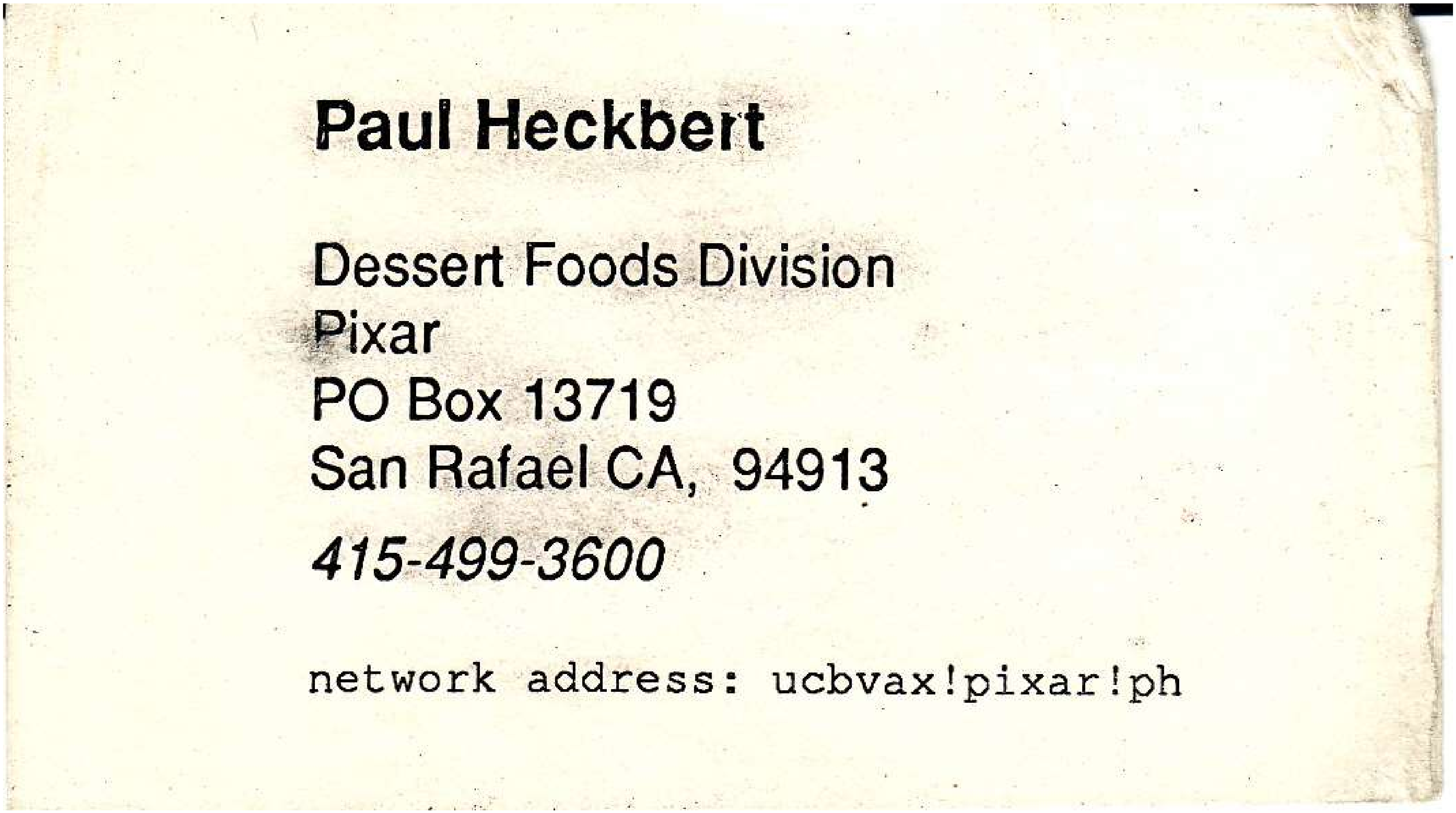
Ray-Tracing

Ray-Tracing – Why Use It?

- Ray-tracing easy to implement
- Simulate rays of light
- Produces natural lighting effects
  - Reflection
  - Refraction
  - Shadows
  - Caustics
  - Depth of Field
  - Motion Blur
- These effects are hard to simulate with rasterization techniques (OpenGL)

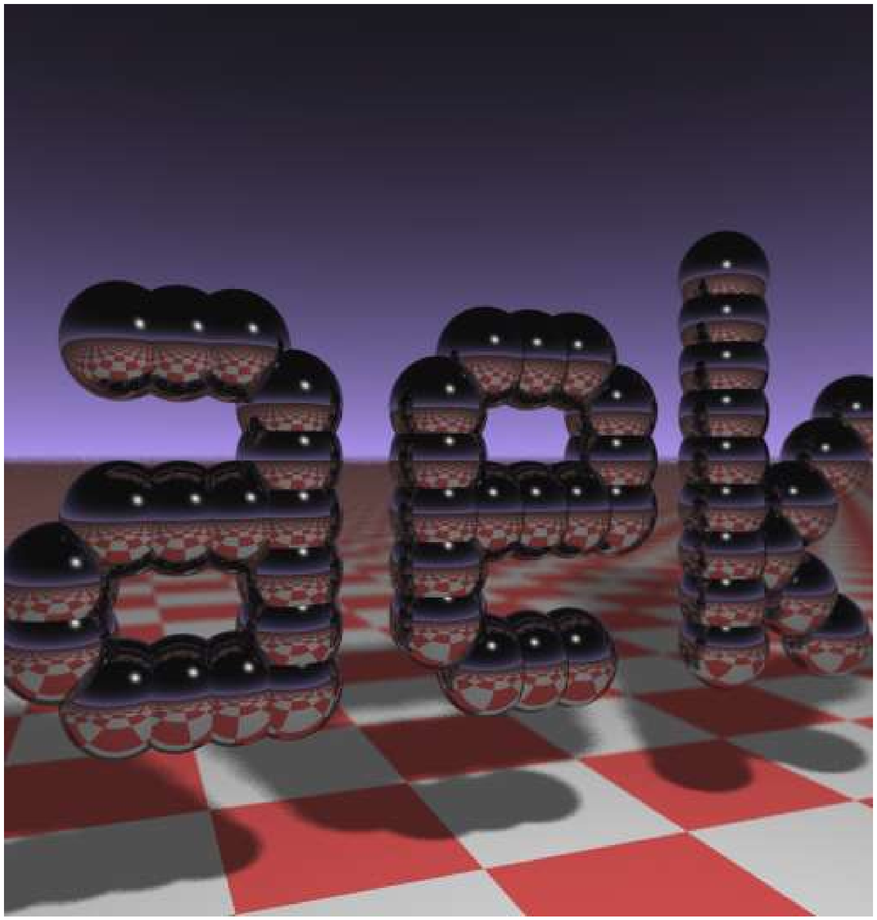
Ray-Tracing – Why Use It?



Ray-Tracing – Why Use It?

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*pixar!ph*/
```

Analysis of the Business Card Ray-Traycer



fabiensanglard.net/rayTracing_back_of_business_card

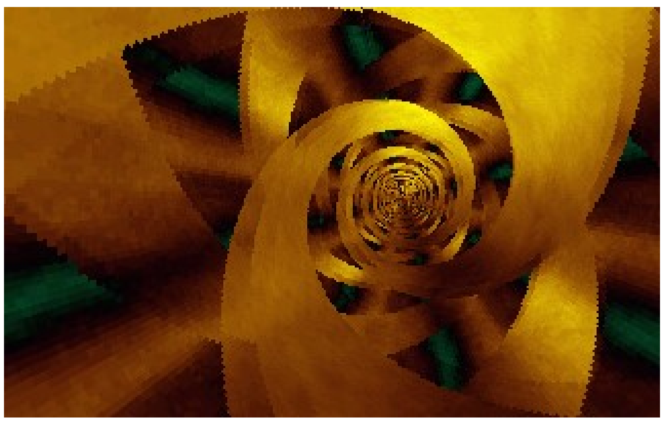
Vector, World, Sampler, Tracer, Main

```
#include <stdlib.h> // card > aek.ppm #include
<stdio.h> #include <math.h> typedef int i;typedef
float f;struct v{ f x,y,z;v operator+(v r){return
v(x+r.x ,y+r.y,z+r.z);}v operator*(f r){return
v(x*r,y*r,z*r);}f operator%(v r){return
x*r.x+y*r.y+z*r.z;}v(){}v operator^(v r ){return
v(y*r.z-z*r.y,z*r.x-x*r.z,x*r. y-y*r.x);}v(f a,f b,f
c){x=a;y=b;z=c;}v operator!()
{return*this*(1/sqrt(*this%* this));}};i
G[]={247570,280596,280600, 249748,18578,18577,
231184,16,16};f R(){ return(f)rand()/RAND_MAX;}i T(v
