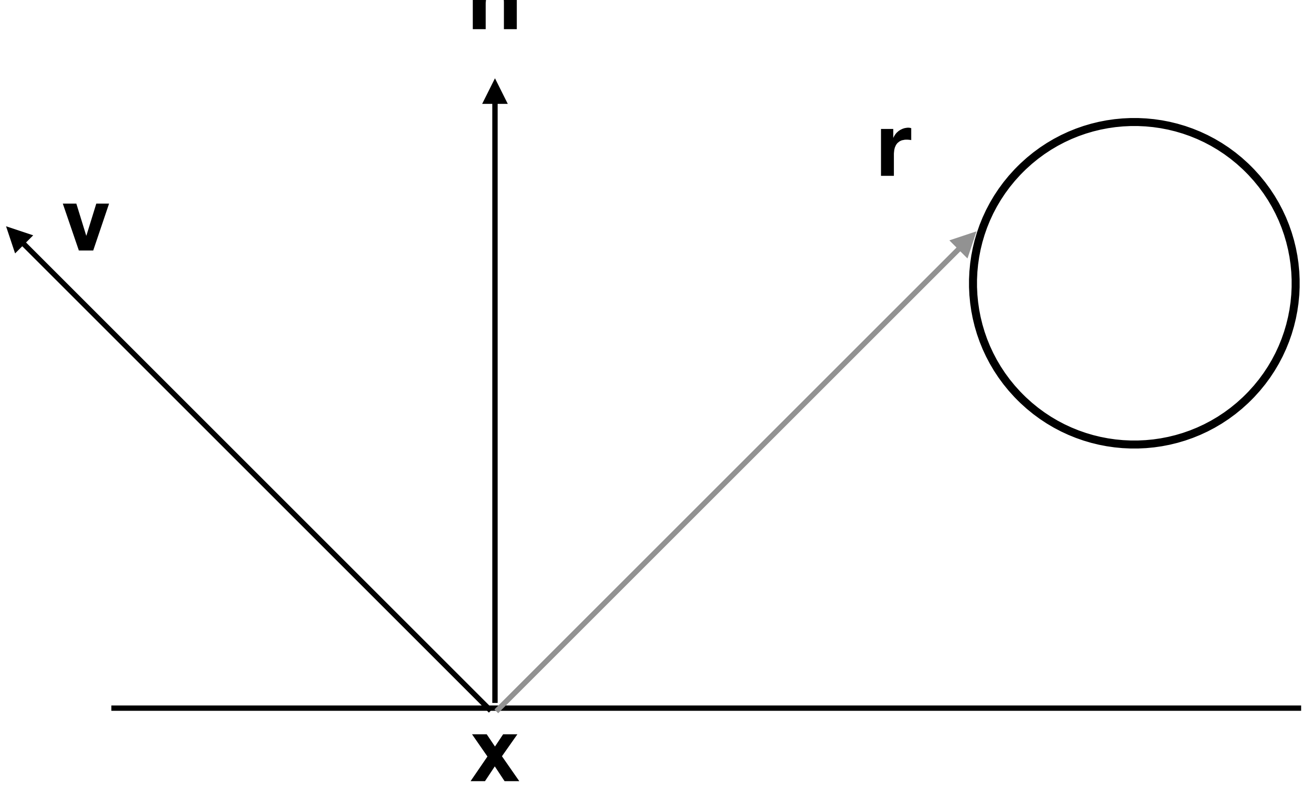


What does a camera see when it looks at a mirror?

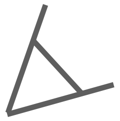


From last time:

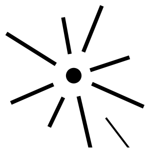
$$\vec{r} = -\vec{v} + 2(\vec{v} \cdot \vec{n})\vec{n}$$



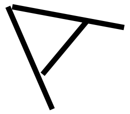
```
mirr_ray.origin = x
mirr_ray.direction = r
```

Hint: 

Mirror Reflection

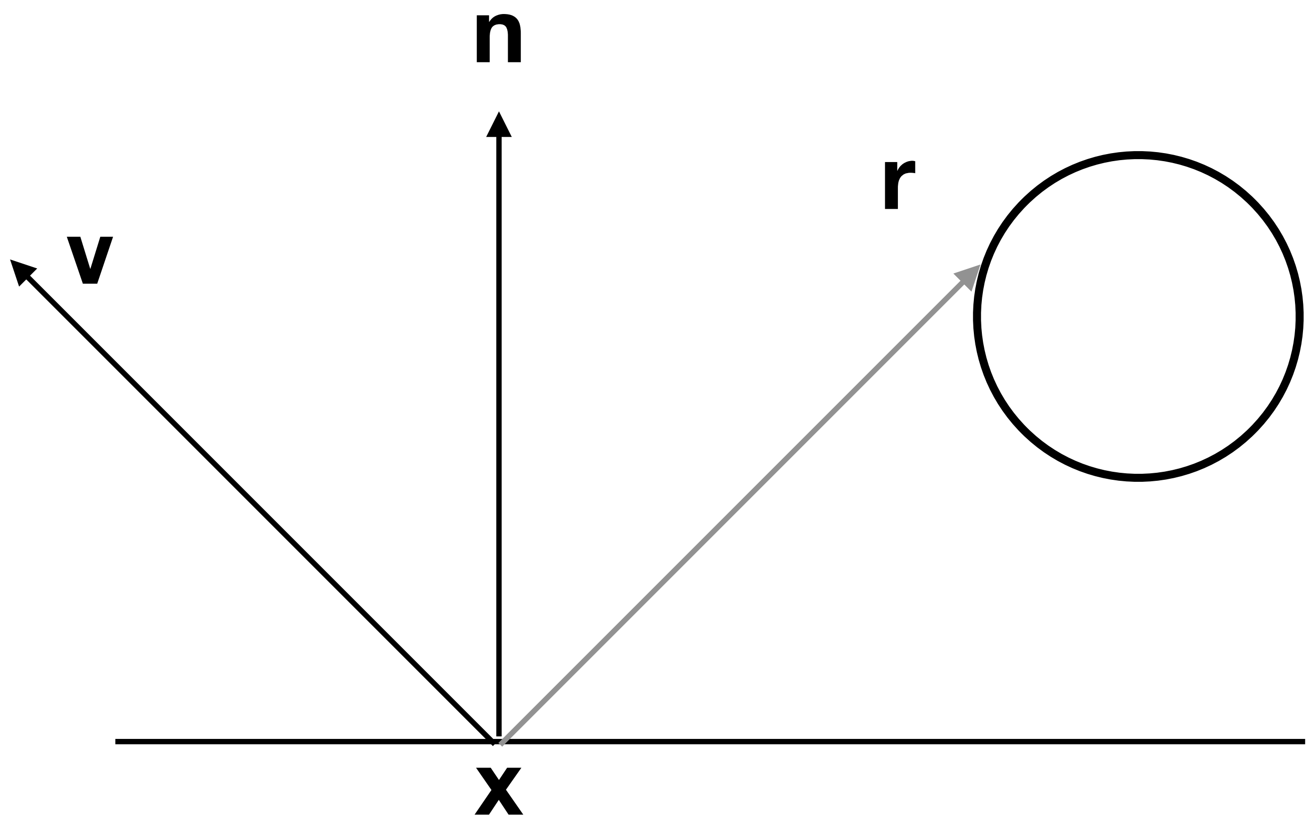


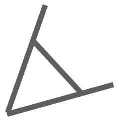
What does a camera see when it looks at a mirror?



From last time:

$$\vec{r} = -\vec{v} + 2(\vec{v} \cdot \vec{n})\vec{n}$$



Hint: 

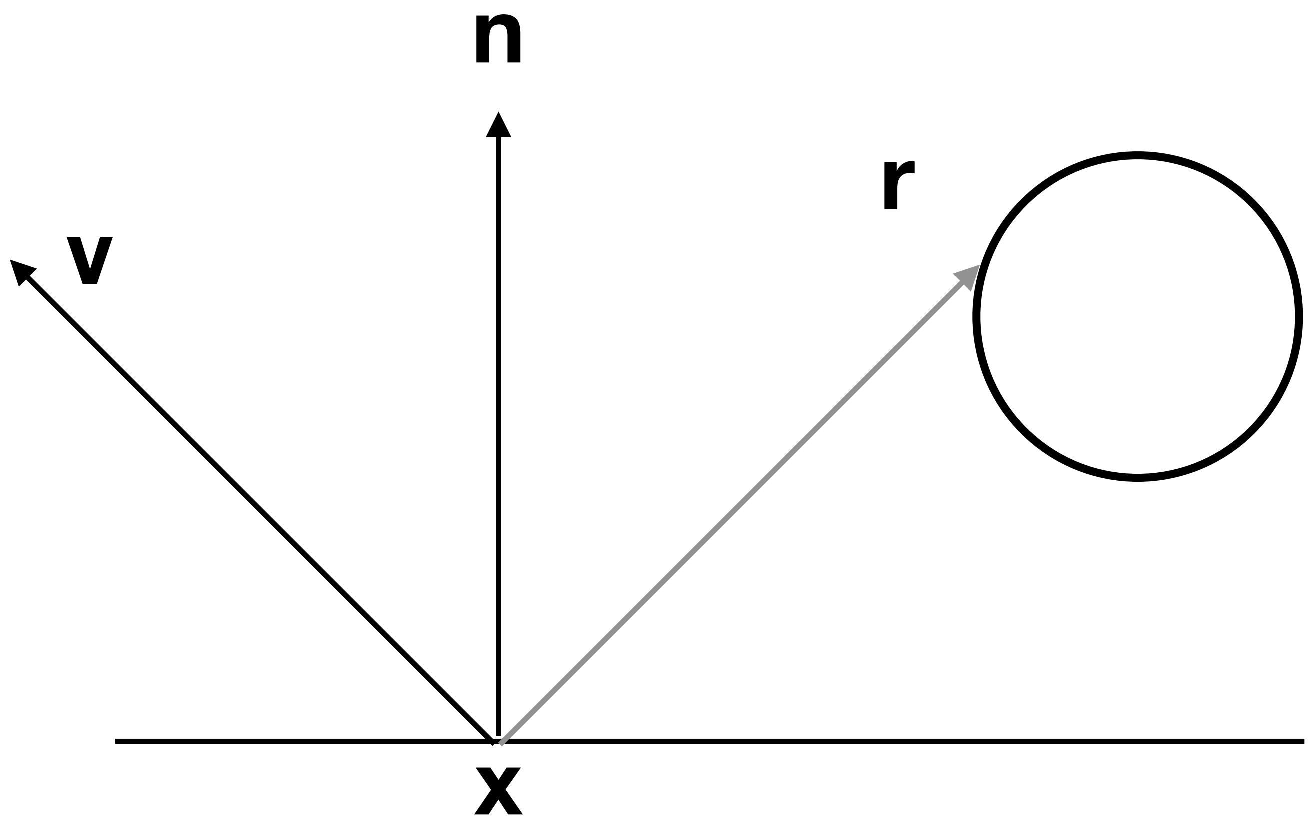
```
mirr_ray.origin = x
mirr_ray.direction = r
color = traceray(scene, mirr_ray, epsilon, Inf):
```

Mirror Reflection

What does a camera see when it looks at a mirror?

