CTP 499
Computer Graphics for CT

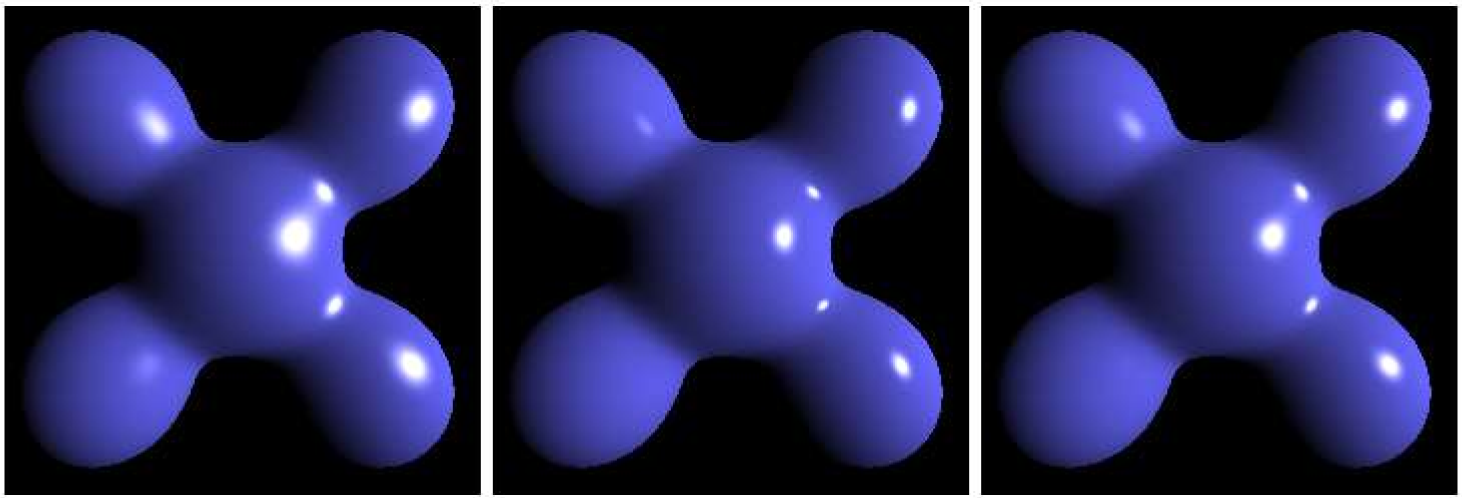
Ray Tracing

October 29, 2025

Professor Junyong Noh

Visual Media Lab KAIST

Image Comparison



Blinn-Phong | Phong | Blinn-Phong (Lower Exponent)

-> Higher

Global Approaches

- Global illumination is hard to solve
  - Multiple reflections of rays with multiple objects
- With a pipeline model, primitives are rendered one at a time
  - No multiple reflections
- Global approaches
  - Ray tracing
  - Radiosity

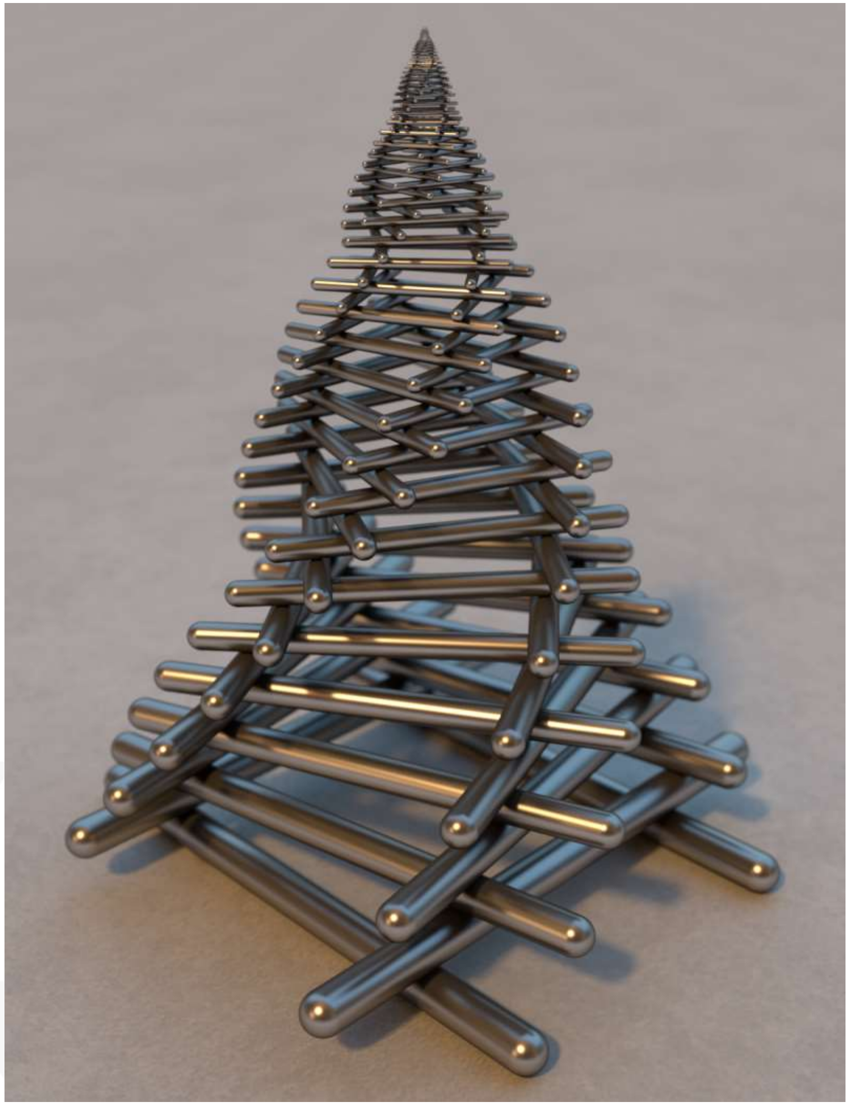
Ray Tracing

- Trace the path of a ray of light
- Model its interactions with the scene
- When a ray intersects an object, send off secondary rays (reflection, transmission, shadow) and determine how they interact with the scene

Ray Tracing

- Produces realistic images
- Strengths
  - Improved realism
  - Specular reflections
  - Transparency
  - Shadow
  - Hidden surface removal
  - Very simple design
- Weaknesses
  - Only approximate global illumination (cannot follow all rays)
  - Color bleeding (diffuse reflections)
  - Very slow per pixel calculations
  - Hard to accelerate with hardware
  - Aliasing