o,v d,f &t,v&n){t=1e9;i m=0;f p=-o.z/d.z;if(.01
<p)t=p,n=v(0,0,1),m=1;for(i k=19;k--;) for(i j=9;j--
;)if(G[j]&1<<k){v p=o+v(-k ,0,-j-4);f b=p%d,c=p%p-
1,q=b*b-c;if(q>0 ){f s=-b-sqrt(q);if(s<t&&s>.01)
t=s,n=!( p+d*t),m=2;}}return m;}v S(v o,v d){f t ;v
n;i m=T(o,d,t,n);if(!m)return v(.7, .6,1)*pow(1-
d.z,4);v h=o+d*t,l=!(v(9+R( ),9+R(),16)+h*-
1),r=d+n*(n%d*-2);f b=l% n;if(b<0||T(h,l,t,n))b=0;f
p=pow(l%r*(b >0),99);if(m&1){h=h*.2;return((i)(ceil(
h.x)+ceil(h.y))&1?v(3,1,1):v(3,3,3))*(b
*.2+.1);}return v(p,p,p)+S(h,r)*.5;}i
main(){printf("P6 512 512 255 ");v g=!v (-6,-
16,0),a=!(v(0,0,1)^g)*.002,b=!(g^a )*.002,c=(a+b)*-
256+g;for(i y=512;y--;) for(i x=512;x--;){v
p(13,13,13);for(i r =64;r--;){v t=a*(R()-
.5)*99+b*(R()-.5)* 99;p=S(v(17,16,8)+t,!(t*-
1+(a*(R()+x)+b *(y+R())+c)*16))*3.5+p;}
printf("%c%c%c" ,(i)p.x,(i)p.y,(i)p.z);}}
```

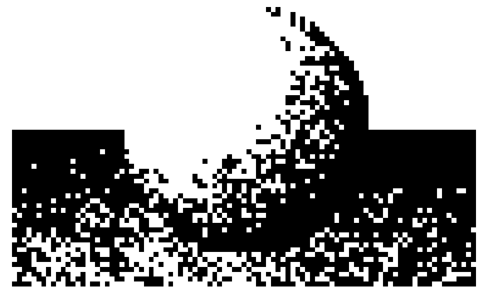
Why Ray Tracing is Great

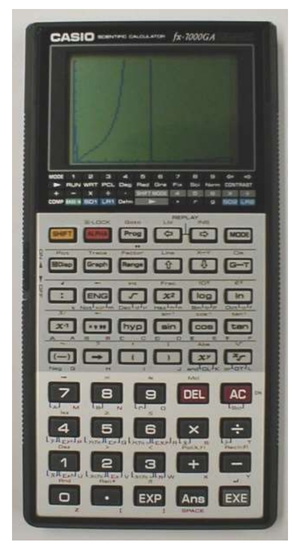
- Size

Tube by Baze



256 byte program

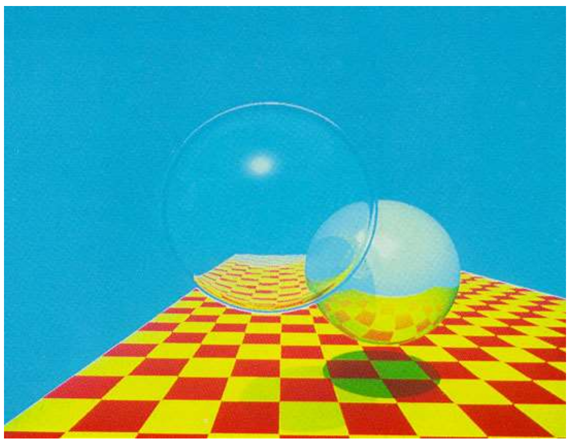




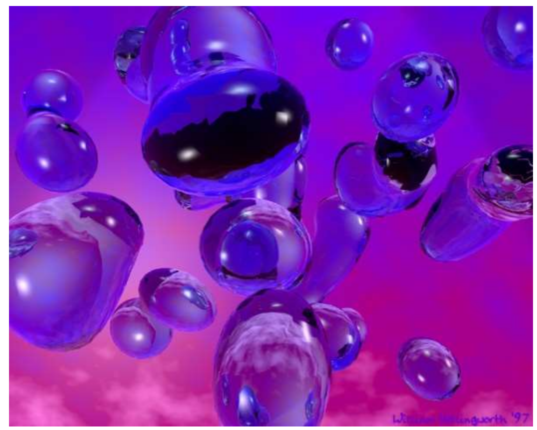
422 byte program for a Casio FX7000Ga, Stéphane Gourichon, 1991

Why Ray Tracing is Great

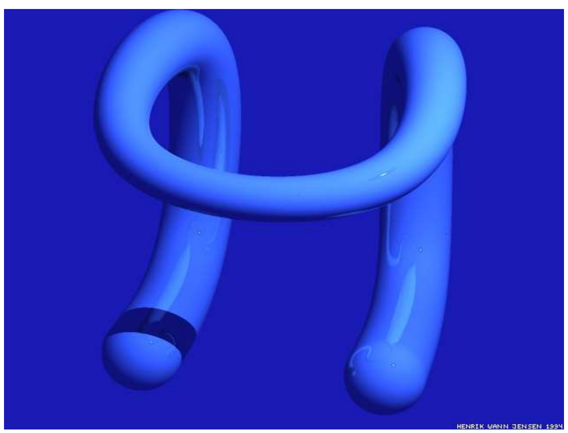
