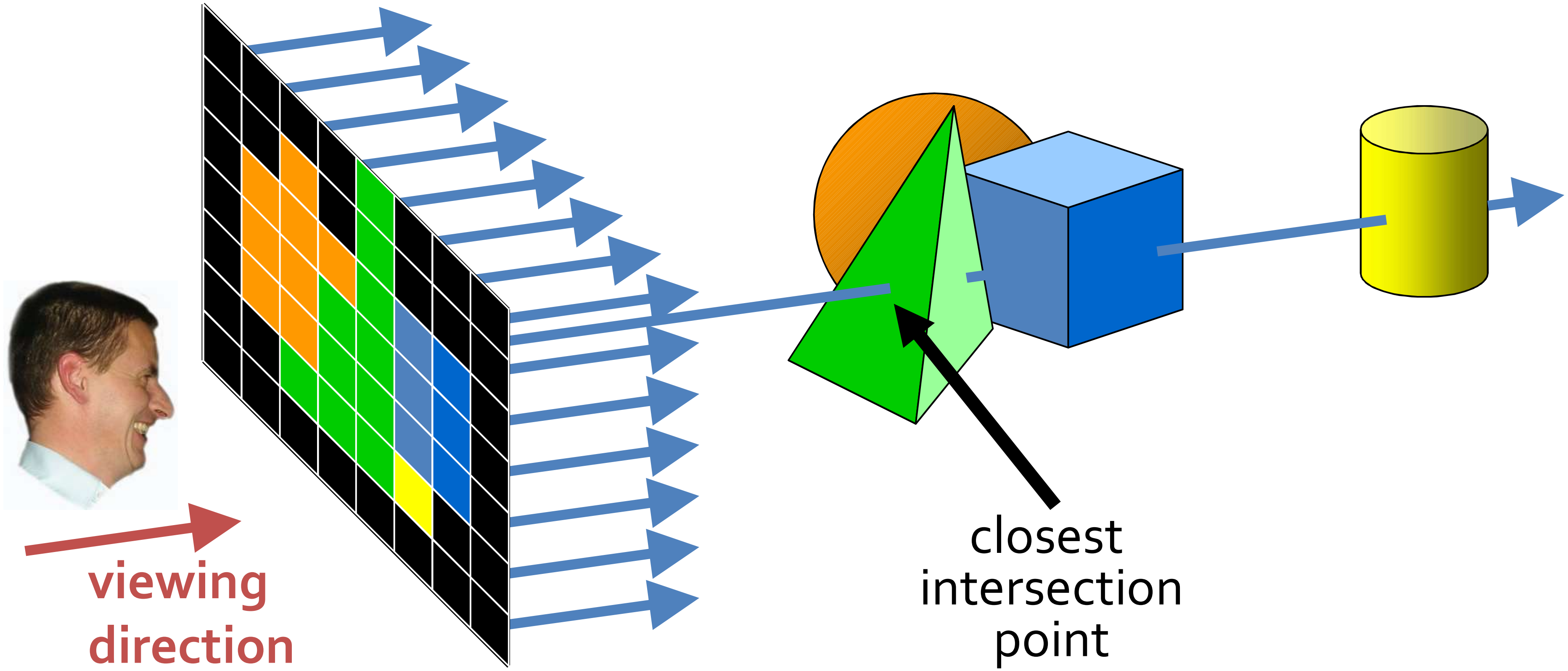
Ray-Casting

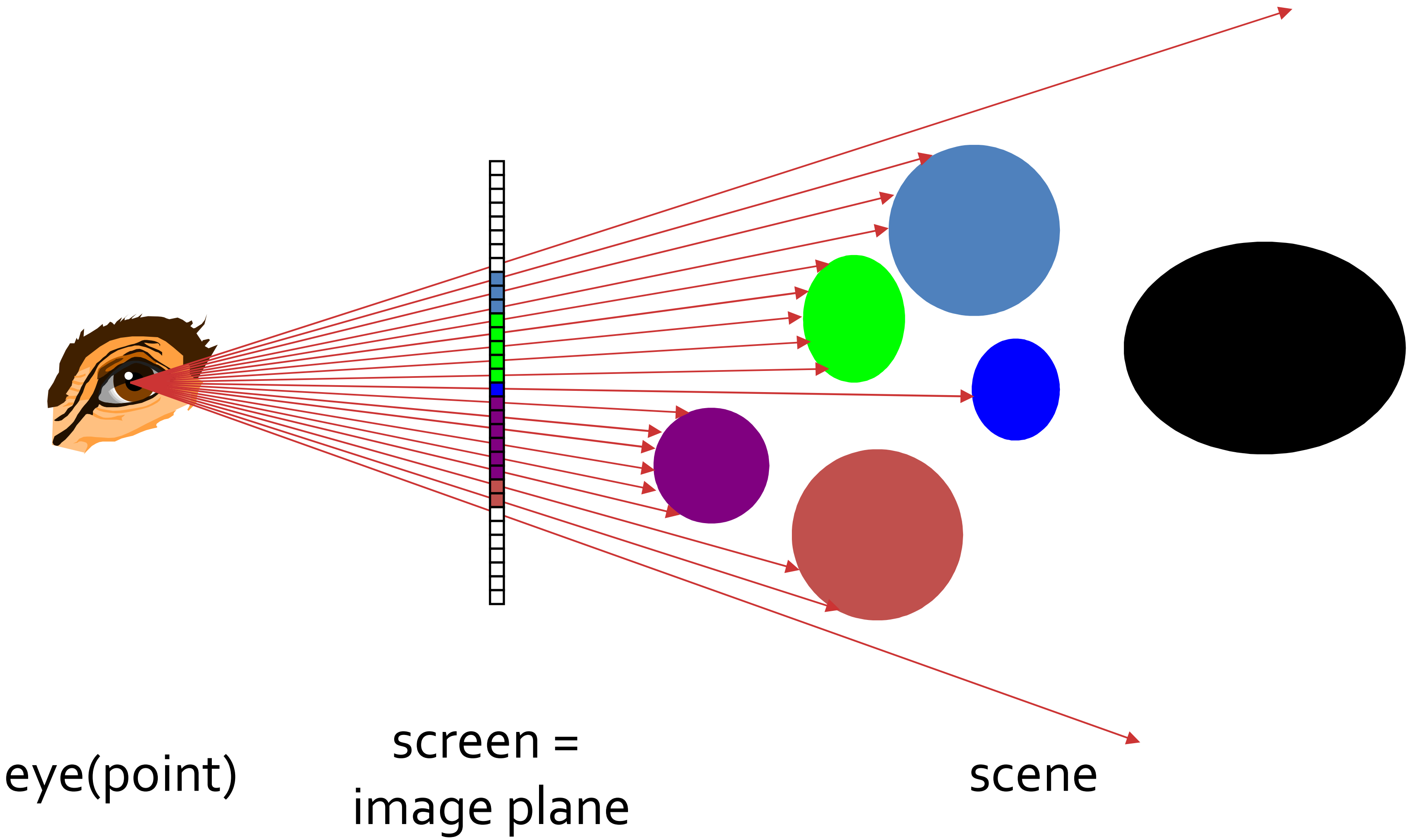
Ray-Casting Method

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



Generating Rays

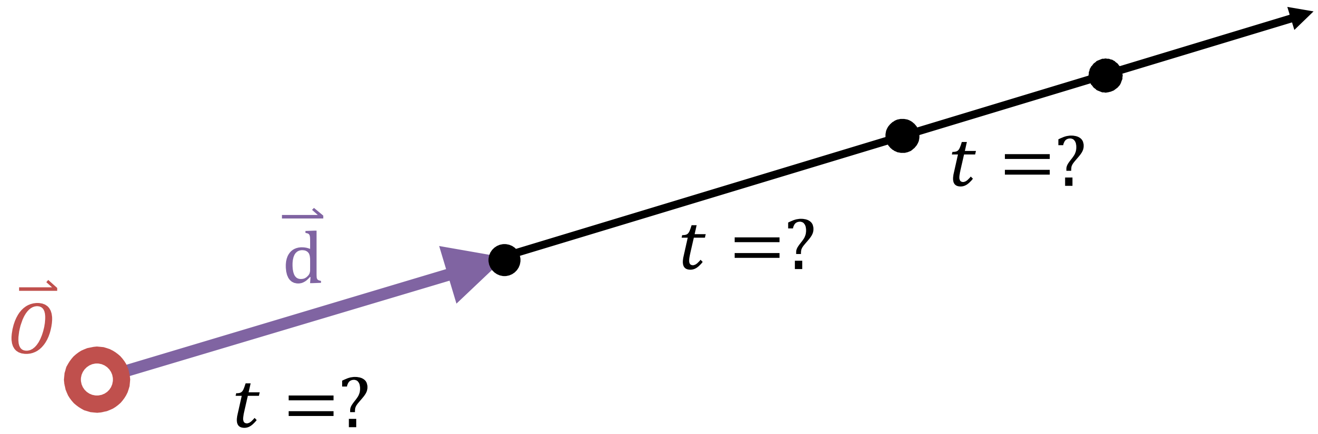
- Trace a ray for each pixel in the image plane



Ray Parametric Form

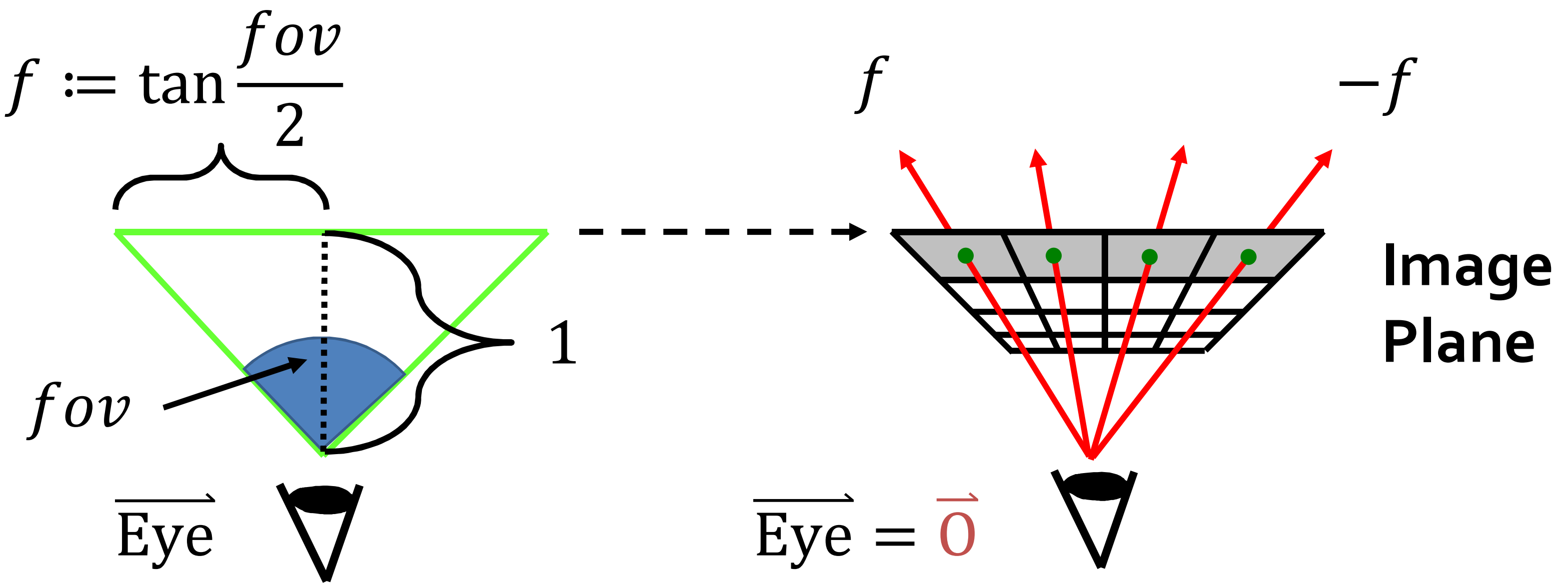
- Ray expressed as function of a single parameter t

$$\vec{\mathrm{P}} = \vec{\mathrm{O}} + t\vec{\mathrm{d}}$$
$$= \begin{pmatrix} \mathrm{O}_x \\ \mathrm{O}_y \\ \mathrm{O}_z \end{pmatrix} + t\begin{pmatrix} \mathrm{d}_x \\ \mathrm{d}_y \\ \mathrm{d}_z \end{pmatrix}$$



