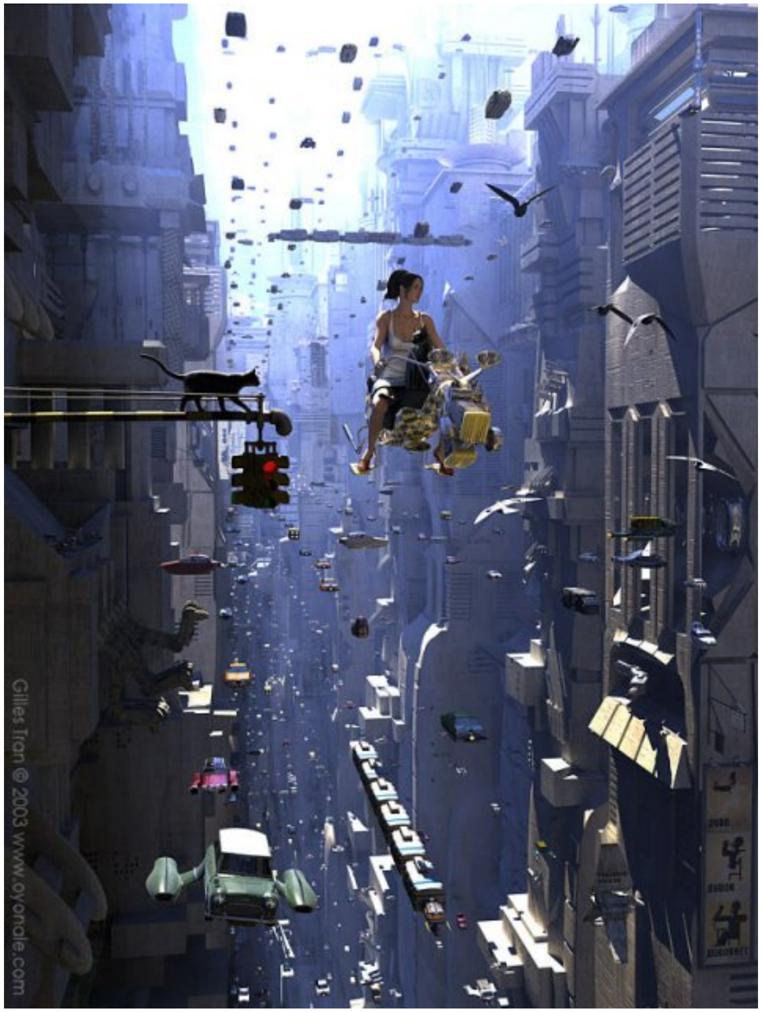
# Ray Tracing Algorithm

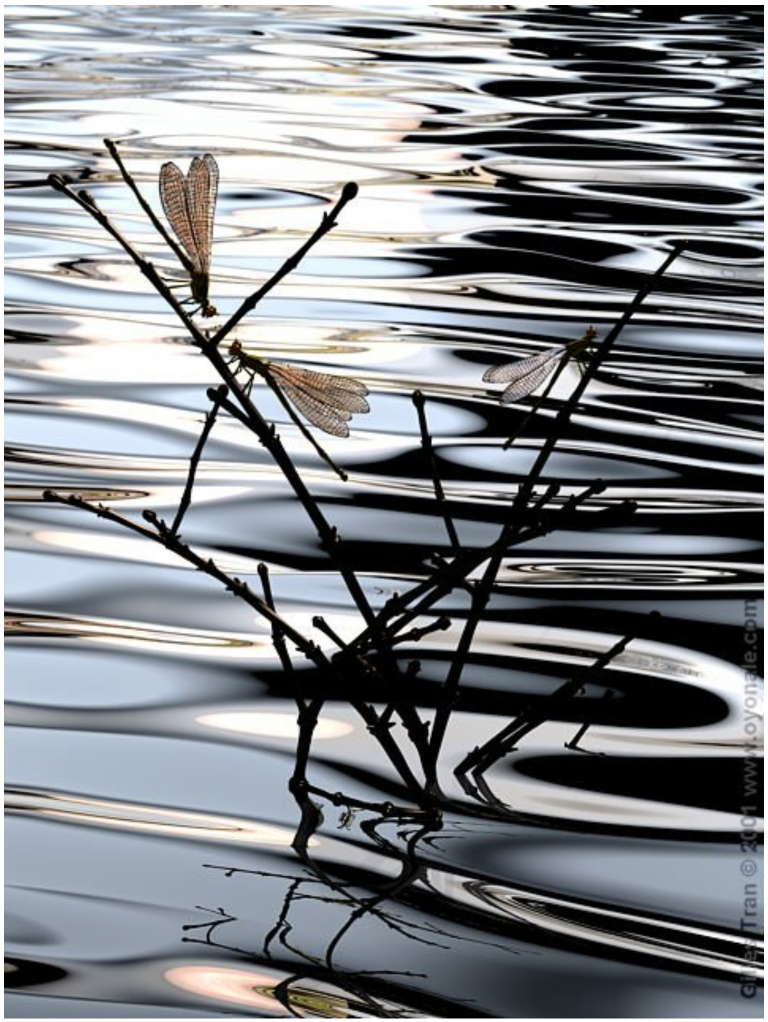
# Ray tracing (Picture from Povray.org)