- Shapes: intersectable == renderable



Turner Whitted



William Hollingworth

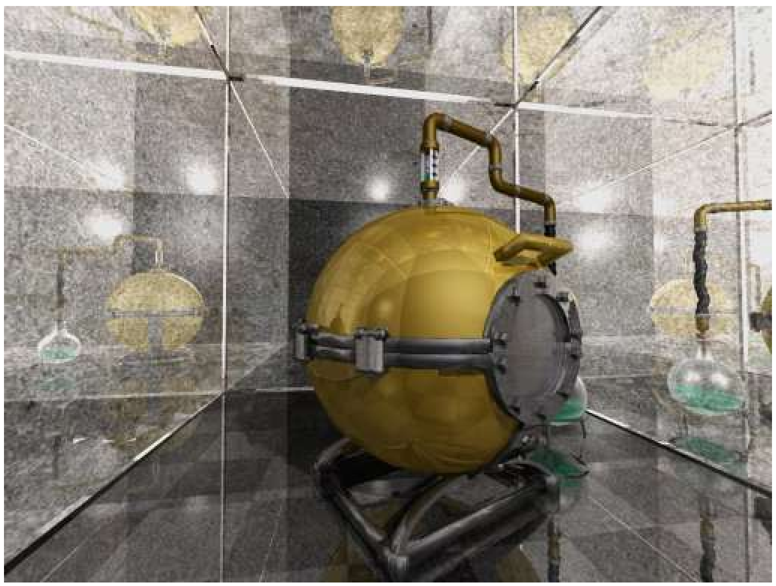


Henrik Wann Jensen

Ken Musgrave

Why Ray Tracing is Great

- Reflections, Refractions



Användare:Mewlek, wikimedia

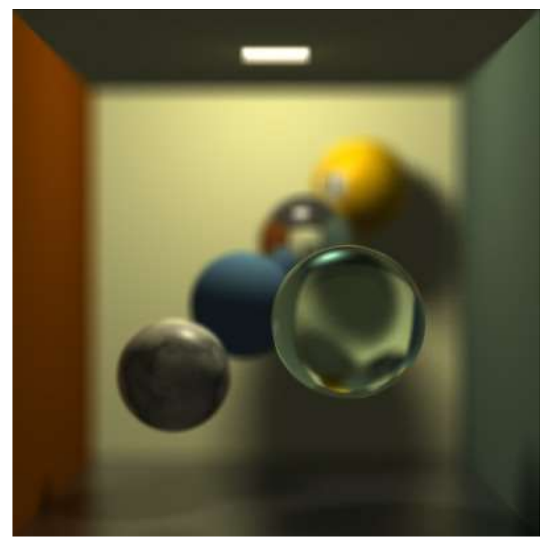
Gilles Tran, wikimedia

Why Ray Tracing is Great

- Stochastic Effects

by Tom Porter based on research by
Rob Cook, Copyright 1984 Pixar

Matt Roberts

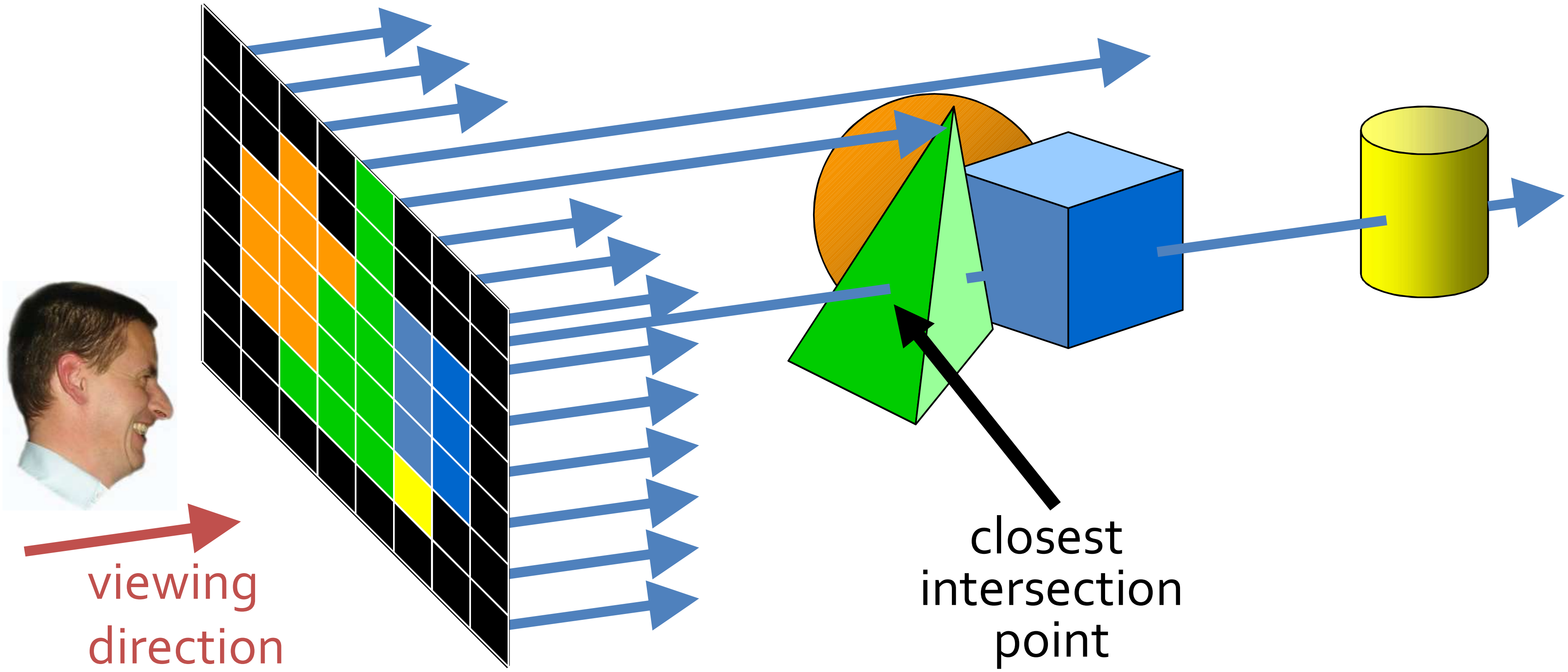


Jason Waltman

Ray-Casting

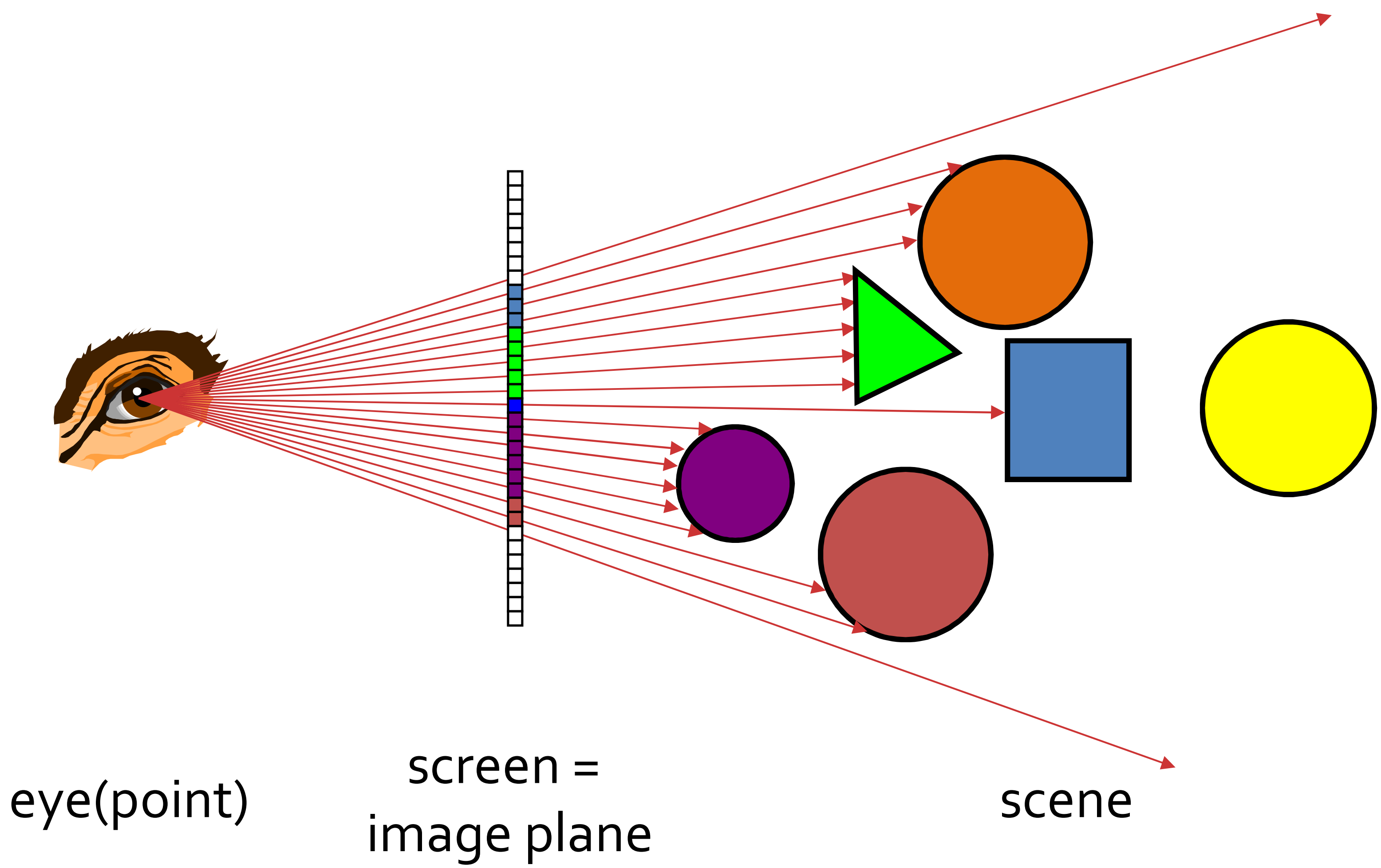
Ray-Casting Method

- Ray from each pixel is intersected with all surfaces
- Calculate color from closest intersected surface
- How Many ray-object intersections?



Ray-Casting – Generating Rays

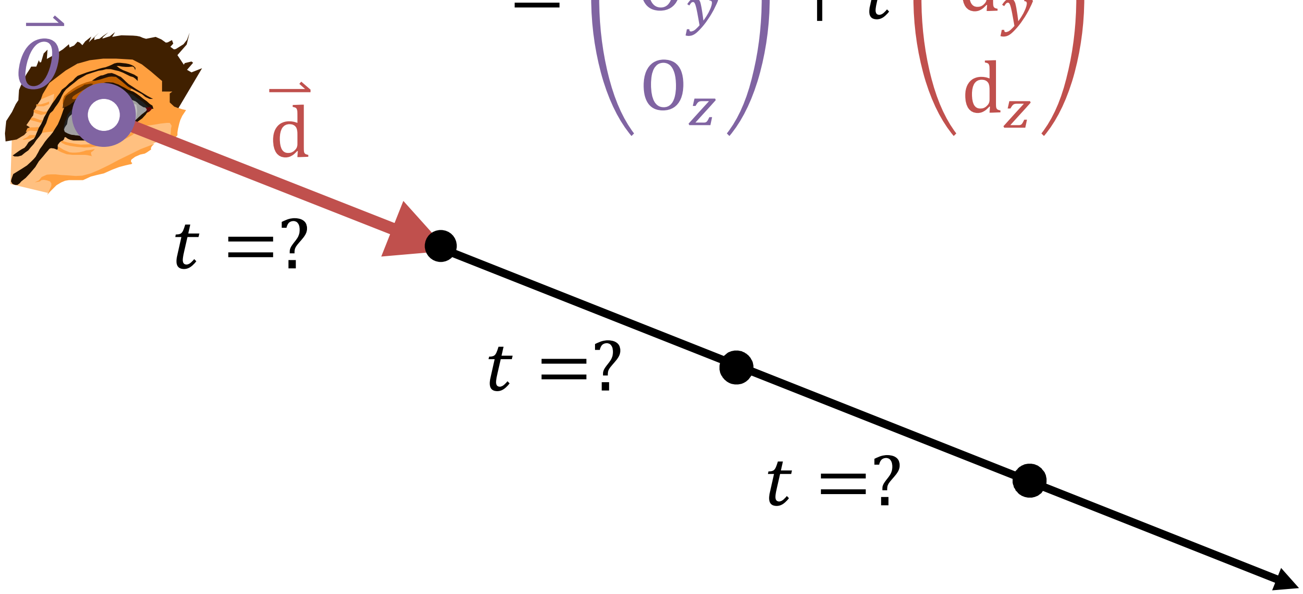
- Trace a ray for each pixel in the image plane



Ray Parametric Form

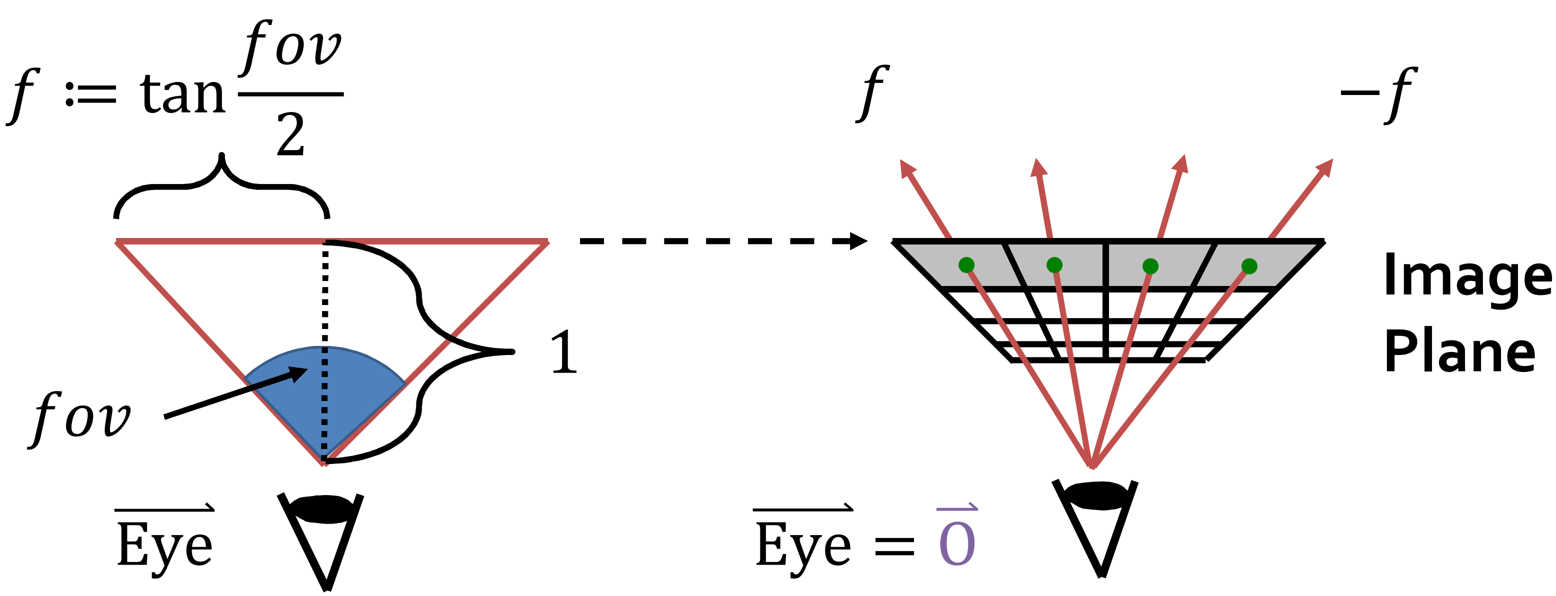
- Ray expressed as function of a single parameter t

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



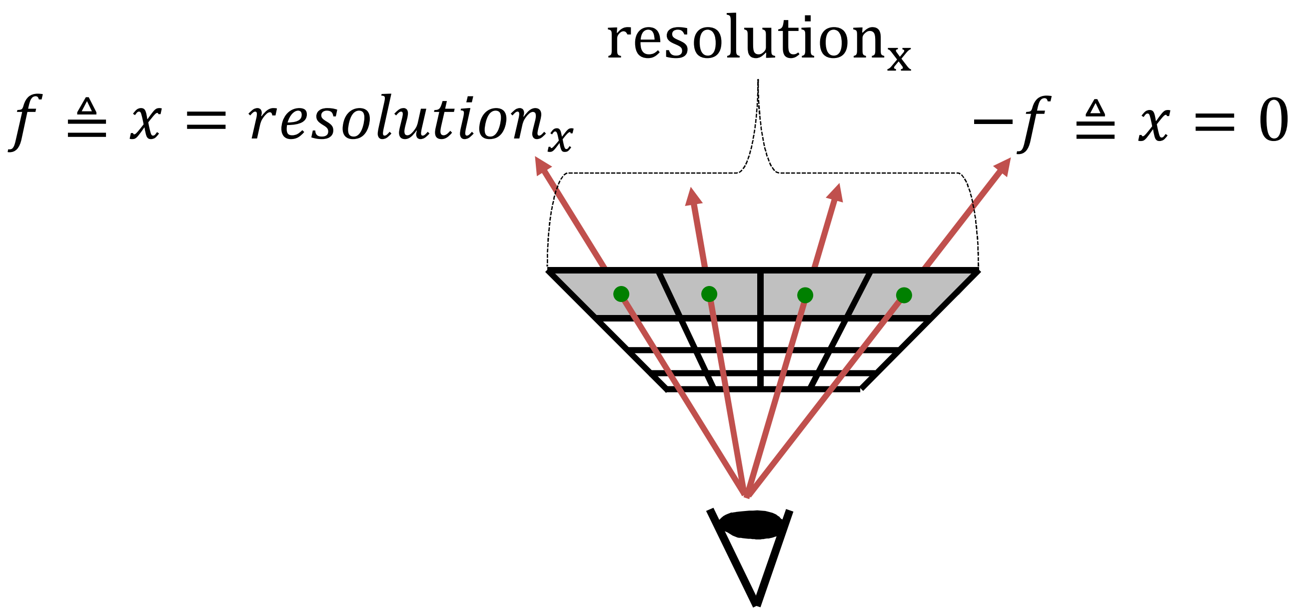
Generating Rays – Top View

- Trace a ray for each pixel in the image plane



Generating Rays – Top View

