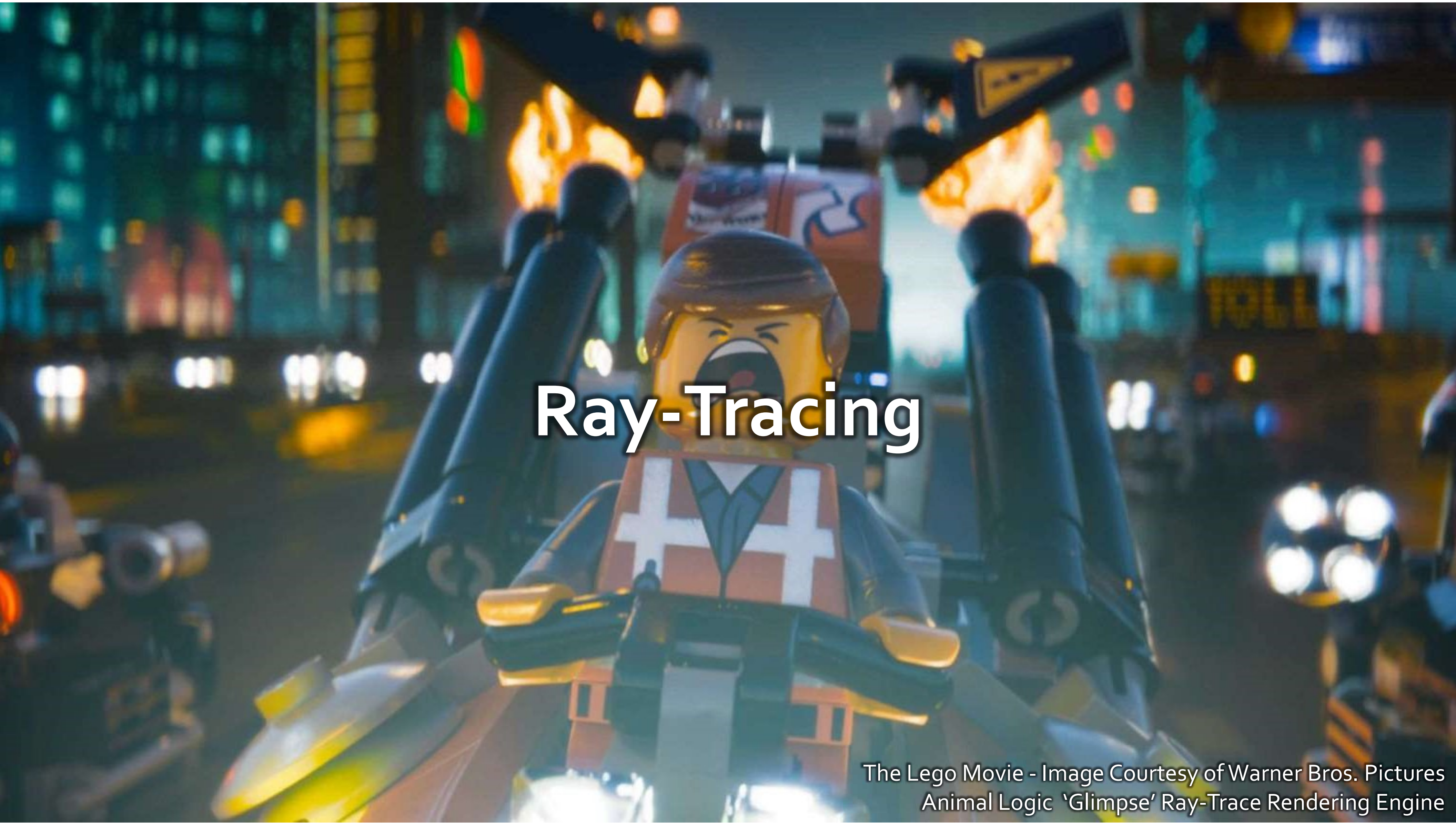
Ray-Tracing



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)

Octane Real-Time Ray-Tracer

Paul Heckbert

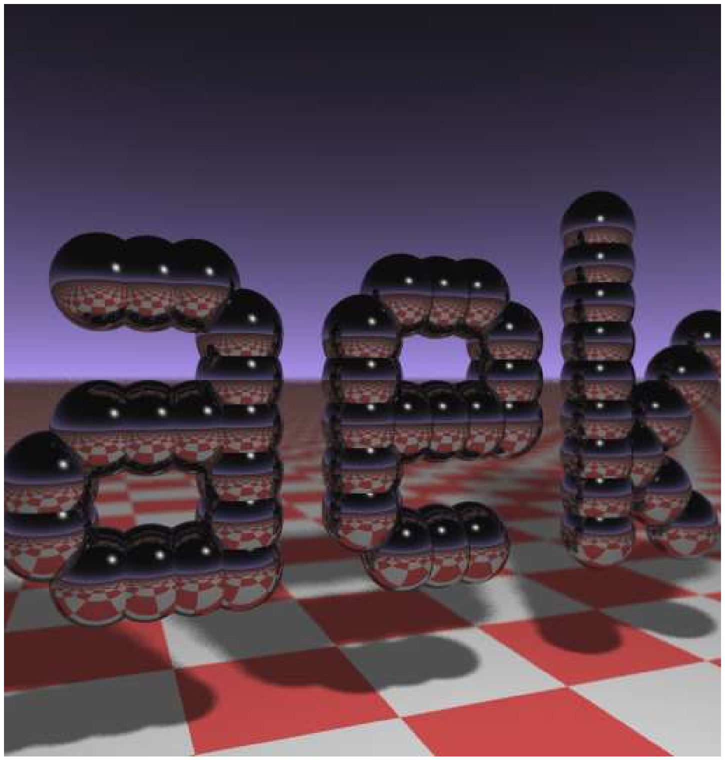
Dessert Foods Division
Pixar
PO Box 13719
San Rafael CA, 94913

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,.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



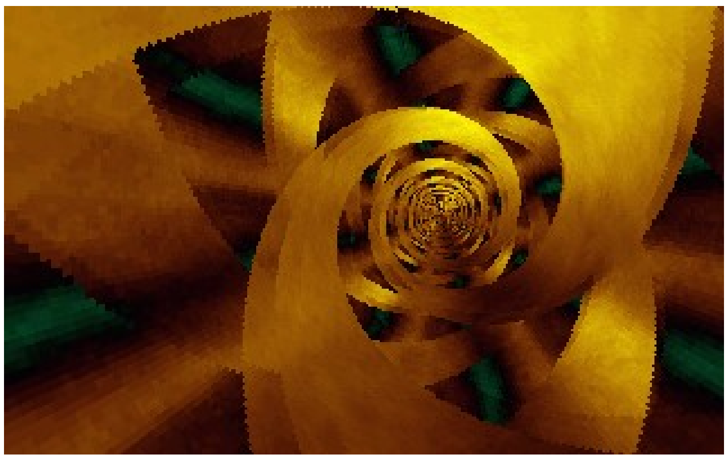
[fabiensanglard.net/rayTracing back of business card](fabiensanglard.net/rayTracing_back_of_business_card)