# Ray tracing (Picture from Povray.org)

# Ray tracing (Picture from Povray.org)




# Ray tracing (Picture from Povray.org)

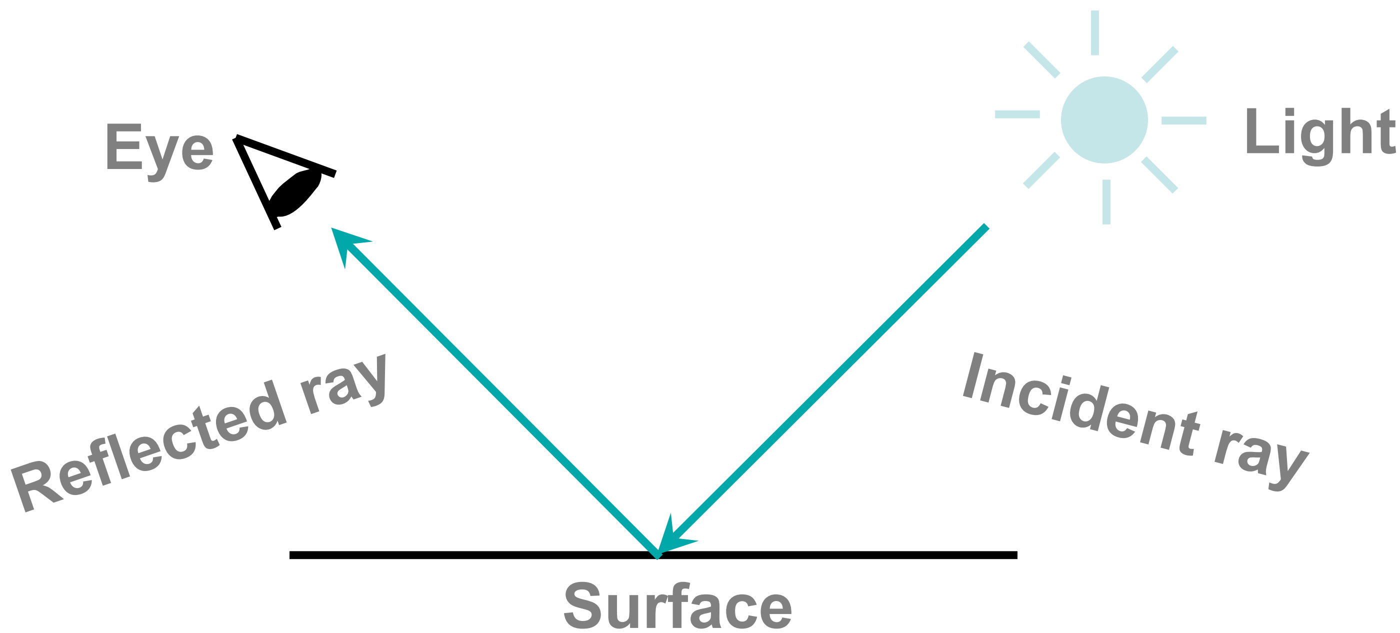


# Ray tracing (Picture from Povray.org)

# Ray tracing (Picture from Povray.org)

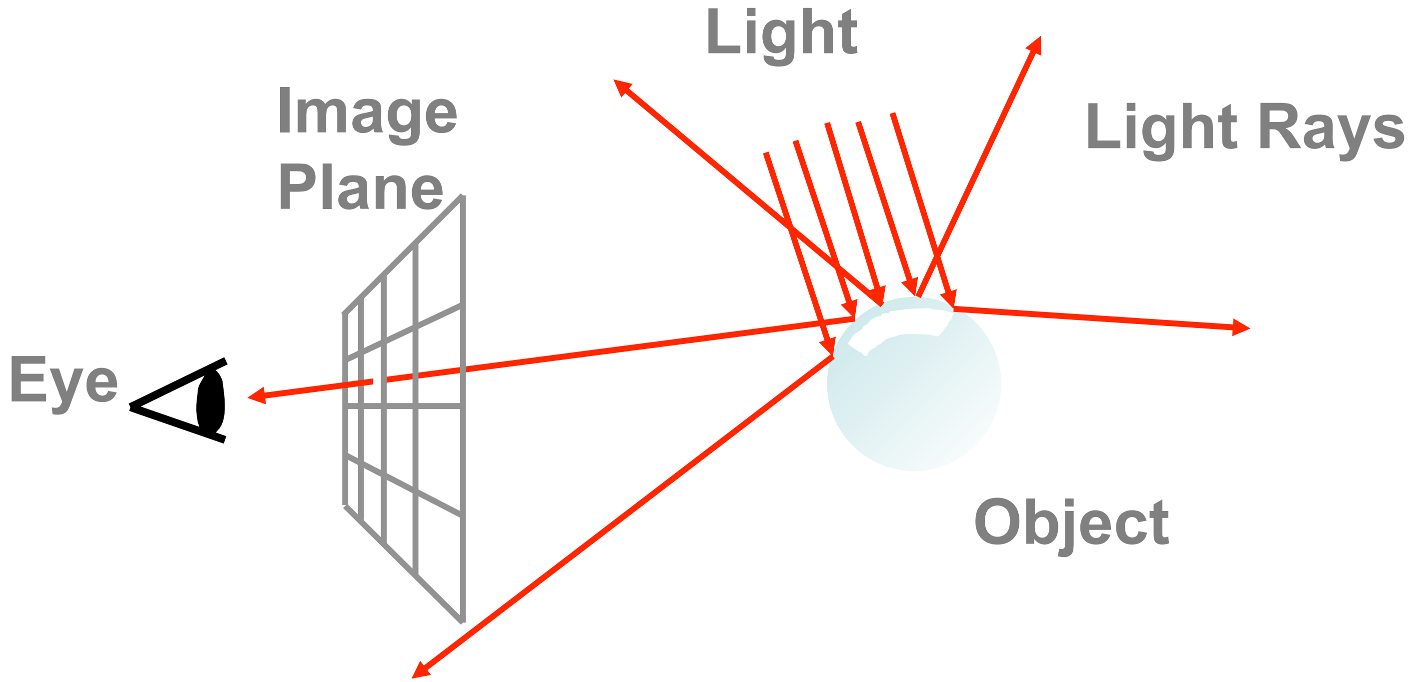

# The Basic Idea

- Simulate light rays from light source to eye



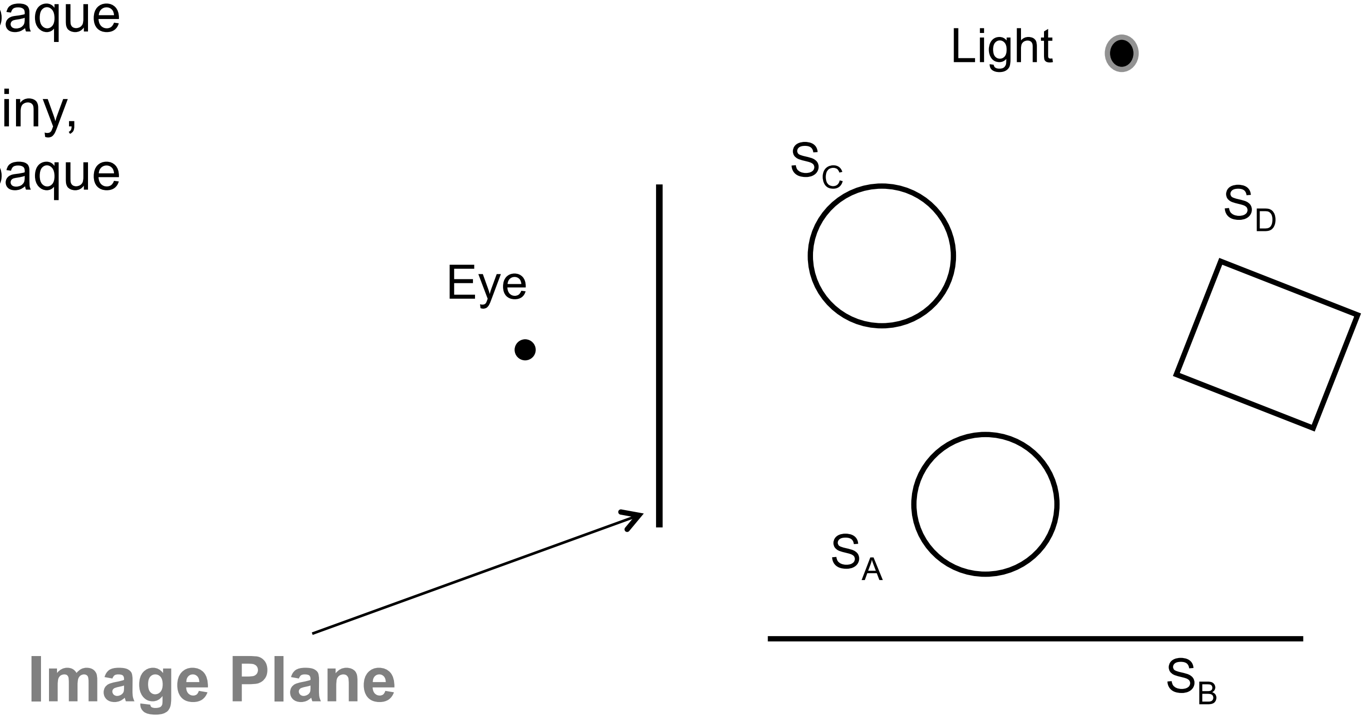
# “Forward” Ray-Tracing

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