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



(Looking down from the top)

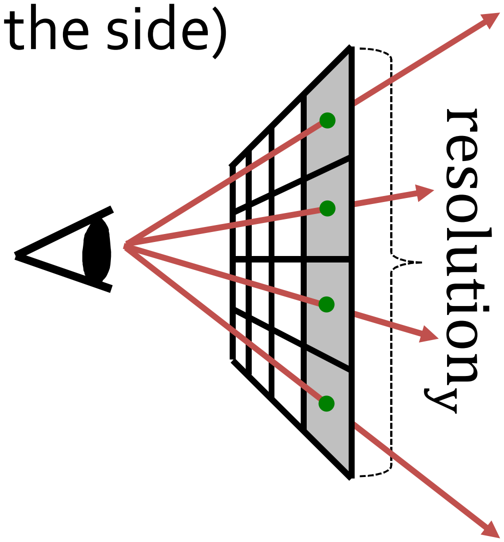
Generating Rays – Side View

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

$$= \frac{\text{resolution}_y}{\text{resolution}_x}\left(\frac{2fy}{resolution_y} - f\right) = \frac{(2y - \text{resolution}_y)f}{\text{resolution}_x}$$

(Looking from the side)

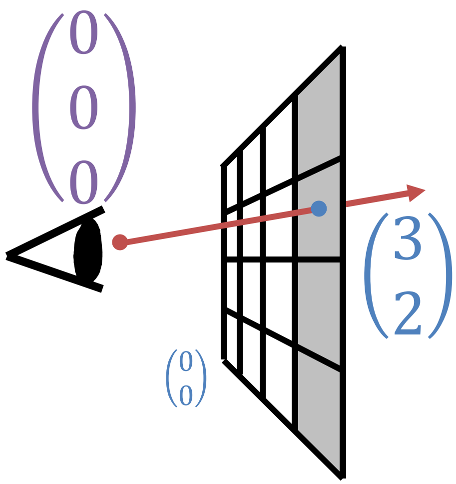
$f * aspect \triangleq y = resolution_y$



$-f * aspect \triangleq y = 0$

Generating Rays

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



Generating Rays

- Trace a ray for each pixel in the image plane

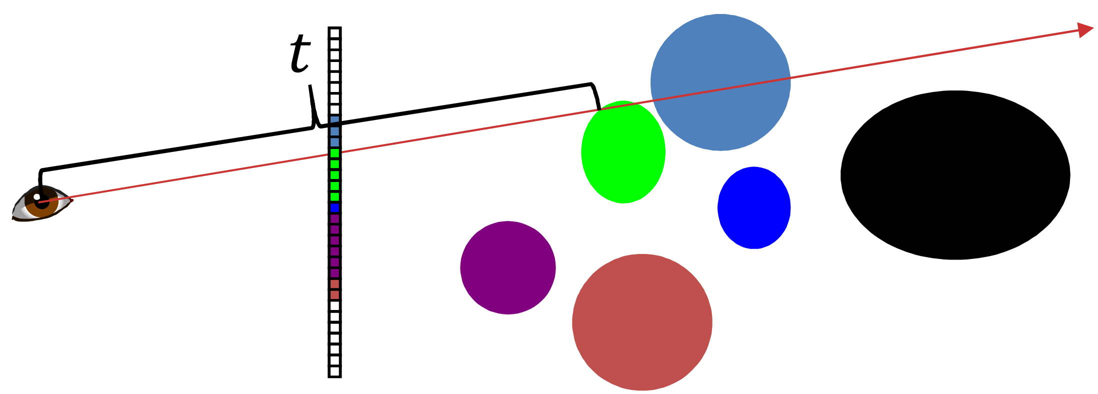
```
renderImage() {
  fov = 90°;
  f = tan(fov / 2) / resolution.x;
  for each pixel x, y in the image
    dx = (2 * x - resolution.x) * f;
    dy = (2 * y - resolution.y) * f;

    ray.O = (0, 0, 0);
    ray.d = normalize(dx, dy, 1);
    image[x][y] = intersect(ray);
}
```

Ray-Object Intersections

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

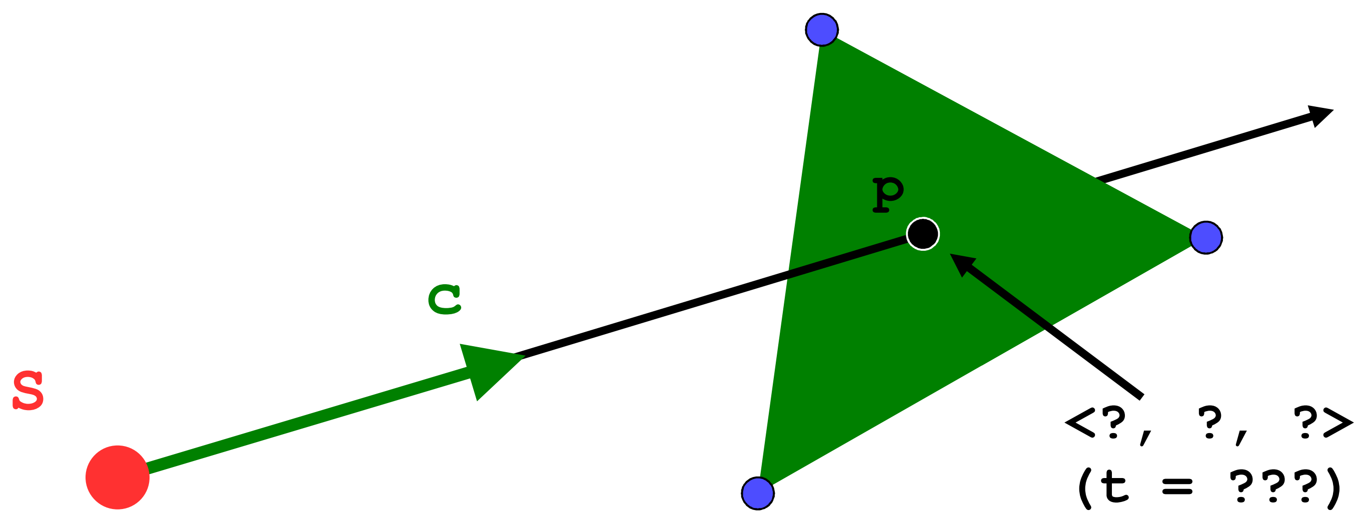
- Ray sphere intersection $|\mathbf{p}-\mathbf{p}_c|^2 = r^2 \quad \mathbf{p} = (x, y, z), \mathbf{p}_c = (a, b, c)$