Example

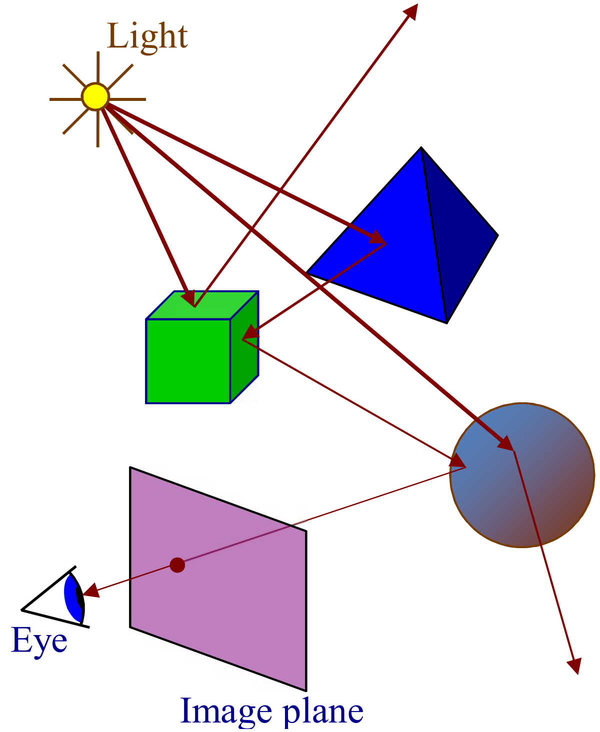


Example

Example

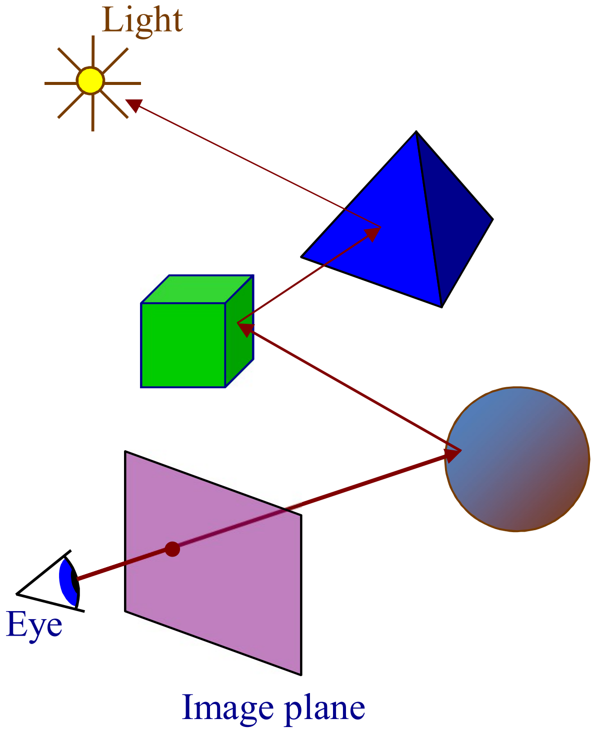
Ray Tracing

- "Backward" ray tracing
  - Traces the ray forward (in time) from the light source through potentially many scene interactions
  - Problem: most rays will never even get close to the eye
  - Very inefficient since it computes many rays that are never seen



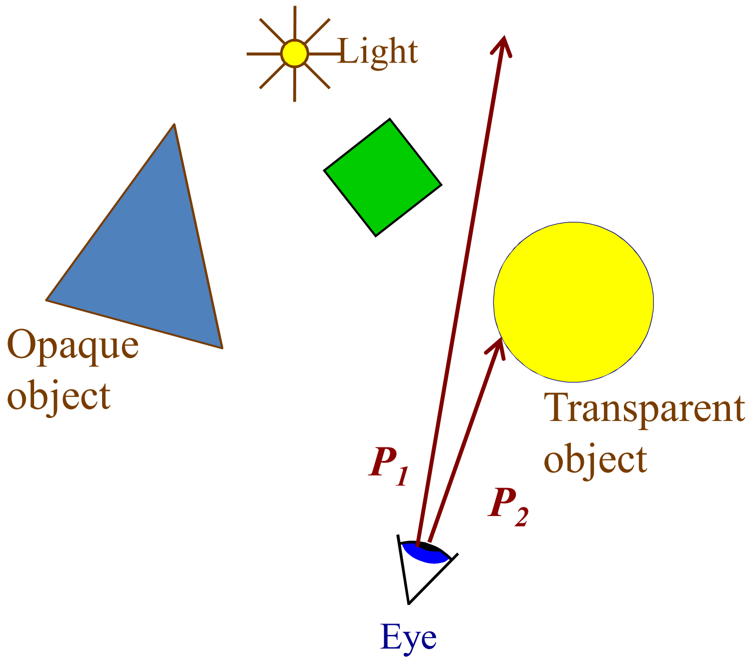
Ray Tracing

- "Forward" ray tracing
  - Traces the ray backward (in time) from the eye, through a point on the screen
  - More efficient: computes only visible rays (since we start at eye)
  - Generally, ray tracing refers to forward ray tracing.



Ray Tracing: Types of Rays

- Primary rays
  - Sent from the eye, through the image plane, and into the scene
  - May or may not intersect an object in the scene.
    - No intersection: set pixel to background color
    - Intersects object: send out secondary rays and compute lighting model



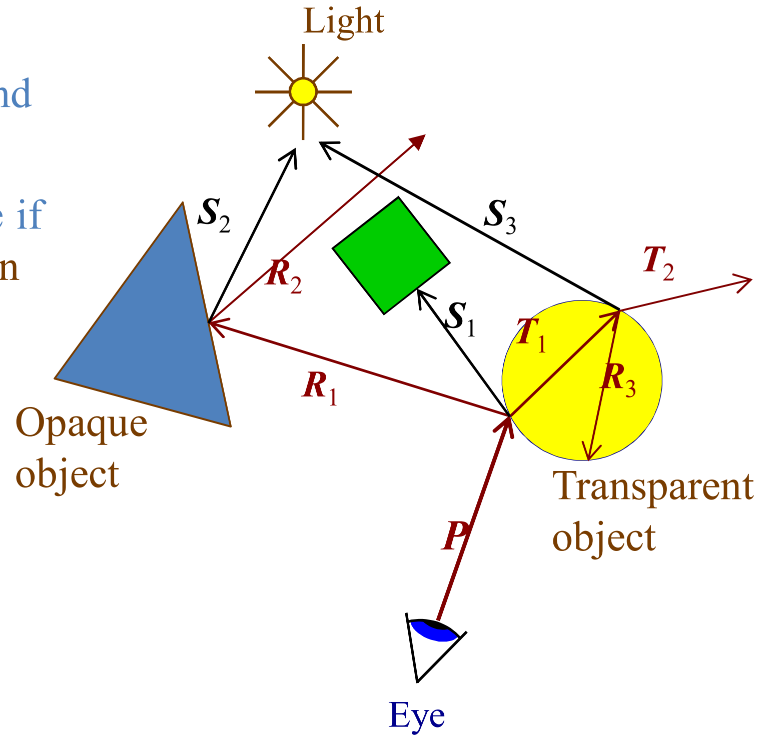
Ray Tracing: Types of Rays

- Secondary Rays
  - Sent from the point at which the ray intersects an object
- Multiple types

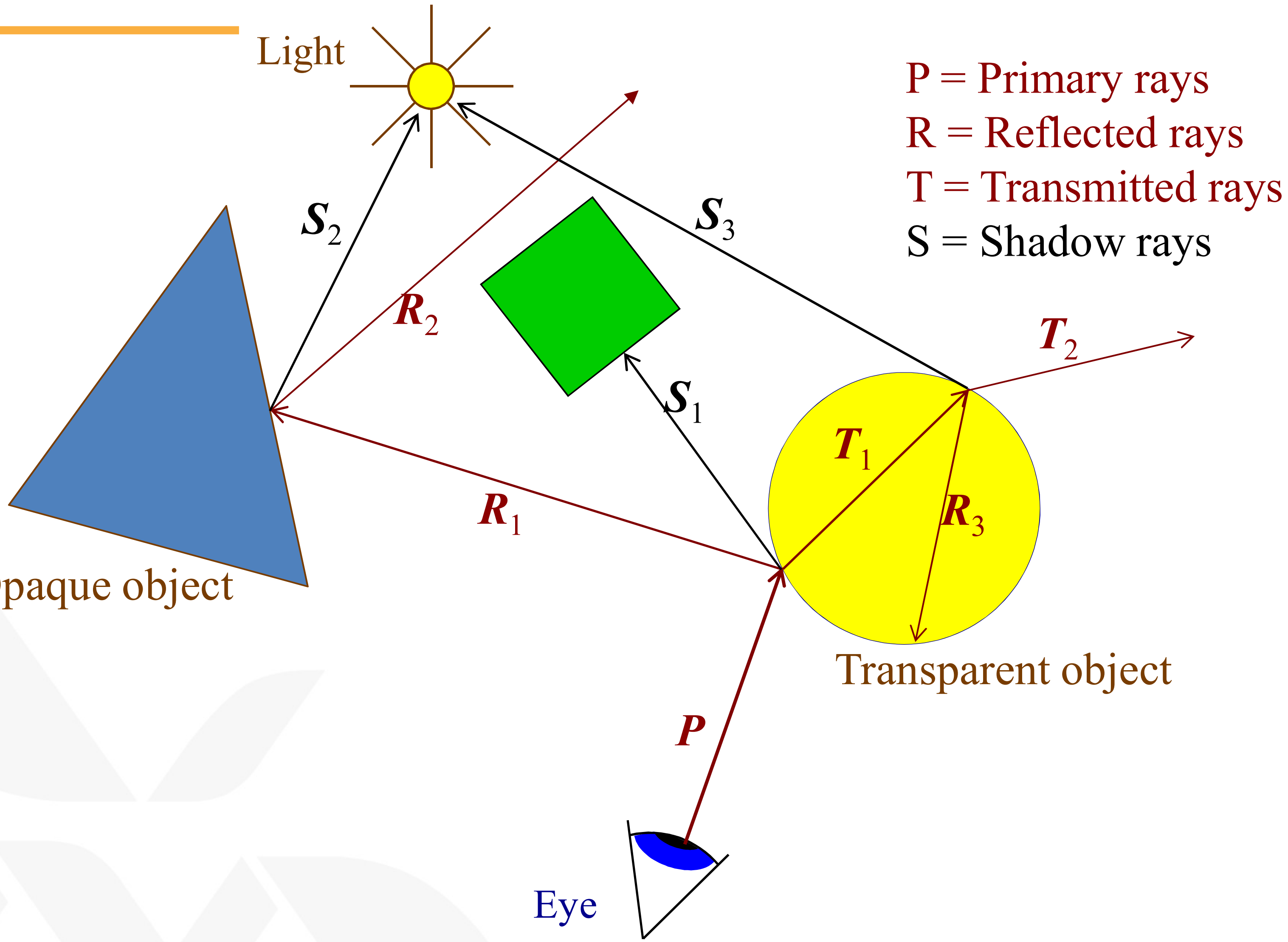
Transmission (T): sent in the direction of refraction

