Ray-Casting
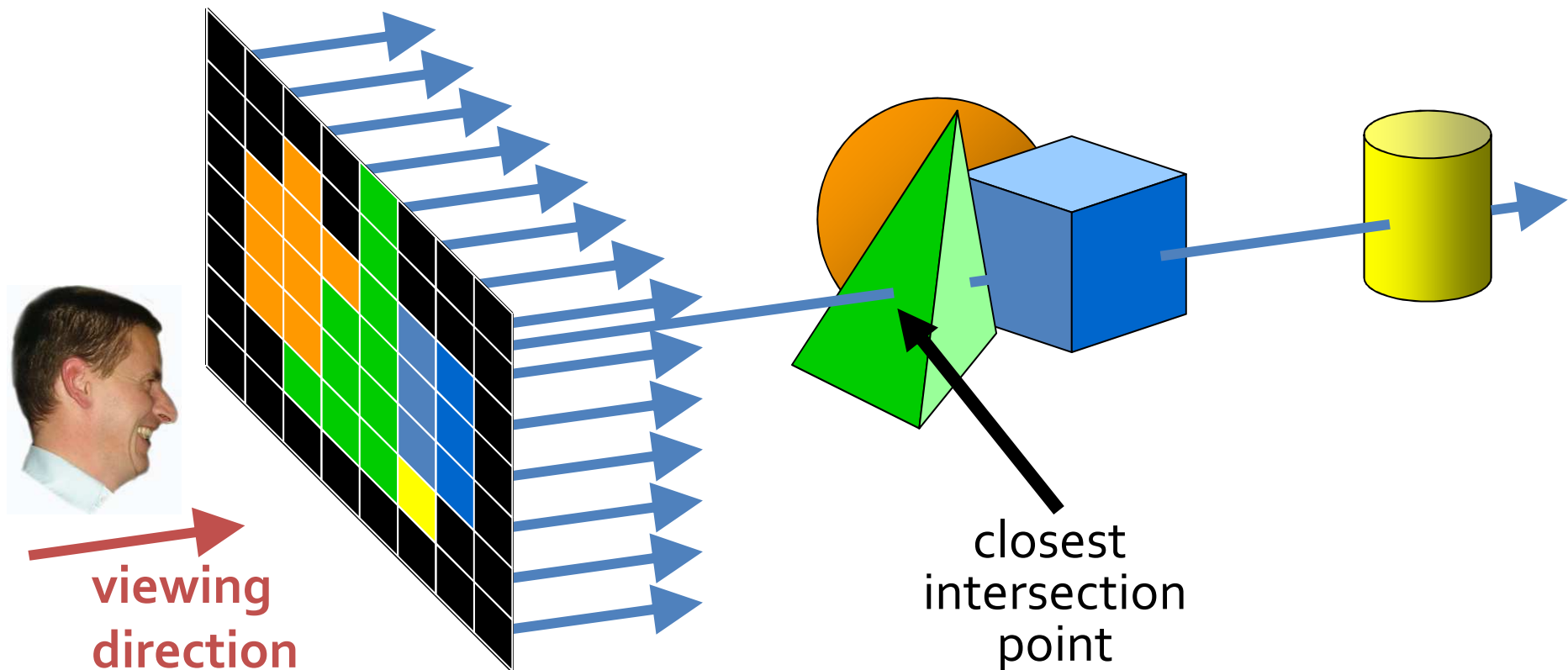
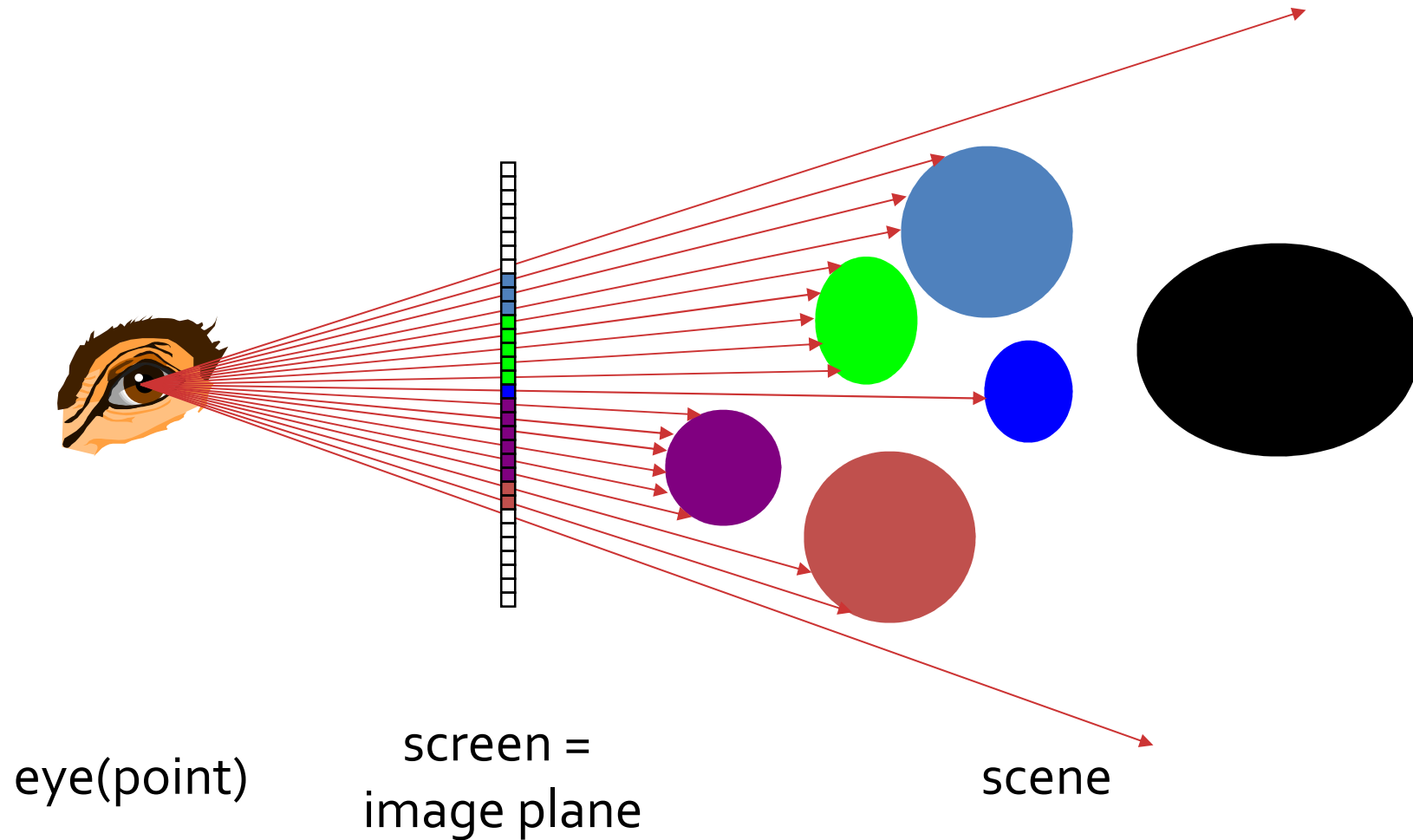# Ray-Casting Method

- line-of-sight of each pixel is intersected with all surfaces

- take closest intersected surface

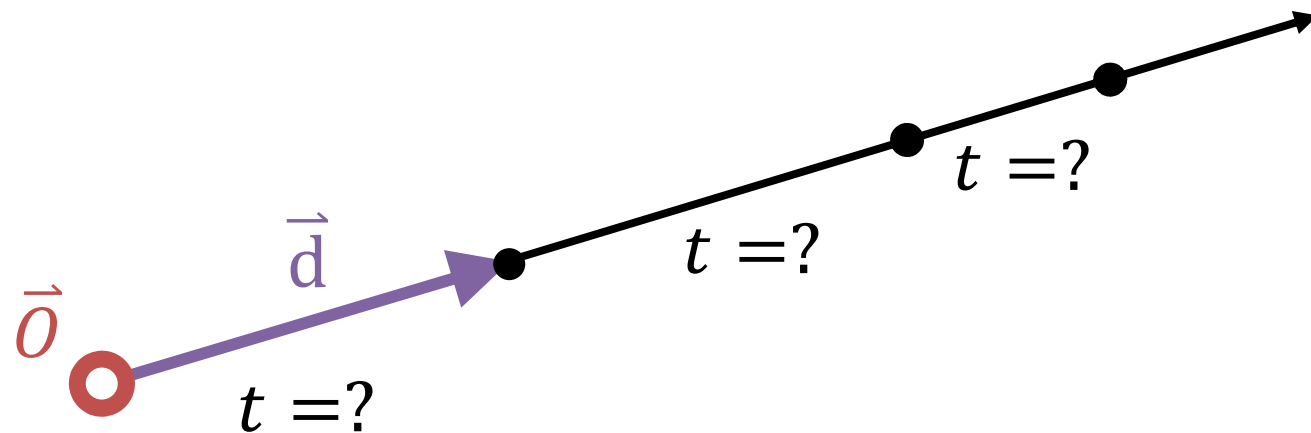viewing direction

closest intersection point

Generating Rays

- Trace a ray for each pixel in the image plane
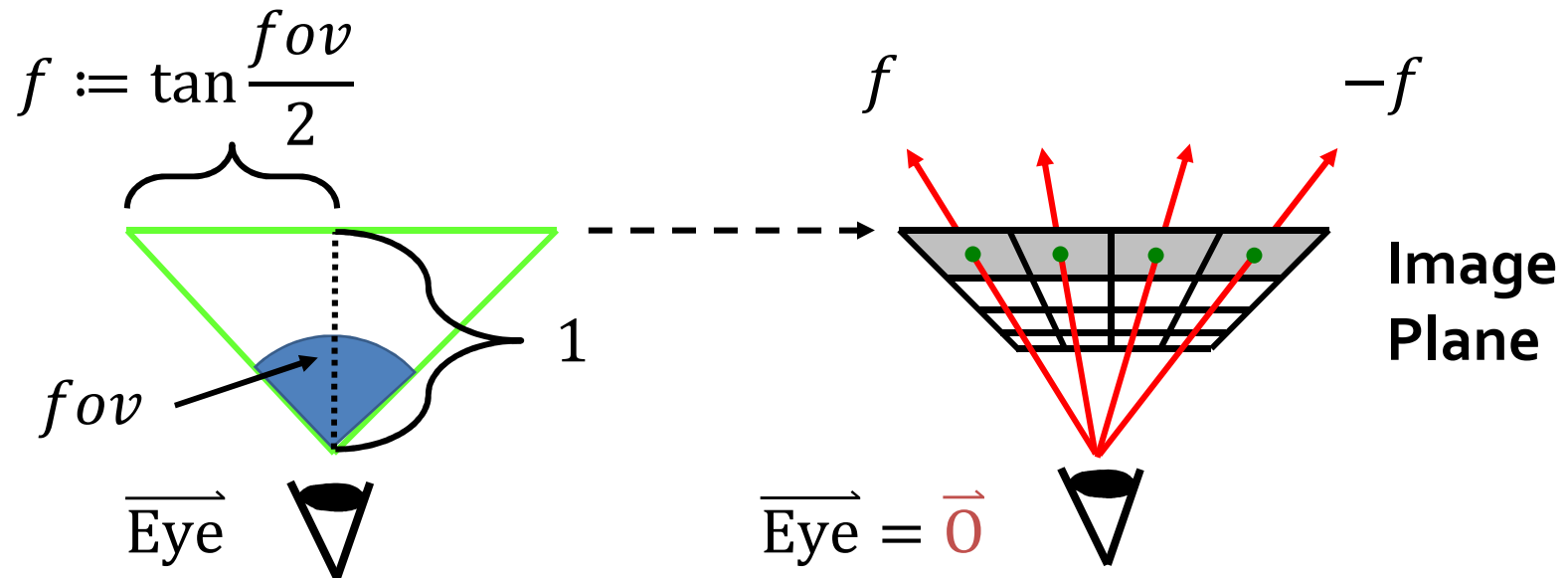

eye(point)

screen =
image plane

scene

Ray Parametric Form

- Ray expressed as function of a single parameter t

$$\vec{P} = \vec{O} + t\vec{d}$$

$$= \begin{pmatrix} O_x \\ O_y \\ O_z \end{pmatrix} + t \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$$