  - Implicit form of sphere given center *(a,b,c)* and radius *r*
- Intersection with *r(t)* gives $|\mathbf{S}+\mathbf{c}t-\mathbf{p}_c|^2 = r^2$
- By the identity $|\mathbf{a}+\mathbf{b}|^2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

Triangle Intersection

- Want to know: at what *point* ($\mathtt{p}$) 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 + *t*D = (1 – *u* – *v*)A + *u*B + *v*C

- 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

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

- Substituting :

$$E_1 = B - A \qquad E_2 = C - A \qquad S = O - A$$

$$P = \boxed{D \times (C - A)} \qquad Q = \boxed{(O - A) \times (B - A)}$$

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

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

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

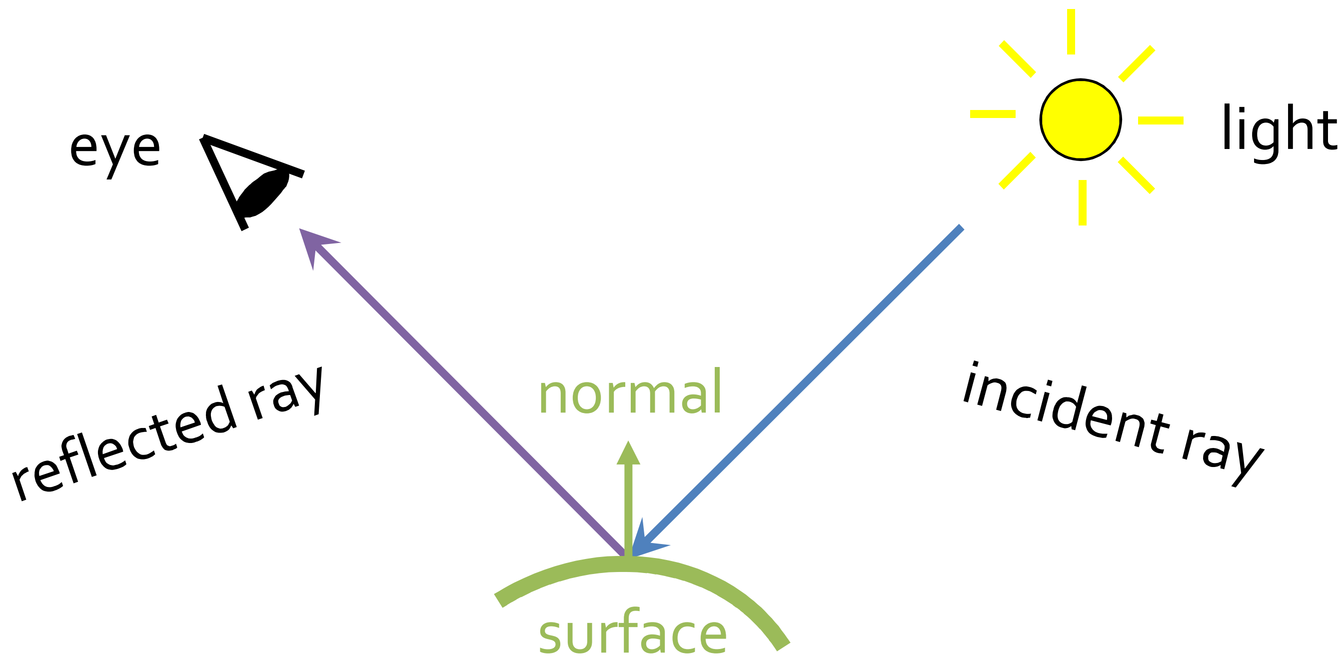
Ray-Casting Method

- 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

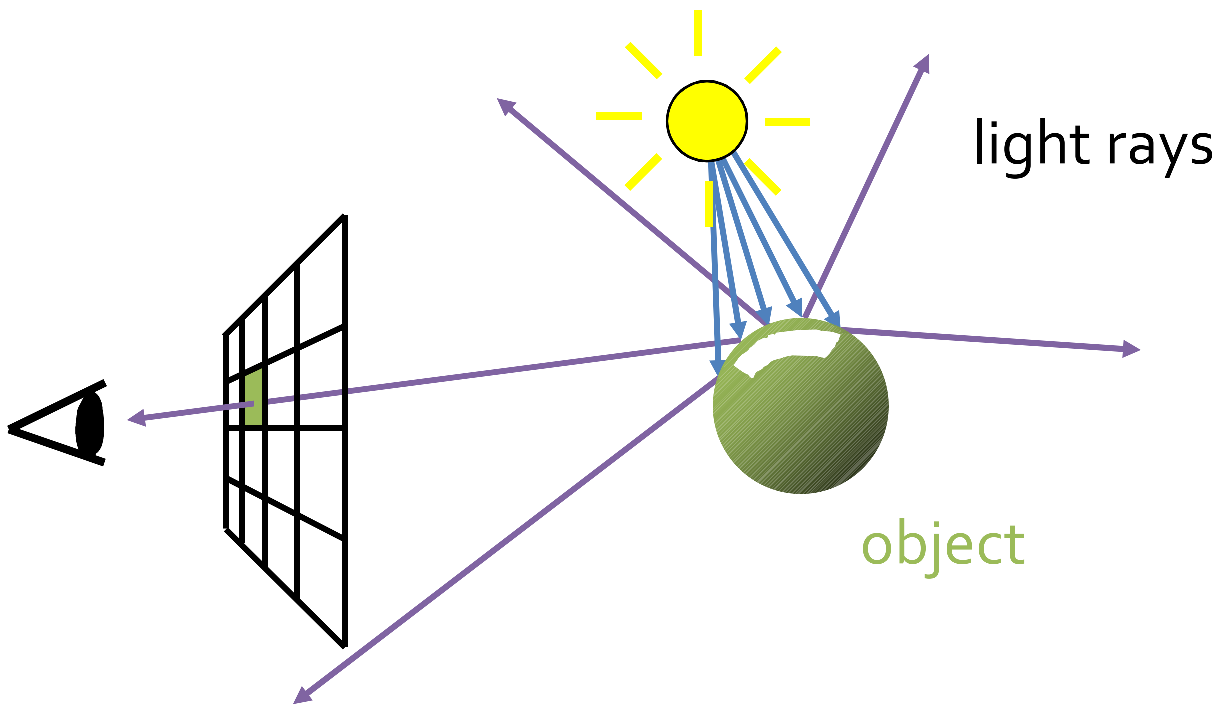
The Basic Idea

- Simulate light rays from light source to eye



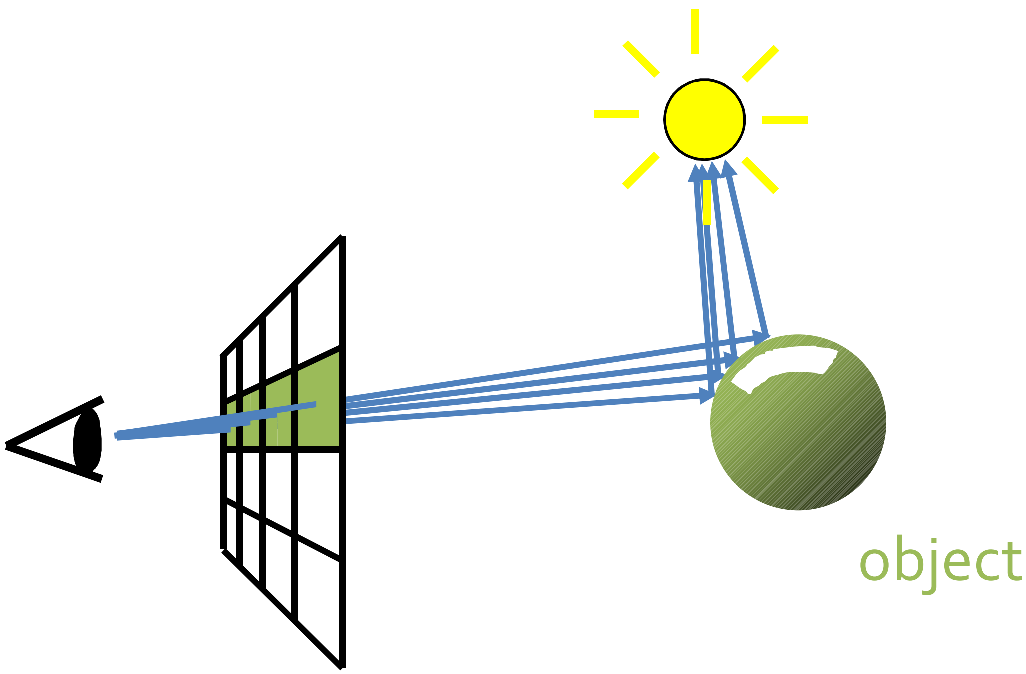
"Forward" Ray-Tracing

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



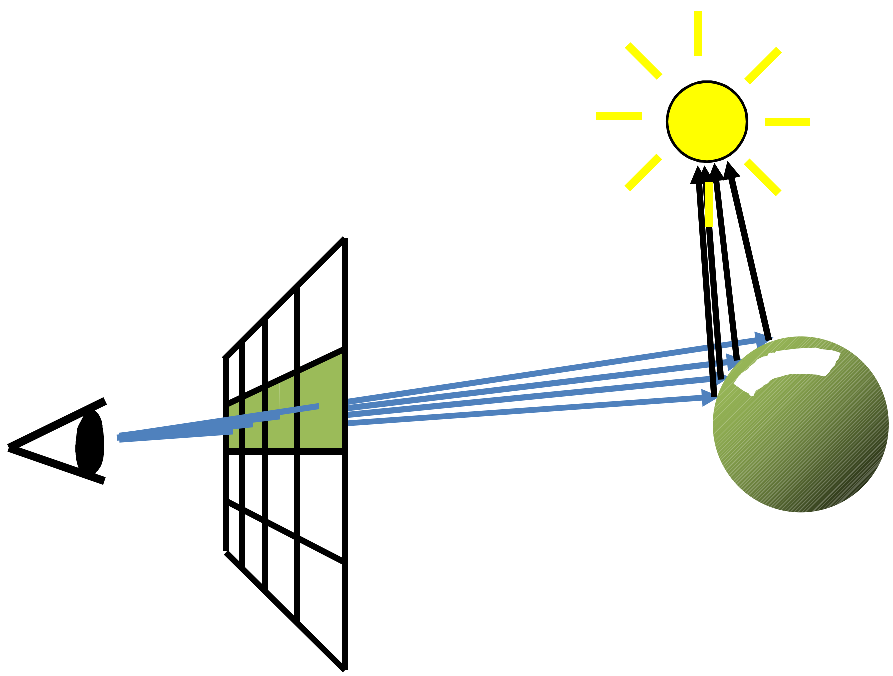