Vector, World, Tracer, Sampler, Main

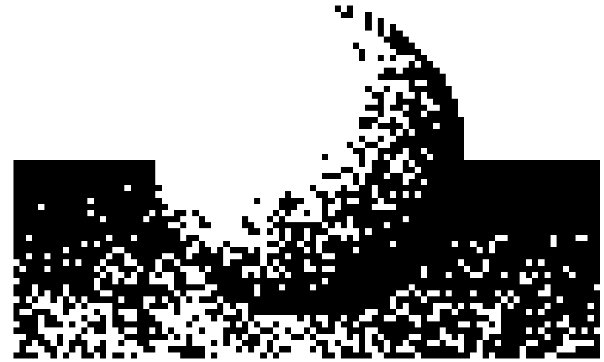
```
#include <stdlib.h> #include <stdio.h>
#include <math.h> typedef int i; typedef float f;
struct v{ f x,y,z;v operator+(v r){return v(x+r.x ,y+r.y,z+r.z);
}v operator*(f r){return v(x*r,y*r,z*r);}f operator%(v r){return
x*r.x+y*r.y+z*r.z;}v(){}v operator^(v r ){return v(y*r.z-z*r.y,
z*r.x-x*r.z,x*r.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

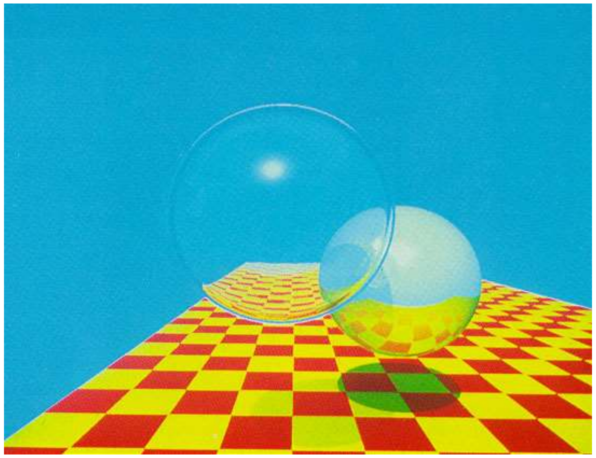


256 byte program

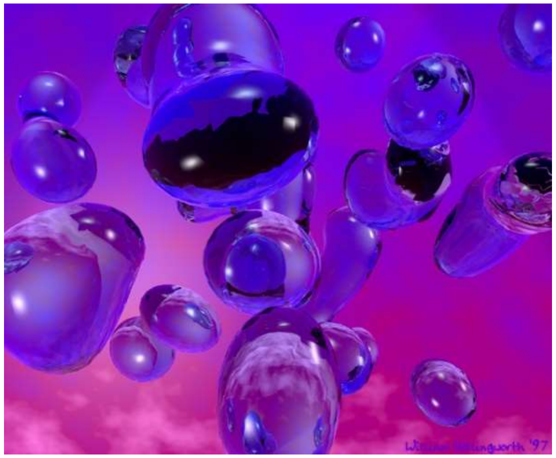


422 byte program for a Casio FX7000Ga, Stéphane Gourichon, 1991

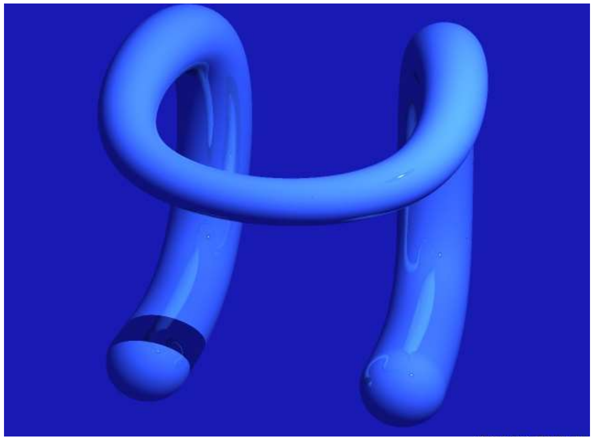
Intersectable = renderable



Turner Whitted



William Hollingworth



Henrik Wann Jensen

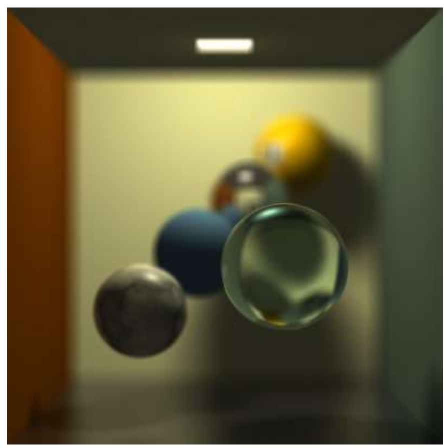
Ken Musgrave

Reflections, Refractions

Gilles Tran, wikimedia

Stochastic Effects

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



Jason Waltman

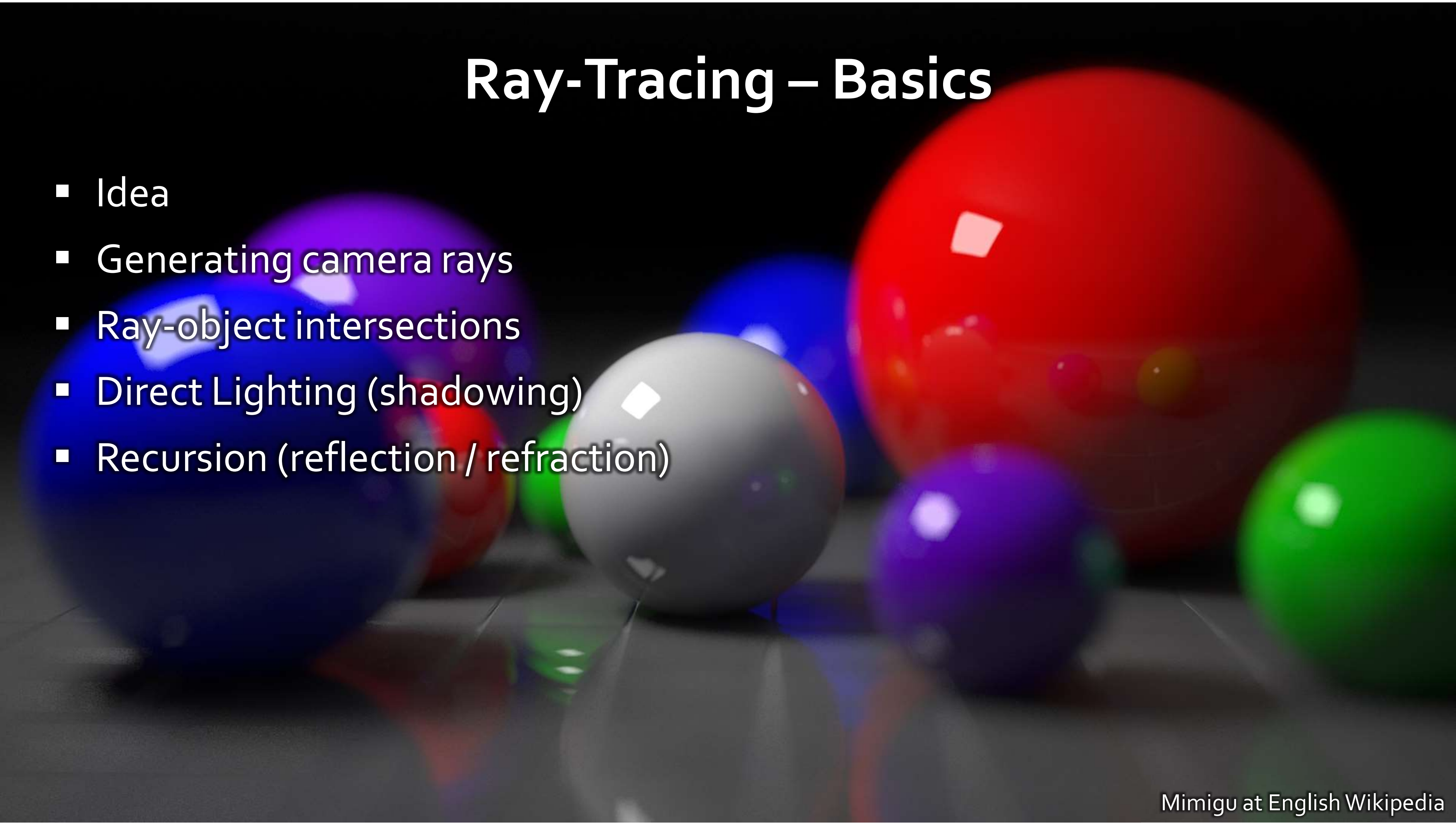
Matt Roberts

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

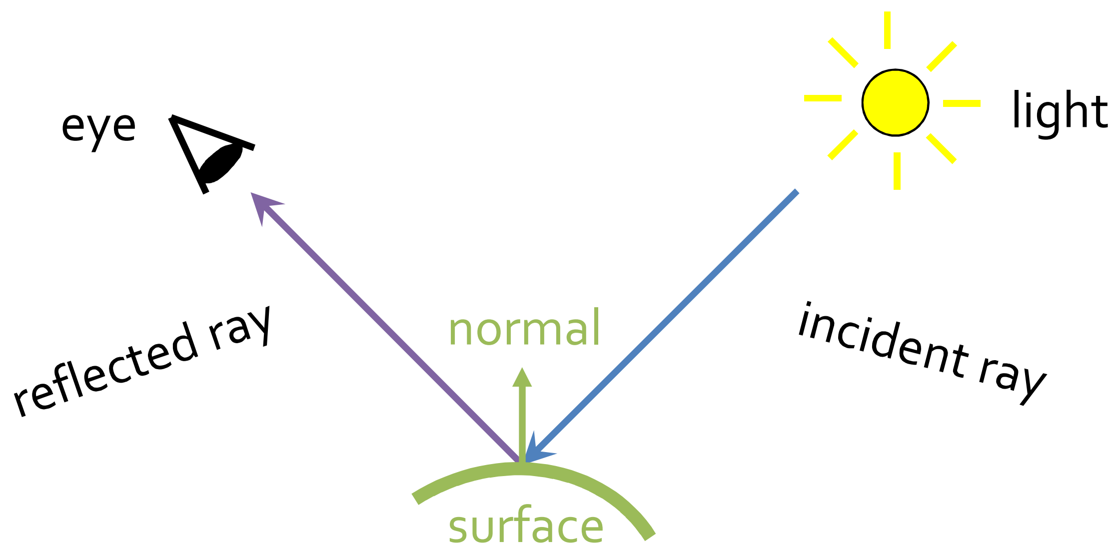
- Idea
- Generating camera rays
