Computer Graphics
Ray tracing

Ming-Te Chi

Department of Computer Science,

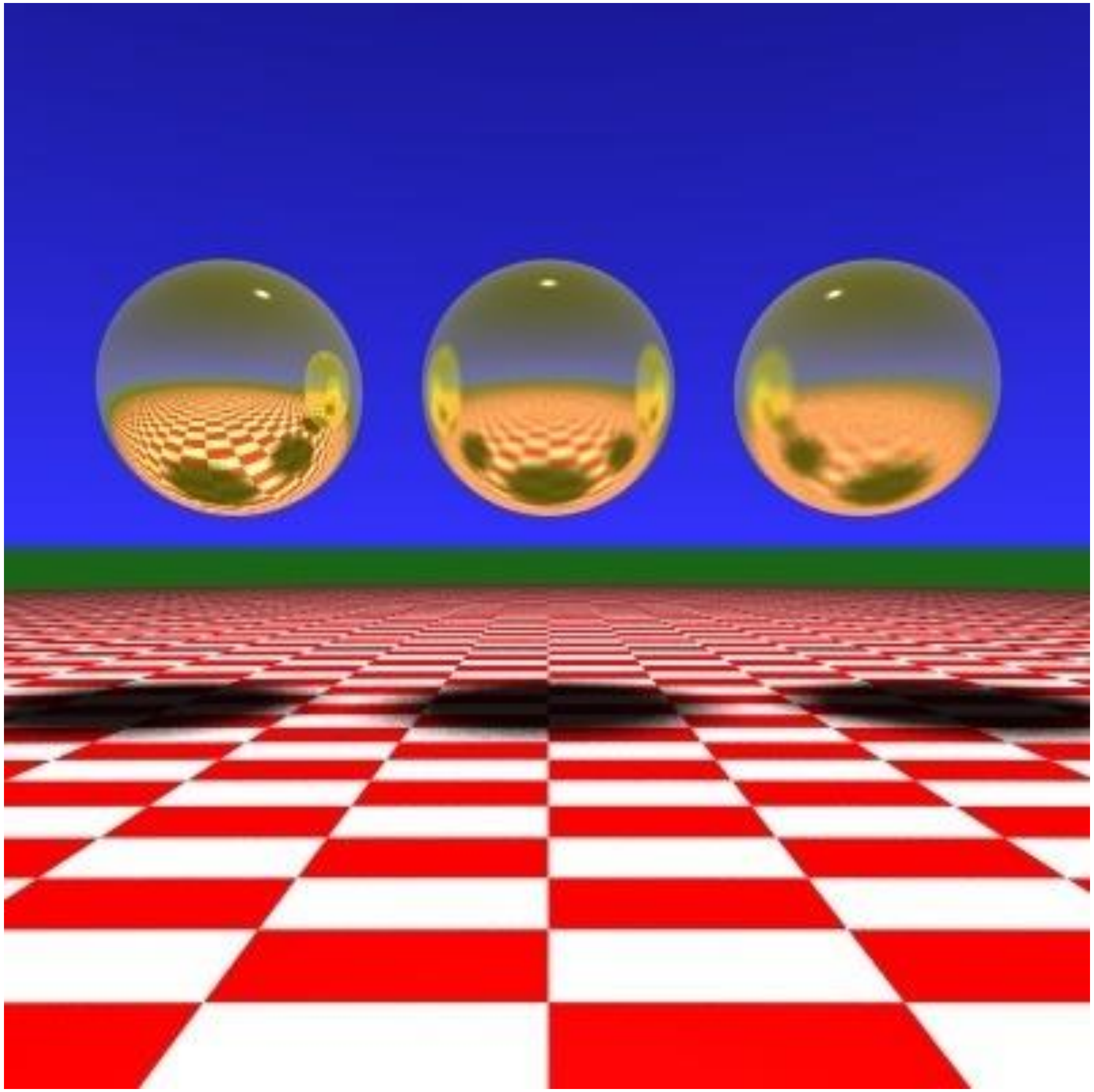
National Chengchi University

Global Illumination

- Ray tracing
  - Ray / Intersections
  - shading
  - Implementation
- Ray tracing in complex scene

- Distributed Ray Tracing
- Rendering equation

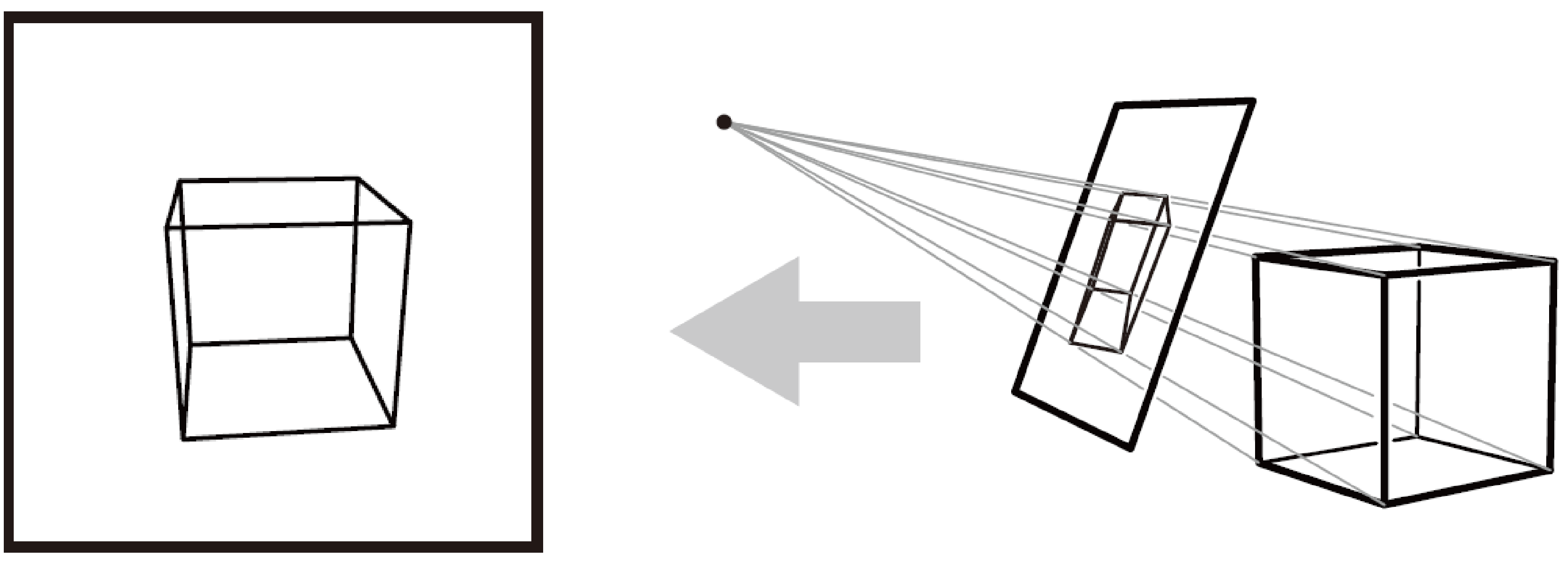
RAY TRACING



Slide Courtesy of Roger Crawfis, Ohio State

Projection

- Project object into the image plane



Two approaches to rendering

Object order

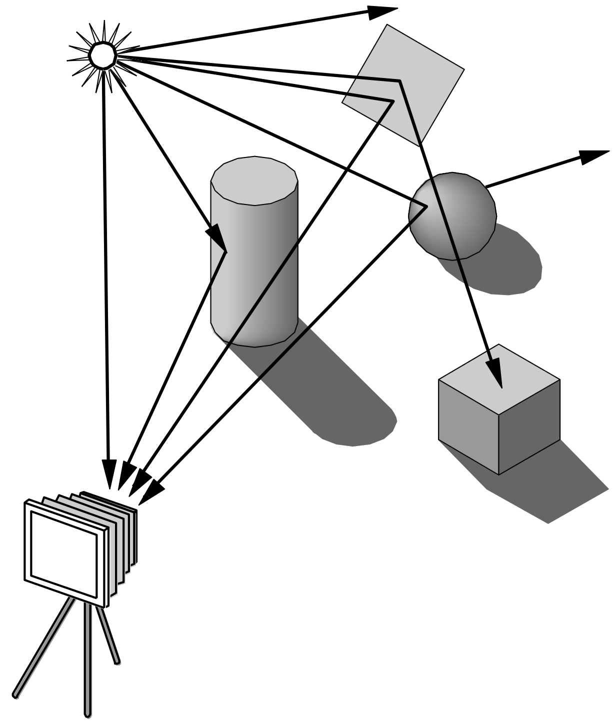
```
for each object {
    for each pixel {
        If (object affect pixel) {
            Do something
        }
    }
}
```

Image order

```
for each pixel {
    for each object {
        If (object affect pixel) {
            Do something
        }
    }
}
```

Ray Tracing

- Follow rays of light from a point source
- Can account for reflection and transmission

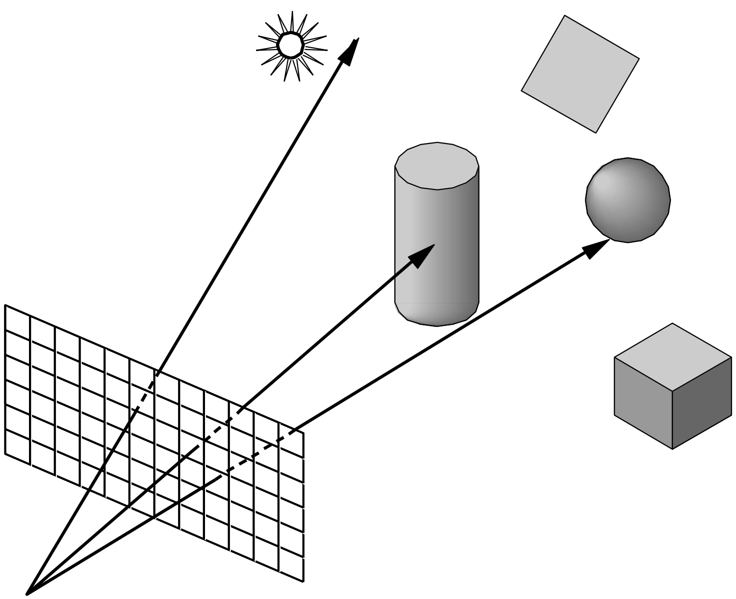


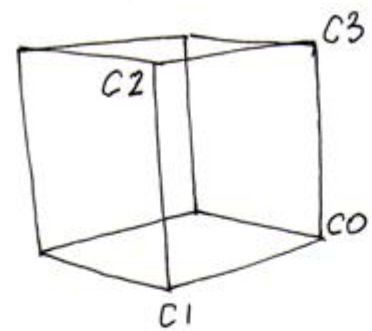
Computation

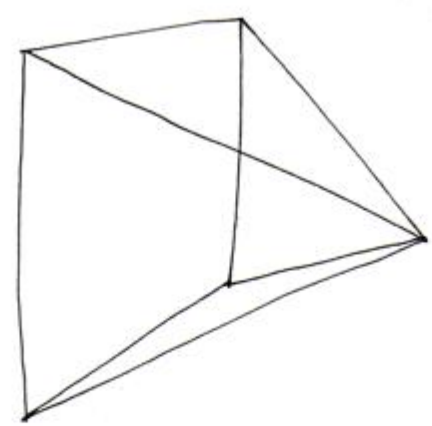
- Should be able to handle all physical interactions
- Ray tracing paradigm is not computational
- Most rays do not affect what we see
- Scattering produces many (infinite) additional rays
- Alternative: ray casting