“Backward” Ray-Tracing

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



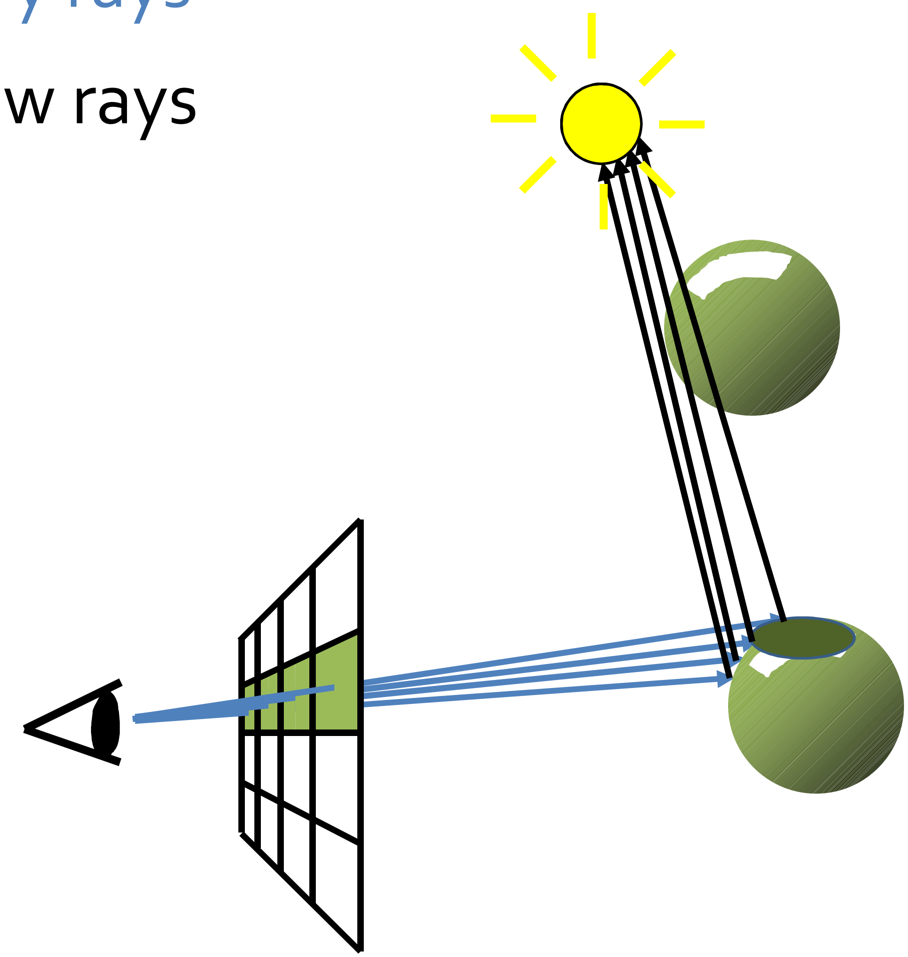
Types of Rays

- Primary rays
- Shadow rays



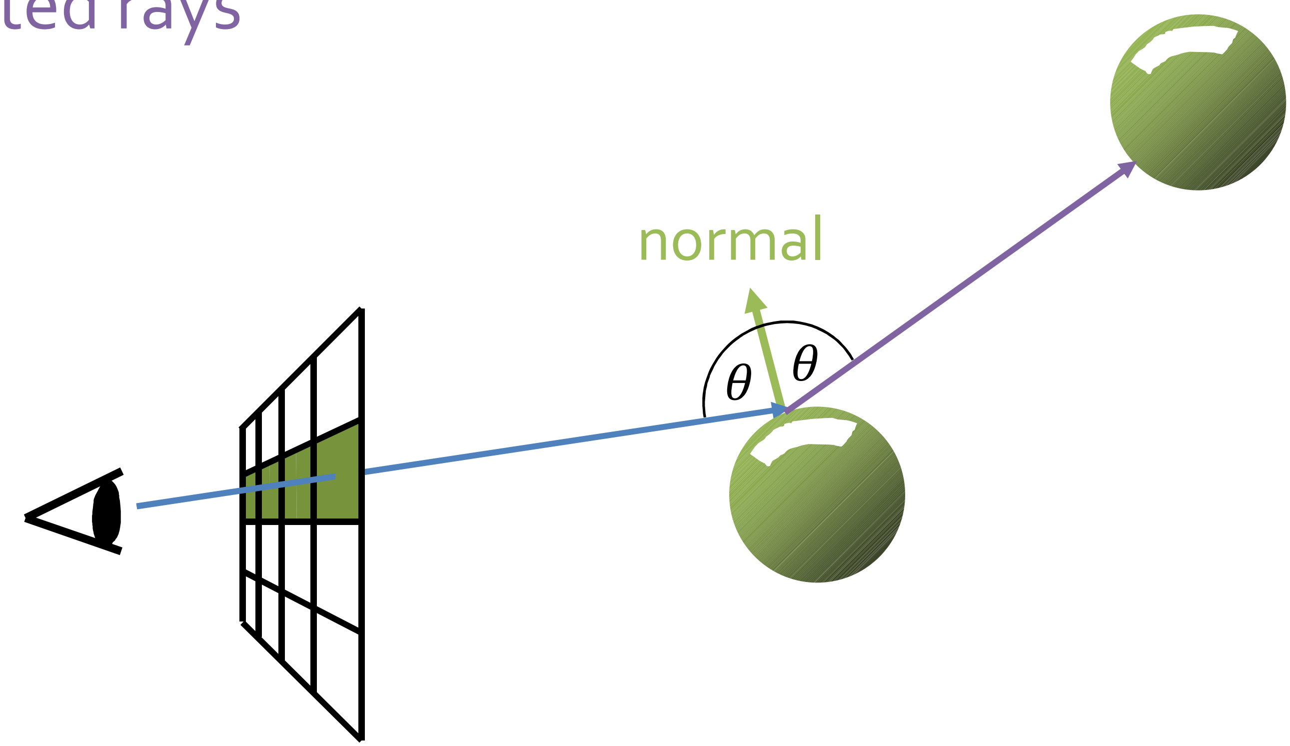
Shadow Rays

- Primary rays
- Shadow rays



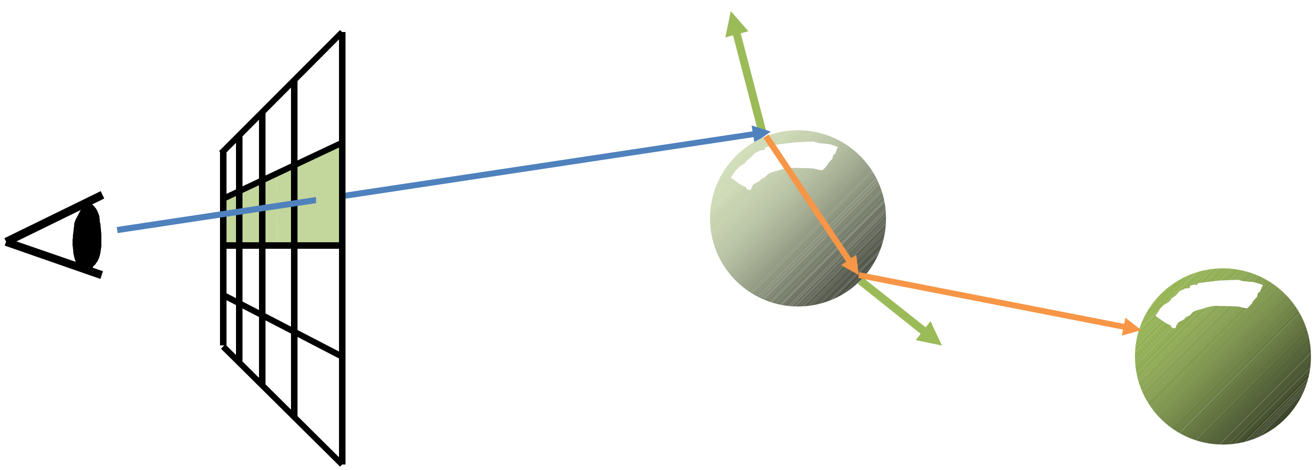
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays



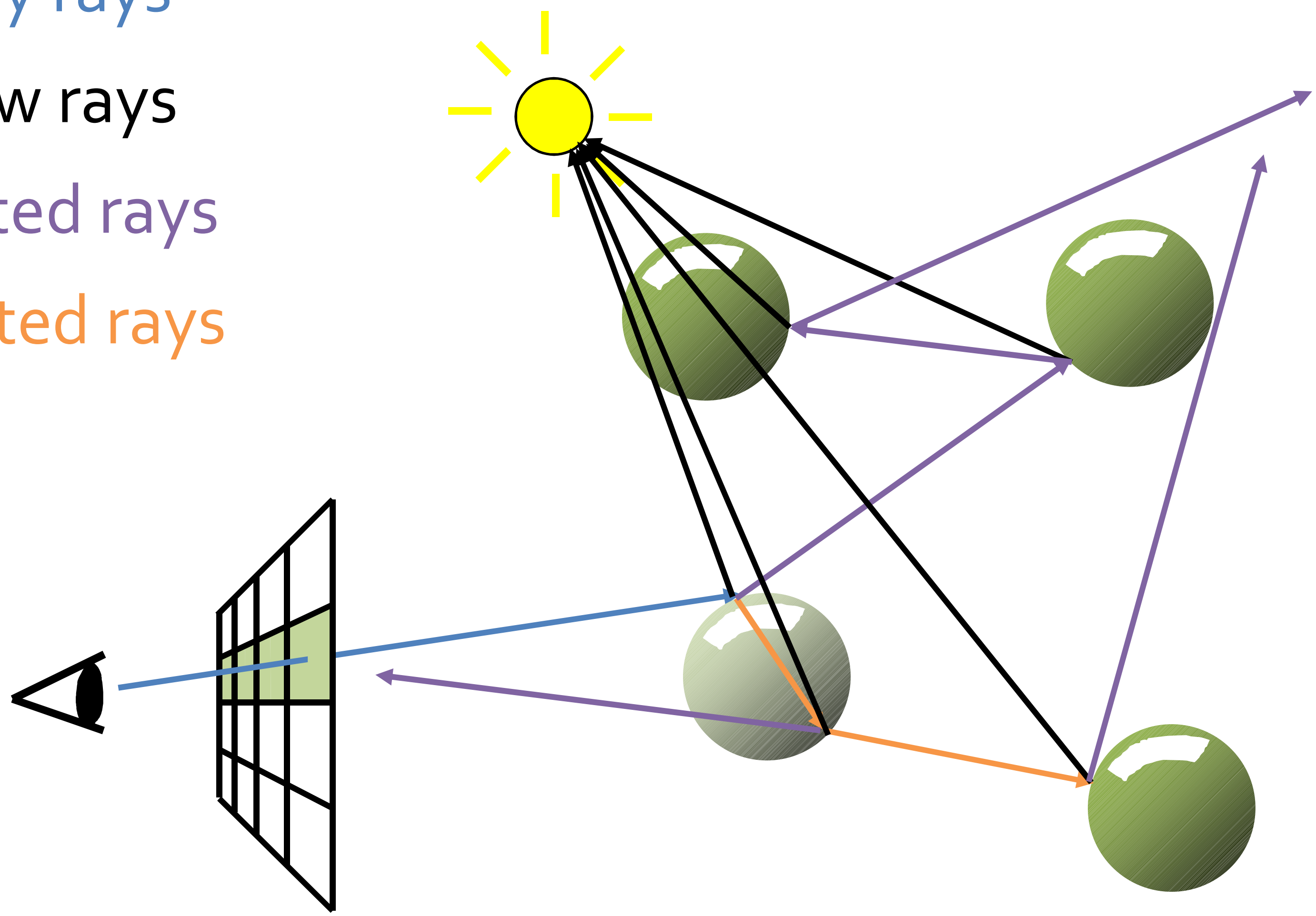
Types of Rays

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



Types of Rays

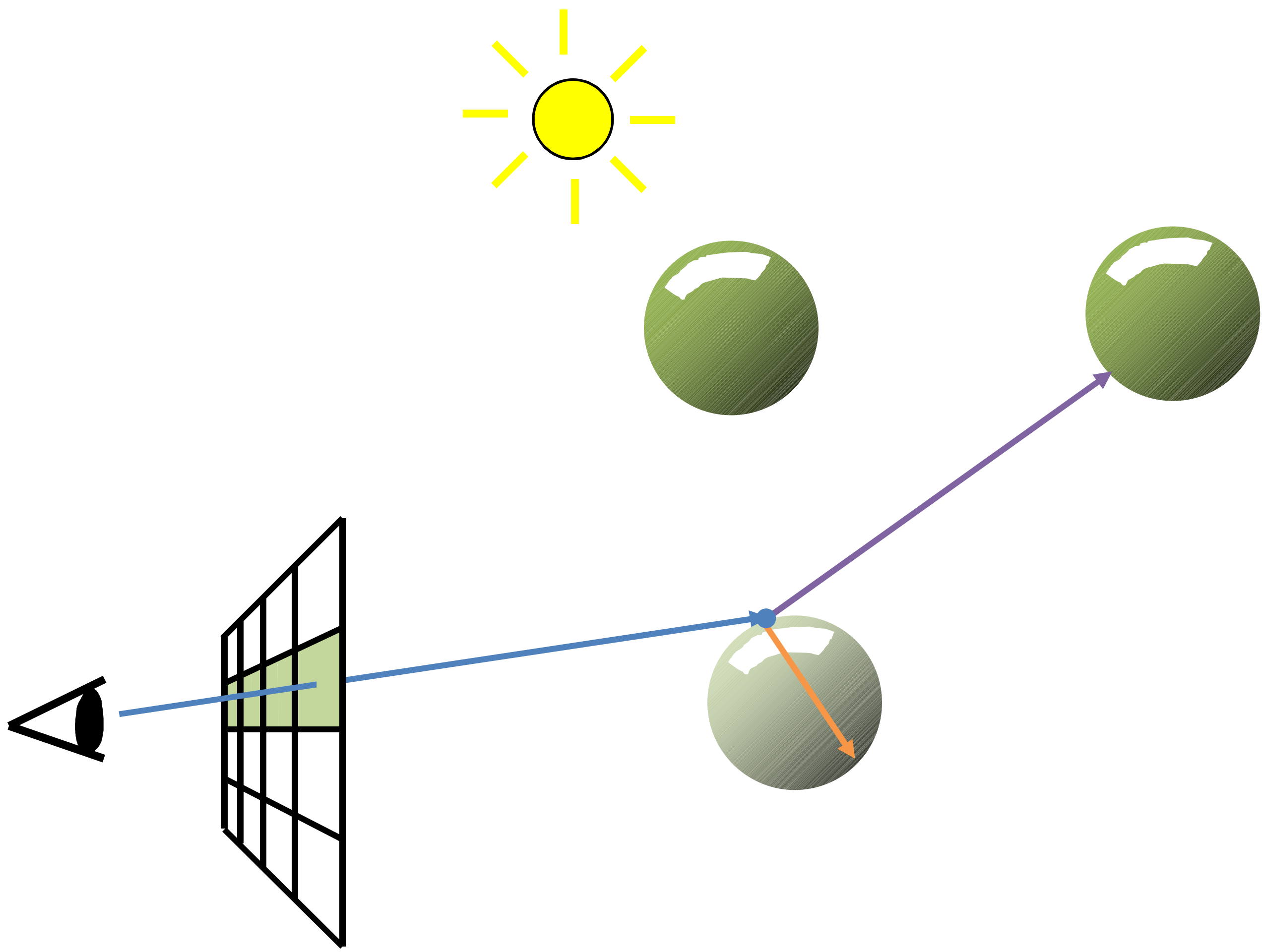
- Primary rays
- Shadow rays
- Reflected rays
- Refracted rays



Lighting

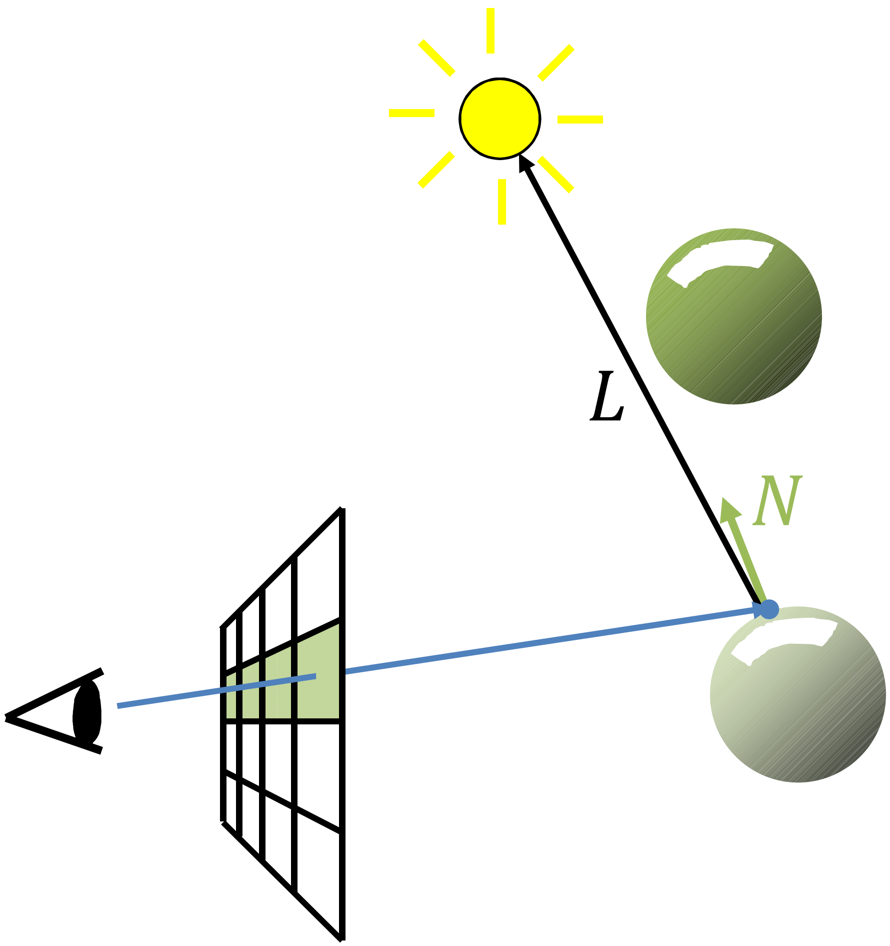
Lighting

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



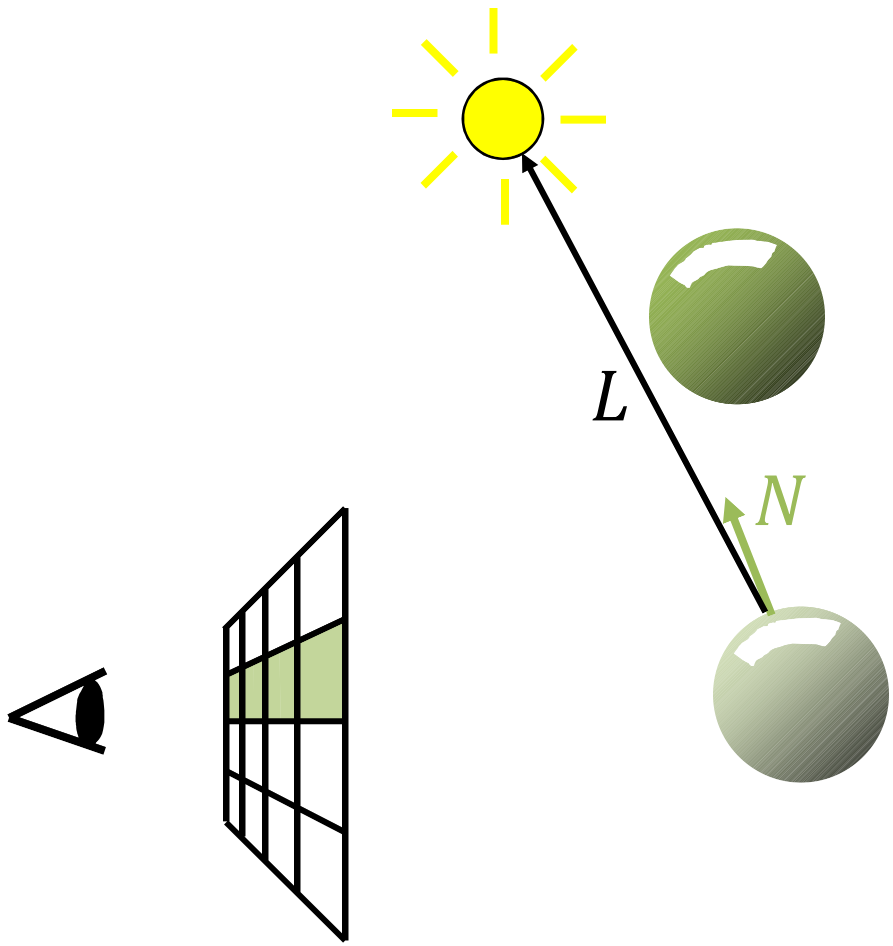
Local – Phong Illumination

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



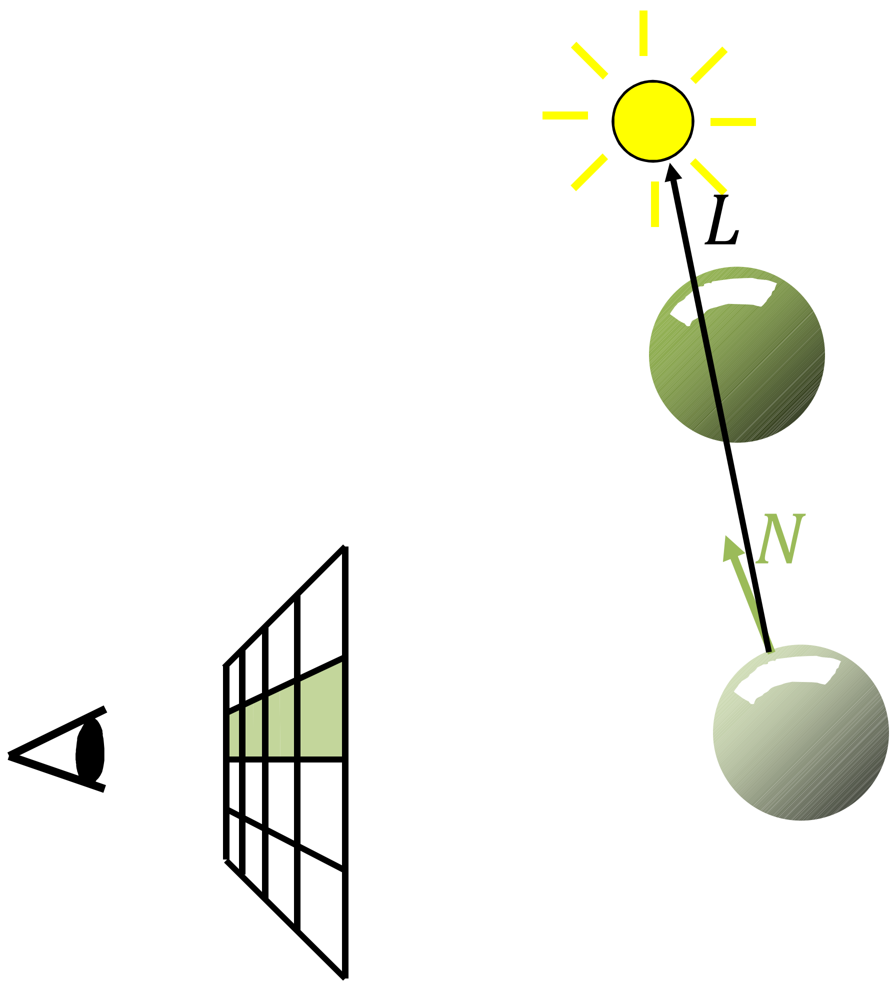
Diffuse (Lambert)

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



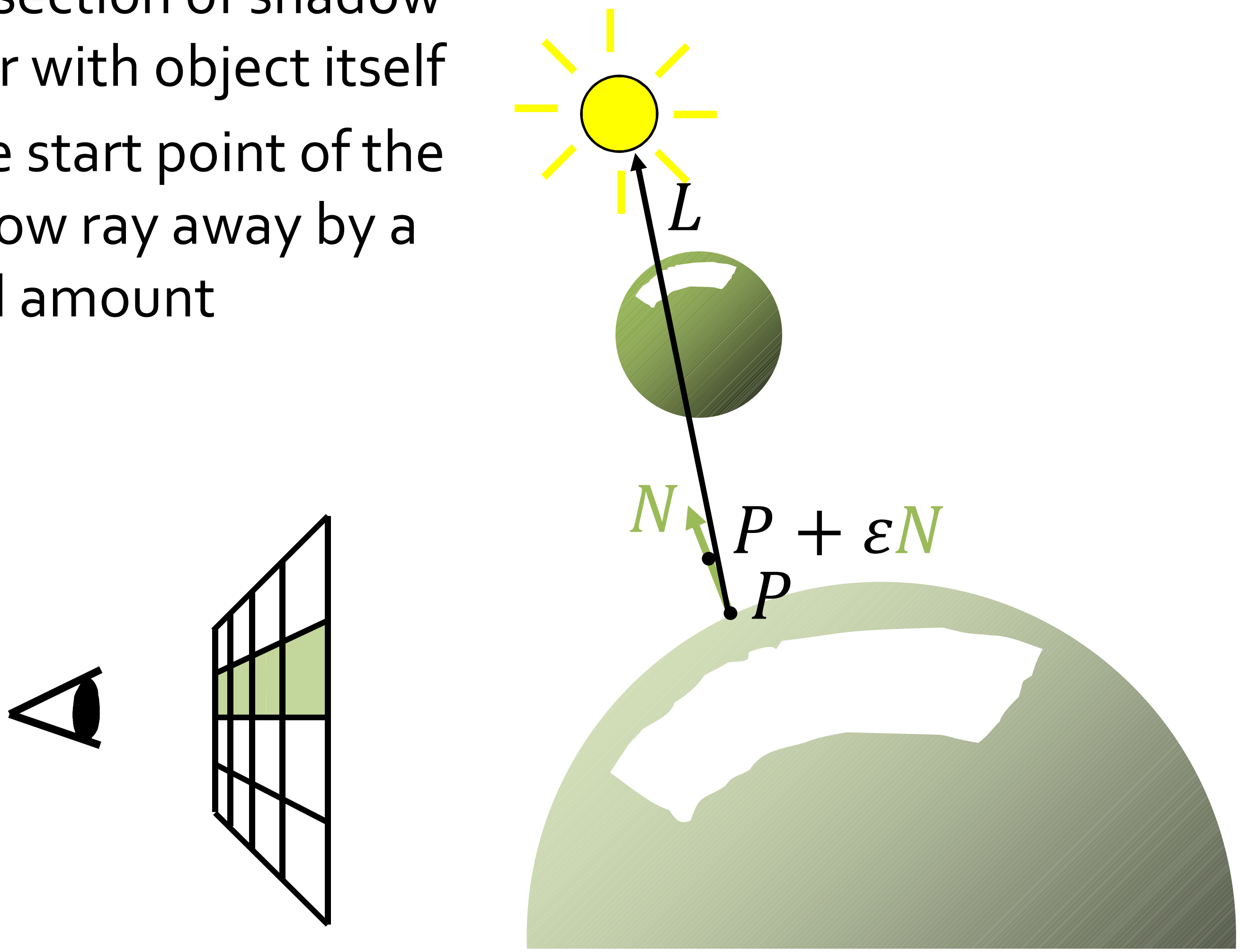
Adding Shadows

- Add local lighting only if point is seen by light



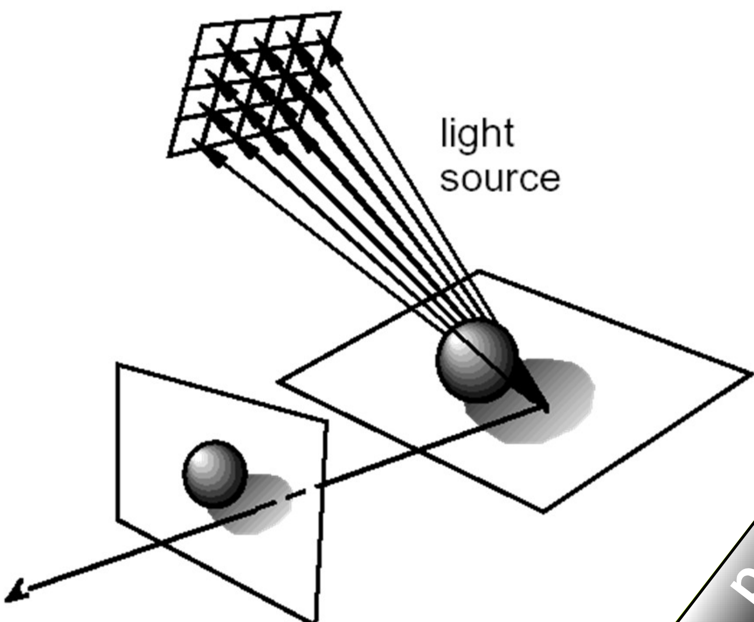
Adding Shadows

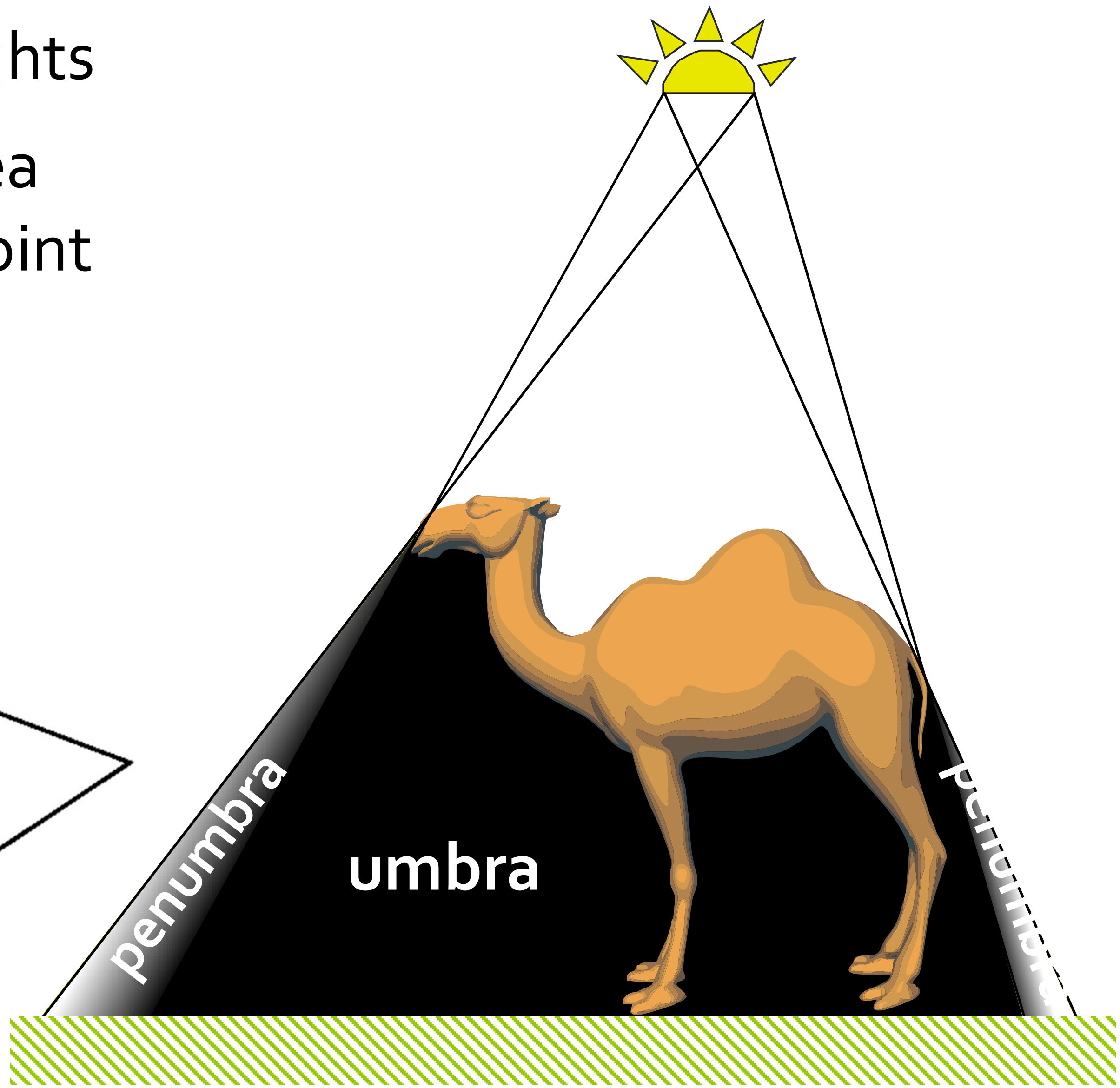
- "Self-Shadowing"