# Scene

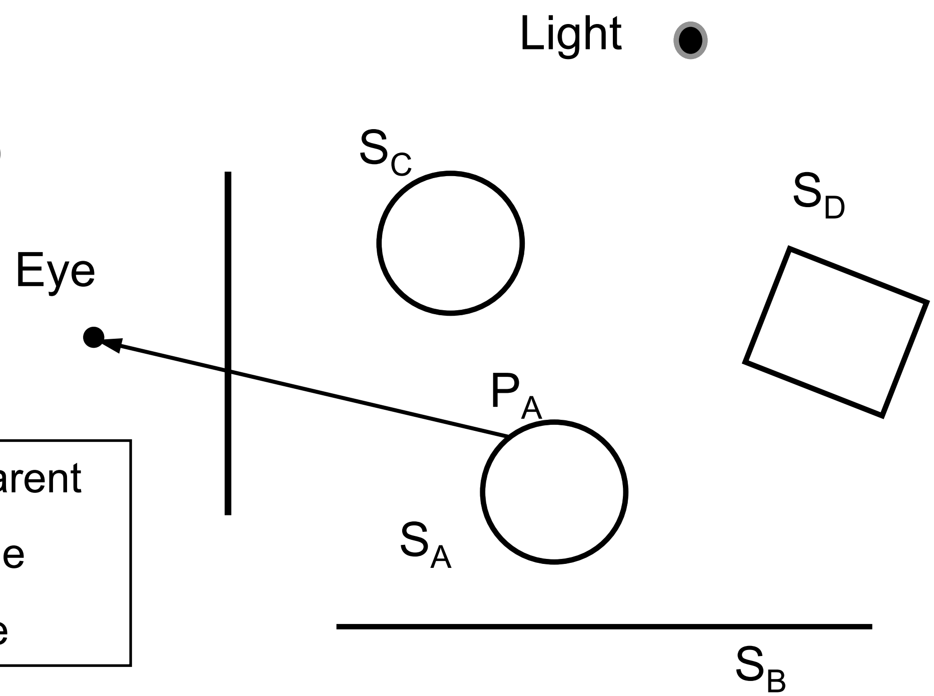
| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse, opaque |
| $S_C$ | shiny, opaque |



# Three sources of light

The light that point $P_A$ emits to the eye comes from:

light sources
other objects (reflection)
other objects (refraction)



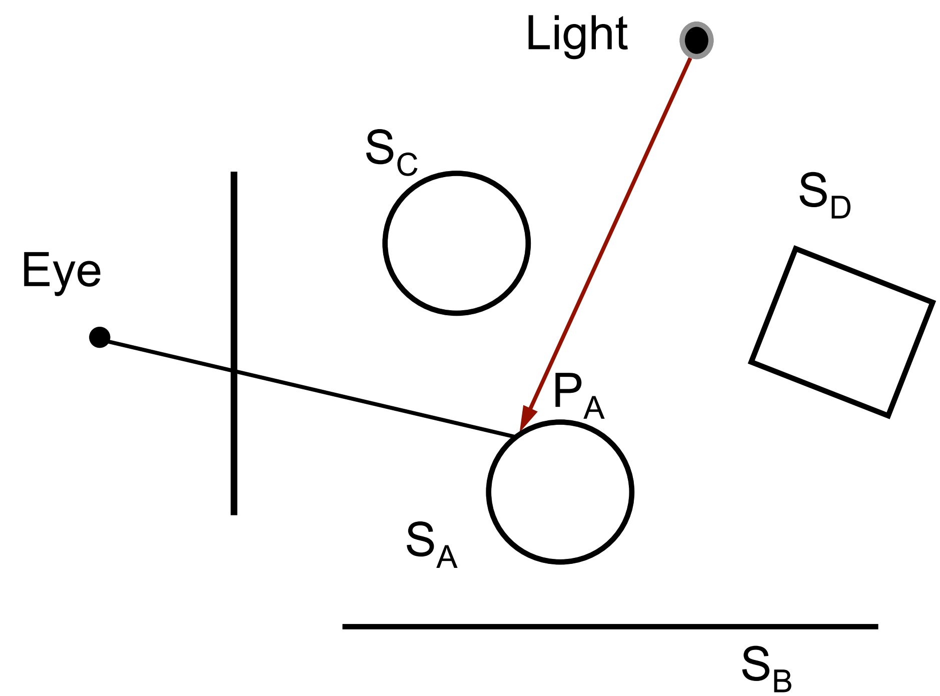

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |

# Directly from light source

Local illumination model:

$$I = I_a + I_{diff} + I_{spec}$$

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



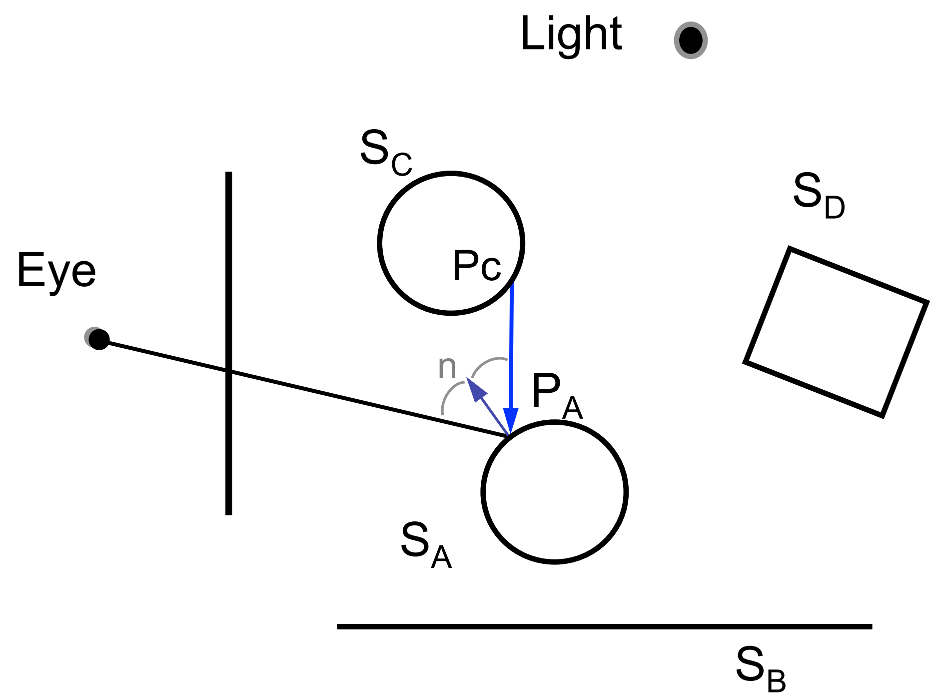
# Reflection

What is the color that is reflected to $P_A$ ?

The color of $P_C$.

What is the color of $P_C$ ?

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



# Reflection

What is the light that is reflected to $P_A$ ?

The color of $P_C$ . as viewed by $P_A$

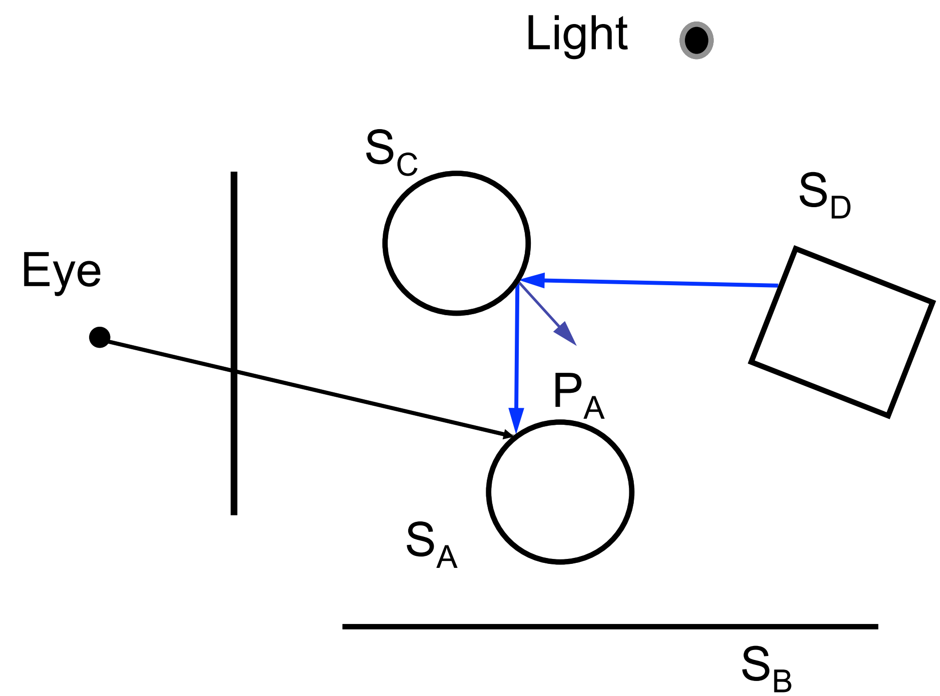
What is the color of $P_C$ reflected towards $P_A$?

Just like $P_A$ :

raytrace $P_C$ i.e compute the three contributions from

1. Light sources
2. Reflection
3. refraction

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



# Refraction

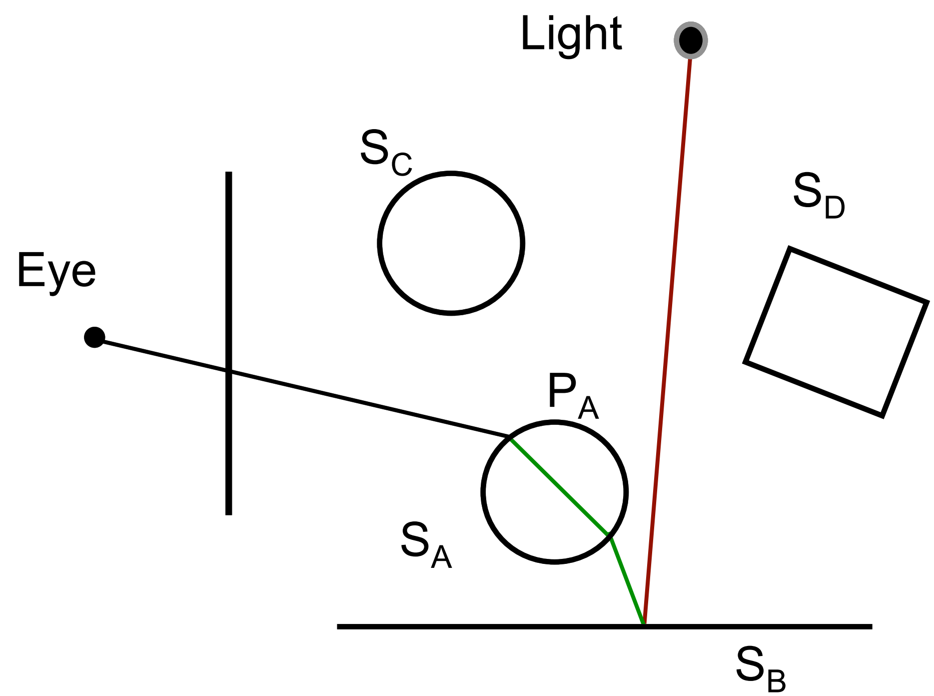
Transparent materials

How do you compute the refracted contribution?

You raytrace the refracted ray.