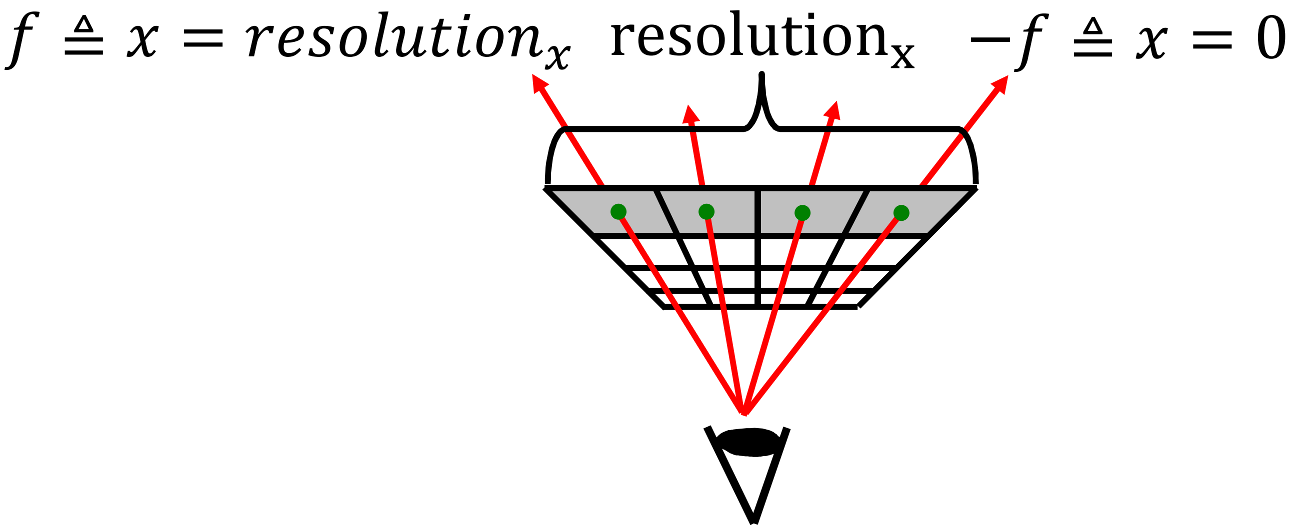
Generating Rays – Top View

- Trace a ray for each pixel in the image plane



Generating Rays – Top View

- Trace a ray for each pixel in the image plane
- $d_x(x) = \frac{2fx}{resolution_x} - f = \frac{f(2x - resolution_x)}{resolution_x}$



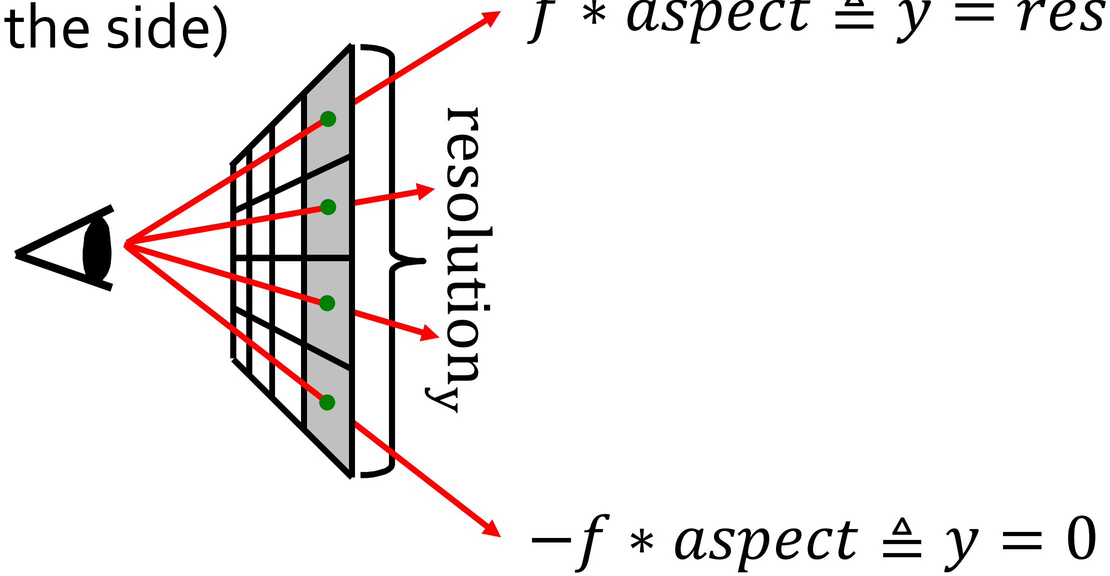
(Looking down from the top)

Generating Rays – Side View

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

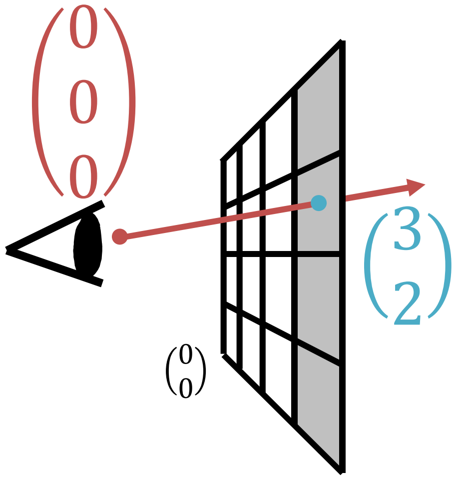
$$\frac{\mathrm{resolution_y}}{\mathrm{resolution_x}}\left(\frac{2fy}{resolution_y} - f\right) = \frac{f\left(2y - \mathrm{resolution_y}\right)}{\mathrm{resolution_x}}$$

(Looking from the side)



Generating Rays

- Trace a ray for each pixel in the image plane
- For a pixel $\begin{pmatrix} x \\ y \end{pmatrix}: \vec{\mathrm{P}} = \vec{\mathrm{O}} + t\vec{\mathrm{d}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + t \begin{vmatrix} d_x(x) \\ d_y(y) \\ 1 \end{vmatrix}$



Generating Rays

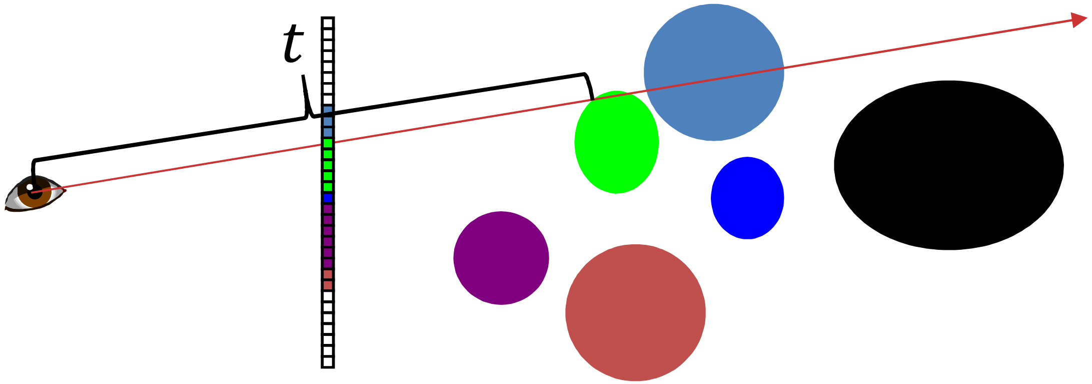
- Trace a ray for each pixel in the image plane

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

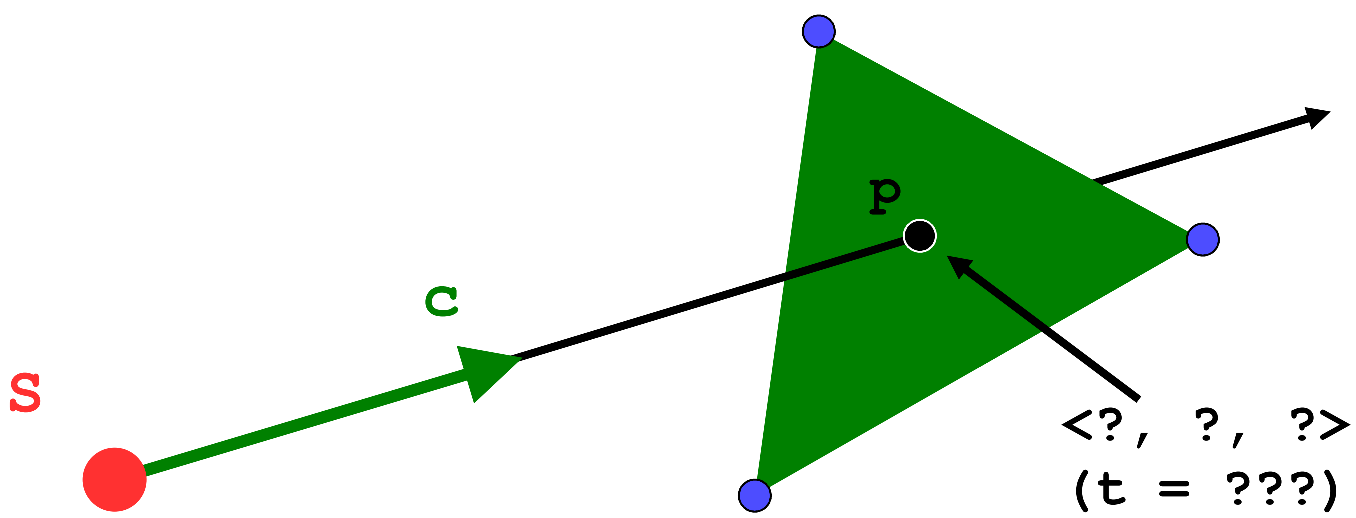
Ray Object Intersection

- Ray sphere intersection $\left|\mathbf{p}-\mathbf{p}_c\right|^2 = r^2 \quad \mathbf{p}=(x,y,z), \mathbf{p}_c=(a,b,c)$
  - Implicit form of sphere given center *(a,b,c)* and radius *r*
- Intersection with *r(t)* gives $\left|\mathbf{S}+\mathbf{c}t-\mathbf{p}_c\right|^2 = r^2$
- By the identity $\left|\mathbf{a}+\mathbf{b}\right|^2 = \left|\mathbf{a}\right|^2 + \left|\mathbf{b}\right|^2 + 2(\mathbf{a}\cdot\mathbf{b})$
  - Intersection equation is quadratic in "*t*"

    $$\left|\mathbf{S}+\mathbf{c}t-\mathbf{p}_c\right|^2 - r^2 = t^2\left|c\right|^2 + 2t\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c) + \left(\left|\mathbf{S}-\mathbf{p}_c\right|^2 - r^2\right)$$
- Solving for "t" $t = -\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c) \pm \sqrt{(\mathbf{c}\cdot(\mathbf{S}-\mathbf{p}_c))^2 - \left|c\right|^2\left(\left|\mathbf{S}-\mathbf{p}_c\right|^2 - r^2\right)}$
  - Real solutions, indicate one or two intersections
  - Negative solutions are behind the eye
  - If discriminant is negative, the ray missed the sphere

Triangle Intersection

- Want to know: at what *point* (`p`) does ray intersect triangle?
- Compute lighting, reflected rays, shadowing *from that point*



Ray Triangle Intersection

- Point on triangle (Barycentric coordinates)
  $t(u,v) = (1 - u - v)A + uB + vC$

- Ray
  $r(t) = O + tD$

- Intersection
  $O + tD = (1 - u - v)A + uB + vC$

Ray Triangle Intersection

- Intersection O + tD = (1 – u – v)A + uB + vC
- Rearranged

$$O - A = \begin{pmatrix} -D & B - A & C - A \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix}$$

- Linear system!
- Solve with Cramer’s rule

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{det(-D, B-A, C-A)} \begin{pmatrix} det(O - A, B - A, C - A) \\ det(-D, O - A, C - A) \\ det(-D, B - A, O - A) \end{pmatrix}$$

Ray Triangle Intersection: Implementation

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{det(-D, B-A, C-A)} \begin{pmatrix} det(O-A, B-A, C-A) \\ det(-D, O-A, C-A) \\ det(-D, B-A, O-A) \end{pmatrix}$$

- Rewrite using:

$$det(A, B, C) = -(A \times C) \cdot B = -(C \times B) \cdot A$$

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(D \times (C-A)) \cdot (B-A)} \begin{pmatrix} ((O-A) \times (B-A)) \cdot (C-A) \\ (D \times (C-A)) \cdot (O-A) \\ ((O-A) \times (B-A)) \cdot D \end{pmatrix}$$

Ray Triangle Intersection: Implementation

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(D\times(C-A))\cdot(B-A)} \begin{pmatrix} ((O-A)\times(B-A))\cdot(C-A) \\ (D\times(C-A))\cdot(O-A) \\ ((O-A)\times(B-A))\cdot D \end{pmatrix}$$

- Substituting :

$$E_1 = B - A \qquad E_2 = C - A \qquad S = O - A$$
$$P = D \times (C - A) \qquad Q = (O - A) \times (B - A)$$

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P\cdot E_1} \begin{pmatrix} Q\cdot E_2 \\ P\cdot S \\ Q\cdot D \end{pmatrix}$$

Ray Triangle Intersection: Code

```
bool rayTriIntersect(in O,D, A,B,C, out u,v,t) {
    E1 = B-A
    E2 = C-A
    P = cross(D,E2)
    detM = dot(P,E1)
    if(detM > -eps && detM < eps)
        return false
    f = 1/detM
    S = O-A
    u = f*dot(P,S)
    if(0 > u || 1 < u)
        return false
    Q = cross(S,E1)
    v = f*dot(Q,D)
    if(0 > v || 1 < u+v)
        return false
    t = f*dot(Q,E2)
    return true
```

0 == detM

u outside [0,1]

vectors

scalars

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P \cdot E_1} \begin{pmatrix} Q \cdot E_2 \\ P \cdot S \\ Q \cdot D \end{pmatrix}$$