  - Intersection of shadow feeler with object itself
  - Move start point of the shadow ray away by a small amount



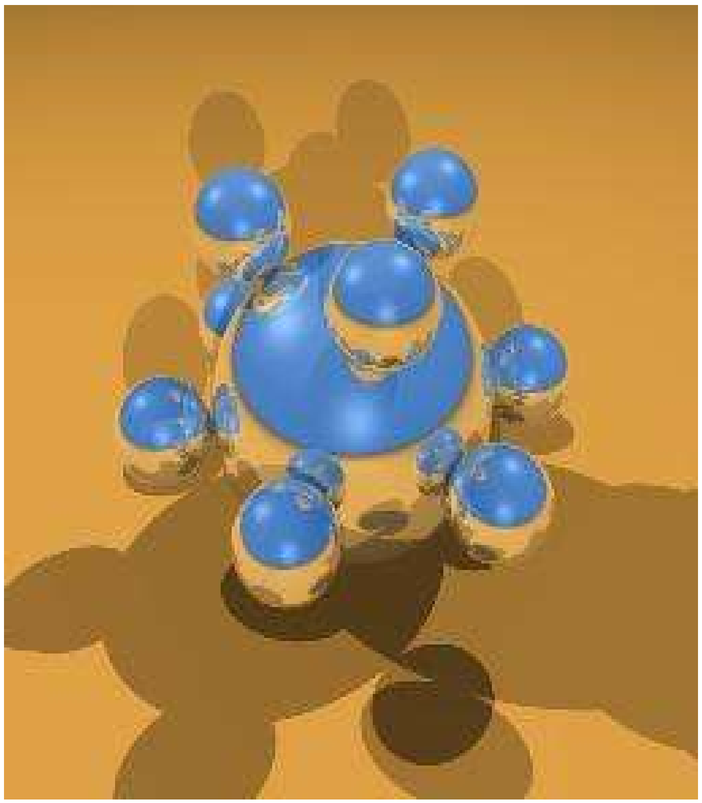
Soft Shadows

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

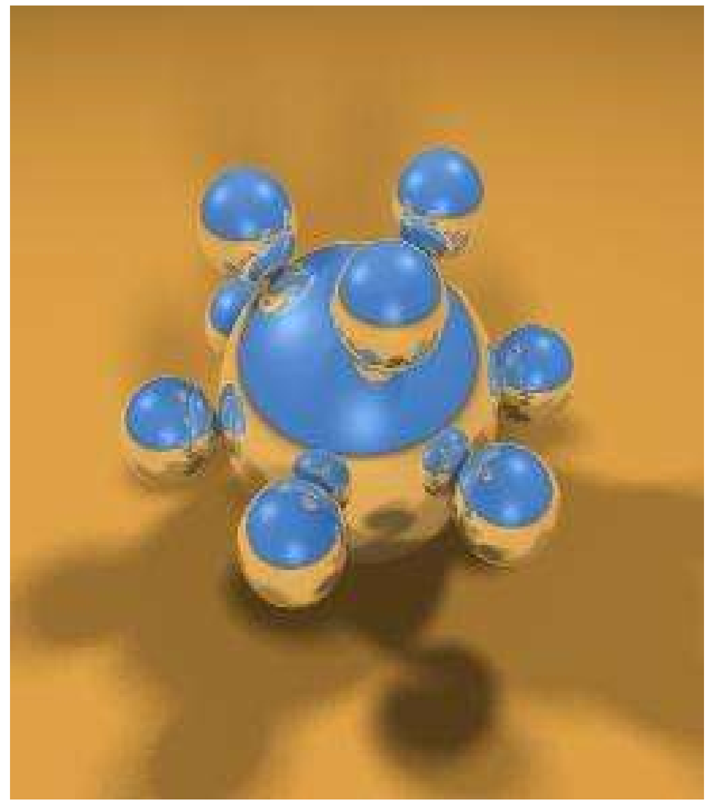




Soft Shadow Example



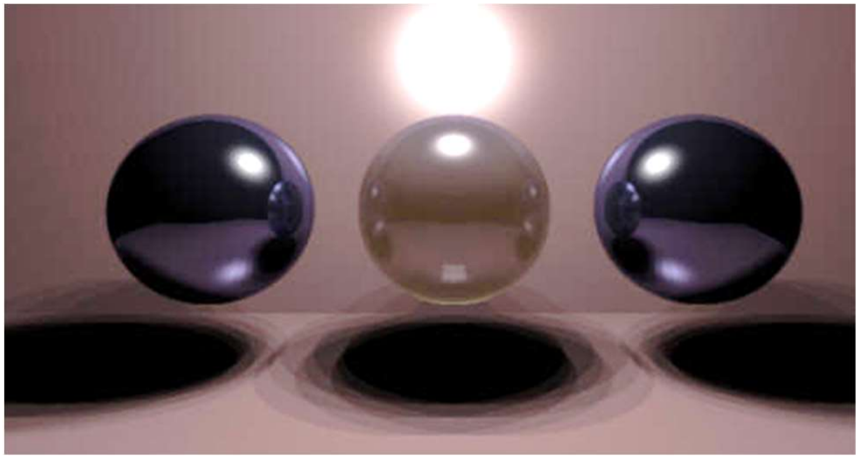
Hard shadow

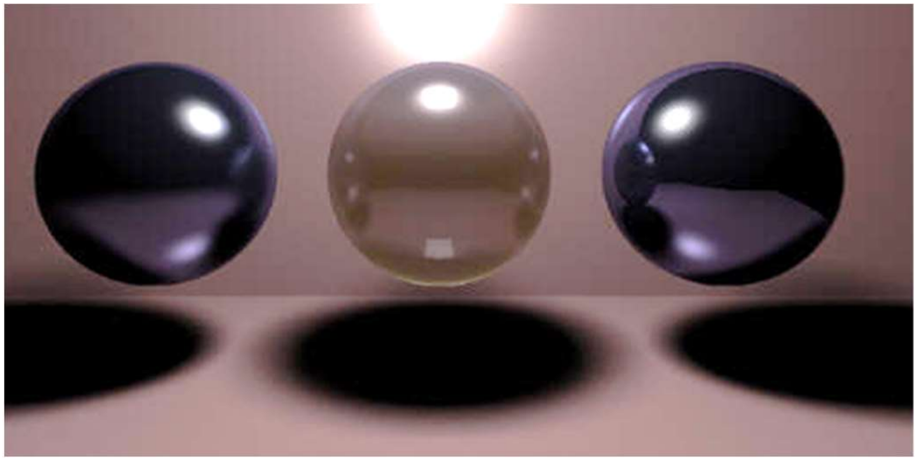


Soft shadow

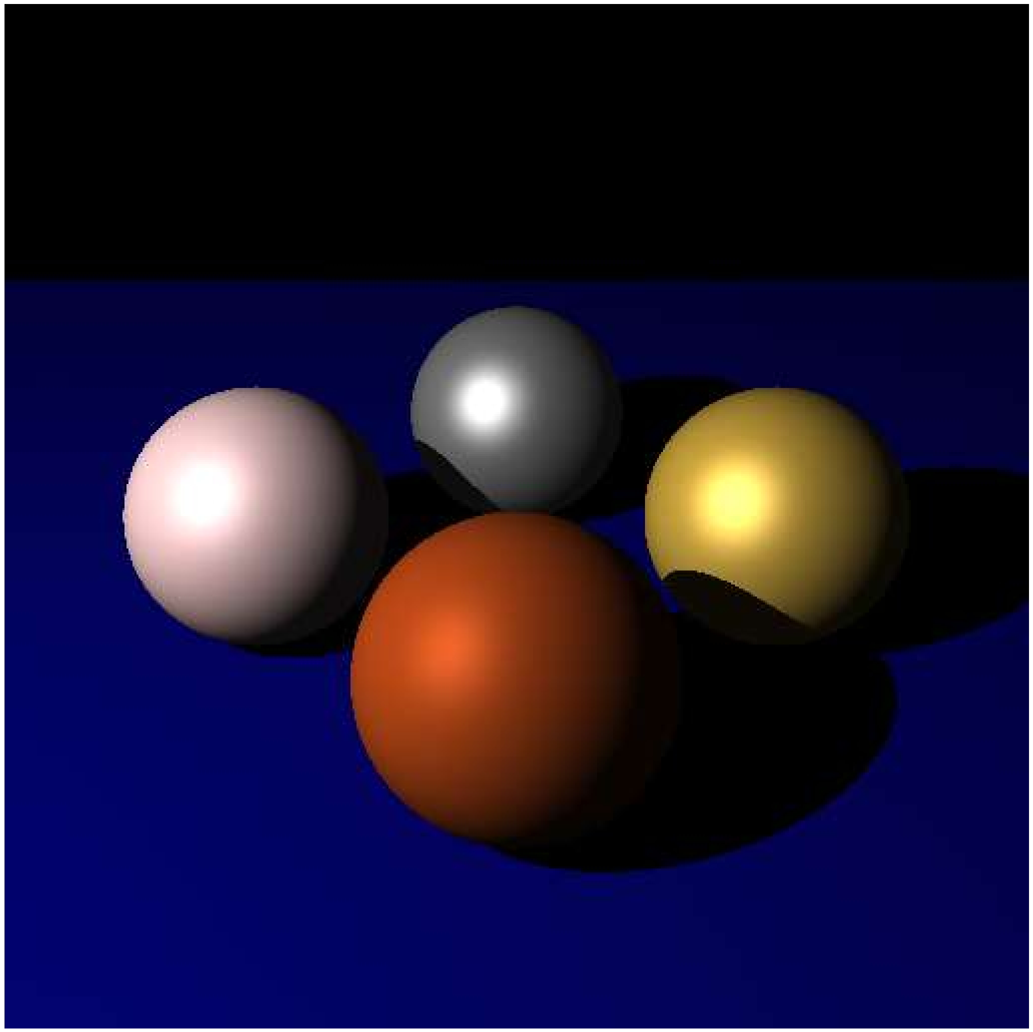
Area Light Sources

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



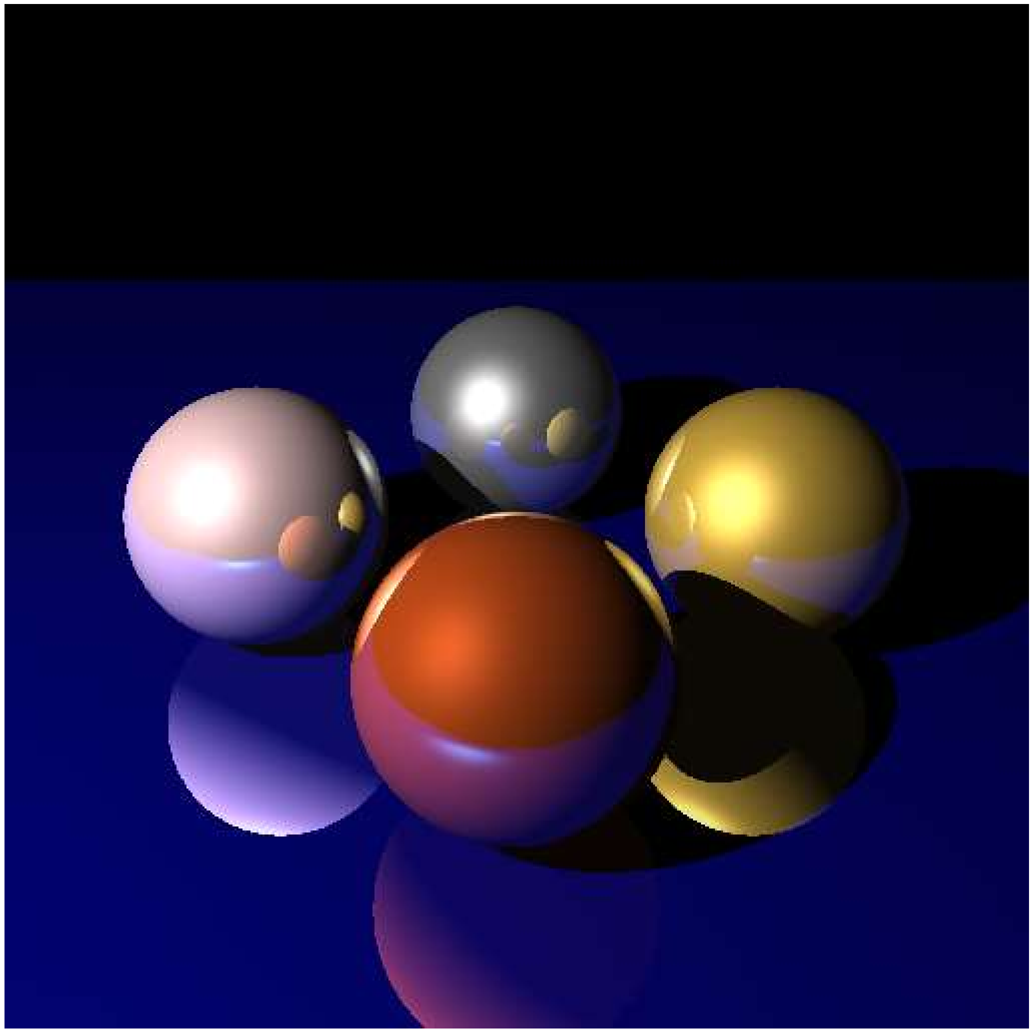


No Reflection



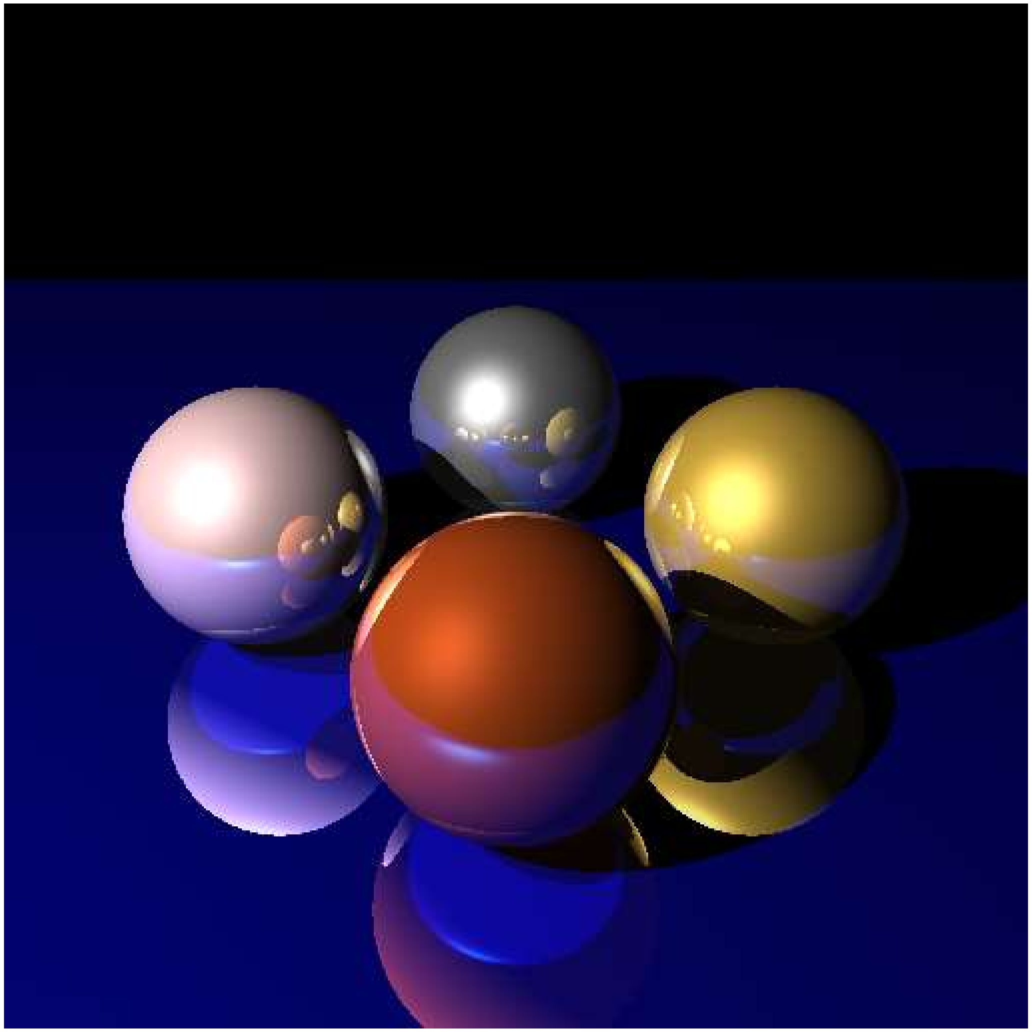
Created by David Derman – CISC 440

Reflection (1)



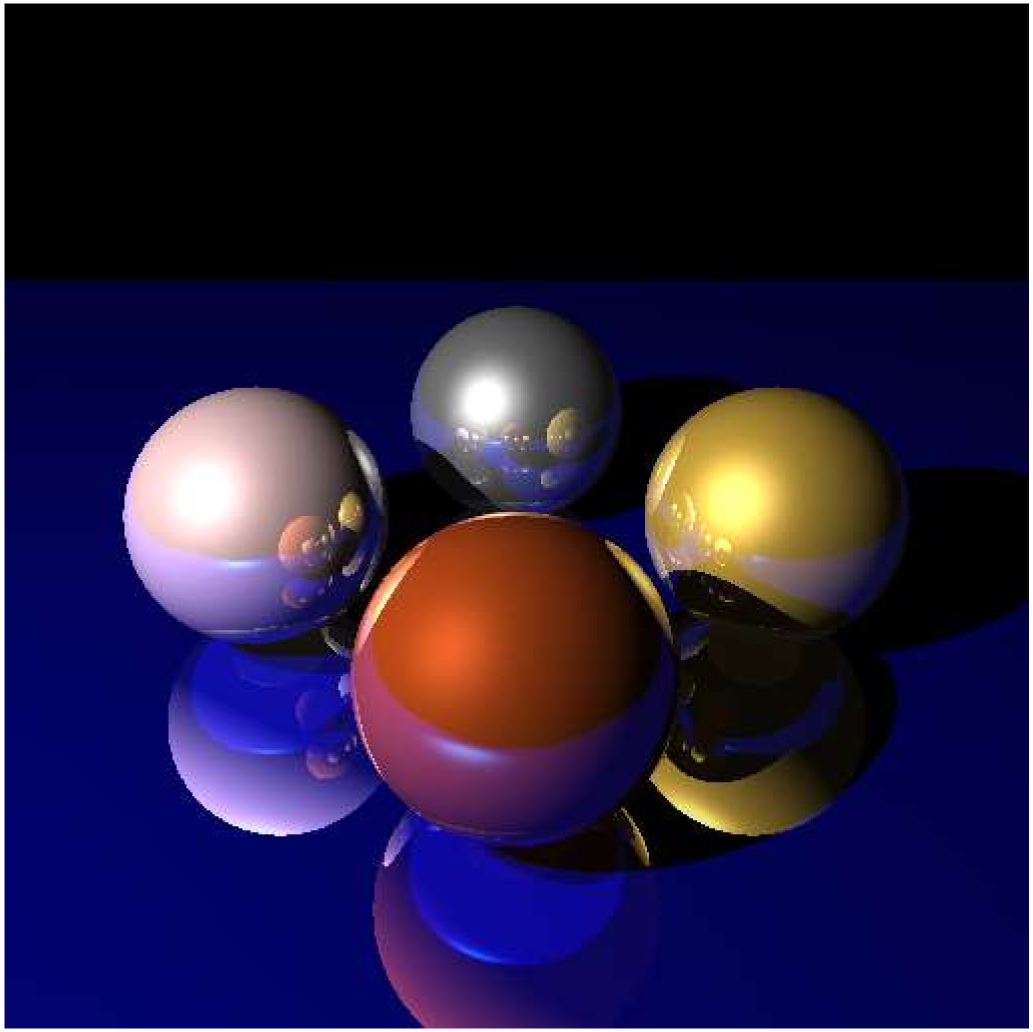
Created by David Derman – CISC 440

Reflection (2)



Created by David Derman – CISC 440

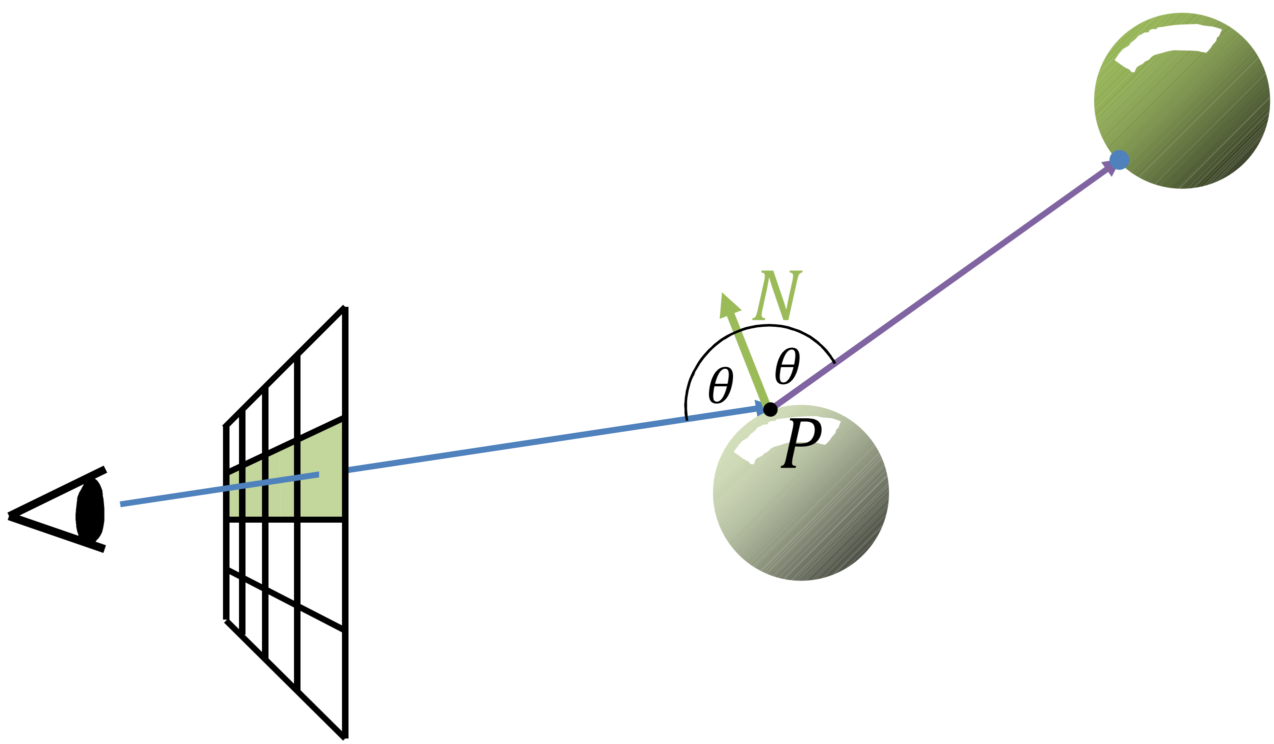
Reflection (3)



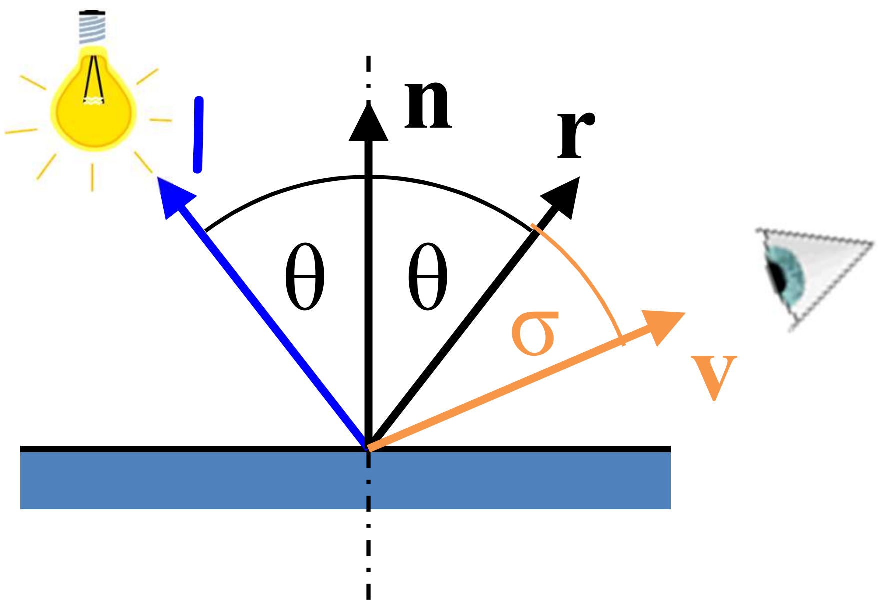
Created by David Derman – CISC 440

Reflection

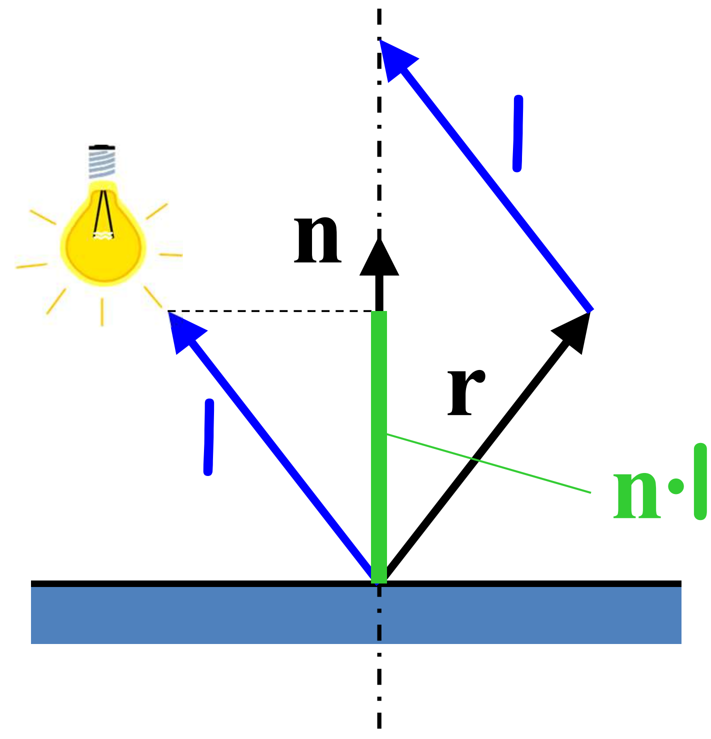
- $C_{reflected} = C_{\text{intersect}(\text{P},\text{reflect}(dir,N))}$



Reflection direction



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

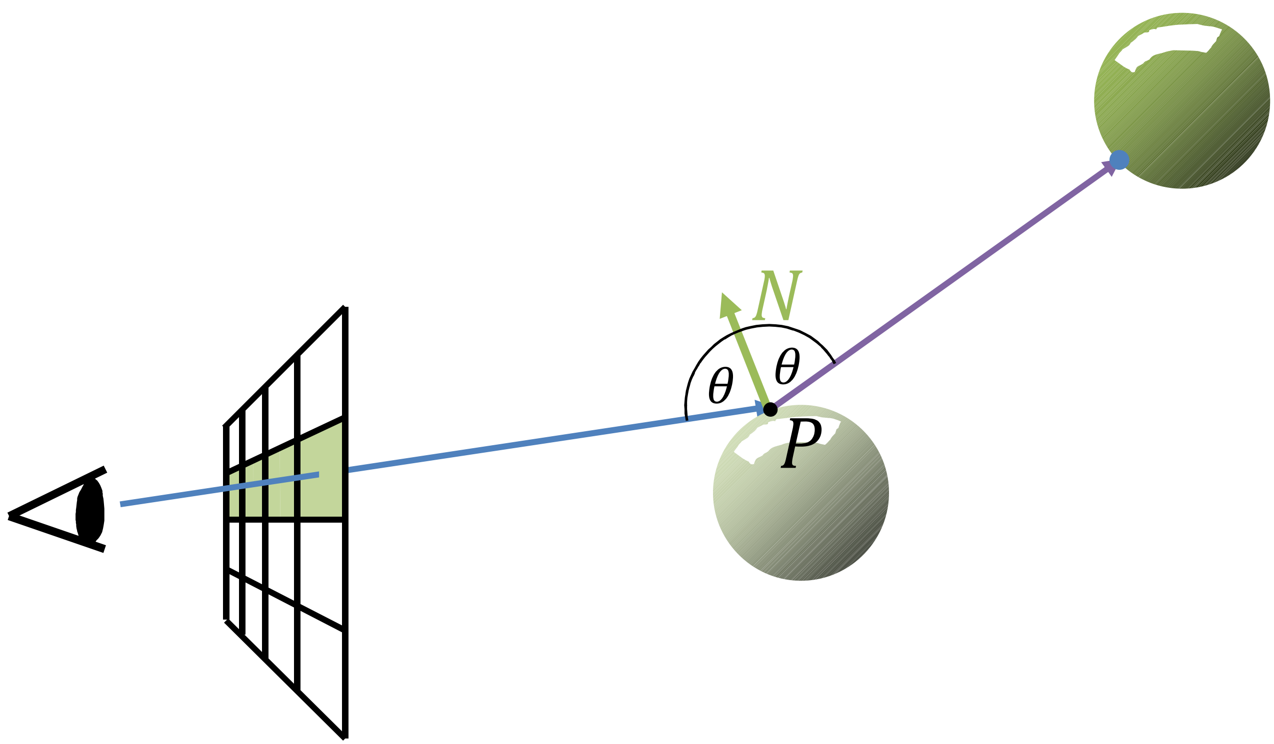