Ray Casting

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel

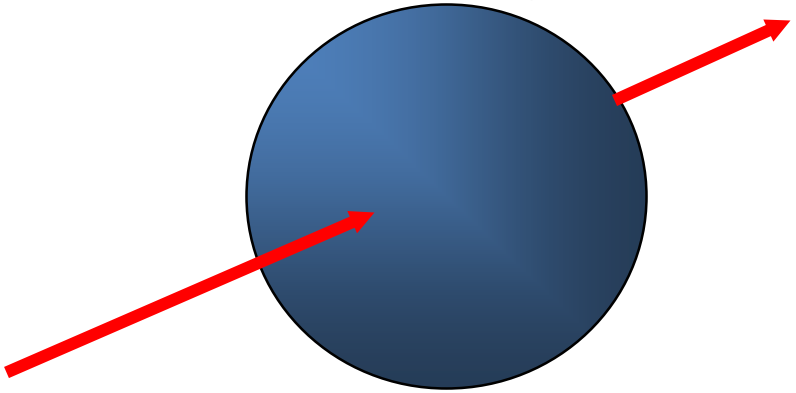






Ray Casting a Sphere

- Ray is parametric
- Sphere is quadric
- Resulting equation is a scalar quadratic equation which gives entry and exit points of ray (or no solution if ray misses)



INTERSECTIONS

Computing Intersections

- Implicit Objects
  - Quadrics
- Planes
- Polyhedra
- Parametric Surfaces

Implicit Surfaces

Ray from $\mathbf{p}_0$ in direction $\mathbf{d}$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

General implicit surface

$$f(\mathbf{p}) = 0$$

Solve scalar equation

$$f(\mathbf{p}(t)) = 0$$

General case requires numerical methods

Sphere

$$f(p) = ||p - c|| - r = 0$$

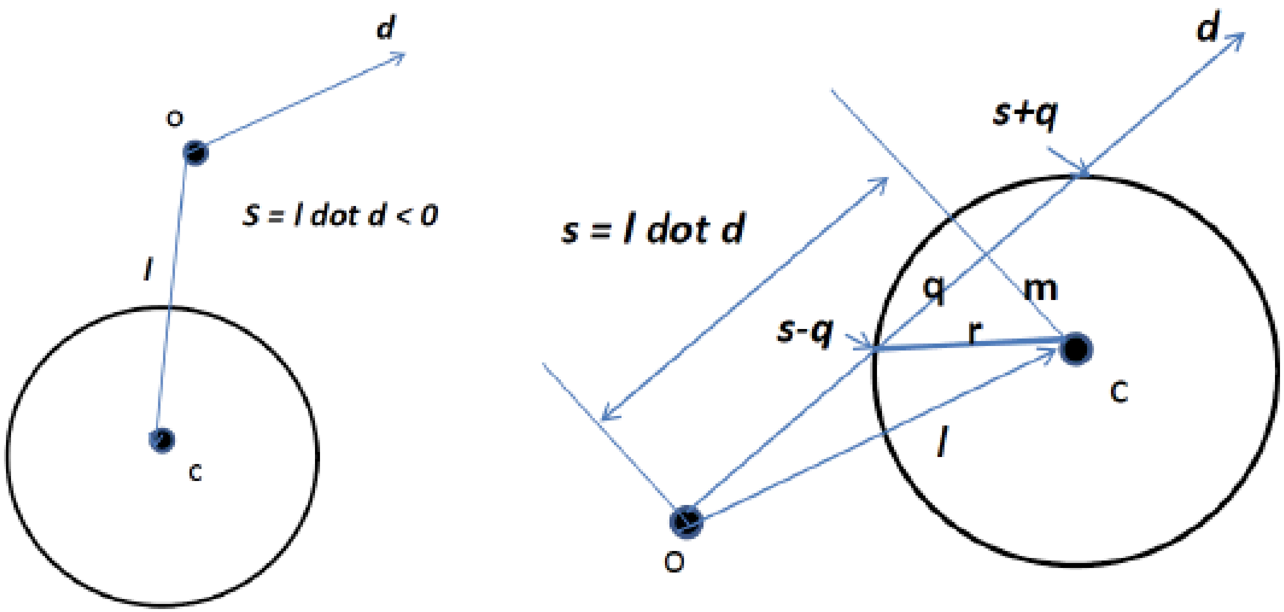
$$f(r(t)) = ||r(t) - c|| - r = 0$$

$$||o + td - c|| = r$$

$$t^2(d \cdot d) + 2t(d \cdot (o - c)) + (o - c) \cdot (o - c) - r^2 = 0$$

$$t^2 + 2t(d \cdot (o - c)) + (o - c) \cdot (o - c) - r^2 = 0$$

$$t^2 + bt - c = 0$$


d
o
S = l dot d < 0
l
c
d
s+q
s = l dot d
q
m
s-q
r
c
l
O

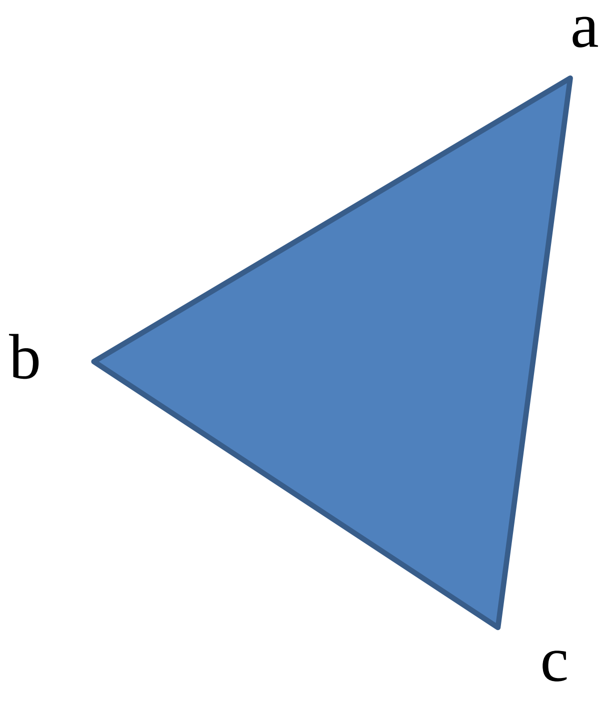
Planes

$\mathbf{p} \bullet \mathbf{n} + c = 0$

$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$

$t = \mathbf{-(p_0 \bullet n} + c)/\ \mathbf{d} \bullet \mathbf{n}$

Triangle



Quadrics

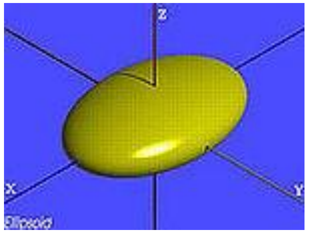
General quadric can be written as

$$\mathbf{p}^T\mathbf{A}\mathbf{p} + b^T\mathbf{p} + c = 0$$

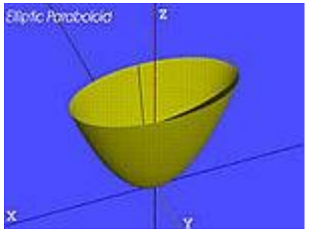
Substitute equation of ray

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

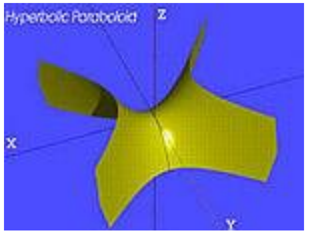
to get quadratic equation



Ellipsoid



Elliptic paraboloid



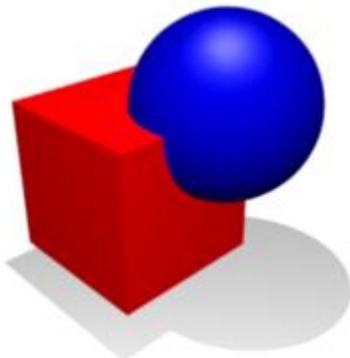
Hyperbolic paraboloid

…..

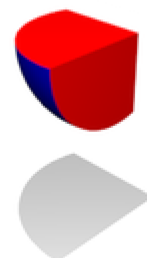
Ray Casting Quadrics

- Ray casting has become the standard way to visualize quadrics which are implicit surfaces in CSG systems
- Constructive Solid Geometry
  - Primitives are solids
  - Build objects with set operations
  - Union, intersection, set difference

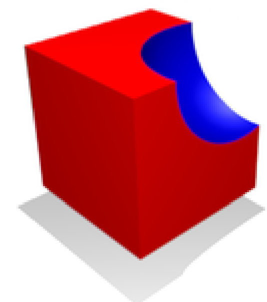
Constructive solid geometry (CSG)