Reflection (R): sent in the direction of reflection, and used in the Phong illumination model

Shadow (S): sent toward a light source to determine if point is in shadow or not. Shadow rays do not spawn additional rays.



Ray Tracing: Types of Rays

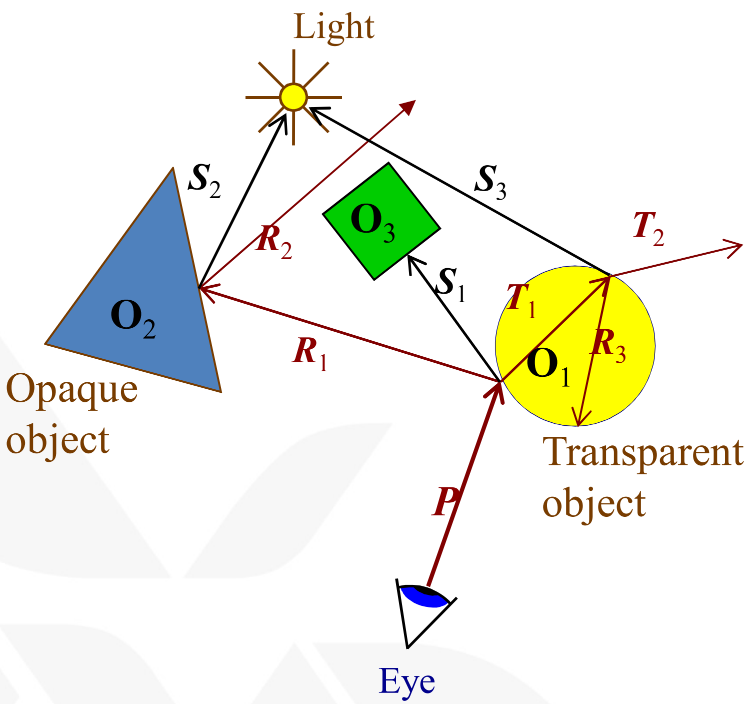


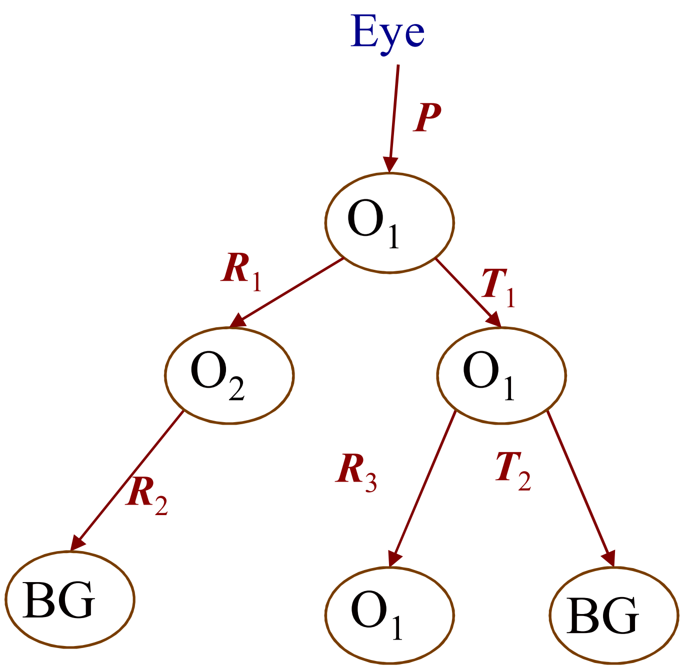
Ray Tracing: Ray Tree

- Each intersection may spawn secondary rays
  - Rays form a ray tree.
  - Nodes are the intersection points.
  - Edges are the primary or secondary rays.
- Rays are recursively spawned until
  - Ray does not intersect any object.
  - Tree reaches a maximum depth.
  - Light reaches some minimum value.

Ray Tracing: Ray Tree Example

- Ray tree is evaluated from bottom up
  - Depth-first traversal
  - The node color is computed based on its children's colors.





Basic Ray Tracing Algorithm

- Generate one ray per pixel
- For each ray
  - Find the first object the ray intersects
  - Compute color for the intersection point using an illumination model
  - If the surface is reflective, trace a reflection ray
  - If the surface is transparent, trace a transmission ray
  - Trace shadow ray
  - Combine results of the intensity computation, reflection, transmission, and shadow information
  - If the ray misses all objects, set to the background color

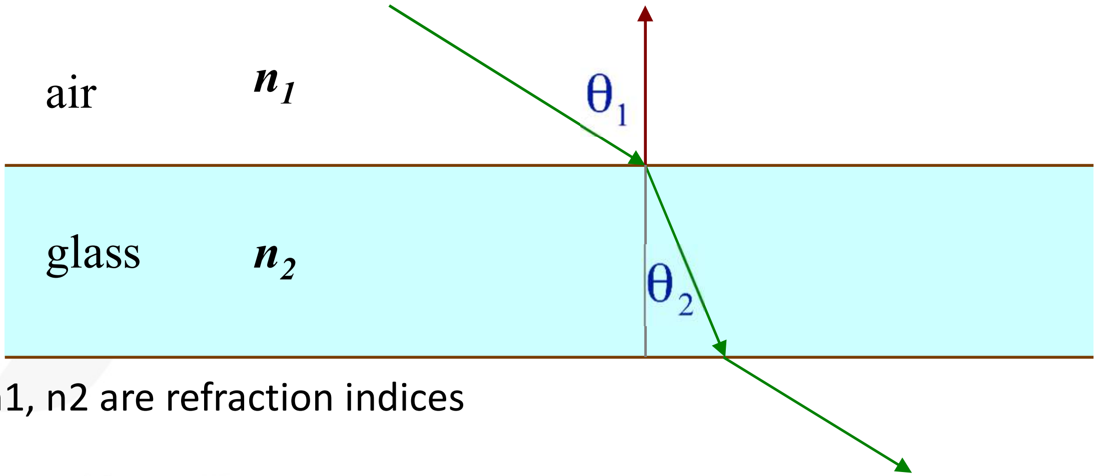
Recursive Ray Tracer

```
color c = trace(point p, vector d, int step) {
        color local, reflected, transmitted;
        point q;    // intersection
        normal n;
        if(step > max) return(background_color);

        q = intersect(p, d, status);
        if(status==light_source) return(light_source_color);
        if(status==no_intersection) return(background_color);

        n = normal(q);
        r = reflect(q, n);
        t = transmit(q, n);

        local = phong(q, n, r);
        reflected = trace(q, r, step+1);
        transmitted = trace(q, t, step+1);

        return(local+reflected+transmitted);
}
```

Refraction (Transparency)

- Light can transmit through transparent objects
- Light bends when moving from one medium to another according to Snell's law