Generating Rays – Top View

- Trace a ray for each pixel in the image plane

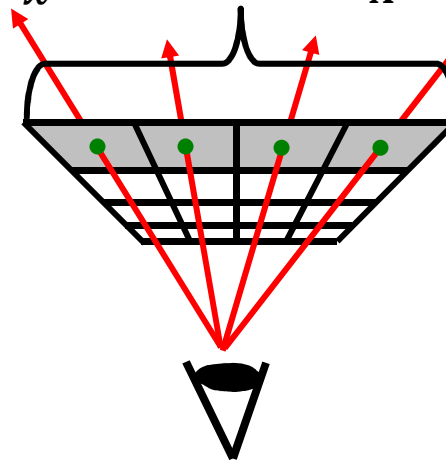$$f := \tan\frac{fov}{2}$$

fov

1

$\overrightarrow{\text{Eye}}$

f

$-f$

Image Plane

$\overrightarrow{\text{Eye}} = \vec{0}$

Generating Rays – Top View

- Trace a ray for each pixel in the image plane

- $d_x(x) = \dfrac{2fx}{resolution_x} - f = \dfrac{f(2x - resolution_x)}{resolution_x}$

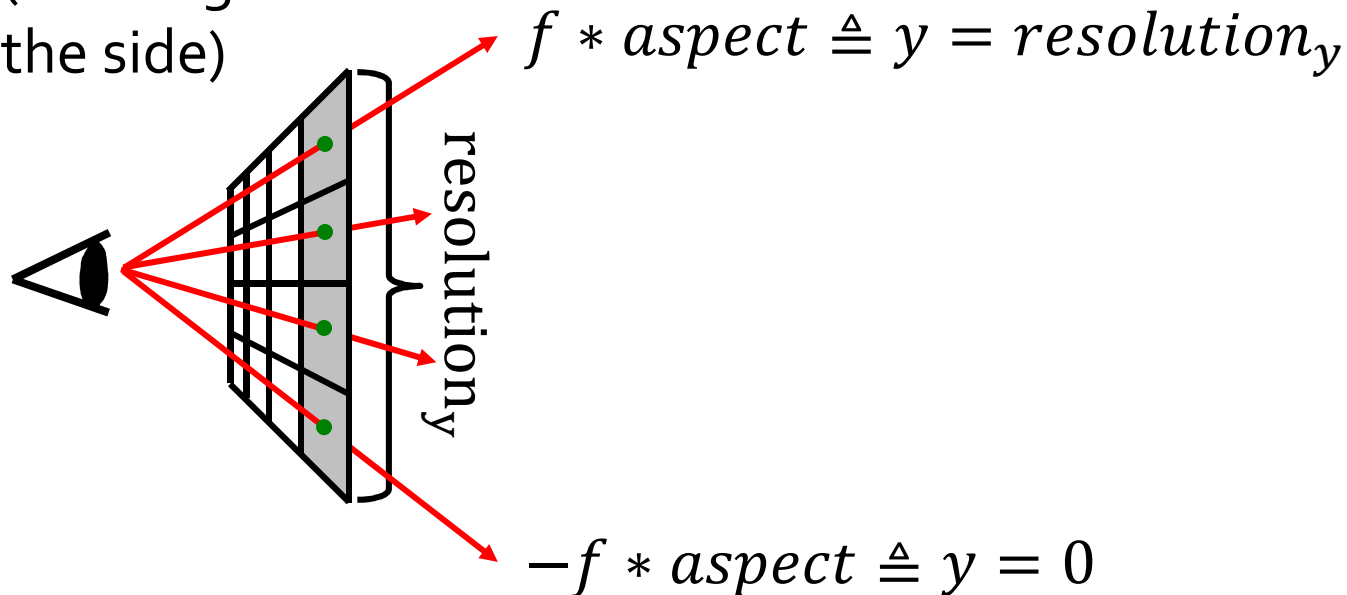$f \triangleq x = resolution_x$ resolution$_x$ $-f \triangleq x = 0$

(Looking down from the top)

Generating Rays – Side View

- Trace a ray for each pixel in the image plane

- $d_y(y) = aspect \left(\frac{2fy}{resolution_y} - f \right) =$

$$\frac{resolution_y}{resolution_x} \left(\frac{2fy}{resolution_y} - f \right) = \frac{f(2y - resolution_y)}{resolution_x}$$
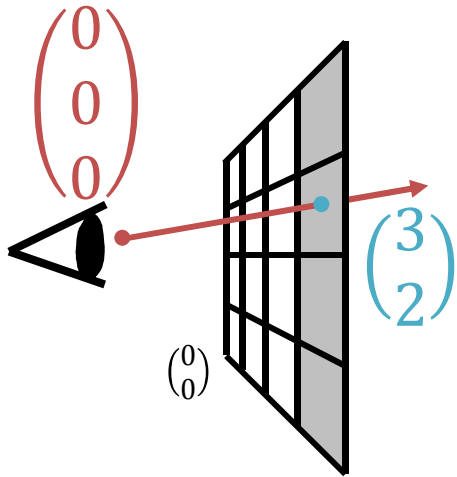
(Looking from the side)