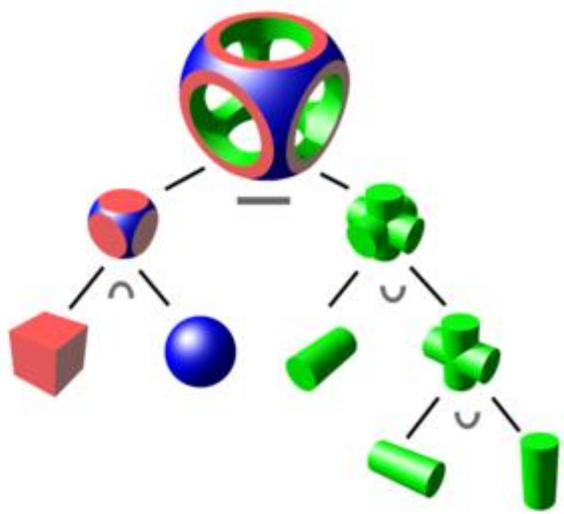
union



intersection



difference



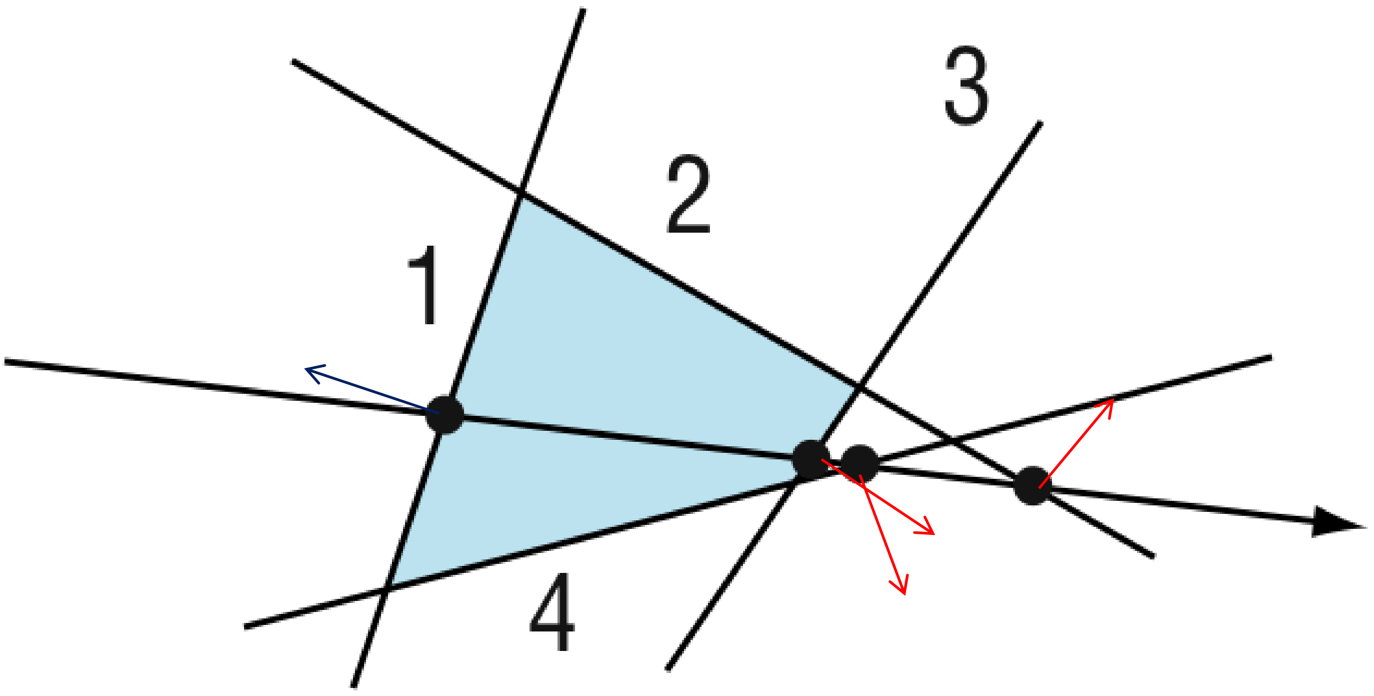
Polyhedra

- Generally we want to intersect with closed objects such as polygons and polyhedra rather than planes
- Hence we have to worry about inside/outside testing
- For convex objects such as polyhedra there are some fast tests

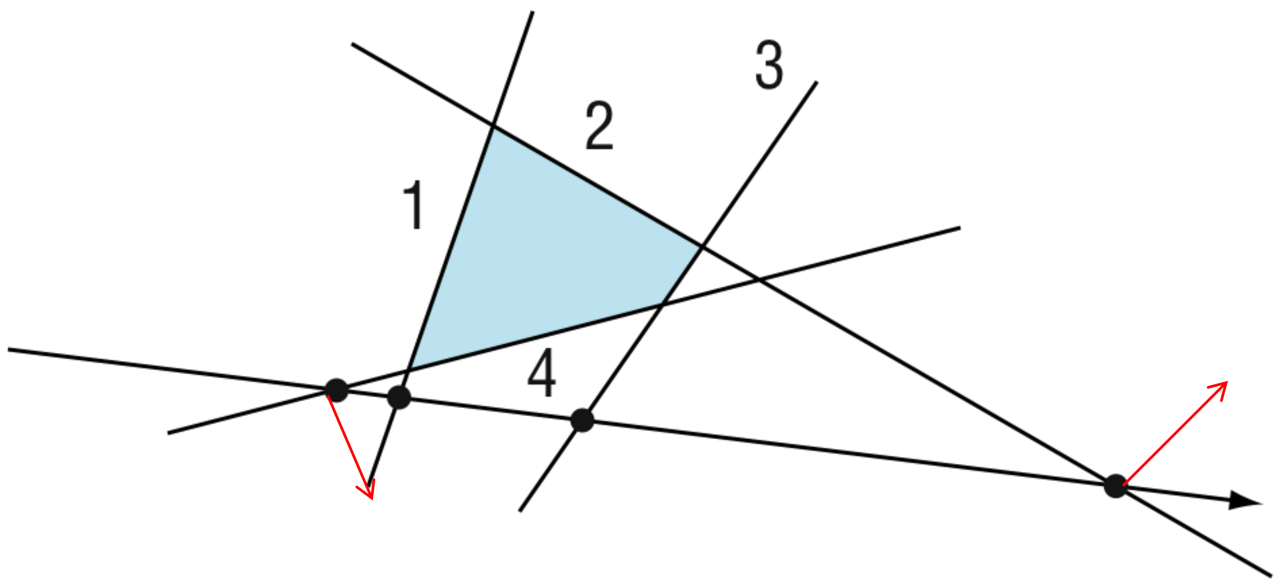
Ray Tracing Polyhedra

- If ray enters an object, it must enter a front facing polygon and leave a back facing polygon
- Polyhedron is formed by intersection of planes
- 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

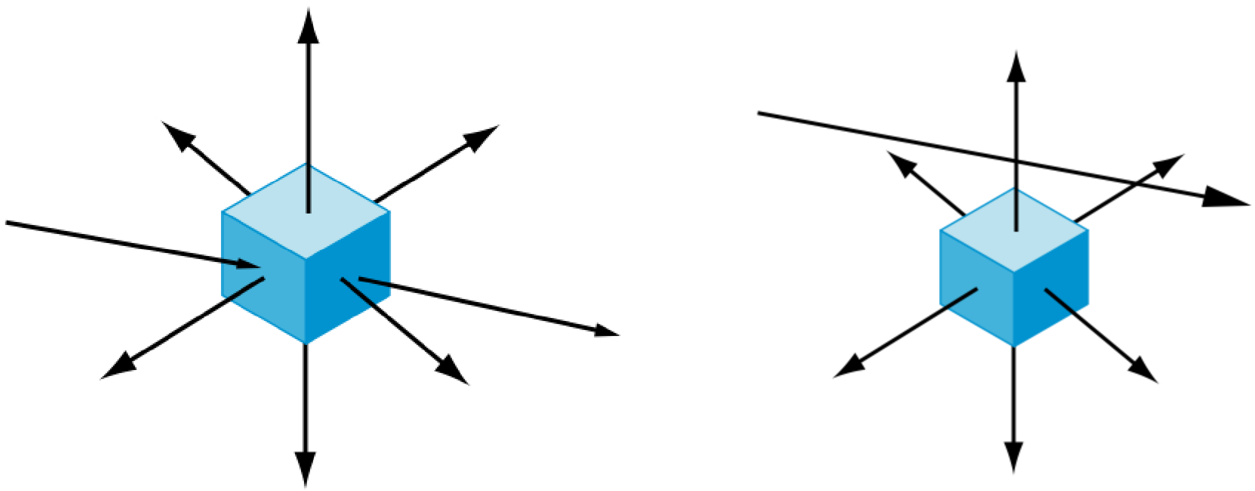
Ray Tracing a Polygon



Ray Tracing a Polygon



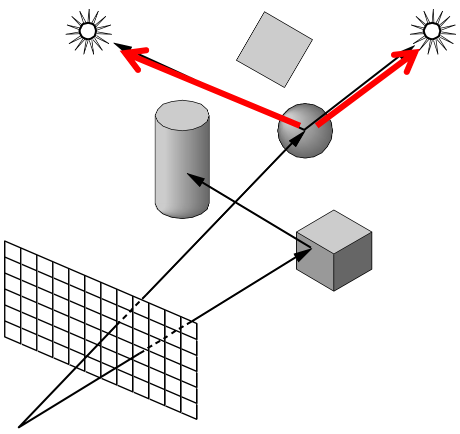
Ray Tracing Polyhedra



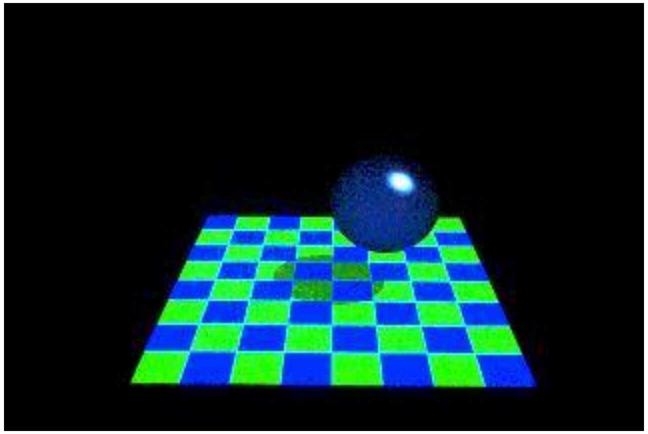
COMPLEX SCENE

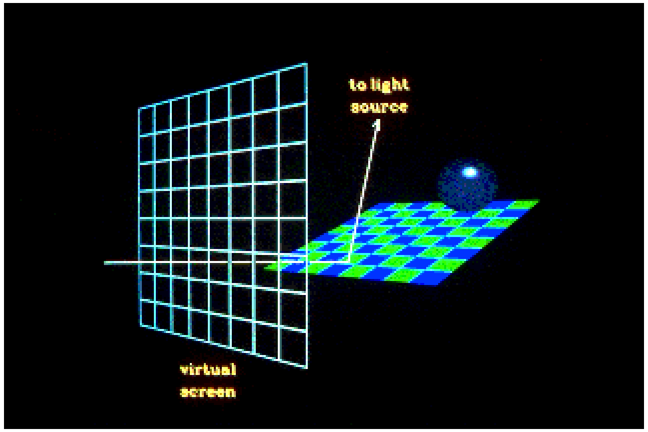
Shadow Rays

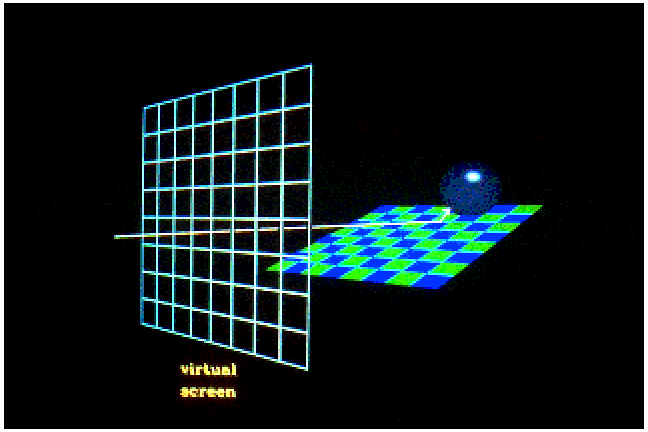
- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow or feeler rays