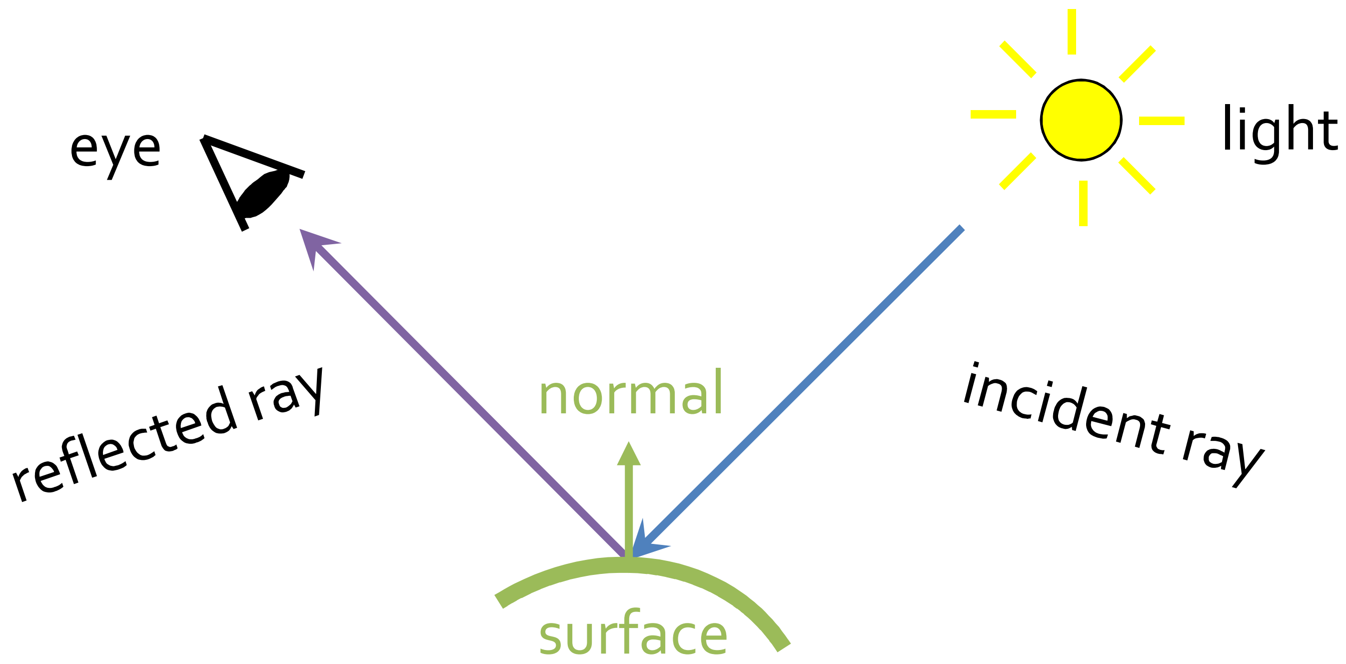
Ray-Casting Method

- 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

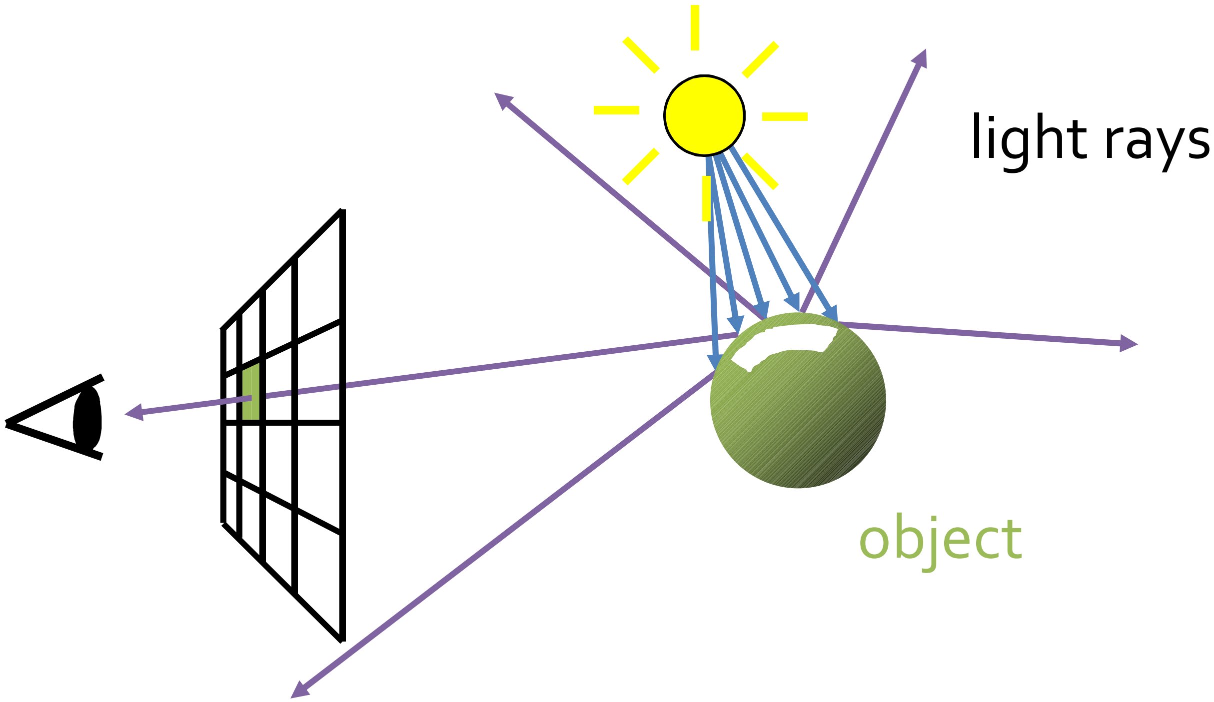
The Basic Idea

- Simulate light rays from light source to eye



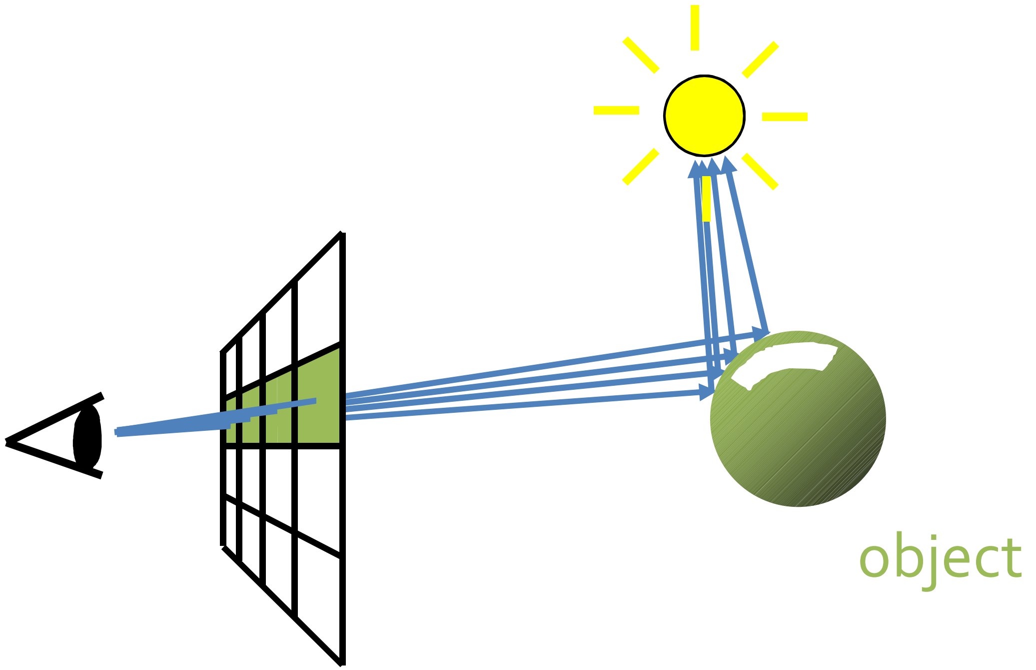
“Forward” Ray-Tracing

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



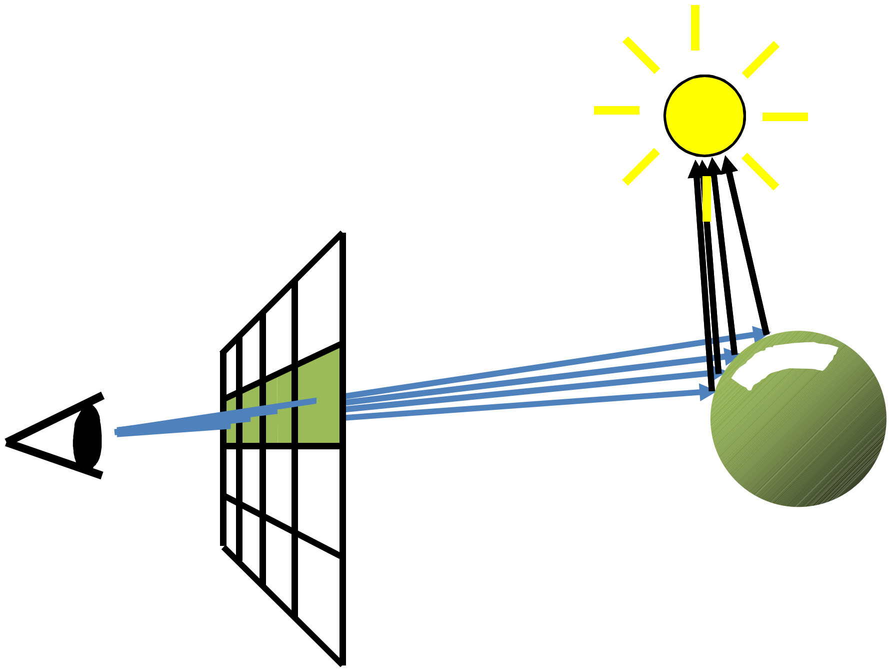
"Backward" Ray-Tracing

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



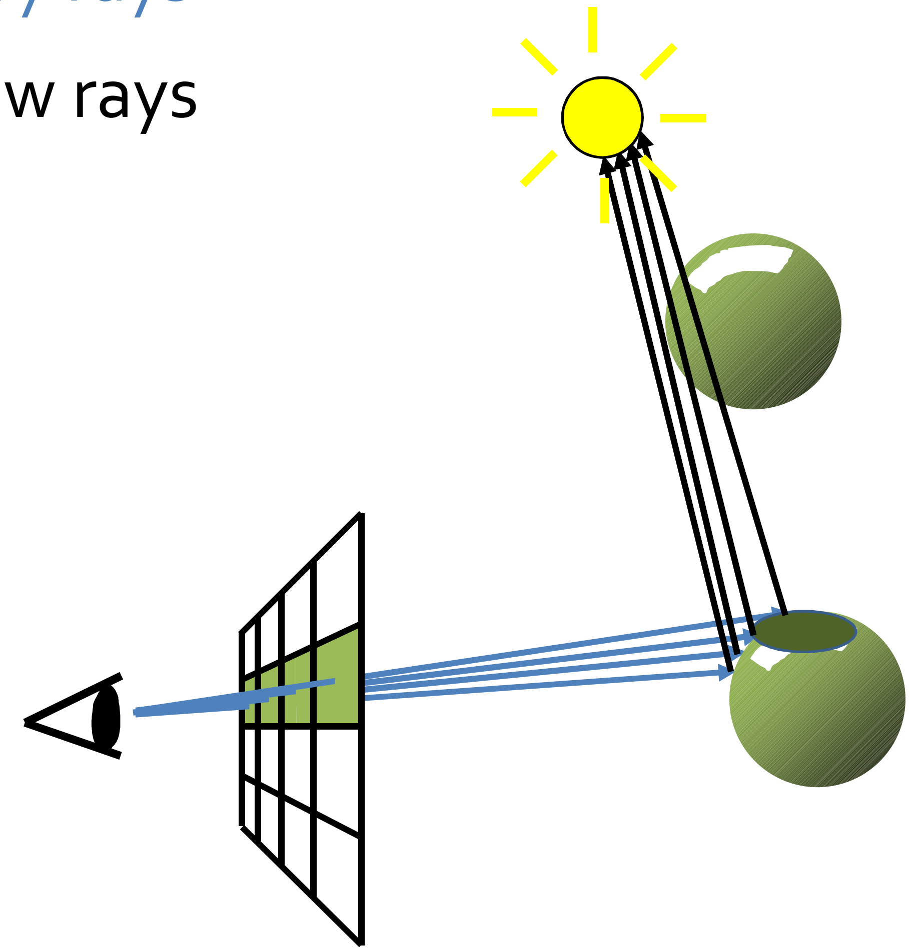