- Ray-object intersections
- Direct Lighting (shadowing)
- Recursion (reflection / refraction)



Mimigu at English Wikipedia

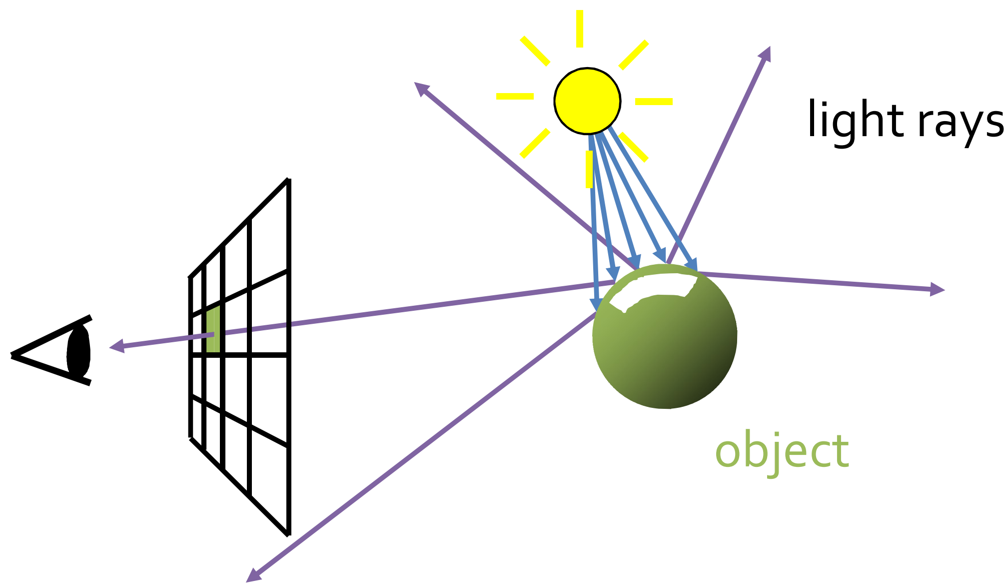
Idea

- Simulate light rays from light source to eye



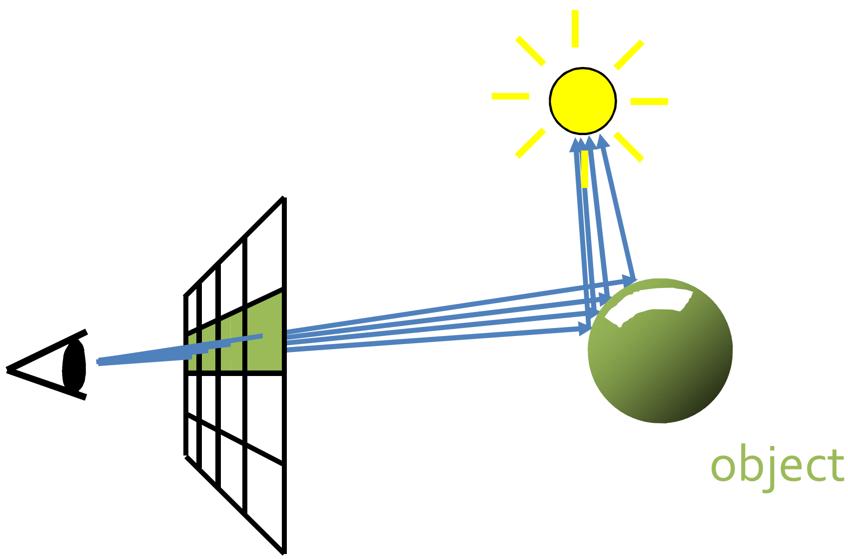
“Forward” Ray-Tracing

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



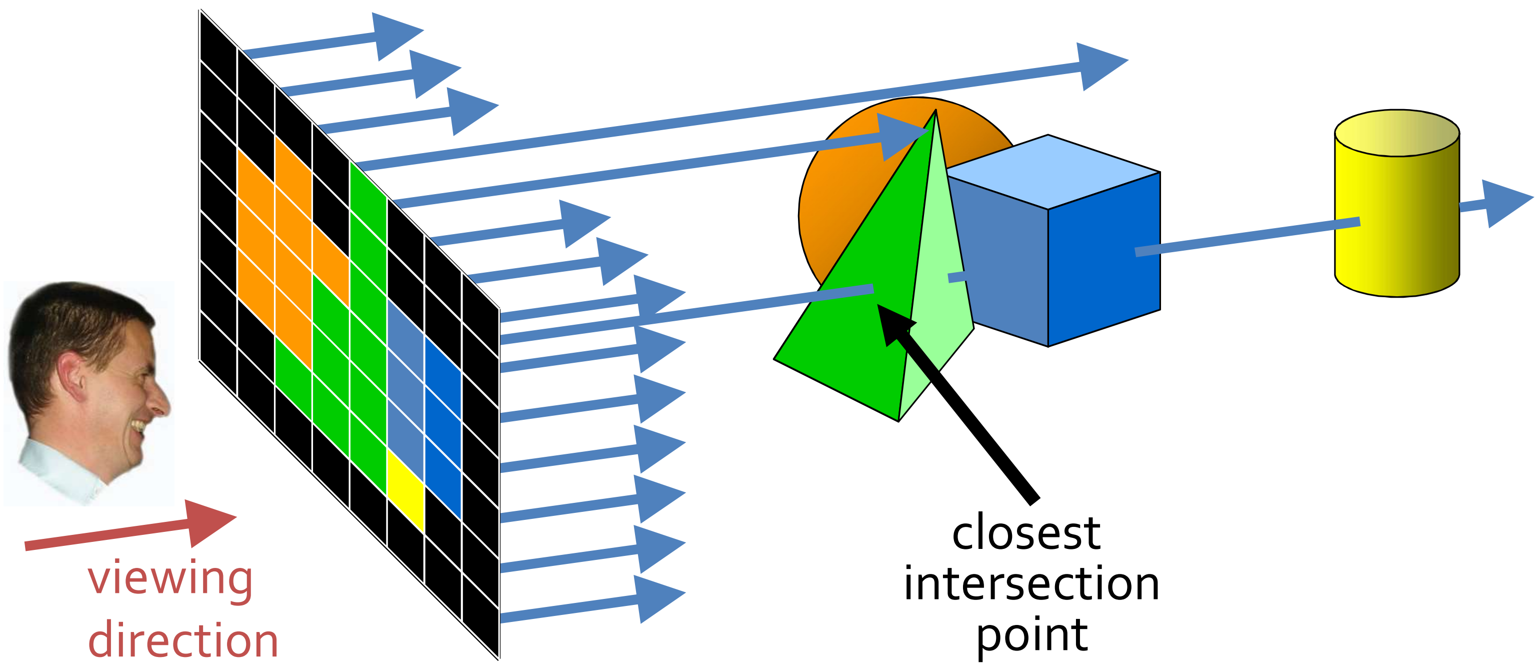
"Backward" Ray-Tracing

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



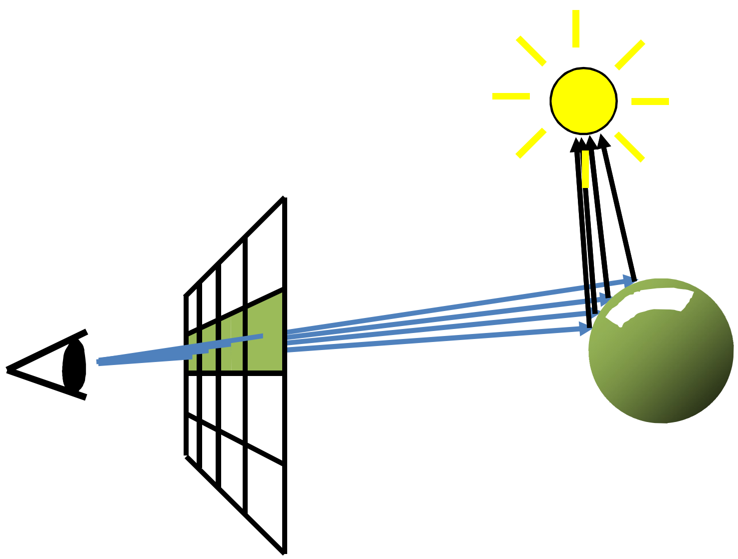
How many Rays?

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



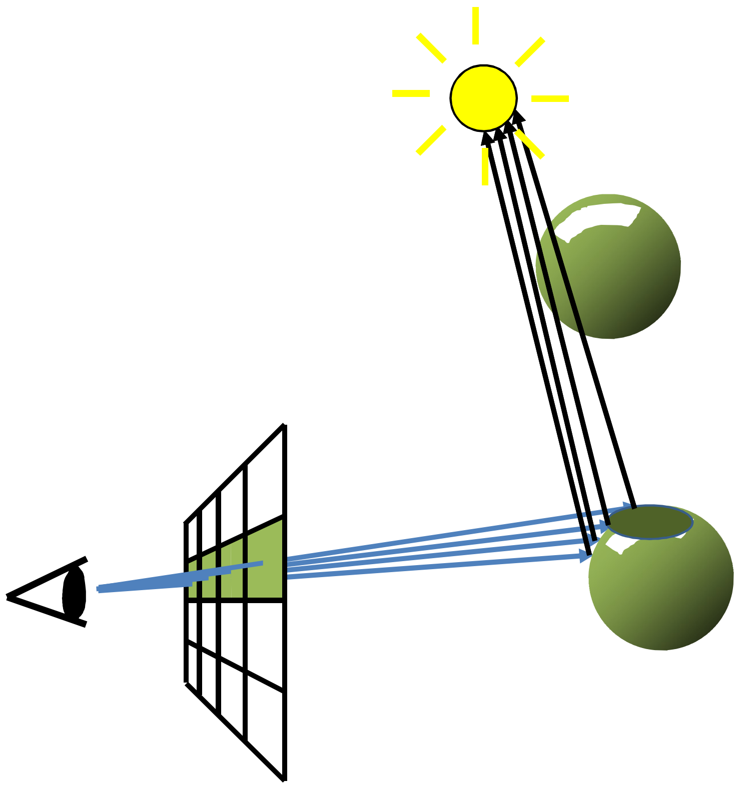
Types of Rays

- Primary rays
- Shadow rays



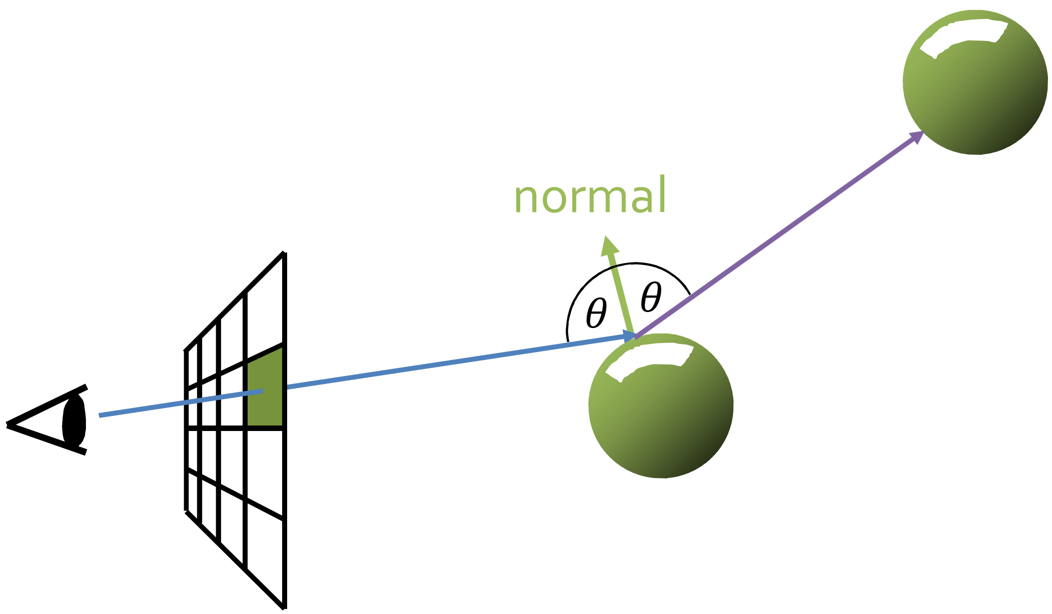
Types of Rays

- Primary rays
- Shadow rays



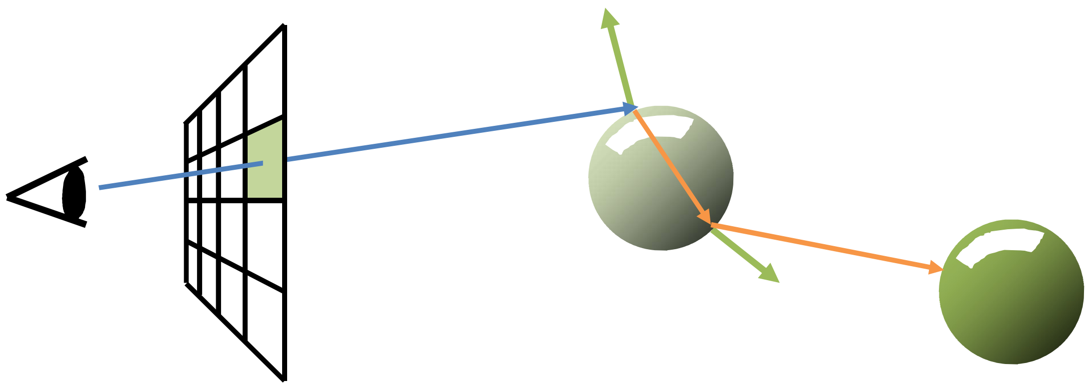
Types of Rays

- Primary rays
- Shadow rays
- Reflected rays



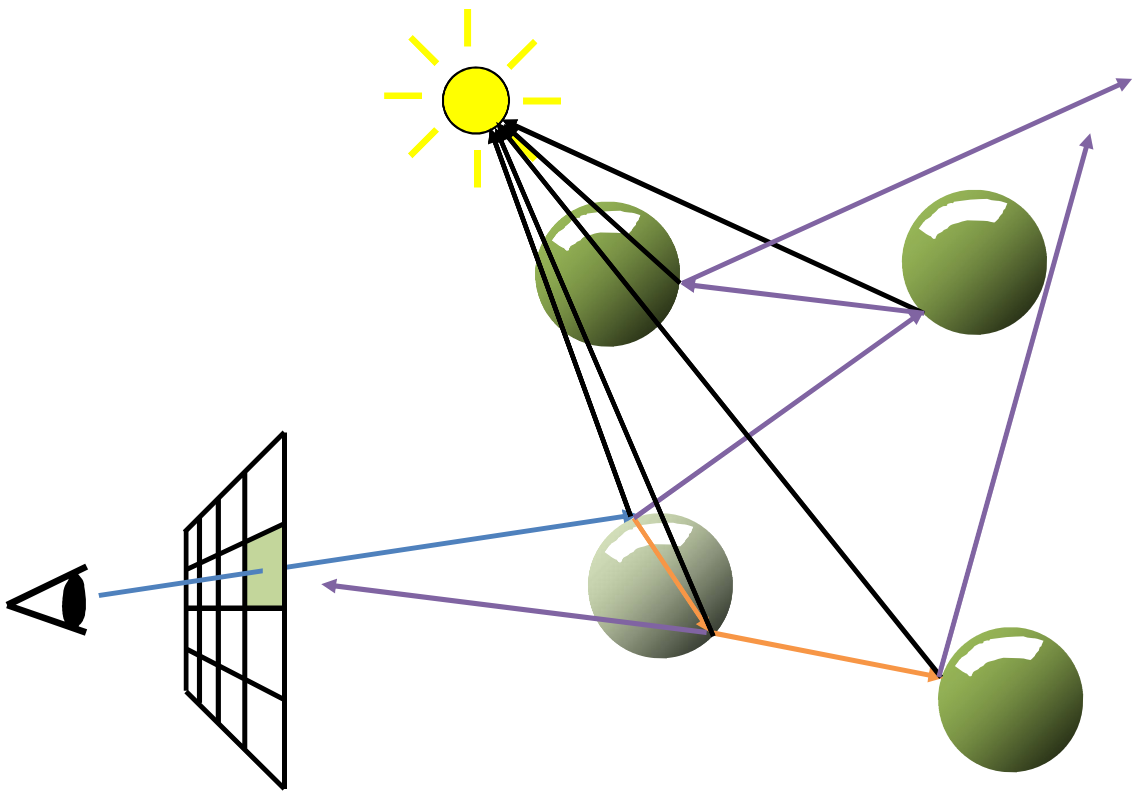
Types of Rays

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



Tree of Rays

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



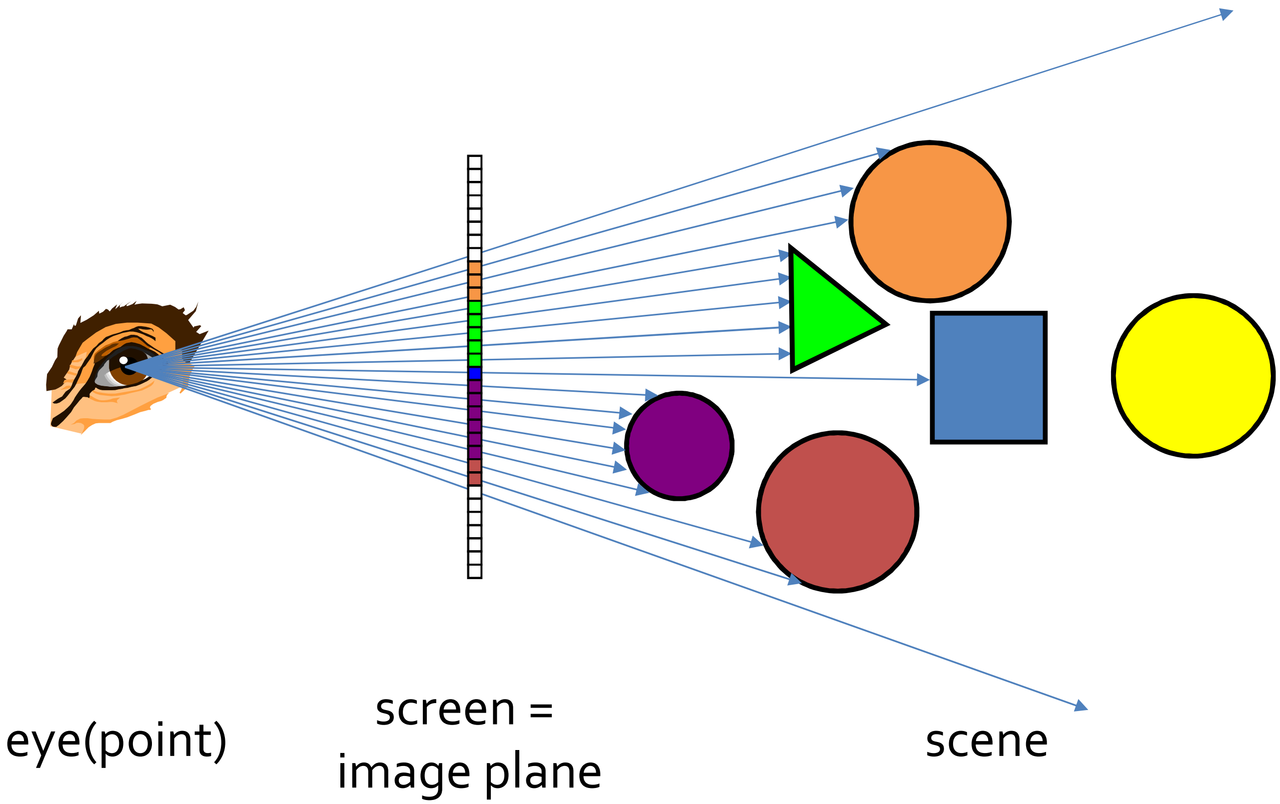
Ray-Tracer Code