From last time:

$$\vec{r} = -\vec{v} + 2(\vec{v} \cdot \vec{n})\vec{n}$$



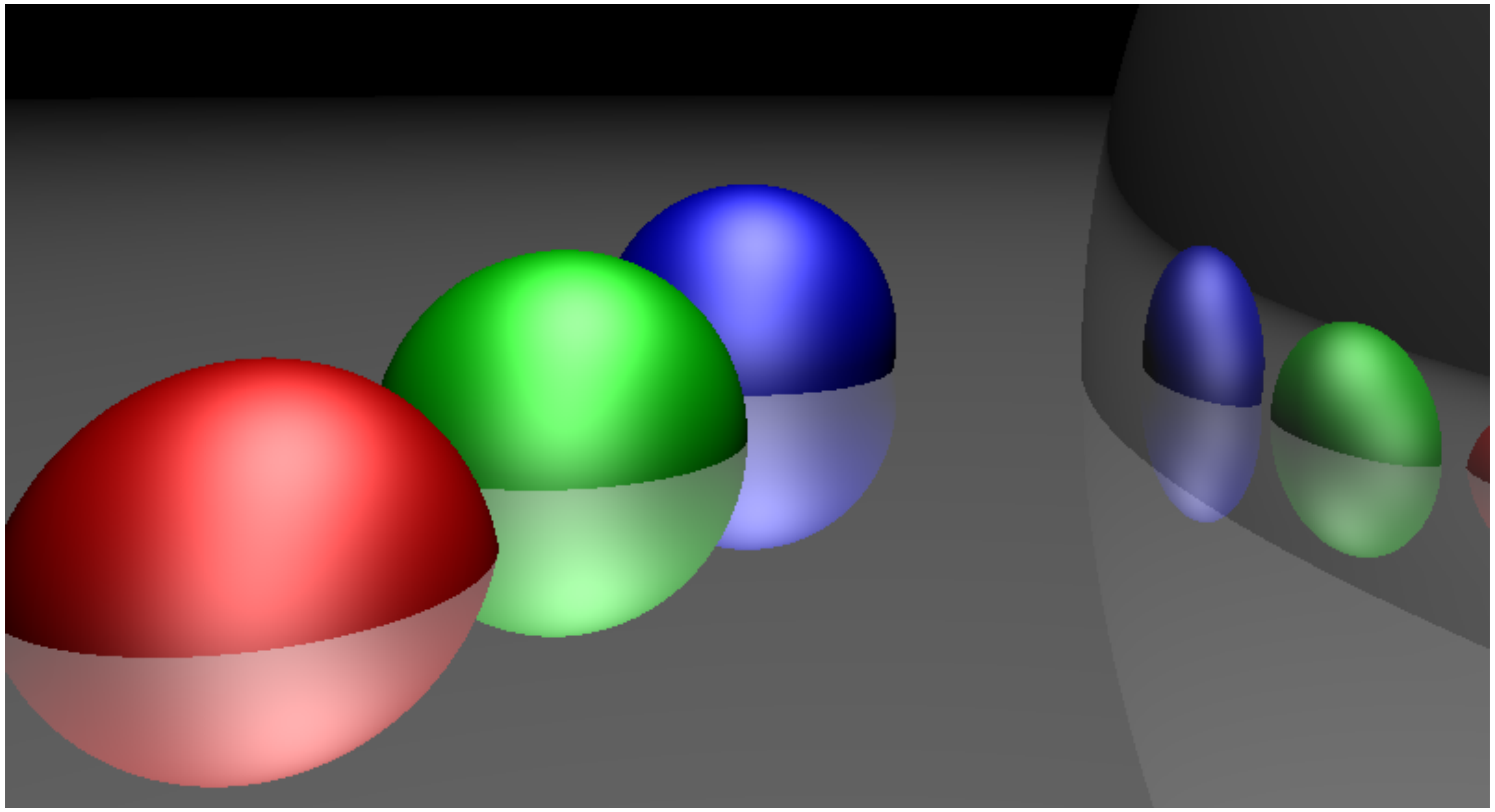
```
mirr_ray.origin = x
mirr_ray.direction = r
color = traceray(scene, mirr_ray, epsilon, Inf):
```

Hint:

small value to avoid hitting the surface x lies on

Partially-Mirrored Surfaces

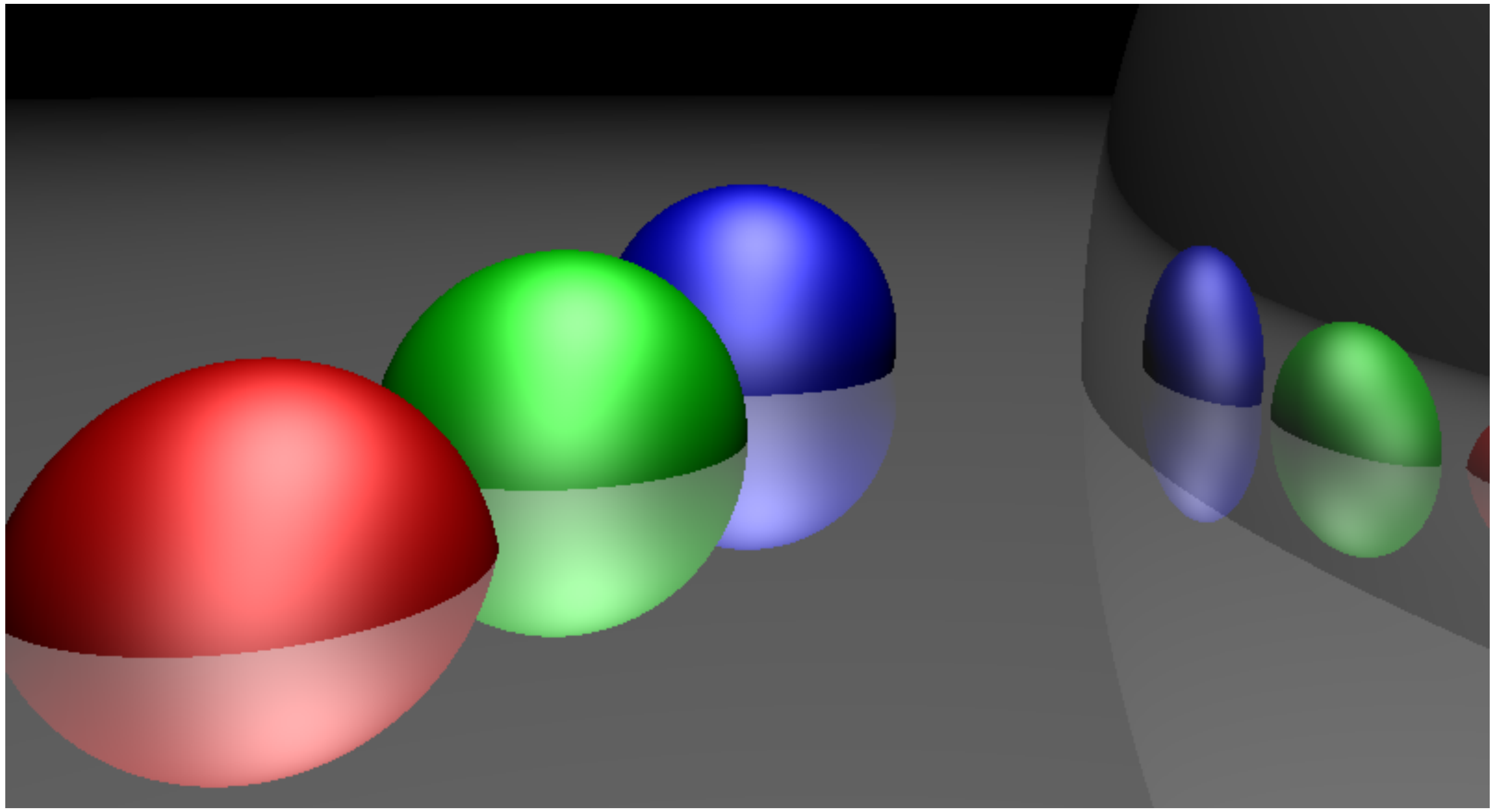
Notice the floor is gray but also mirror-reflective.



Materials store a **mirror coefficient:** fraction of light that is reflected in a mirror-like fashion

Partially-Mirrored Surfaces

Notice the floor is gray but also mirror-reflective.