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$



n1, n2 are refraction indices

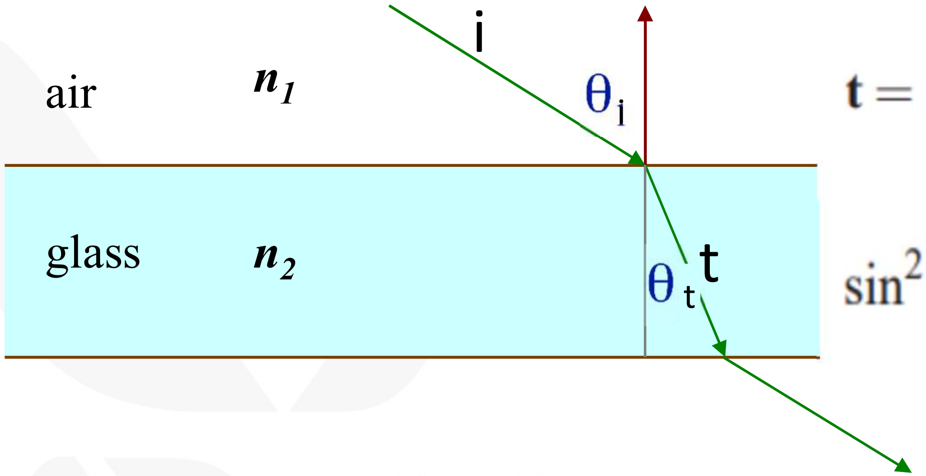
Refraction Indices

Material	Index
Vacuum	1.0
Air	1.0003
Water	1.33
Alcohol	1.36
Fused quartz	1.46
Crown glass	1.52
Flint glass	1.65
Sapphire	1.77
Heavy flint glass	1.89
Diamond	2.42

Refraction (Transparency)

- Light transmits through transparent objects
- Light bends when moving from one medium to another according to Snell's law

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{n_2}{n_1}$$



$$\mathbf{t} = \frac{\eta_1}{\eta_2}\mathbf{i} + \left(\frac{\eta_1}{\eta_2}\cos\theta_i - \sqrt{1-\sin^2\theta_t}\right)\mathbf{n}$$

$$\sin^2\theta_t = \left(\frac{\eta_1}{\eta_2}\right)^2 \sin^2\theta_i = \left(\frac{\eta_1}{\eta_2}\right)^2 \left(1-\cos^2\theta_i\right)$$

https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf

Minimal Ray Tracer

- A basic (minimal) ray tracer is simple to implement
  - The code can even fit on a 3×5 card (code courtesy of Paul Heckbert)

```c
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(){puts("P3\n32 32\n255");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);}/*minray!*/
```

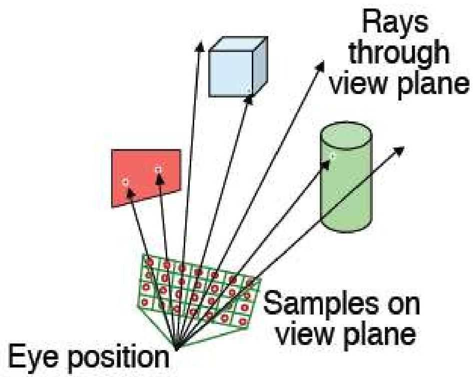
Minimal Ray Tracer

- This code implements
  - Multiple spheres (with different properties)
  - Multiple levels of recursion
    - Reflections
  - Transparency
    - Refraction
  - One point light source
    - Hard shadows
  - Hidden surface removal
  - Phong illumination model

```
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(){puts("P3\n32 32\n255");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);}/*minray!*/
```

Things to Consider

- Ray casting (non-recursive) algorithm
  - Send a ray from the eye through the screen
  - Determine which object that ray first intersects
  - Compute pixel color
- Most (approx. 75%) of the time in step 2
  - Simple method
    - Compare every ray against every object and determine the closest object hit by each ray
  - Very time consuming
    - Several optimizations possible

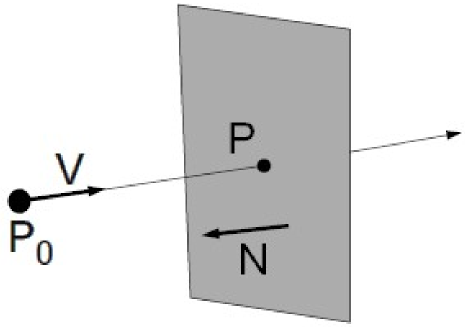


Computing Intersections

- Planes
- Spheres
- Polyhedra

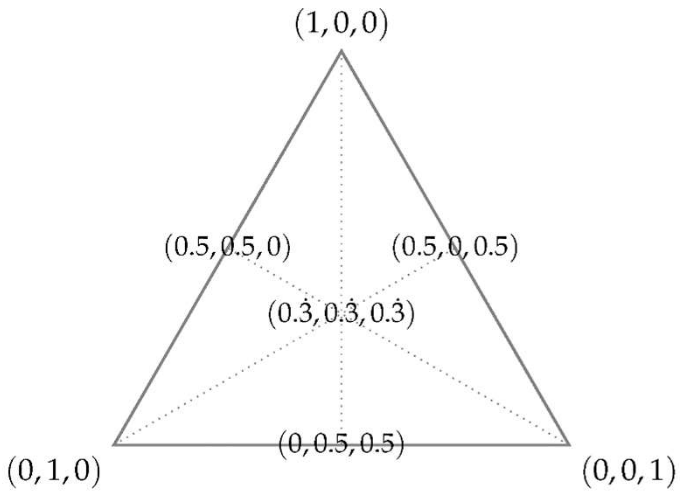
Ray-Plane/Polygon Intersection

- Plane-line intersection ray:
  - Line: $\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{V}$
  - Plane: $\mathbf{p} \bullet \mathbf{n} + c = 0$



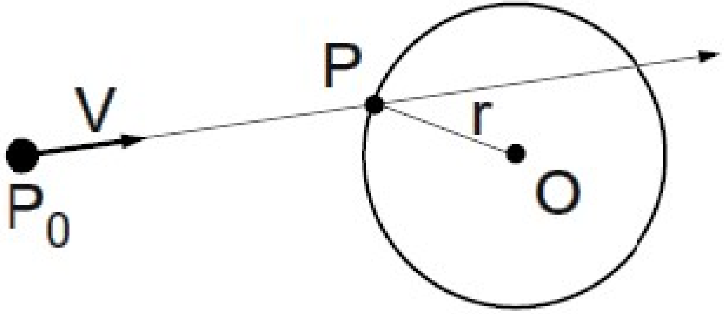
$$t = -(\mathbf{p}_0 \bullet \mathbf{n} + c) / \mathbf{V} \bullet \mathbf{n}$$

- For intersection with polygon, check if intersection point lies inside polygon
- Barycentric coordinates system