```
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
  color = directLighting(ray, objectHit)
  color += trace(reflect(ray, objectHit))
  color += trace(refract(ray, objectHit))
  return color
}
```

Generating Camera Rays

Generating Camera Rays

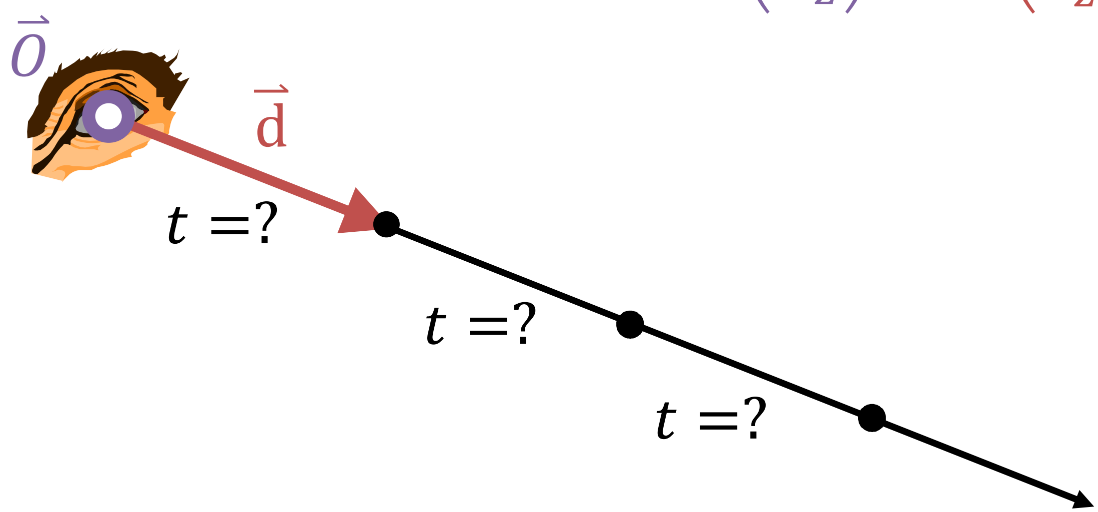
- Trace a ray for each pixel in the image plane



Ray Parametric Form

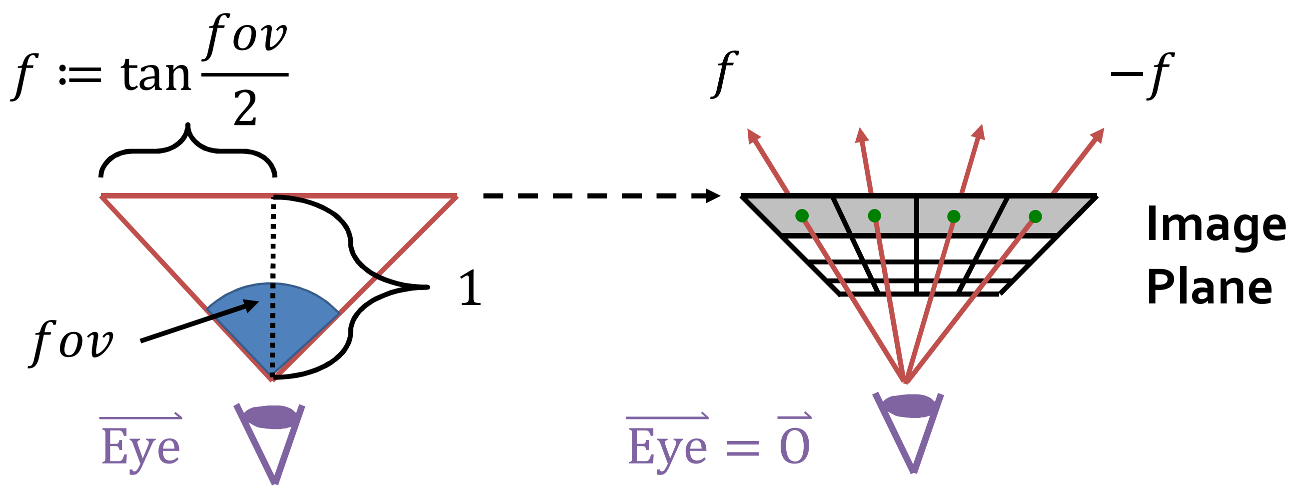
- Ray expressed as function of a single parameter t

$$ray(t) = \vec{\mathrm{o}} + t\vec{\mathrm{d}} = \begin{pmatrix} \mathrm{o}_x \\ \mathrm{o}_y \\ \mathrm{o}_z \end{pmatrix} + t \begin{pmatrix} \mathrm{d}_x \\ \mathrm{d}_y \\ \mathrm{d}_z \end{pmatrix}$$



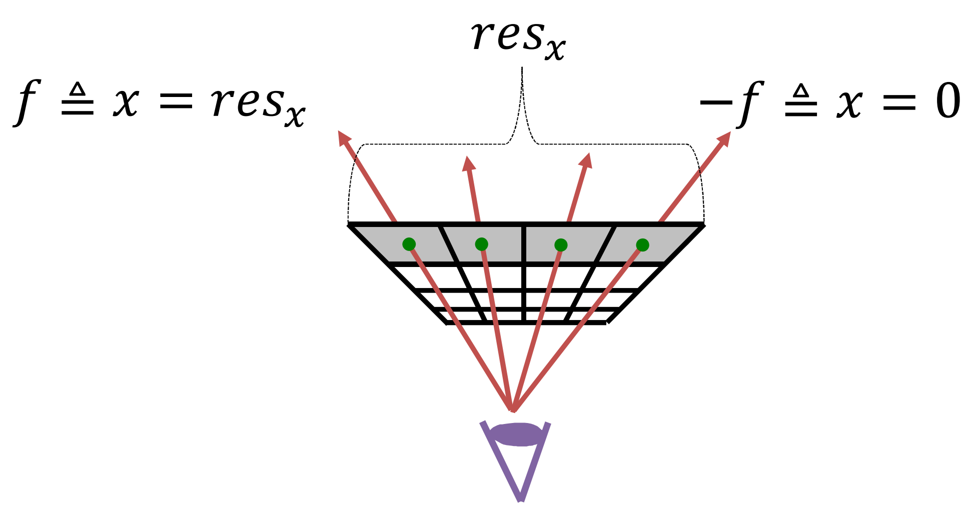
Generating Camera Rays – Top View

- 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
- $d_x(x) = \frac{x2f}{res_x} - f = (2x - res_x)\frac{f}{res_x} = (2x - res_x)f_x$ $\quad f_x := \frac{f}{res_x}$

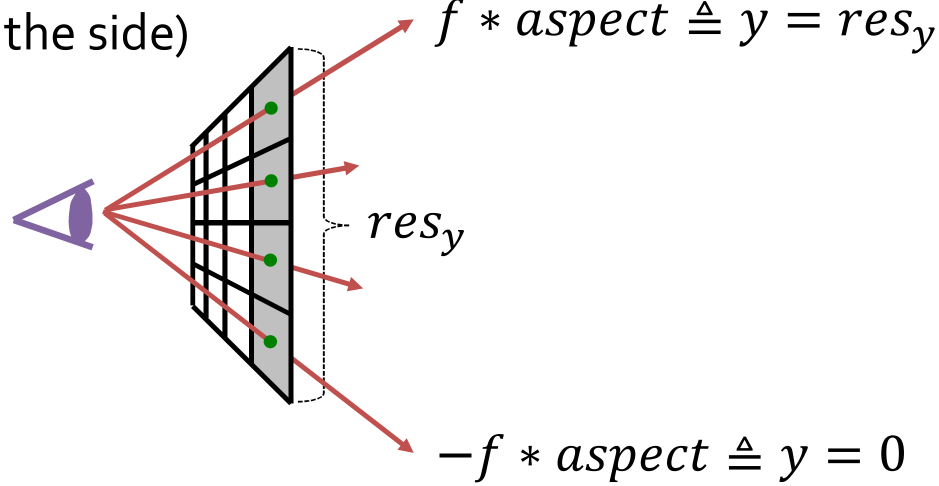


(Looking down from the top)

Generating Camera Rays – Side View

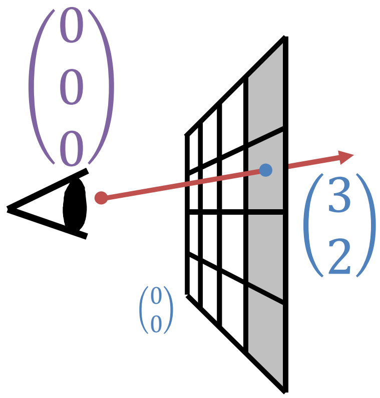
- Trace a ray for each pixel in the image plane
- $d_y(y) = aspect\left(\frac{y2f}{res_y} - f\right) = \frac{\mathrm{res_y}}{\mathrm{res_x}}\left(\frac{y2f}{res_y} - f\right) = (2y - res_y)f_x$

(Looking from the side)



Generating Camera 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 Camera Rays

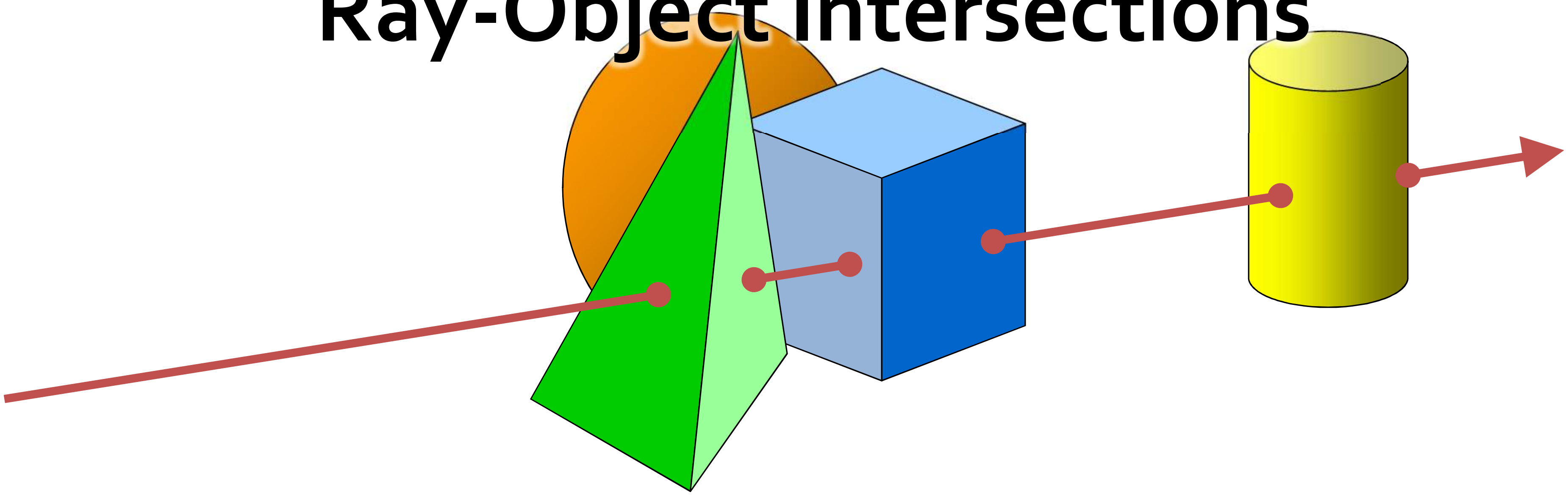
- Trace a ray for each pixel in the image plane