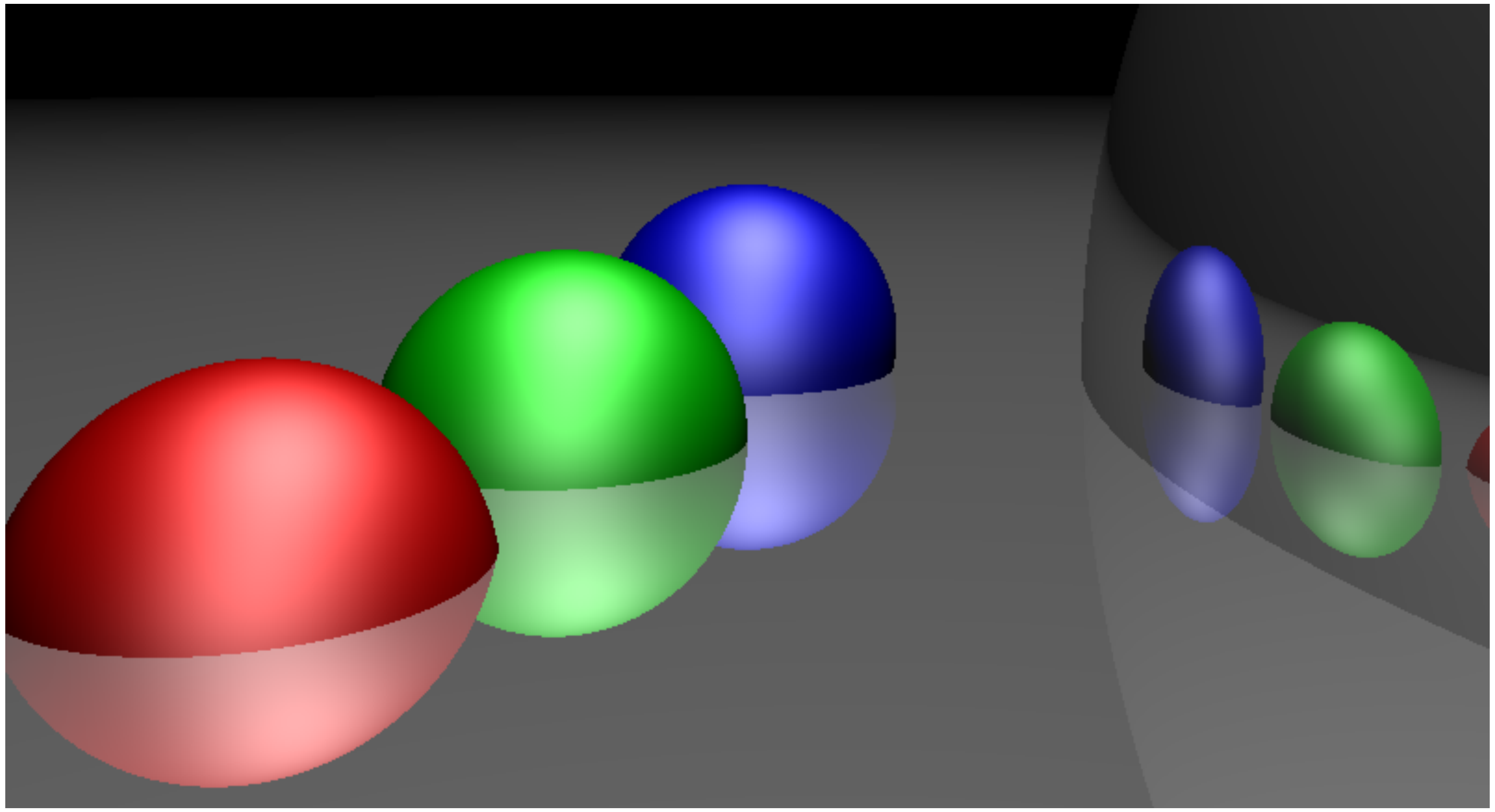


Materials store a **mirror coefficient:** fraction of light that is reflected in a mirror-like fashion

$$L = k_m L_r + (1 - k_m)(L_d + L_s)$$

Partially-Mirrored Surfaces

Notice the floor is gray but also mirror-reflective.



Materials store a **mirror coefficient**: fraction of light that is reflected in a mirror-like fashion