1. Lights
2. Reflection
3. Refraction

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



# What are we missing?

- **Diffuse objects do not receive light from other objects.**

# Three sources of light together

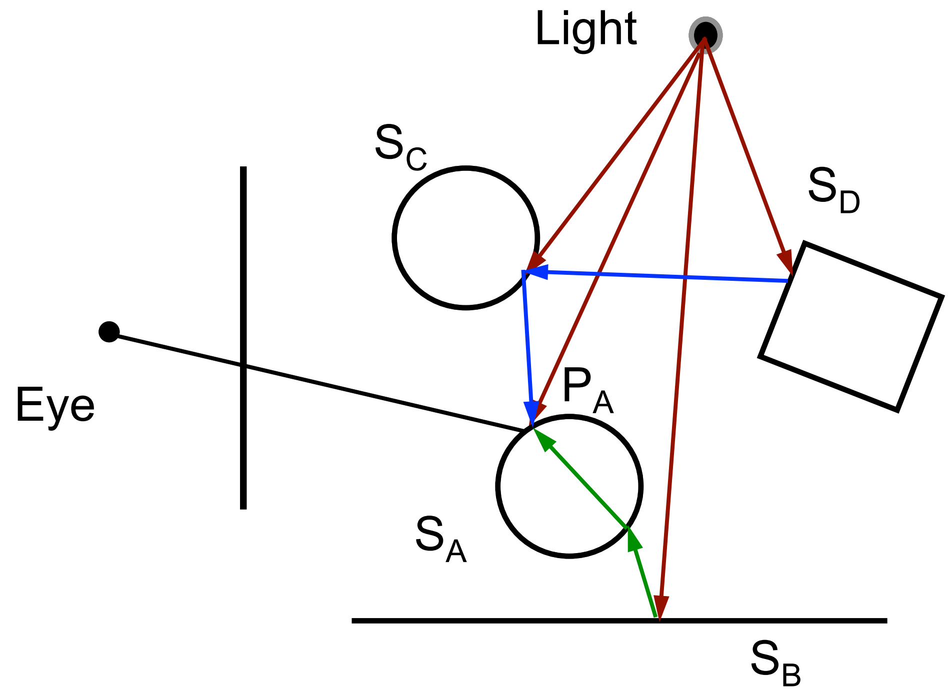
The color that the pixel is assigned comes from:

*light sources*

*other objects (reflection)*

*other objects (refraction)*

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B, S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



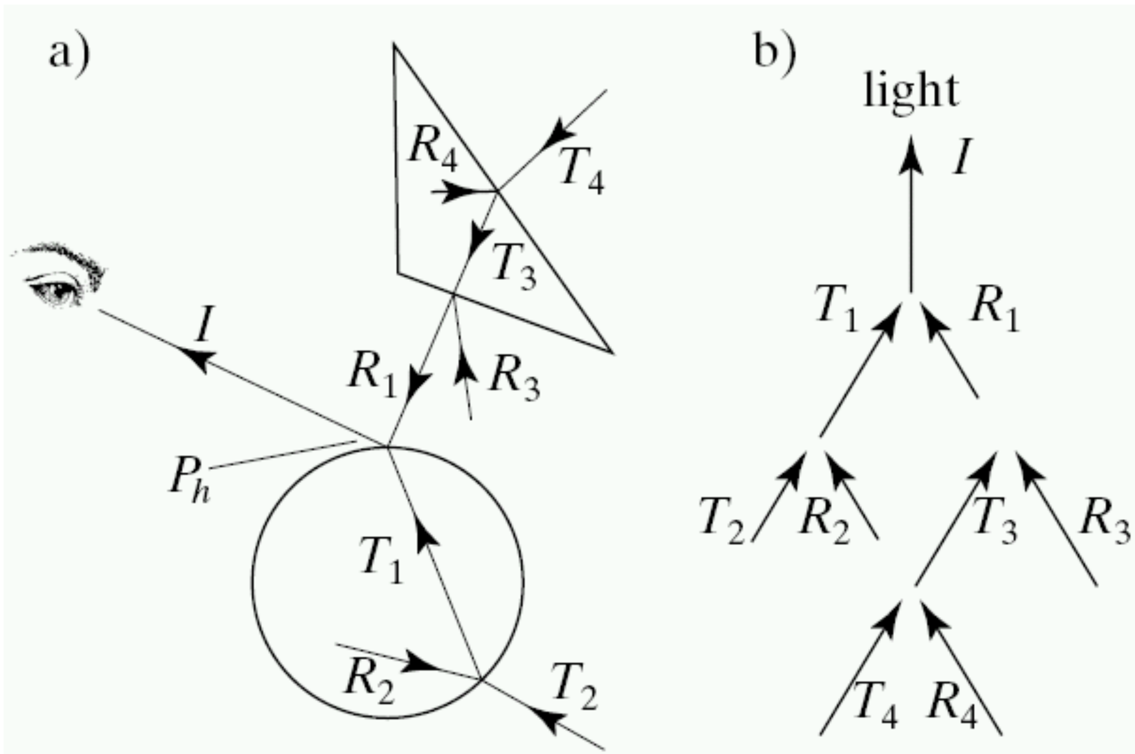
# Backwards Raytracing Algoritm

- **For each pixel construct a ray: eye→ pixel**

```
raytrace( ray )
  P = closest intersection
  color_local = ShadowRay(light1, P)+…
                + ShadowRay(lightN, P)
  color_reflect = raytrace(reflected_ray )
  color_refract = raytrace(refracted_ray )
  color = color_local
   + k_re*color_reflect
   + k_ra*color_refract
return( color )
```