Shadow Rays

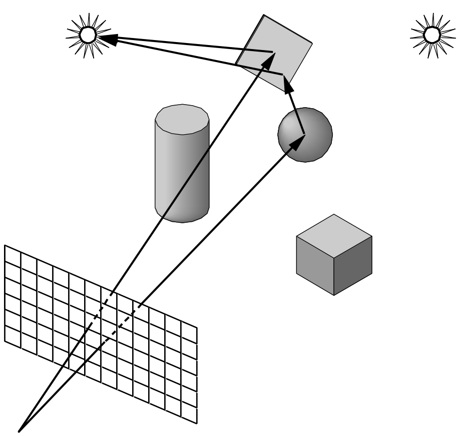






Reflection

- Must follow shadow rays off reflecting or transmitting surfaces
- Process is recursive



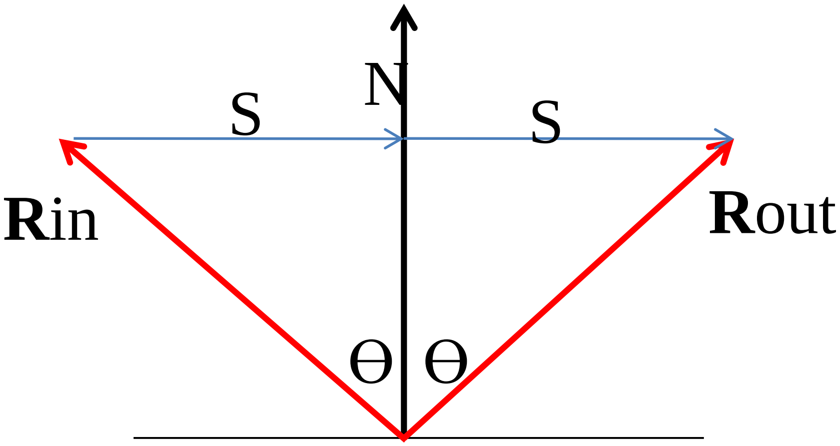
Computing a Reflected Ray

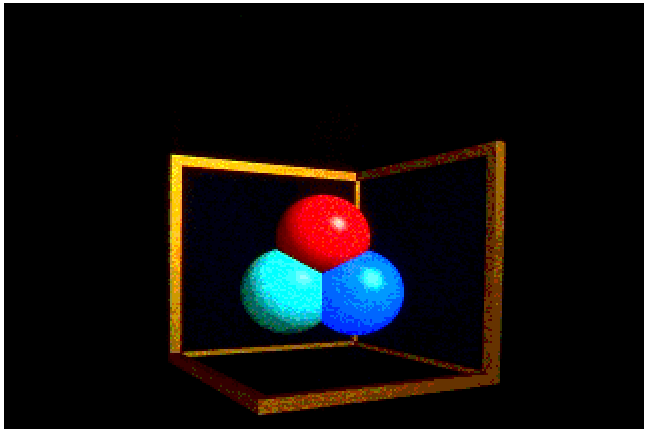
Note that for our rays: $R_{in} = -R_d$

The projection of R_{in} onto N is $N\cos(\theta)$

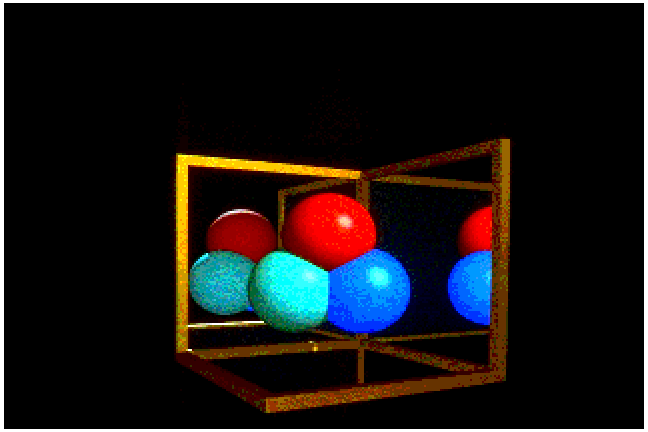
so

$$R_{out} = N\cos\theta + S$$
$$R_{in} + S = N\cos\theta$$
$$S = N\cos\theta - R_{in}$$
$$R_{out} = 2N\cos\theta - R_{in} = 2N(N \cdot R_{in}) - R_{in}$$

