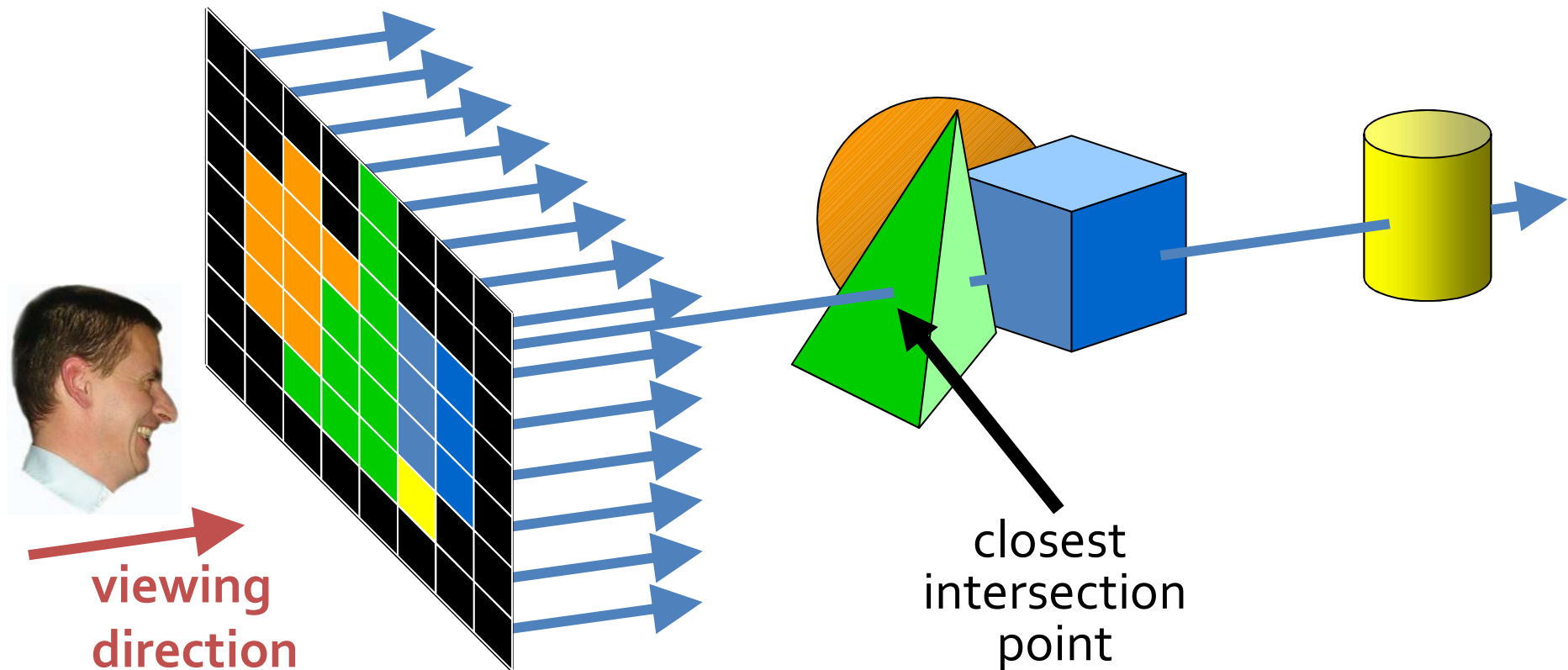


Ray-Casting

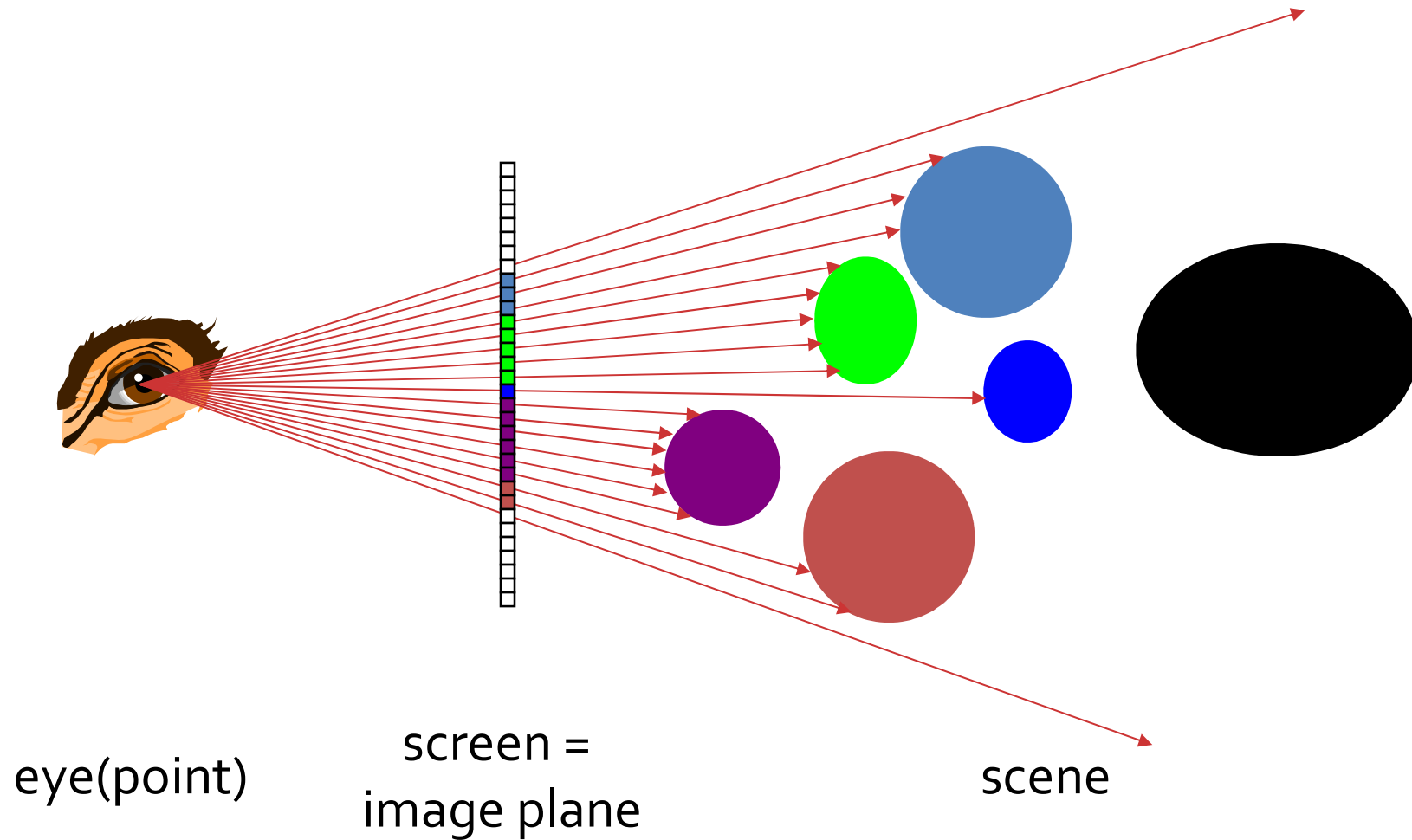
Ray-Casting Method

- line-of-sight of each pixel is intersected with all surfaces
- take closest intersected surface



Generating Rays

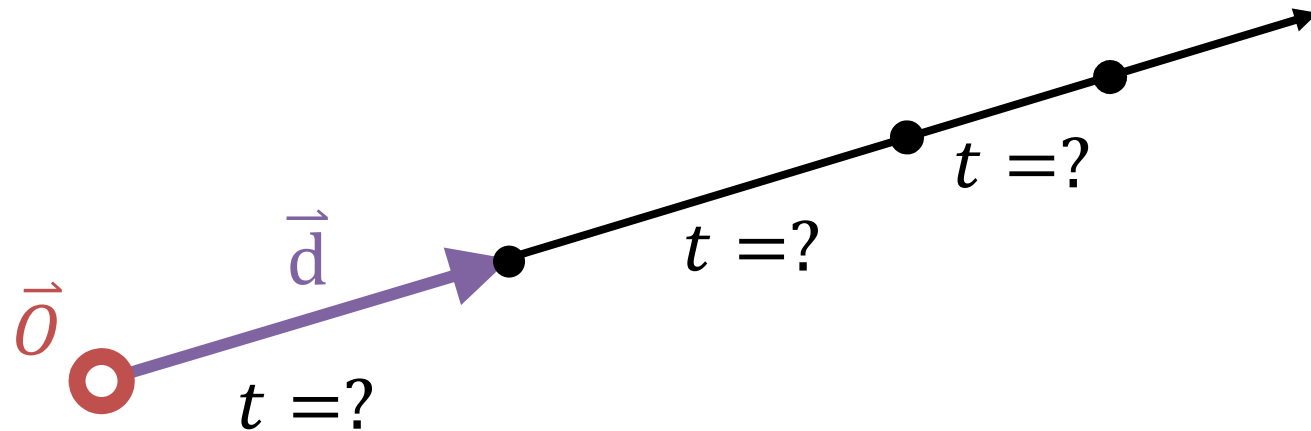
- Trace a ray for each pixel in the image plane



Ray Parametric Form

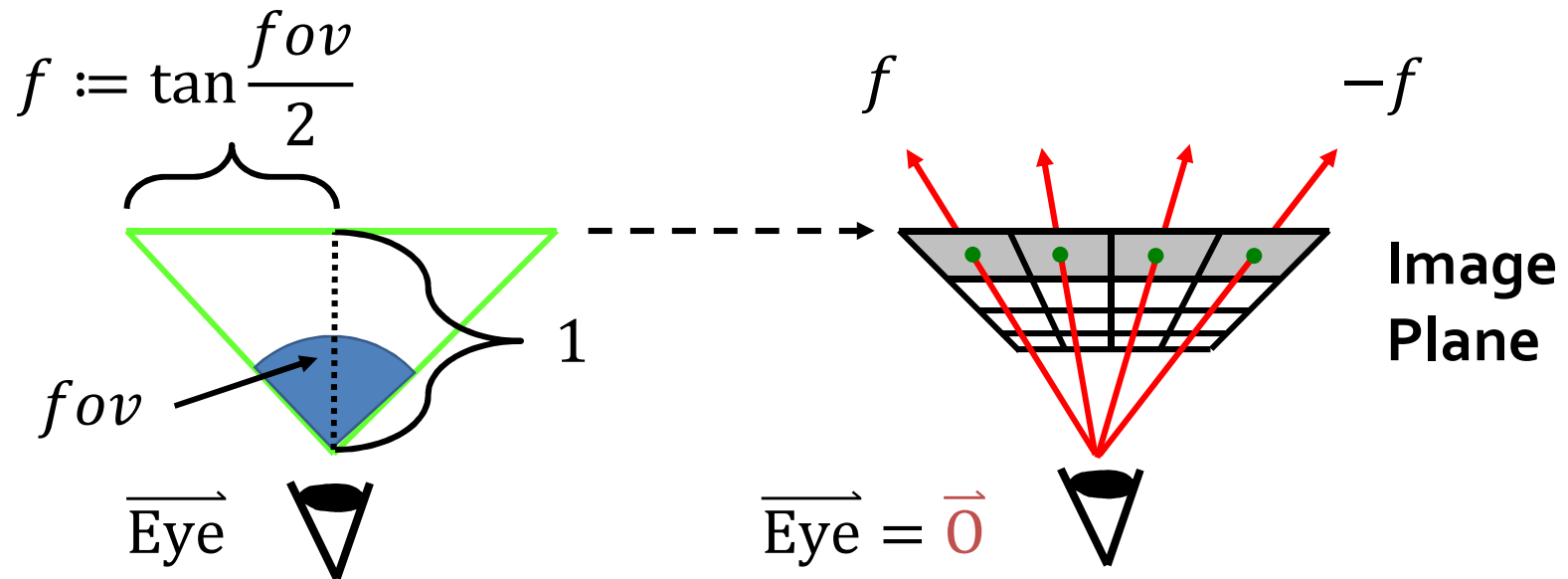
- Ray expressed as function of a single parameter t

$$\begin{aligned}\vec{P} &= \vec{O} + t\vec{d} \\ &= \begin{pmatrix} O_x \\ O_y \\ O_z \end{pmatrix} + t \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}\end{aligned}$$