```
ray createCameraRay(x,y) {
  fov = 45°
  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

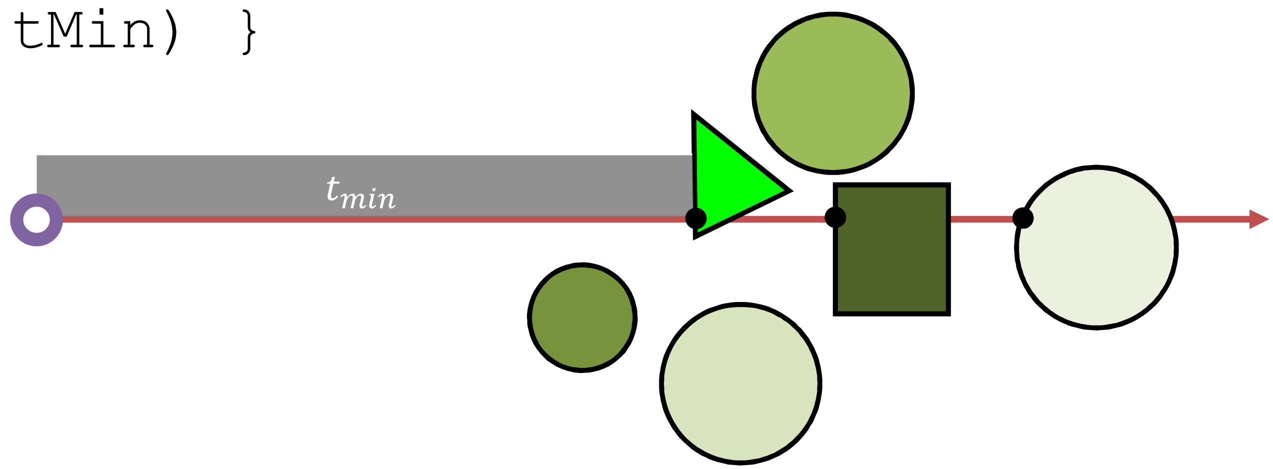
```
renderImage() {
  foreach pixel x,y in image {
    ray = createCameraRay(x,y))
    image[x][y] = trace(ray)
}}

color trace(ray) {
  objectHit = findNearestObjectHit(ray)
  if(objectHit == background) return bckGrndColor
  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) }
```

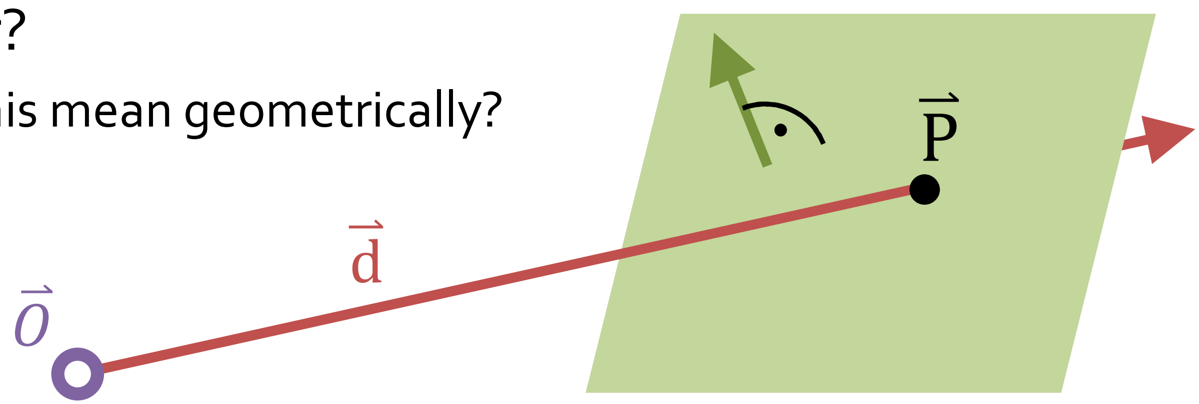


Ray-Object Intersection

- Find t at which the ray intersects the object
  - Given: ray equation: $ray(t) = \overrightarrow{\mathrm{O}} + t\overrightarrow{\mathrm{d}}$
  - Given: Implicit equation of object: $f\left(\overrightarrow{\mathrm{P}}\right) = 0$
    - Only points on the surface satisfy implicit equation
  - Solve $f\left((ray(t)\right) = 0$ for t