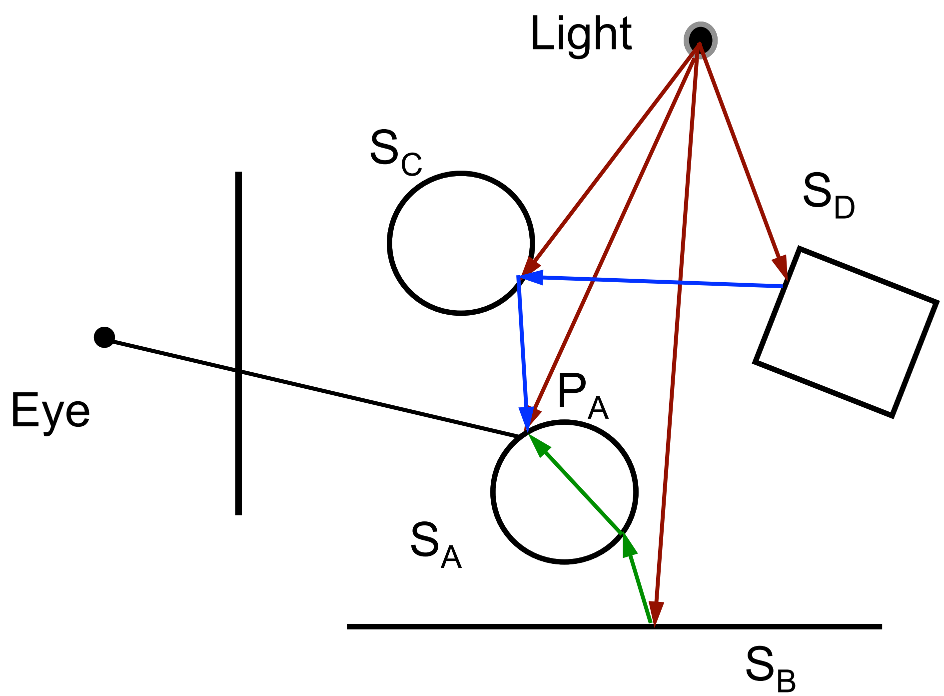
**A recursive function!**

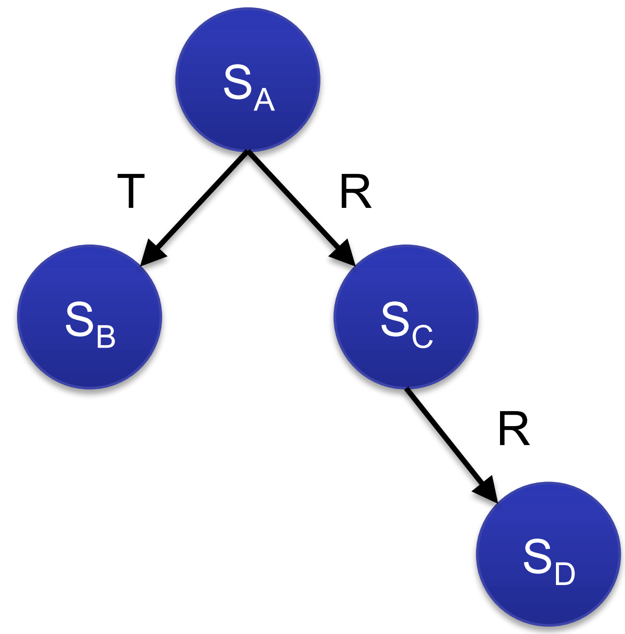
# Tree of Rays



# Tree of Rays

| | |
|---|---|
| $S_A$ | shiny, transparent |
| $S_B$,$S_D$ | diffuse,opaque |
| $S_C$ | shiny, opaque |



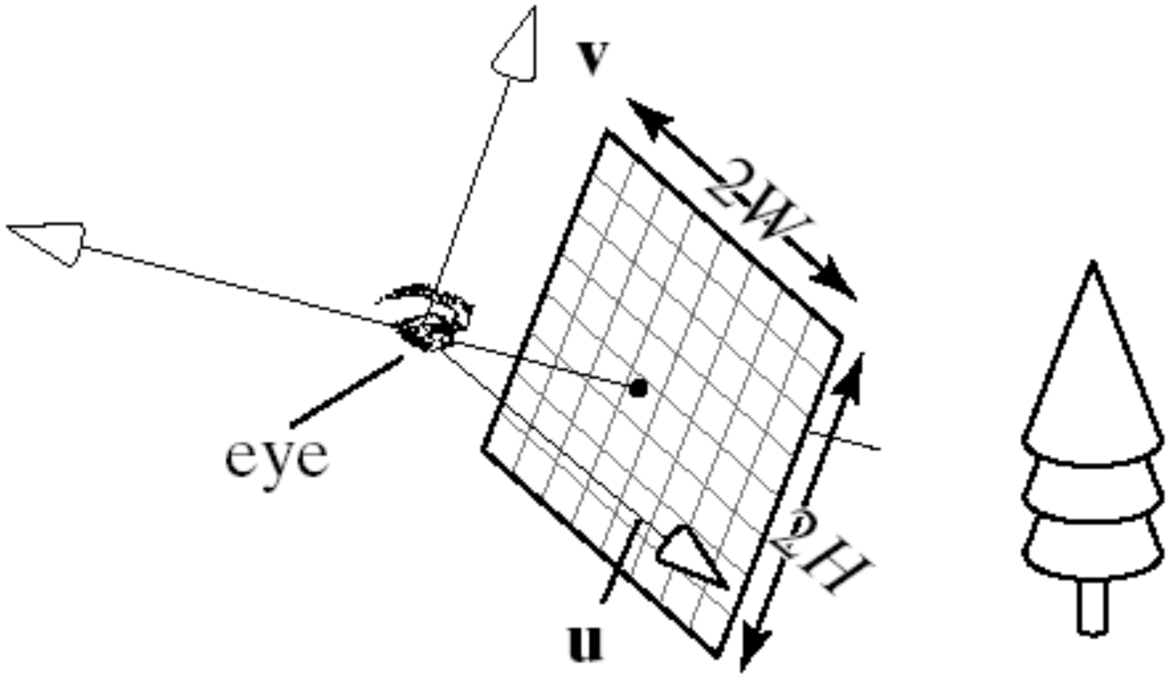
# How many levels of recursion do we use?

- The more the better.
- Infinite reflections at the limit.

## Stages of raytracing

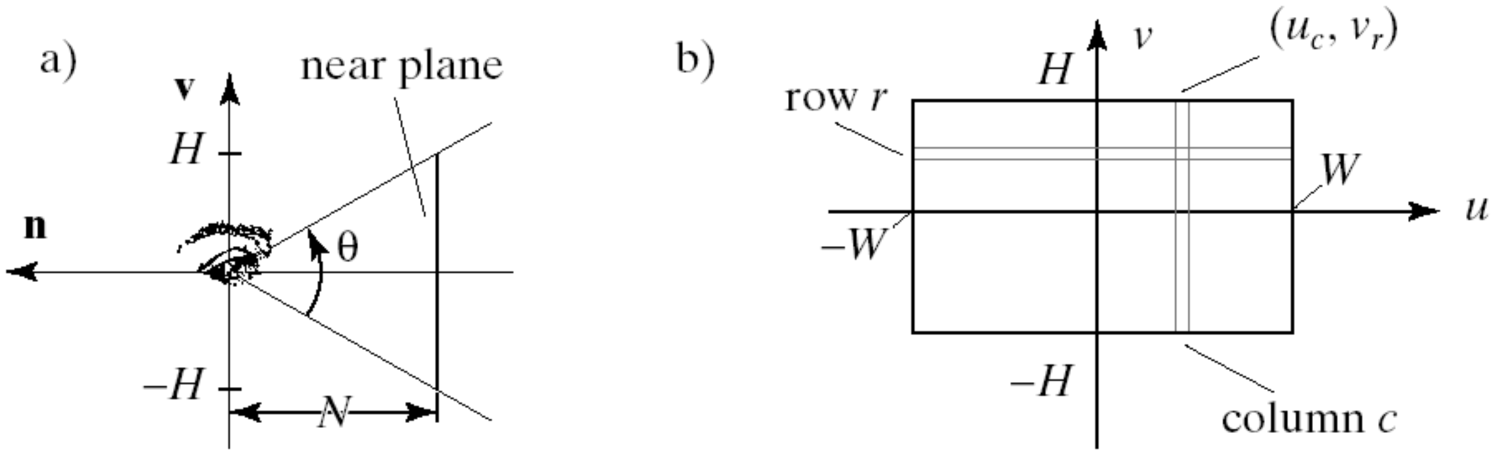
- Setting the camera and the image plane
- Computing a ray from the eye to every pixel and trace it in the scene
- Object-ray intersections
- Shadow, reflected and refracted ray at each intersection