Types of Rays

- Primary rays
- Shadow rays



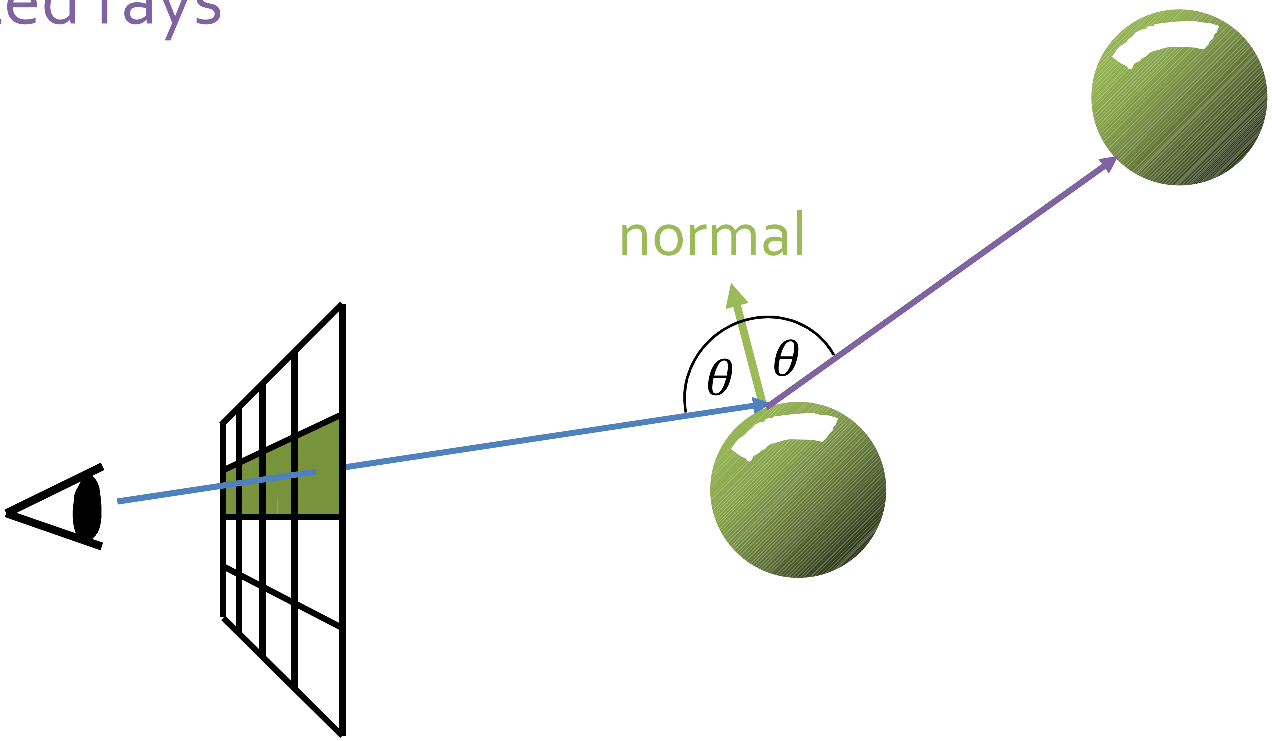
Shadow Rays

- Primary rays
- Shadow rays



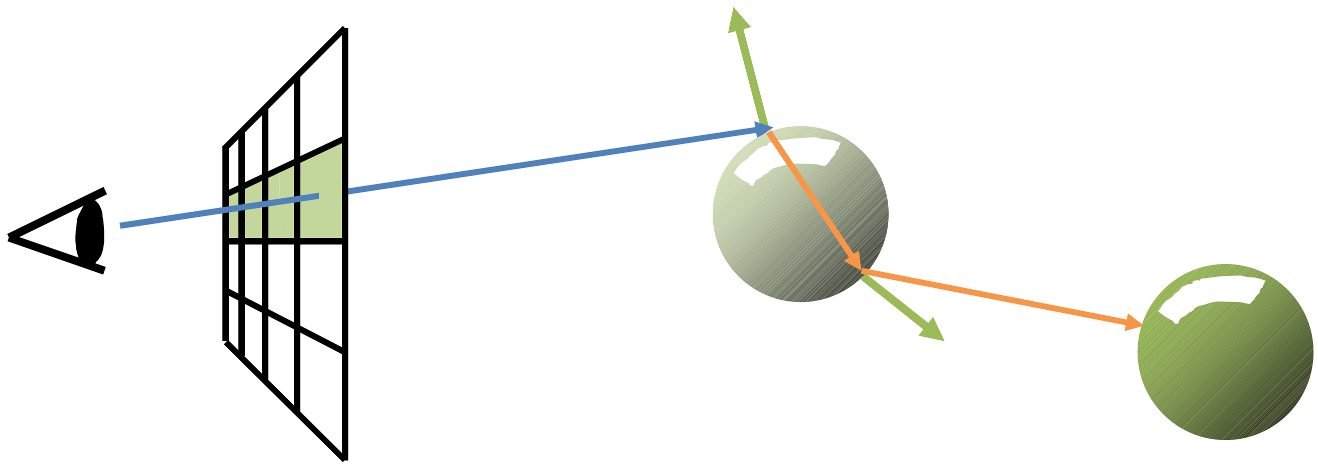
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays



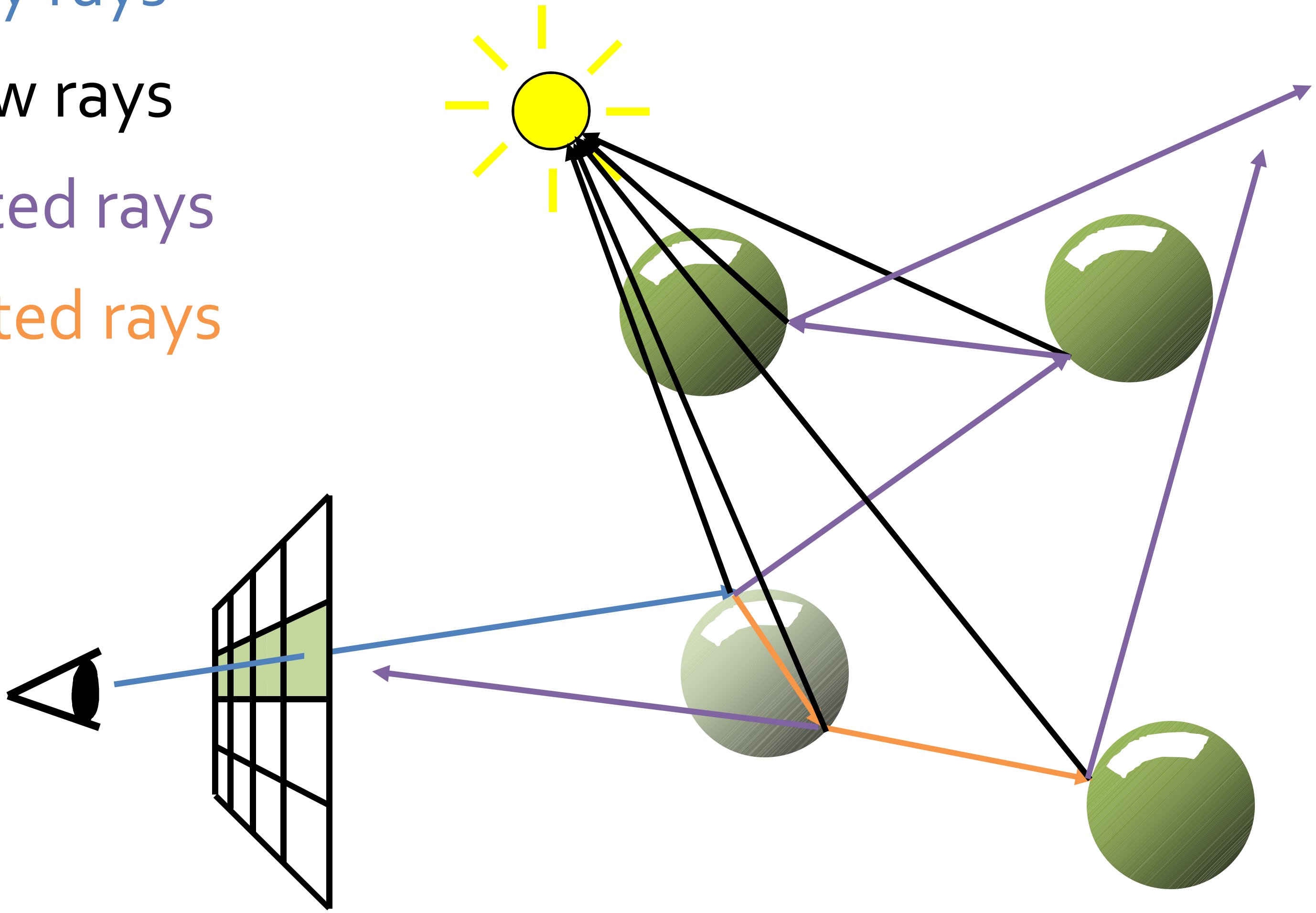
Types of Rays

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



Types of Rays

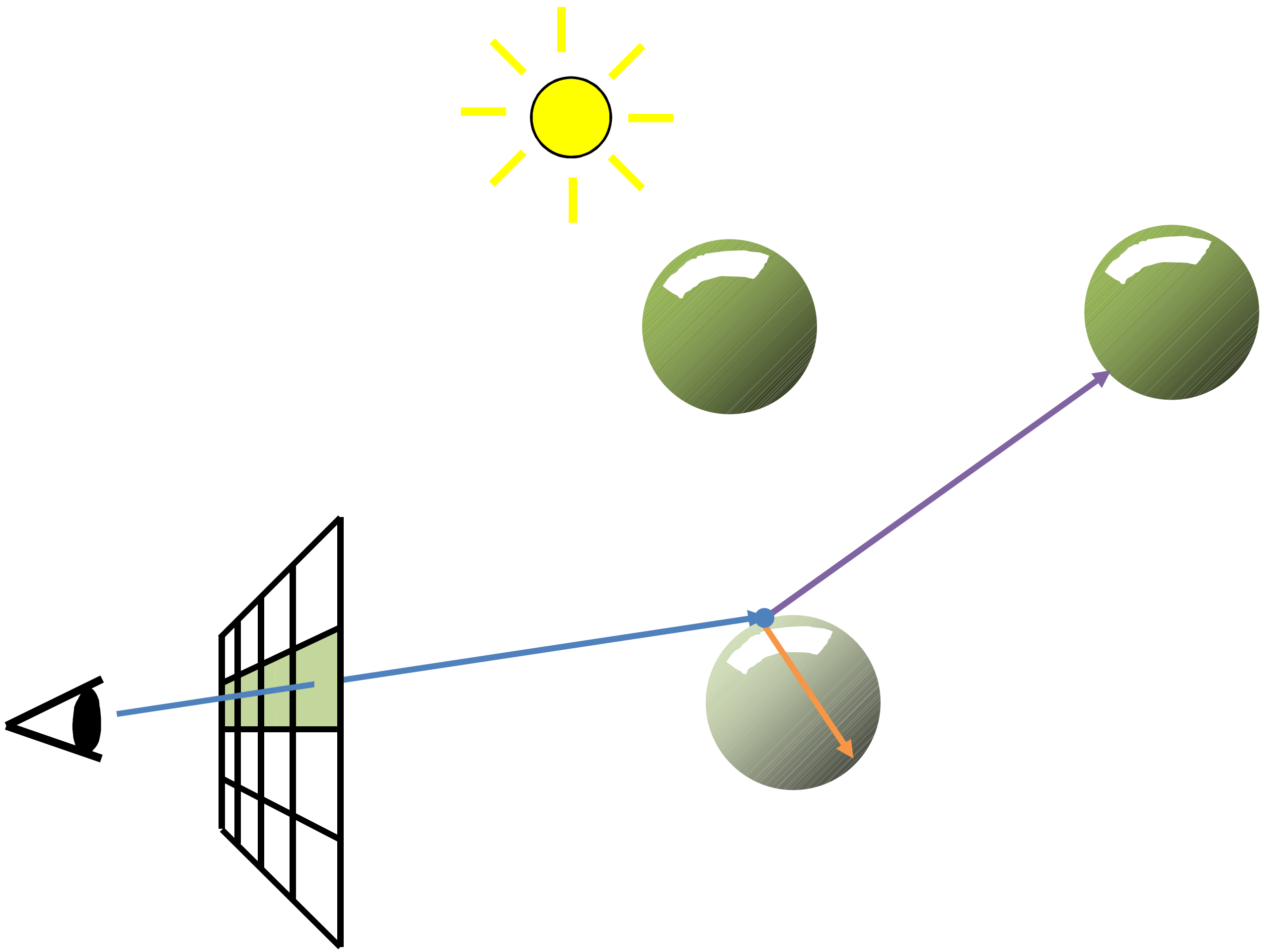
- Primary rays
- Shadow rays
- Reflected rays
- Refracted rays