Generating Rays – Top View

- Trace a ray for each pixel in the image plane

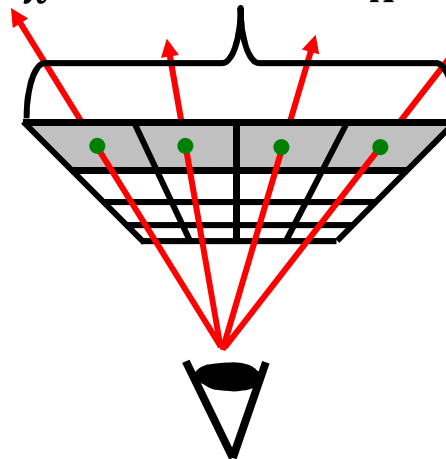


Generating Rays – Top View

- Trace a ray for each pixel in the image plane

- $$d_x(x) = \frac{2fx}{\text{resolution}_x} - f = \frac{f(2x - \text{resolution}_x)}{\text{resolution}_x}$$

$$f \triangleq x = \text{resolution}_x \quad \text{resolution}_x \quad -f \triangleq x = 0$$

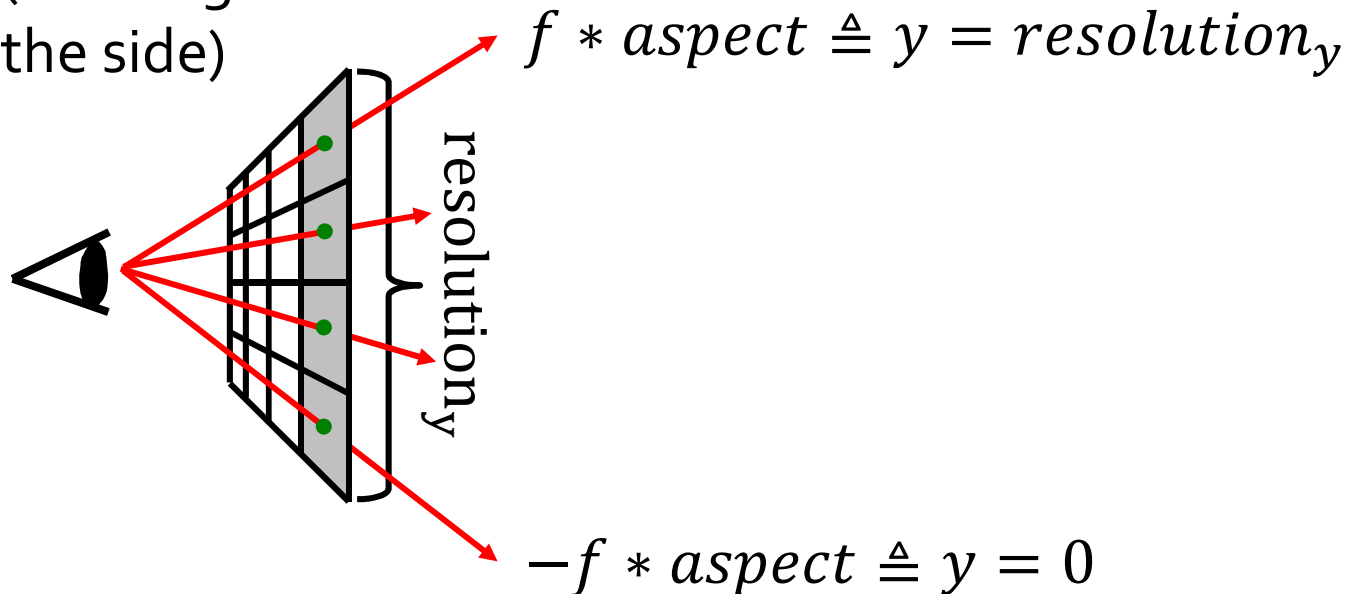


(Looking down from the top)

Generating Rays – Side View

- Trace a ray for each pixel in the image plane
- $d_y(y) = aspect \left(\frac{2fy}{resolution_y} - f \right) =$
 $\frac{resolution_y}{resolution_x} \left(\frac{2fy}{resolution_y} - f \right) = \frac{f(2y - resolution_y)}{resolution_x}$

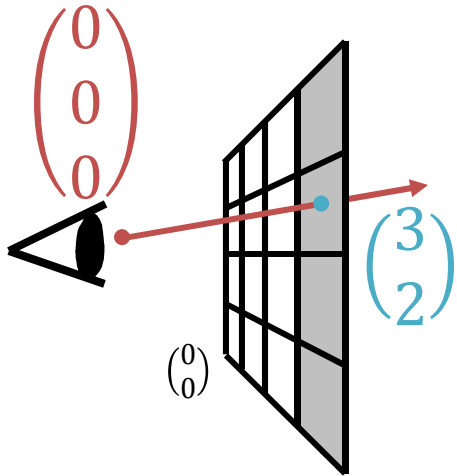
(Looking from
the side)



Generating Rays

- Trace a ray for each pixel in the image plane

- For a pixel $\begin{pmatrix} x \\ y \end{pmatrix}$: $\vec{P} = \vec{O} + t\vec{d} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + t \begin{pmatrix} d_x(x) \\ d_y(y) \\ 1 \end{pmatrix}$



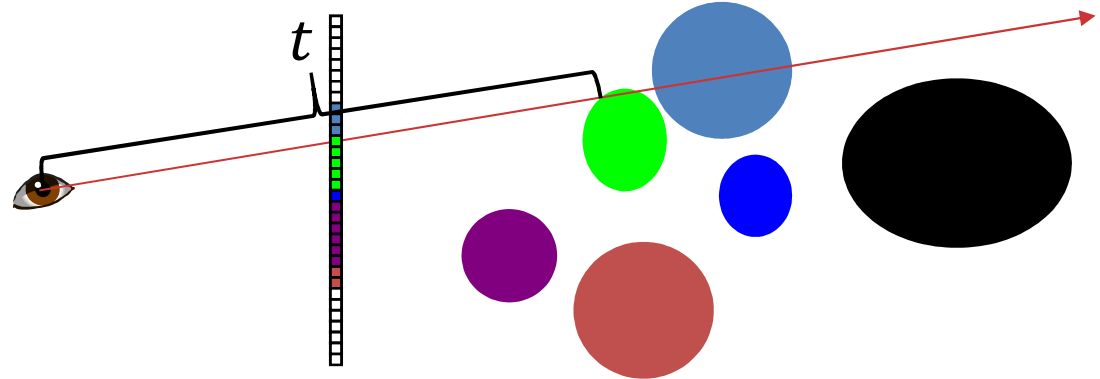
Generating Rays

- Trace a ray for each pixel in the image plane

```
renderImage() {  
    fov = 90°;  
    fakt = tan(fov / 2) / resolution.x;  
    for each pixel x, y in the image  
        dx = fakt * (2 * x - resolution.x);  
        dy = fakt * (2 * y - resolution.y);  
  
        ray.o = (0, 0, 0);  
        ray.d = normalize(dx, dy, 1);  
        image[x][y] = intersect(ray);  
}
```

Ray-Object Intersections

```
intersect(Ray r) {  
    foreach object in the scene  
        find minimum  $t > 0$ :  $r.O + t * r.d$  hits object  
        if ( object hit )  
            return object  
        else  
            return background  
}
```



Ray-Object Intersections

- Aim: Find the parameter value, t_i , at which the ray first meets object i
- Write the surface of the object implicitly: $f(\mathbf{x})=0$
 - Unit sphere at the origin is $\mathbf{x} \bullet \mathbf{x} - 1 = 0$
 - Plane with normal \mathbf{n} passing through origin is: $\mathbf{n} \bullet \mathbf{x} = 0$
- Put the ray equation in for \mathbf{x}
 - Result is an equation of the form $f(t)=0$ where we want t
 - Now it's just root finding

Ray Object Intersection

- Equation of a ray $r(t) = \mathbf{S} + \mathbf{c}t$
 - “ \mathbf{S} ” is the starting point and “ \mathbf{c} ” is the direction of the ray
- Given a surface in implicit form $F(x, y, z)$
 - *plane*: $F(x, y, z) = ax + by + cz + d = \mathbf{n} \cdot \mathbf{x} + d$
 - *sphere*: $F(x, y, z) = x^2 + y^2 + z^2 - 1$
 - *cylinder*: $F(x, y, z) = x^2 + y^2 - 1 \quad 0 < z < 1$
- All points on the surface satisfy $F(x, y, z) = 0$
- Thus for ray $r(t)$ to intersect the surface $F(r(t)) = 0$
- “ t ” can be got by solving $F(\mathbf{S} + \mathbf{c}t_{hit}) = 0$

Ray Object Intersection

- Ray polygon intersection
 - Plug the ray equation into the implicit representation of the surface
 - Solve for " t "
 - Substitute for " t " to find point of intersection
 - Check if the point of intersection falls within the polygon

Ray Object Intersection

- Ray sphere intersection $|\mathbf{p} - \mathbf{p}_c|^2 = r^2$ $\mathbf{p} = (x, y, z), \mathbf{p}_c = (a, b, c)$
 - Implicit form of sphere given center (a, b, c) and radius r
- Intersection with $r(t)$ gives $|\mathbf{S} + \mathbf{c}t - \mathbf{p}_c|^2 = r^2$
- By the identity $|\mathbf{a} + \mathbf{b}|^2 = |\mathbf{a}|^2 + |\mathbf{b}|^2 + 2(\mathbf{a} \cdot \mathbf{b})$
 - Intersection equation is quadratic in "t"
$$|\mathbf{S} + \mathbf{c}t - \mathbf{p}_c|^2 - r^2 = t^2|\mathbf{c}|^2 + 2t\mathbf{c} \cdot (\mathbf{S} - \mathbf{p}_c) + (|\mathbf{S} - \mathbf{p}_c|^2 - r^2)$$
- Solving for "t" $t = -\mathbf{c} \cdot (\mathbf{S} - \mathbf{p}_c) \pm \sqrt{(\mathbf{c} \cdot (\mathbf{S} - \mathbf{p}_c))^2 - |\mathbf{c}|^2 (|\mathbf{S} - \mathbf{p}_c|^2 - r^2)}$
 - Real solutions, indicate one or two intersections
 - Negative solutions are behind the eye
 - If discriminant is negative, the ray missed the sphere

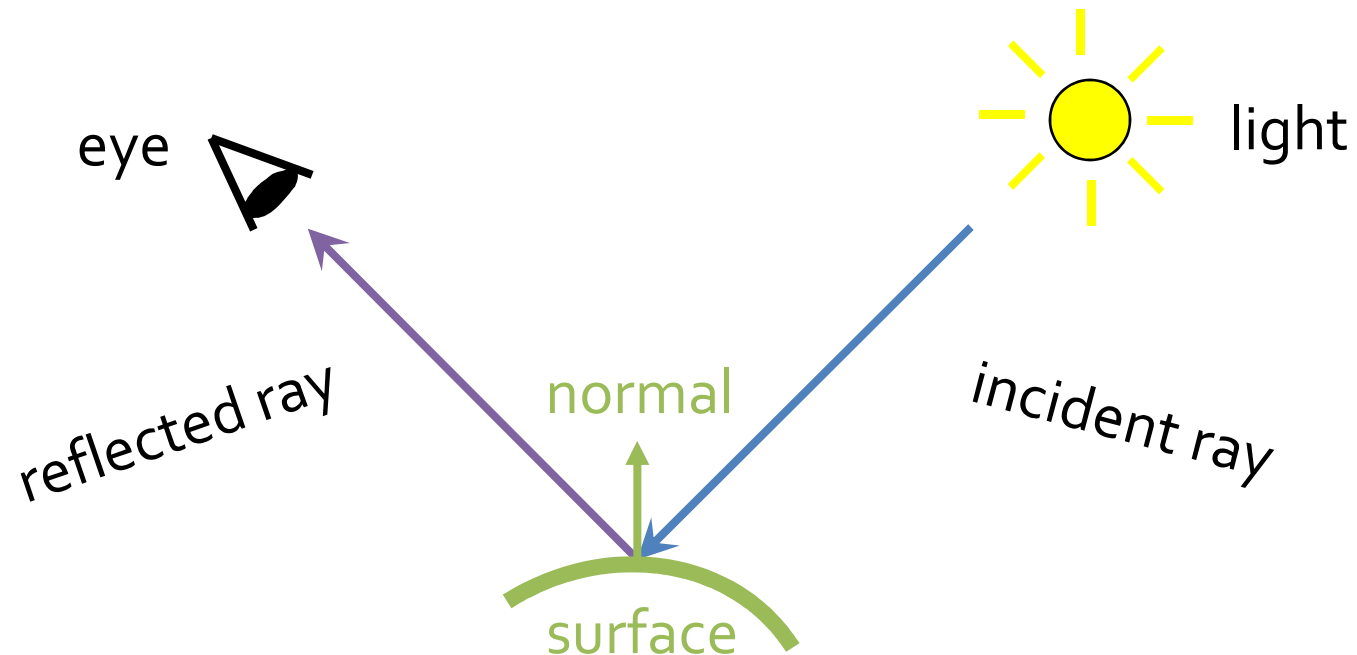
Ray-Casting Method

- based on geometric optics, tracing paths of light rays
- backward tracing of light rays
- suitable for complex, curved surfaces
- special case of ray-tracing algorithms
- efficient ray-surface intersection techniques necessary
 - intersection point
 - normal vector