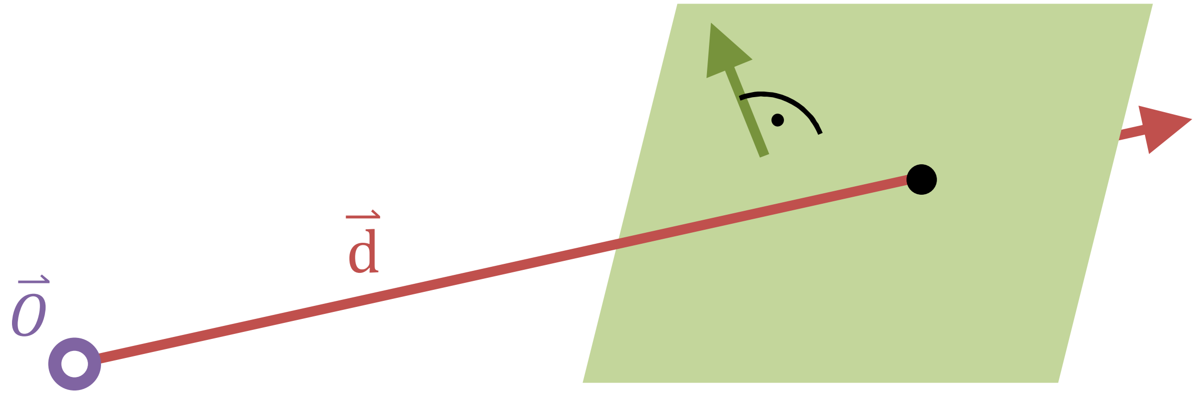
Ray-Plane Intersection

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



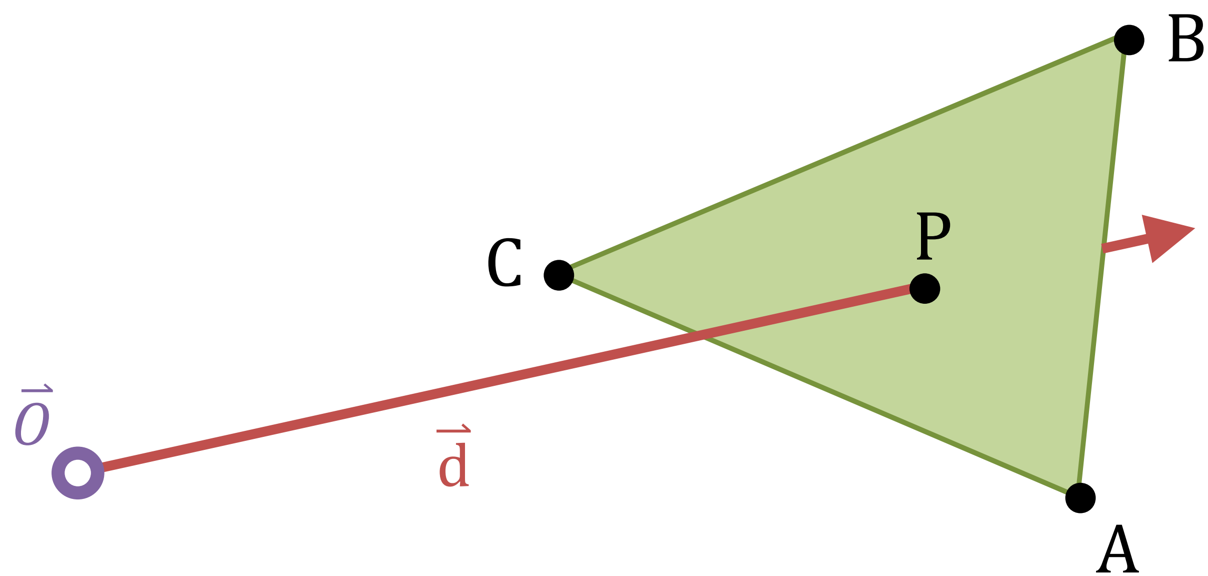
Ray-Plane Intersection – Code

```
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



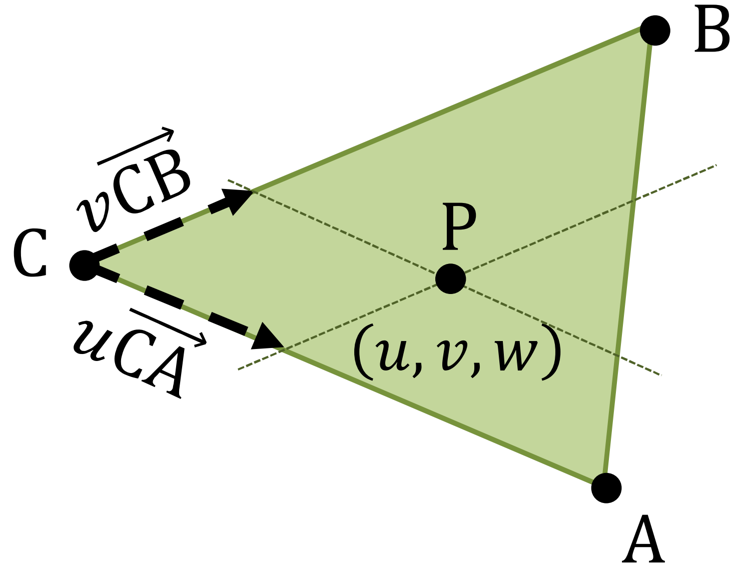
Barycentric Coordinates of P

- Define $\mathrm{P} = \mathrm{C} + u\overrightarrow{\mathrm{CA}} + v\overrightarrow{\mathrm{CB}}$

$$= u\mathrm{A} + v\mathrm{B} + (1 - u - v)\mathrm{C}$$

$= u\mathrm{A} + v\mathrm{B} + w\mathrm{C}$ 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 $(u, v, w) \geq 0$

1.2 -1.4 1.2	1.0 -1.2 1.2	0.8 -1.0 1.2	0.6 -0.8 1.2	0.4 -0.6 1.2	0.2 -0.4 1.2	0.0 -0.2 1.2	-0.2 0.0 1.2
1.2 -1.2 1.0	1.0 -1.0 1.0	0.8 -0.8 1.0	0.6 -0.6 1.0	0.4 -0.4 1.0	0.2 -0.2 1.0	0.0 0.0 1.0	-0.2 0.2 1.0
1.2 -1.0 0.8	1.0 -0.8 0.8	0.8 -0.6 0.8	0.6 -0.4 0.8	0.4 -0.2 0.8	0.2 0.0 0.8	0.0 0.2 0.8	-0.2 0.4 0.8
1.2 -0.8 0.6	1.0 -0.6 0.6	0.8 -0.4 0.6	0.6 -0.2 0.6	0.4 0.0 0.6	0.2 0.2 0.6	0.0 0.4 0.6	-0.2 0.6 0.6
1.2 -0.6 0.4	1.0 -0.4 0.4	0.8 -0.2 0.4	0.6 0.0 0.4	0.4 0.2 0.4	0.2 0.4 0.4	0.0 0.6 0.4	-0.2 0.8 0.4
1.2 -0.4 0.2	1.0 -0.2 0.2	0.8 0.0 0.2	0.6 0.2 0.2	0.4 0.4 0.2	0.2 0.6 0.2	0.0 0.8 0.2	-0.2 1.0 0.2
1.2 -0.2 0.0	1.0 0.0 0.0	0.8 0.2 0.0	0.6 0.4 0.0	0.4 0.6 0.0	0.2 0.8 0.0	0.0 1.0 0.0	-0.2 1.2 0.0
1.2 0.0 -0.2	1.0 0.2 -0.2	0.8 0.4 -0.2	0.6 0.6 -0.2	0.4 0.8 -0.2	0.2 1.0 -0.2	0.0 1.2 -0.2	-0.2 1.4 -0.2

Ray Triangle Intersection

- $ray(t) = \vec{\mathrm{O}} + t\vec{\mathrm{d}}$

- Point on triangle (Barycentric coordinates)

$$triangle(u, v) = u\vec{\mathrm{A}} + v\vec{\mathrm{B}} + (1 - u - v)\vec{\mathrm{C}}$$

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

Ray Triangle Intersection

- Intersection $\vec{\mathrm{O}} + t\vec{\mathrm{d}} = u\vec{\mathrm{A}} + v\vec{\mathrm{B}} + (1-u-v)\vec{\mathrm{C}}$
- Rearranged $\vec{\mathrm{O}} - \vec{\mathrm{C}} = \begin{pmatrix} -\vec{\mathrm{d}} & \vec{\mathrm{A}} - \vec{\mathrm{C}} & \vec{\mathrm{B}} - \vec{\mathrm{C}} \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix}$
- Linear system!
- Solve with Cramer's rule

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

Ray Triangle Intersection: Implementation

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

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

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

Ray Triangle Intersection: Implementation

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

$$\mathrm{E}_1 = \vec{\mathrm{A}} - \vec{\mathrm{C}} \qquad \mathrm{E}_2 = \vec{\mathrm{B}} - \vec{\mathrm{C}} \qquad \mathrm{S} = \vec{\mathrm{O}} - \vec{\mathrm{C}}$$

$$\mathrm{P} = \left(\vec{\mathrm{d}} \times \mathrm{E}_2\right) \qquad \mathrm{Q} = \mathrm{S} \times \mathrm{E}_1$$

- Substituting gives $\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\mathrm{P} \cdot \mathrm{E}_1} \begin{pmatrix} \mathrm{Q} \cdot \mathrm{E}_2 \\ \mathrm{P} \cdot \mathrm{S} \\ \mathrm{Q} \cdot \mathrm{D} \end{pmatrix}$

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            0 == 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
```

vectors

scalars

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

1.2 -1.4 1.2	1.0 -1.2 1.2	0.8 -1.0 1.2	0.6 -0.8 1.2	0.4 -0.6 1.2	0.2 -0.4 1.2	0.0 -0.2 1.2	-0.2 0.0 1.2
1.2 -1.2 1.0	1.0 -1.0 1.0	0.8 -0.8 1.0	0.6 -0.6 1.0	0.4 -0.4 1.0	0.2 -0.2 1.0	0.0 0.0 1.0	-0.2 0.2 1.0
1.2 -1.0 0.8	1.0 -0.8 0.8	0.8 -0.6 0.8	0.6 -0.4 0.8	0.4 -0.2 0.8	0.2 0.0 0.8	0.0 0.2 0.8	-0.2 0.4 0.8
1.2 -0.8 0.6	1.0 -0.6 0.6	0.8 -0.4 0.6	0.6 -0.2 0.6	0.4 0.0 0.6	0.2 0.2 0.6	0.0 0.4 0.6	-0.2 0.6 0.6
1.2 -0.6 0.4	1.0 -0.4 0.4	0.8 -0.2 0.4	0.6 0.0 0.4	0.4 0.2 0.4	0.2 0.4 0.4	0.0 0.6 0.4	-0.2 0.8 0.4
1.2 -0.4 0.2	1.0 -0.2 0.2	0.8 0.0 0.2	0.6 0.2 0.2	0.4 0.4 0.2	0.2 0.6 0.2	0.0 0.8 0.2	-0.2 1.0 0.2
1.2 -0.2 0.0	1.0 0.0 0.0	0.8 0.2 0.0	0.6 0.4 0.0	0.4 0.6 0.0	0.2 0.8 0.0	0.0 1.0 0.0	-0.2 1.2 0.0
1.2 0.0 -0.2	1.0 0.2 -0.2	0.8 0.4 -0.2	0.6 0.6 -0.2	0.4 0.8 -0.2	0.2 1.0 -0.2	0.0 1.2 -0.2	-0.2 1.4 -0.2

Ray-Sphere Intersection

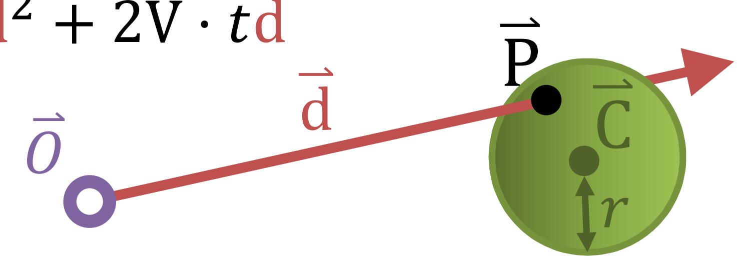
- $ray(t) = \vec{\mathrm{O}} + t\vec{\mathrm{d}}$
- Implicit equation of sphere with center $\vec{\mathrm{C}}$ and radius r:

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

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

$$\vec{\mathrm{V}} \coloneqq \vec{\mathrm{O}} - \vec{\mathrm{C}} \qquad \left|\vec{\mathrm{A}} + \vec{\mathrm{B}}\right|^2 = \vec{\mathrm{A}}^2 + \vec{\mathrm{B}}^2 + 2\left(\vec{\mathrm{A}} \cdot \vec{\mathrm{B}}\right)$$

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



Ray-Sphere Intersection

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

$$\overset{\vec{\mathrm{d}}^2=1}{\Longleftrightarrow} t^2 + \left(2\vec{\mathrm{V}} \cdot \vec{\mathrm{d}}\right) t + \left(\vec{\mathrm{V}}^2 - r^2\right) = 0$$

$$t = -\left(\vec{\mathrm{V}} \cdot \vec{\mathrm{d}}\right) \pm \sqrt{\left(\vec{\mathrm{V}} \cdot \vec{\mathrm{d}}\right)^2 - \left(\vec{\mathrm{V}}^2 - r^2\right)}$$

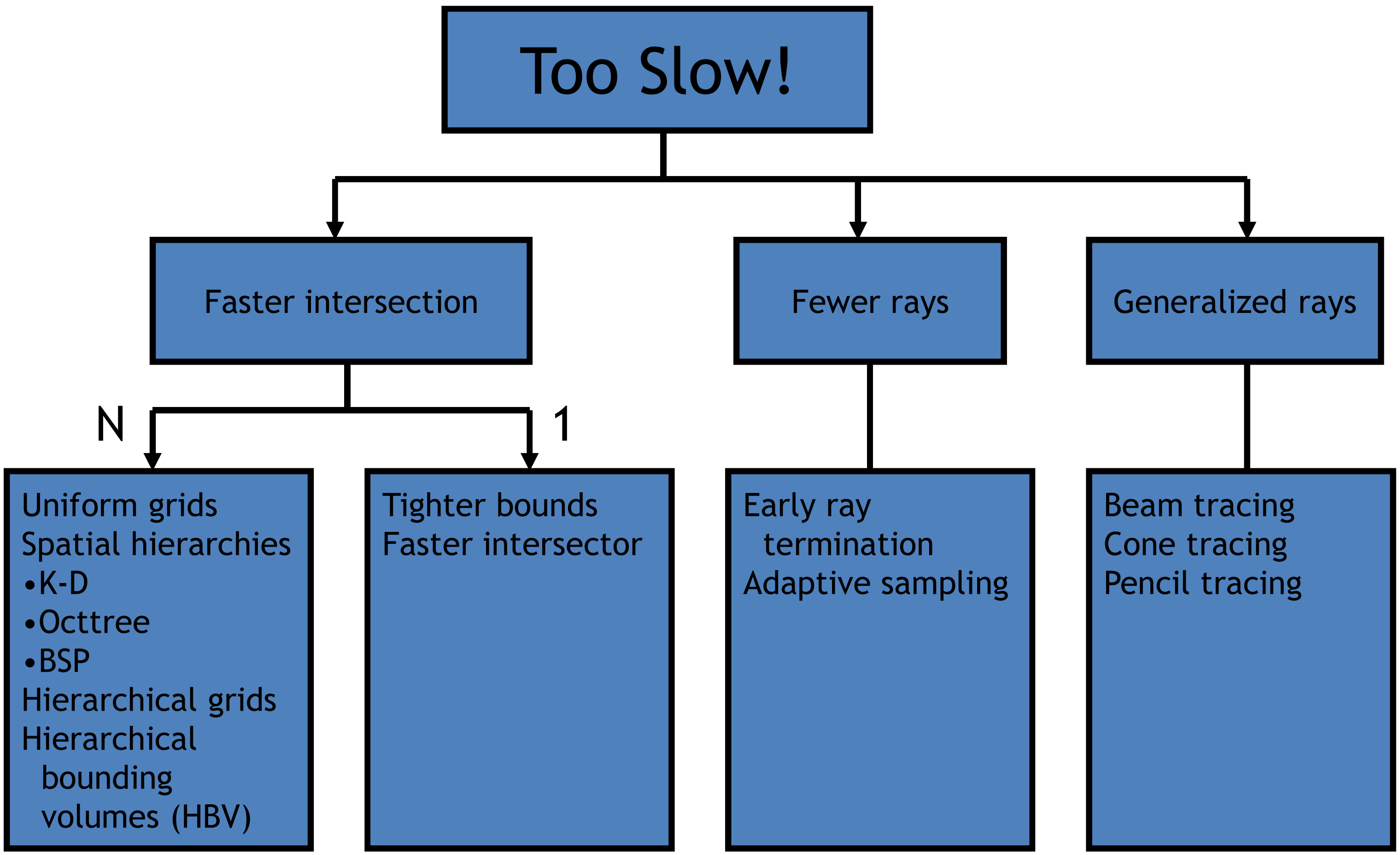
- Real solutions, indicates one or two intersections
- Negative solutions are behind the eye
- If discriminant is negative, the ray missed the sphere

