

A scene from the movie "The LEGO Movie" featuring Emmet, a yellow LEGO minifigure with a determined expression, shouting with his mouth wide open. He is wearing a blue vest over a white shirt with a red cross on it. He is surrounded by other LEGO figures, some of whom are holding large, glowing blue energy swords. The background is a vibrant, futuristic city at night, filled with colorful neon signs and billboards. One sign clearly reads "TOLL".

# Ray-Tracing

The LEGO Movie - Image Courtesy of Warner Bros. Pictures  
Animal Logic 'Glimpse' Ray-Trace Rendering Engine



# 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)

**Paul Heckbert**

Dessert Foods Division  
Pixar

PO Box 13719  
San Rafael CA, 94913

**415-499-3600**

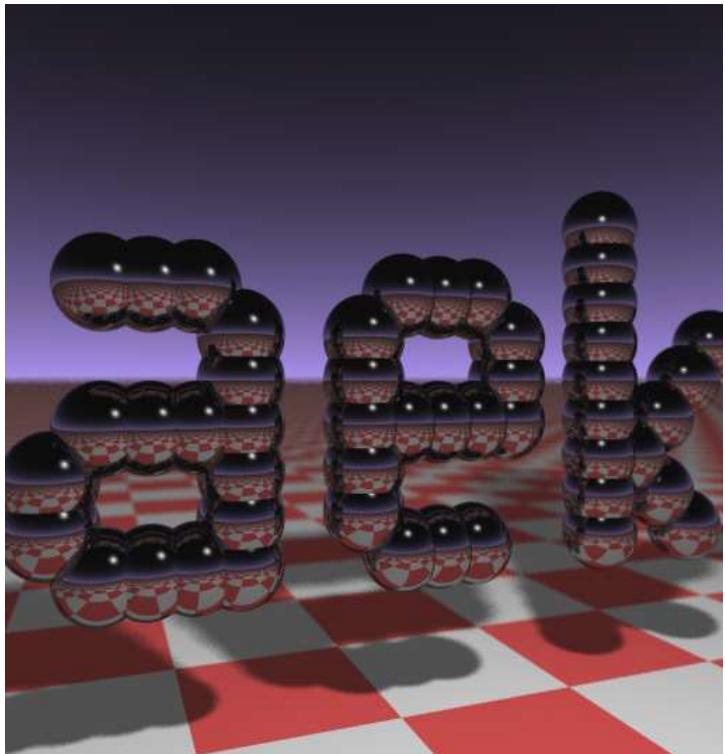
network address: ucbvax!pixar!ph

```

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,.25,0.,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.,25.,1.,.8,1.,7.,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
->ad,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=vcc~b(-1.,N,black),eta=1/eta,d=-d;l=sph+5;while(l-->sph)if((e=1
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==1)color=vcomb(e
,1->color,color);J=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(25/114.591550261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*pixar!ph*/

```

# Analysis of the Business Card Ray-Tracer



[fabiensanglard.net/rayTracing/back\\_of\\_business\\_card](http://fabiensanglard.net/rayTracing/back_of_business_card)

Vector, World, Tracer, Sampler,  
Main

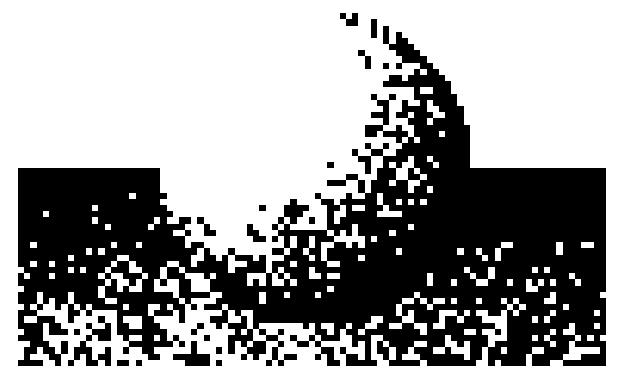
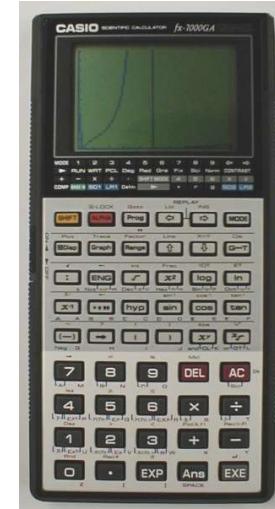
```
#include <stdlib.h> #include <stdio.h>
#include <math.h> typedef int i; typedef float f;
struct v{ f x,y,z;v operator+(v r){return v(x+r.x ,y+r.y,z+r.z);}
v operator*(f r){return v(x*r,y*r,z*r);}f operator%(v r){return
x*r.x+y*r.y+z*r.z;}v(){}v operator^(v r ){return v(y*r.z-z*r.y,
z*r.x-x*r.z,x*r. 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,1,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

# Intersectable = renderable



Turner  
Whitted



William  
Hollingworth



Henrik Wann  
Jensen



Ken  
Musgrave

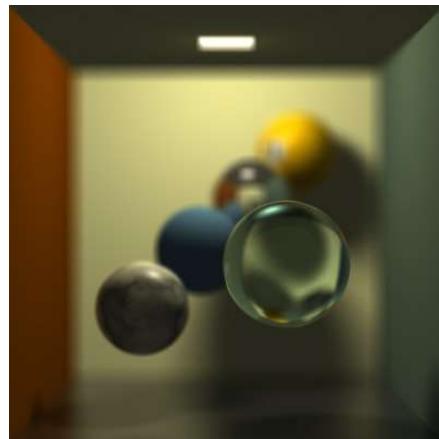
# Reflections, Refractions

Gilles Tran, wikimedia

# Stochastic Effects



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



Matt Roberts

Jason Waltman

# Ray-Tracing – History

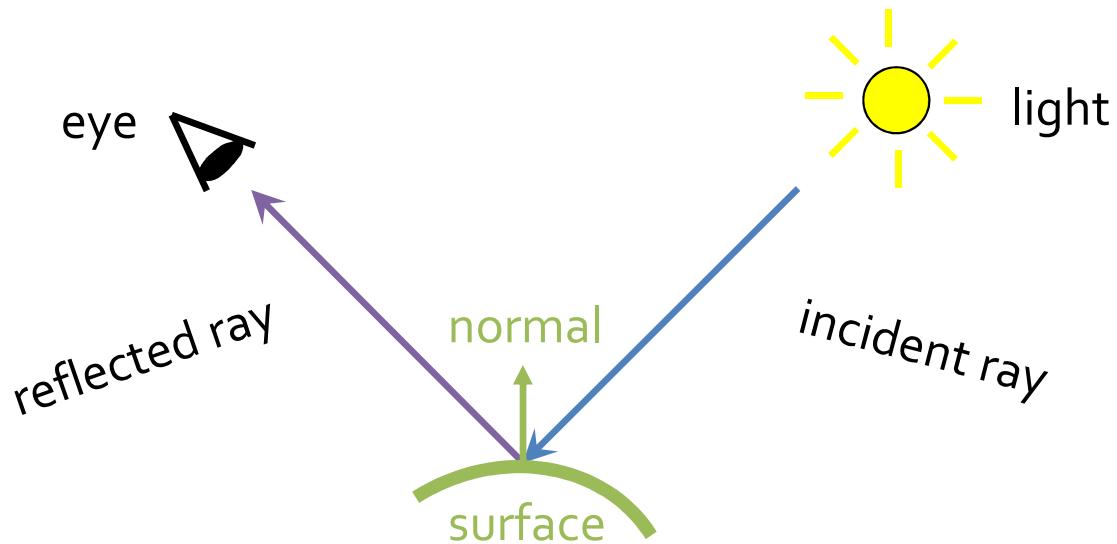
- Decartes, 1637 A.D. – analysis of rainbow
- Arthur Appel, 1968 – used for lighting 3D models
  - A.k.a. ray casting
- Turner Whitted, 1980 – “An Improved Illumination Model for Shaded Display”
  - Recursion
  - Start into main stream
- 1980-now – Lots of research
  - Speed
  - Realisme

# Ray-Tracing – Basics

- Idea
- Generating camera rays
- Ray-object intersections
- Direct Lighting (shadowing)
- Recursion (reflection / refraction)

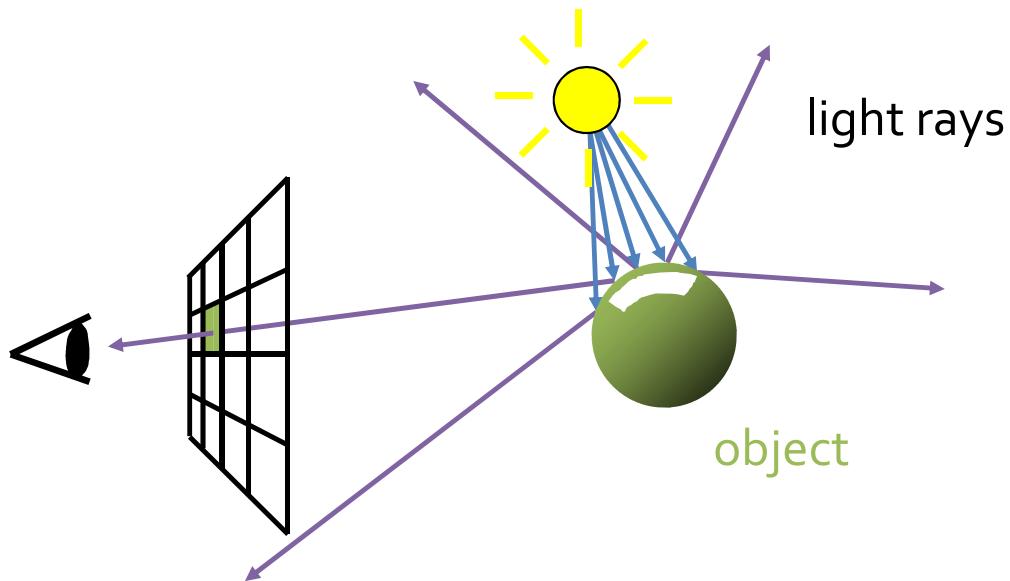
# Idea

- Simulate light rays from light source to eye



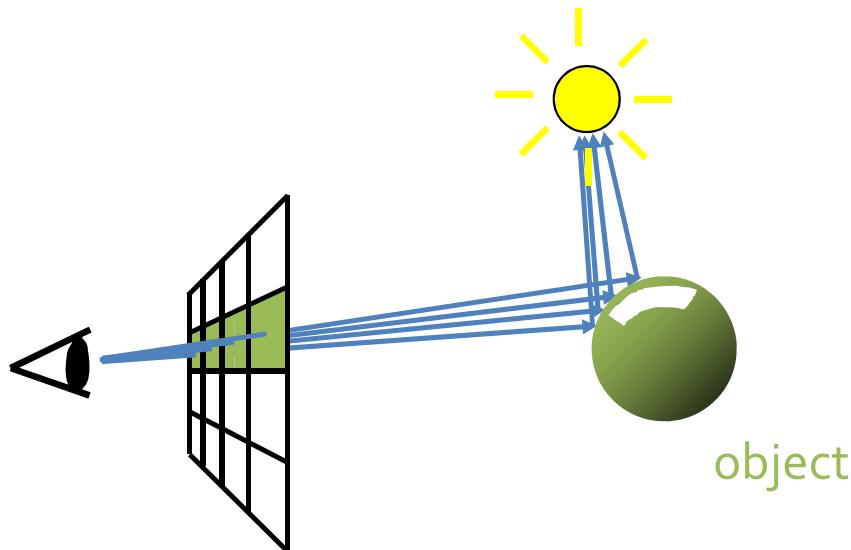
# “Forward” Ray-Tracing

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



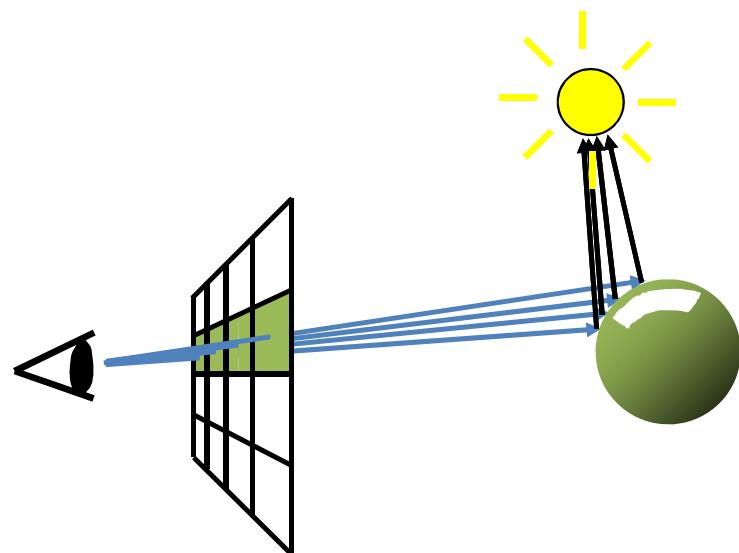
# “Backward” Ray-Tracing

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



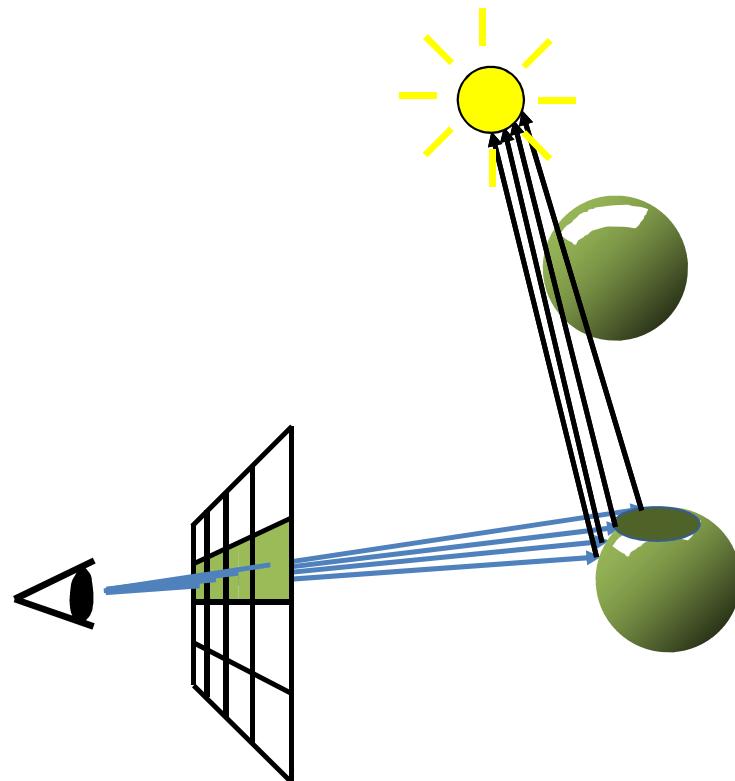
# Types of Rays

- Primary rays
- Shadow rays



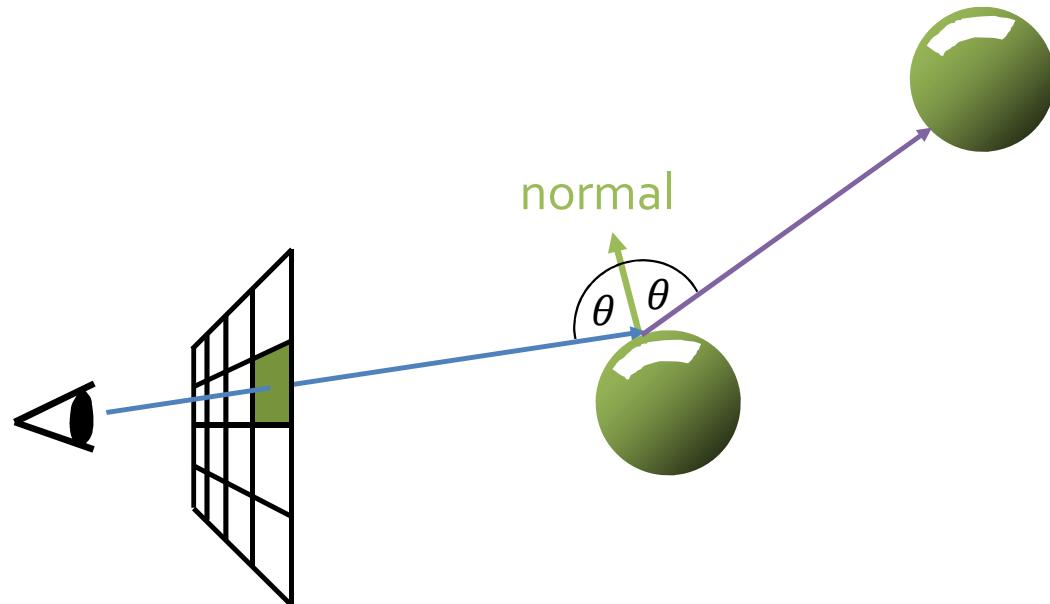
# Types of Rays

- Primary rays
- Shadow rays



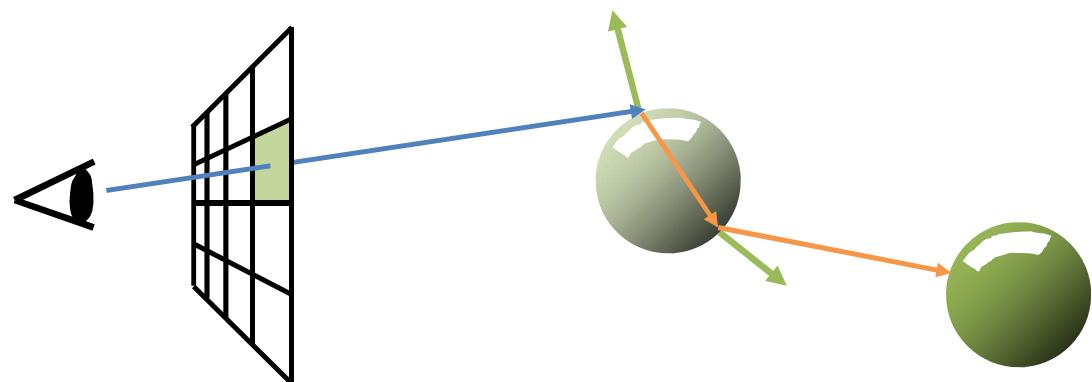
# Types of Rays

- Primary rays
- Shadow rays
- Reflected rays



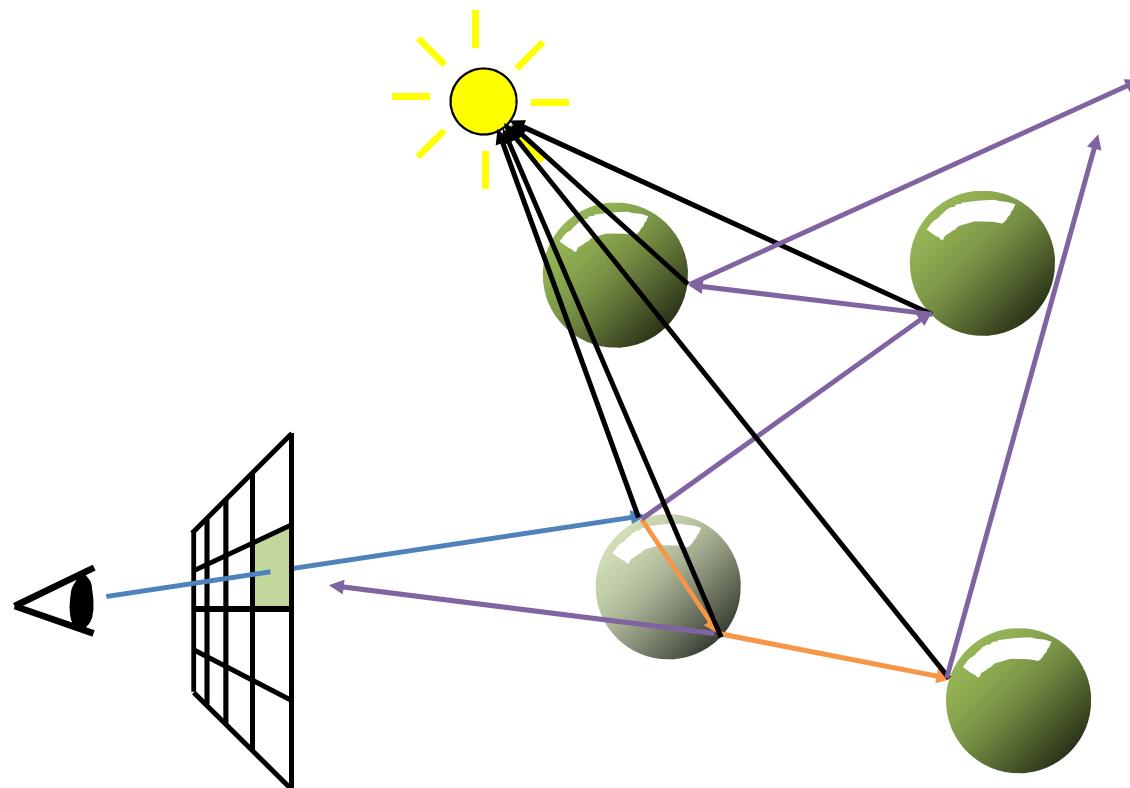
# Types of Rays

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



# Tree of Rays

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



# Ray-Tracer Code