Ray-Tracing

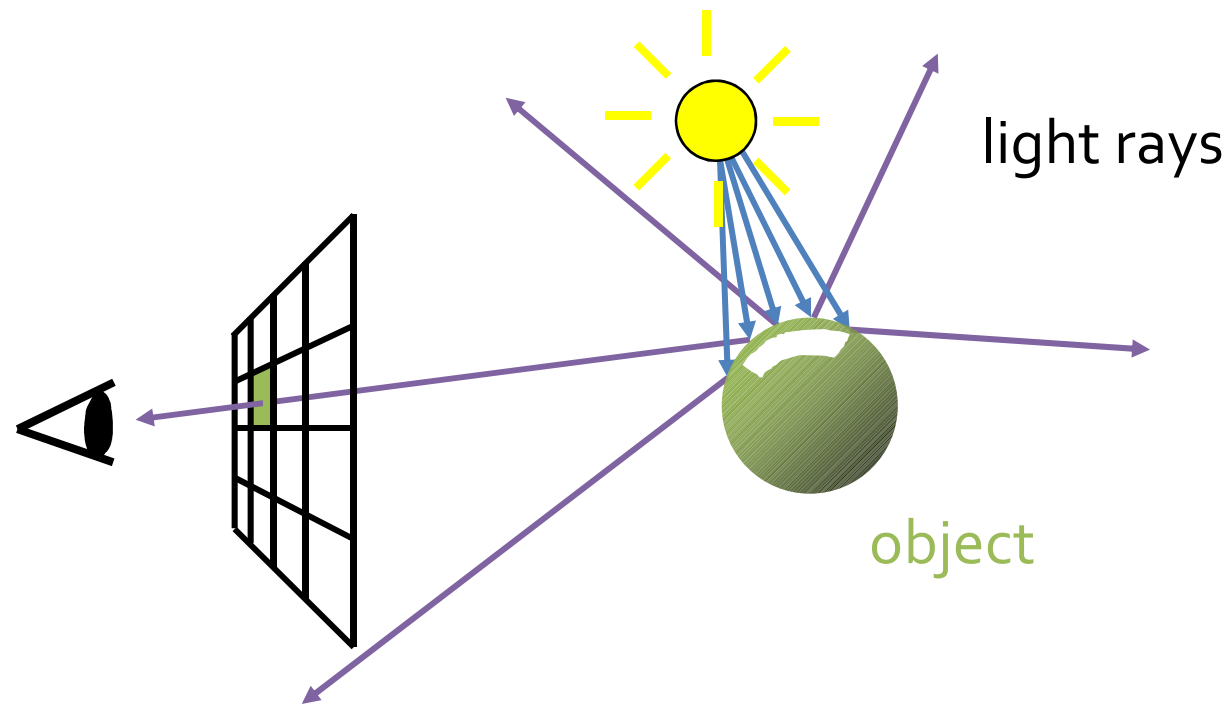
The Basic Idea

- Simulate light rays from light source to eye



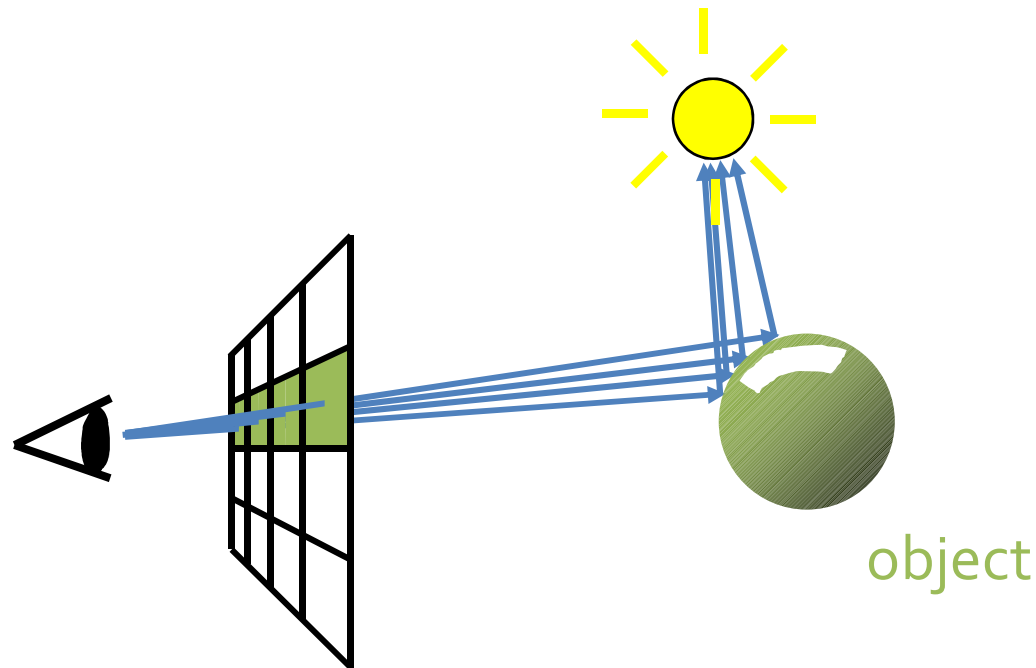
“Forward” Ray-Tracing

- Trace rays from light
- Lots of work for little return



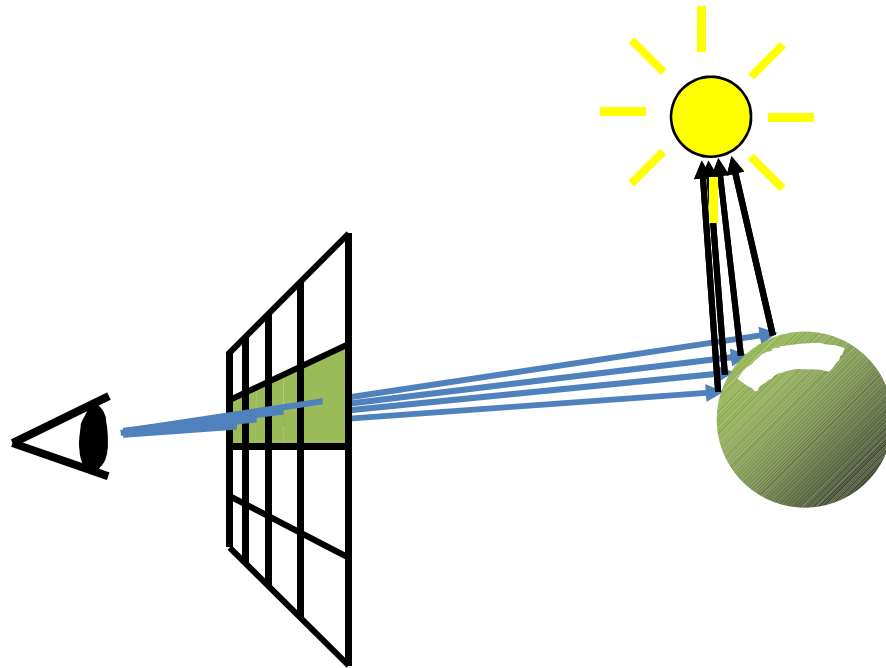
“Backward” Ray-Tracing

- Trace rays from eye instead
- Do work where it matters
- *This is what most people mean by “ray tracing”.*