Lighting

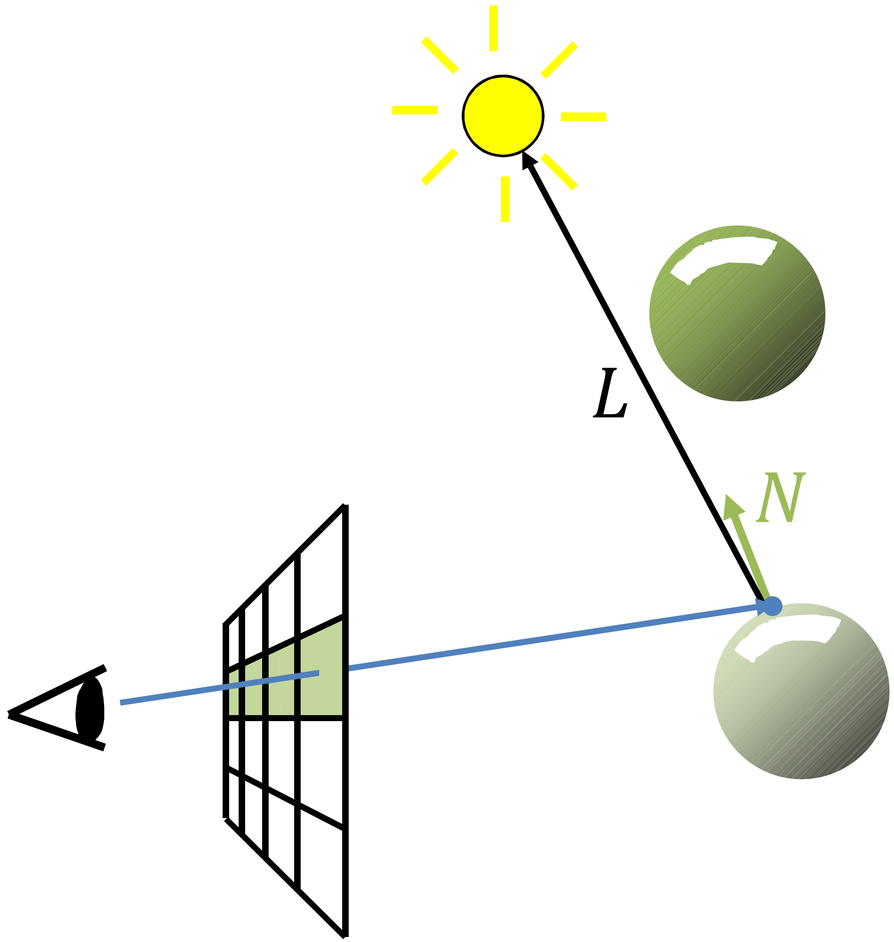
Lighting

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



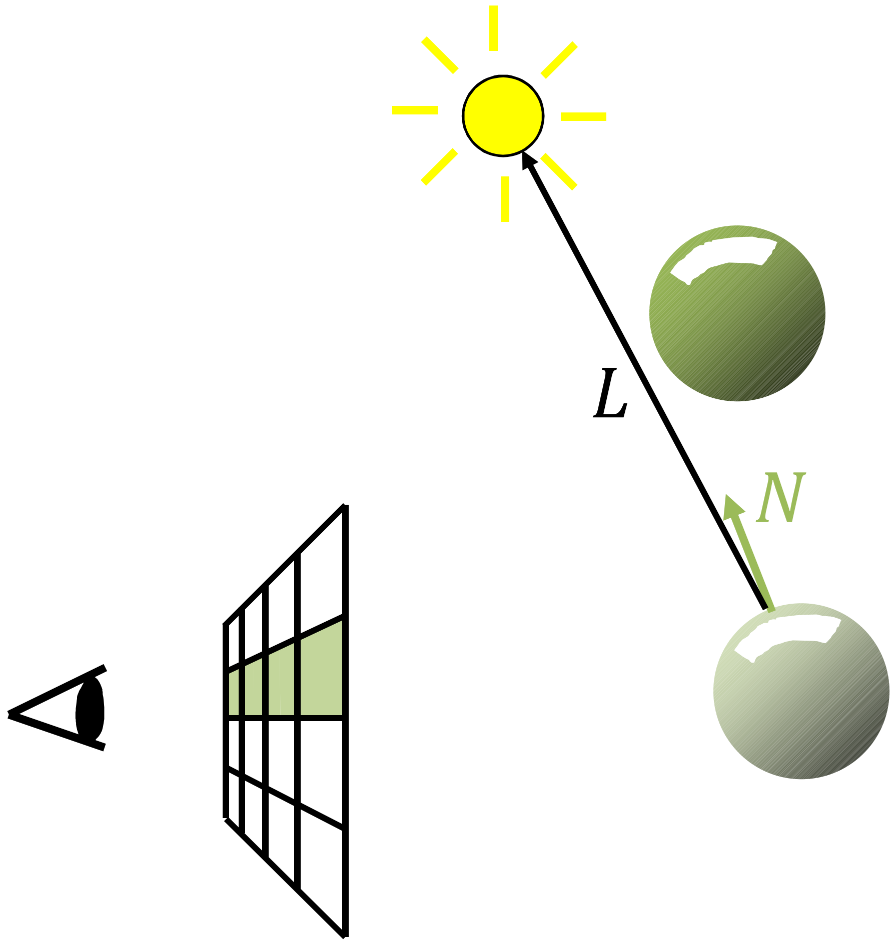
Local – Phong Illumination

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



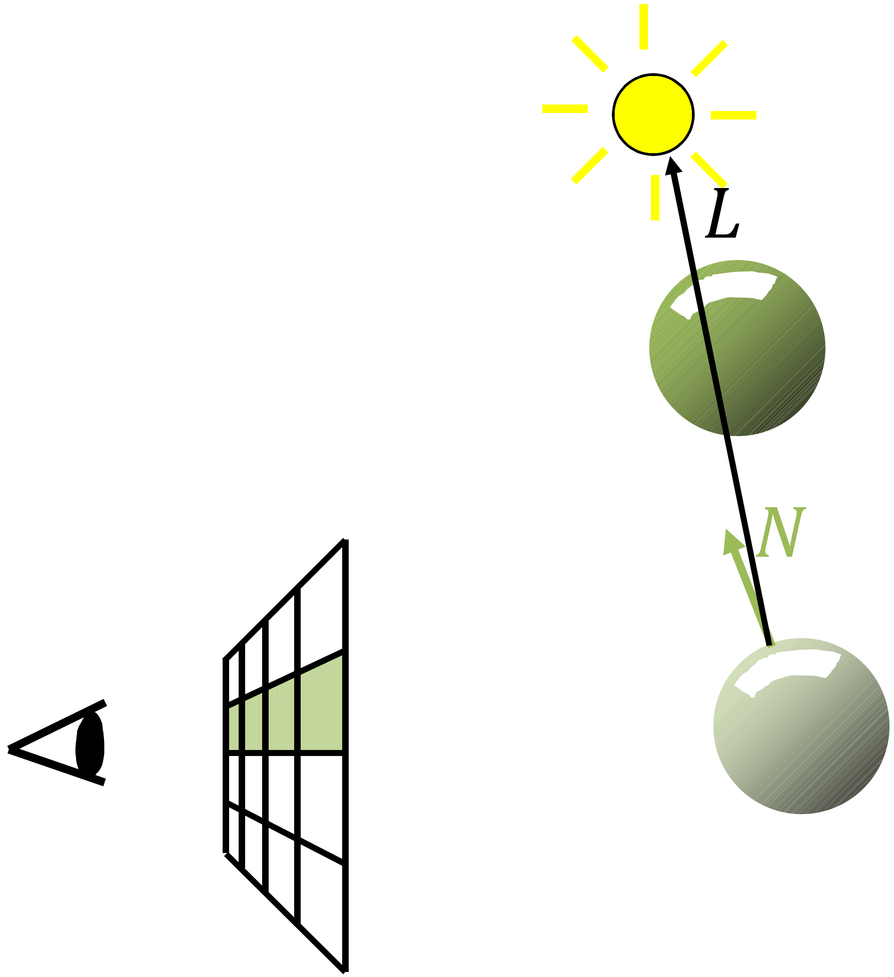
Diffuse (Lambert)

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



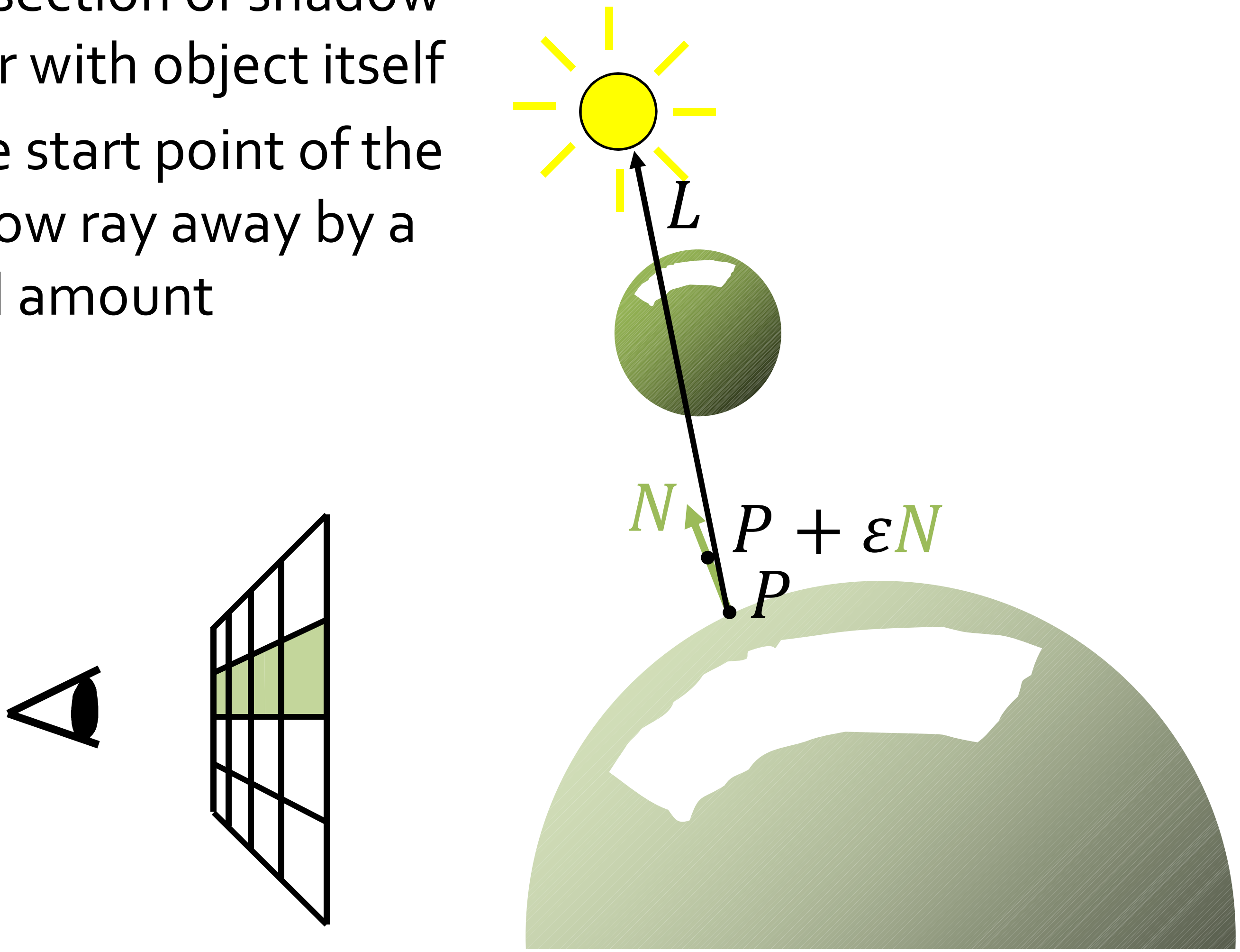
Adding Shadows

- Add local lighting only if point is seen by light



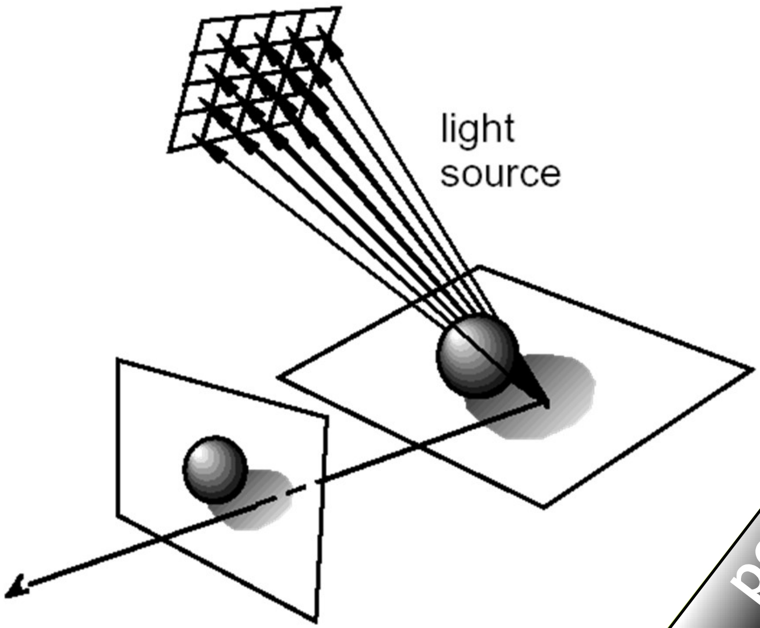
Adding Shadows

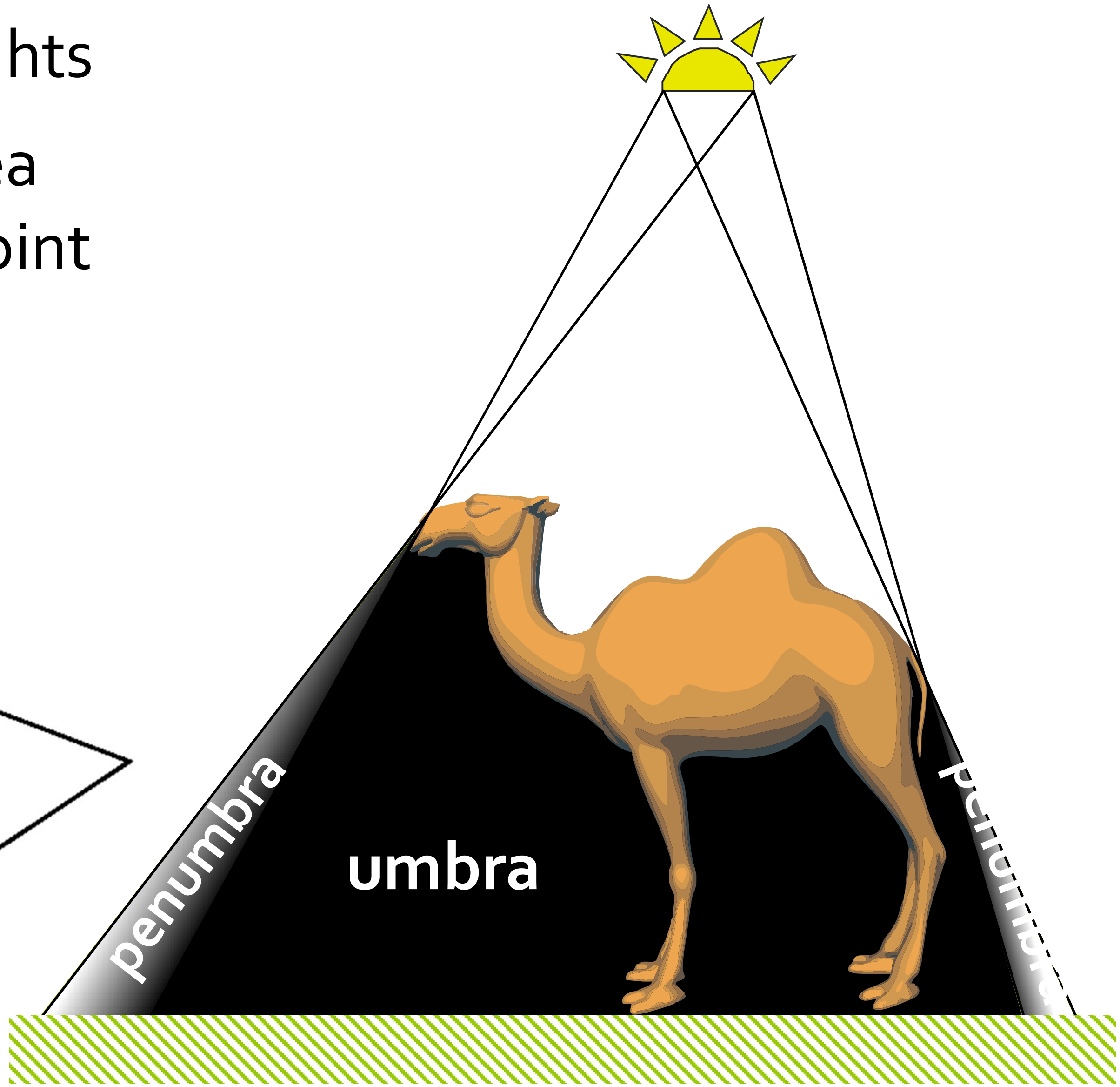
- "Self-Shadowing"
  - Intersection of shadow feeler with object itself
  - Move start point of the shadow ray away by a small amount