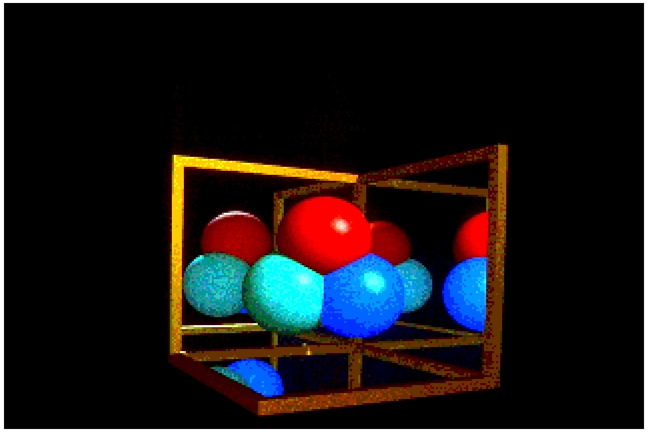




Scene with no reflection rays

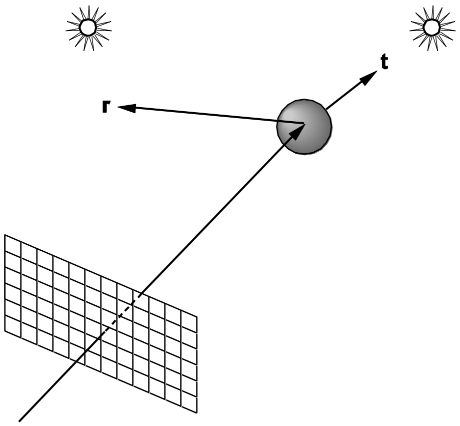


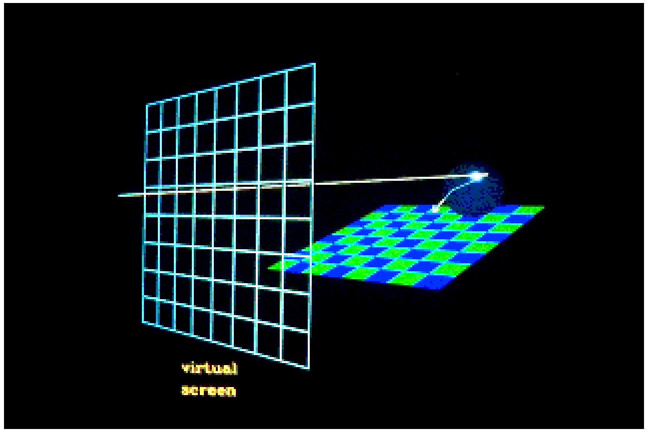
Scene with one layer of reflection

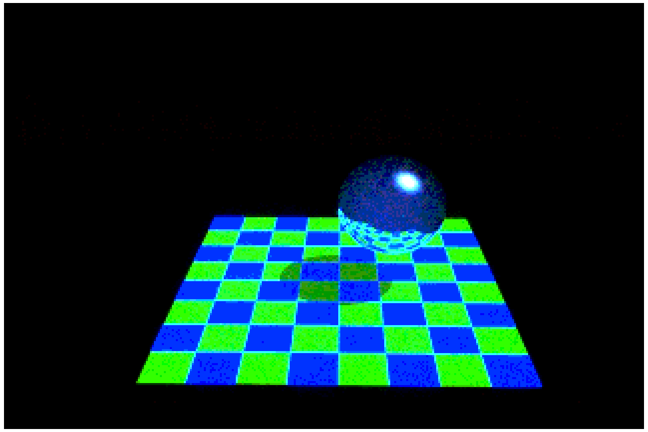


Scene with two layer of reflection

Transmission







Transformed ray

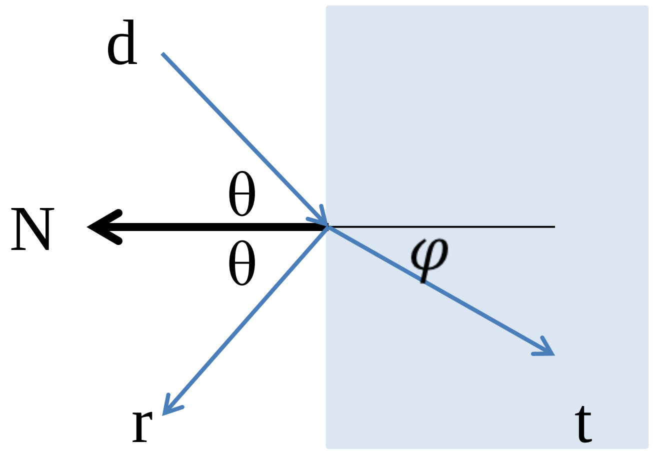
- Snell's law

$$n \sin\theta = n_t \sin\varphi$$

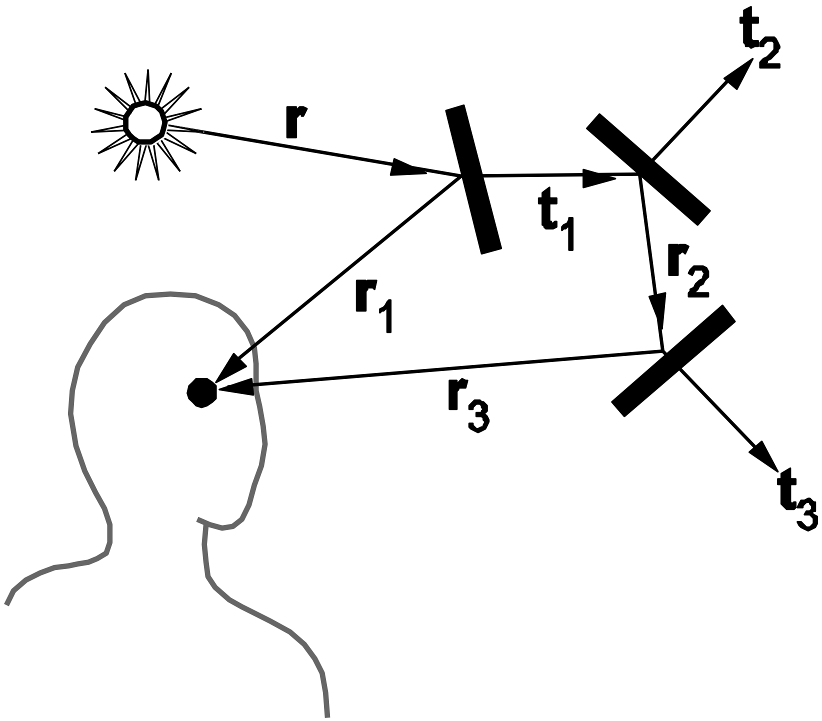
…

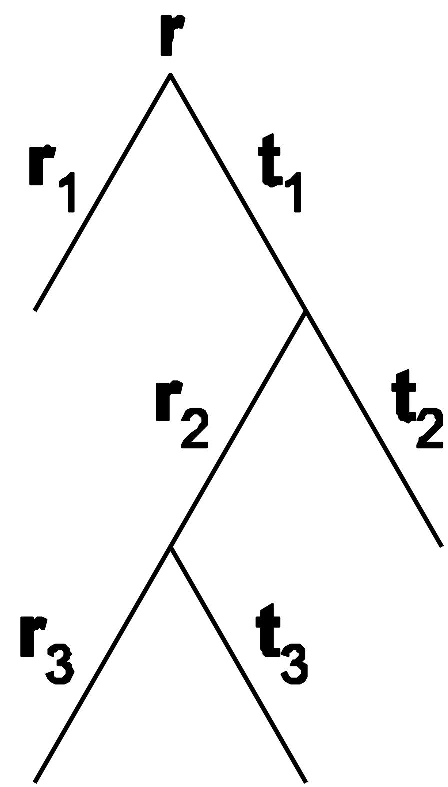
$$\eta = \frac{n}{n_t}$$

$$t = \eta(d + N\cos\theta) - N\sqrt{1 - \eta(1 - (d \cdot N)^2)}$$

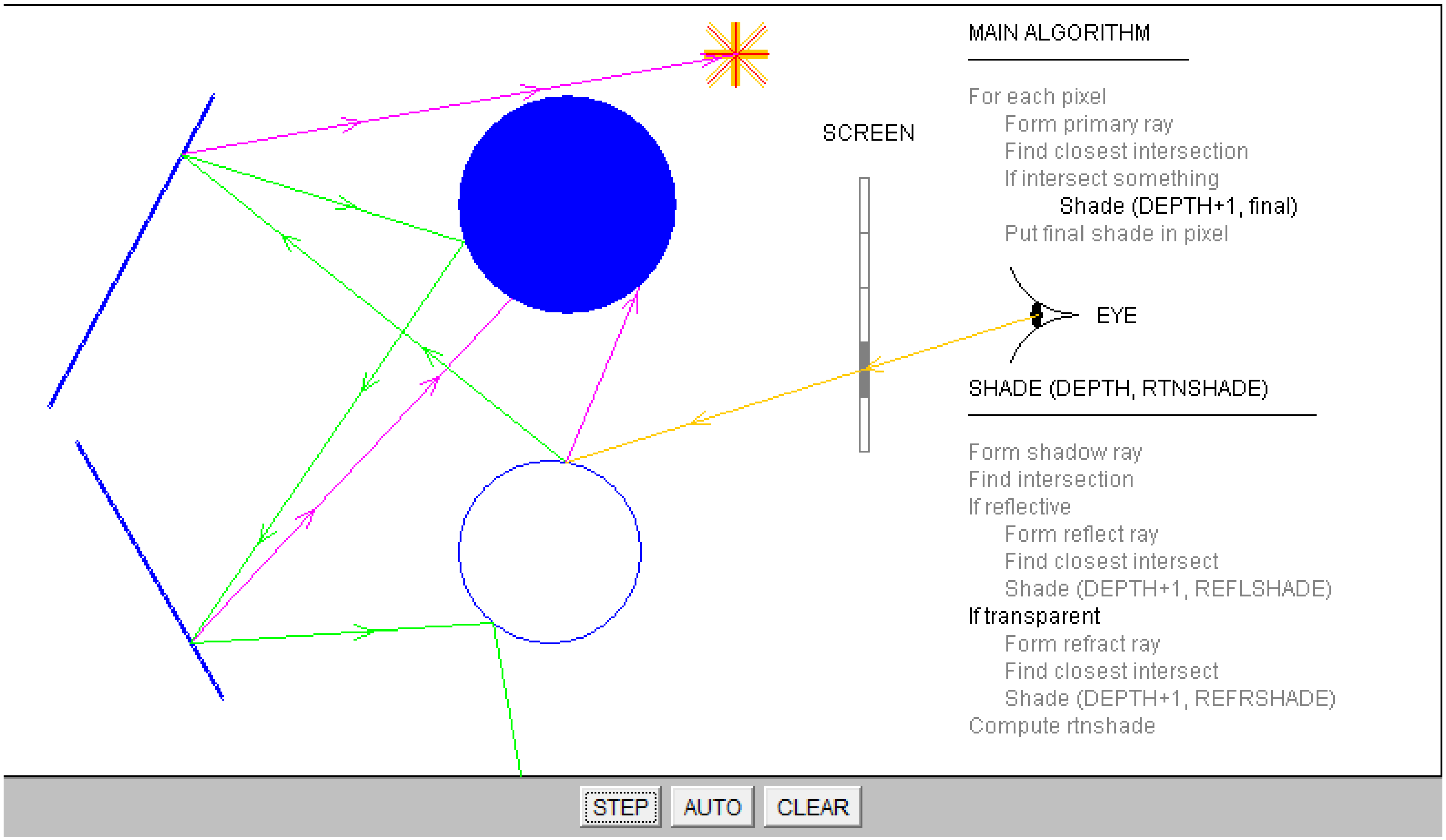


Ray Trees





A Ray Tracing demonstration program



Diffuse Surfaces

- Theoretically the scattering at each point of intersection generates an infinite number of new rays that should be traced
- In practice, we only trace the ***transmitted*** and ***reflected*** rays, but use the Phong model to compute shade at point of intersection
- Radiosity works best for perfectly diffuse (Lambertian) surfaces

Building a Ray Tracer

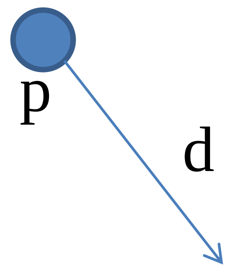
- Best expressed recursively
- Can remove recursion later
- Image based approach
  - For each ray …….
- Find intersection with closest surface
  - Need whole object database available
  - Complexity of calculation limits object types
- Compute lighting at surface
- Trace reflected and transmitted rays

When to stop

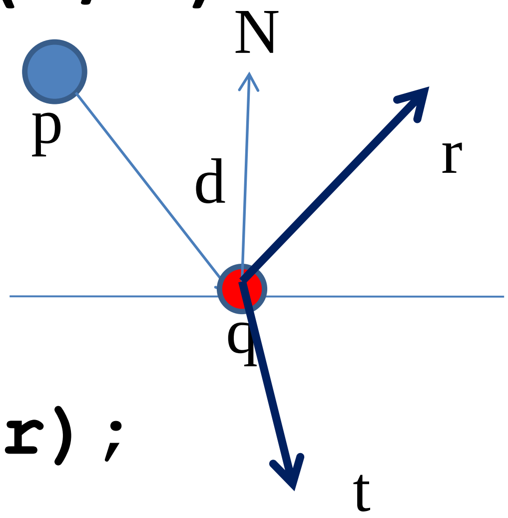
- Some light will be absorbed at each intersection
  - Track amount left
- Ignore rays that go off to infinity
  - Put large sphere around problem
- Count steps

Recursive Ray Tracer(1/3)

```
color c = trace(point p, vector d, int
  step)
{
   color local, reflected, transmitted;
   point q;
   normal n;

   if(step > max)
       return(background_color);
```



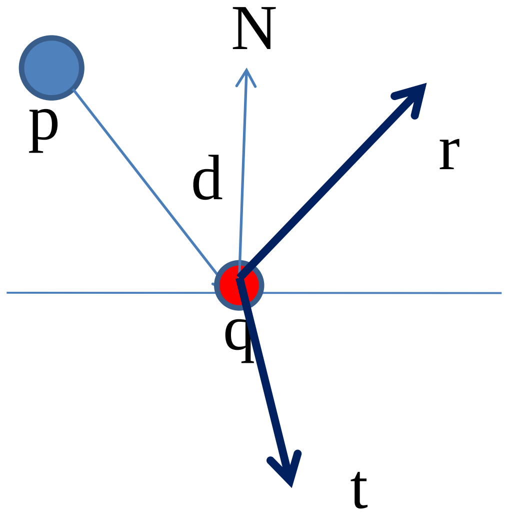
Recursive Ray Tracer (2/3)



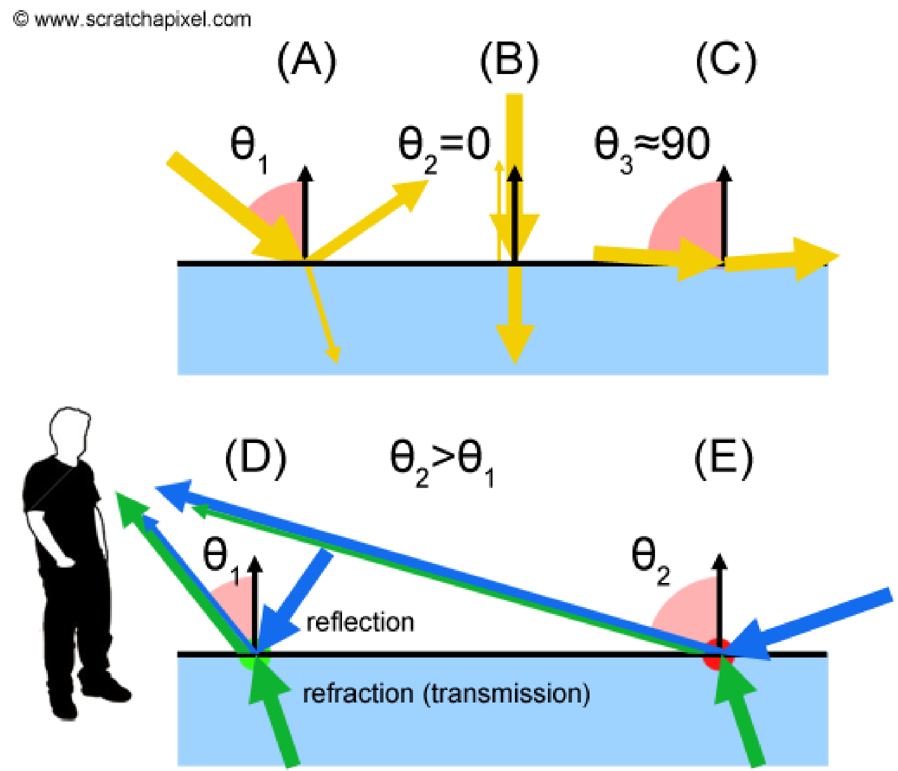

```
q = intersect(p, d, status);

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);
```