# Setting up the camera



# Image parameters

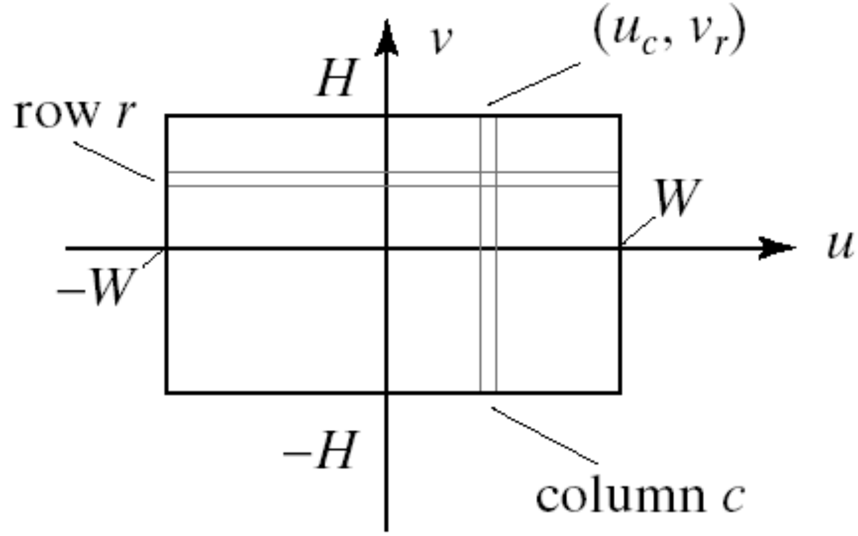
- Width 2W, Height 2H
  Number of pixels nCols x nRows
- Camera coordinate system (eye, u,v,n)
- Image plane at -N



# Pixel coordinates in camera coordinate system

- **Pixel P(r,c) has coordinates in camera space:**

$$u_c = -W + W\frac{2c}{nCols}, \quad c = 0, 1, \ldots, nCols - 1,$$
$$v_r = -H + H\frac{2r}{nRows}, \quad r = 0, 1, \ldots, nRows - 1,$$



# Ray through pixel

- **Pixel location**

$$Camera\ \ coordinates: \quad P(r,c) = (u_c, v_r, -N)$$

$$Wolrd\ \ coordinates: \quad P(r,c) = eye - N\mathbf{n} + u_c\mathbf{u} + v_r\mathbf{v}$$

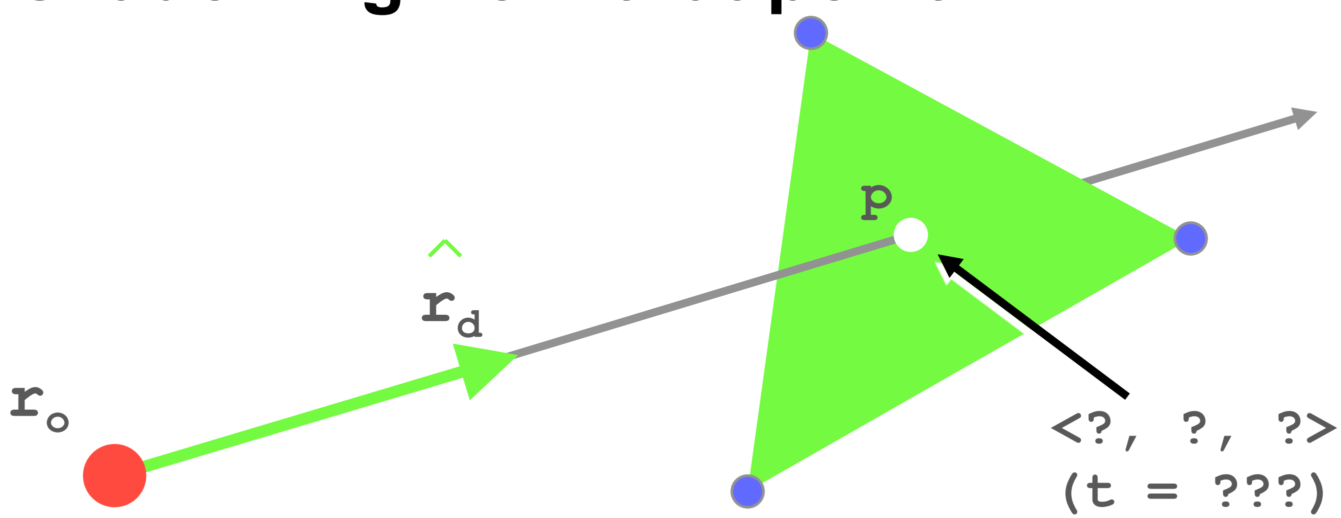
- **Ray through pixel:**

$$ray(r,c,t) = eye + t(P(r,c) - eye)$$

$$ray(r,c,t) = eye + t(-N\mathbf{n} + w(\frac{2c}{nCols} - 1)\mathbf{u} + H(\frac{2r}{nRows} - 1)\mathbf{v})$$