$f * aspect \triangleq y = resolution_y$

resolution$_y$

$-f * aspect \triangleq y = 0$

Generating Rays

- Trace a ray for each pixel in the image plane

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

$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

$\begin{pmatrix} 3 \\ 2 \end{pmatrix}$

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$

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
}
```
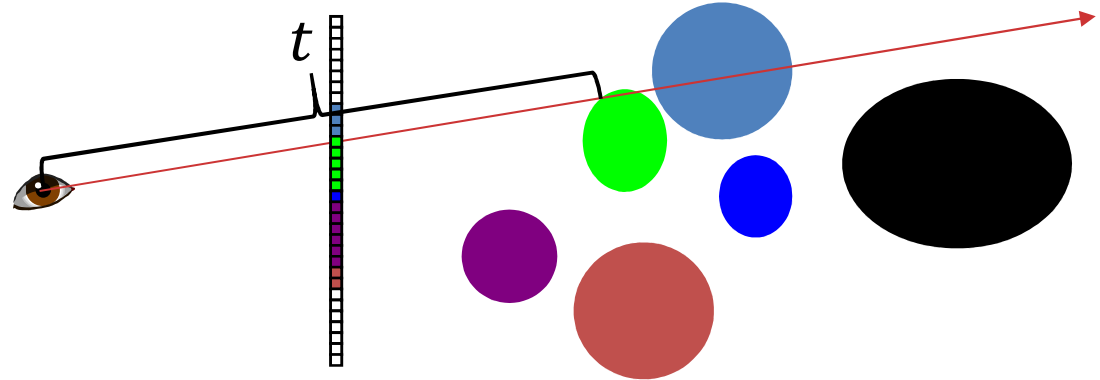
Ray-Object Intersections

- 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 $\mathbf{x} \cdot \mathbf{x} - 1 = 0$
  - Plane with normal $\mathbf{n}$ passing through origin is: $\mathbf{n} \cdot \mathbf{x} = 0$

- 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 $r(t) = \mathbf{S} + \mathbf{c}t$

  - **"S"** is the starting point and **"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:* $F(x, y, z) = x^2 + y^2 - 1 \qquad 0 < z < 1$