Recursive Ray Tracer (3/3)

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

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

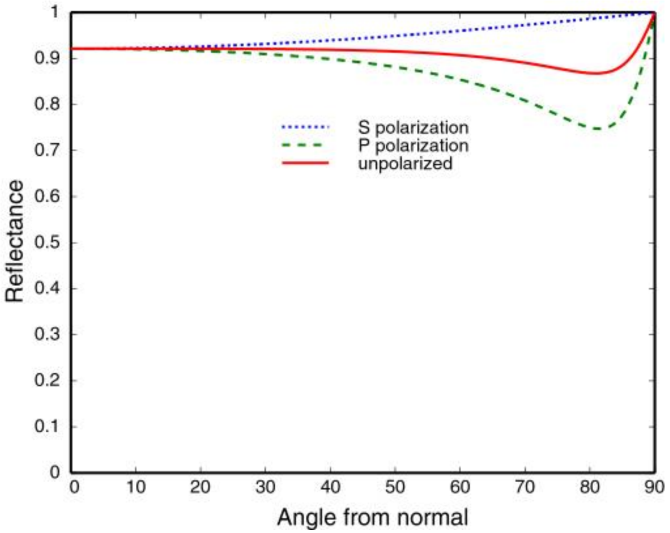


Reflection and refraction

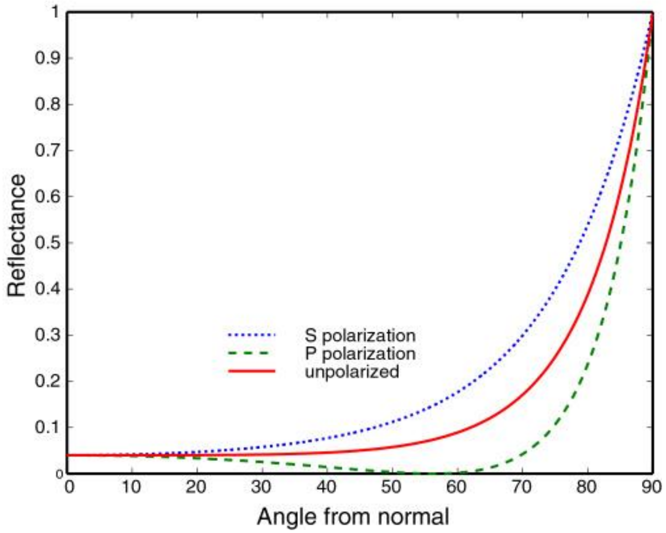


Fresnel Reflectance

- **Fresnel equation** describe the behaviour of light when moving between media of differing refractive indices.



conductive materials
aluminum



dielectric
glasses

- Schlick's approximation

the specular reflection coefficient R

$$R(\theta) = R_0 + (1 - R_0)(1 - cos\theta)^5$$

$$R_0 = (\frac{n_1 - n_2}{n_1 + n_2})^2$$

Robert L. Cook, Thomas Porter, Loren Carpenter

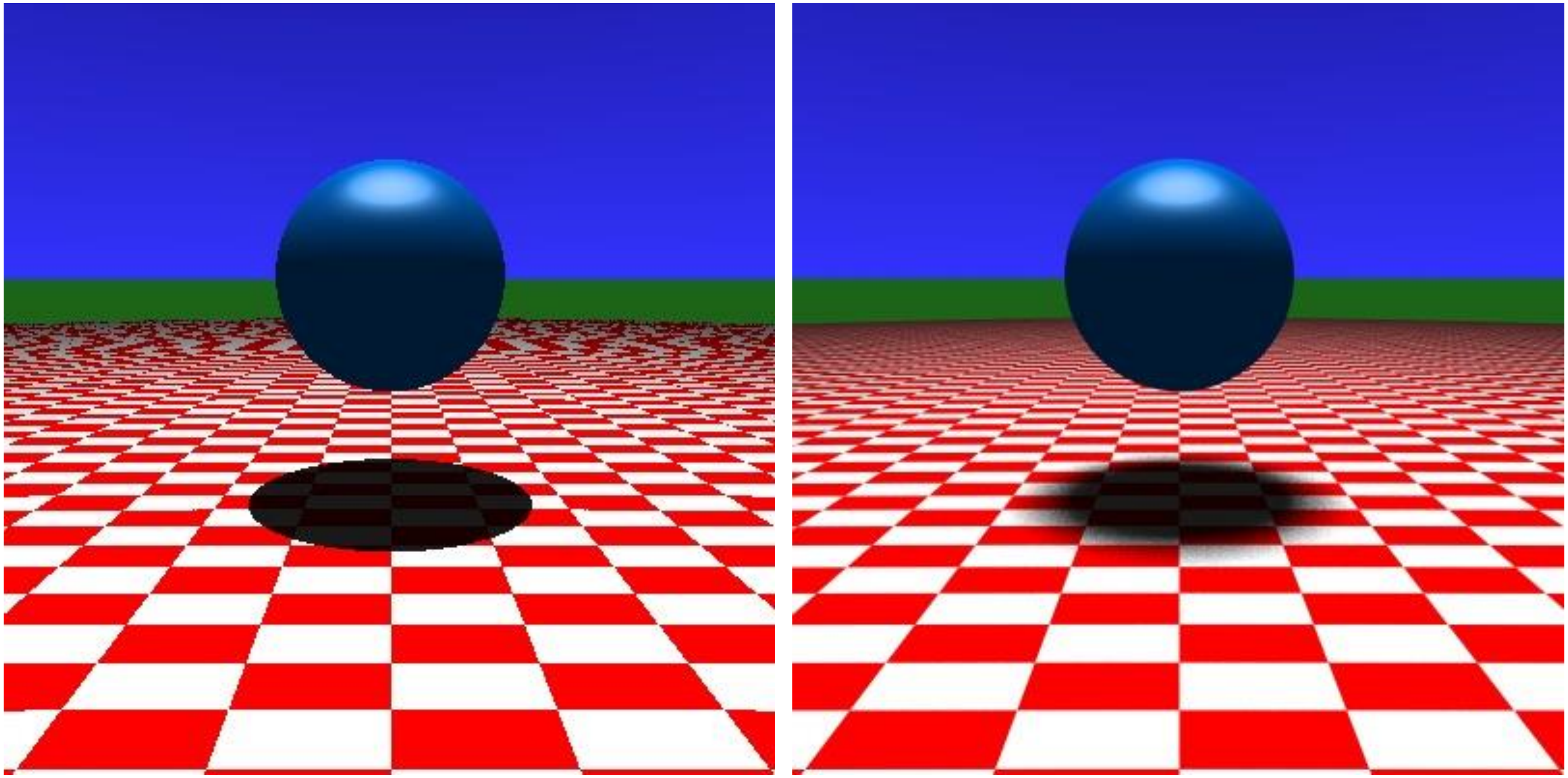
1984

DISTRIBUTED RAY TRACING

Shadows

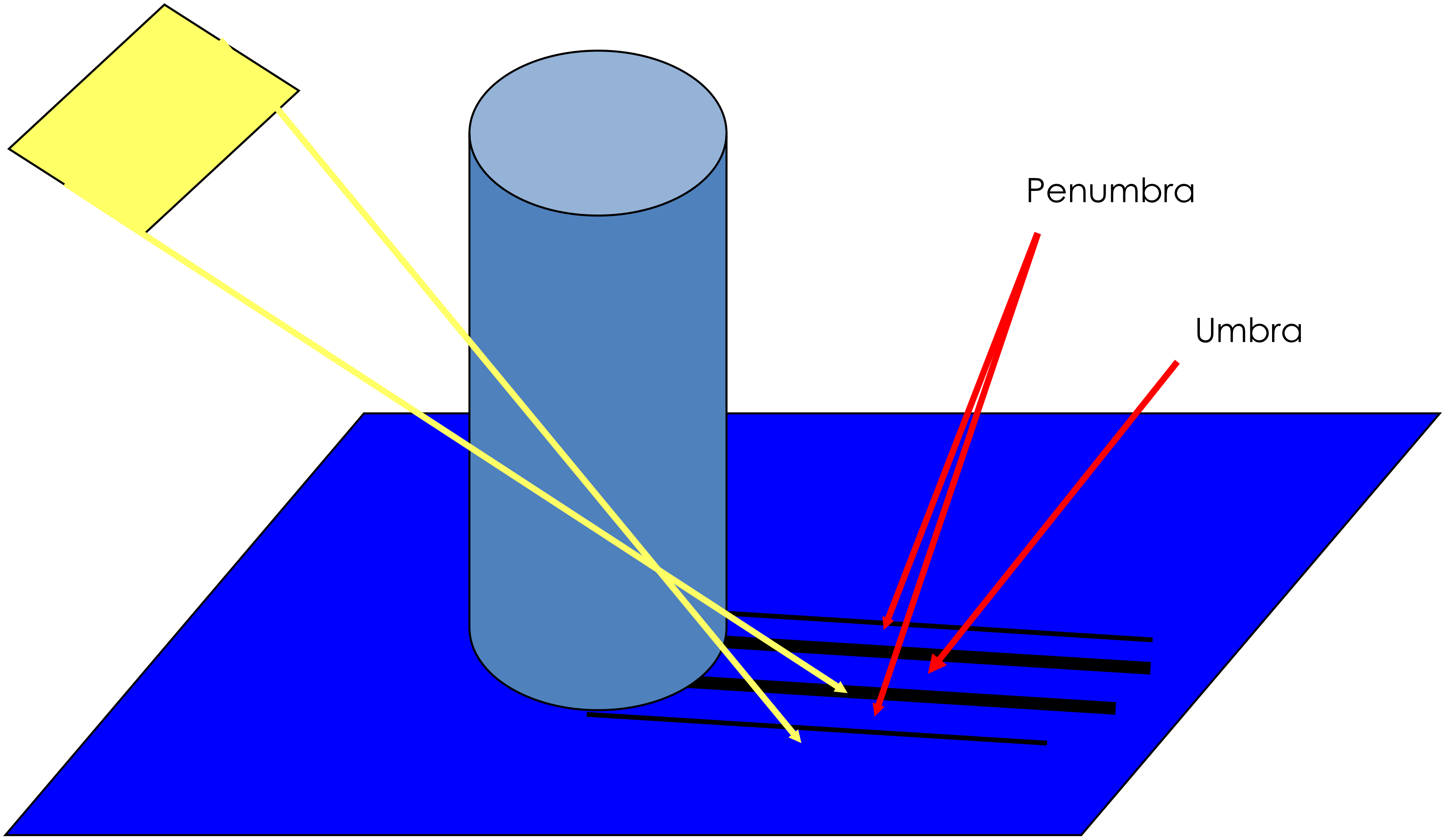
- Ray tracing casts shadow feelers to a point light source.
- Many light sources are illuminated over a finite area.
- The shadows between these are substantially different.
- Area light sources cast soft shadows
  - Penumbra
  - Umbra