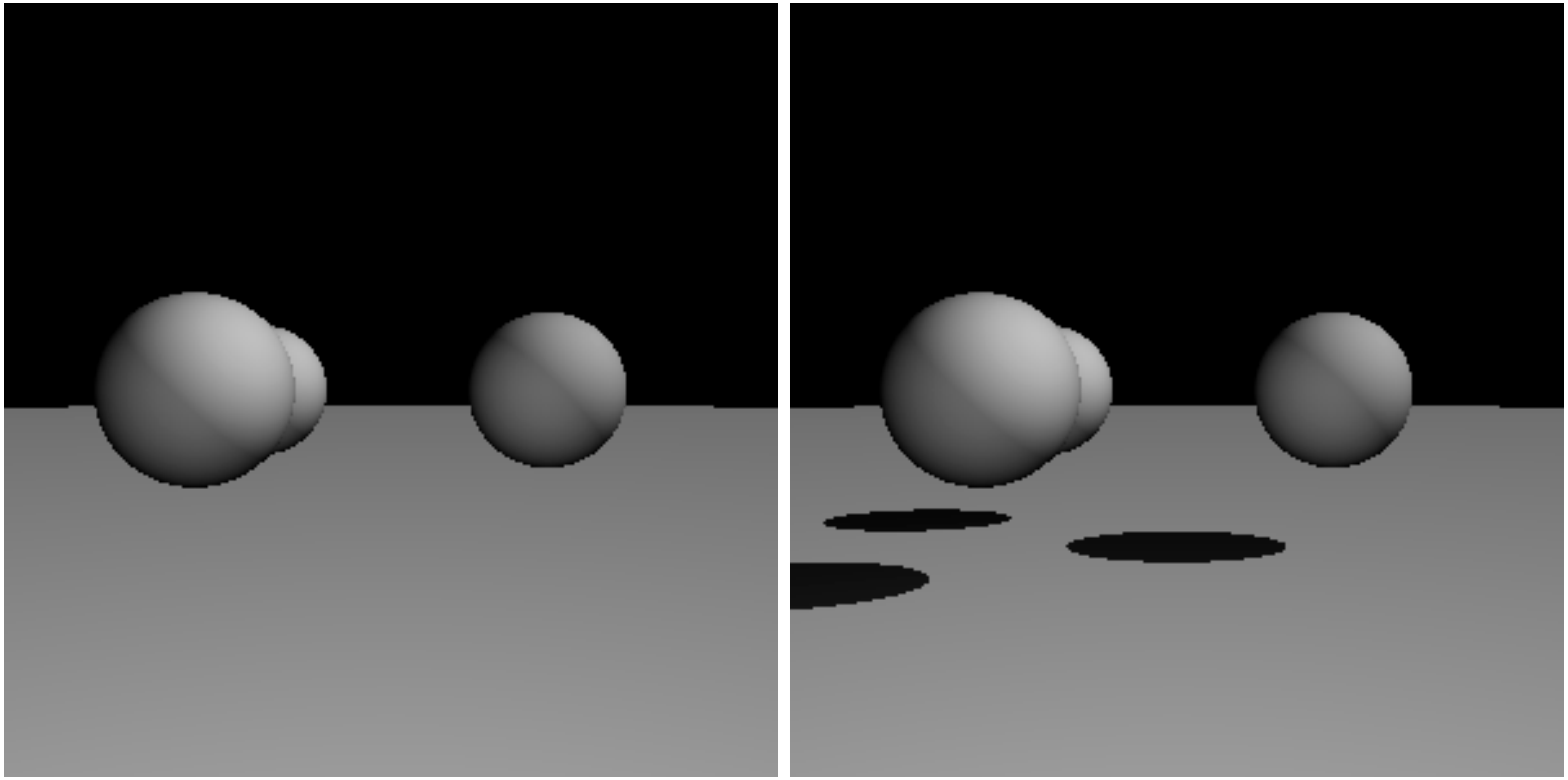
$$L = k_m L_r + (1 - k_m)(L_d + L_s)$$

mirror coefficient

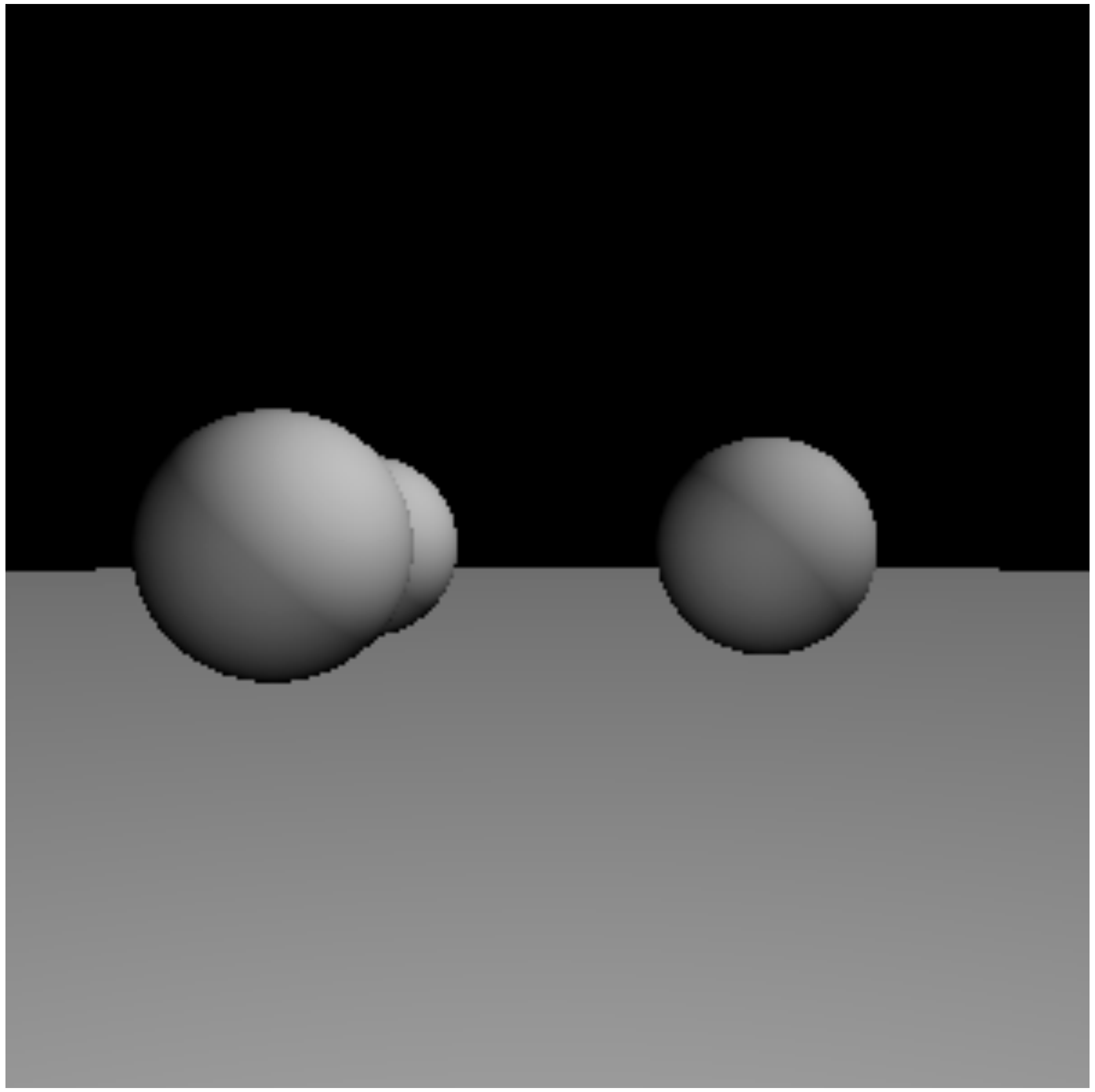
mirror-reflected light

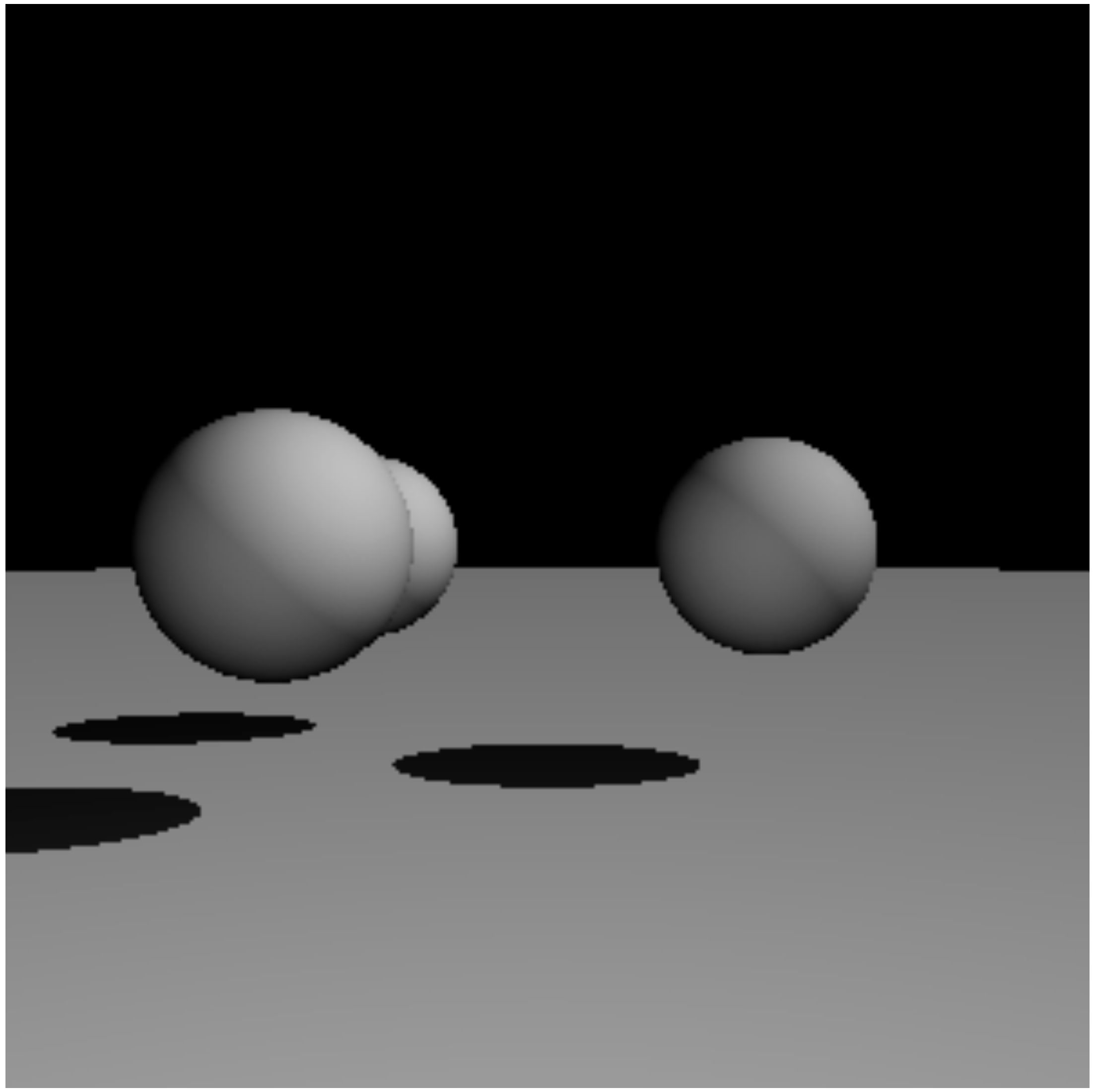
"local" color (Blinn-Phong)