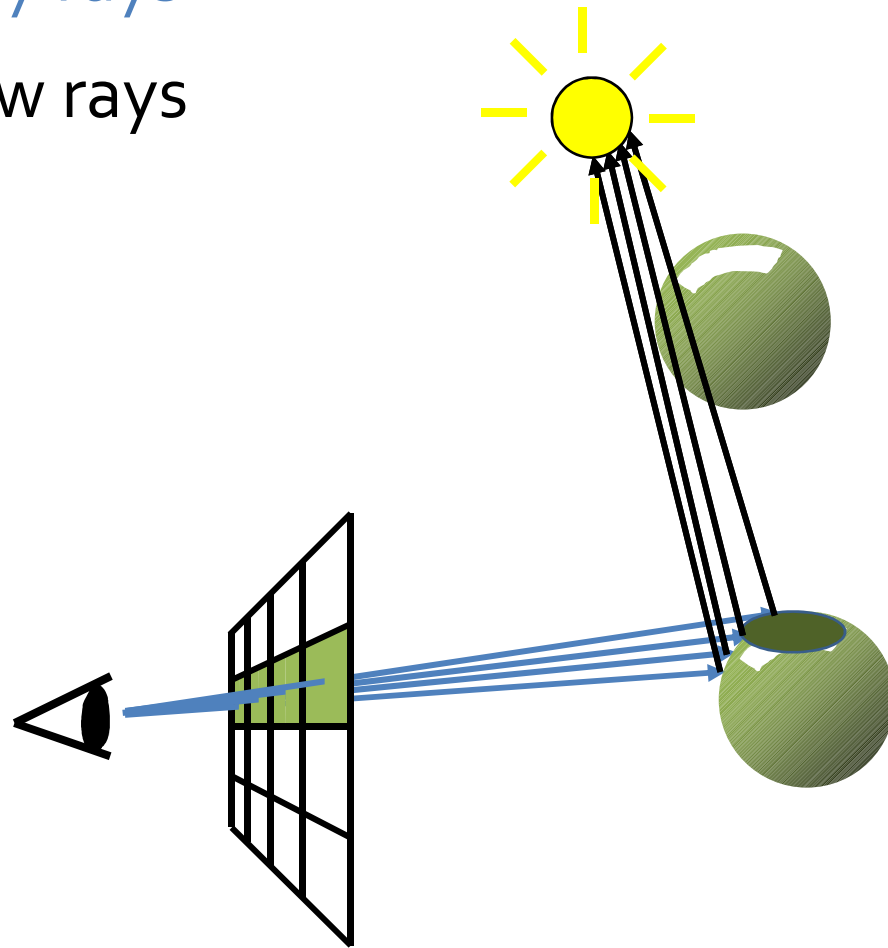
Types of Rays

- Primary rays
- Shadow rays



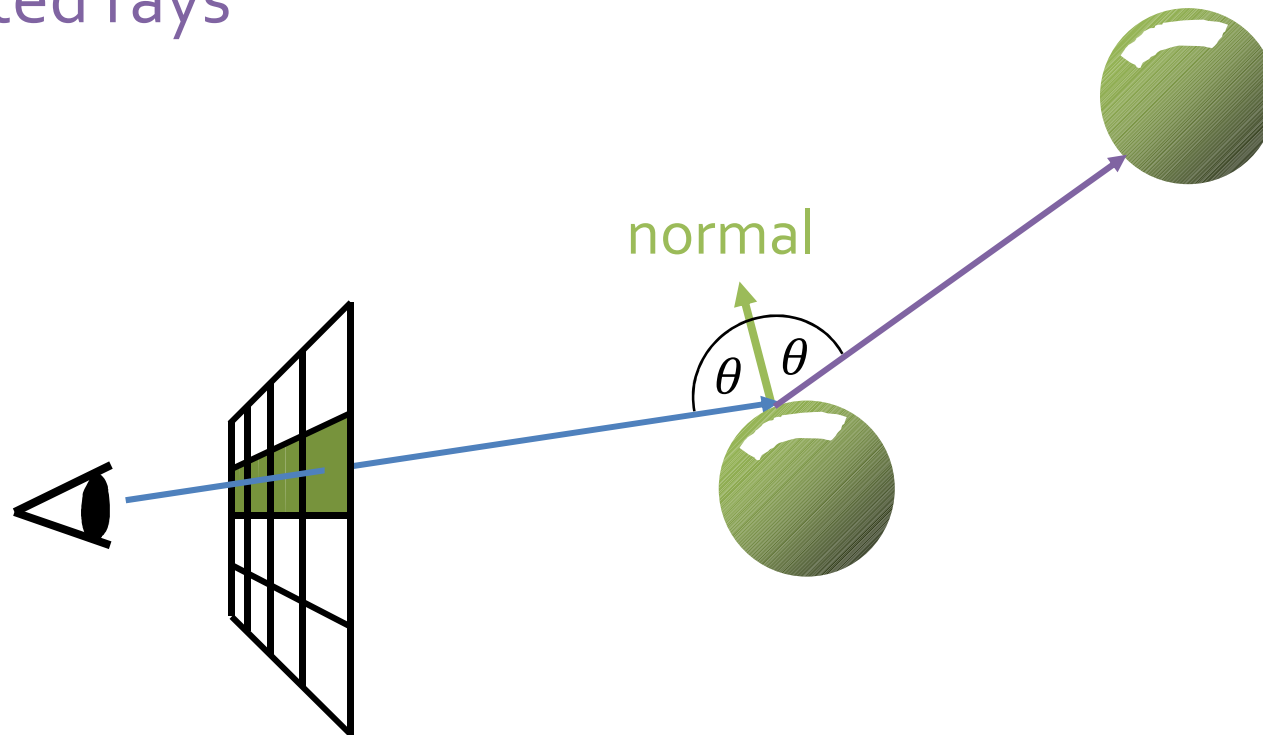
Shadow Rays

- Primary rays
- Shadow rays



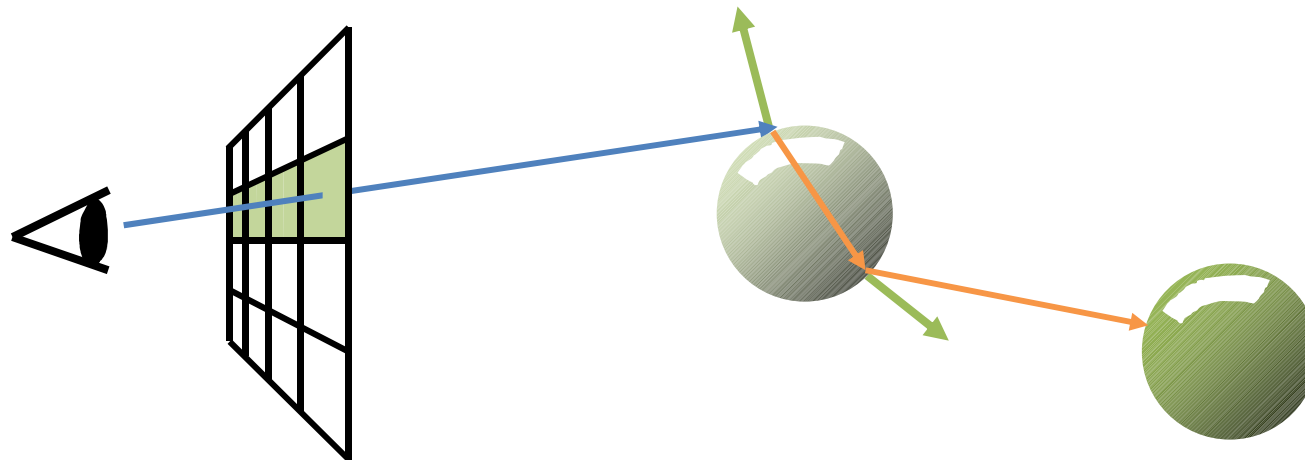
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays



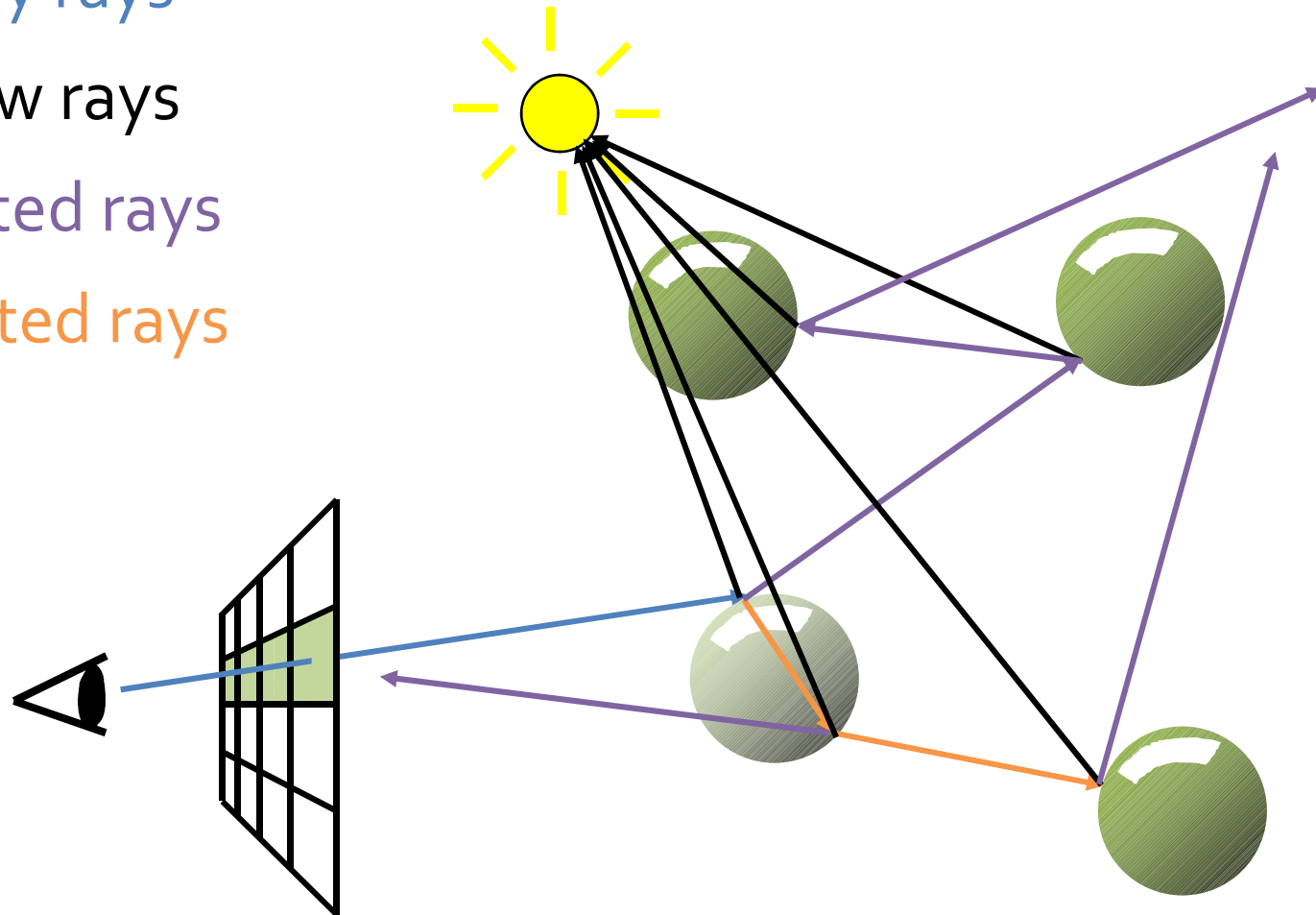
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays
- Refracted rays



Types of Rays

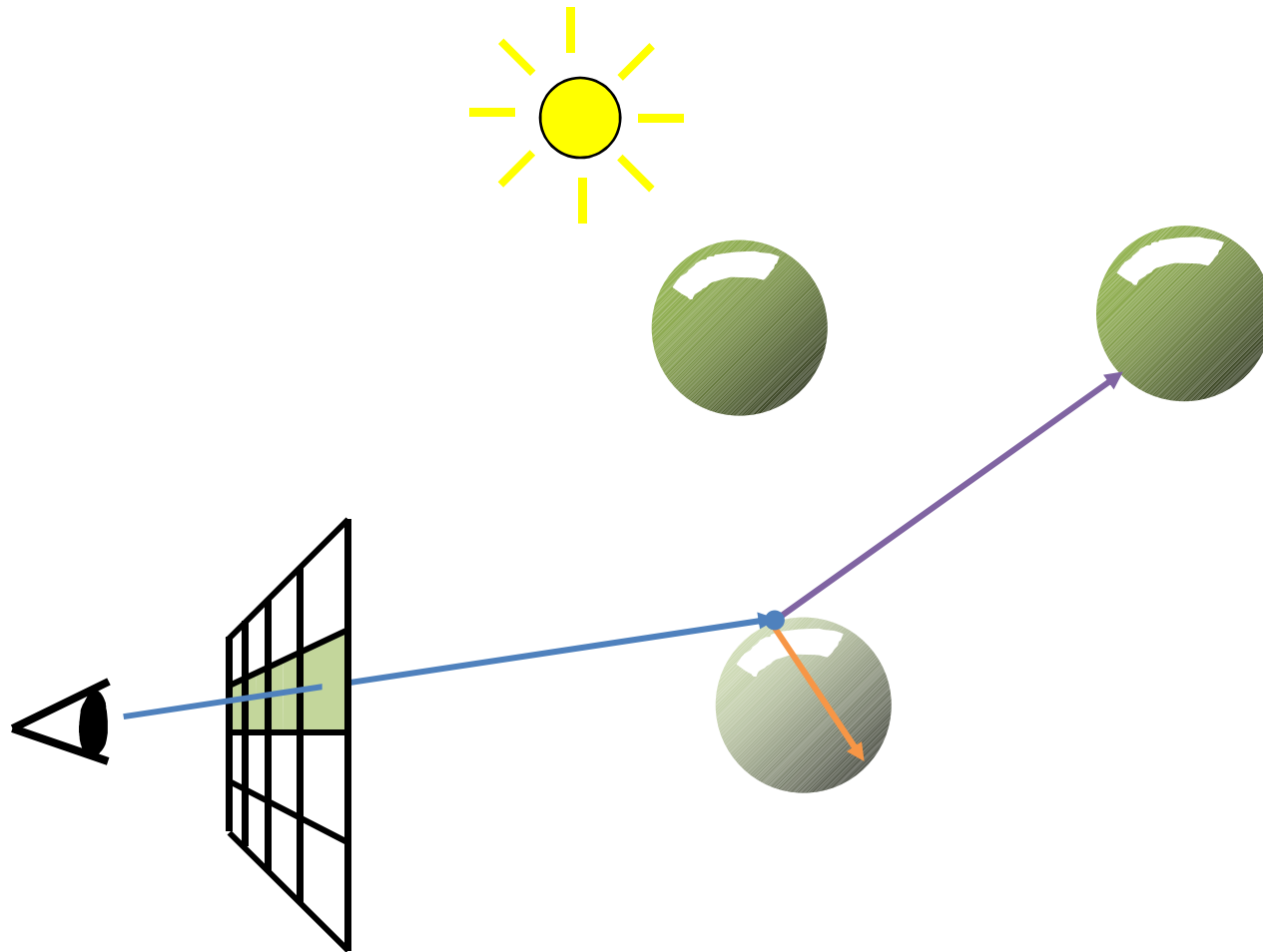
- Primary rays
- Shadow rays
- Reflected rays
- Refracted rays