```
renderImage() {  
    foreach pixel x,y in image {  
        ray = createCameraRay(x,y)  
        image[x][y] = trace(ray)  
    } }  
  
color trace(ray) {  
    objectHit = findNearestObjectHit(ray)  
    if(objectHit == background) return bckGrndColor  
    if(obectjHit == light) return lightColor  
    color = directLighting(ray, objectHit)  
    color += trace(reflect(ray, objectHit))  
    color += trace(refract(ray, objectHit))  
    return color  
}
```

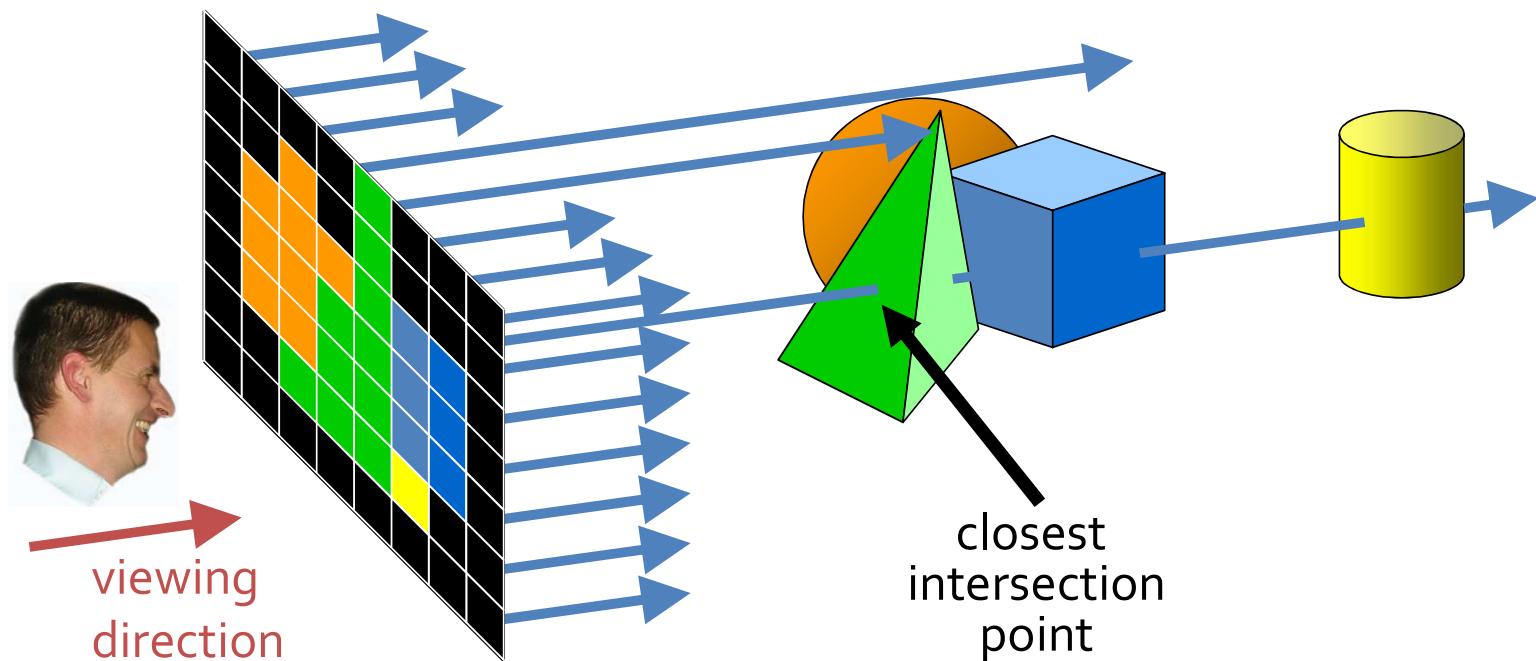
# Ray-Tracer Code