Acceleration

- 1280x1024 image with 10 rays/pixel
- 1000 objects (triangle)
- 3 levels recursion

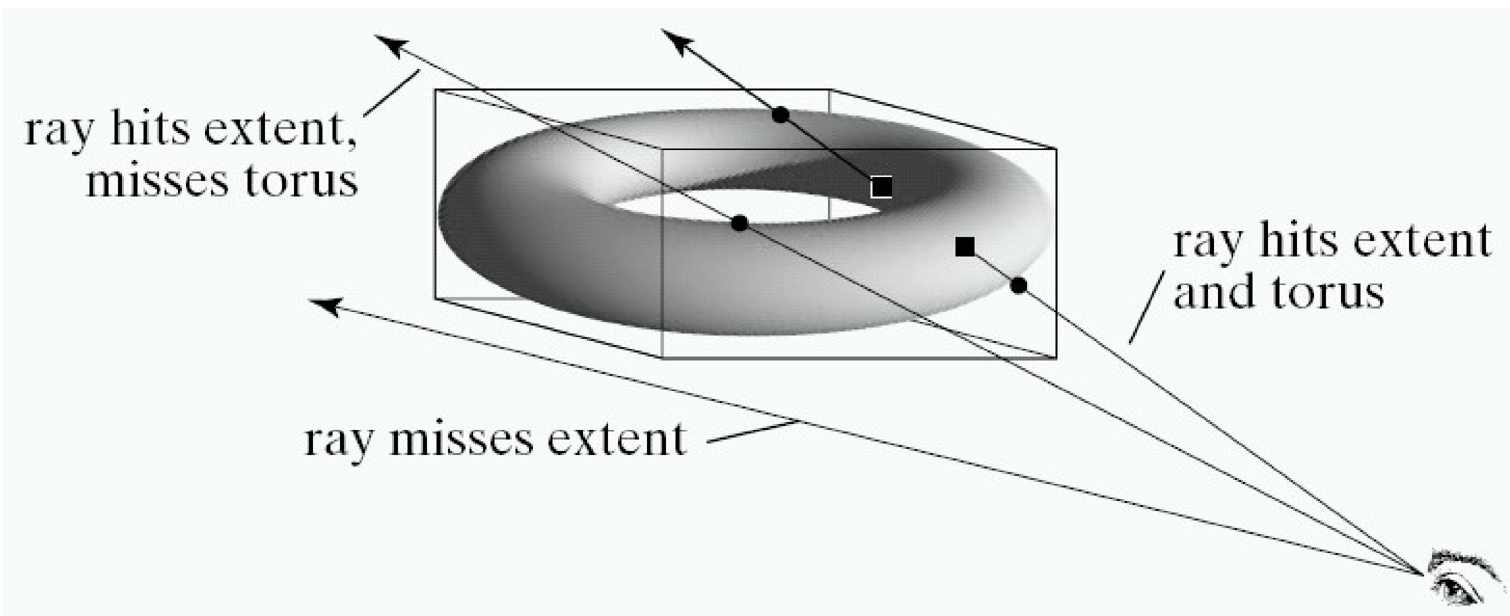
- 39321600000 intersection tests
- 100000 tests/second -> 109 days!
- Must use an acceleration method!