Lighting

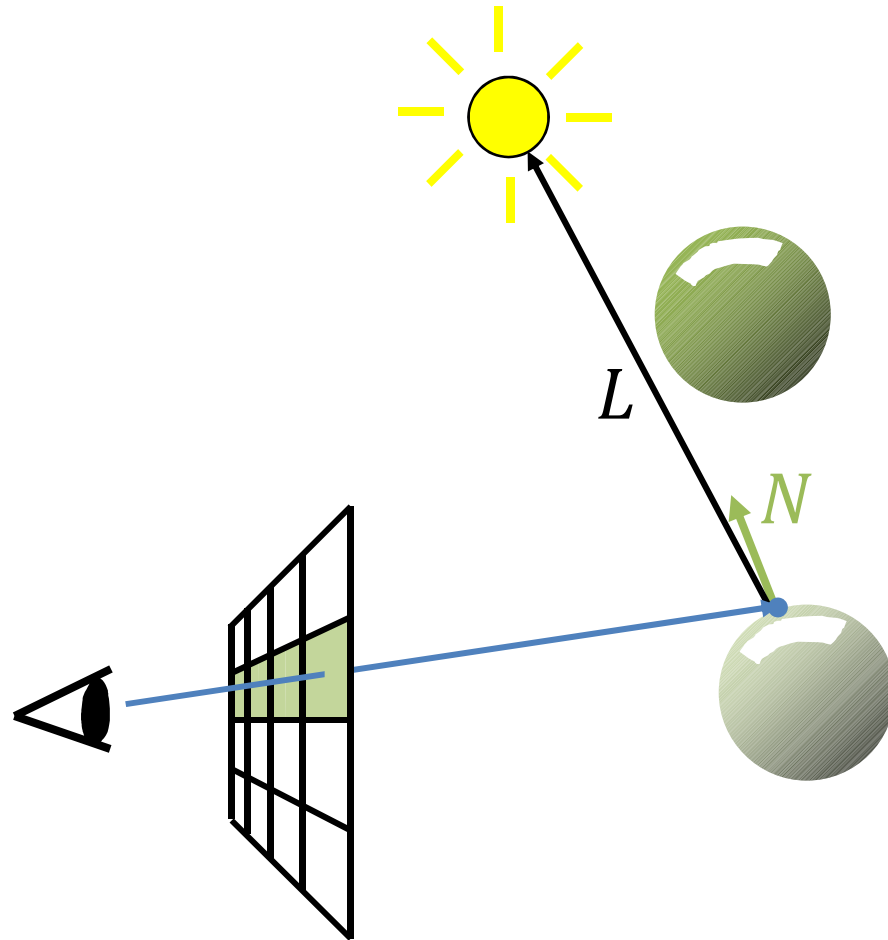
Lighting

- $C = C_{local} + C_{reflected} + C_{transmitted}$



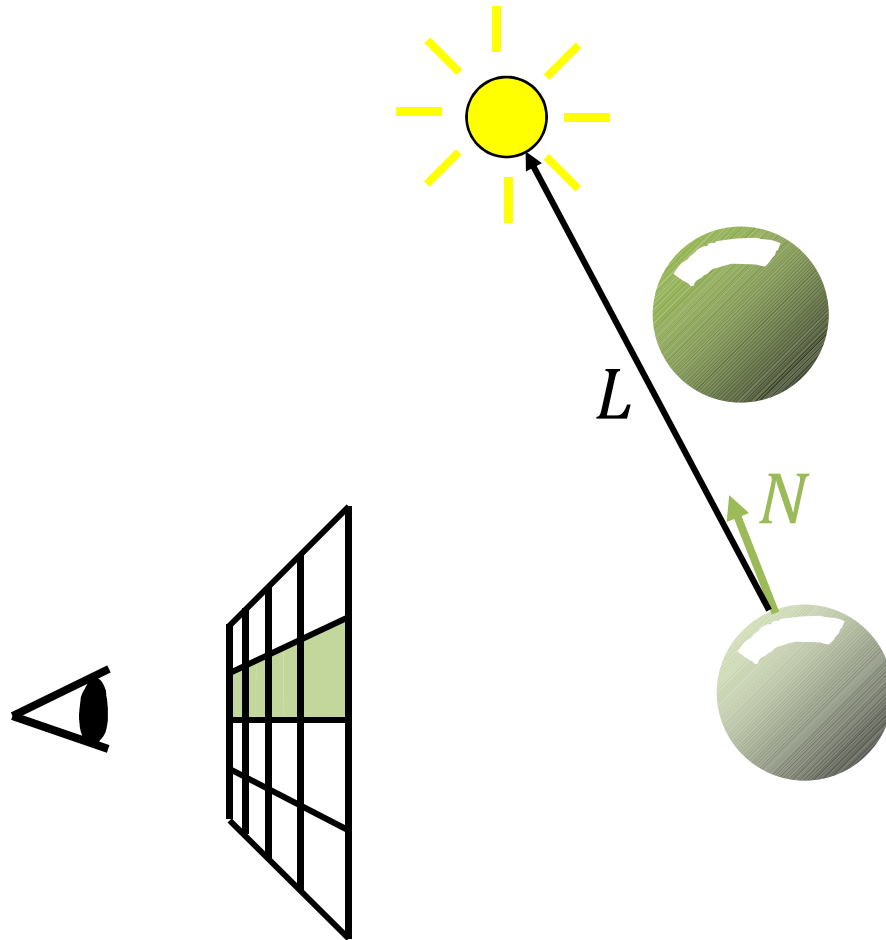
Local – Phong Illumination

- $C_{local} = C_{ambient} + C_{diffuse} + C_{specular}$



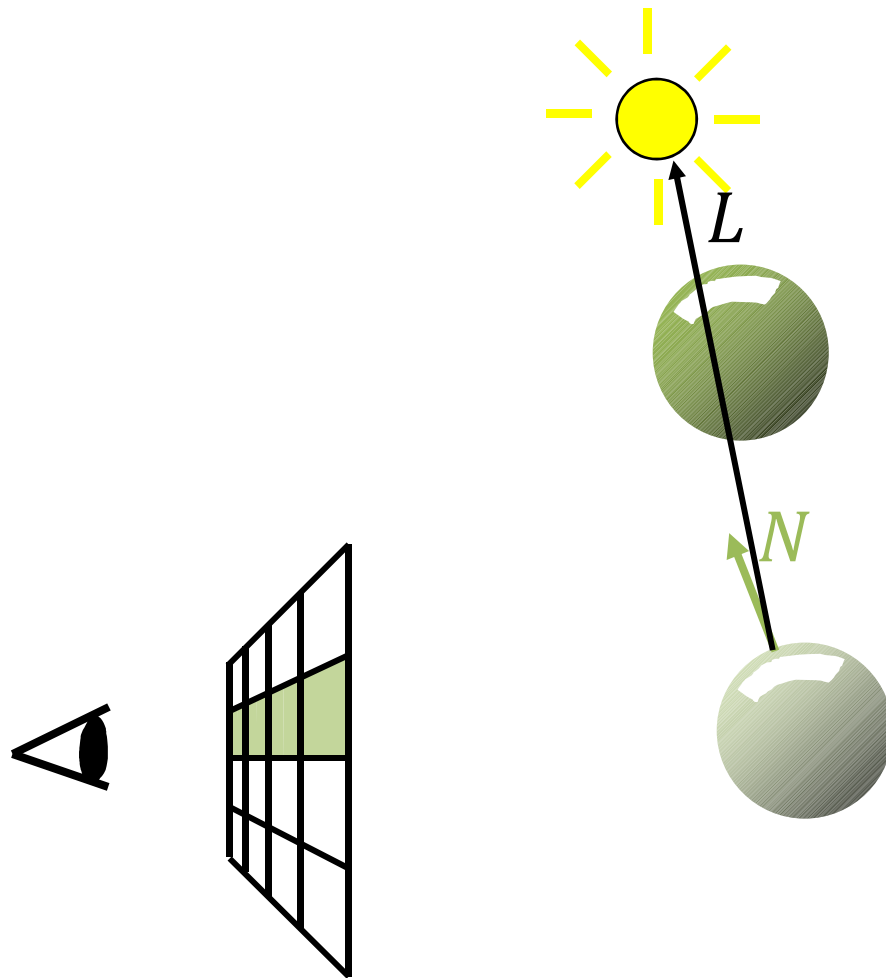
Diffuse (Lambert)

- $C_{local} = \max(0, N \cdot L) * Color_{object} * Color_{light}$



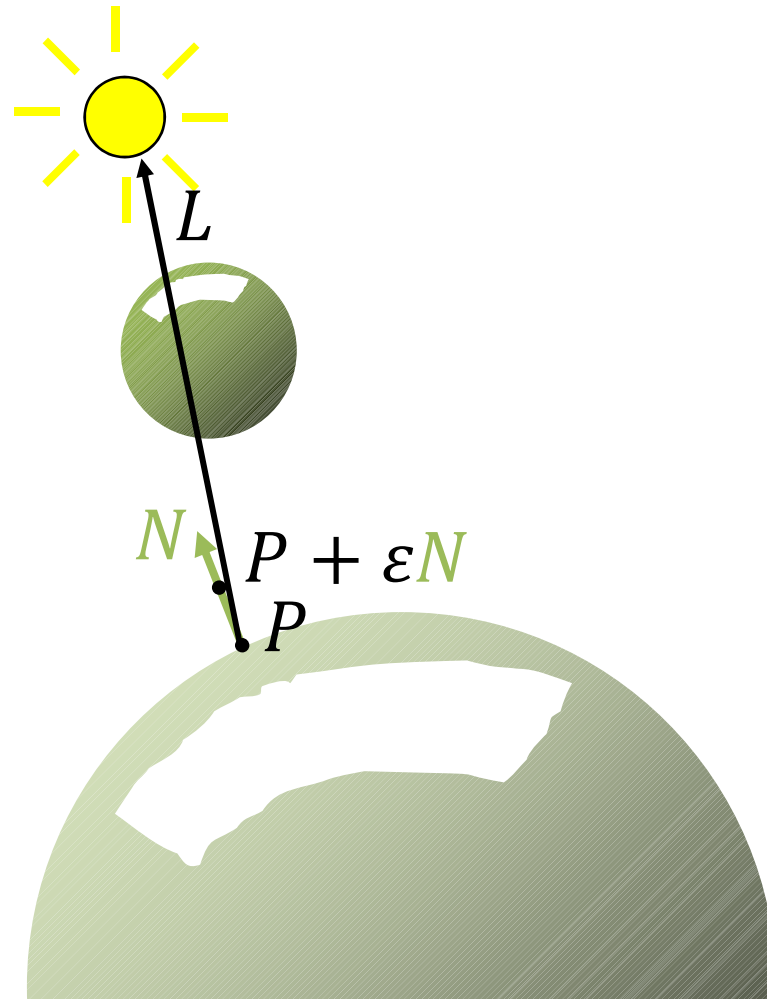
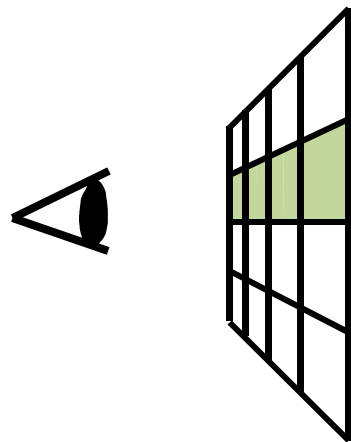
Adding Shadows

- $C_{local} = C_{ambient}$



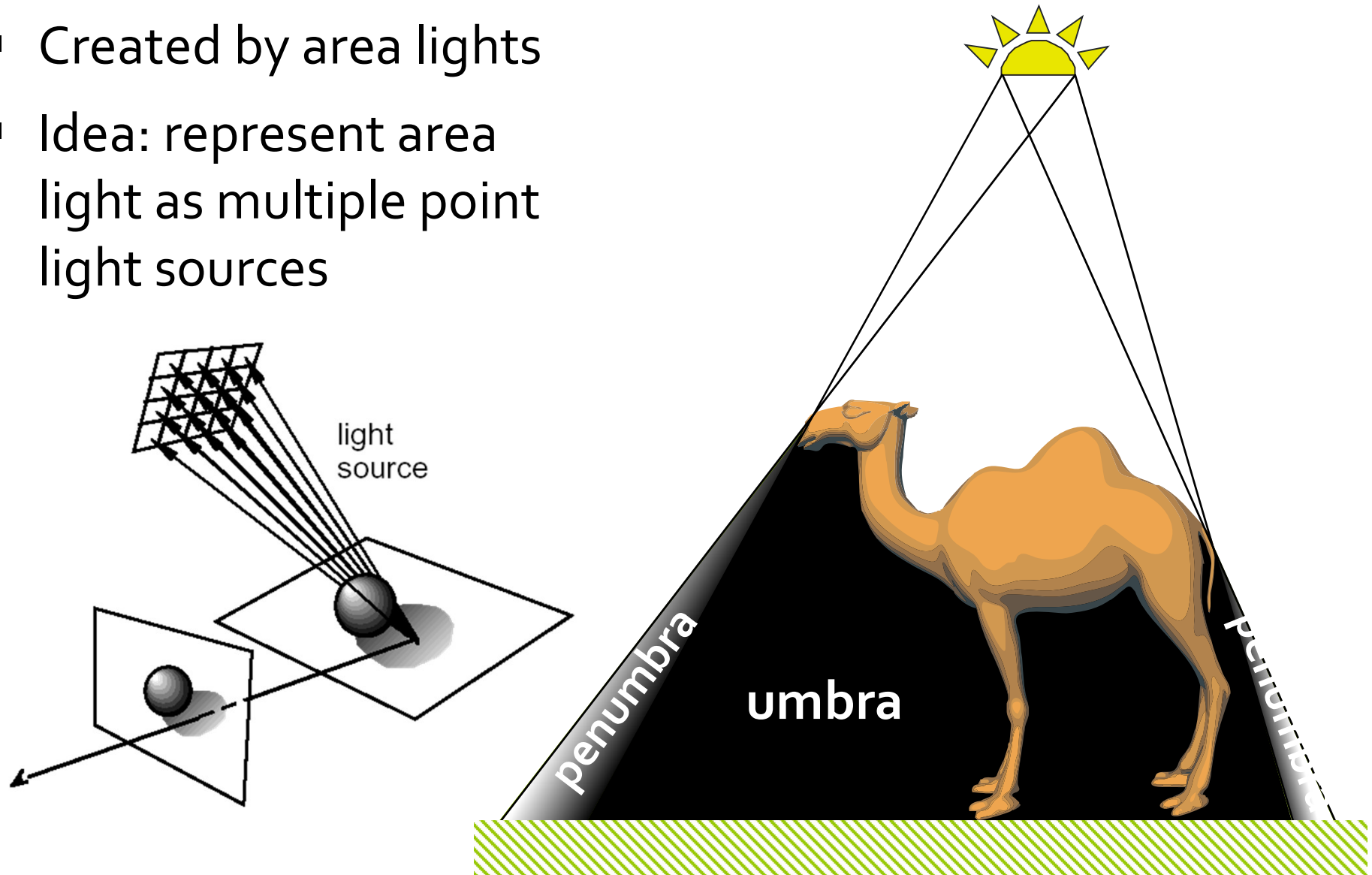
Adding Shadows

- “Self-Shadowing”
 - Intersection of shadow feeler with object itself
 - Move start point of the shadow ray away by a small amount