- 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 $|\mathbf{p}-\mathbf{p}_c|^2 = r^2 \quad \mathbf{p} = (x,y,z), \mathbf{p}_c = (a,b,c)$
    - 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*"

    $$|\mathbf{S}+\mathbf{c}t-\mathbf{p}_c|^2 - r^2 = t^2|c|^2 + 2t\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c) + \left(|\mathbf{S}-\mathbf{p}_c|^2 - r^2\right)$$

- ## Solving for "t" $t = -\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c) \pm \sqrt{(\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c))^2 - |c|^2\left(|\mathbf{S}-\mathbf{p}_c|^2 - r^2\right)}$
    - 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
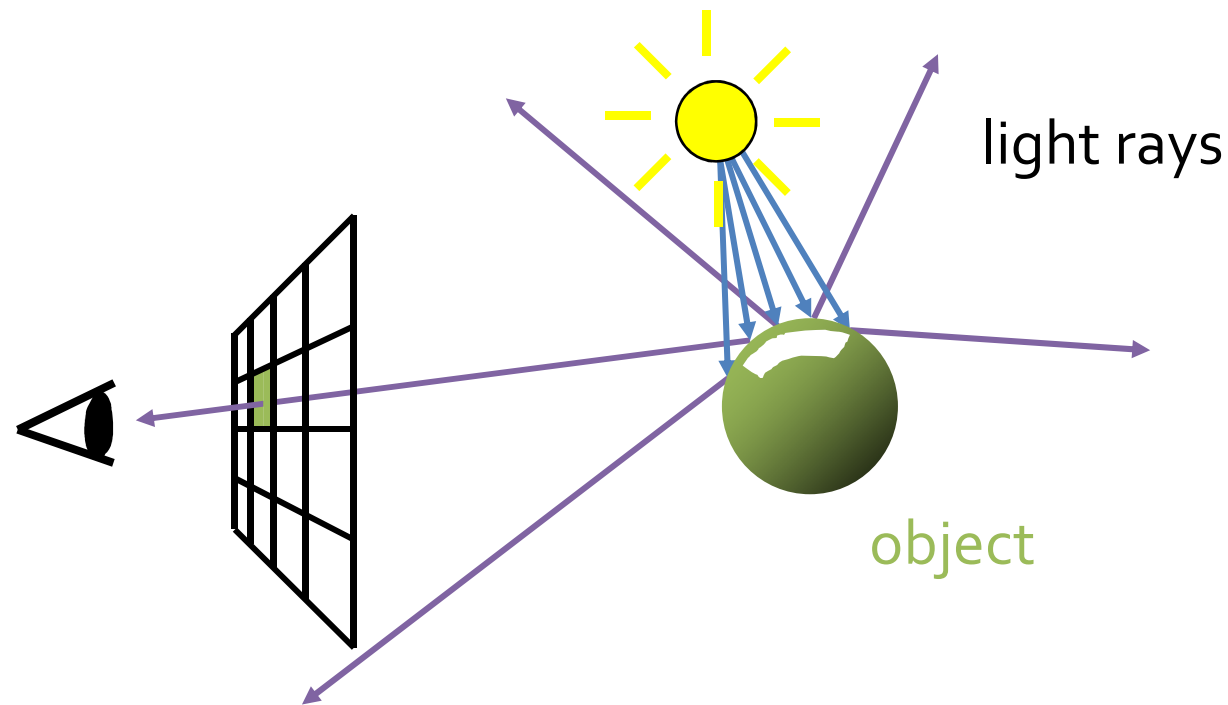 - intersection point
  - normal vector

Ray-Tracing

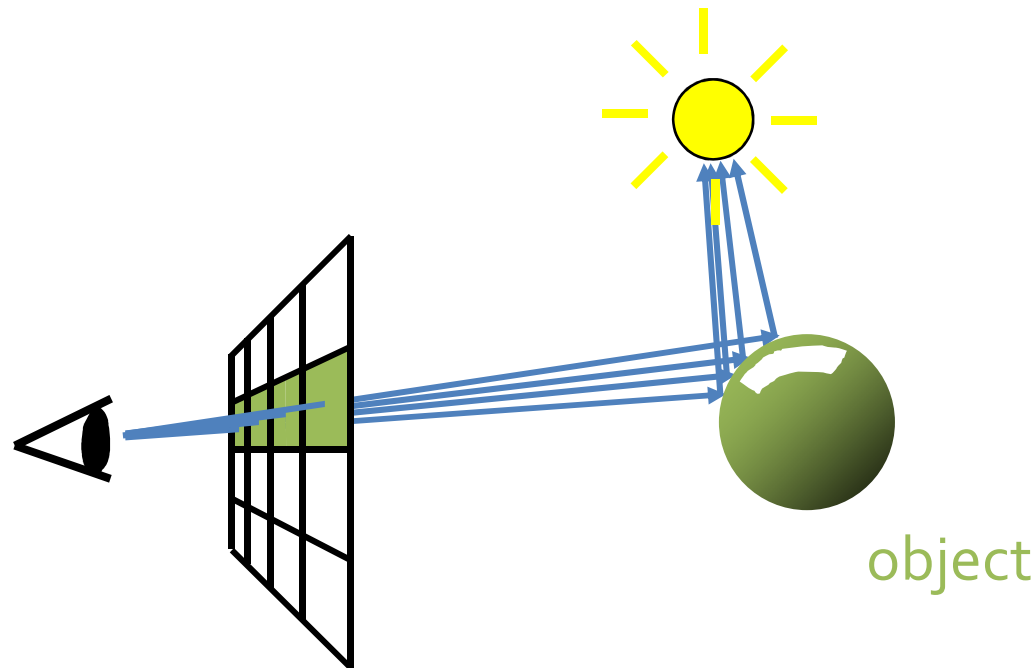# The Basic Idea