```
renderImage() {  
    foreach pixel x,y in image {  
        ray = createCameraRay(x,y)  
        image[x][y] = trace(ray)  
    } }  
  
color trace(ray) {  
    objectHit = findNearestObjectHit(ray)  
    if(objectHit == background) return bckGrndColor  
    if(obectjHit == light) return lightColor  
    color = directLighting(ray, objectHit)  
    color += trace(reflect(ray, objectHit))  
    color += trace(refract(ray, objectHit))  
    return color  
}
```

# **Generating Camera Rays**

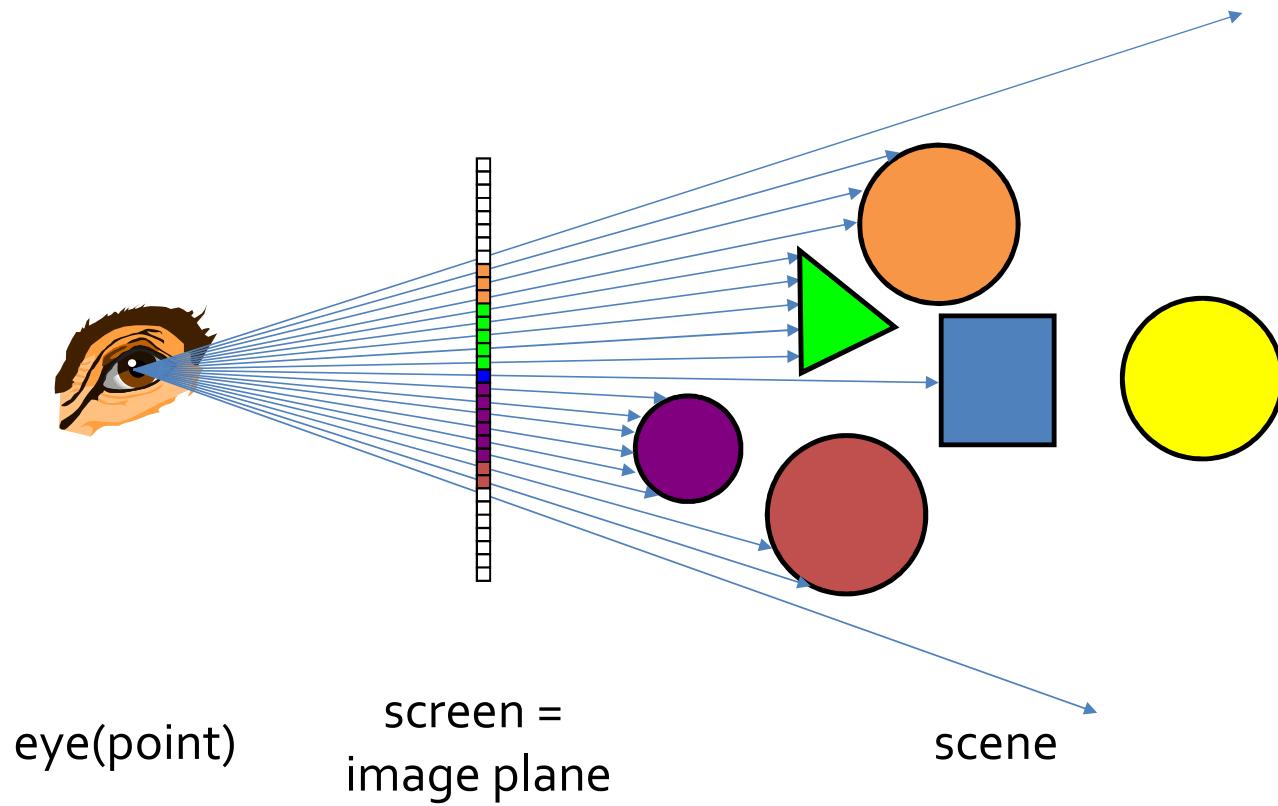
# Generating Camera Rays

- Cast ray from each pixel and intersect with all surfaces
- Calculate color from closest intersected surface
- How Many ray-object intersections?



# Generating Camera Rays

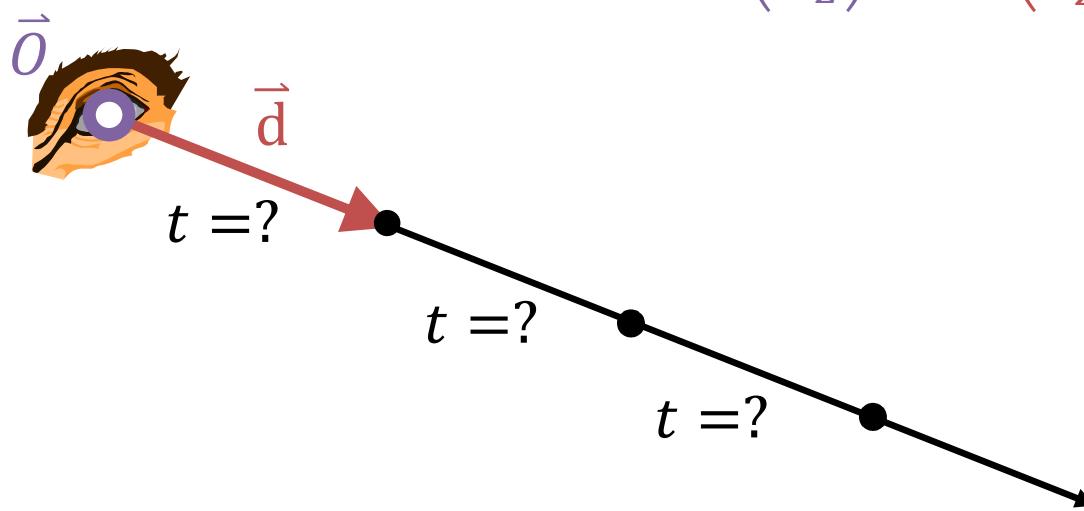
- Trace a ray for each pixel in the image plane



# Ray Parametric Form

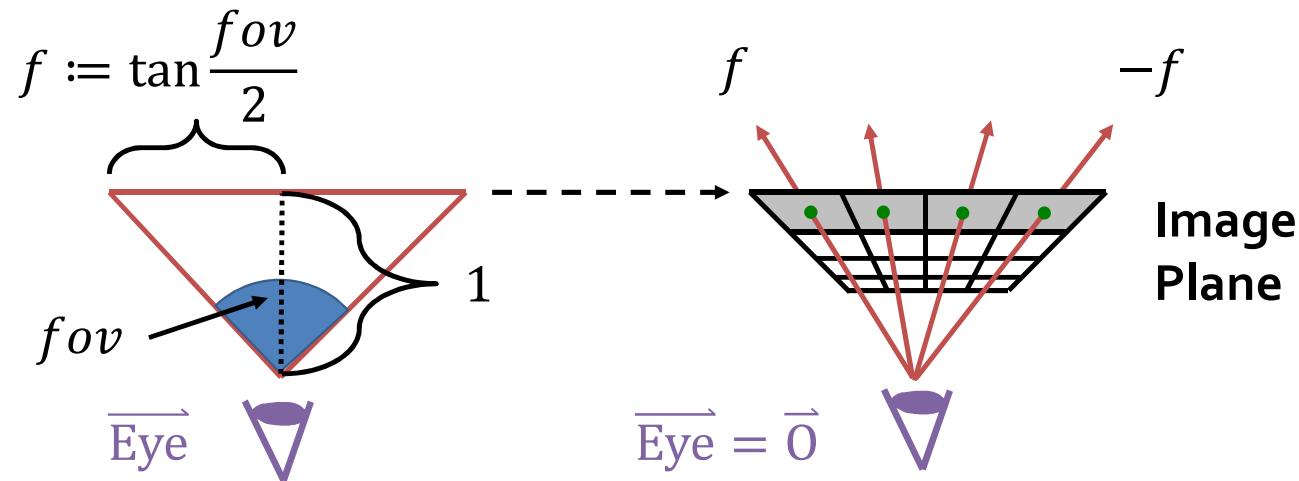
- Ray expressed as function of a single parameter  $t$

$$ray(t) = \vec{o} + t\vec{d} = \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} + t \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$$



# Generating Camera Rays – Top View

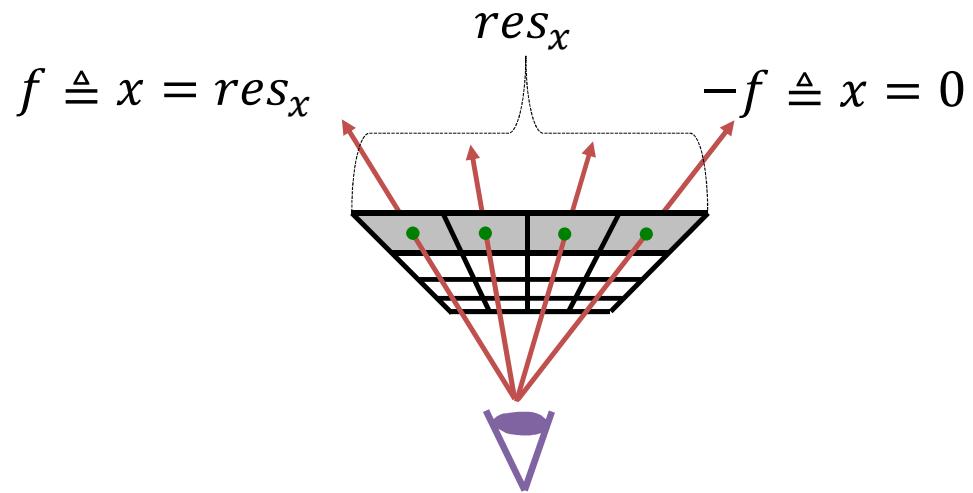
- Trace a ray for each pixel in the image plane



# Generating Camera Rays – Top View

- Trace a ray for each pixel in the image plane

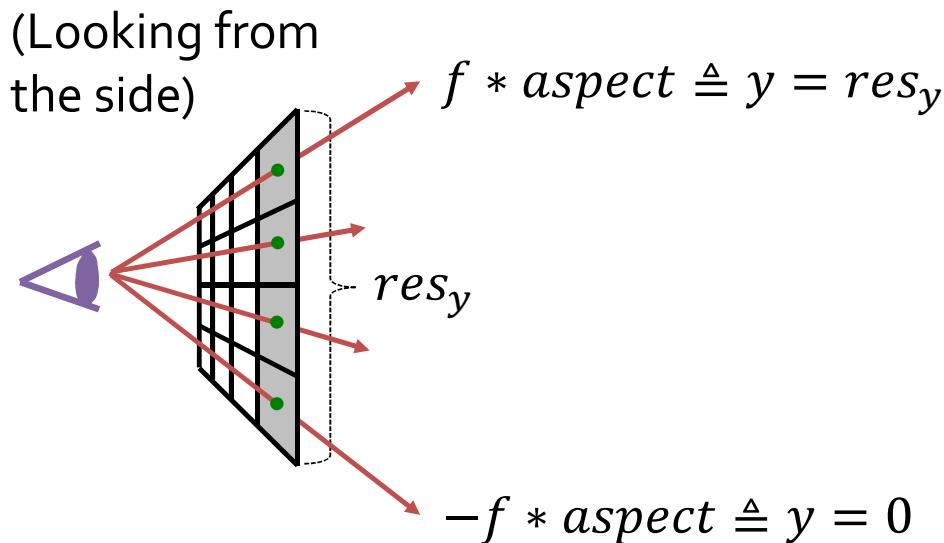
- $d_x(x) = \frac{x^2 f}{res_x} - f = (2x - res_x) \frac{f}{res_x} = (2x - res_x) f_x \quad f_x := \frac{f}{res_x}$



(Looking down from the top)

# Generating Camera Rays – Side View

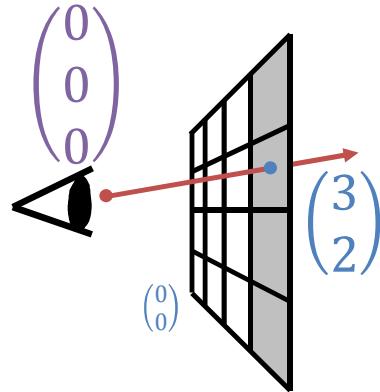
- Trace a ray for each pixel in the image plane
- $d_y(y) = aspect \left( \frac{y^2 f}{res_y} - f \right) = \frac{res_y}{res_x} \left( \frac{y^2 f}{res_y} - f \right) = (2y - res_y) f_x$



# Generating Camera Rays

- Trace a ray for each pixel in the image plane

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



# Generating Camera Rays

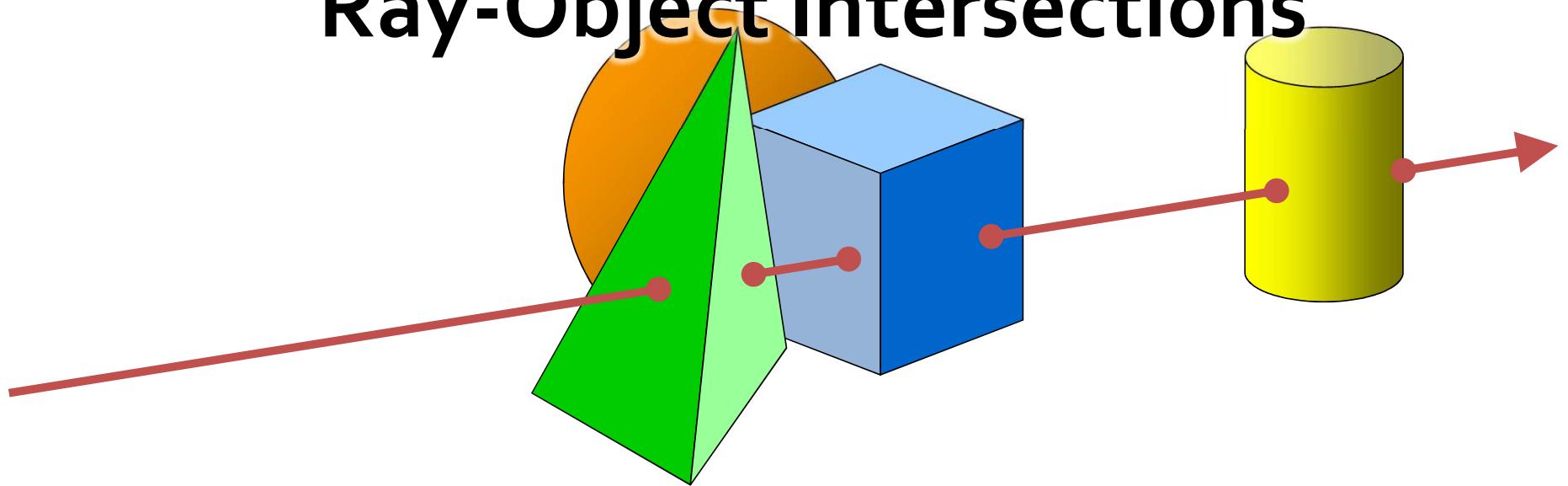
- Trace a ray for each pixel in the image plane