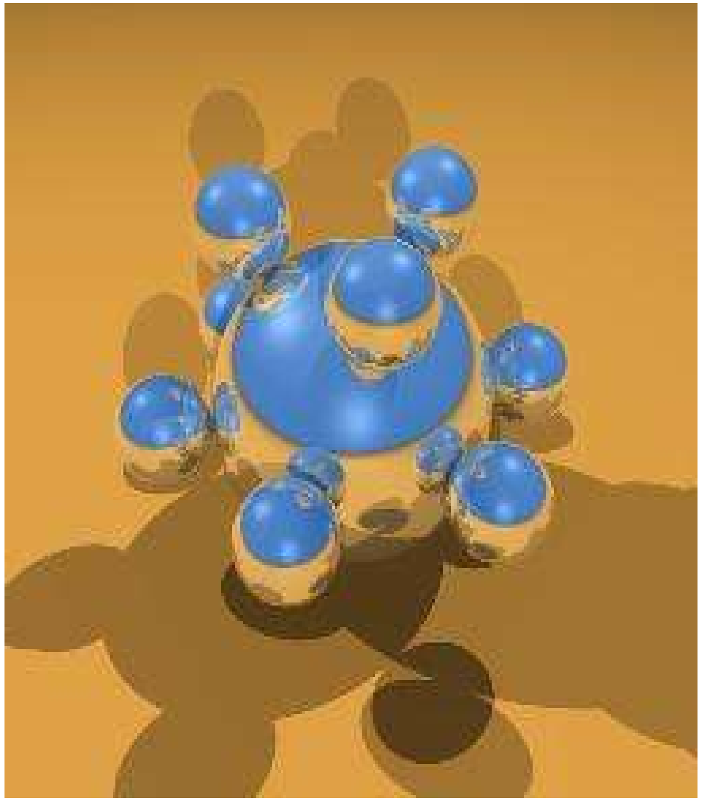
Soft Shadows

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

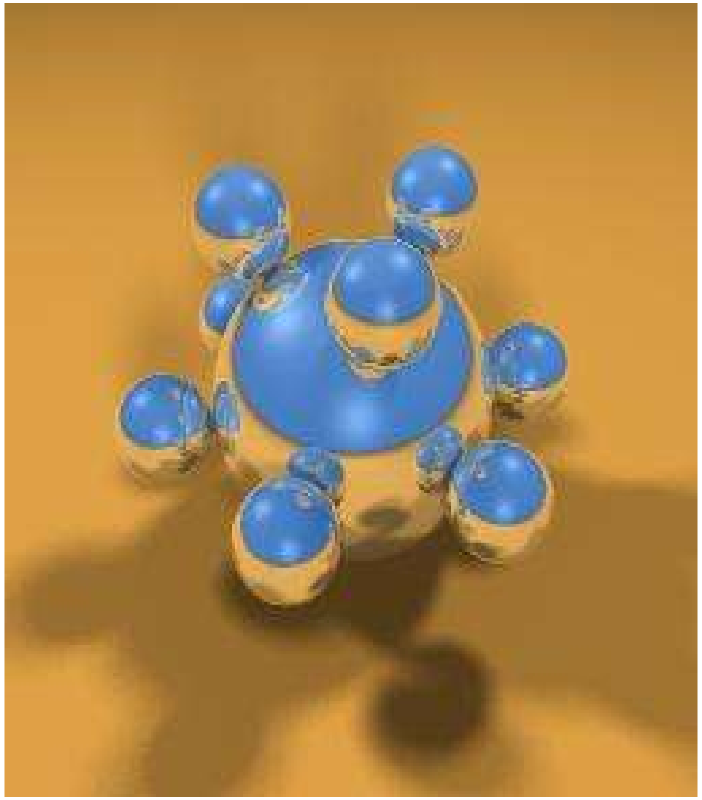




Soft Shadow Example



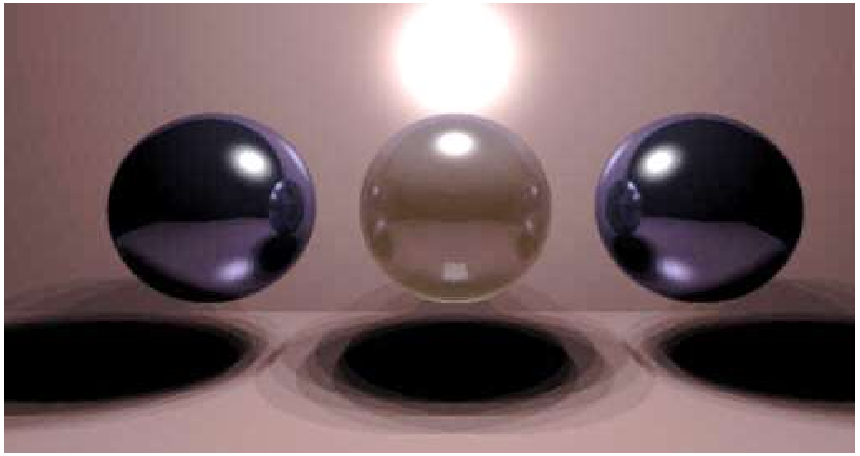
Hard shadow

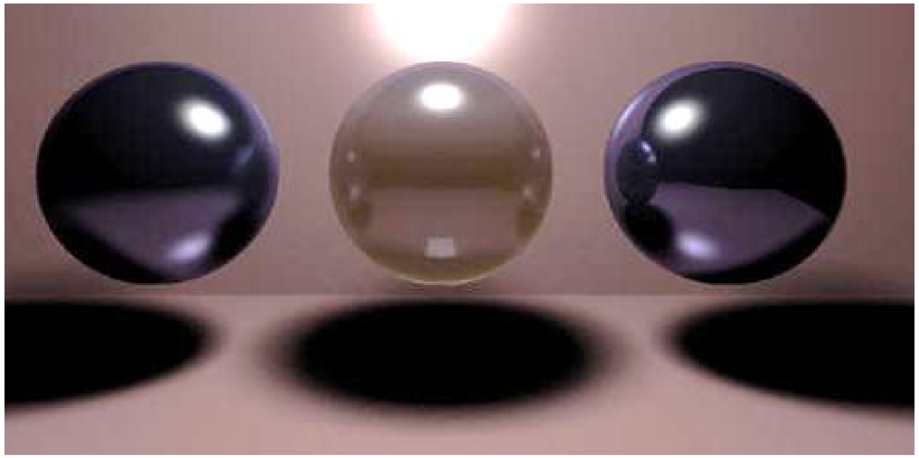


Soft shadow

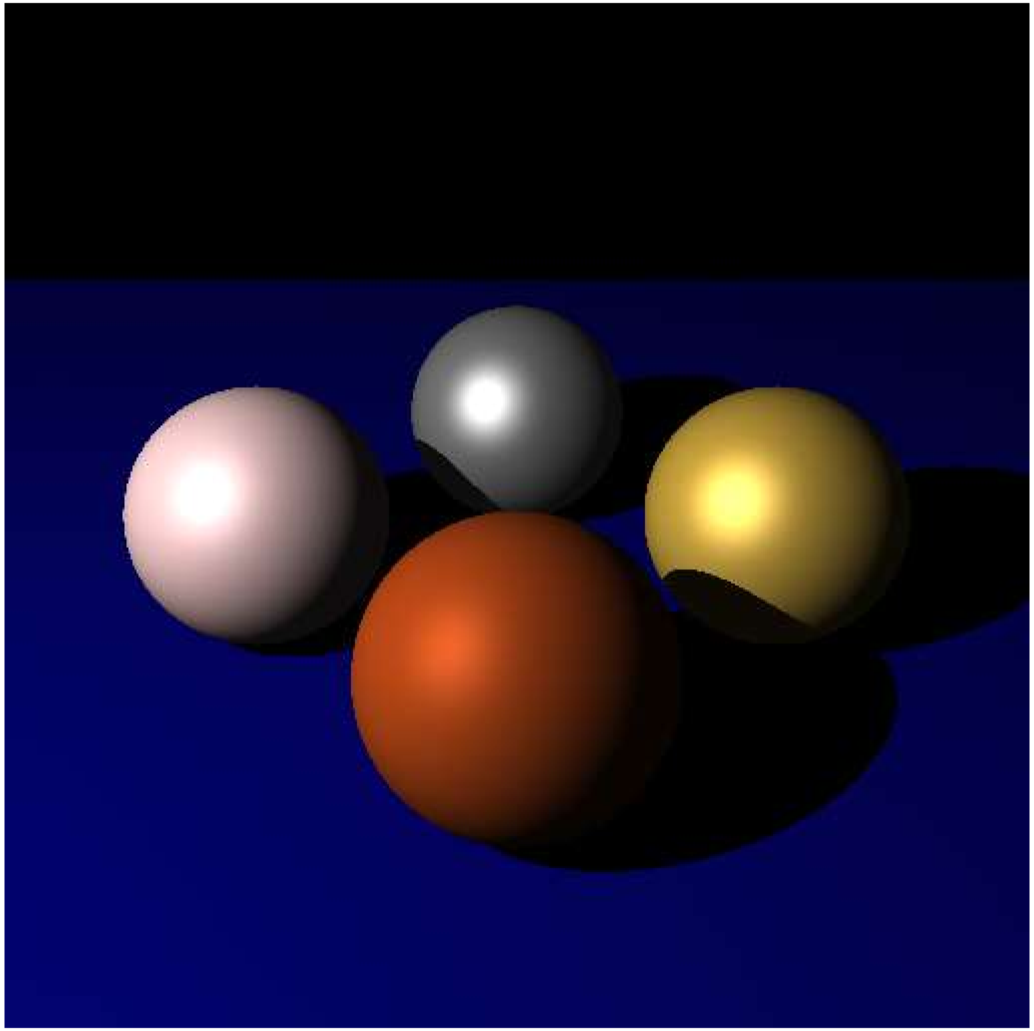
Area Light Sources

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



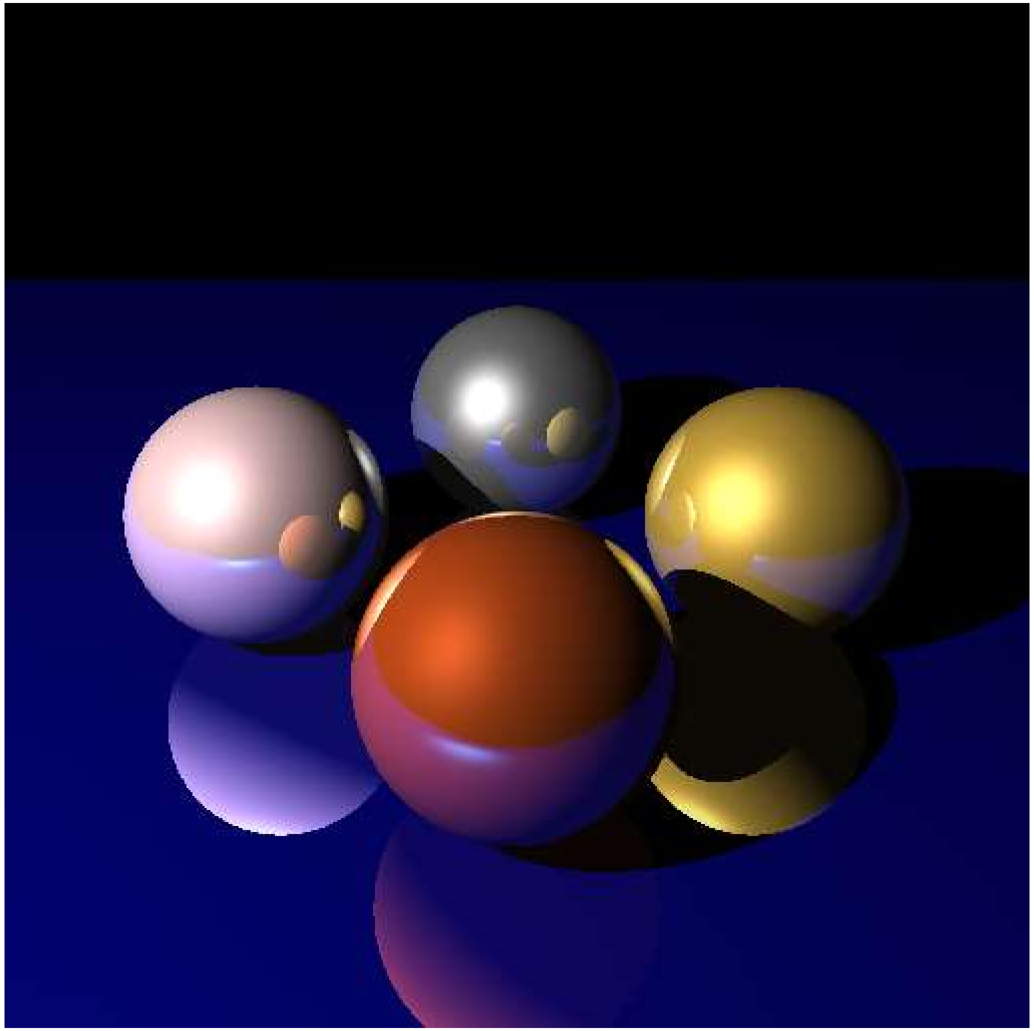


No Reflection