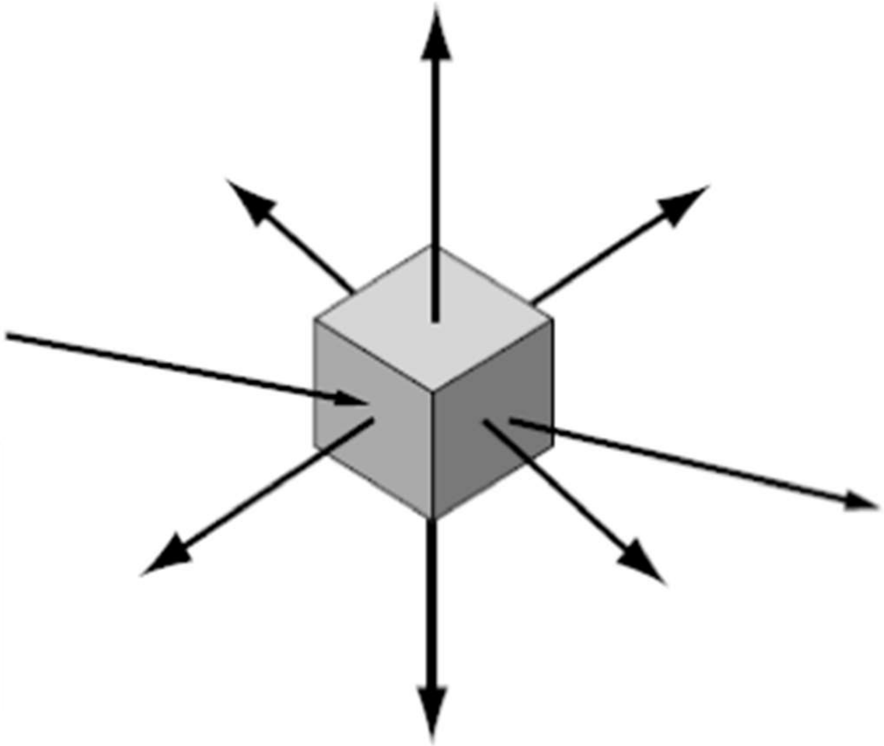


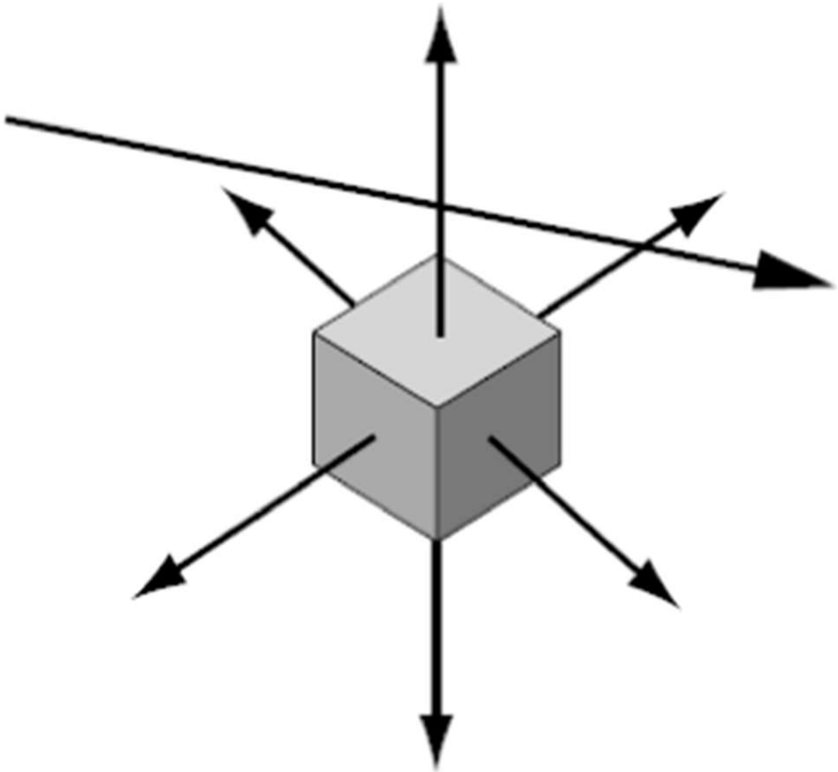
Ray-Sphere Intersection

- Intersect a sphere with the ray (algebraic)
- Ray parameterization: $P(t) = P_0 + tV$
- Sphere equation: $\| P - O \|^2 - r^2 = 0$
- Substitute: $\| P_0 + tV - O \|^2 - r^2 = 0$
- Solve: $t^2 + 2V^t(P_0 - O)t + (\| P_0 - O \|^2 - r^2) = 0$