```
ray createCameraRay(x, y) {  
    fov = 90°  
    fx = tan(fov / 2) / resolution.x  
    dx = (2 * x - resolution.x) * fx  
    dy = (2 * y - resolution.y) * fx  
    ray.o = vec3(0, 0, 0)  
    ray.d = normalize(dx, dy, 1)  
    return ray  
}
```

# Ray-Tracer Code

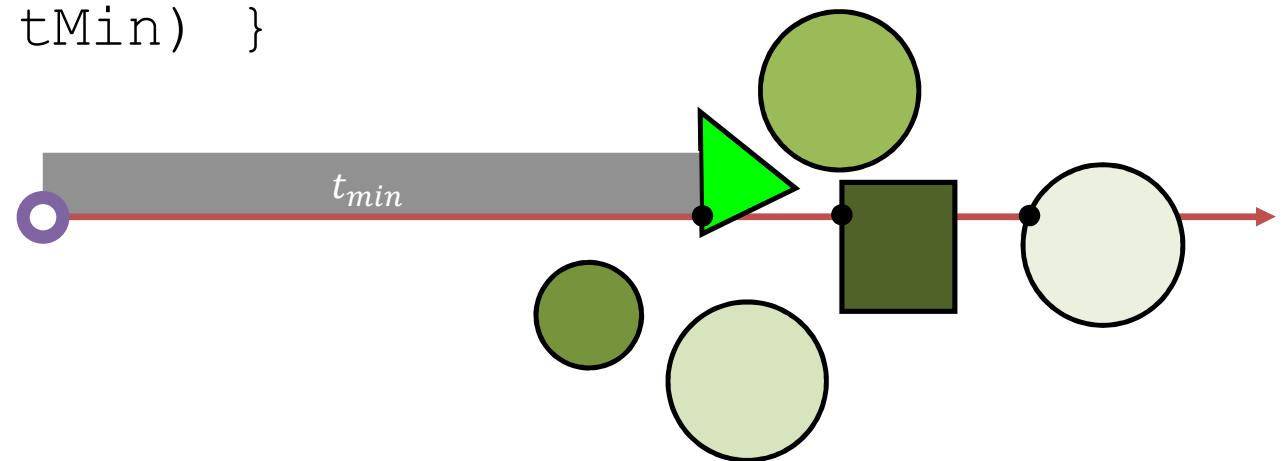
```
renderImage() {  
    foreach pixel x,y in image {  
        ray = createCameraRay(x,y)  
        image[x][y] = trace(ray)  
    } }  
  
color trace(ray) {  
    objectHit = findNearestObjectHit(ray)  
    if(objectHit == background) return bckGrndColor  
    if(obectjHit == light) return lightColor  
    color = directLighting(ray, objectHit)  
    color += trace(reflect(ray, objectHit))  
    color += trace(refract(ray, objectHit))  
    return color  
}
```

# Ray-Object Intersections



# Nearest Object Hit

```
objectHit findNearestObjectHit(ray) {  
    tMin = infinite, objNear = background  
    foreach object in the scene {  
        t = intersect(object, ray)  
        if (0 < t < tMin) {  
            tMin = t, objNear = object } }  
    return (objNear, tMin) }
```

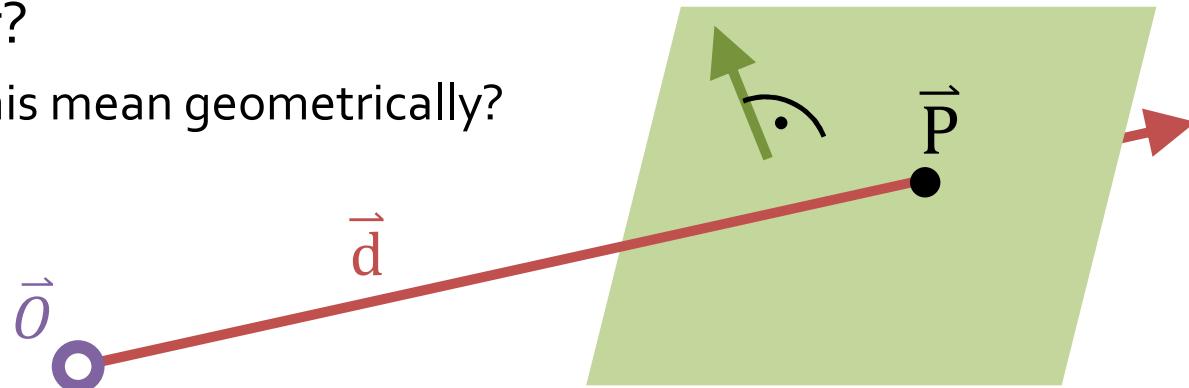


# Ray-Object Intersection

- Find  $t$  at which the *ray* intersects the object
  - Given: ray equation:  $\text{ray}(t) = \vec{o} + t\vec{d}$
  - Given: Implicit equation of object:  $f(\vec{P}) = 0$ 
    - Only points on the surface satisfy implicit equation
  - Solve  $f((\text{ray}(t))) = 0$  for  $t$

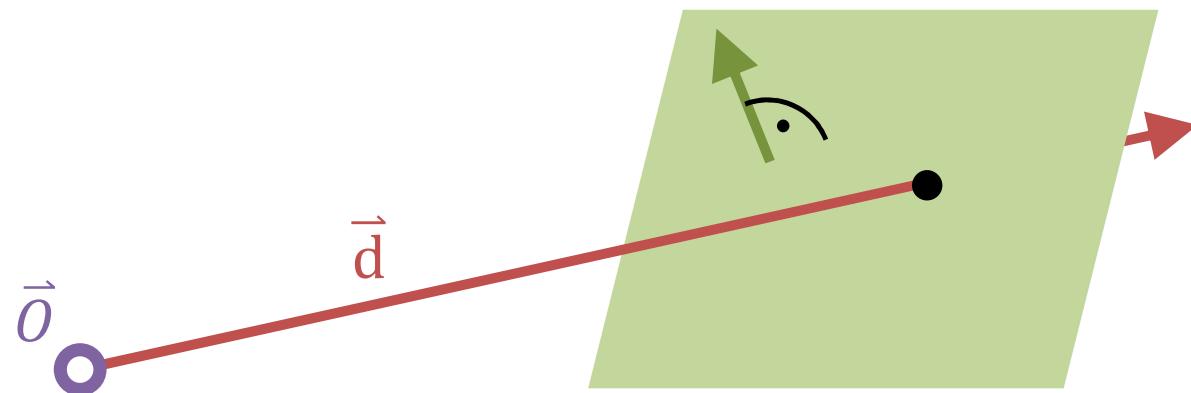
# Ray-Plane Intersection

- Find  $t$  at which the ray intersects the plane
  - Ray equation:  $\text{ray}(t) = \vec{o} + t\vec{d}$
  - Implicit equation of plane with normal  $\vec{n}$  and distance  $d$  to the origin:  
 $\text{plane}(\vec{P}) = \vec{n} \cdot \vec{P} + d = 0$
  - Solve  $\text{plane}(\text{ray}(t)) = \vec{n} \cdot (\vec{o} + t\vec{d}) + d = 0 \Leftrightarrow t = \frac{-d - \vec{n} \cdot \vec{o}}{\vec{n} \cdot \vec{d}}$ 
    - Watch out for?
      - What does this mean geometrically?
    - Code?



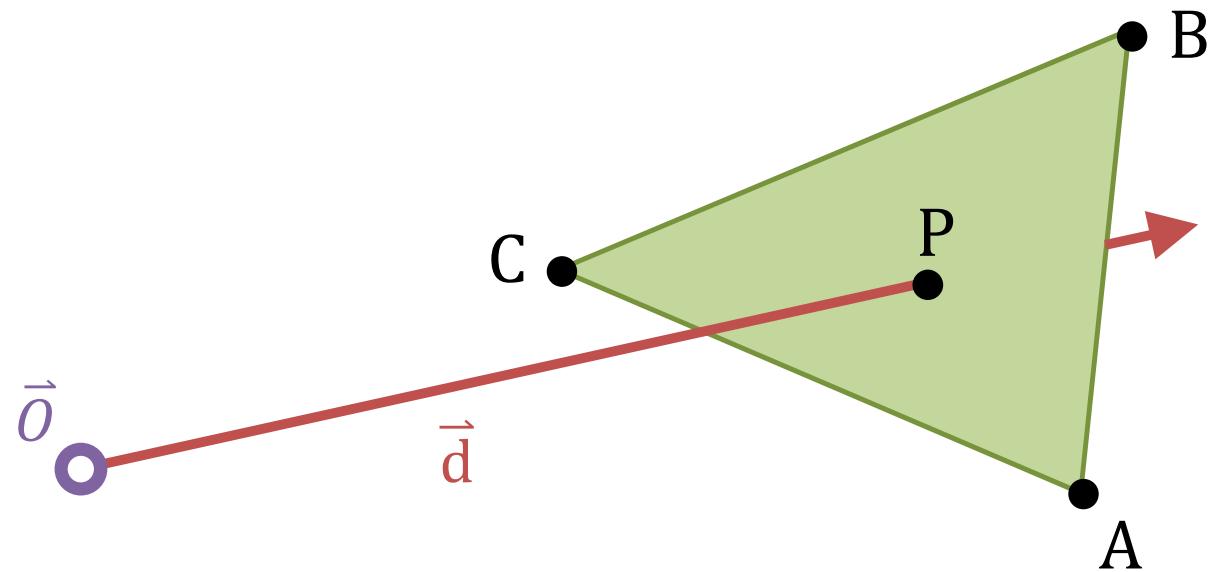
# Ray-Plane Intersection – Code

```
float intersect(plane, ray)
{
    float denom = dot(plane.N, ray.d);
    if(abs(denom) < EPSILON)
    {
        //no intersection
        return -BIGNUMBER;
    }
    return (-plane.d - dot(plane.N, ray.o)) / denom;
}
```



# Ray-Triangle Intersection – Idea

- Similar to ray-plane intersection
- But hit only inside triangle boundaries
- Point inside triangle boundaries problem

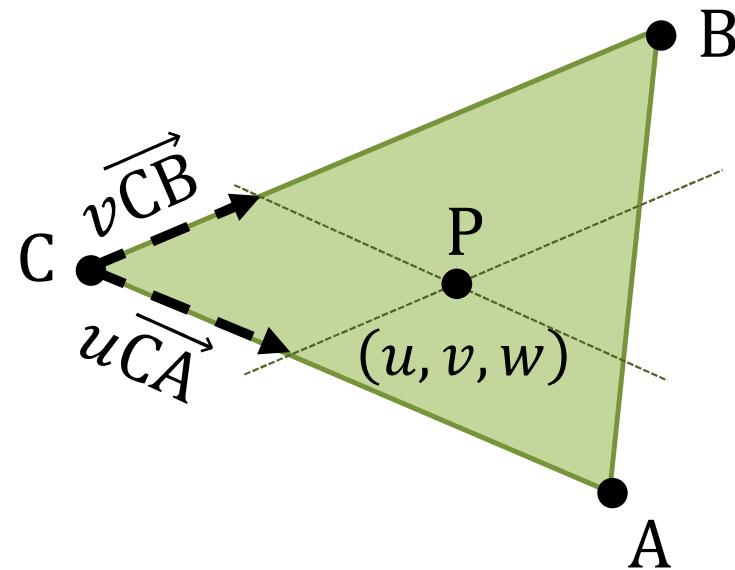


# Barycentric Coordinates of P

- Define  $P = C + u\overrightarrow{CA} + v\overrightarrow{CB}$   
 $= uA + vB + (1 - u - v)C$

$$= uA + vB + wC \text{ with } 1 = u + v + w$$

- Triangle can also be 3d



## BC – Inside Triangle Test

- Also outside triangle
  - In triangle if  $(u, v, w)$  all same sign
    - For CCW  $(u, v, w) \geq 0$

# Ray Triangle Intersection

- $ray(t) = \vec{o} + t\vec{d}$
- Point on triangle (Barycentric coordinates)  
$$triangle(u, v) = u\vec{A} + v\vec{B} + (1 - u - v)\vec{C}$$
- Intersection  $\vec{o} + t\vec{d} = u\vec{A} + v\vec{B} + (1 - u - v)\vec{C}$

# Ray Triangle Intersection

- Intersection  $\vec{O} + t\vec{d} = u\vec{A} + v\vec{B} + (1 - u - v)\vec{C}$

- Rearranged  $\vec{O} - \vec{C} = (-\vec{d} \quad \vec{A} - \vec{C} \quad \vec{B} - \vec{C}) \begin{pmatrix} t \\ u \\ v \end{pmatrix}$

- Linear system!

- Solve with Cramer's rule

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

# Ray Triangle Intersection: Implementation

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\det(-\vec{d} \ \vec{A}-\vec{C} \ \vec{B}-\vec{C})} \begin{pmatrix} \det(\vec{o} - \vec{C} \ \vec{A} - \vec{C} \ \vec{B} - \vec{C}) \\ \det(-\vec{d} \ \vec{o} - \vec{C} \ \vec{B} - \vec{C}) \\ \det(-\vec{d} \ \vec{A} - \vec{C} \ \vec{o} - \vec{C}) \end{pmatrix}$$

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

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

# Ray Triangle Intersection: Implementation

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

$$E_1 = \vec{A} - \vec{C} \quad E_2 = \vec{B} - \vec{C} \quad S = \vec{O} - \vec{C}$$

$$P = (\vec{d} \times E_2) \quad Q = S \times E_1$$

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

# Ray Triangle Intersection: Code