Soft Shadows



Slide Courtesy of Roger Crawfis, Ohio State

Soft Shadows



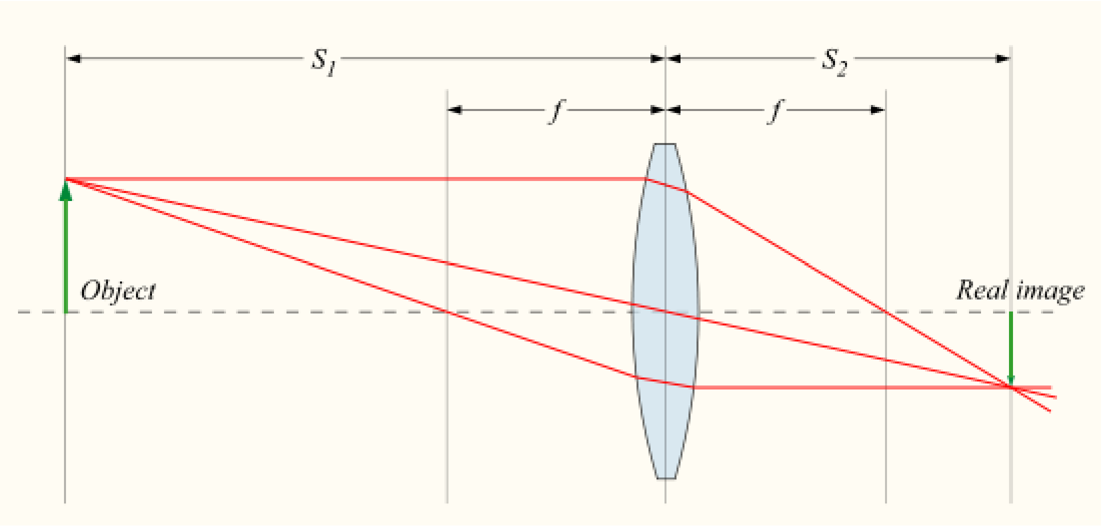
Slide Courtesy of Roger Crawfis, O

Camera Models

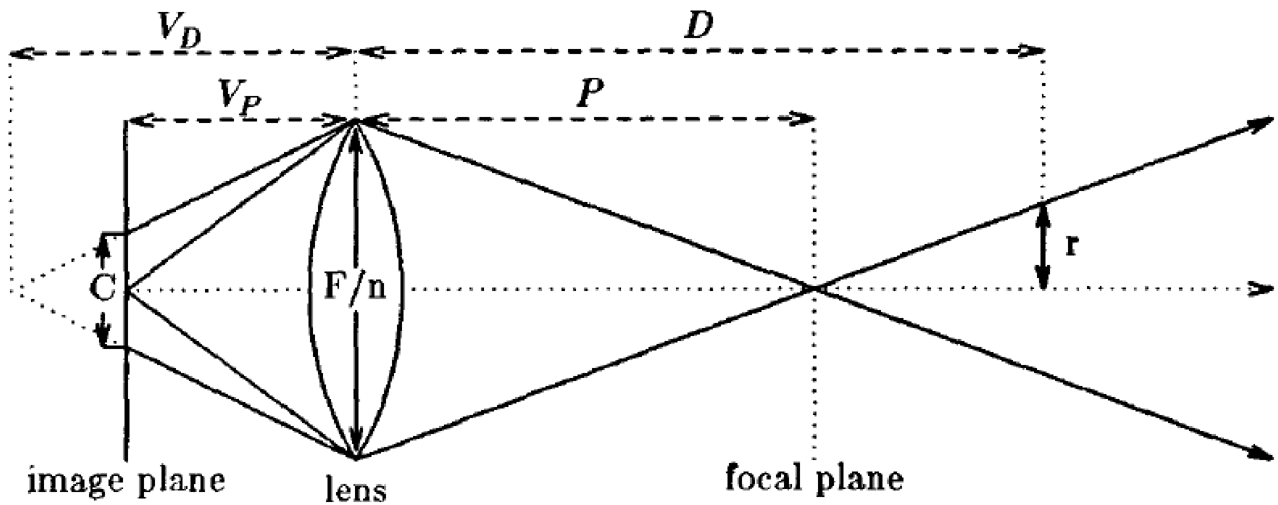
- Up to now, we have used a pinhole camera model.
- These has everything in focus throughout the scene.
- The eye and most cameras have a larger lens or aperature.

thin lens formula

- $\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}$



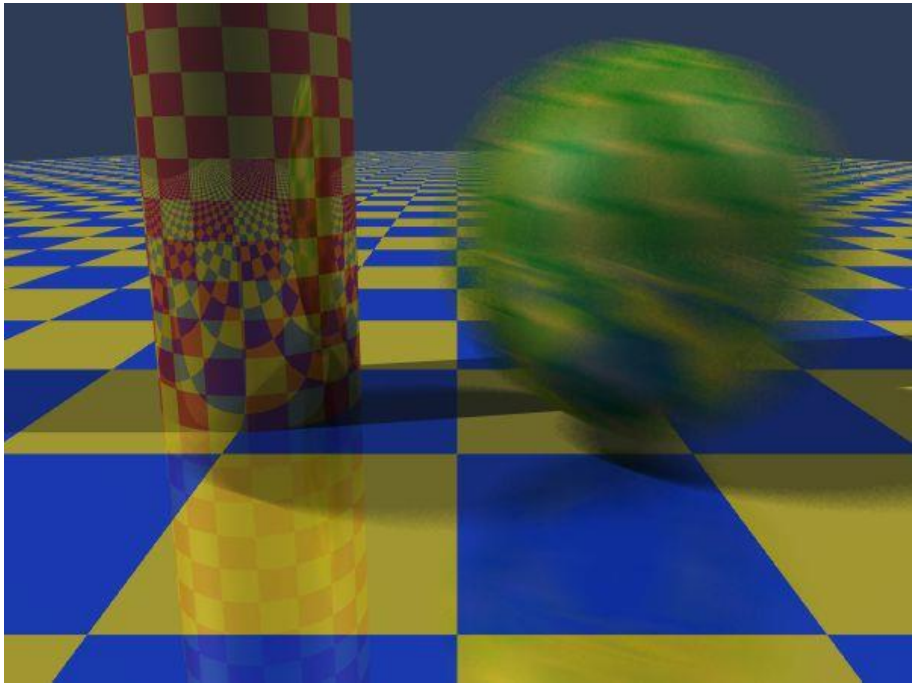
Circle of confusion



- F: Focal length; n: aperture number
- $V_P = \frac{FP}{P-F}$, for P>F, $V_D = \frac{FD}{D-F}$, for D>F
- $C = |V_D - V_P| \frac{F}{nV_D}$