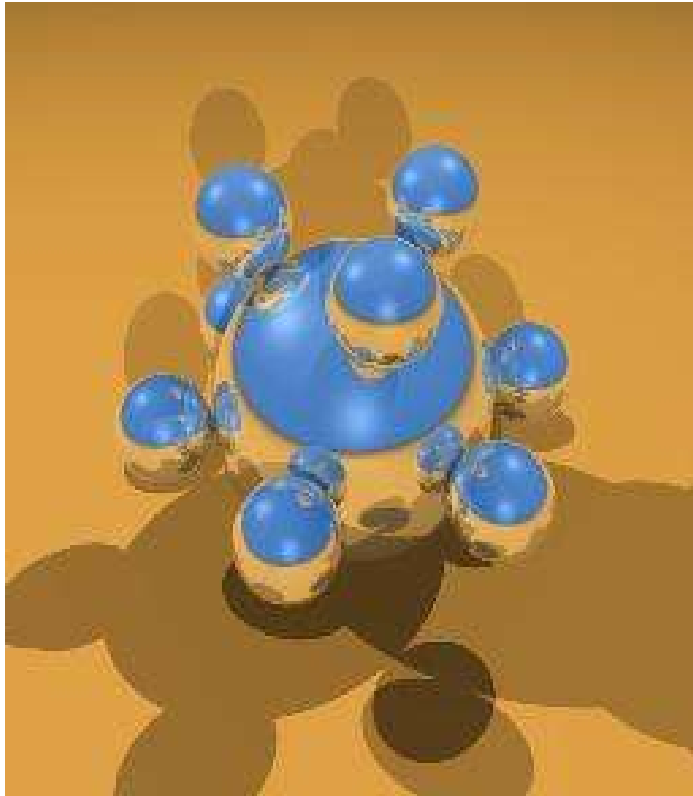


Soft Shadows

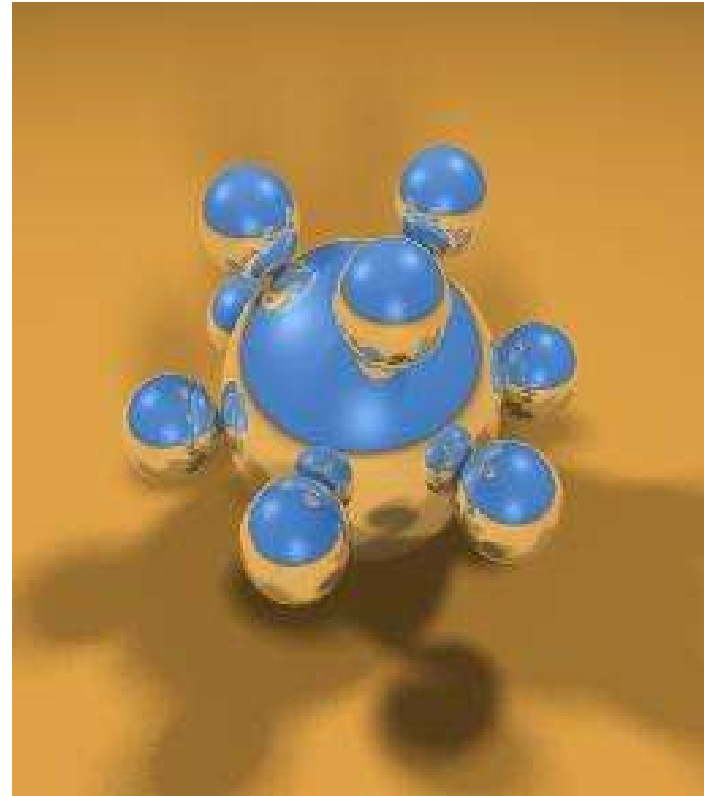
- Created by area lights
- Idea: represent area light as multiple point light sources



Soft Shadow Example



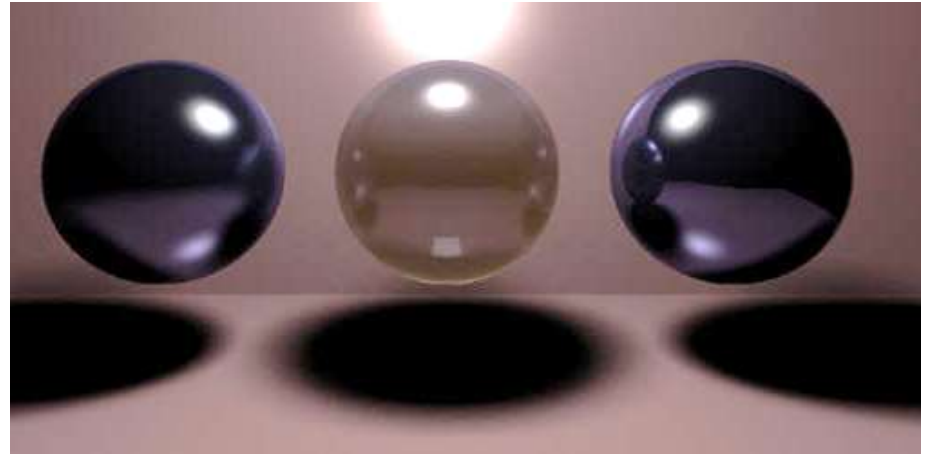
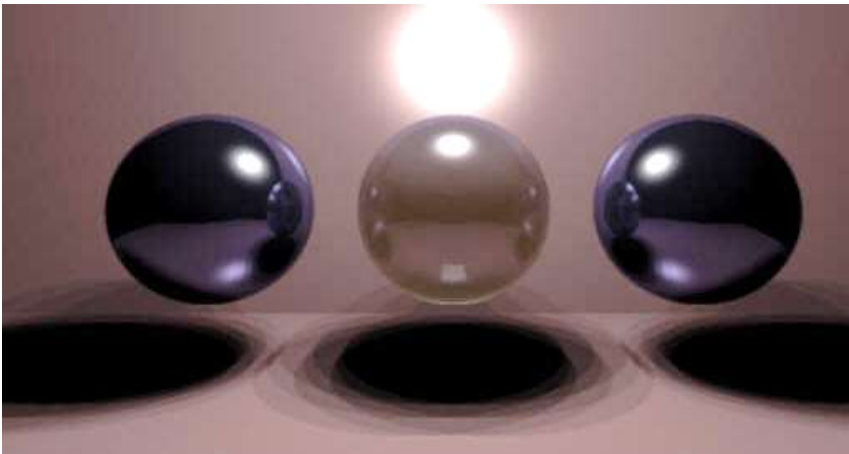
Hard shadow



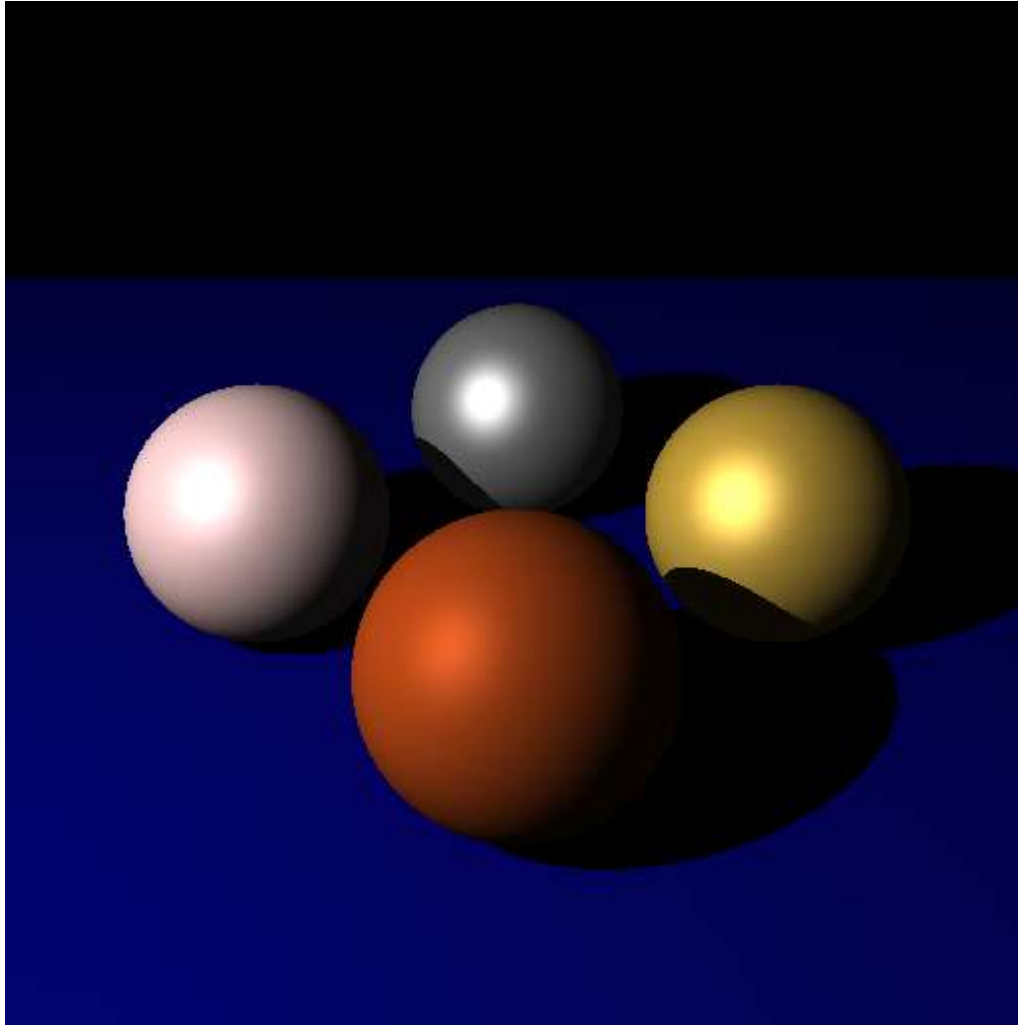
Soft shadow

Area Light Sources

- Shadow Feelers to multiple points on light source
 - Left: 9 shadow rays (3*3 grid)
 - Right: 128*128 grid