Shadows



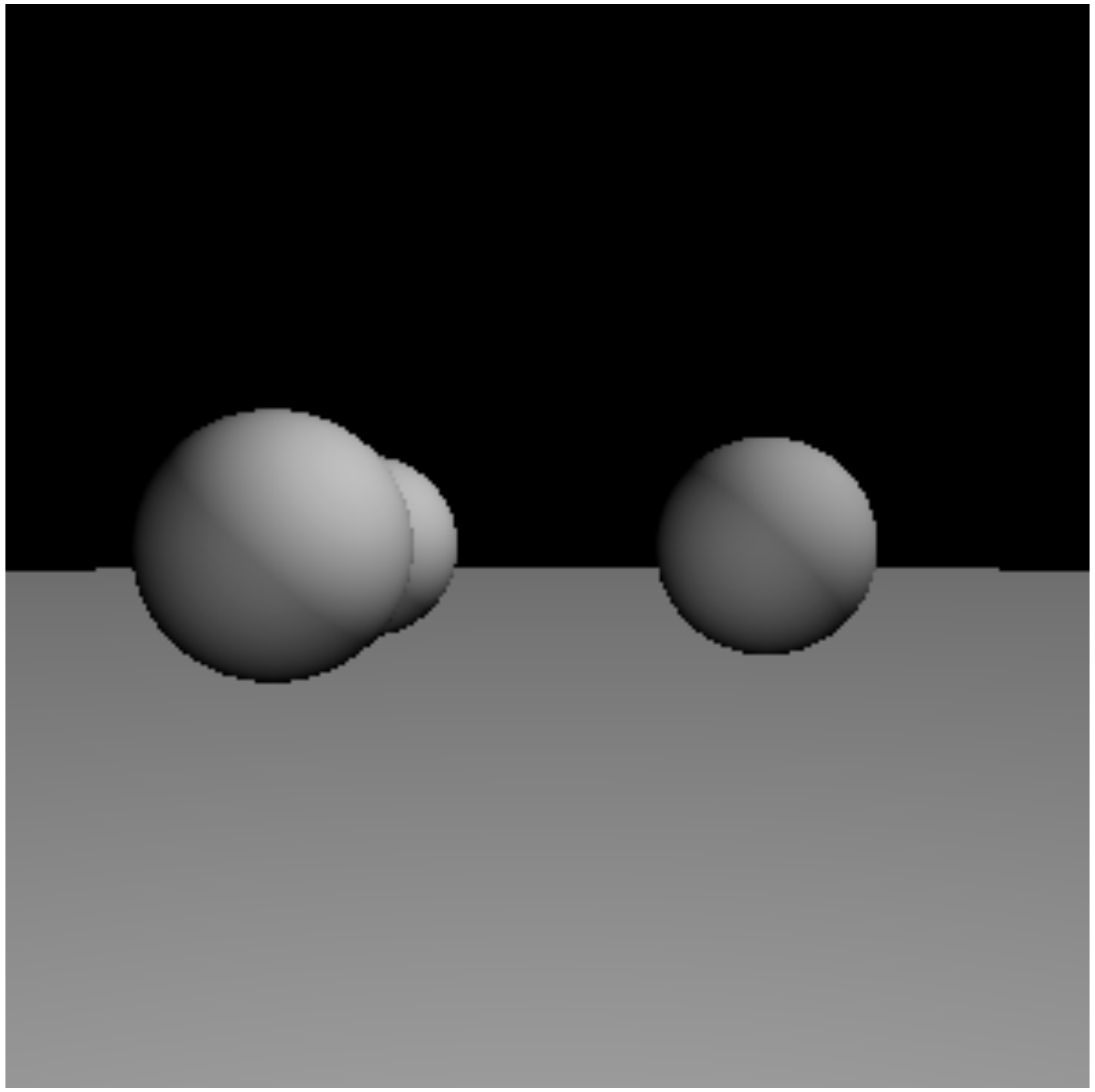
Shadows



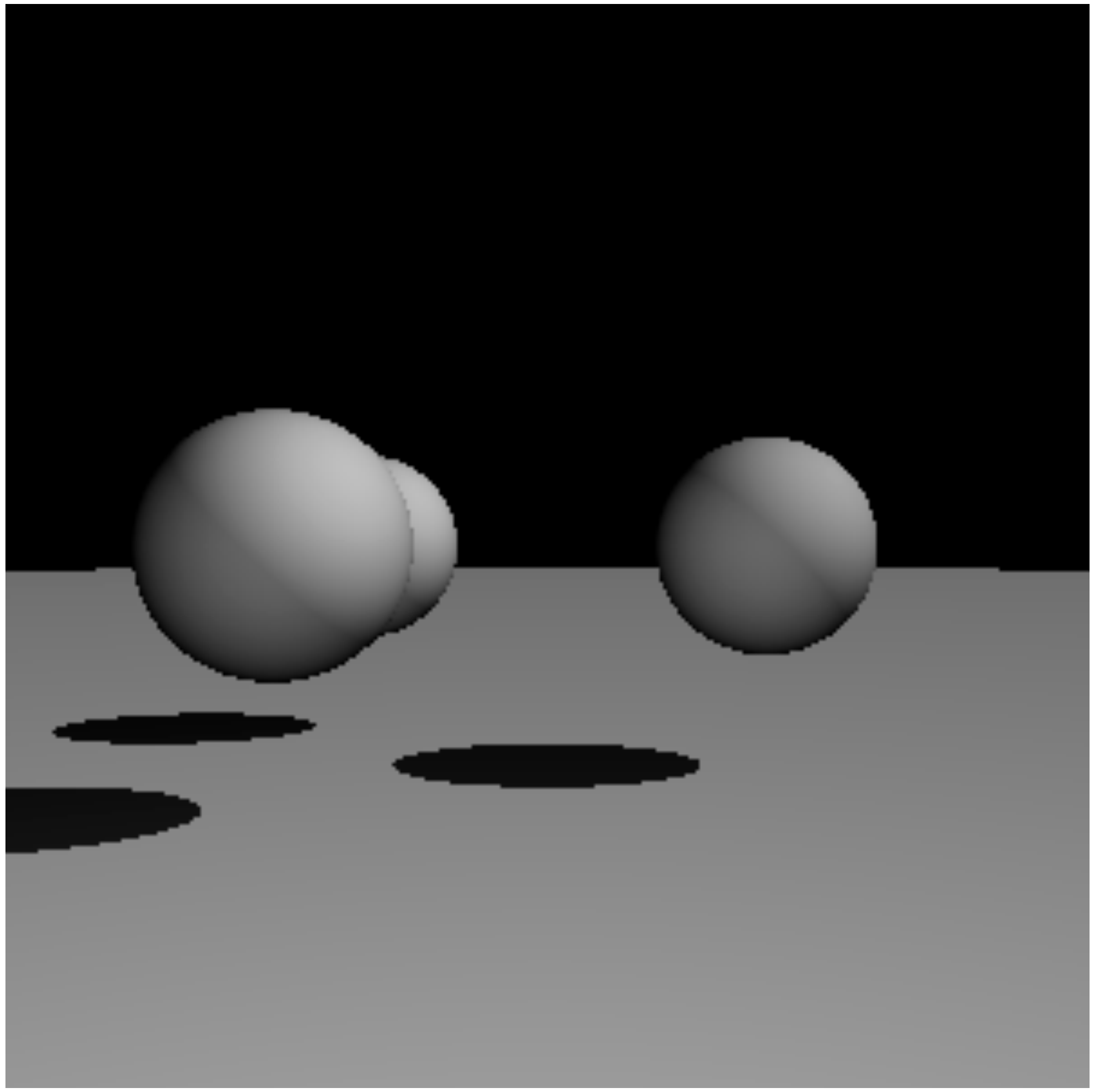


Less Wrong

Shadows



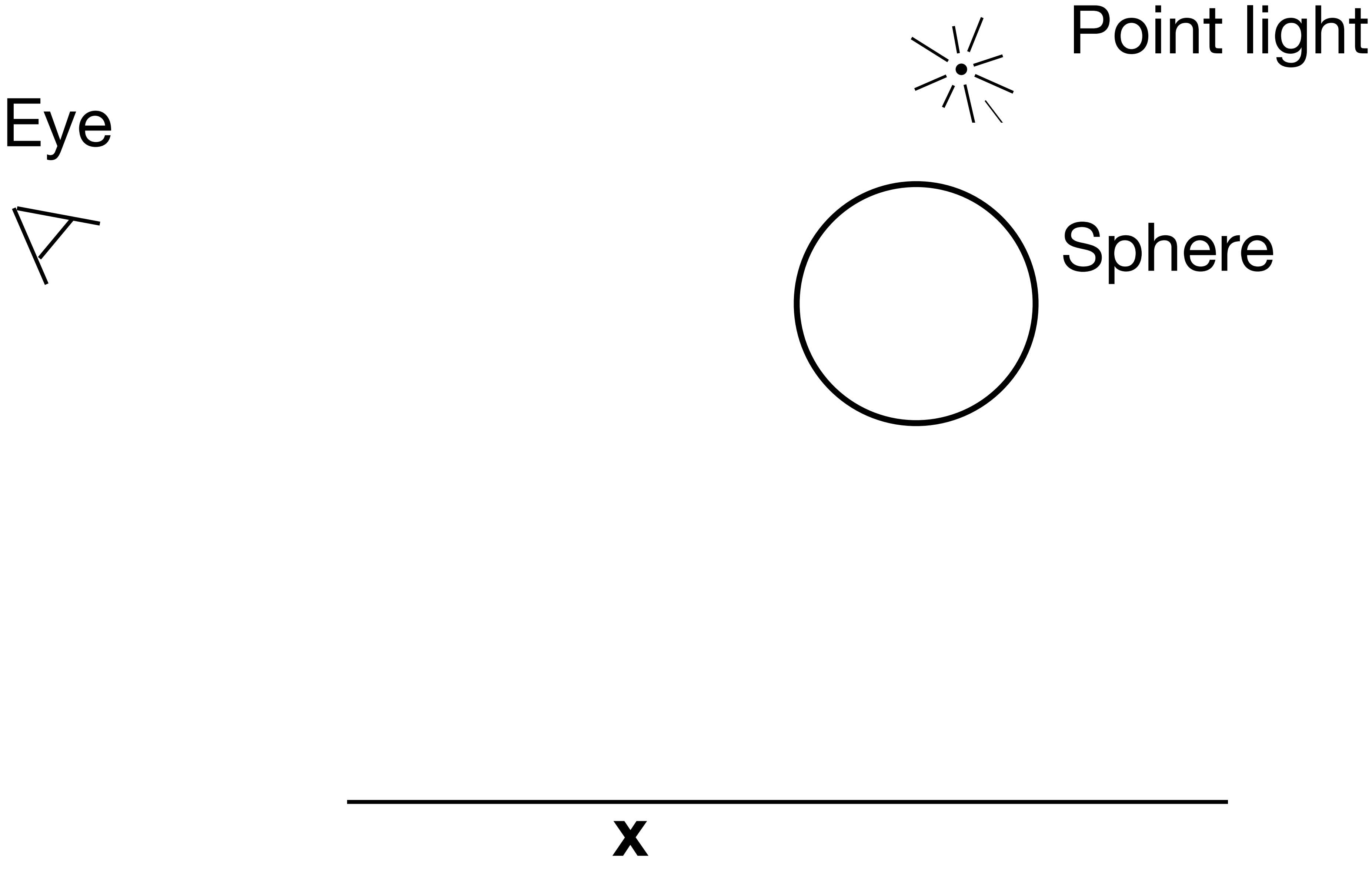
Wrong



Less Wrong

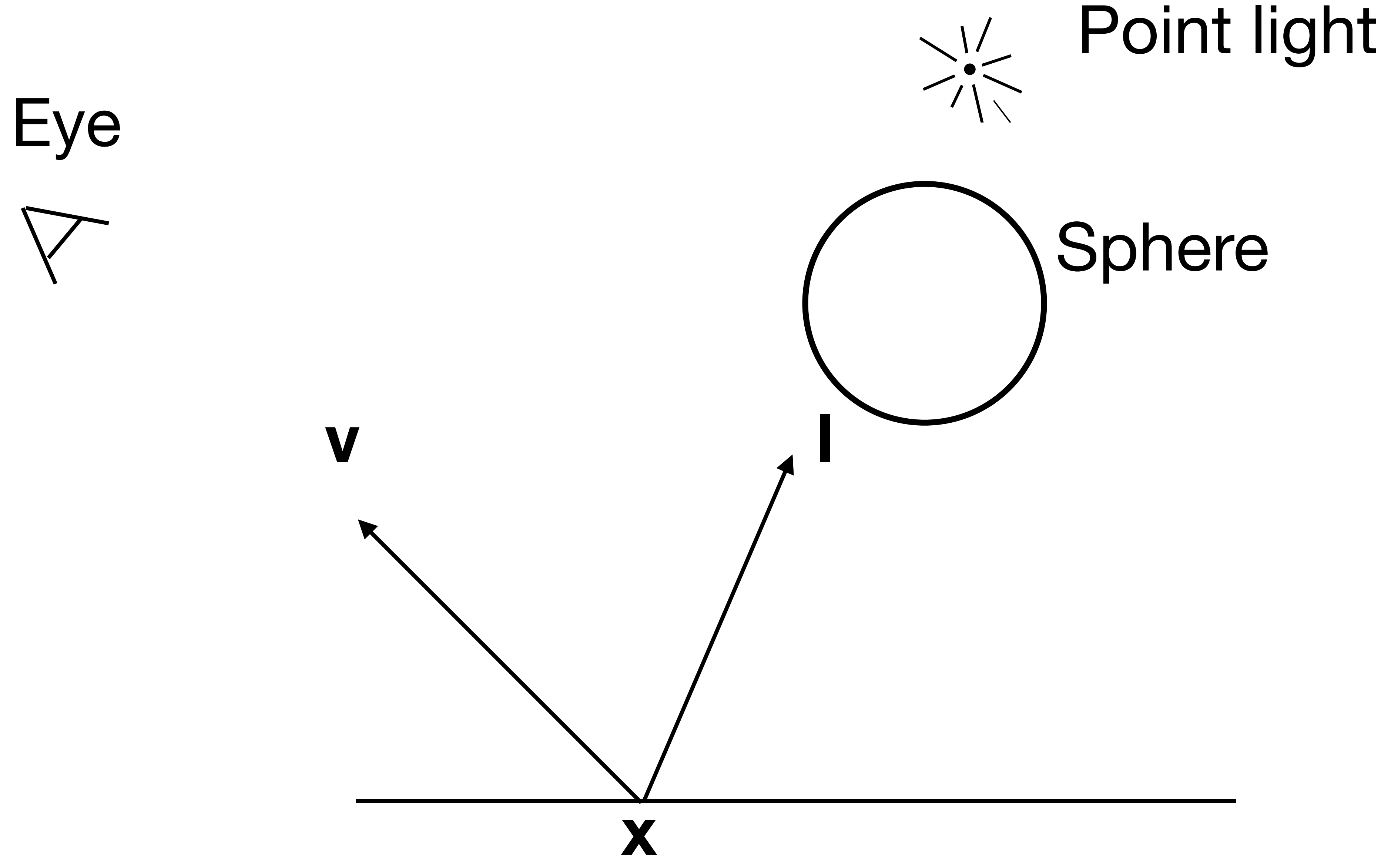
Shadows

How can we tell if a point is in shadow?