Convex Polyhedra Intersection

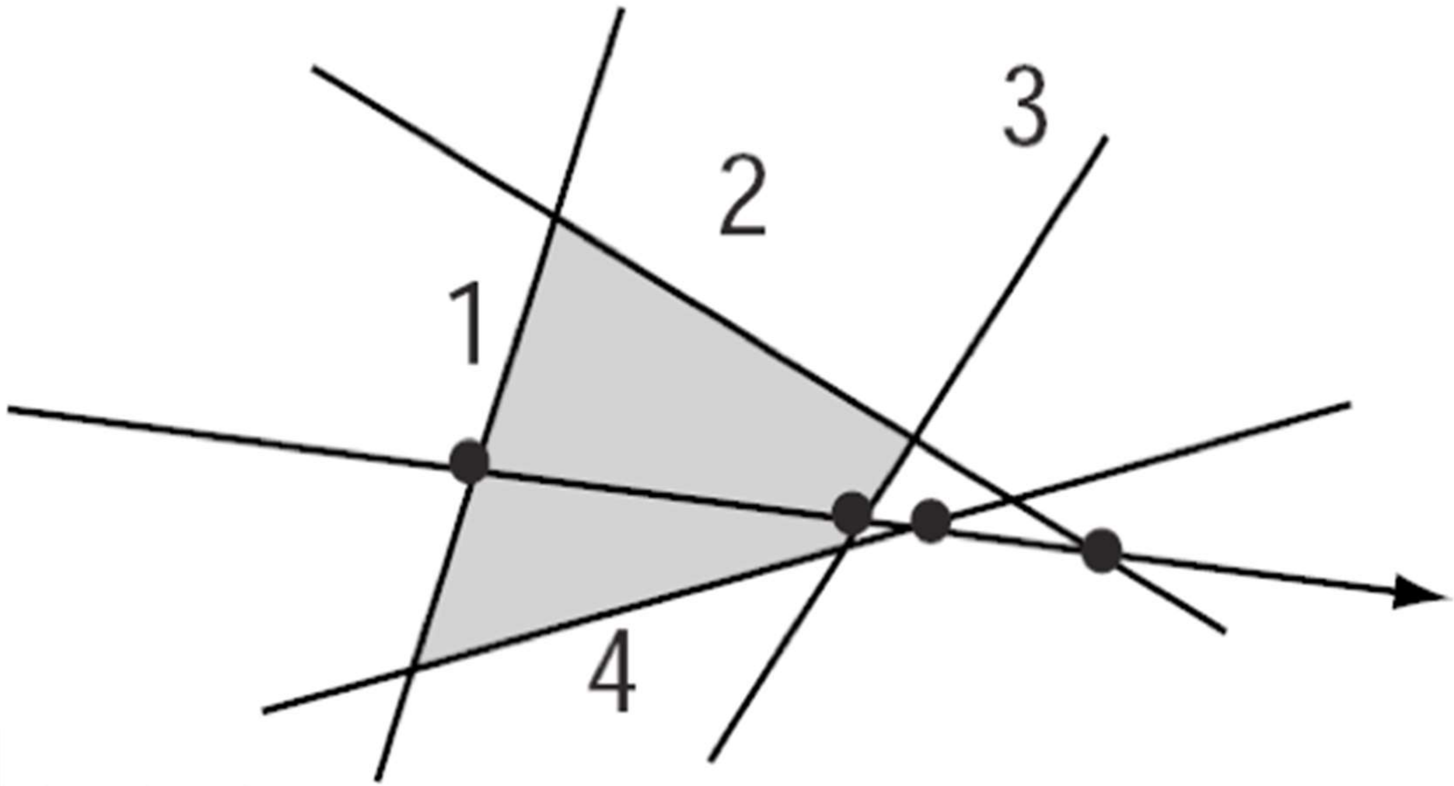




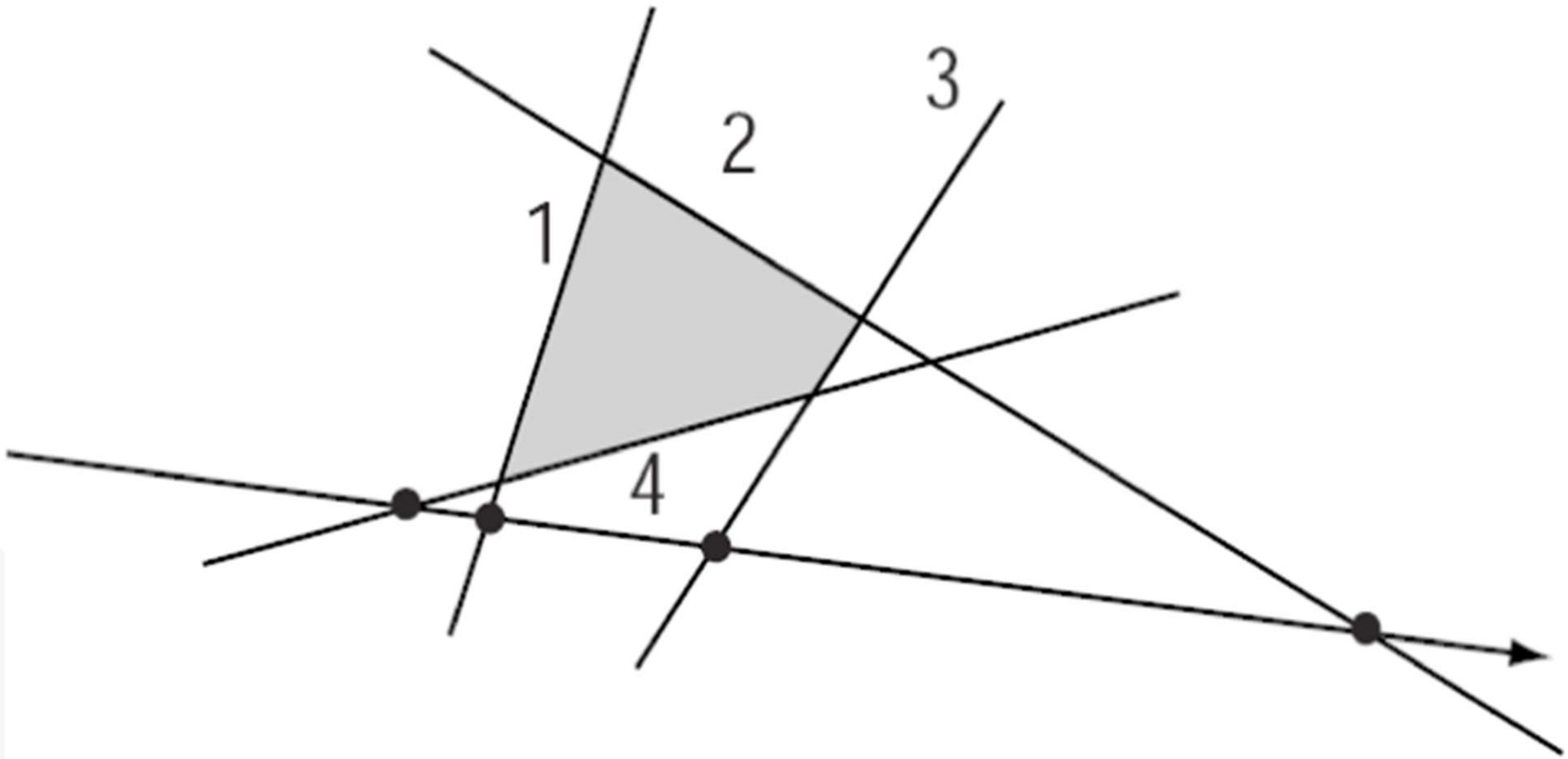
Convex Polyhedra Intersection

- Polyhedron is formed by intersection of planes.
- If ray enters an object, it must enter a front facing polygon and leave a back facing polygon.
- Ray enters at furthest intersection with front facing planes.
- Ray leaves at closest intersection with back facing planes.
- If entry is further away than exit, ray must miss the polyhedron.