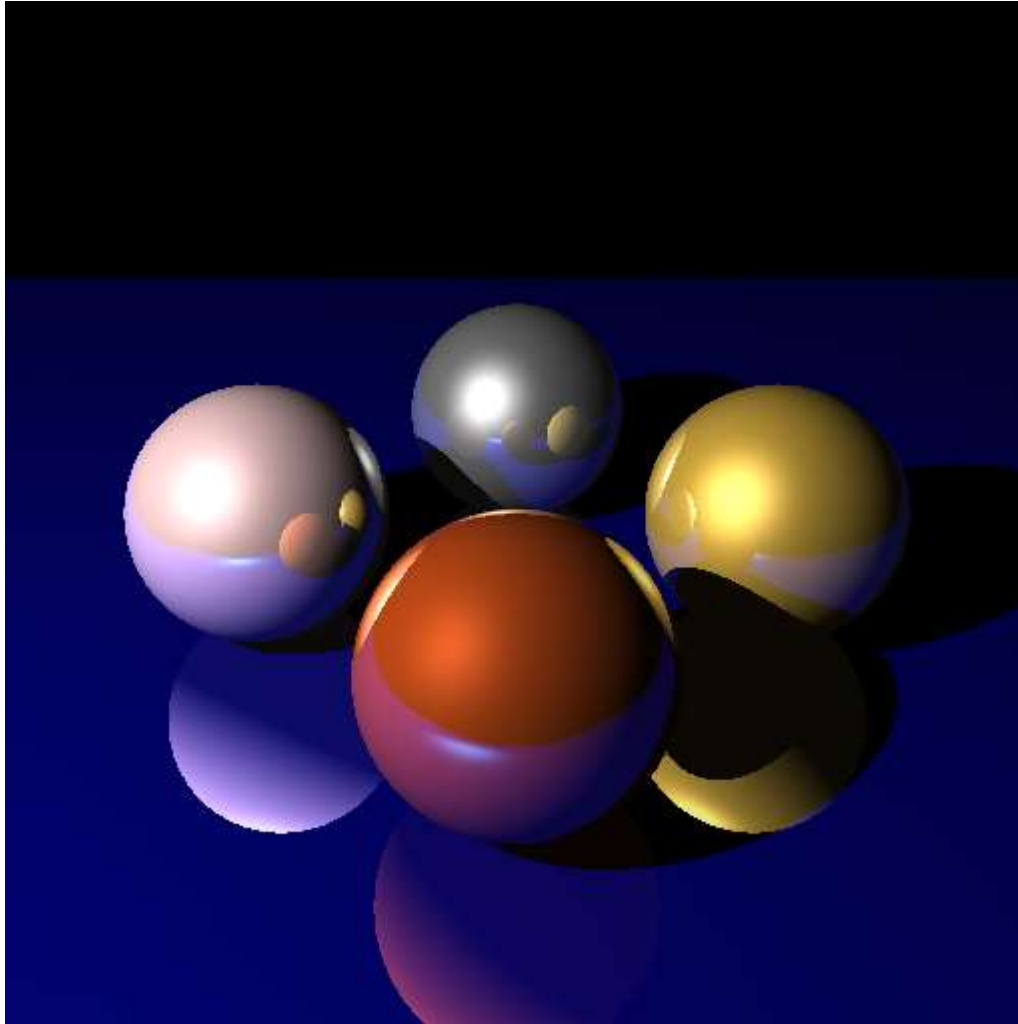


No Reflection



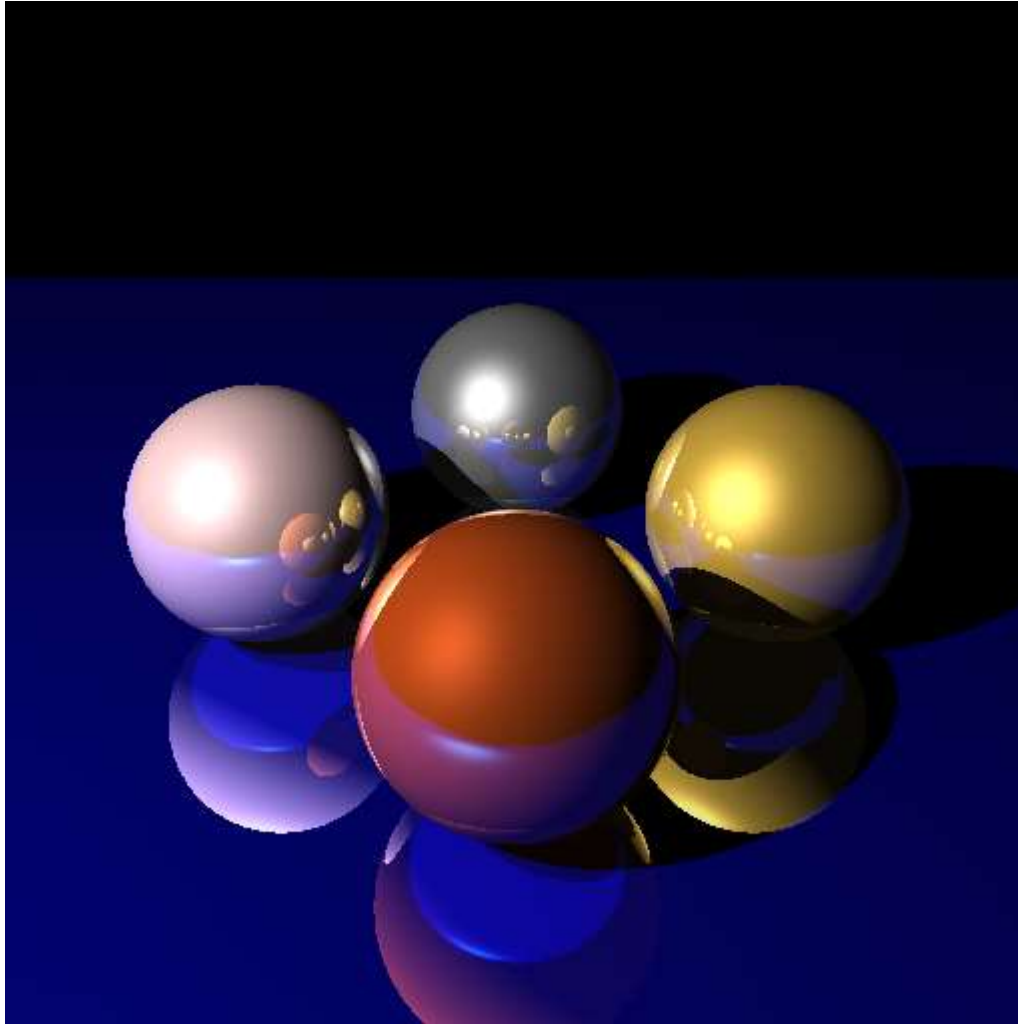
Created by David Derman – CISC 440

Reflection (1)



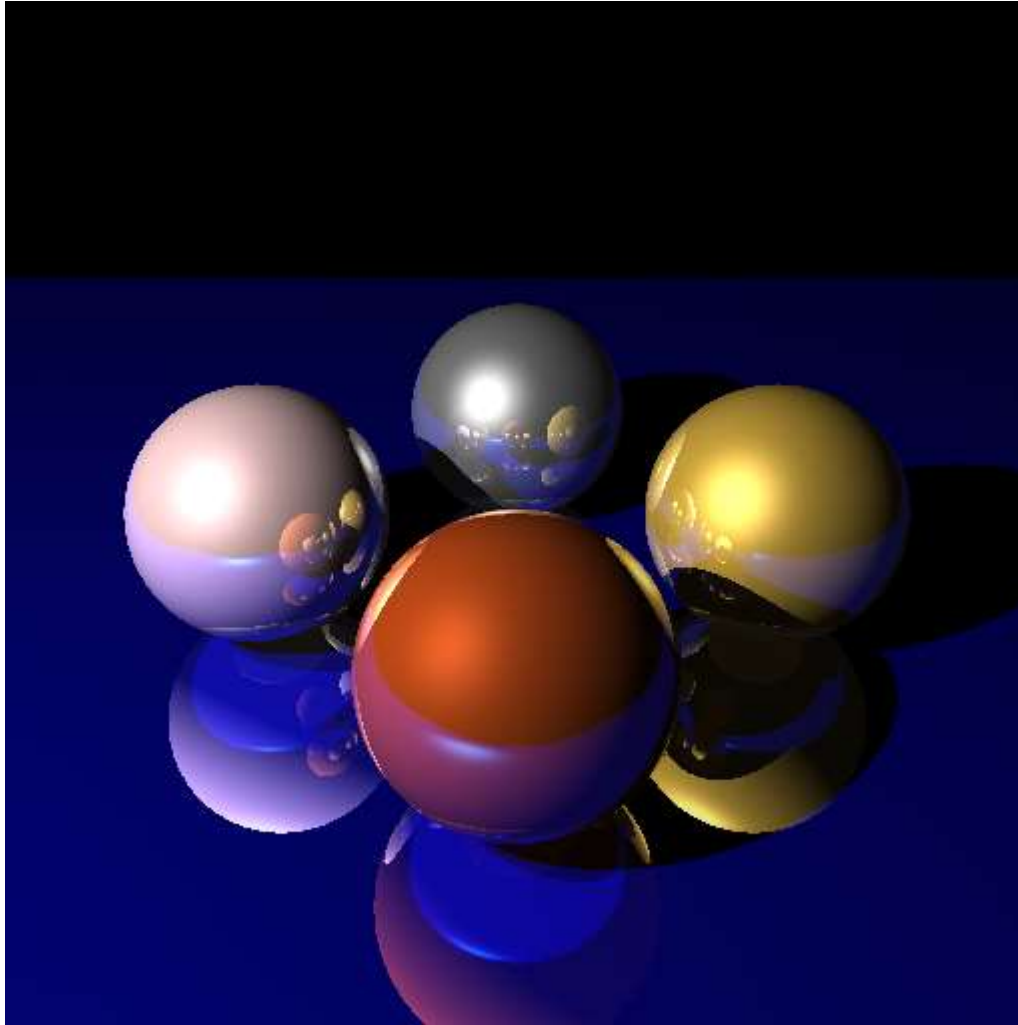
Created by David Derman – CISC 440

Reflection (2)



Created by David Derman – CISC 440

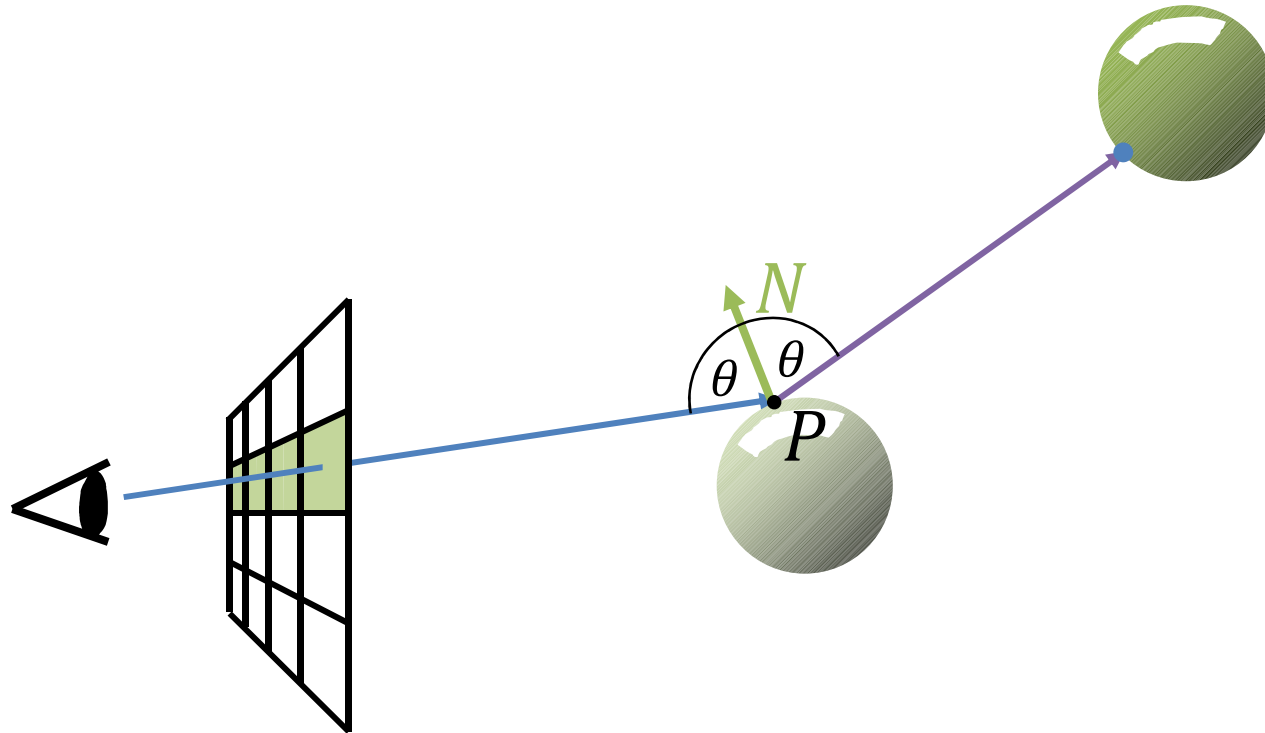
Reflection (3)