```

bool rayTriIntersect(in O,D, A,B,C, out u,v,t)
{
    E1 = A-C
    E2 = B-C
    P = cross(D,E2)
    detM = dot(P,E1)

    if(detM > -eps && detM < eps)
        return false
    o == detM

    f = 1/detM
    S = O-C
    u = f*dot(P,S)

    if(u < 0 || 1 < u)
        return false
    u outside [0,1]

    Q = cross(S,E1)
    v = f*dot(Q,D)

    if(v < 0 || 1 < u+v)
        return false

    t = f*dot(Q,E2)

    return true
}

```



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

# Ray-Sphere Intersection

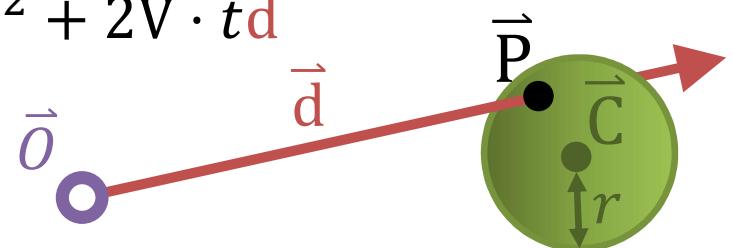
- $ray(t) = \vec{0} + t\vec{d}$
- Implicit equation of sphere with center  $\vec{C}$  and radius  $r$ :

$$sphere(\vec{P}) = |\vec{P} - \vec{C}|^2 - r^2 = 0$$

- Solve  $sphere(ray(t)) = |\vec{0} + t\vec{d} - \vec{C}|^2 - r^2 = 0$

$$\vec{V} := \vec{0} - \vec{C} \quad |\vec{A} + \vec{B}|^2 = \vec{A}^2 + \vec{B}^2 + 2(\vec{A} \cdot \vec{B})$$

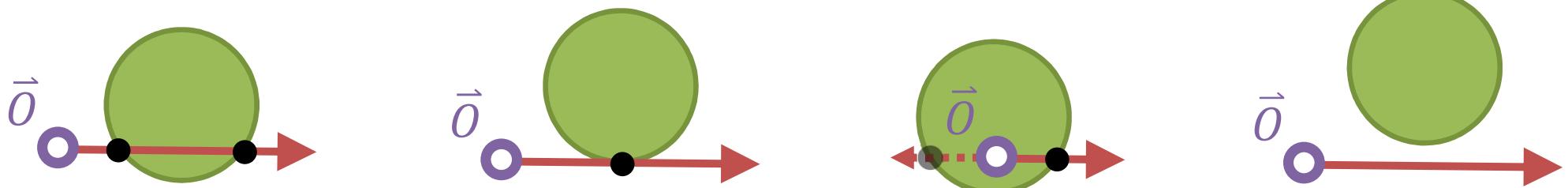
$$|\vec{V} + t\vec{d}|^2 - r^2 = \vec{V}^2 - r^2 + t^2\vec{d}^2 + 2\vec{V} \cdot t\vec{d}$$



# Ray-Sphere Intersection

$$\begin{aligned} |\vec{V} + t\vec{d}|^2 - r^2 &= \vec{V}^2 - r^2 + t^2\vec{d}^2 + 2\vec{V} \cdot t\vec{d} \\ \stackrel{\vec{d}^2=1}{\iff} t^2 + (2\vec{V} \cdot \vec{d})t + (\vec{V}^2 - r^2) &= 0 \\ t = -(\vec{V} \cdot \vec{d}) \pm \sqrt{(\vec{V} \cdot \vec{d})^2 - (\vec{V}^2 - r^2)} \end{aligned}$$

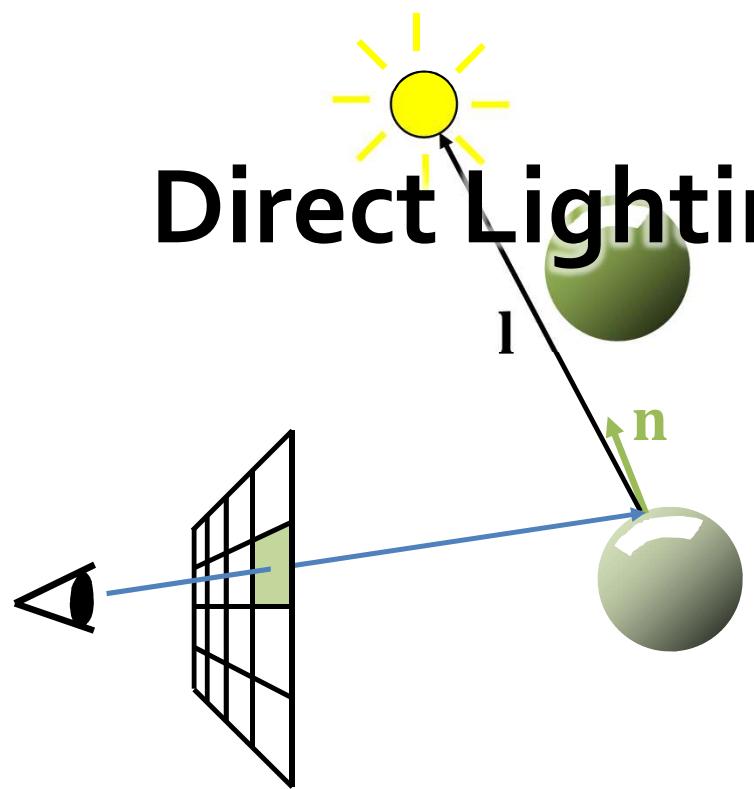
- Real solutions, indicates one or two intersections
- Negative solutions are behind the eye
- If discriminant is negative, the ray missed the sphere



# Ray-Tracer Code

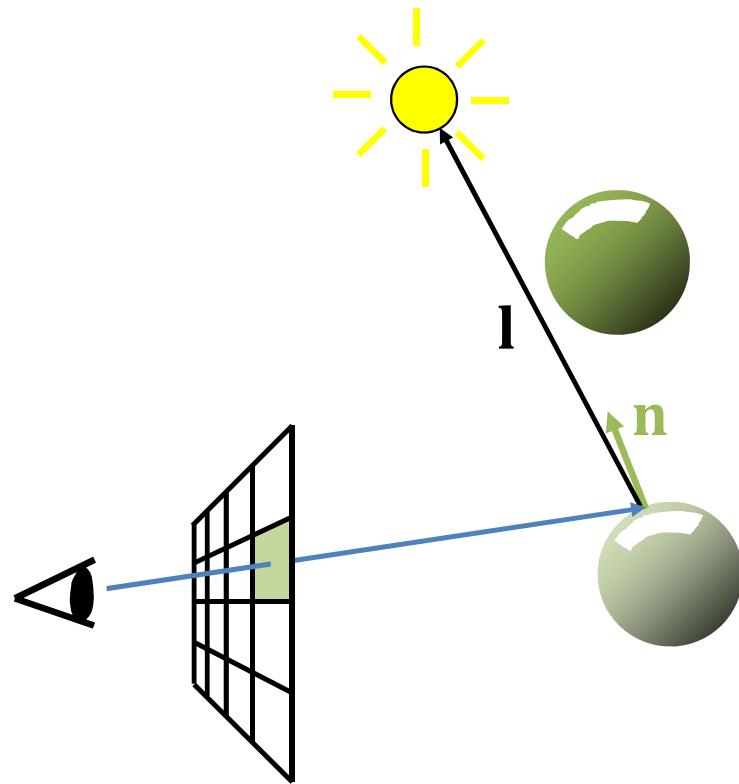
```
renderImage() {  
    foreach pixel x,y in image {  
        ray = createCameraRay(x,y)  
        image[x][y] = trace(ray)  
    } }  
  