Convex Polyhedra Intersection

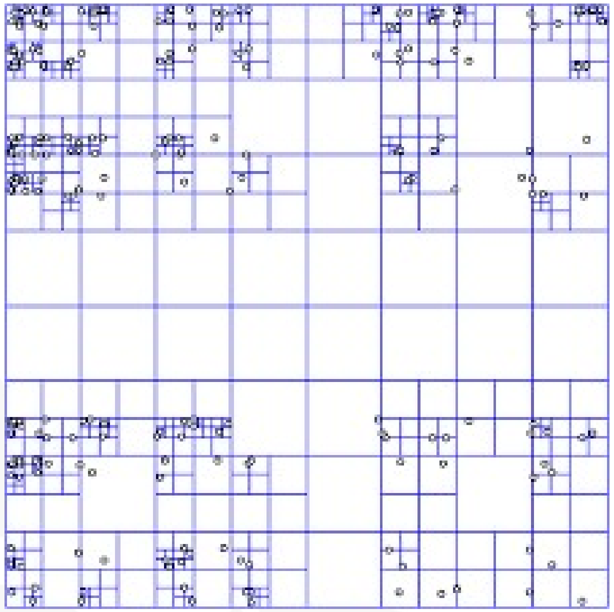


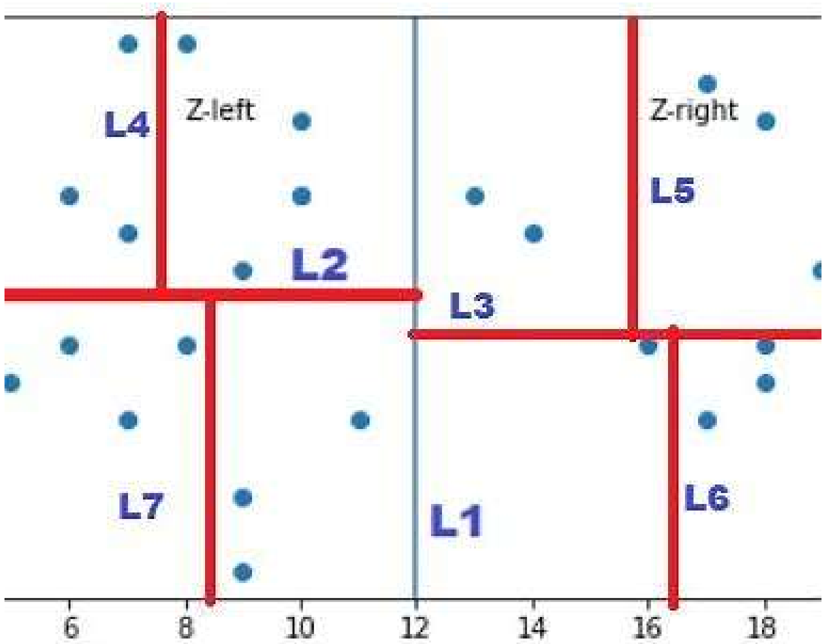
Convex Polyhedra Intersection



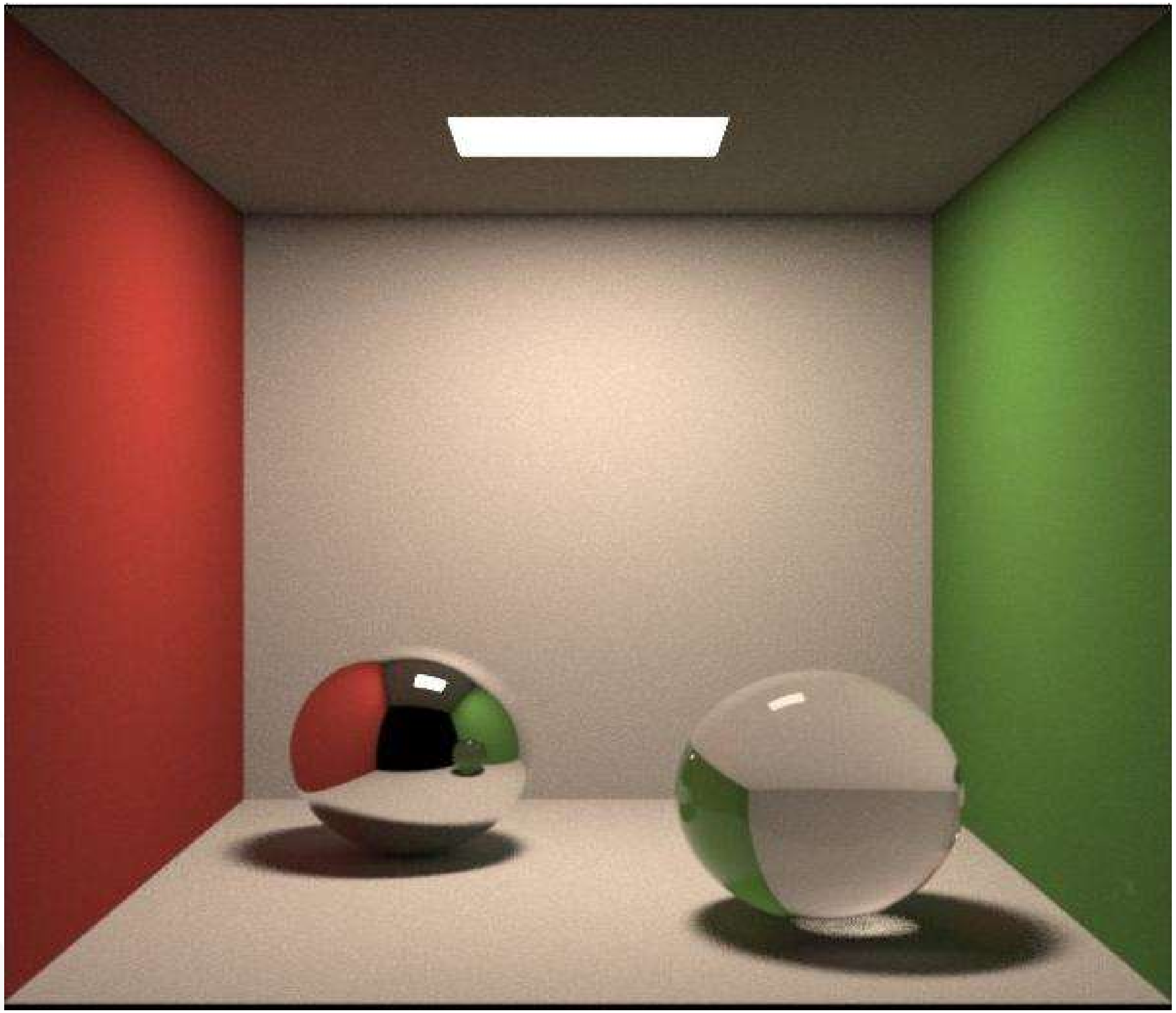
Further Acceleration

- Bounding box
- Group of bounding boxes
- Quad tree (Octree)
- K-D tree





Ray Traced Images

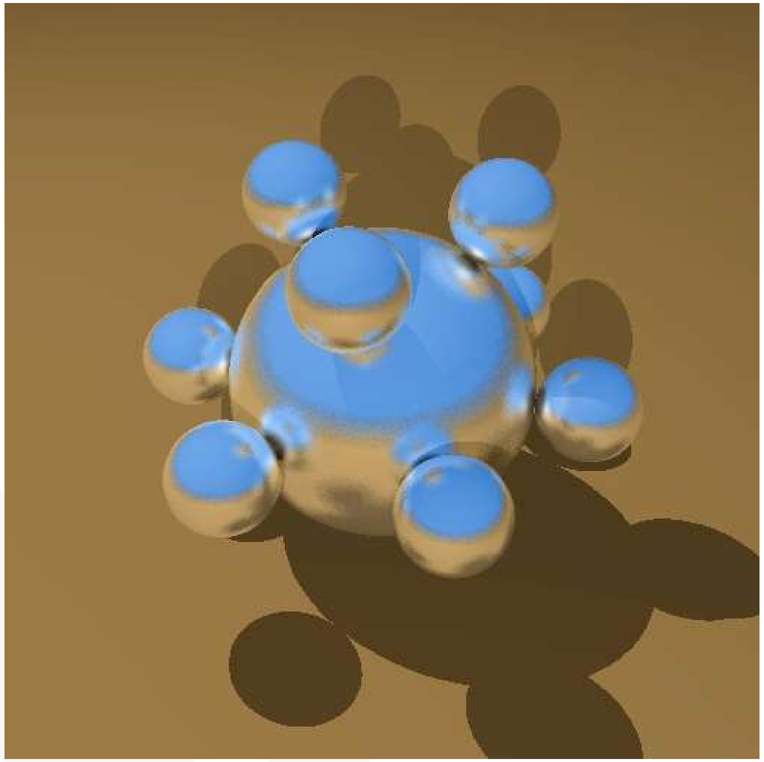


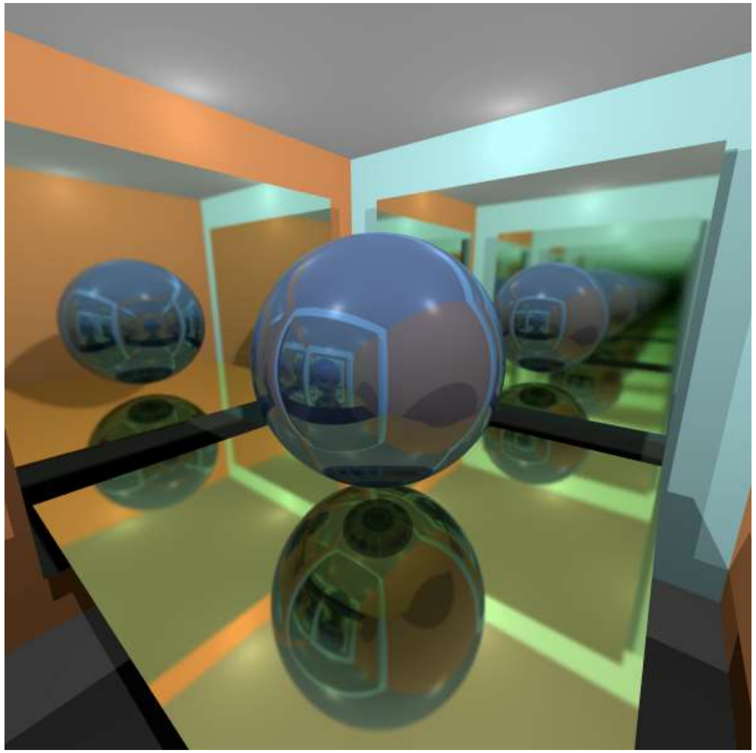
Ray Traced Images

Ray Tracing Capabilities

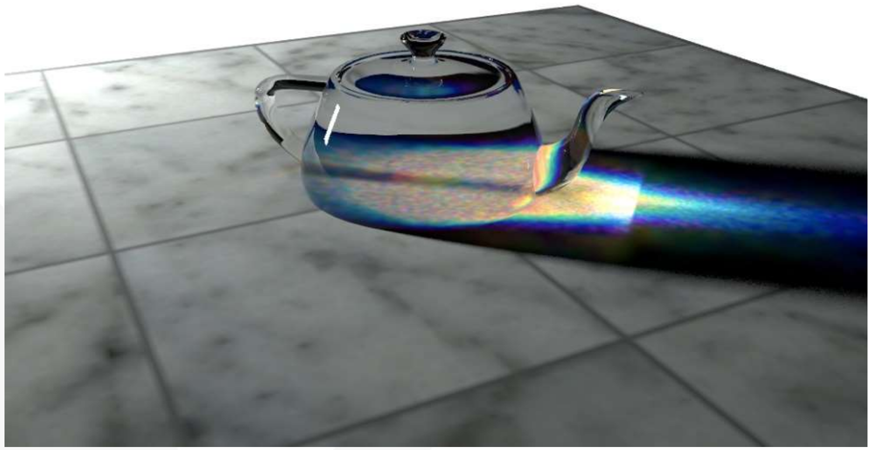
- Basic algorithm allows for
  - Hidden surface removal (like z-buffering)
  - Multiple light sources
  - Reflections
  - Transparent refractions
  - Hard shadows
- Extensions can achieve
  - Soft shadows
  - Motion blur
  - Blurred reflections (glossiness)
  - Depth of field (finite apertures)
  - Translucent refractions and more