- Simulate light rays from light source to eye
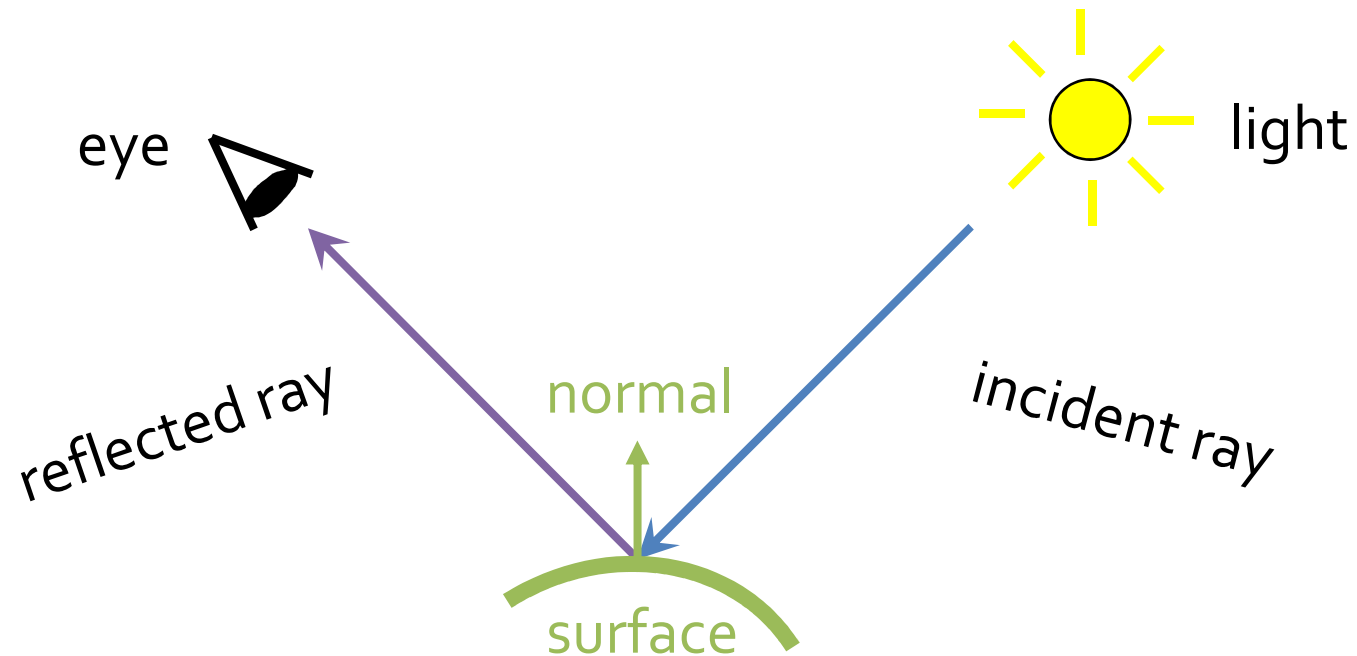
"Forward" Ray-Tracing
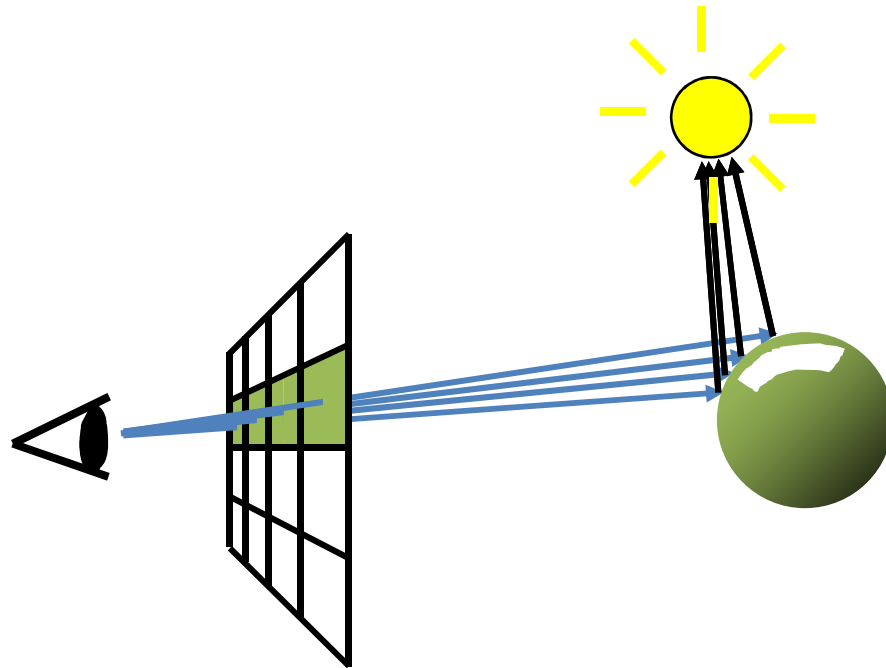
- Trace rays from light
- Lots of work for little return

light rays

object

"Backward" Ray-Tracing

- Trace rays from eye instead

- Do work where it matters

- *This is what most people mean by "ray tracing".*

object

Types of Rays

- Primary rays
- Shadow rays

Shadow Rays

- Primary rays
- Shadow rays
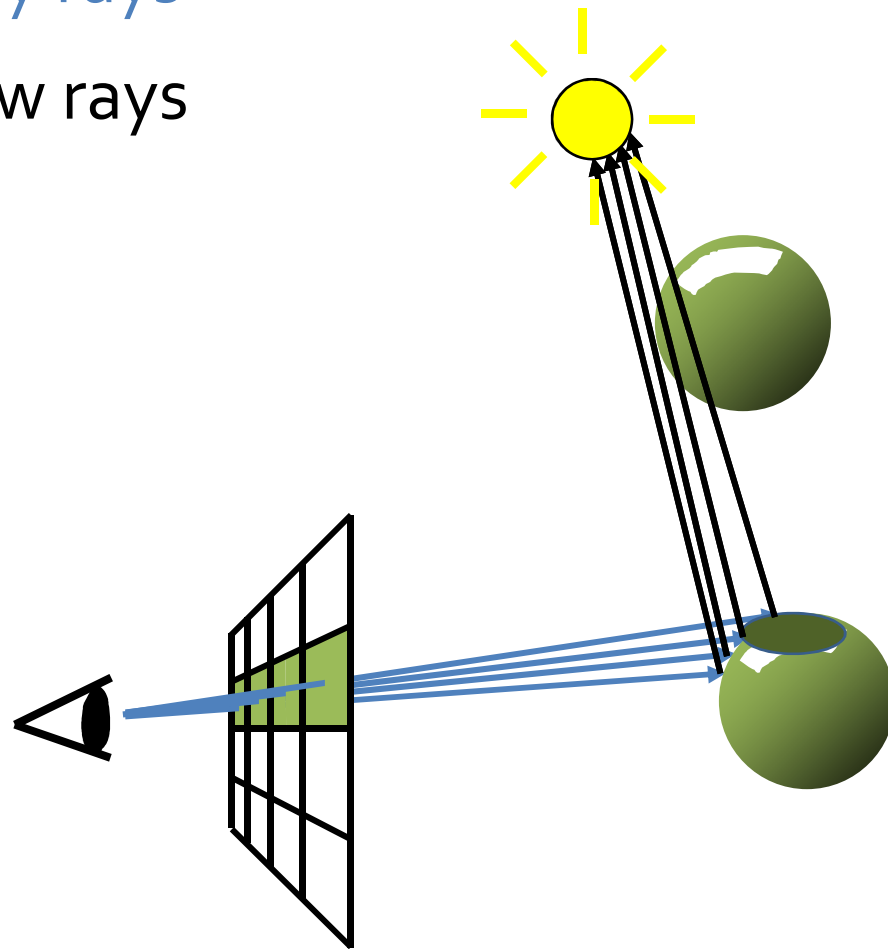
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays
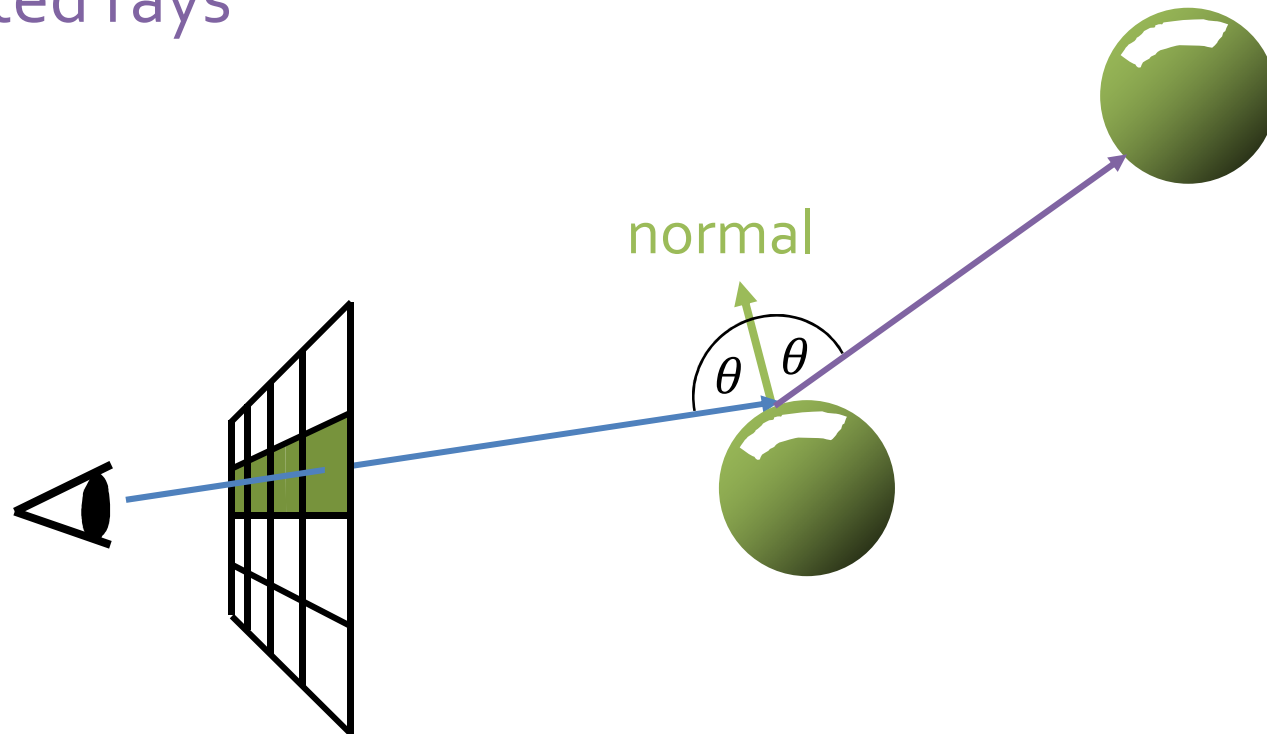
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays
- Refracted rays
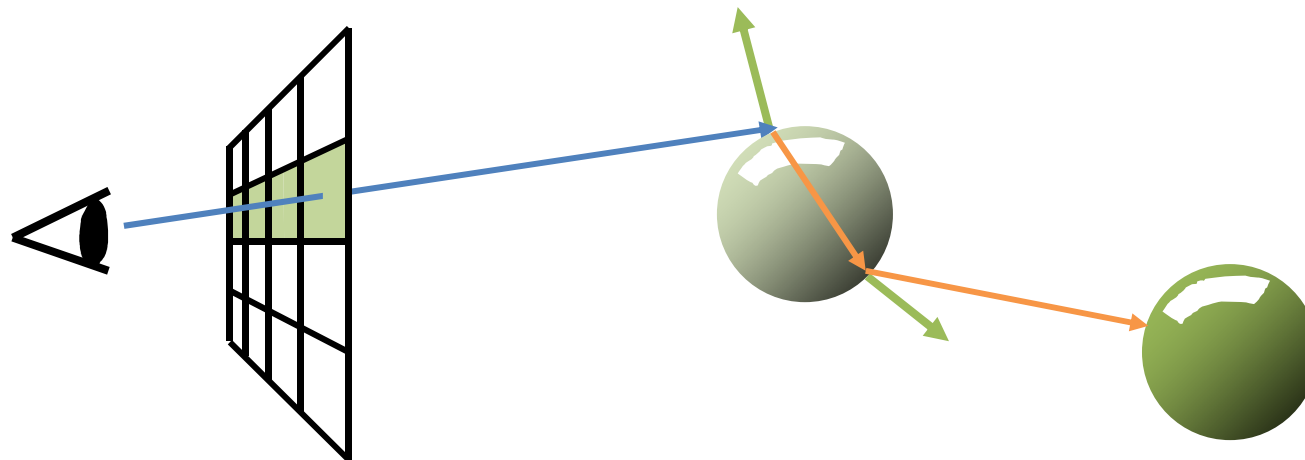
Types of Rays

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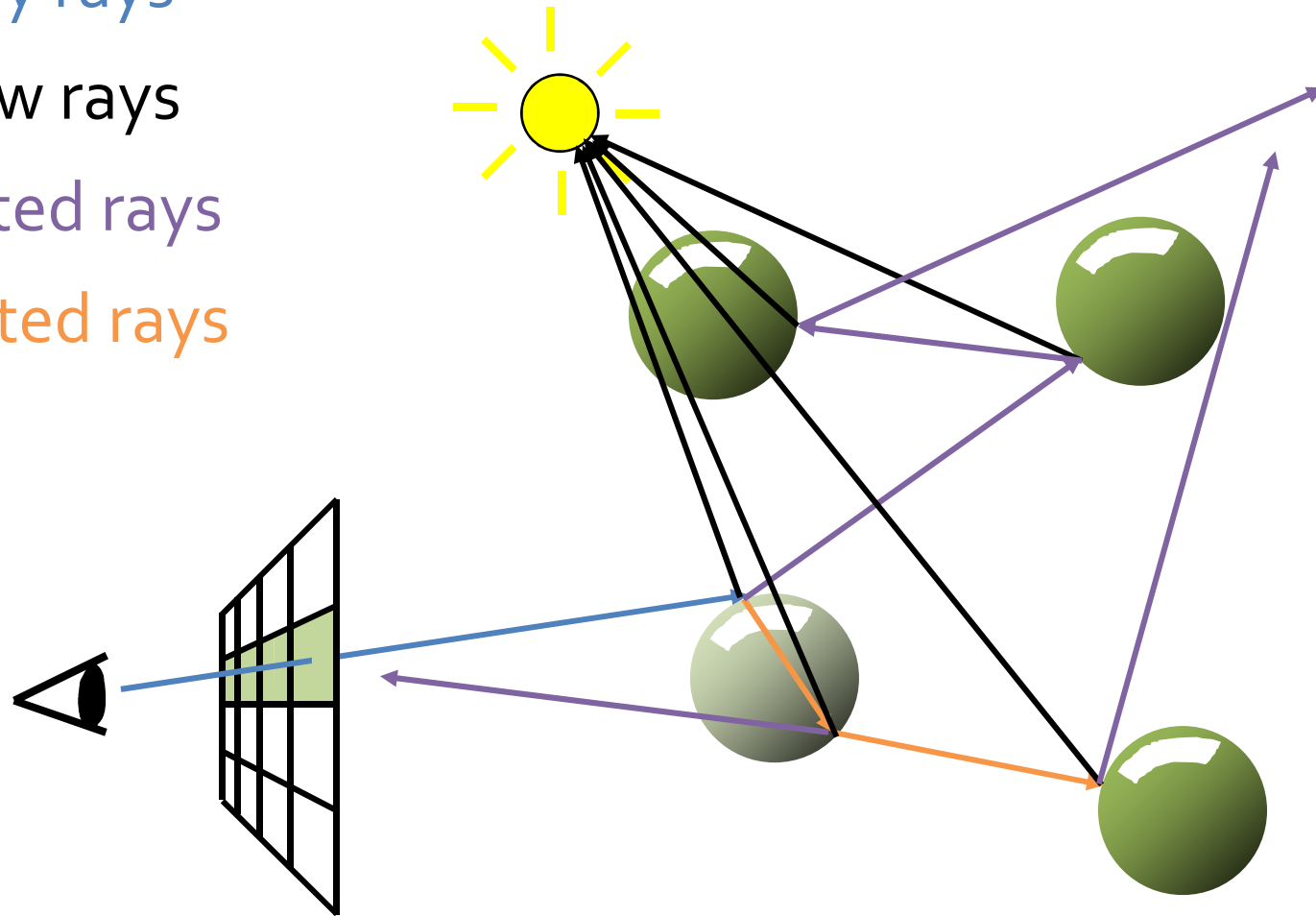