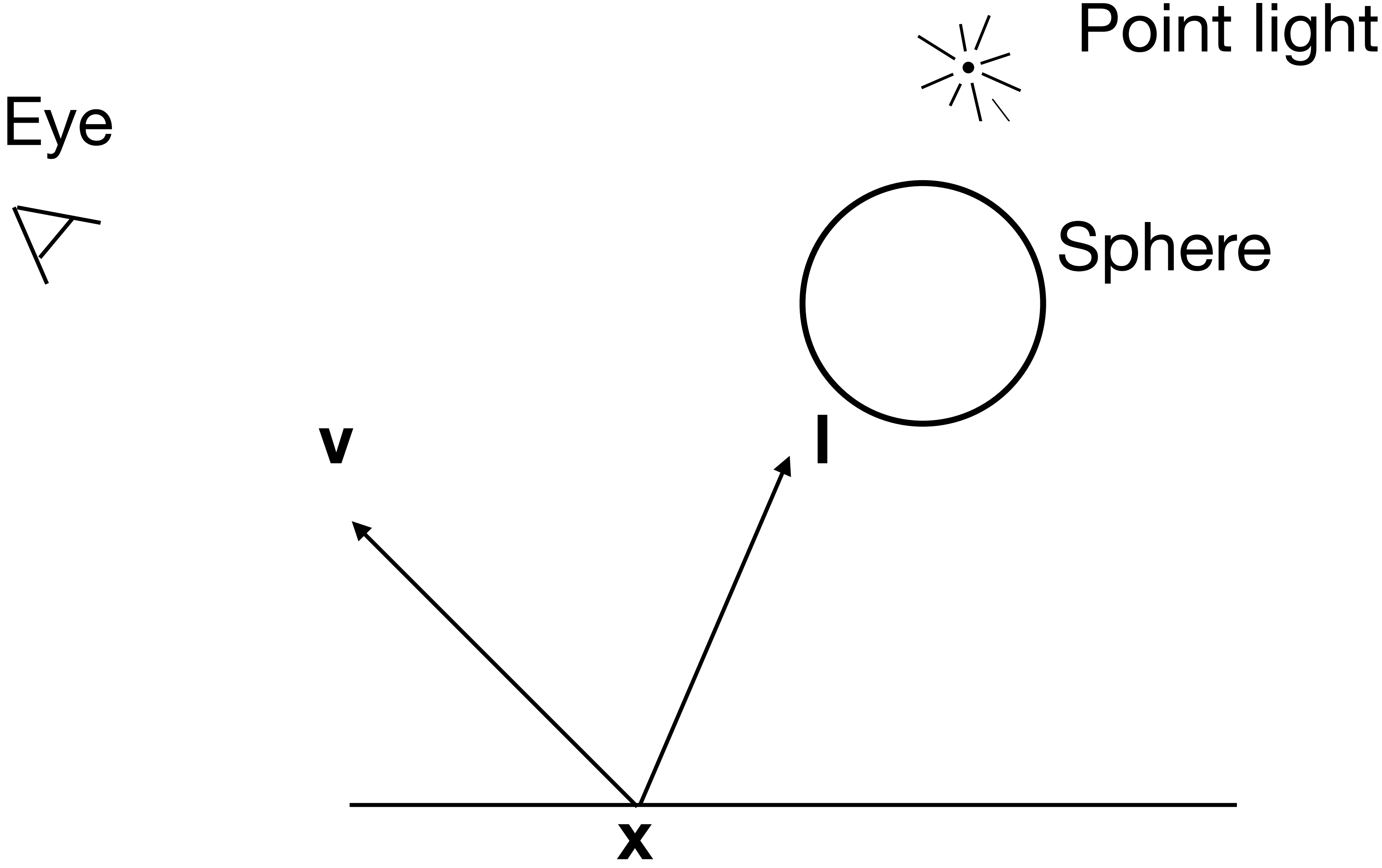
Shadows

How can we tell if a point is in shadow?



Shadows

How can we tell if a point is in shadow?



Point is shadowed iff:

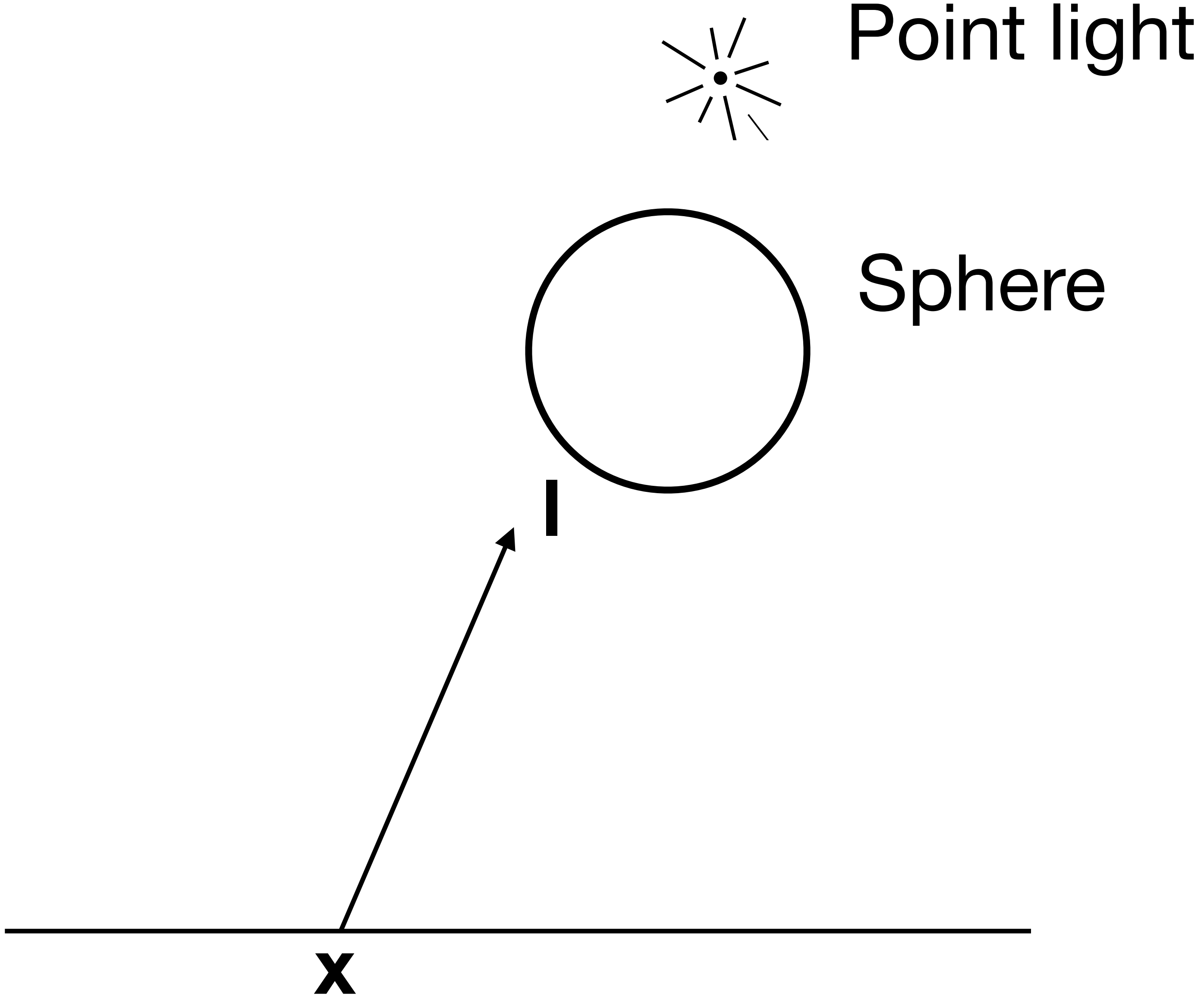
```
closest_intersect(objs, Ray(x, l), tmin, tmax) != nothing
```

Shadows

How can we tell if a point is in shadow?

Exercise: What do we use for tmin, tmax?

	Directional light $\vec{\ell}$	**Point light** $\vec{s}$
`r.orig`	$\mathbf{x}$	$\mathbf{x}$
`r.dir`	$\vec{\ell}$	$\vec{s} - \mathbf{x}$
`tmin`	`eps`	`eps`
`tmax`	`Inf`	`1`



Point is shadowed iff:

```
closest_intersect(objs, Ray(x, l), tmin, tmax) != nothing
```

```
function determine_color(hitrec, ray, scene, ...):
```

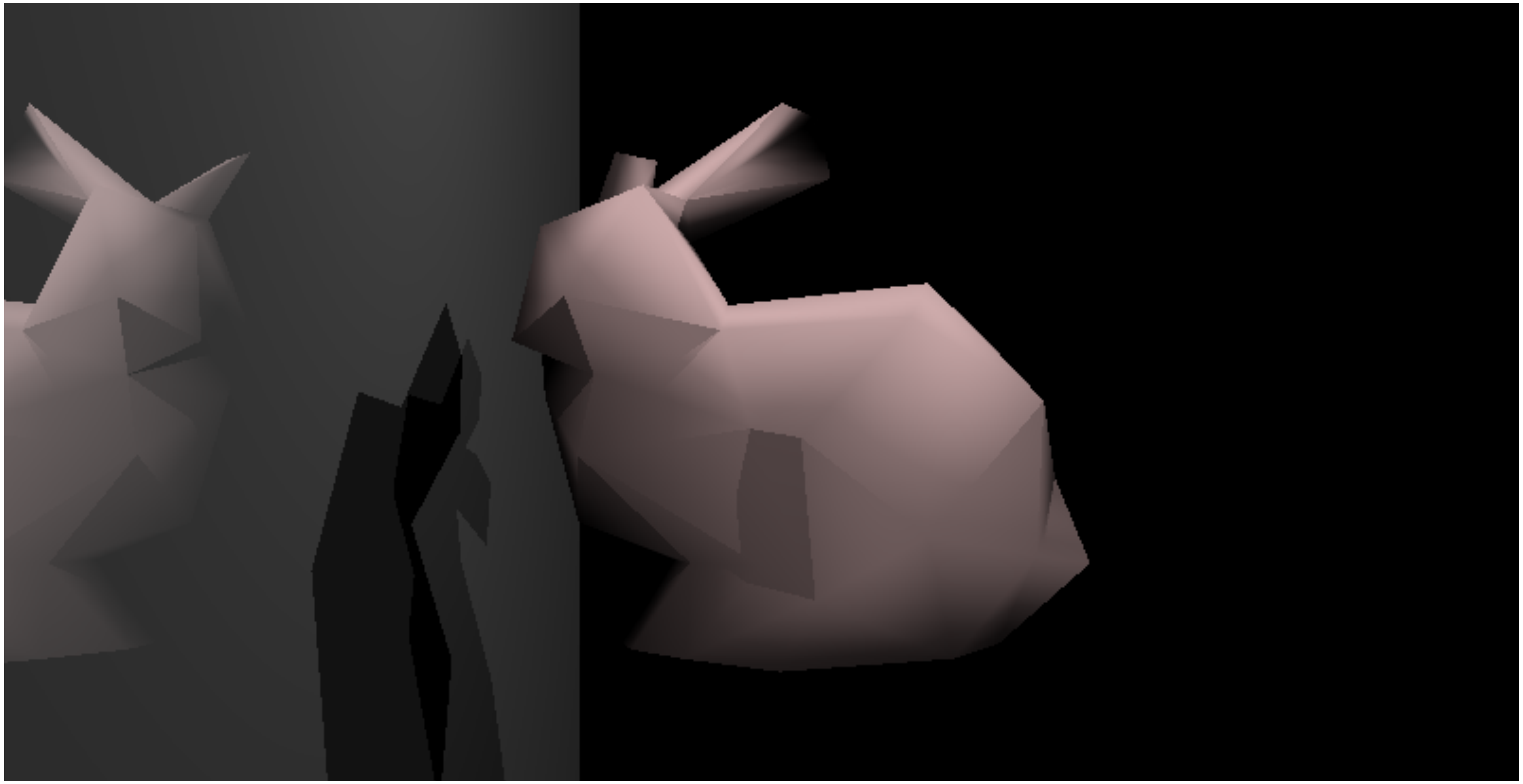
```
function determine_color(hitrec, ray, scene, ...):
    color = black
```

```
function determine_color(hitrec, ray, scene, ...):
    color = black
    for light in scene.lights:
```

```
function determine_color(hitrec, ray, scene, ...):
    color = black
    for light in scene.lights:
        if !is_shadowed(scene, light, hitrec)
```

```
function determine_color(hitrec, ray, scene, ...):
    color = black
    for light in scene.lights:
        if !is_shadowed(scene, light, hitrec)
            color += shade_light(light, hitrec, ...)
```

Let's talk about bunnies.