$$\mathbf{r} + l = (2\mathbf{n} \cdot l)\mathbf{n}$$

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

Reflection

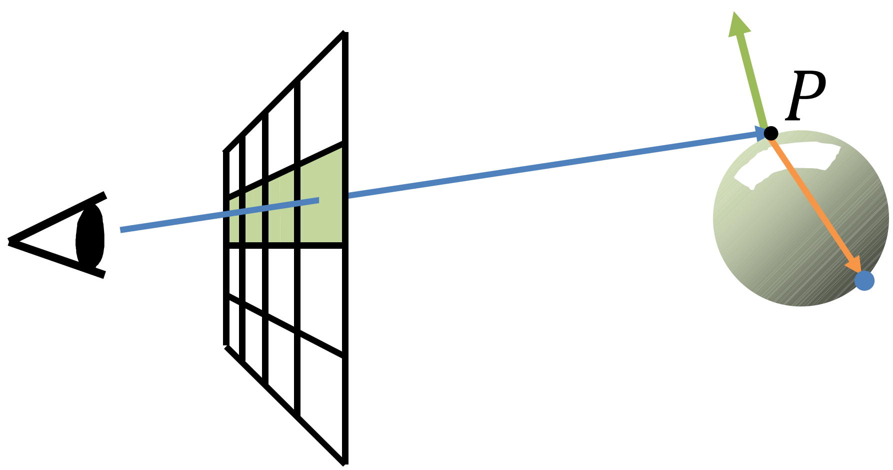
- $\text{reflect}(dir, N) = (2N \cdot -dir)N + dir$
- $C_{reflected} = C_{\text{intersect}(\text{P},\text{reflect}(dir,N))}$



Refraction

Refraction

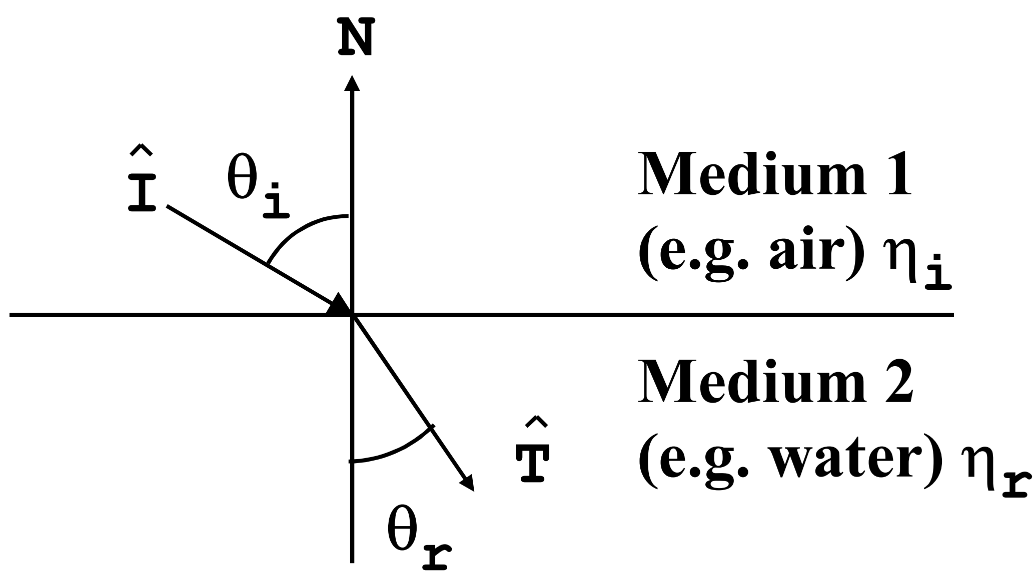
- $C_{refracted} = C_{\mathrm{intersect(P,refract(}dir,N\mathrm{))}}$



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**)

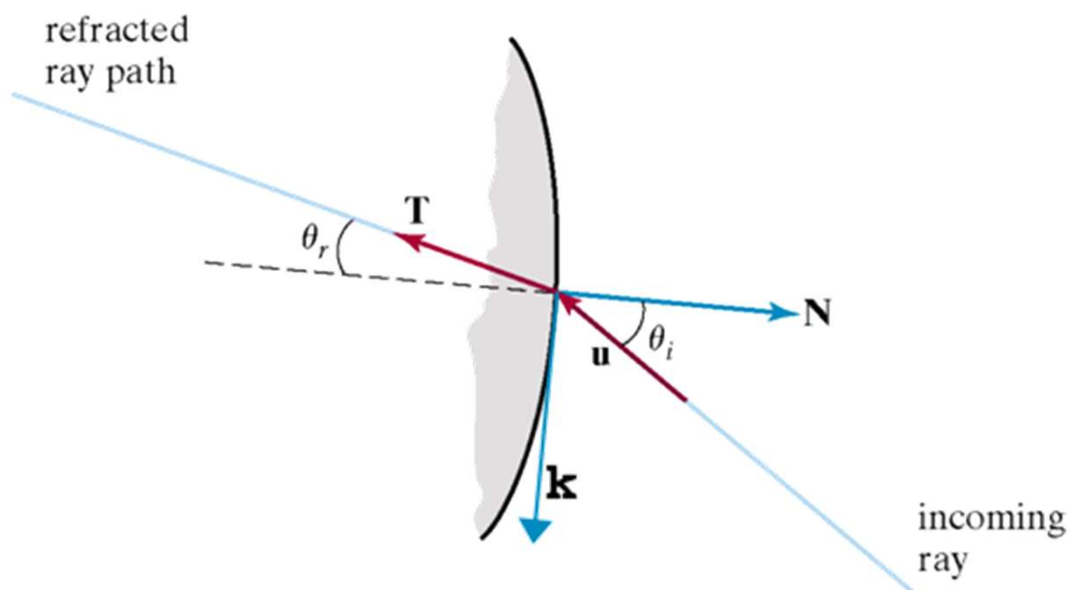
$$\begin{aligned}\mathbf{u} &= (\mathbf{u}\cdot\mathbf{n})(-\mathbf{n}) + (\mathbf{u}\cdot\mathbf{k})(-\mathbf{k})\\ &= -(\mathbf{u}\cdot\mathbf{k})\mathbf{k} - (\mathbf{u}\cdot\mathbf{n})\mathbf{n}\\ &= -(\sin\theta_i)\mathbf{k} - (\cos\theta_i)\mathbf{n}\end{aligned}$$

- 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**

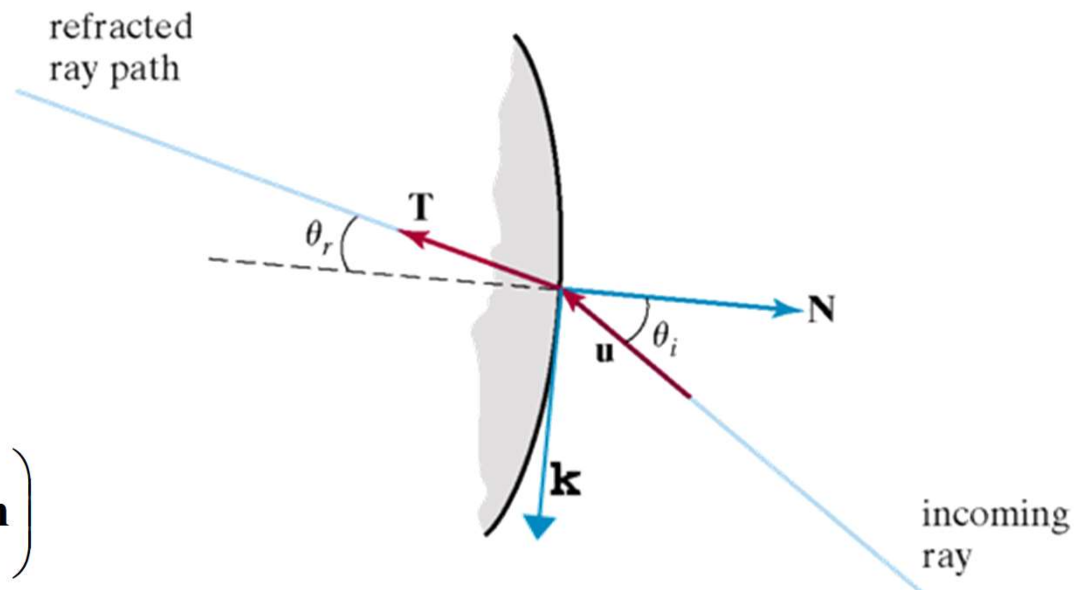
$$\mathbf{T} = -(\cos\theta_r)\mathbf{n} + \frac{\sin\theta_r}{\sin\theta_i}(\mathbf{u} + (\cos\theta_i)\mathbf{n})$$

- From Snell's Law

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

- Solving for **T**

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



Acceleration structures for ray-tracing