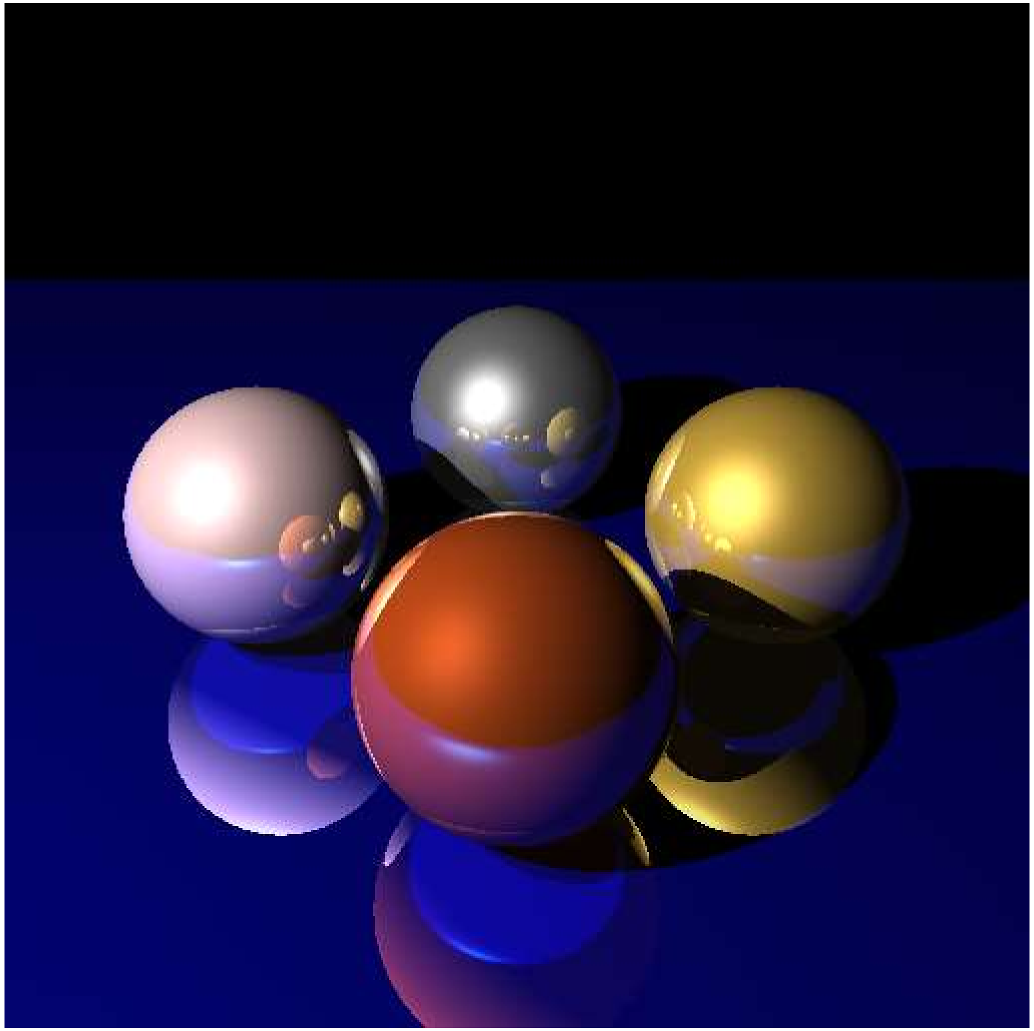
Created by David Derman – CISC 440

Reflection (1)



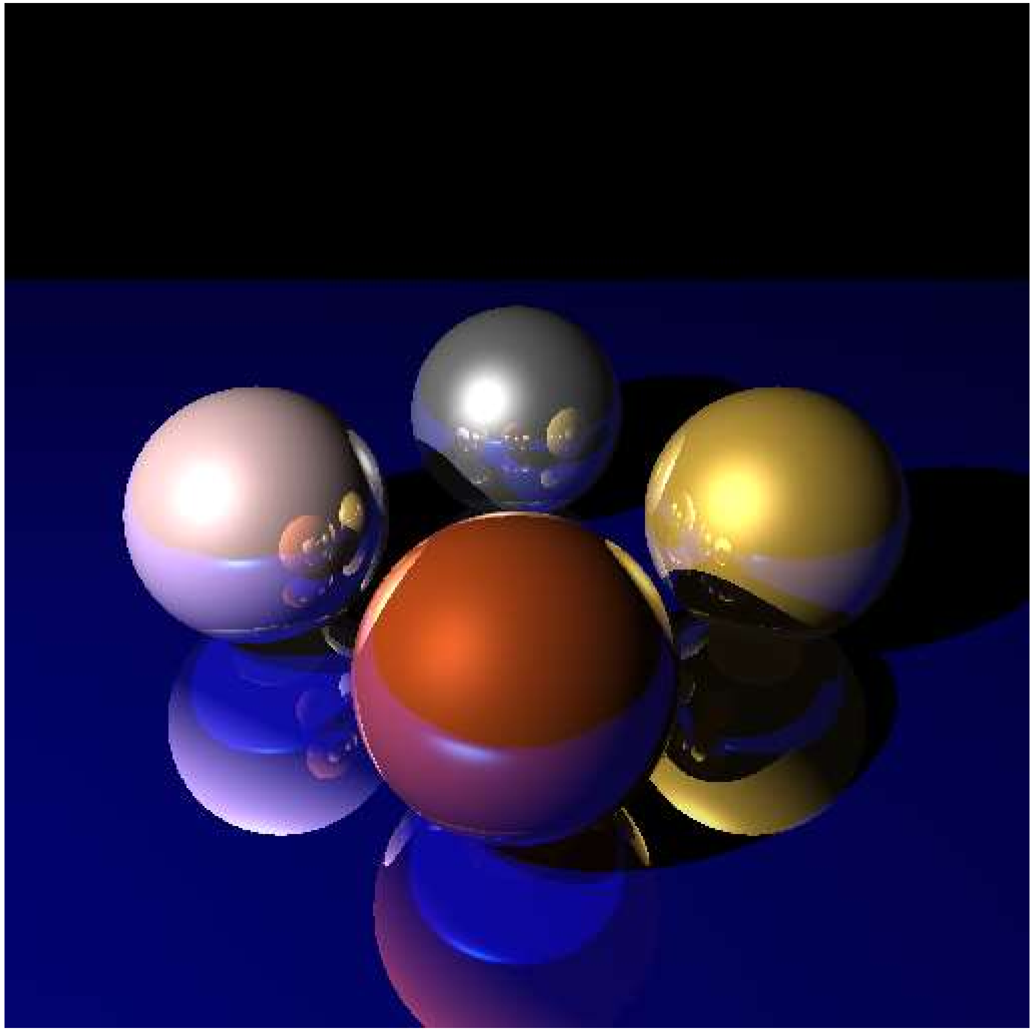
Created by David Derman – CISC 440

Reflection (2)



Created by David Derman – CISC 440

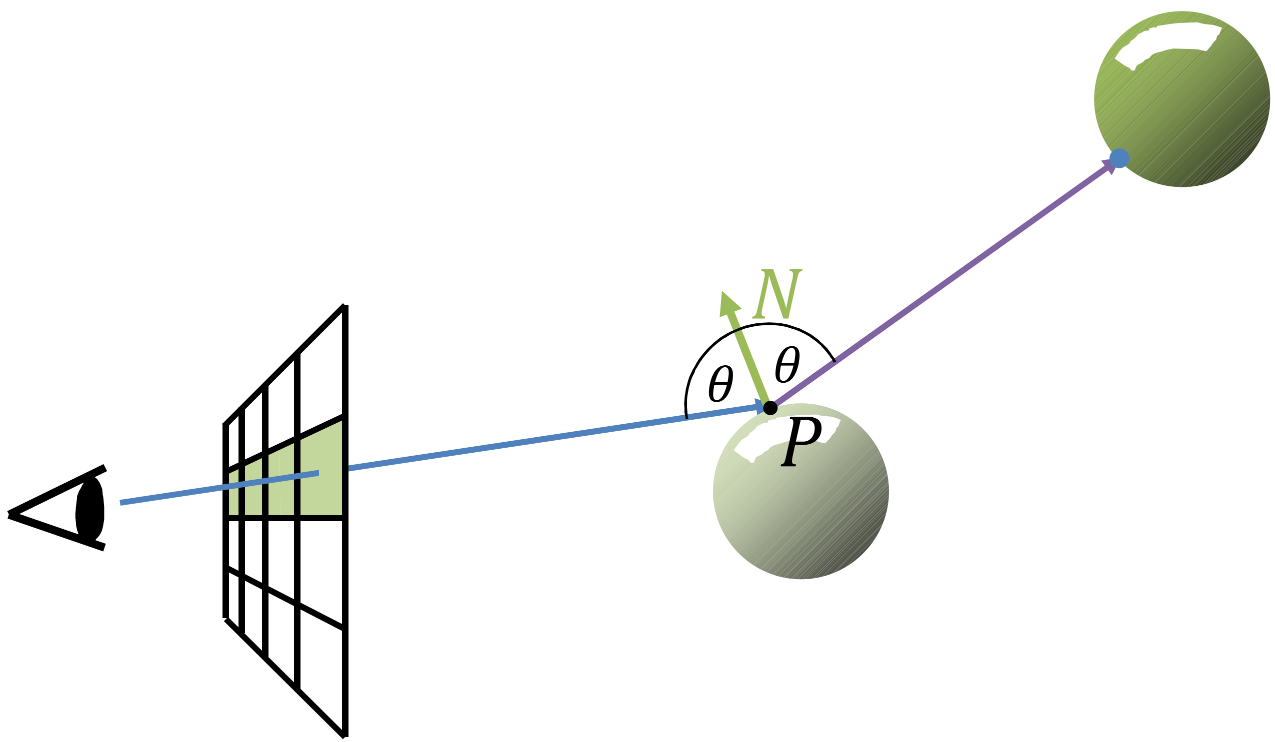
Reflection (3)



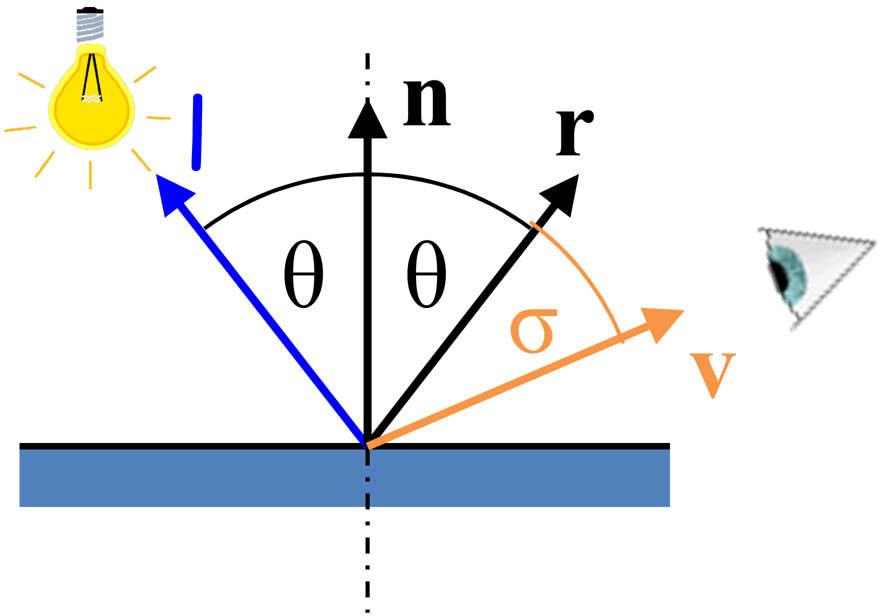
Created by David Derman – CISC 440

Reflection

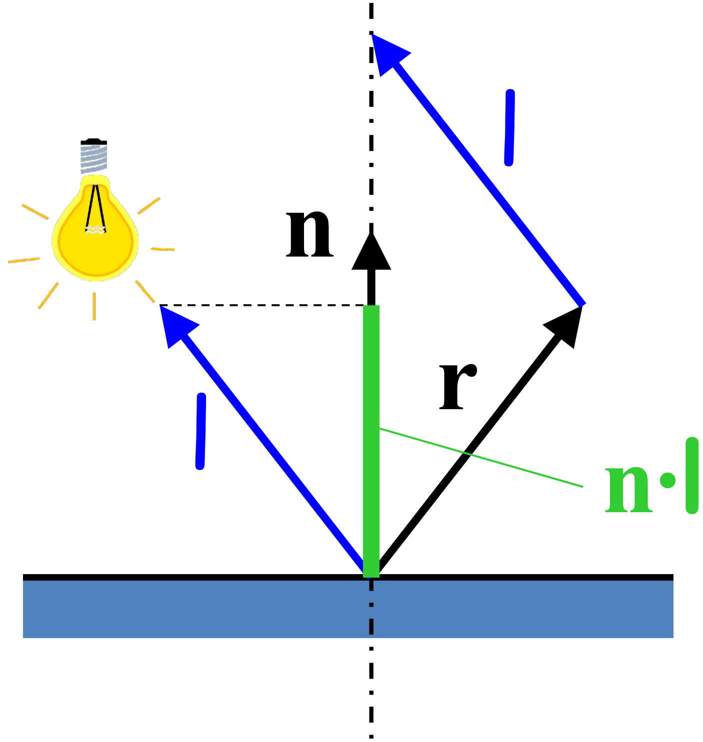
- $C_{reflected} = C_{\text{intersect(P,reflect(}dir\text{,}N\text{))}}$