If we want bunnies, we still need to implement

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

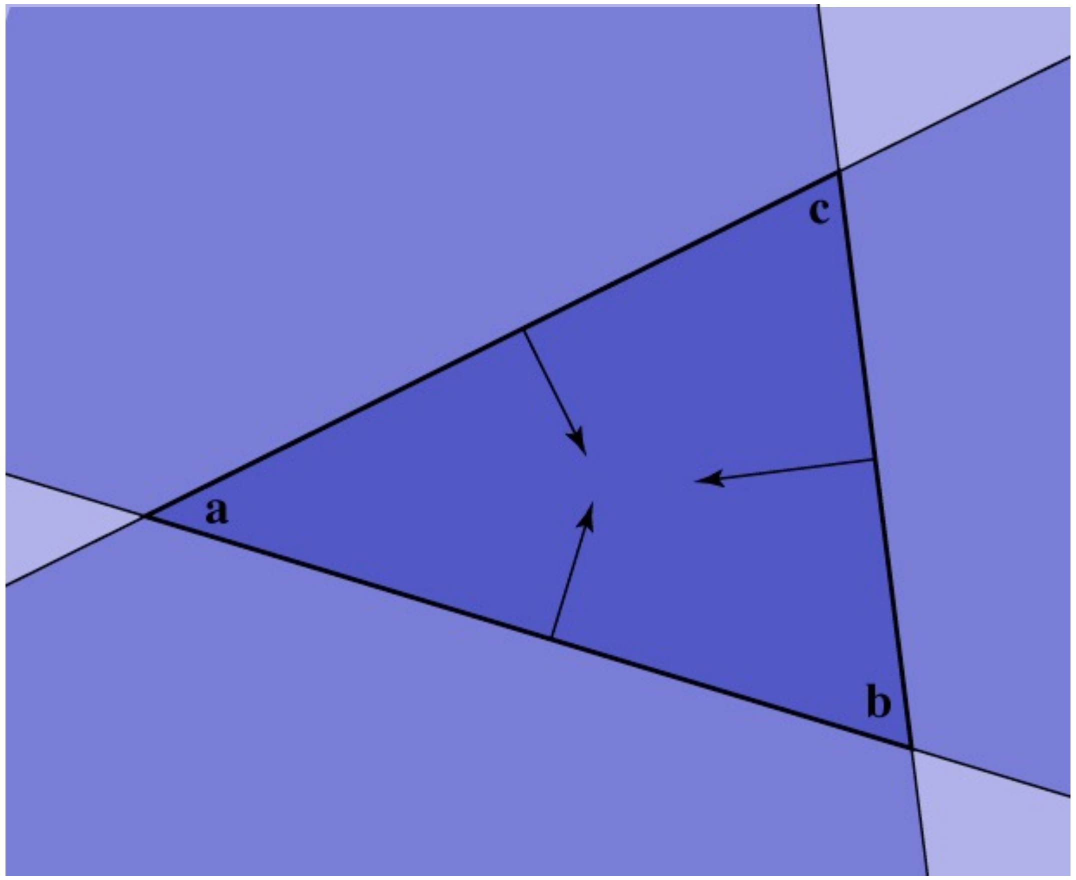
Then, we can treat a triangle mesh as simply a list of triangles.

Let's talk about triangles.

A triangle is the intersection of three half-planes

High-level approach:

1. Intersect with the plane
2. Check if intersection is inside the triangle

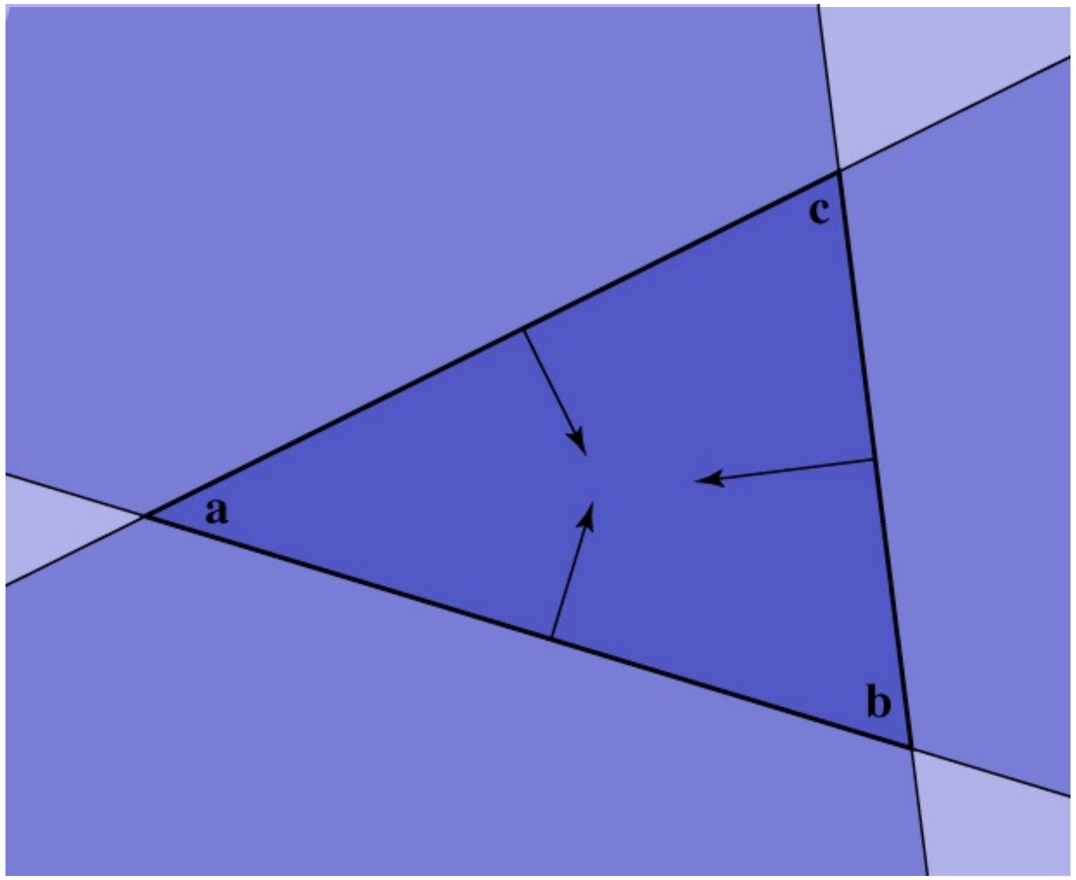


Let's talk about triangles.

A triangle is the intersection of three half-planes

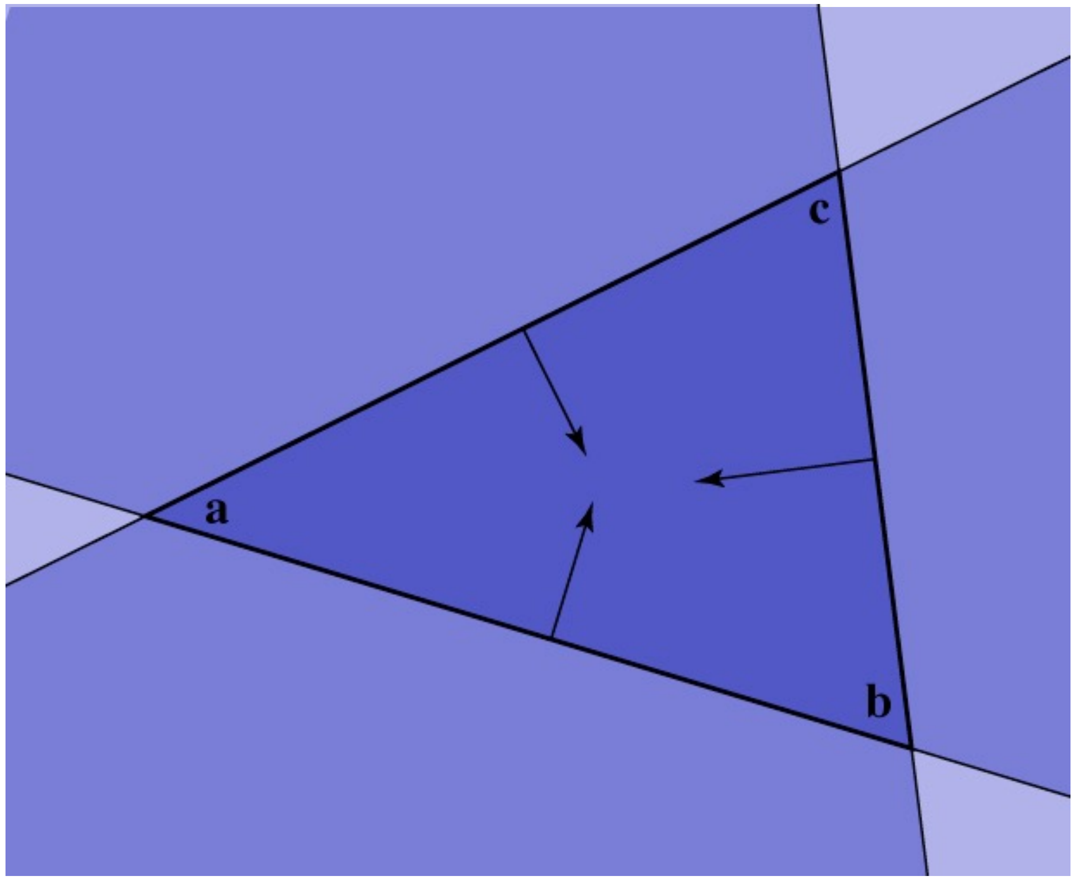
High-level approach:

1. Intersect with the plane
2. **Check if intersection is inside the triangle**



Let's talk about triangles.