color trace(ray) {  
    objectHit = findNearestObjectHit(ray)  
    if(objectHit == background) return bckGrndColor  
    if(obectjHit == light) return lightColor  
    color = directLighting(ray, objectHit)  
    color += trace(reflect(ray, objectHit))  
    color += trace(refract(ray, objectHit))  
    return color  
}
```

# Direct Lighting



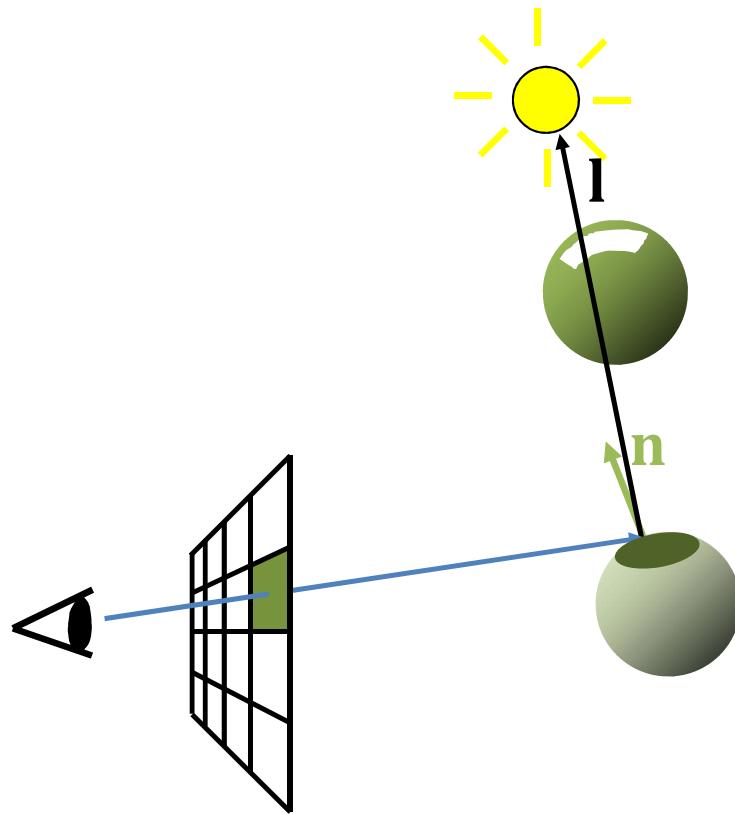
# Direct Lighting – Diffuse (Lambert)

- For each light  $C_{direct} = \max(0, \mathbf{n} \cdot \mathbf{l}) * Color_{object} * Color_{light}$



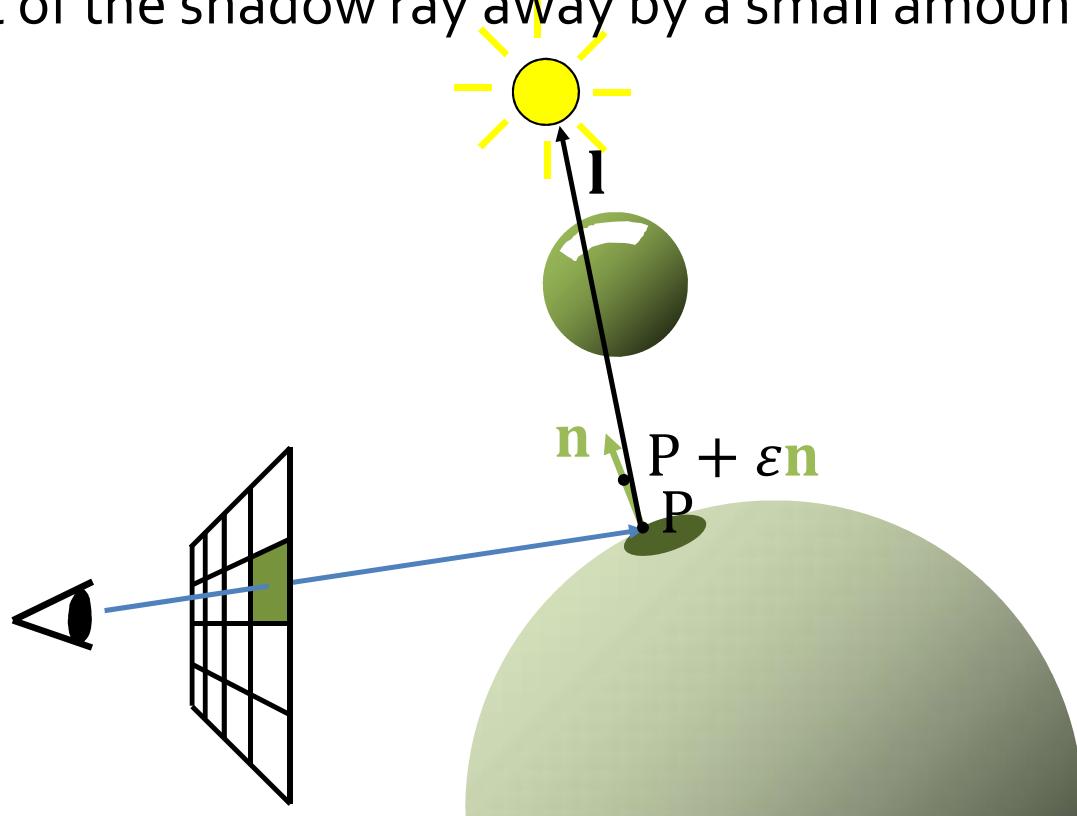
# Direct Lighting – Adding Shadows

- Add local lighting from a light source only if point is seen by this light

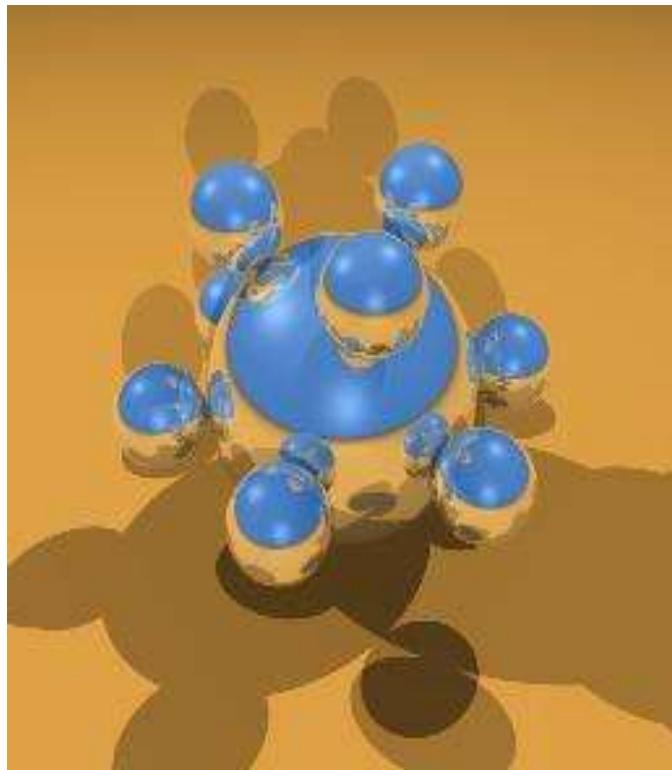


# Direct Lighting – “Self-Shadowing”

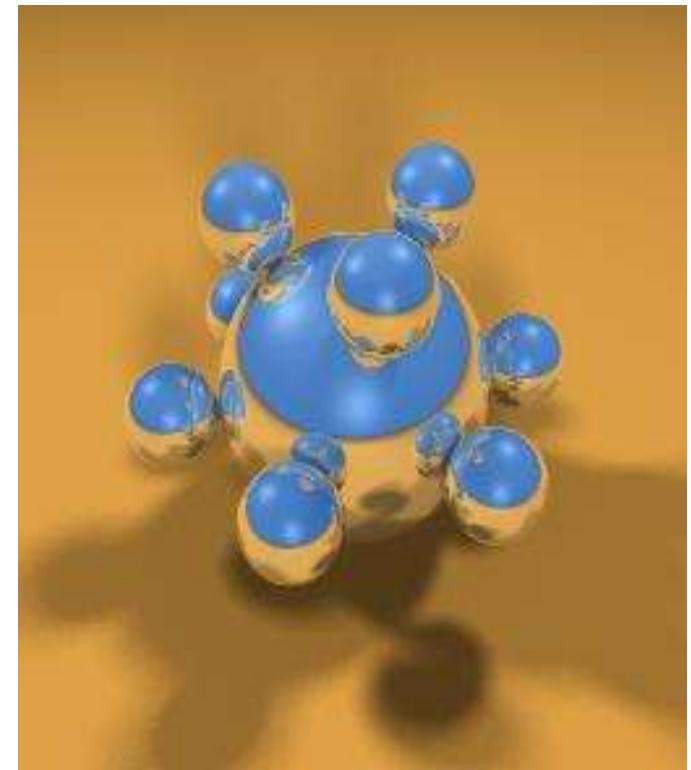
- Intersection of shadow ray with object itself