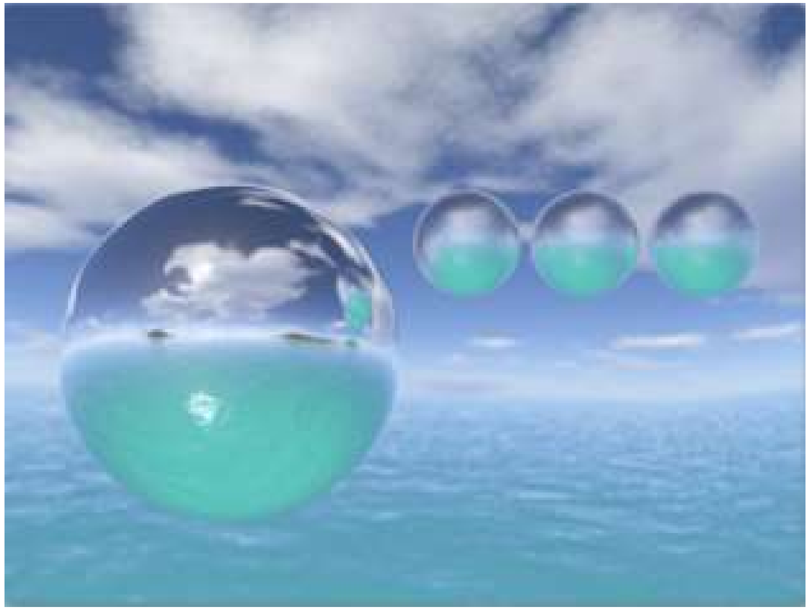
Glossy Reflection Examples



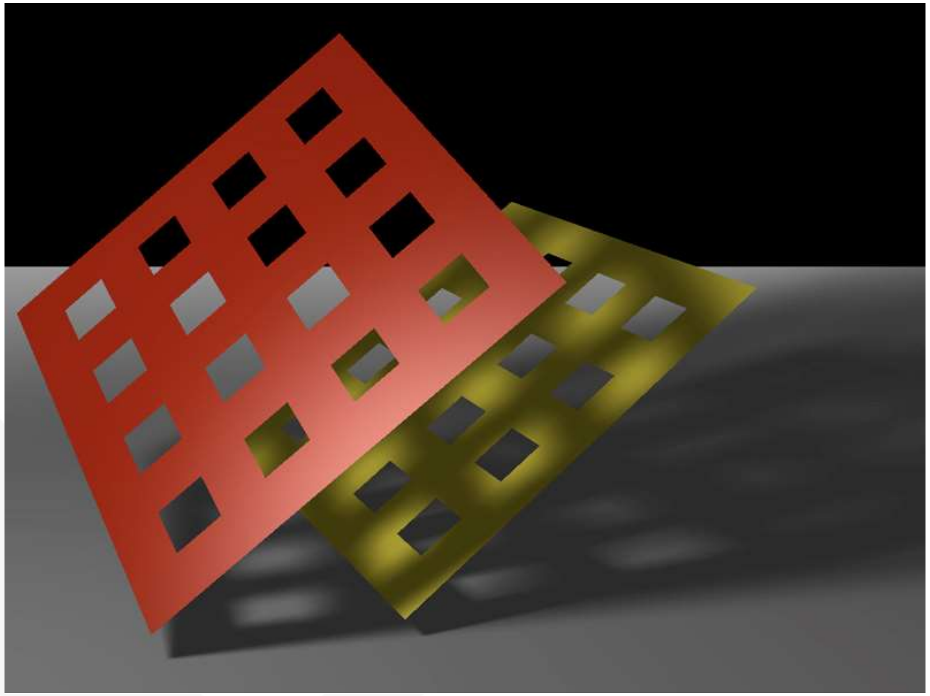


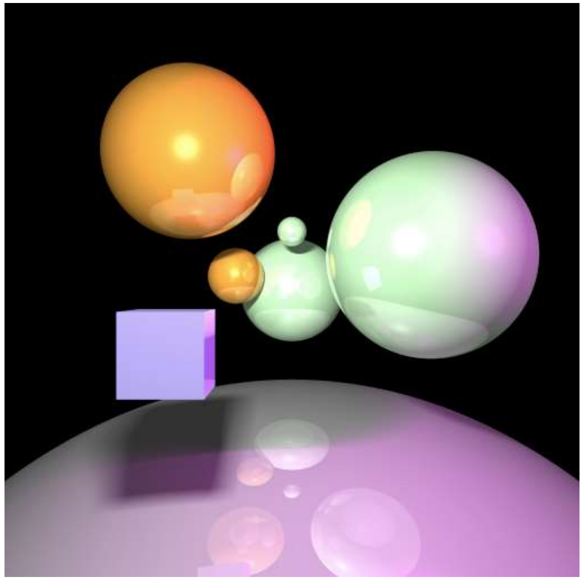
Translucency Examples

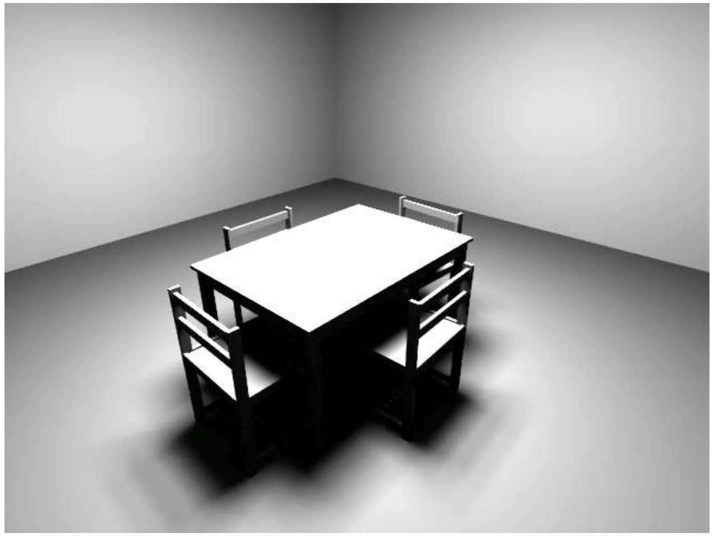




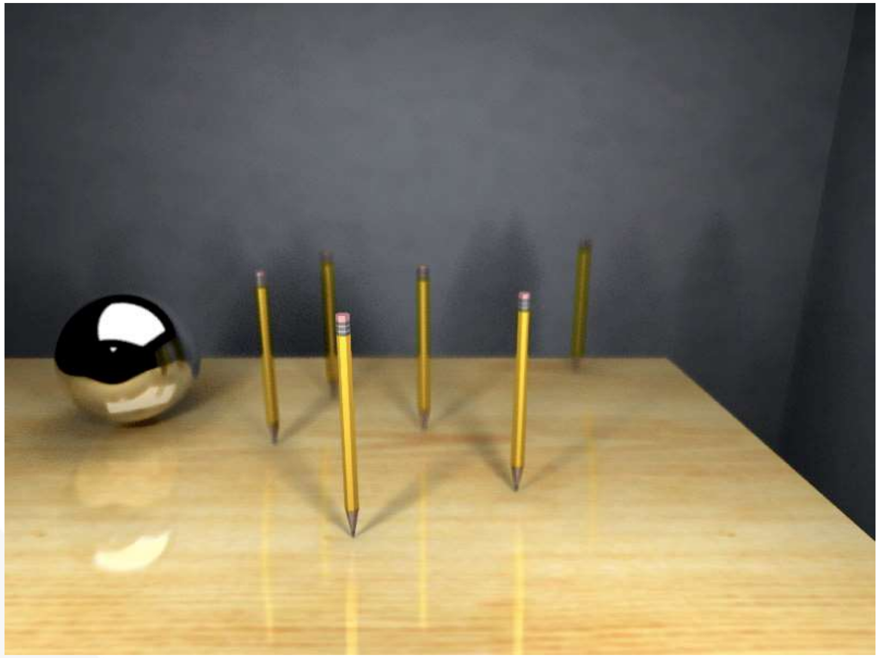
Soft Shadow Examples

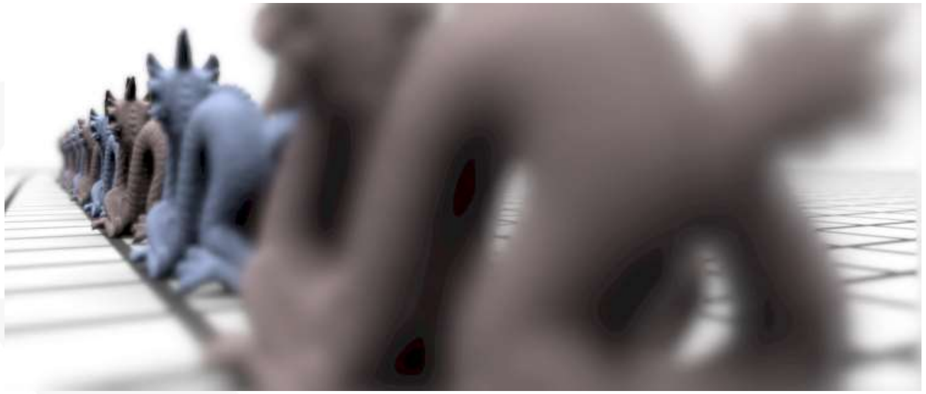






Depth of Field Examples





Motion Blur Examples