A triangle is the intersection of three half-planes



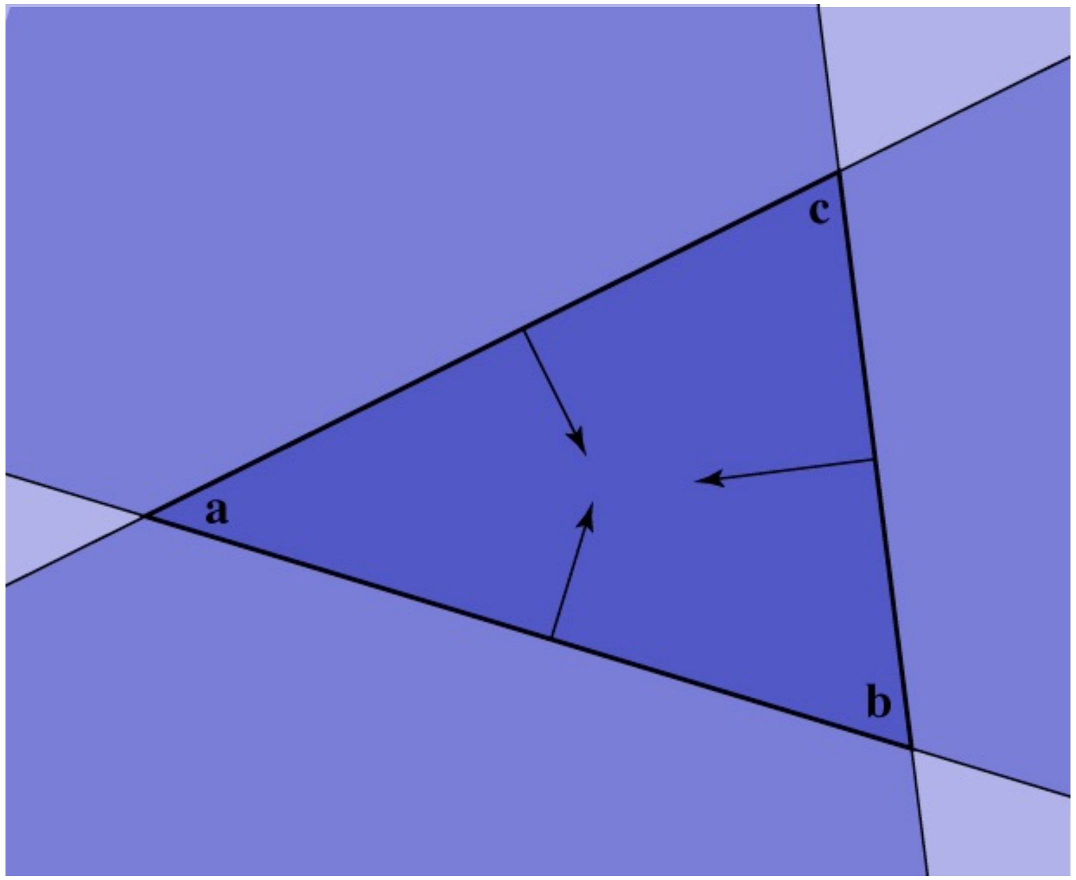
High-level approach:

1. Intersect with the plane
2. **Check if intersection is inside the triangle**

To make this easy, we'll introduce the
weirdest coordinate system you've ever seen.

Let's talk about triangles.

A triangle is the intersection of three half-planes



High-level approach:

1. Intersect with the plane
2. **Check if intersection is inside the triangle**

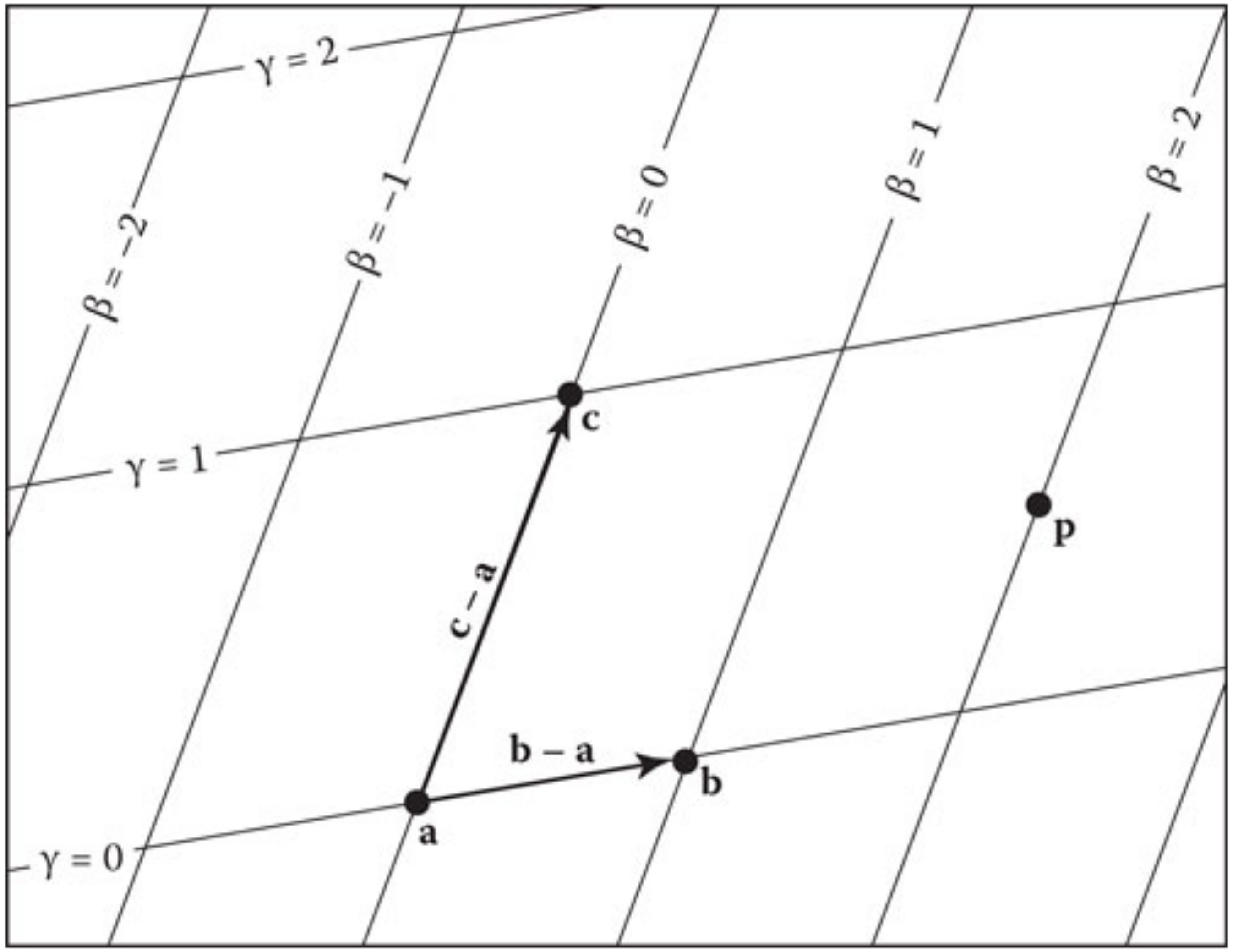
To make this easy, we'll introduce the
weirdest coordinate system you've ever seen.

As a bonus, we'll get interpolation of vertex quantities for free!

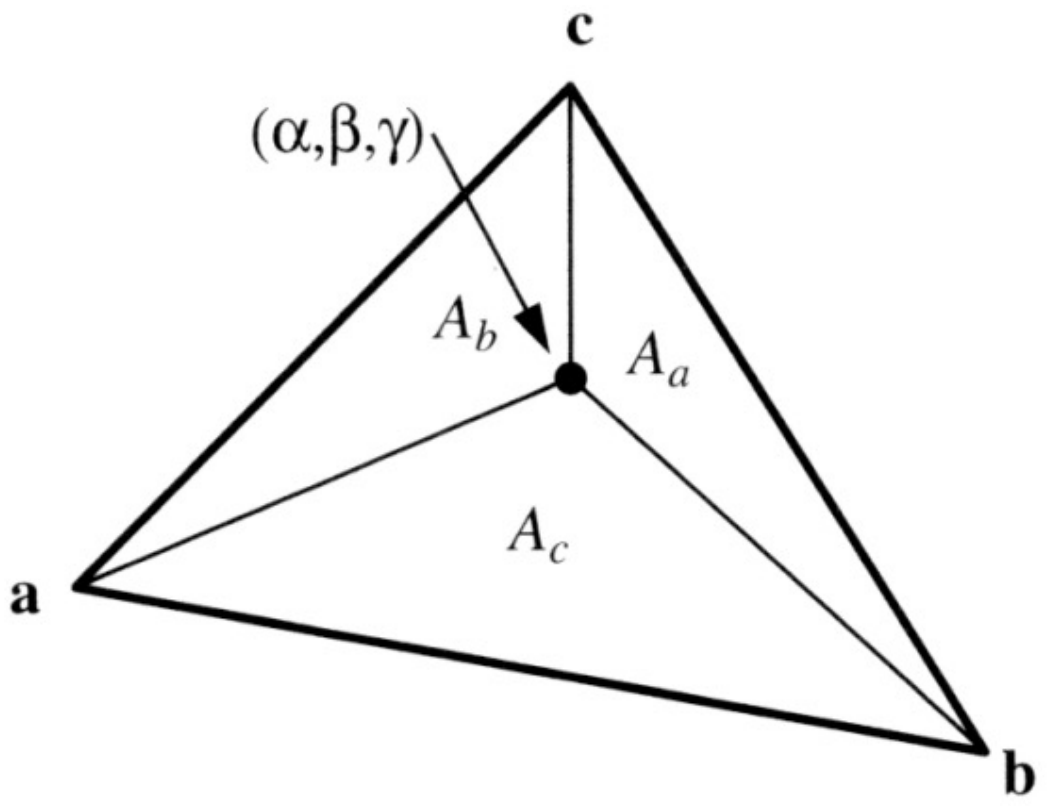
Barycentric Coordinates

A purpose-built coordinate system for talking about points in a specific triangle's plane.

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$



- Coordinates are proportional to area of subtriangles:


c
(α,β,γ)
A_b
A_a
A_c
a
b