- Move start point of the shadow ray away by a small amount



# Direct Lighting – Soft Shadow



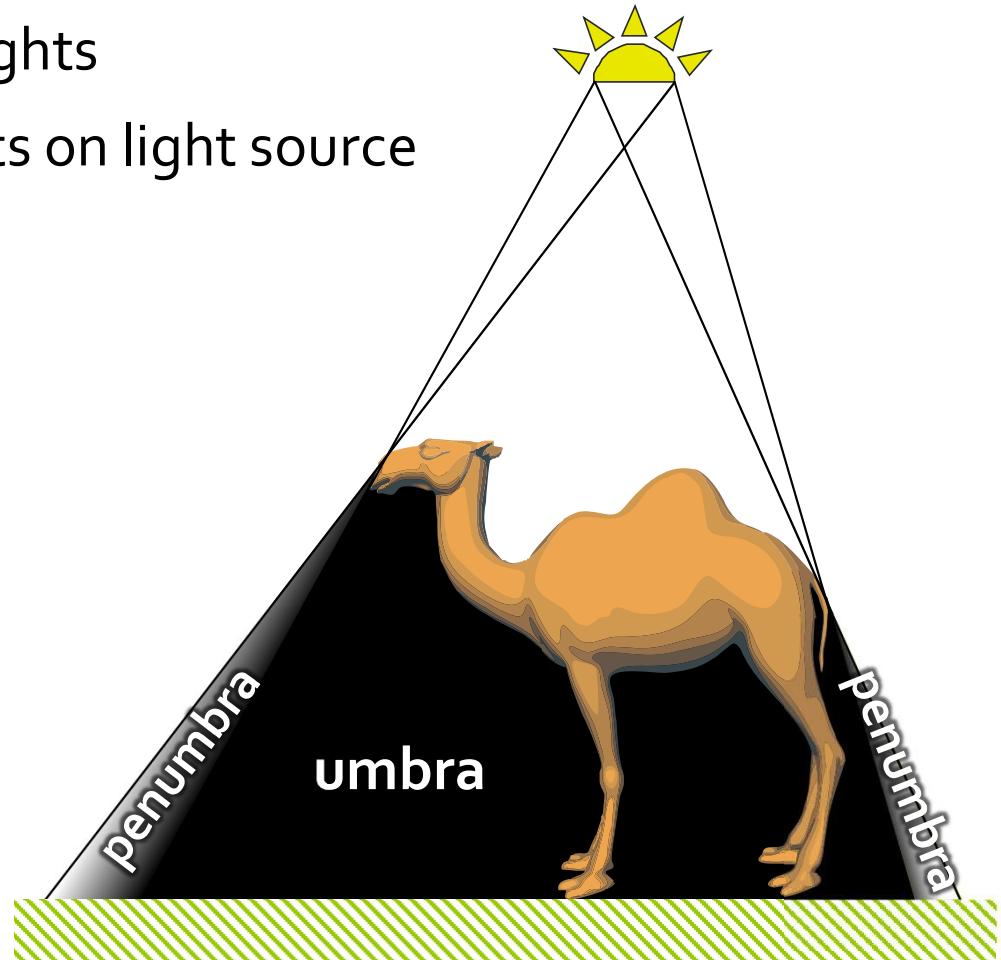
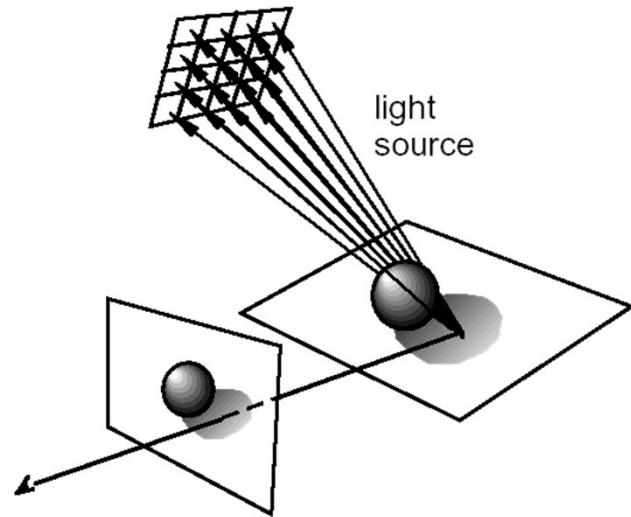
Hard shadow



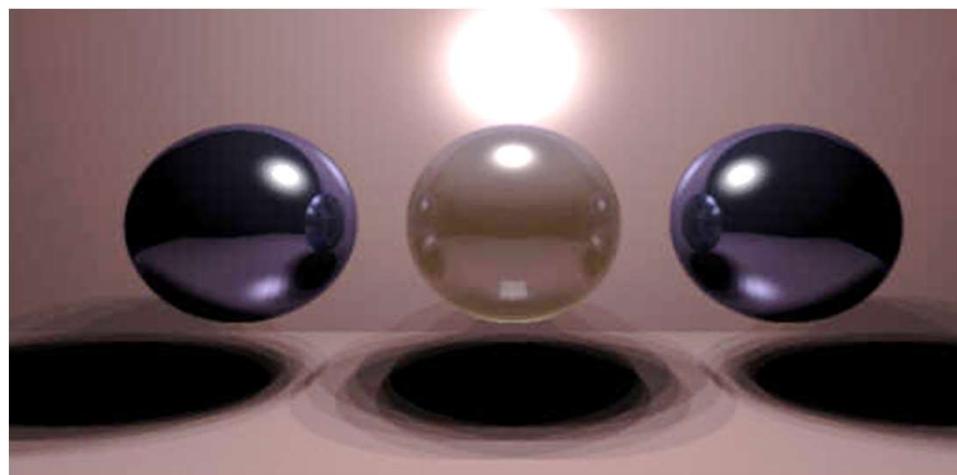
Soft shadow

# Direct Lighting – Area Lights

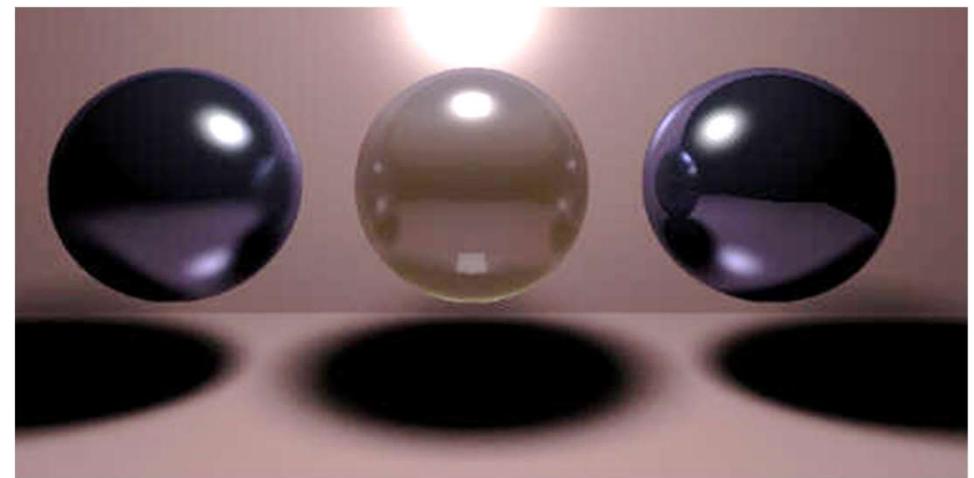
- Soft shadows are created by area lights
- Trace shadow rays to multiple points on light source



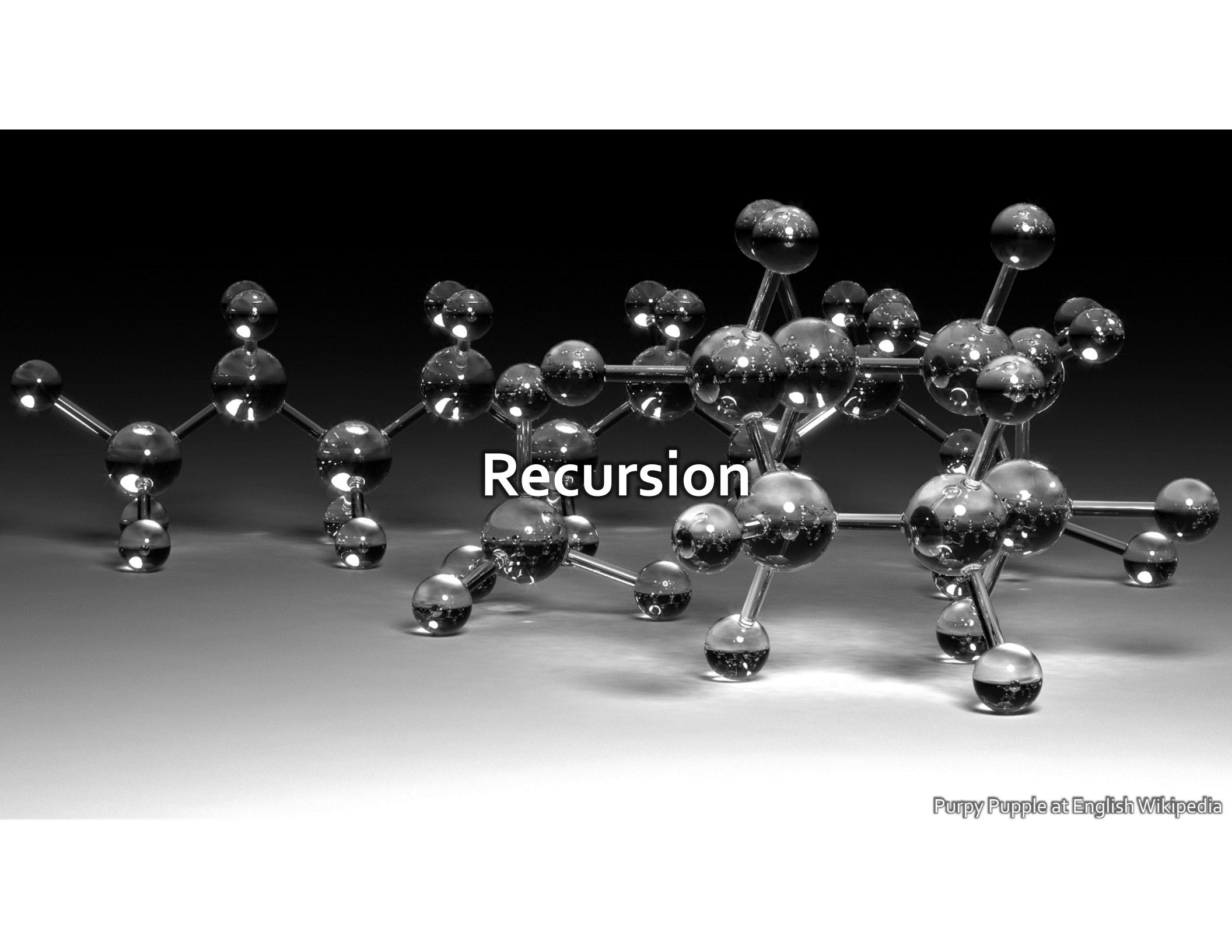
# Direct Lighting – Area Light Sources



9 shadow rays (3\*3 grid)



16,384 shadow rays (128\*128 grid)



# Recursion

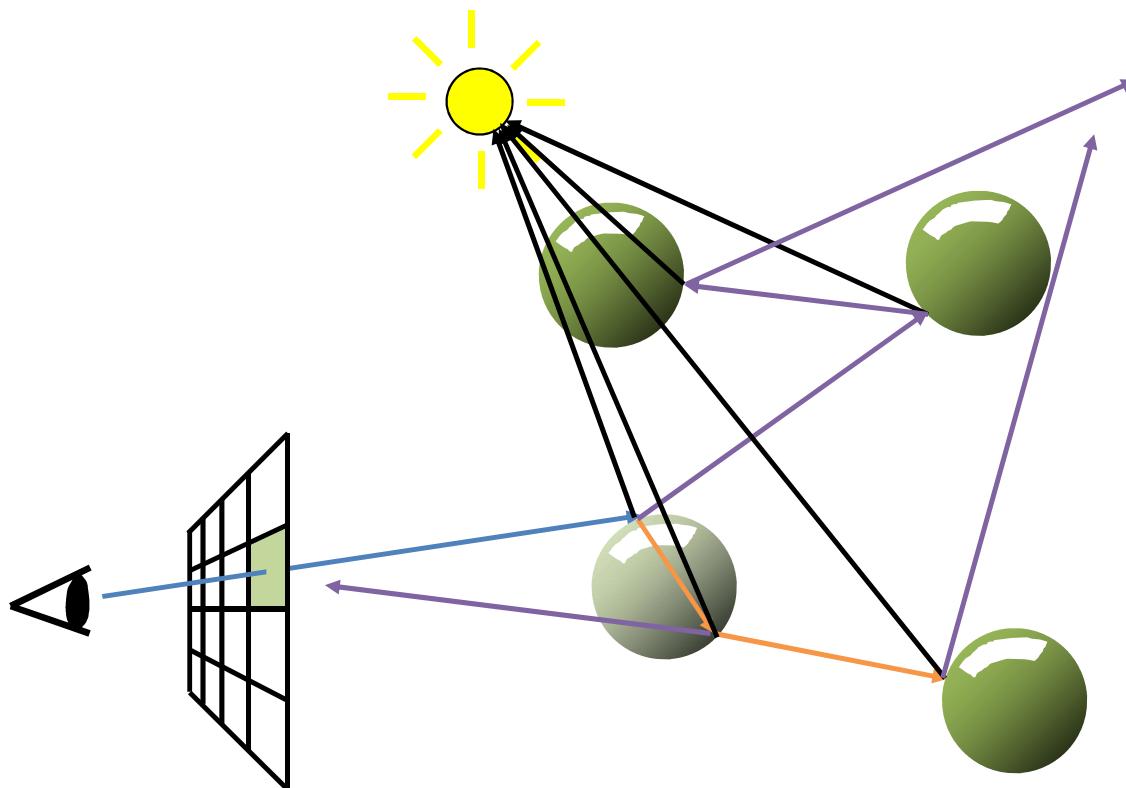
Purpy Pupple at English Wikipedia

# Ray-Tracer Code

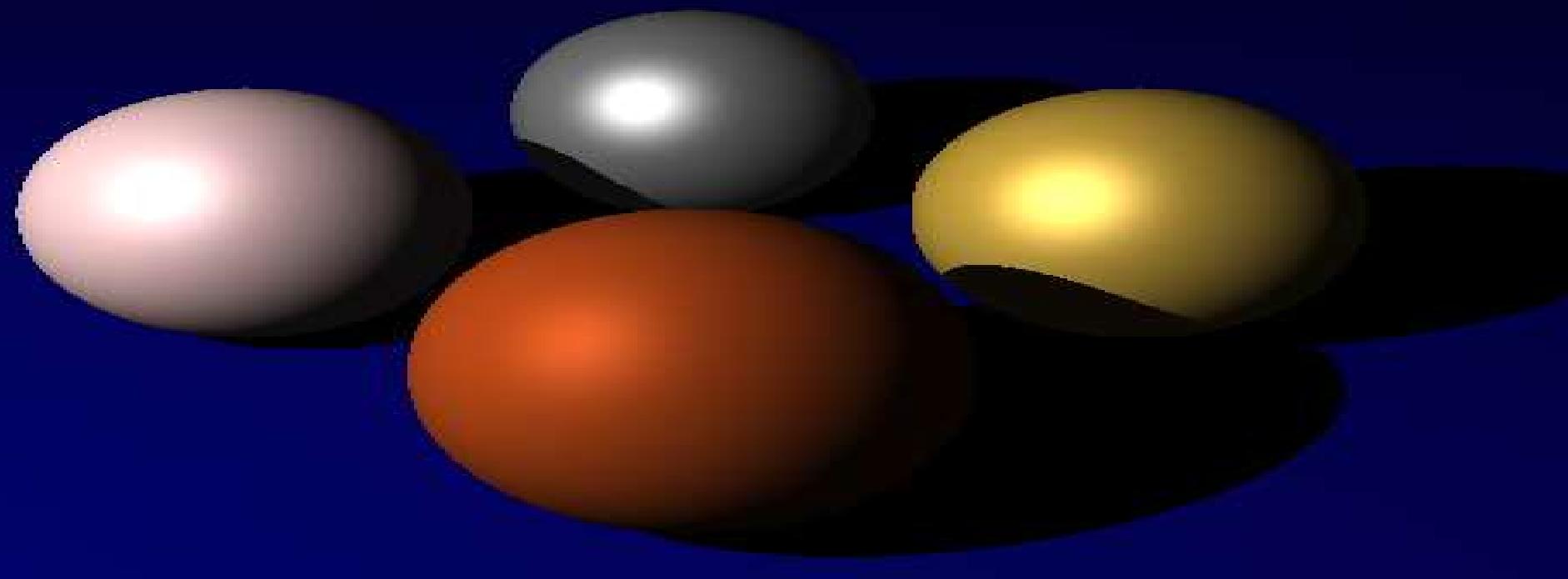
```
renderImage() {  
    foreach pixel x,y in image {  
        ray = createCameraRay(x,y)  
        image[x][y] = trace(ray)  
    } }  
  