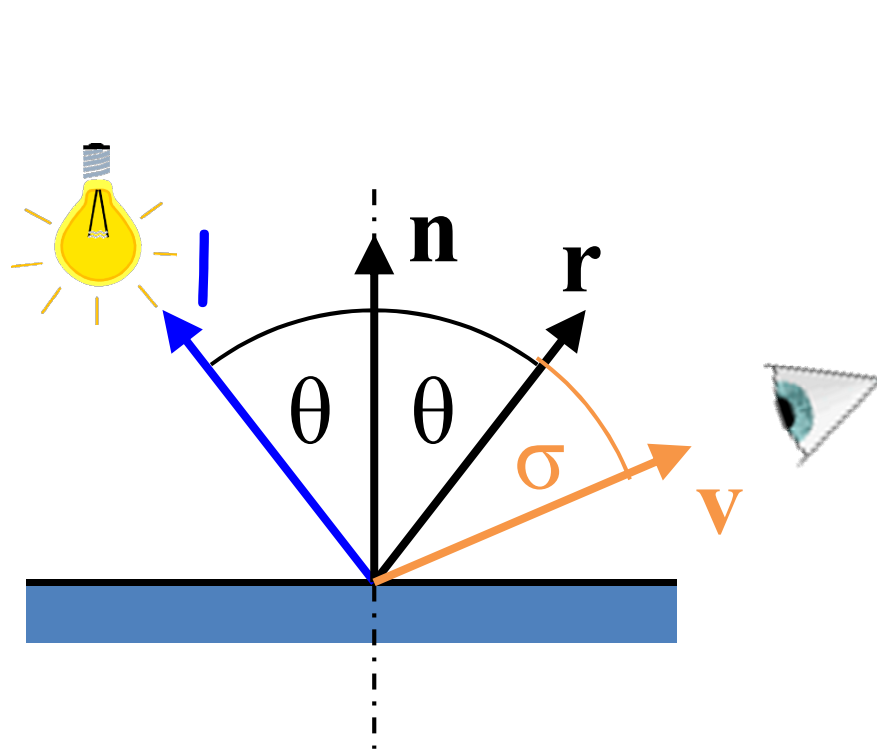
Created by David Derman – CISC 440

Reflection

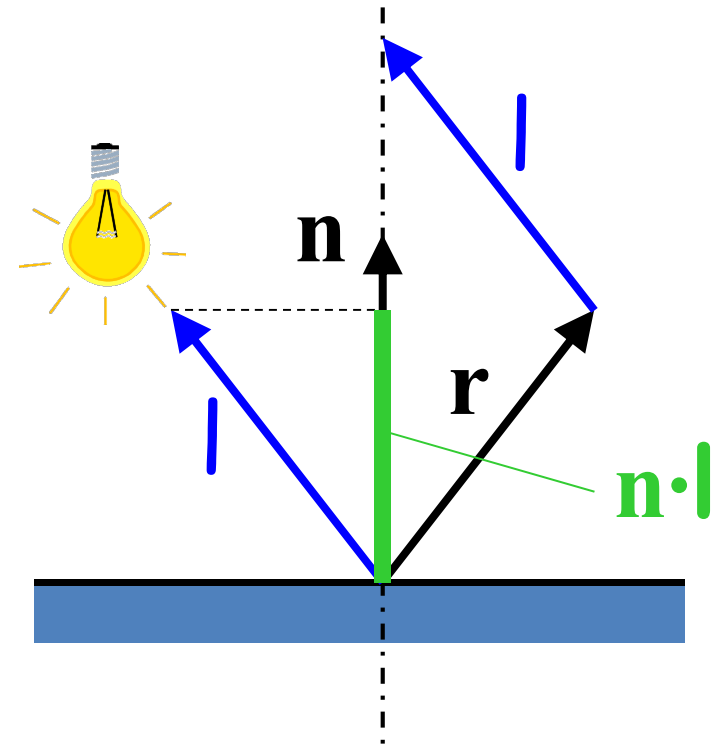
- $C_{reflected} = C_{\text{intersect}(P, \text{reflect}(\text{dir}, N))}$



Reflection direction



$$L_{\text{spec}} = k_s \cdot S \cdot (v \cdot r)^p$$



$$r + l = (2n \cdot l)n$$

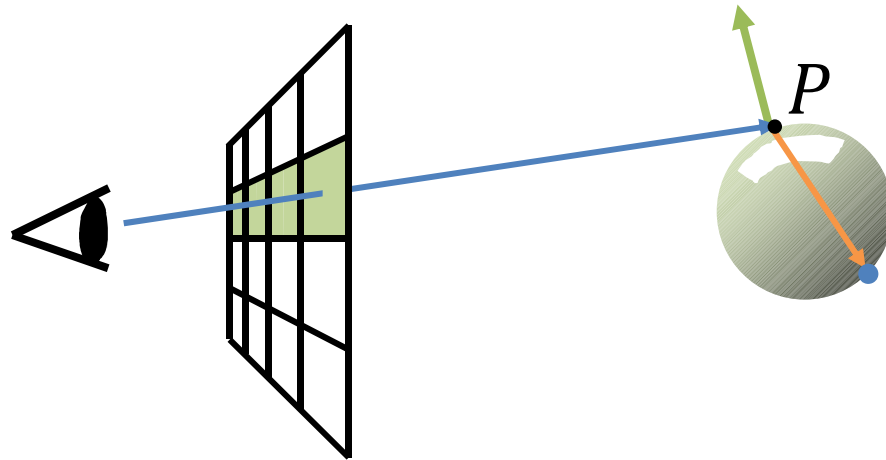
$$r = (2n \cdot l)n - l$$

Refraction



Refraction

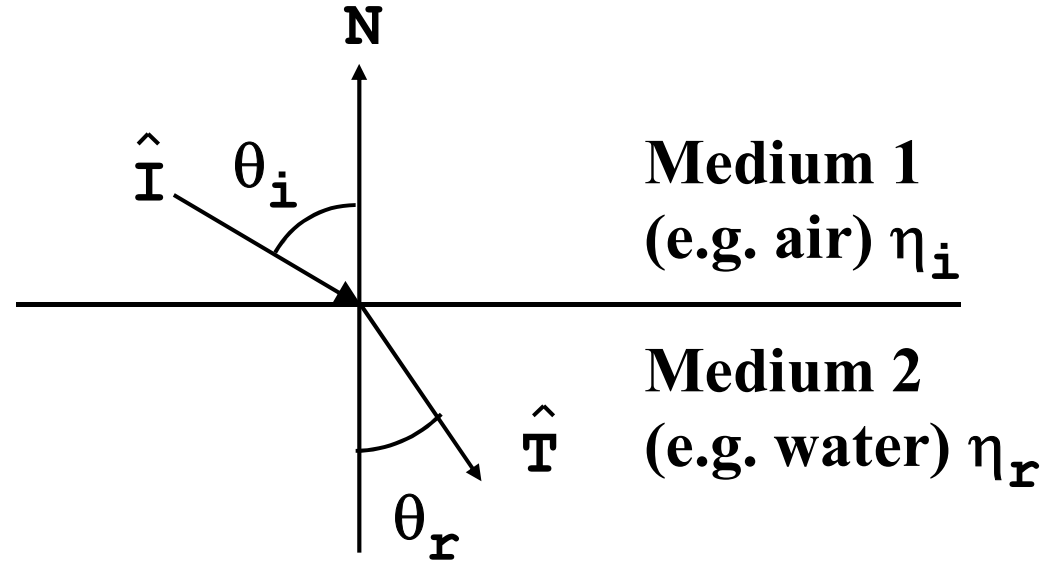
- $C_{refracted} = C_{intersect}(P, \text{refract}(\text{dir}, N))$



Refraction

- Keep track of medium (air, glass, etc)
- Need *index of refraction* (η)
- Need solid objects

$$\frac{\sin(\theta_i)}{\sin(\theta_r)} = \frac{\eta_2}{\eta_1}$$



Refraction

- Decomposing the incident ray (\mathbf{u})

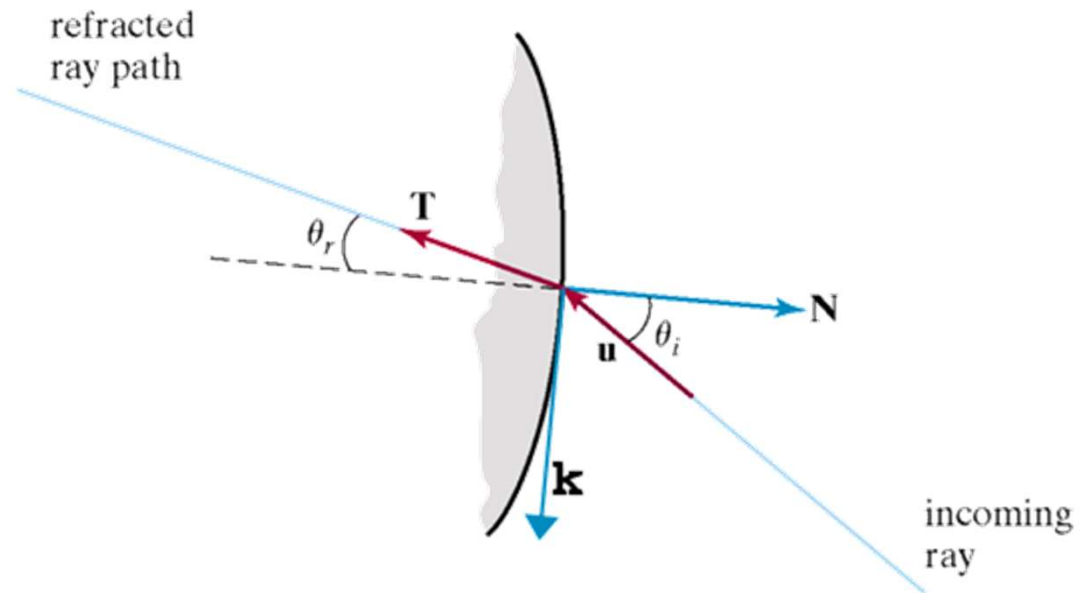
$$\begin{aligned}\mathbf{u} &= (\mathbf{u} \cdot \mathbf{n})(-\mathbf{n}) + (\mathbf{u} \cdot \mathbf{k})(-\mathbf{k}) \\ &= -(\mathbf{u} \cdot \mathbf{k})\mathbf{k} - (\mathbf{u} \cdot \mathbf{n})\mathbf{n} \\ &= -(\sin \theta_i)\mathbf{k} - (\cos \theta_i)\mathbf{n}\end{aligned}$$

- Decomposing the refracted ray (\mathbf{T})

$$\begin{aligned}\mathbf{T} &= (\mathbf{T} \cdot \mathbf{n})(-\mathbf{n}) + (\mathbf{T} \cdot \mathbf{k})(-\mathbf{k}) \\ &= -(\mathbf{T} \cdot \mathbf{k})\mathbf{k} - (\mathbf{T} \cdot \mathbf{n})\mathbf{n} \\ &= -(\sin \theta_r)\mathbf{k} - (\cos \theta_r)\mathbf{n}\end{aligned}$$

- Solving for \mathbf{k} from \mathbf{u}

$$\mathbf{k} = -\frac{1}{\sin \theta_i}(\mathbf{u} + \cos \theta_i \mathbf{n})$$



Refraction

- Substituting in **T**

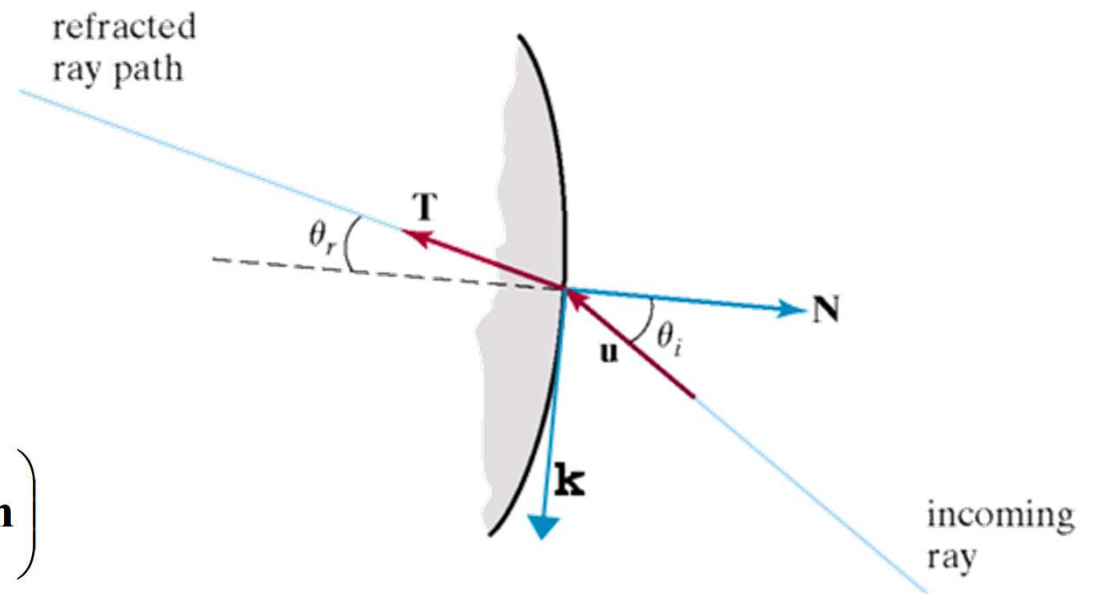
$$\mathbf{T} = -(\cos \theta_r) \mathbf{n} + \frac{\sin \theta_r}{\sin \theta_i} (\mathbf{u} + (\cos \theta_i) \mathbf{n})$$

- From Snell's Law

$$\frac{\sin \theta_r}{\sin \theta_i} = \frac{n_i}{n_r}$$

- Solving for **T**

$$\begin{aligned} \mathbf{T} &= -(\cos \theta_r) \mathbf{n} + \frac{n_i}{n_r} (\mathbf{u} + (\cos \theta_i) \mathbf{n}) \\ &= \frac{n_i}{n_r} \mathbf{u} + \left(\frac{n_i}{n_r} (\cos \theta_i) \mathbf{n} - (\cos \theta_r) \mathbf{n} \right) \\ &= \frac{n_i}{n_r} \mathbf{u} - \left(\cos \theta_r - \frac{n_i}{n_r} \cos \theta_i \right) \mathbf{n} \end{aligned}$$



GPU acceleration structures for real-time ray-tracing