Lighting

Lighting

- $C = C_{local} + C_{reflected} + C_{transmitted}$
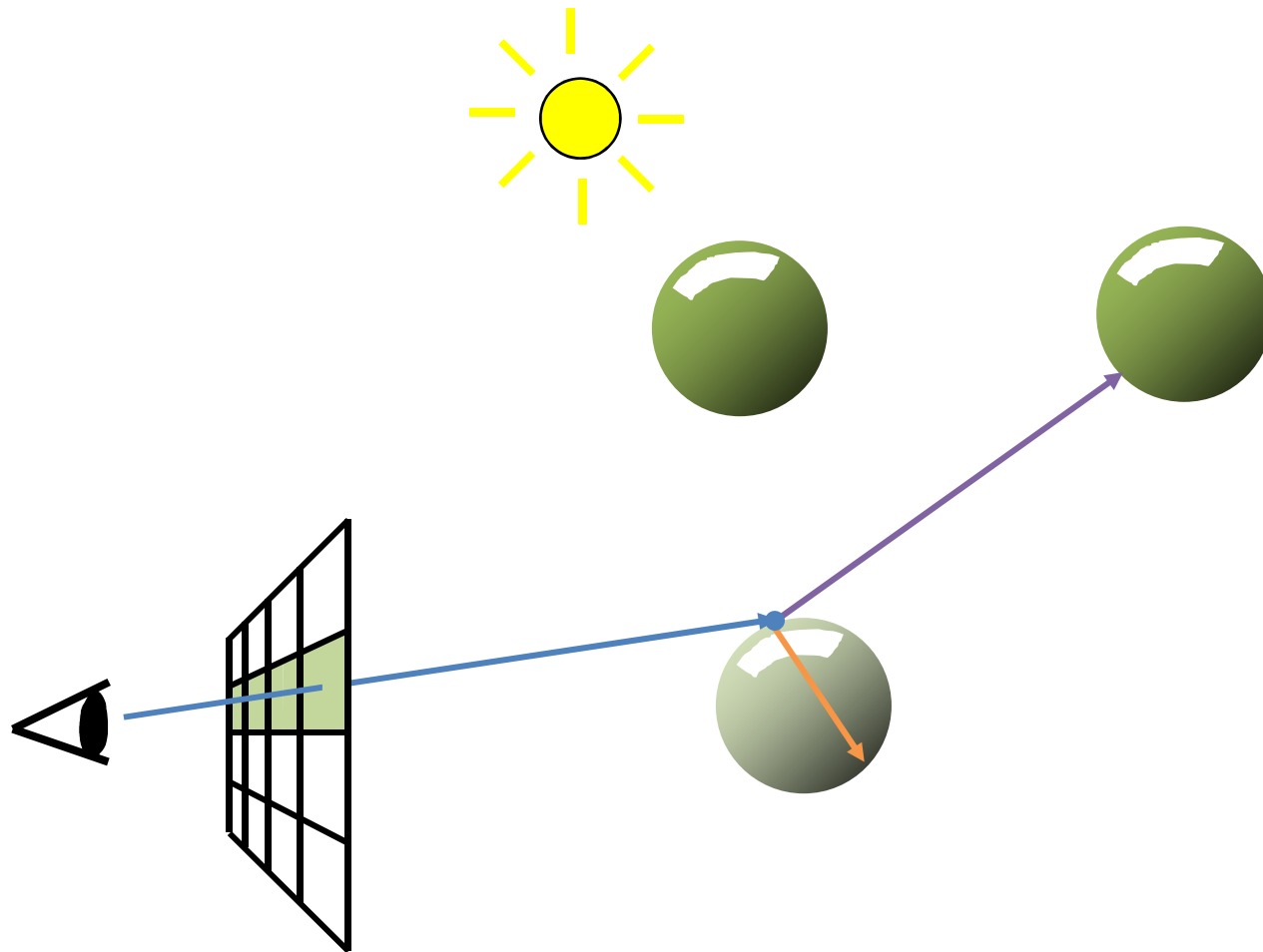
Local – Phong Illumination
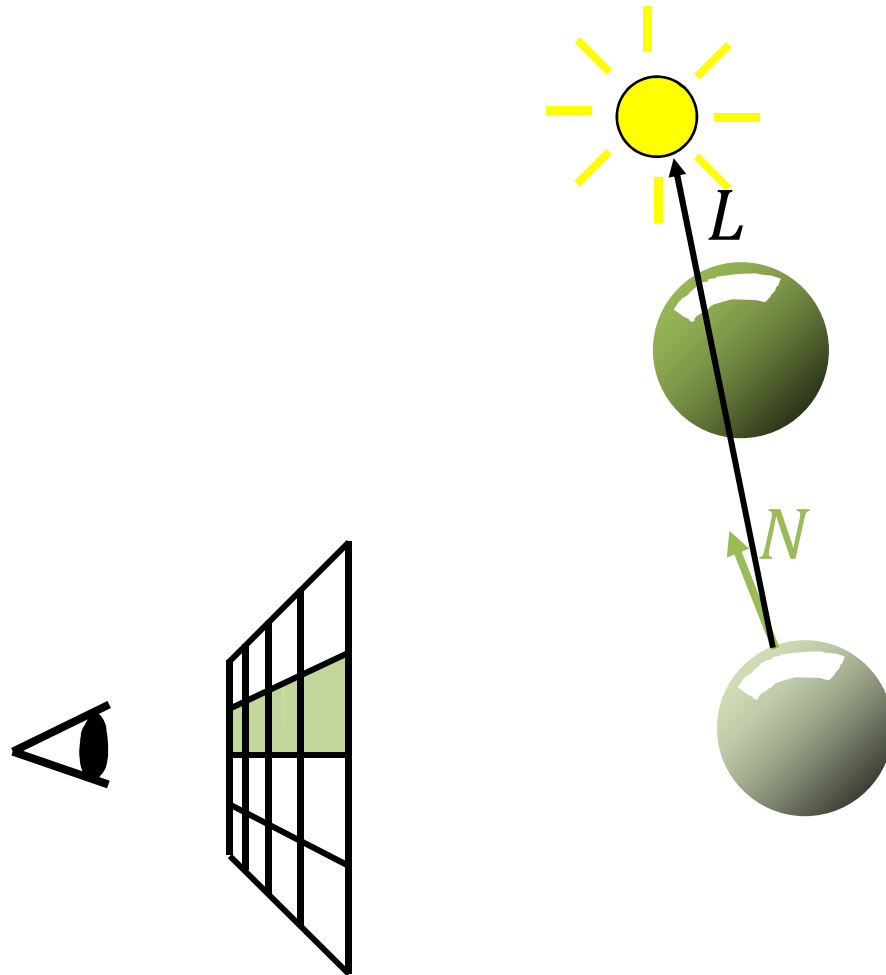
- $C_{local} = C_{ambient} + C_{diffuse} + C_{specular}$
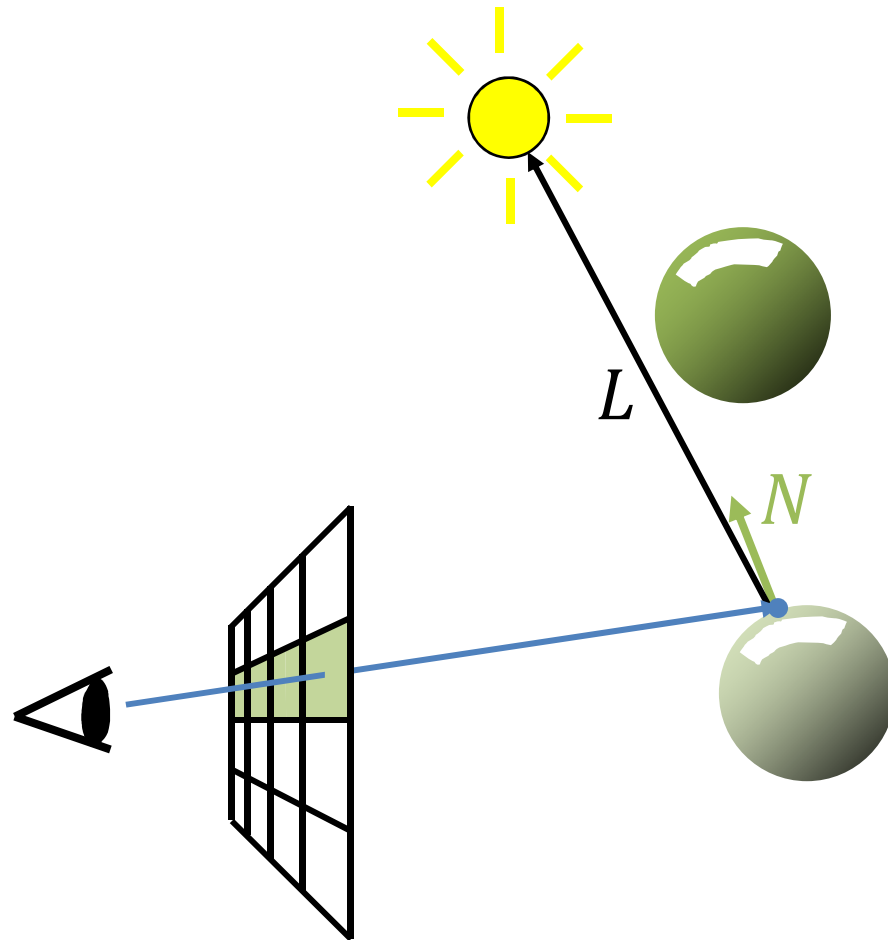
Diffuse (Lambert)

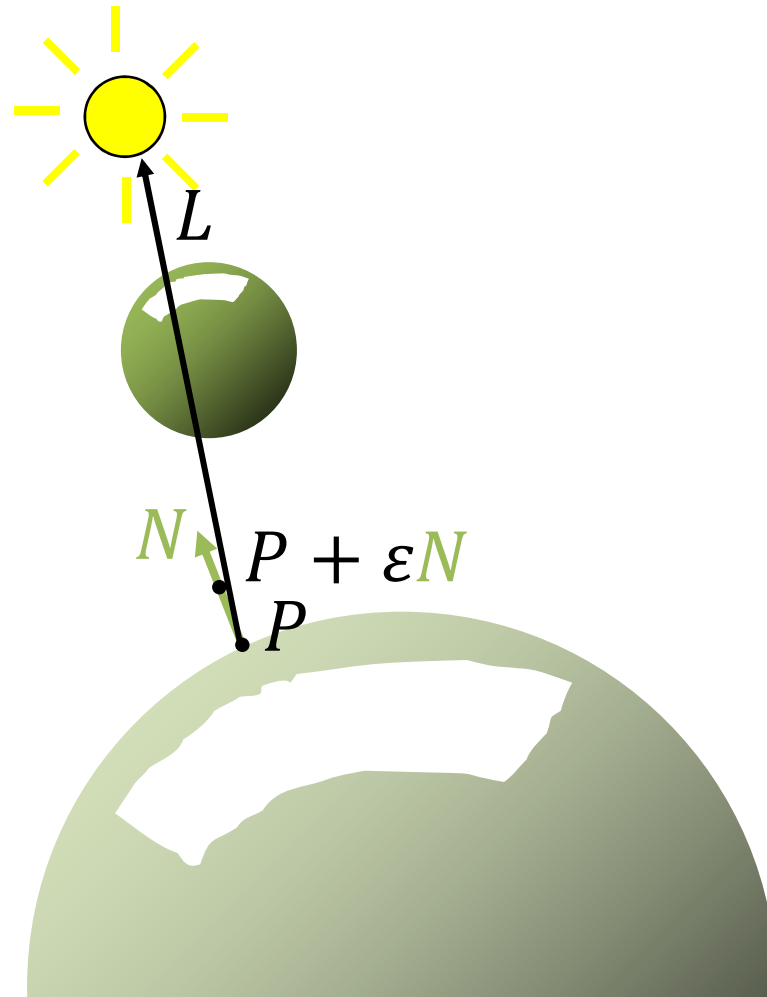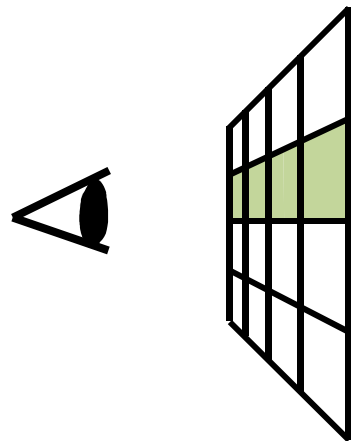- $C_{local} = \max(0, N \cdot L) * Color_{object} * Color_{light}$
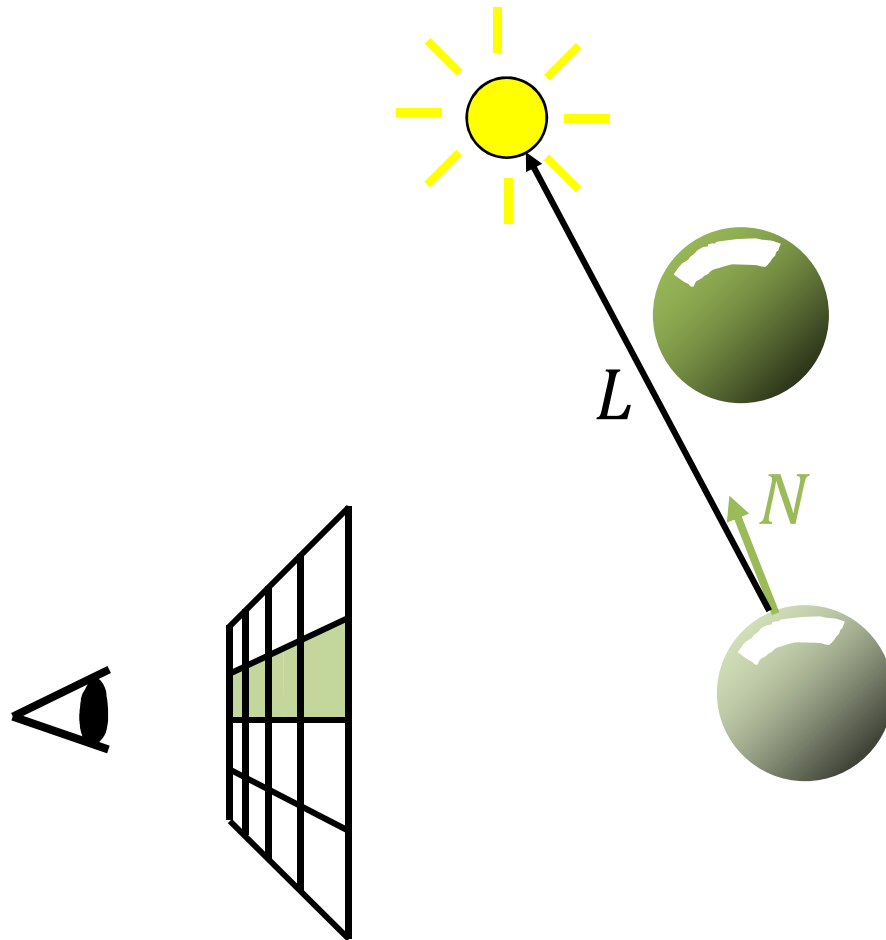
Adding Shadows

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

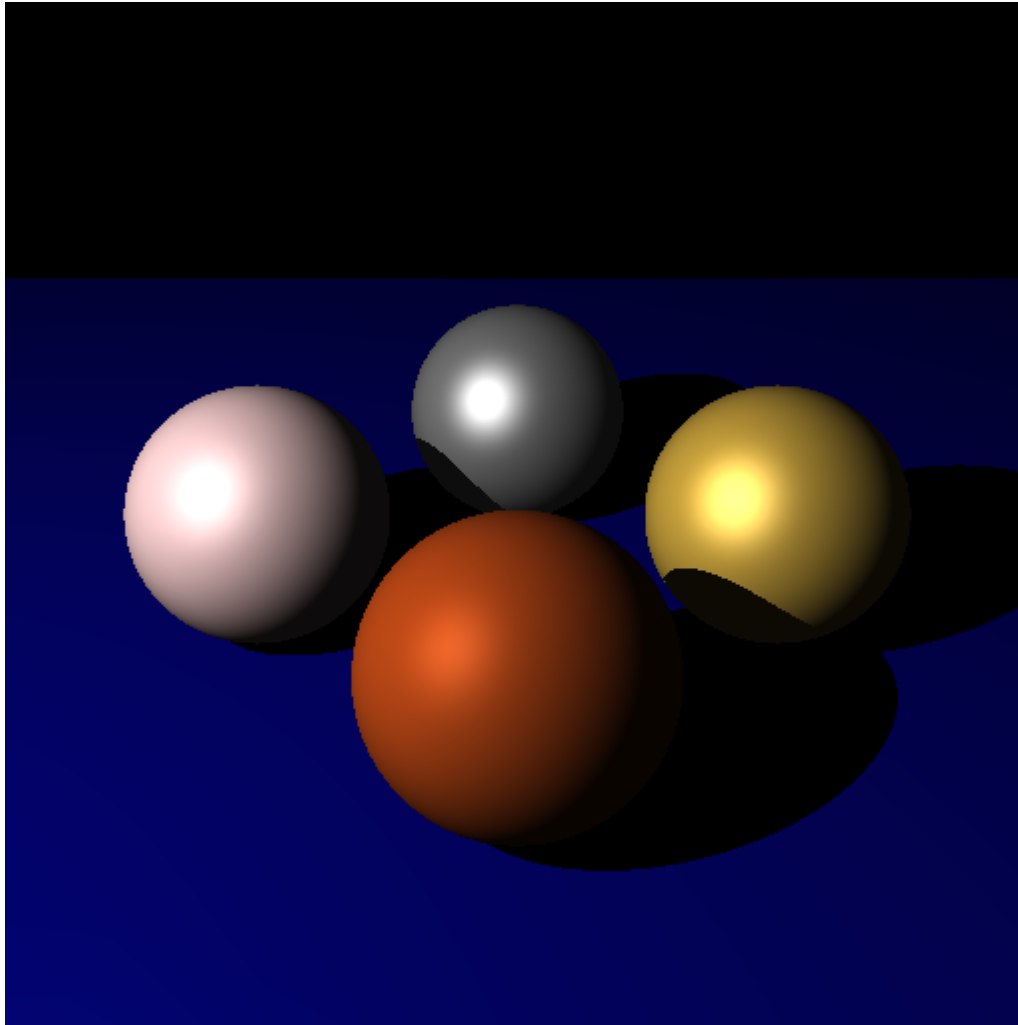# Adding Shadows

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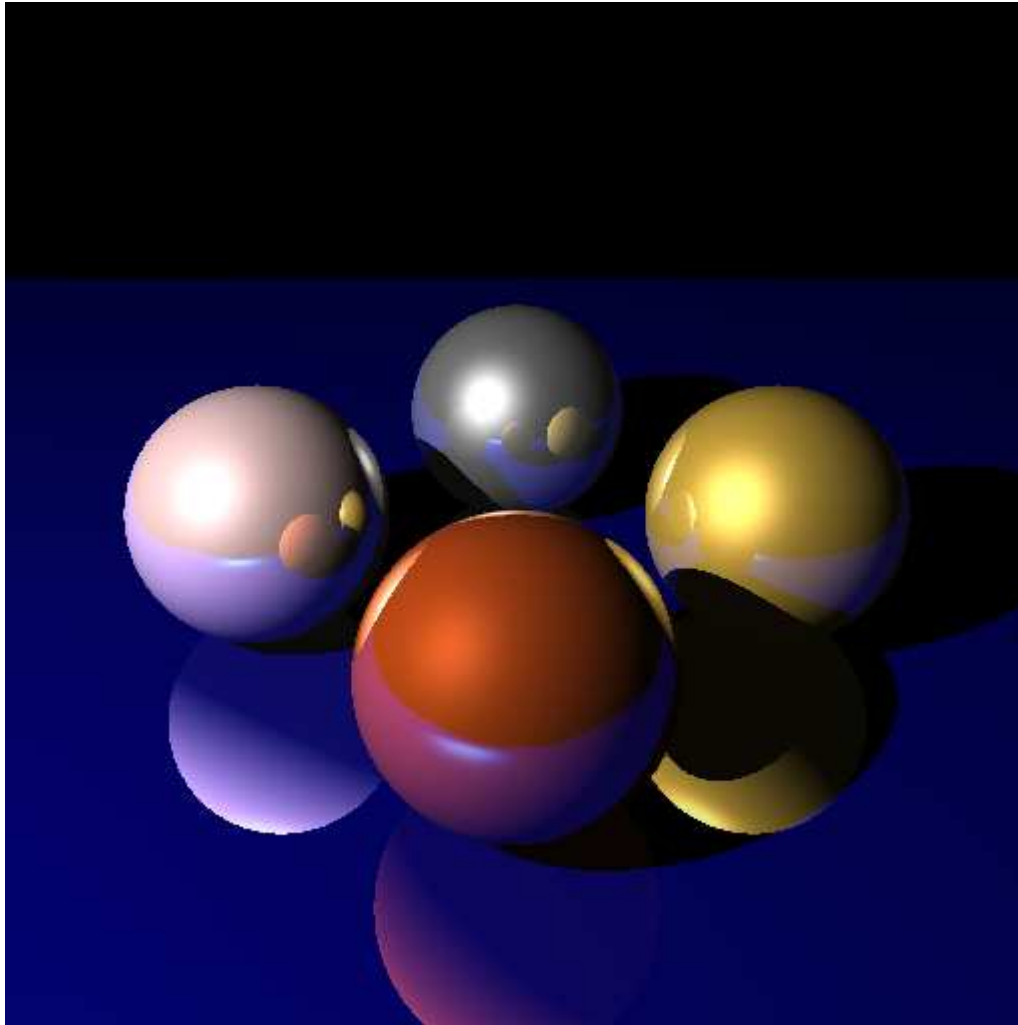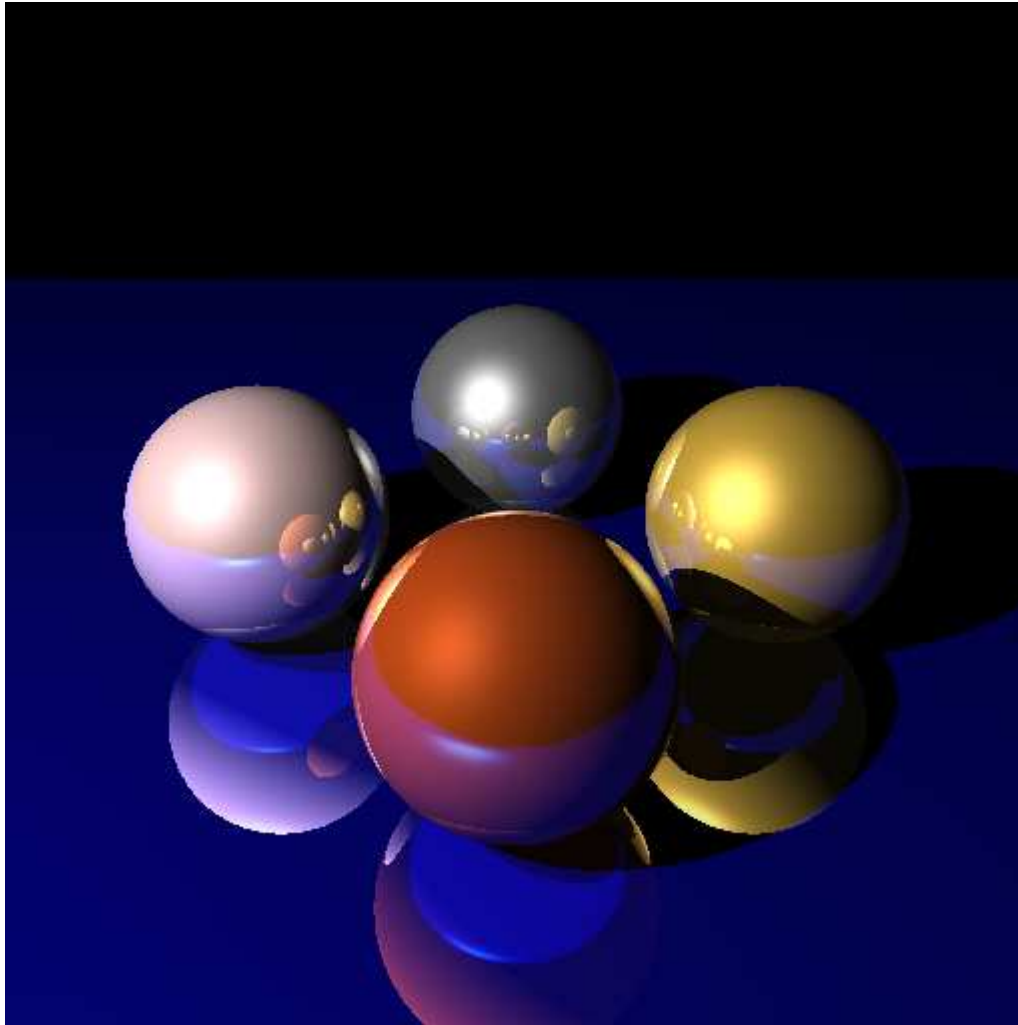 - Move start point of the shadow ray away by a small amount

L

N $P + \varepsilon N$

P

Reflection