color trace(ray) {  
    objectHit = findNearestObjectHit(ray)  
    if(objectHit == background) return bckGrndColor  
    if(obectjHit == light) return lightColor  
    color = directLighting(ray, objectHit)  
    color += trace(reflect(ray, objectHit))  
    color += trace(refract(ray, objectHit))  
    return color  
}
```

# Ray-Tracer Recursion

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

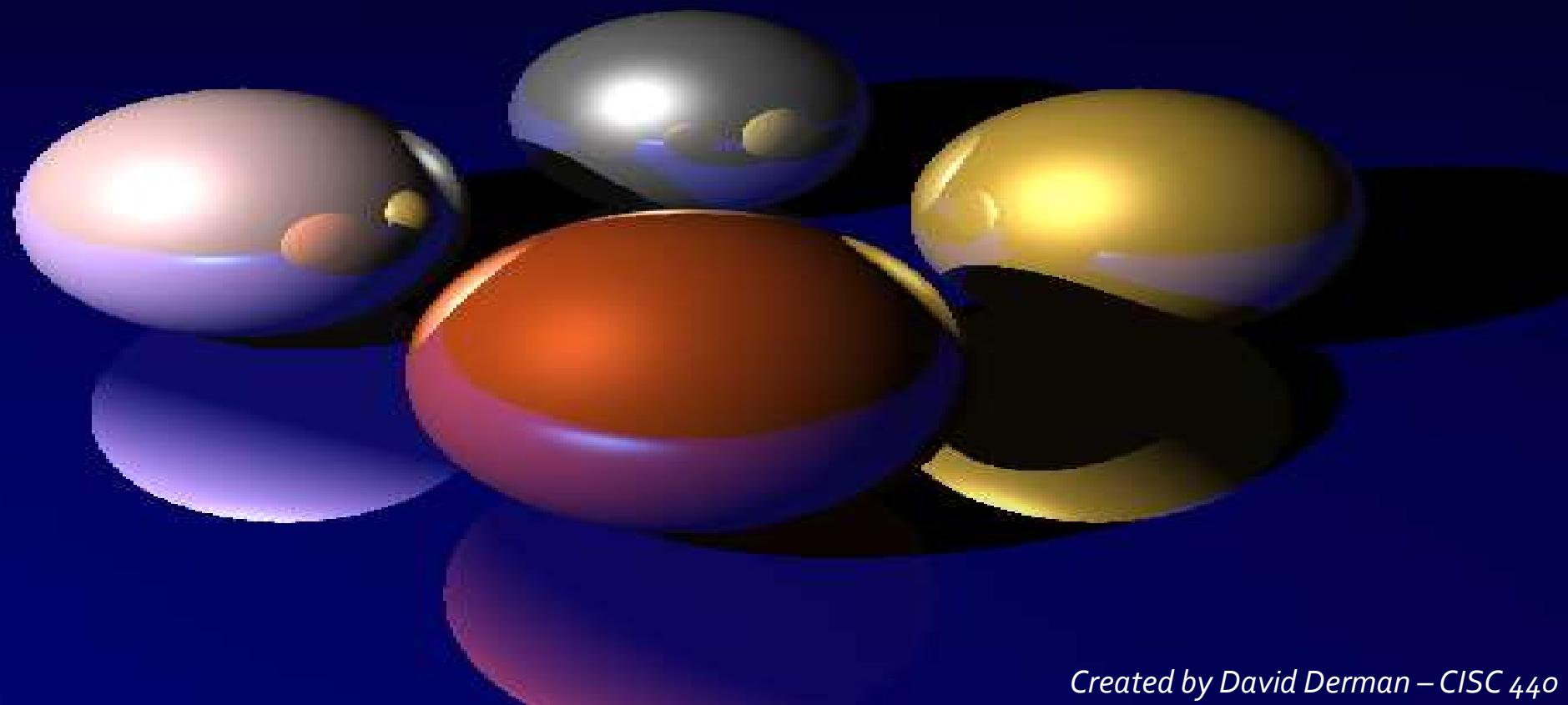


# No Reflection



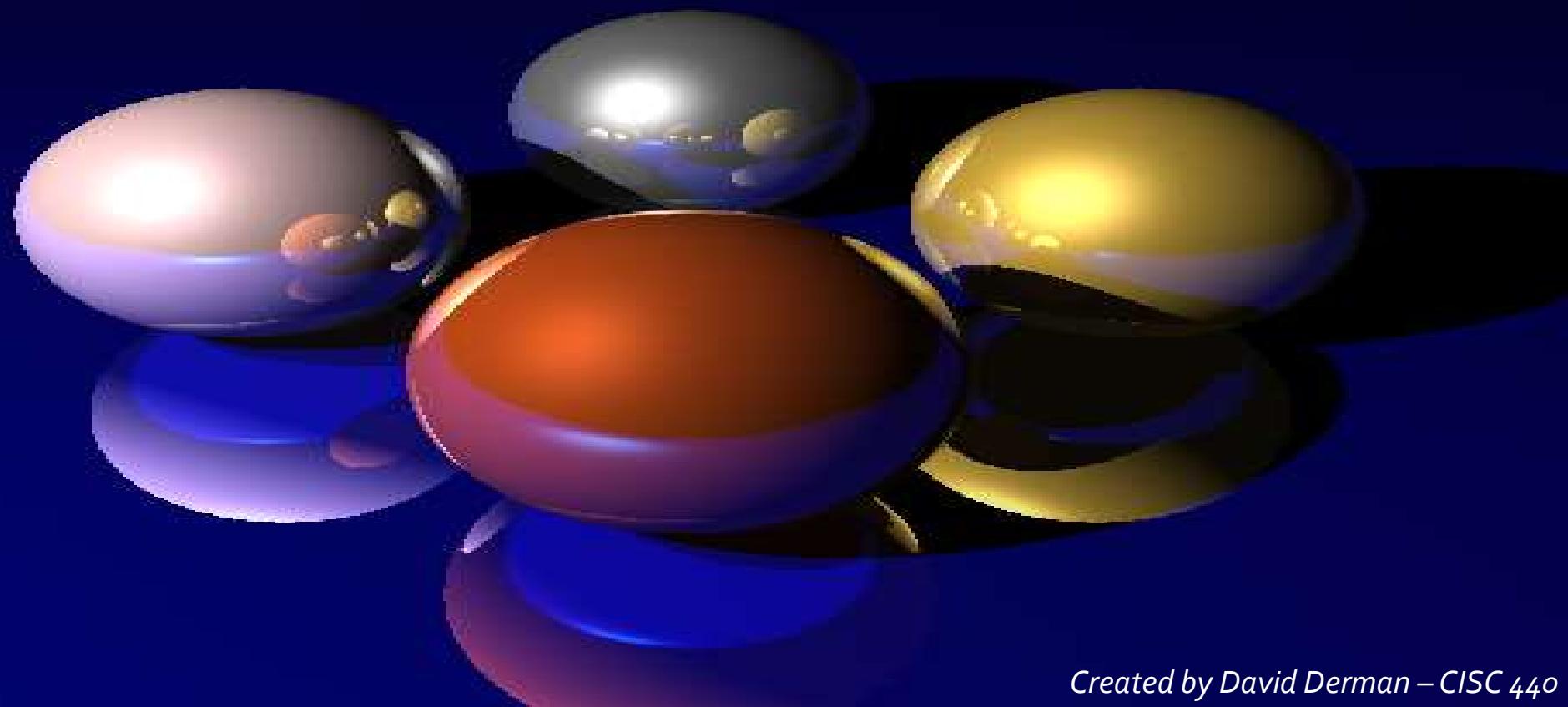
*Created by David Derman – CISC 440*

# Reflection (1)



*Created by David Derman – CISC 440*

# Reflection (2)



*Created by David Derman – CISC 440*

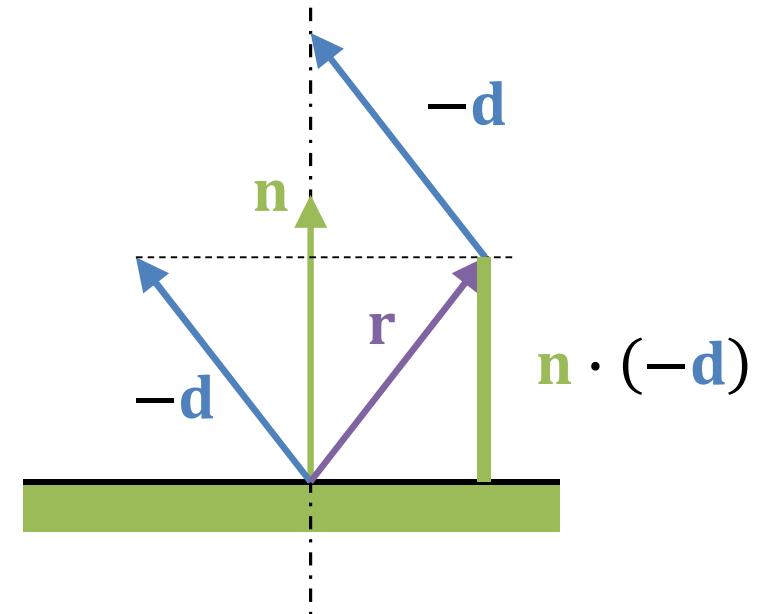
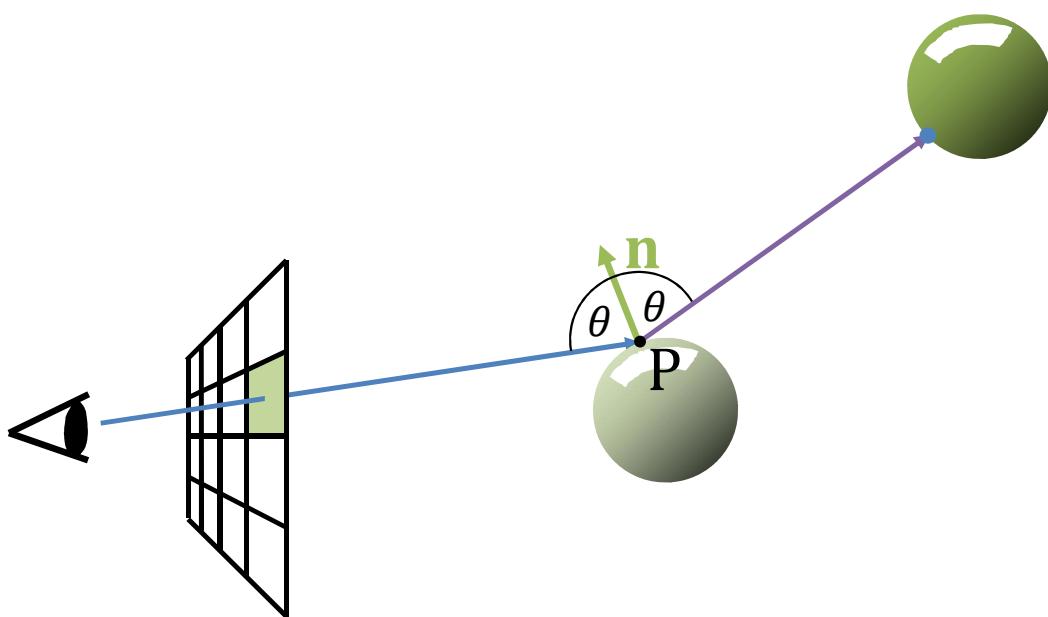
# Reflection (3)



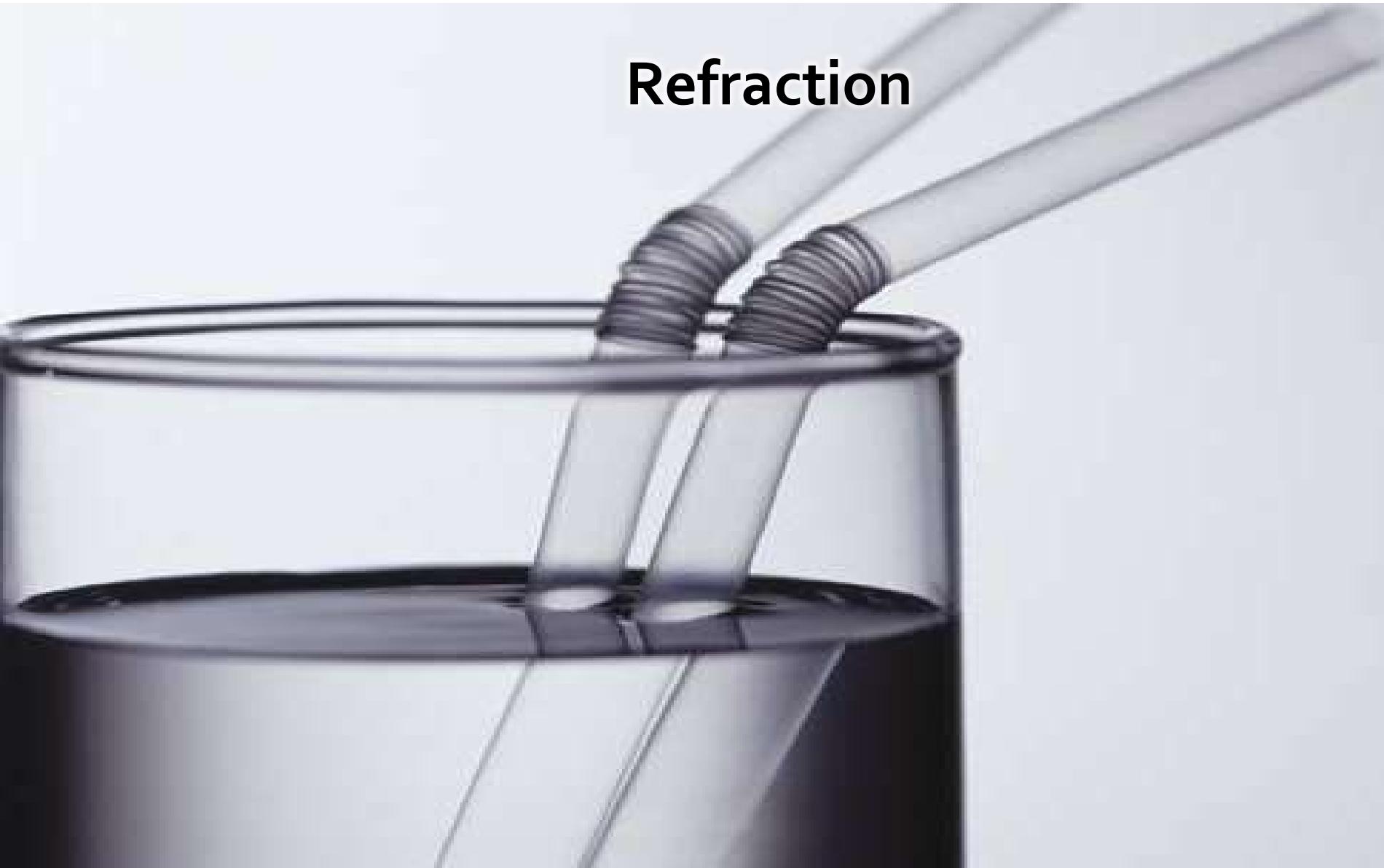
*Created by David Derman – CISC 440*

# Reflection

- Incoming  $ray(O, \mathbf{d})$
- Outgoing  $ray(P, \text{reflect}(\mathbf{d}, \mathbf{n}))$
- $\text{reflect}(\mathbf{d}, \mathbf{n}) = \mathbf{d} - 2(\mathbf{n} \cdot \mathbf{d})\mathbf{n}$
- $\mathbf{r} + (-\mathbf{d}) = 2(\mathbf{n} \cdot (-\mathbf{d}))\mathbf{n}$
- $\mathbf{r} = \mathbf{d} - 2(\mathbf{n} \cdot \mathbf{d})\mathbf{n}$

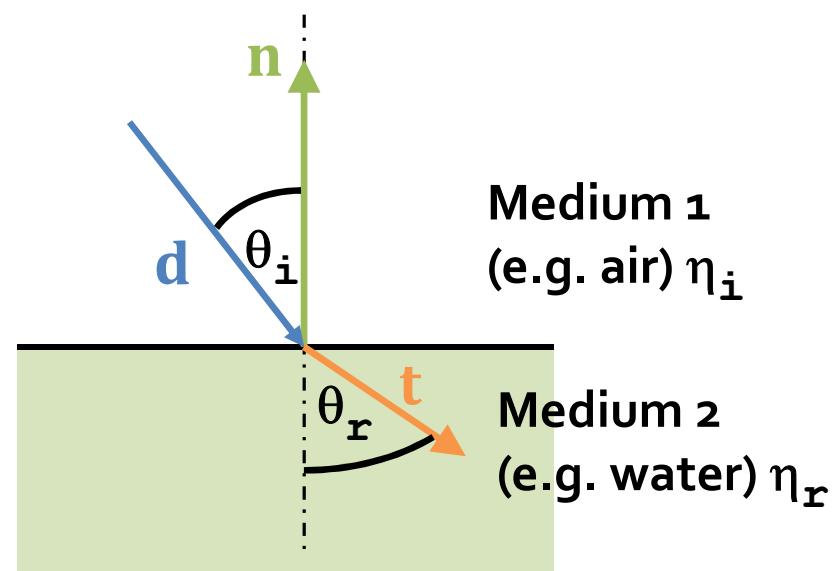
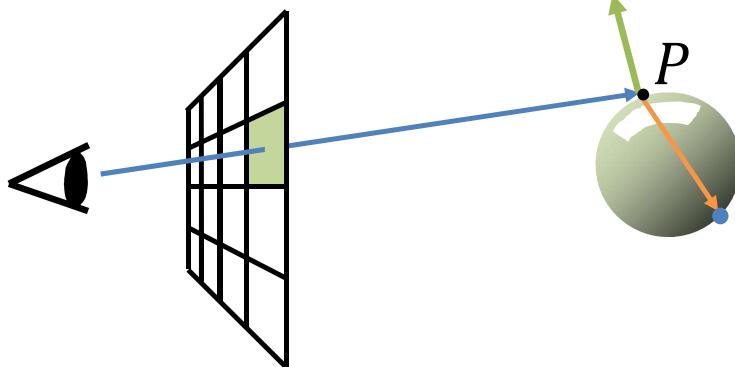


# Refraction



# Refraction

- Need *index of refraction*  $\eta$  of medium (air, glass, ...)
- Snell's Law  $\frac{\sin i}{\sin \theta_r} = \frac{\eta_r}{\eta_i}$
- Outgoing *ray*(P, refract( $\mathbf{d}$ ,  $\mathbf{n}$ ))
- refract( $\mathbf{d}$ ,  $\mathbf{n}$ ) is complex



# Acceleration structures for ray-tracing