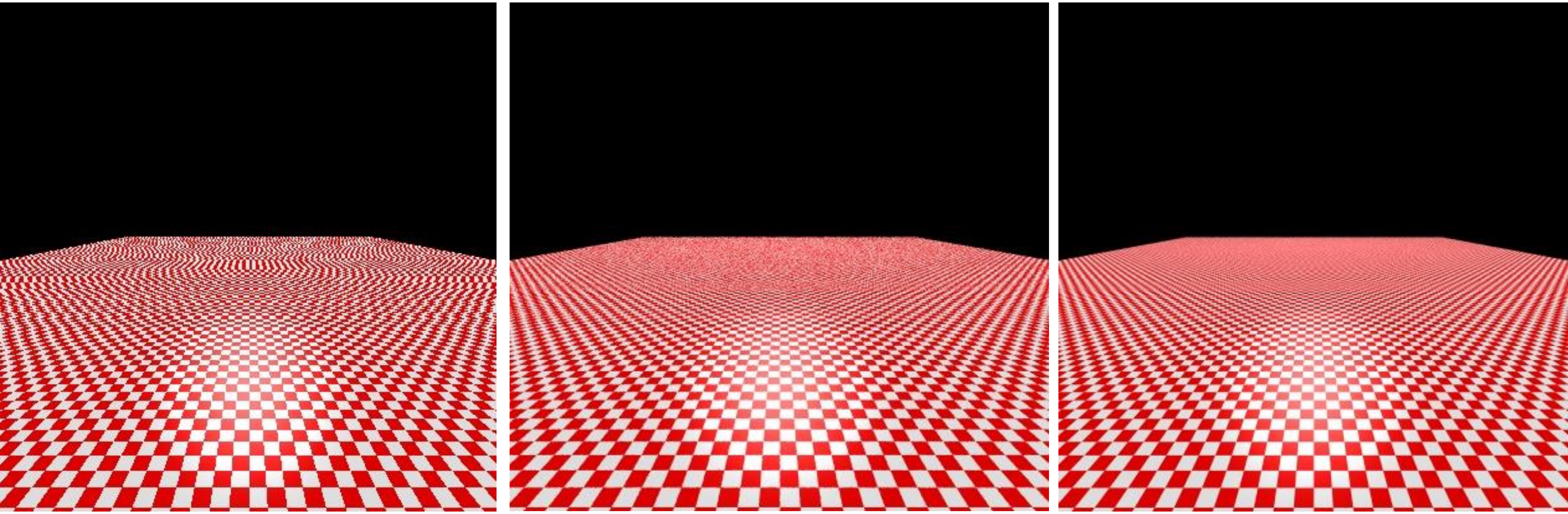

Depth-of-Field
Ollivier de Langlais
31/12/1998

Motion Blur



Slide Courtesy of Roger Crawfis, Ohio State

Supersampling



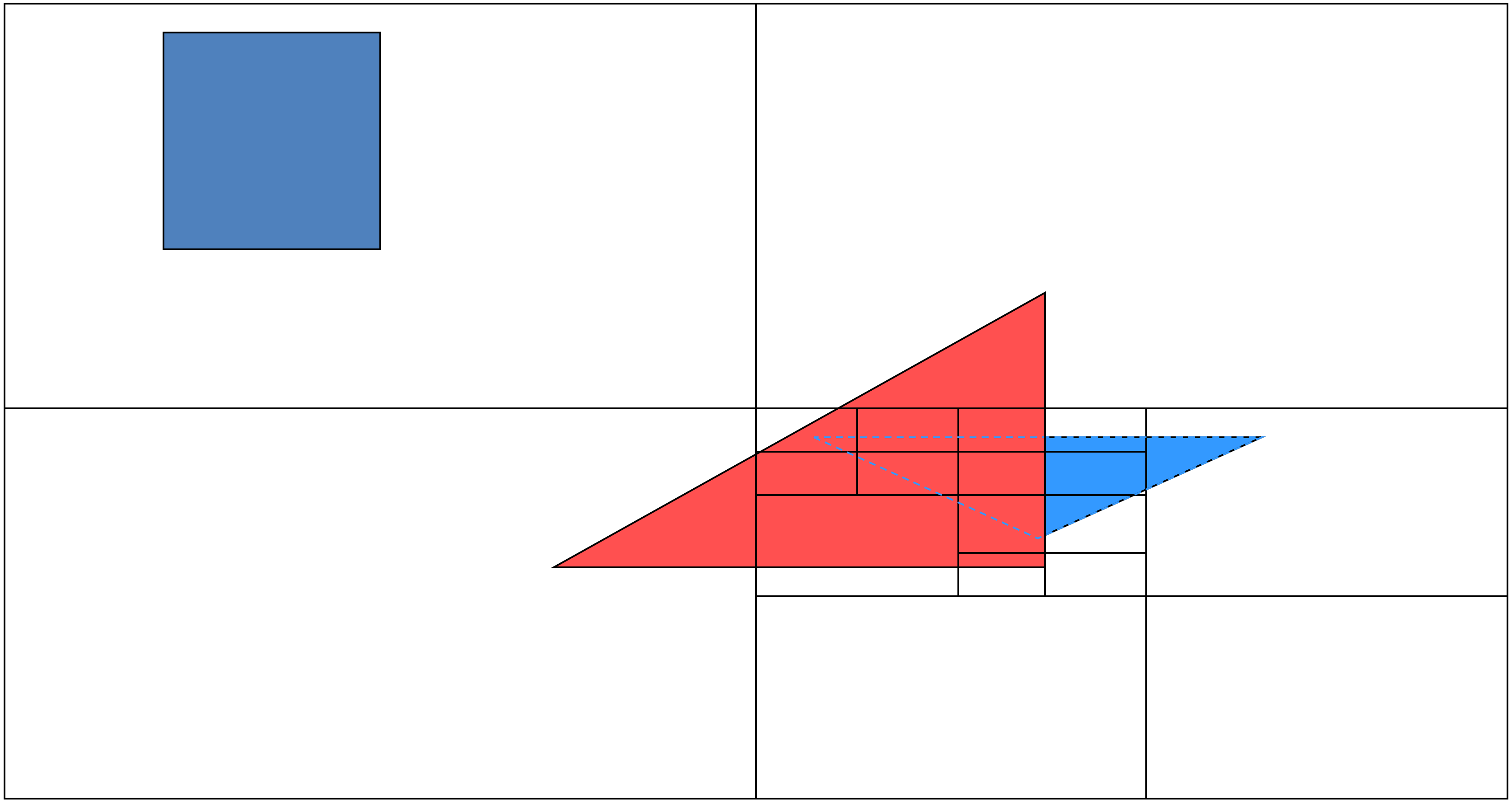
1 sample per pixel   16 sample per pixel   256 sample per pixel

Slide Courtesy of Roger Crawfis, C

More On Ray-Tracing

- Already discussed recursive ray-tracing!
- Improvements to ray-tracing!
  - Area sampling variations to address aliasing
- Distributed ray-tracing!

Area Subdivision (Warnock)
(mixed object/image space)



Clipping used to subdivide polygons that are across regions

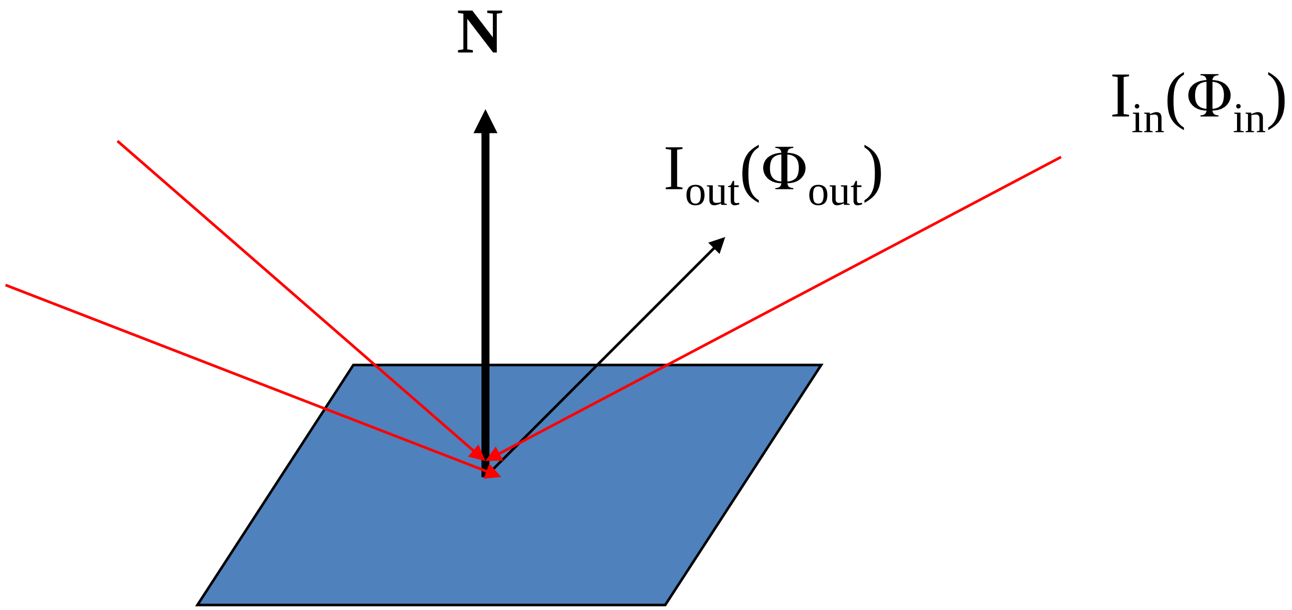
Softwares

- POV-ray (http://www.povray.org/)
  - A free rendering tool (not a modeling tool)
    - Uses a text based scene description language (SDL)
- Blender (http://www.blender3d.org)
  - Modeling, Animation, rendering tool
    - Especially useful in 3D game creation

RENDERING EQUATION

Rendering Equation (Kajiya 1986)

- Consider a point on a surface



Rendering Equation

- Outgoing light is from two sources
  - Emission
  - Reflection of incoming light
- Must integrate over all incoming light
  - Integrate over hemisphere
- Must account for foreshortening of incoming light

Rendering Equation

$$I_{out}(\Phi_{out}) = E(\Phi_{out}) + \int_{2\pi} R_{bd}(\Phi_{out}, \Phi_{in})I_{in}(\Phi_{in}) \cos\theta \, d\omega$$

emission

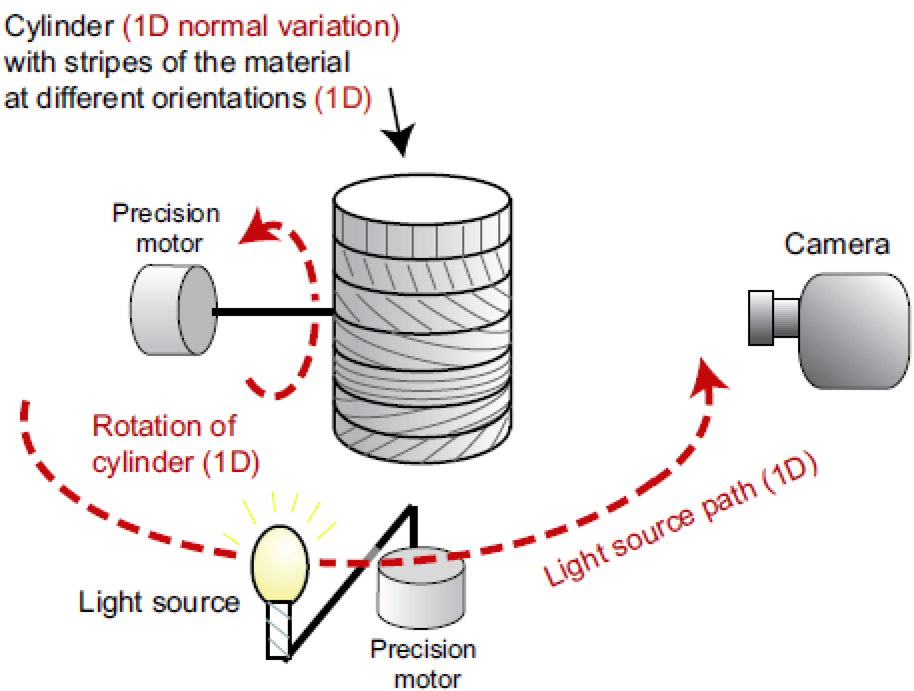
angle between ***Normal*** and ***Φin***

bidirectional reflection coefficient

Note that angle is really two angles in 3D and wavelength is fixed

BRDF database

- http://www.merl.com/brdf/



Cylinder (1D normal variation)
with stripes of the material
at different orientations (1D)
Precision motor
Camera
Rotation of cylinder (1D)
Light source path (1D)
Light source
Precision motor

1-Felt
2-Polyester
3-Terrycloth
4-Rough Plastic
5-Leather
6-Sandpaper
7-Velvet
8-Pebbles
9-Frosted Glass
10-Plaster_a
11-Plaster_b
12-Rough Paper
13-Artificial Grass
14-Roof Shingle
15-Foil
16-Cork
17-Rough Tile
18-Rug_a
19-Rug_b
20-Styrofoam
21-Sponge
22-Lambswool
23-Lettuce Leaf
24-Rabbit Fur
25-Quarry Tile
26-Loofa
27-Insulation
28-Crumpled Paper
29-(2 zoomed)
30-(11 zoomed)
31-(12 zoomed)
32-(14 zoomed)
33-Slate_a
34-Slate_b
35-Spheres
36-Limestone
37-Brick_a
38-Ribbed Paper
39-Human Skin
40-Straw
41-Brick_b
42-Corduroy
43-Salt Crystals
44-Linen
45-Concrete_a
46-Cotton
47-Stones
48-Brown Bread
49-Concrete_b
50-Concrete_c
51-Corn Husk
52-White Bread
53-Plant
54-Wood_a
55-Orange Peel
56-Wood_b
57-Peacock Feather
58-Tree Bark
59-Cracker_a
60-Cracker_b
61-Moss

Rendering Equation

- Rendering equation is an energy balance
  - Energy in = energy out
- Integrate over hemisphere
- Fredholm integral equation
  - Cannot be solved analytically in general
- Various approximations of R_{bd} give standard rendering models
- Should also add an occlusion term in front of right side to account for other objects blocking light from reaching surface

Another version

Consider light at a point **p** arriving from **p**'

$$i(\mathbf{p}, \mathbf{p}') = \boxed{\upsilon(\mathbf{p}, \mathbf{p}')}\left(\varepsilon(\mathbf{p},\mathbf{p}') + \int \rho(\mathbf{p}, \mathbf{p}', \mathbf{p}'')\, i(\mathbf{p}', \mathbf{p}'')\, d\mathbf{p}''\right)$$

occlusion = 0
or
attenuation $=1/d^2$

emission from **p**' to **p**

light reflected at **p**' from all points **p**'' towards **p**