Reflection (1)

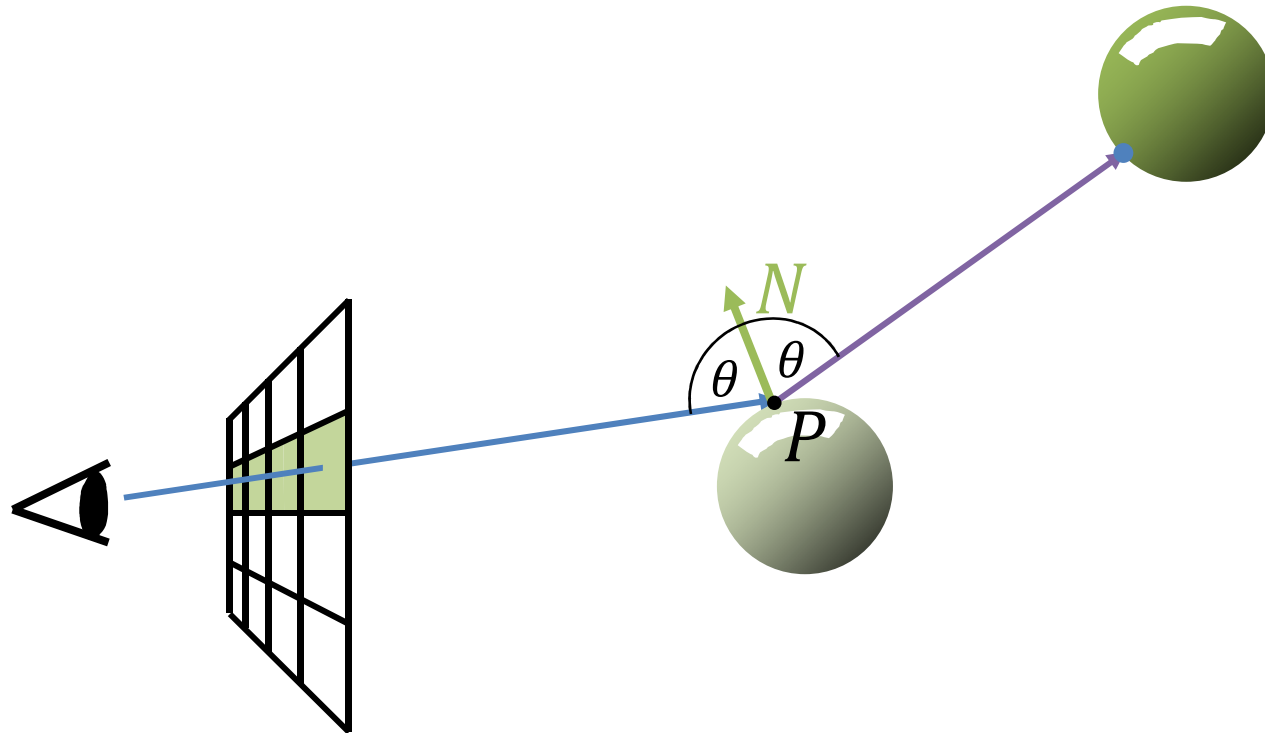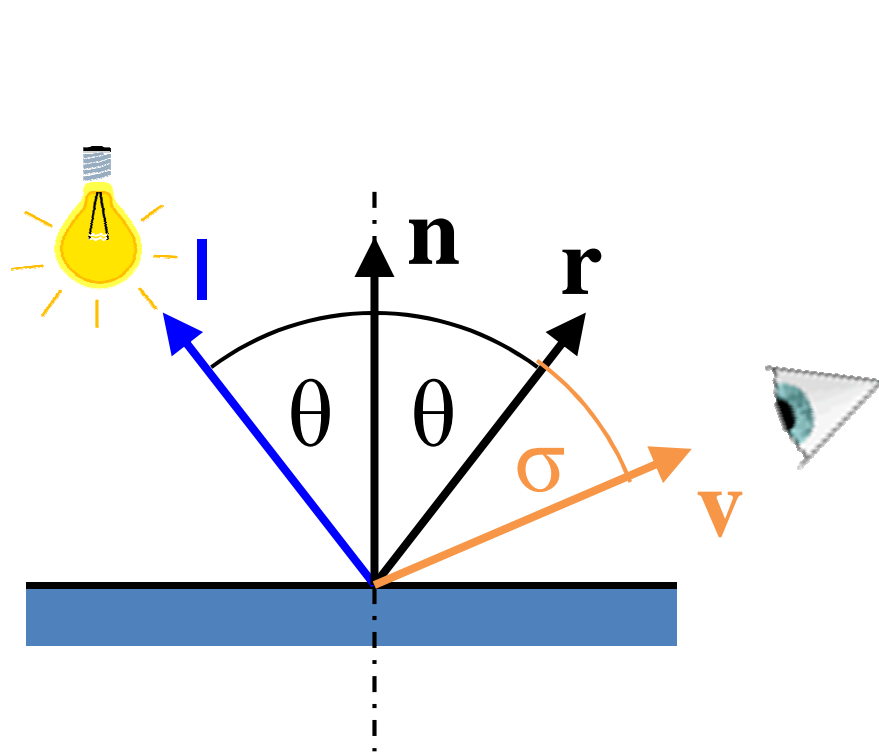*Created by David Derman – CISC 440*

Reflection (2)
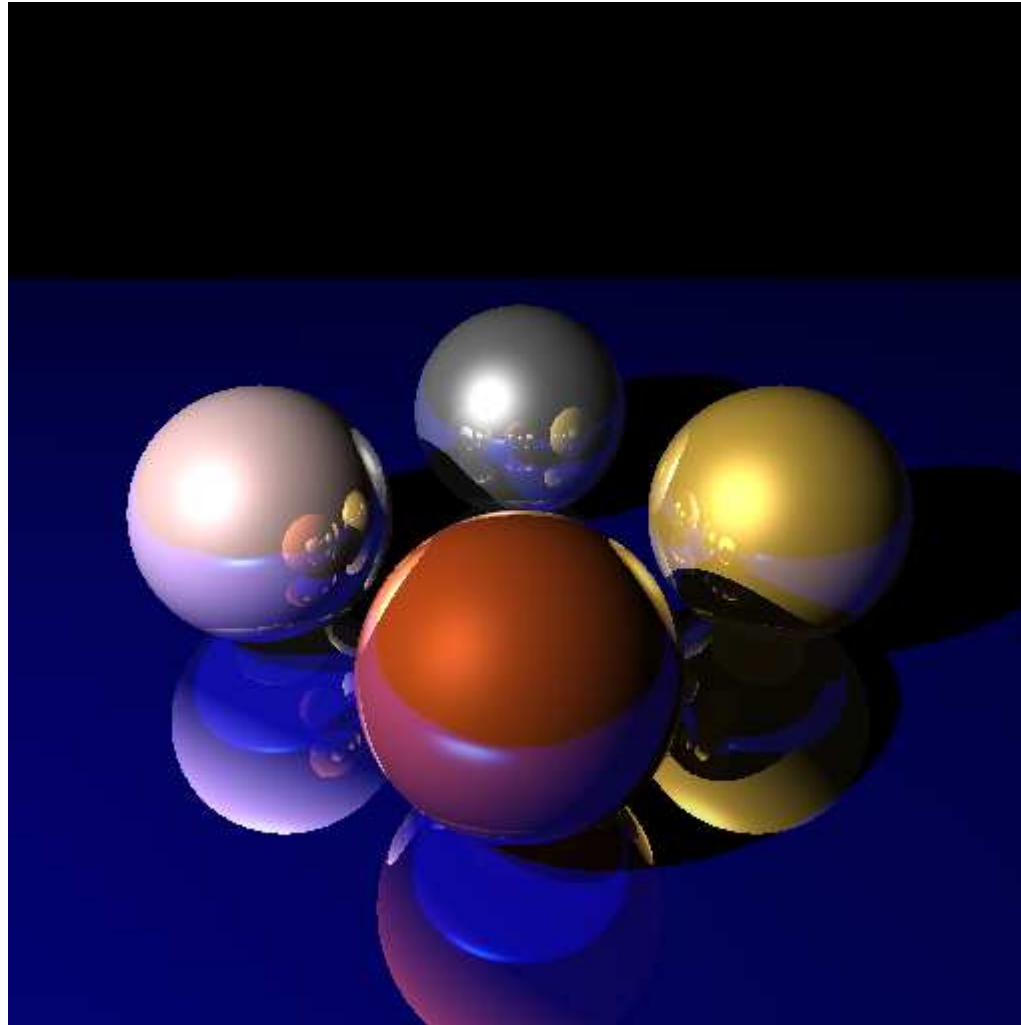
Reflection (3)

Created by David Derman – CISC 440

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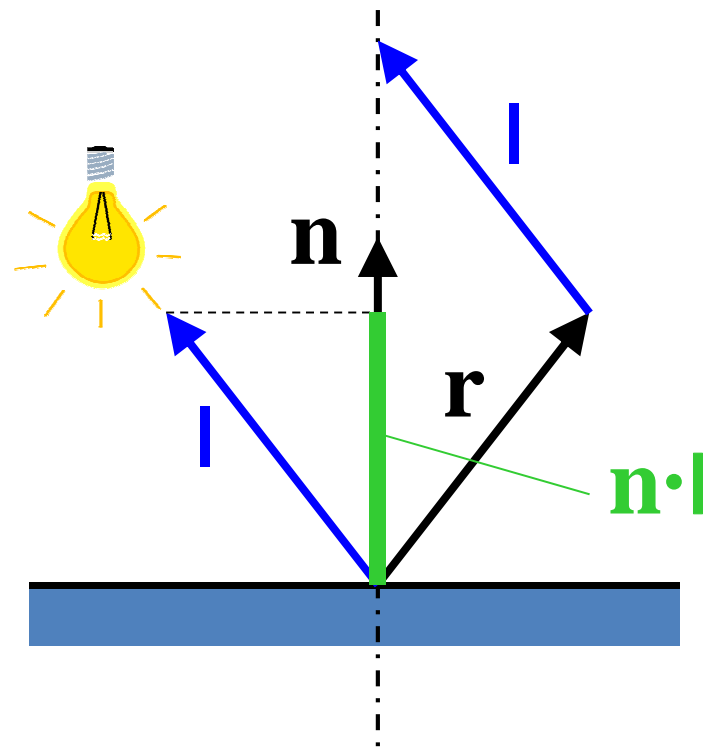
- $C_{reflected} = C_{\text{intersect}(P,\text{reflect}(dir,N))}$

Reflection direction

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

$$\mathbf{r} + \mathbf{l} = (2\mathbf{n} \cdot \mathbf{l})\mathbf{n}$$

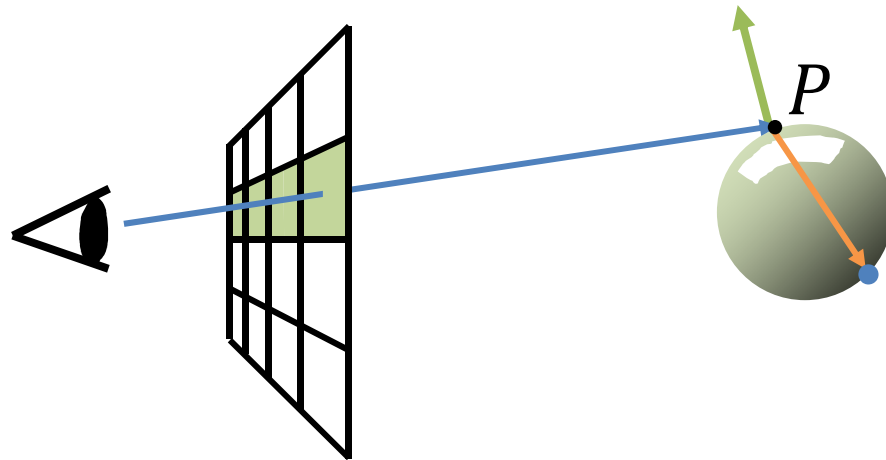$$\mathbf{r} = (2\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$$

Refraction

Refraction

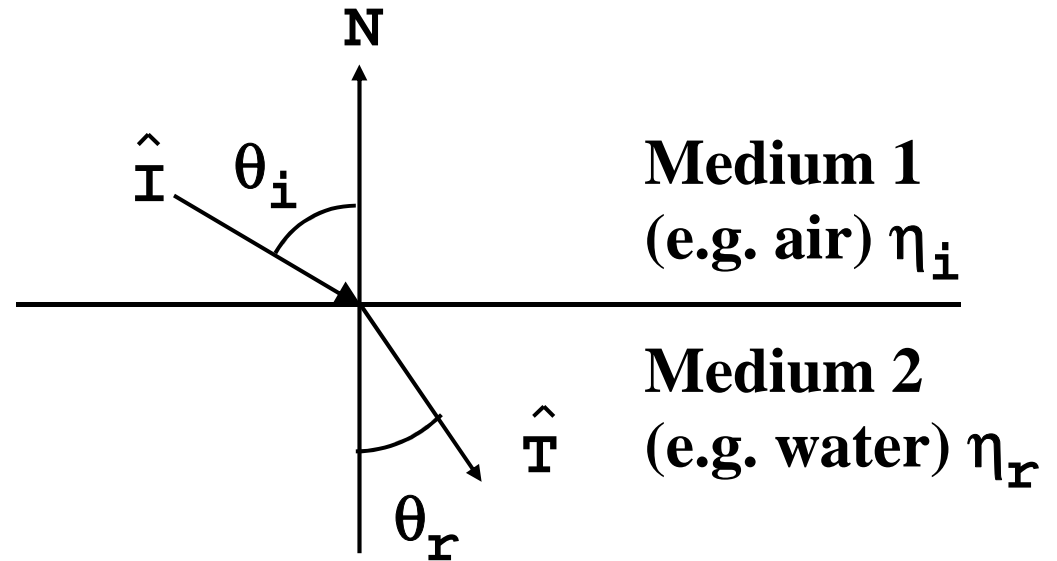- $C_{refracted} = C_{\text{intersect}(P,\text{refract}(dir,N))}$

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