Ray Tracing Acceleration Techniques

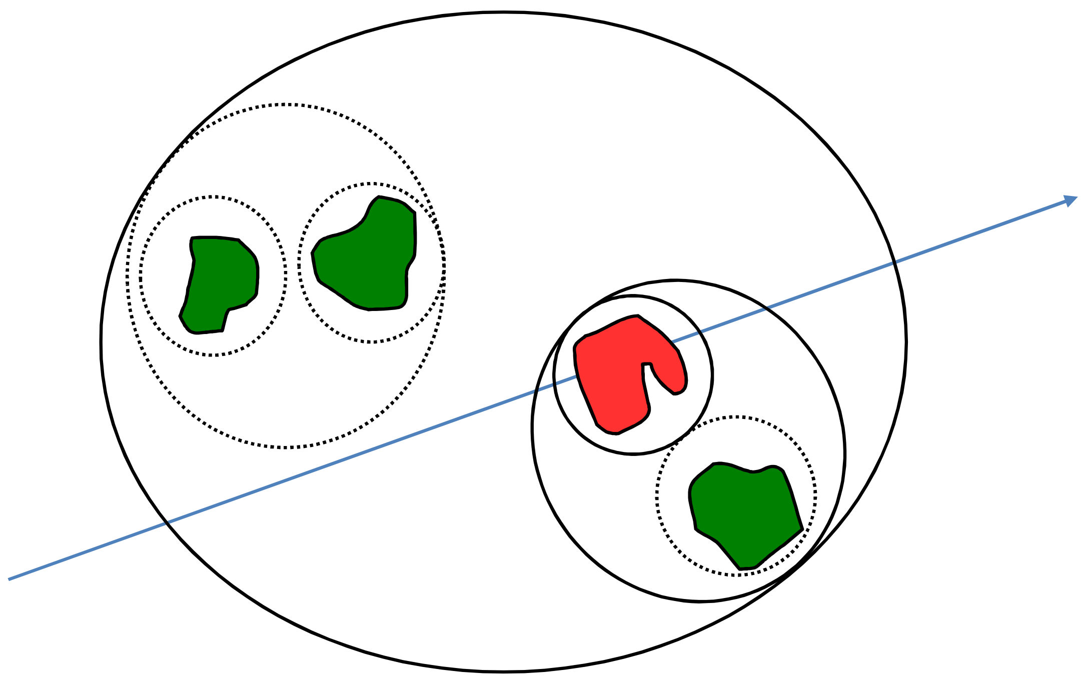


Bounding volumes

- Use simple shape for quick test

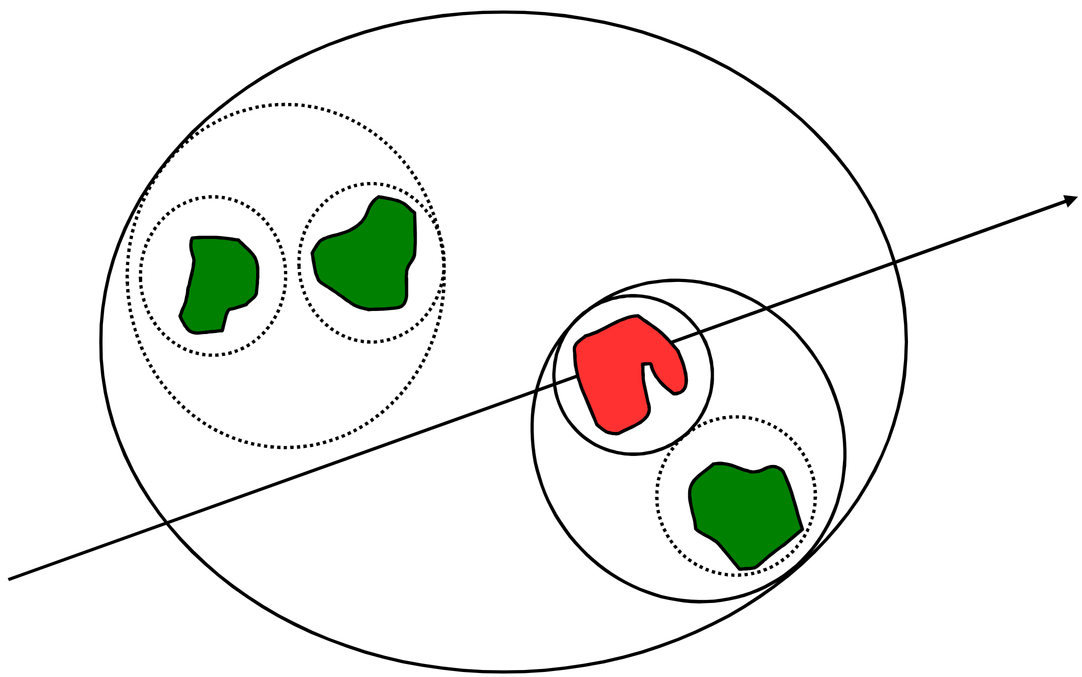


Acceleration structures for ray-tracing



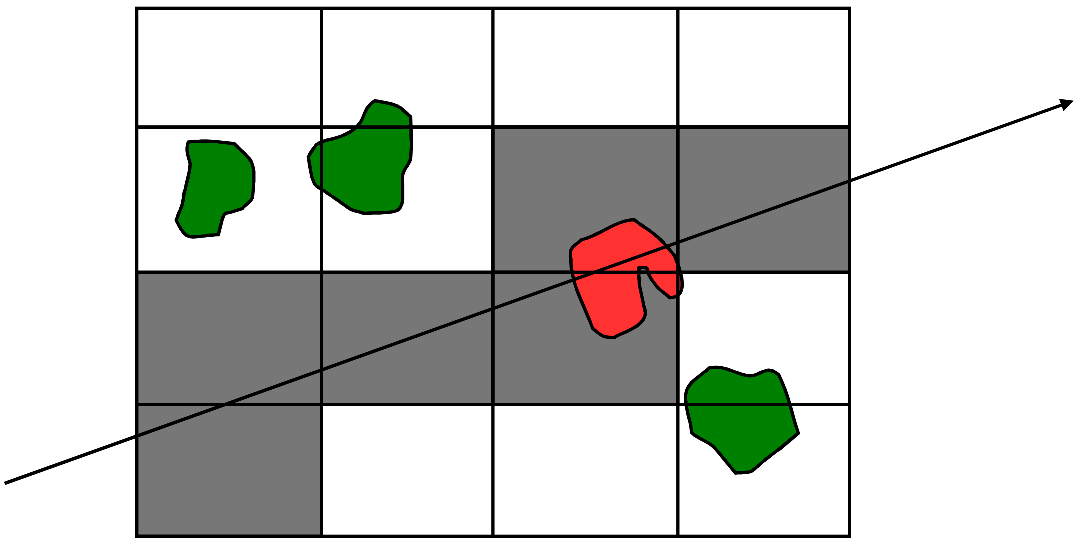
Bounding volumes

- Use simple shape for quick test
- Keep a hierarchy

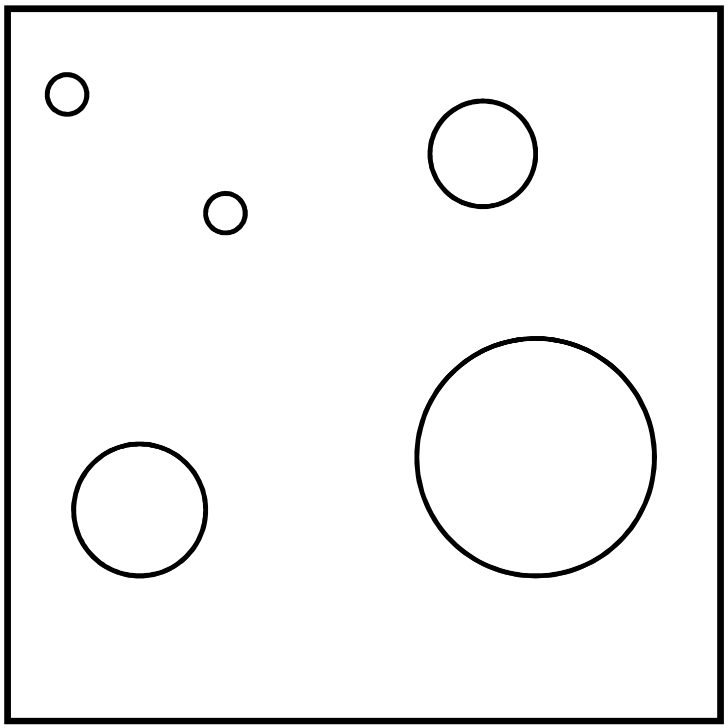


Space Subdivision

- Break your space into pieces
- Search the structure linearly



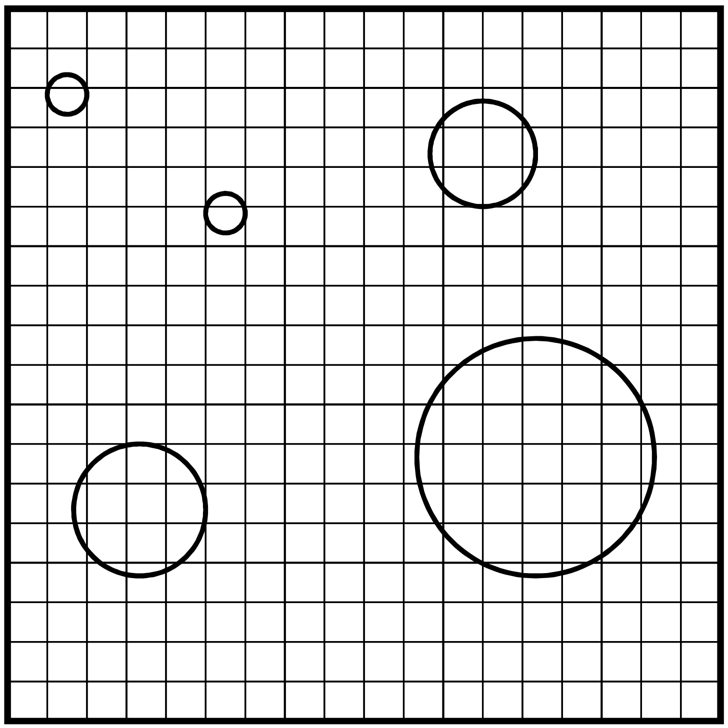
Uniform Grids



- Preprocess scene

1. Find bounding box

Uniform Grids

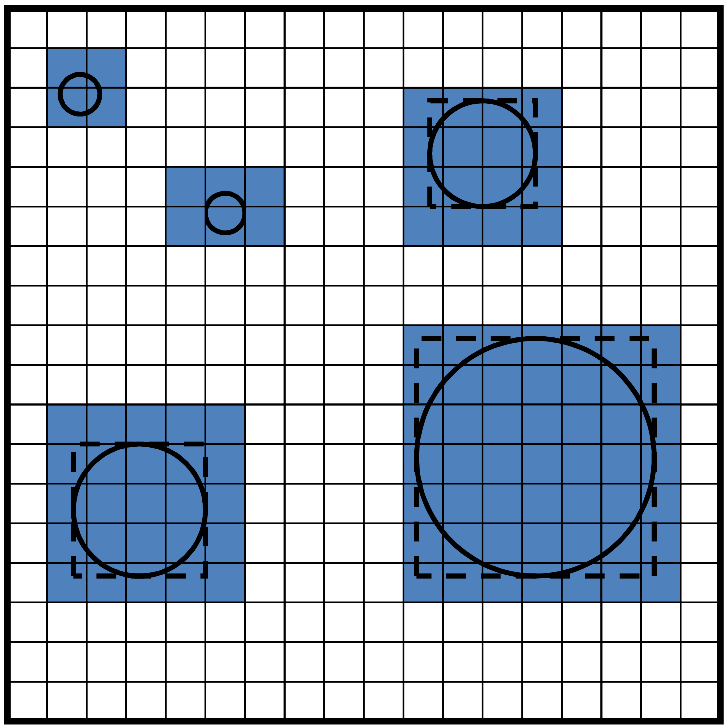


- Preprocess scene

1. Find bounding box
2. Determine grid resolution

$$n^3 = d\,|O|$$

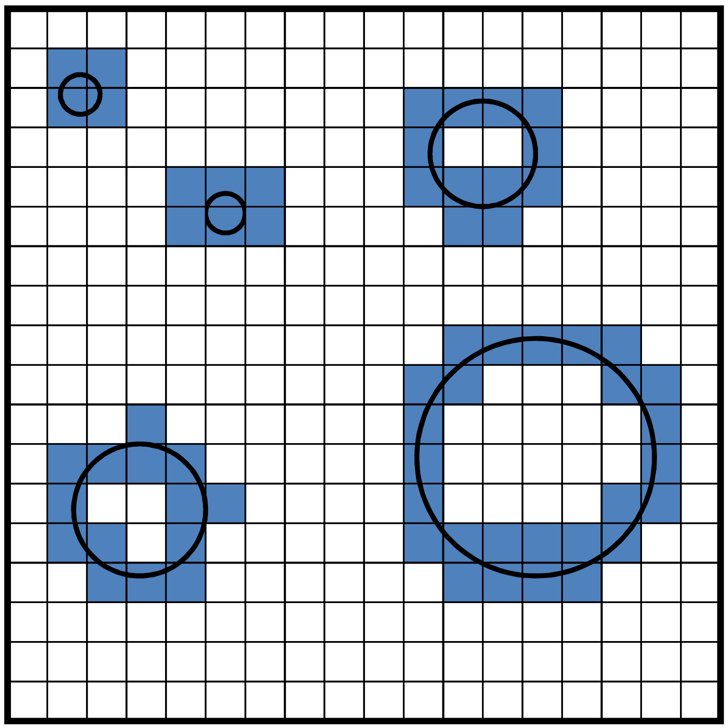
Uniform Grids



- Preprocess scene

1. Find bounding box
2. Determine grid resolution $n^3 = d\,|O|$
3. Place object in cell if its bounding box overlaps the cell

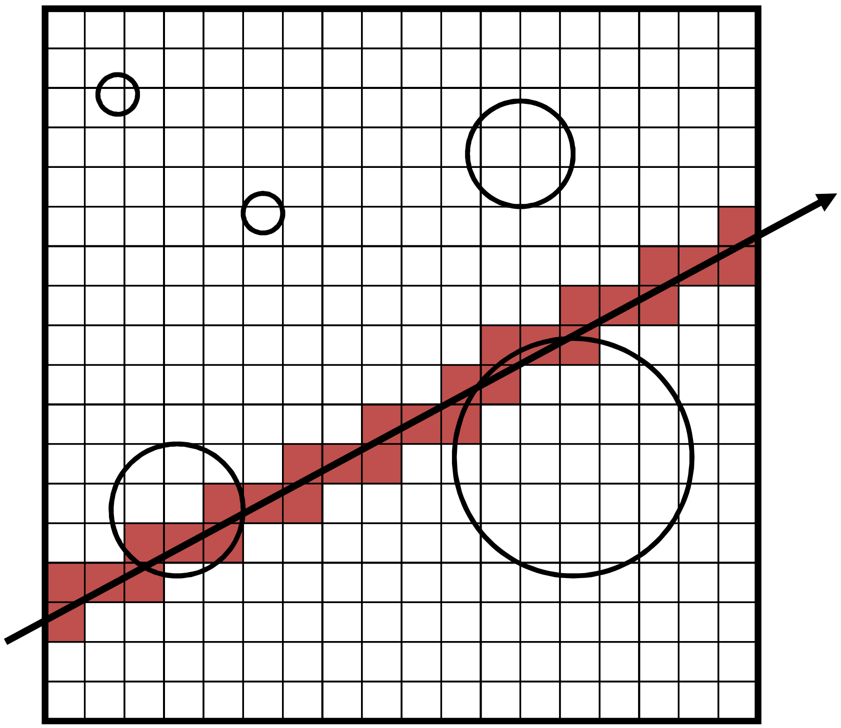
Uniform Grids



- Preprocess scene

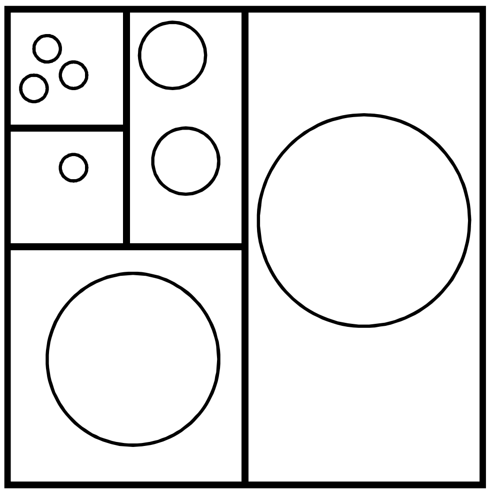
1. Find bounding box
2. Determine grid resolution $n^3 = d\,|O|$
3. Place object in cell if its bounding box overlaps the cell
4. Check that object overlaps cell (expensive!)

Uniform Grids

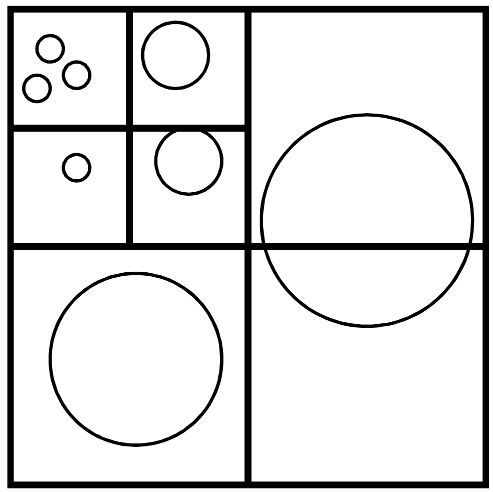


- Preprocess scene
- Traverse grid
  - 3D line = 3D-DDA
  - 6-connected line

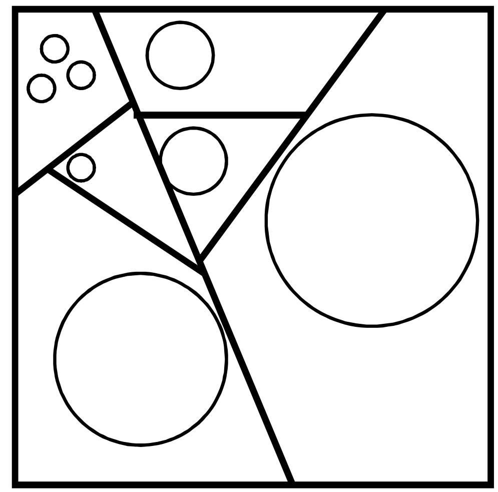
Variations



KD tree



octtree



BSP tree