Reflection direction



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

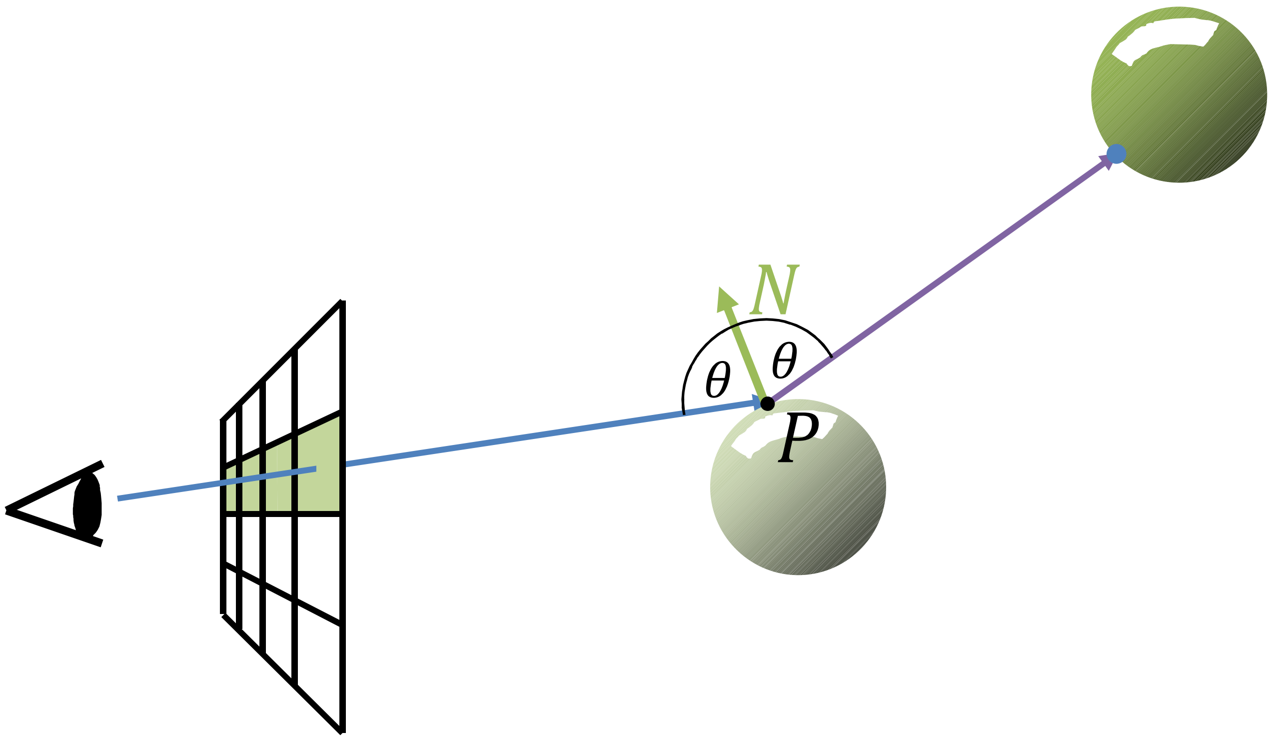


$$\mathbf{r} + \mathbf{l} = (2\mathbf{n} \cdot \mathbf{l})\mathbf{n}$$

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

Reflection

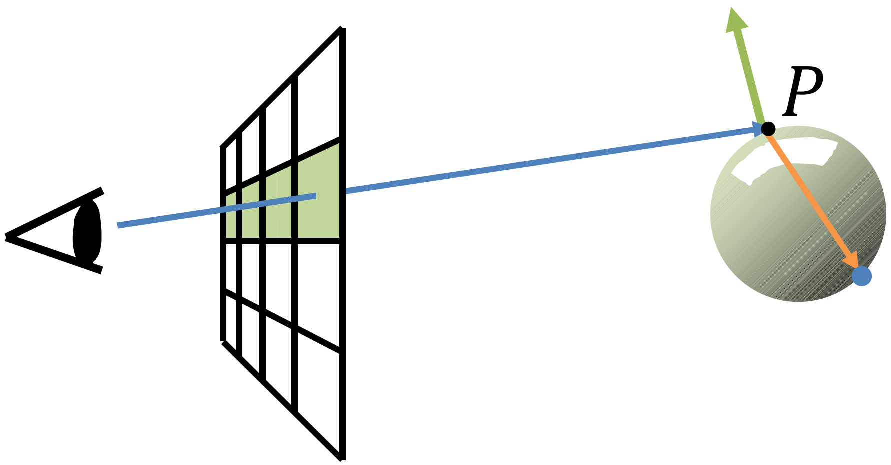
- $\text{reflect}(dir, N) = (2N \cdot -dir)N + dir$
- $C_{reflected} = C_{\text{intersect}(\text{P},\text{reflect}(dir,N))}$



Refraction

Refraction

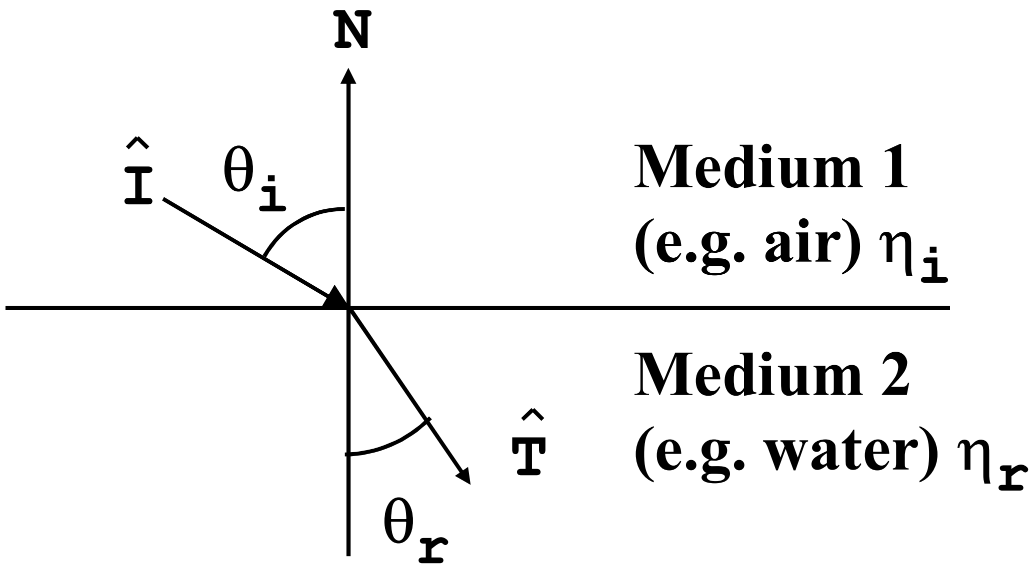
- $C_{refracted} = C_{\text{intersect}(\text{P},\text{refract}(dir,N))}$



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**)

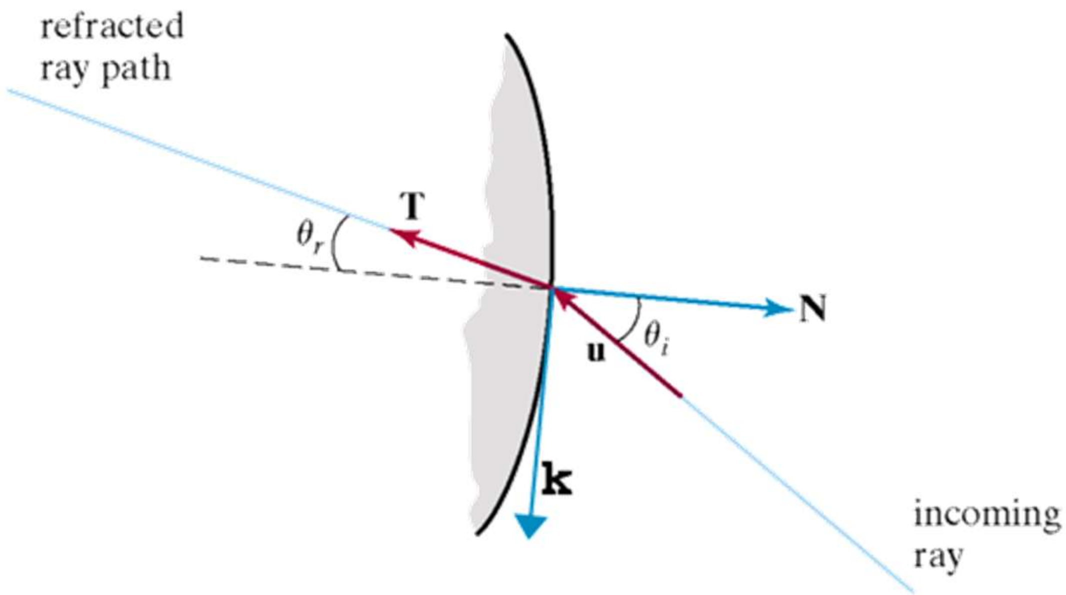
$$\begin{aligned}\mathbf{u} &= (\mathbf{u}\cdot\mathbf{n})(-\mathbf{n}) + (\mathbf{u}\cdot\mathbf{k})(-\mathbf{k}) \\ &= -(\mathbf{u}\cdot\mathbf{k})\mathbf{k} - (\mathbf{u}\cdot\mathbf{n})\mathbf{n} \\ &= -(\sin\theta_i)\mathbf{k} - (\cos\theta_i)\mathbf{n}\end{aligned}$$

- Decomposing the refracted ray (**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 **k** from **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

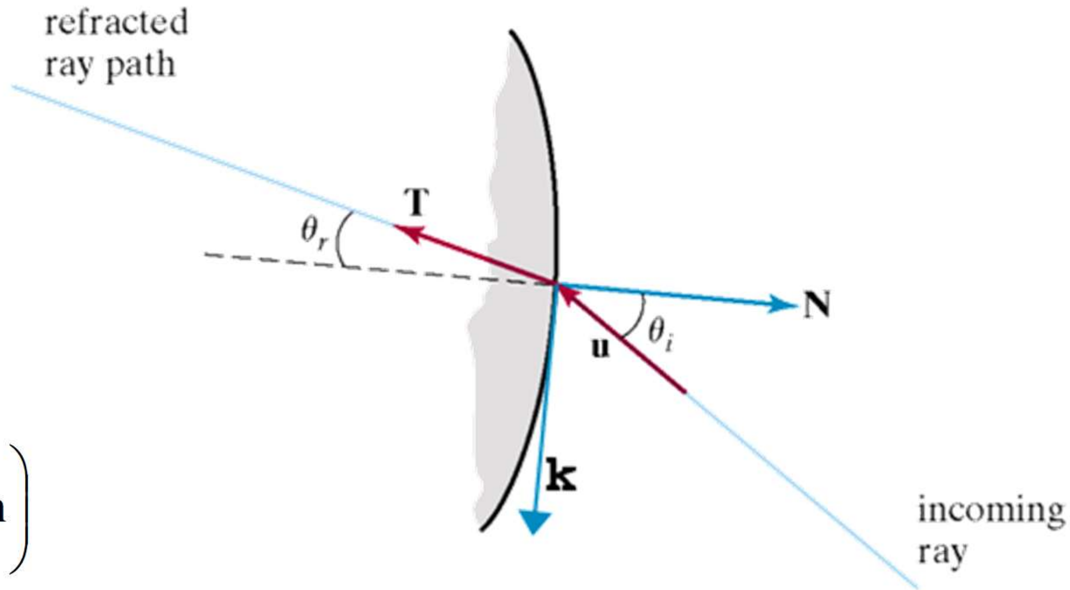
$$\frac{\sin\theta_r}{\sin\theta_i} = \frac{n_i}{n_r}$$

- Solving for **T**

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

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

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



GPU acceleration structures for real-time ray-tracing