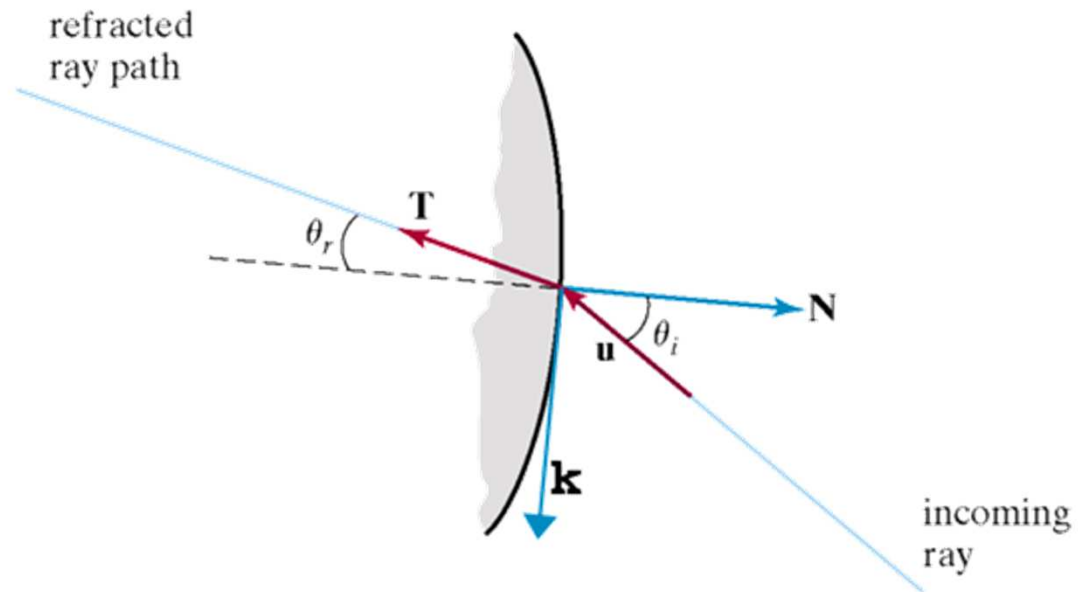- Decomposing the incident ray (**u**)

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

- Decomposing the refracted ray (**T**)

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

- Solving for **k** from **u**

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

refracted ray path

incoming ray

Refraction

- Substituting in **T**

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

- From Snell's Law

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

- Solving for **T**

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

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

$$= \frac{n_i}{n_r}\mathbf{u} - \left(\cos\theta_r - \frac{n_i}{n_r}\cos\theta_i\right)\mathbf{n}$$
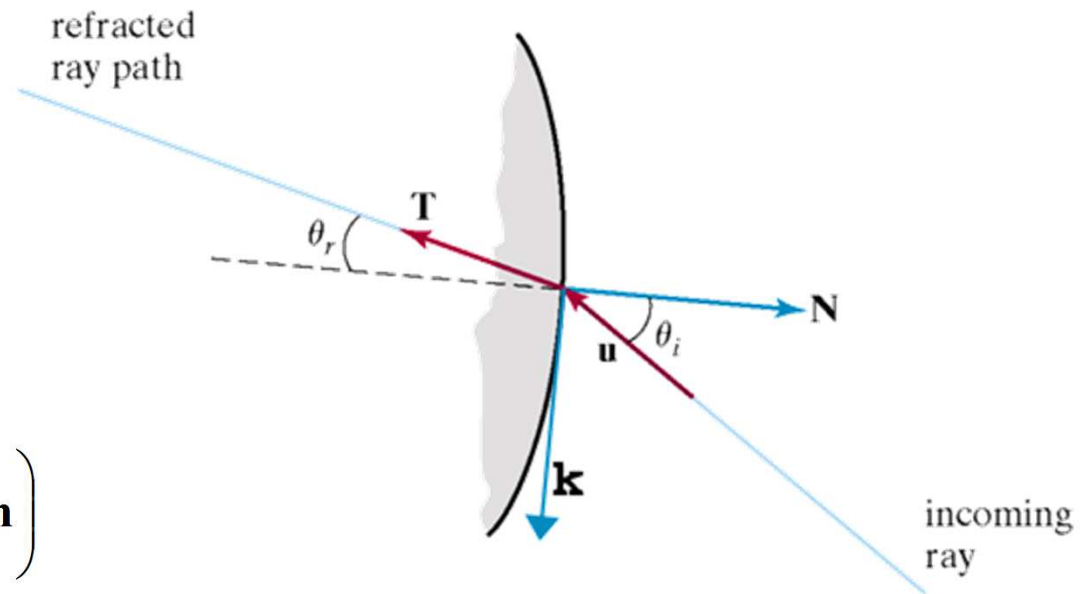
GPU acceleration structures for real-time ray-tracing