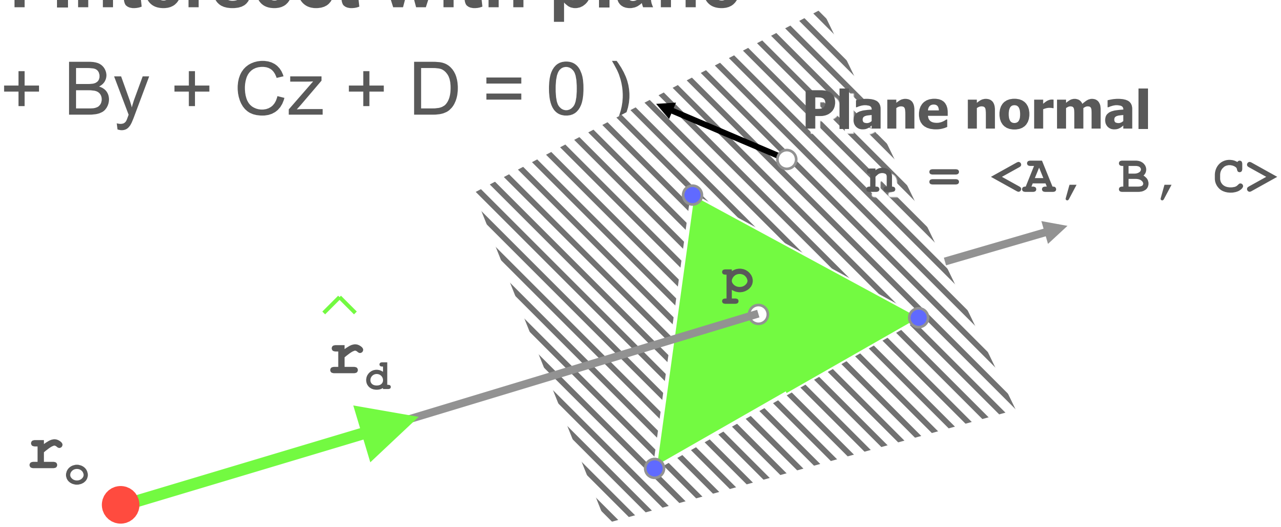
# Triangle Intersection

- Want to know: at what point (p) does ray intersect triangle?
- Compute lighting, reflected rays, shadowing from that point



# Triangle Intersection

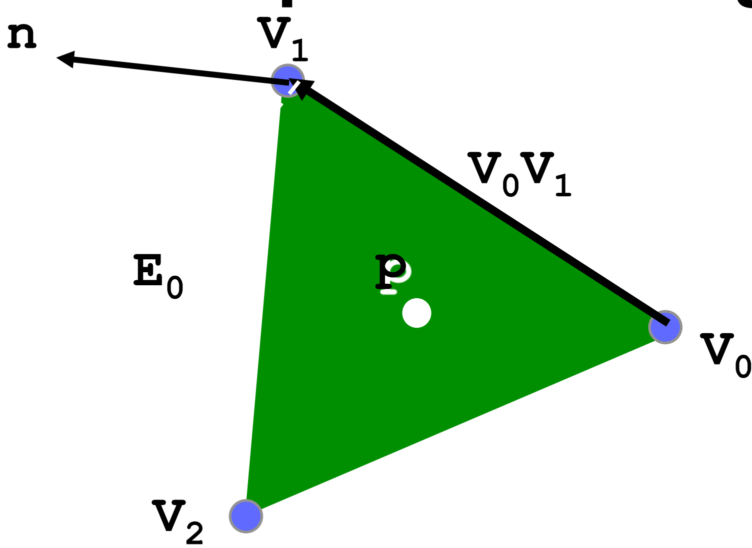
- Step 1 : Intersect with plane
  - ( Ax + By + Cz + D = 0 )




$$p = -(n.\ r_o + D)\ /\ (n.\ r_d\ )$$

# Triangle Intersection

- **Step 2 : Check against triangle edges**




$E_i = V_iV_{i+1} \times n$ (plane A, B, C)

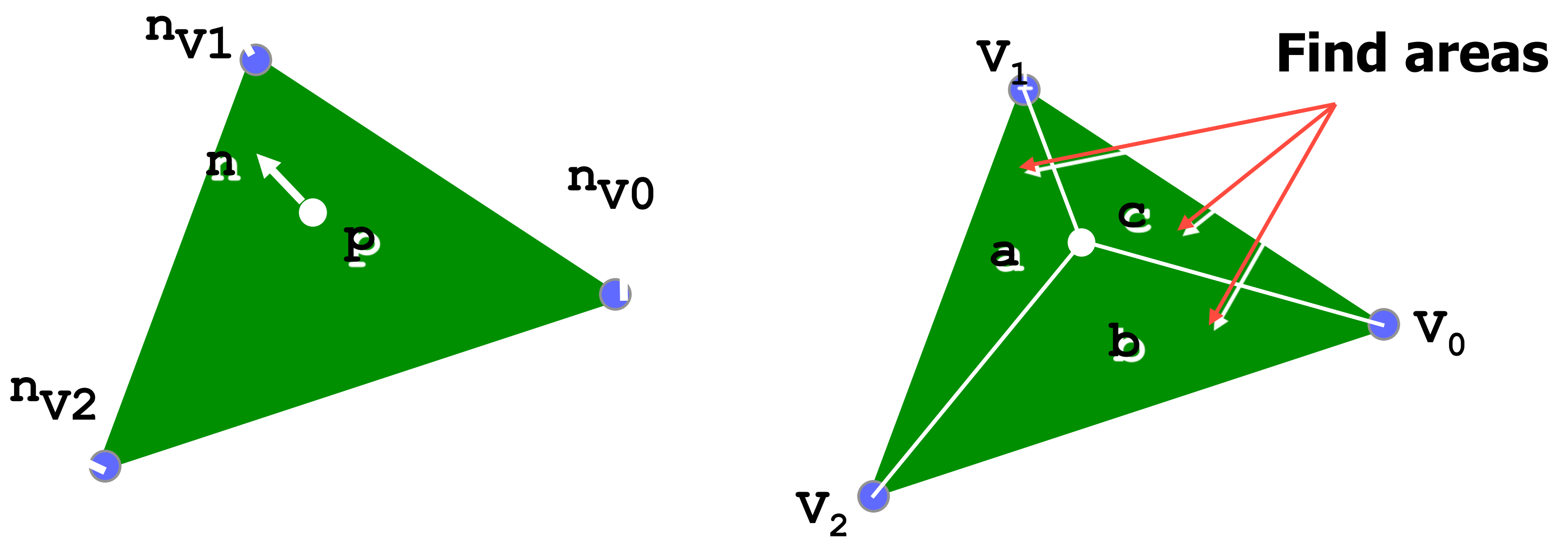
$d_i = -A \cdot N$ (plane D)

**Plug p into ($p \cdot E_i + d_i$) for each edge**

**if signs are all positive or negative, point is inside triangle!**

# Triangle Normals

- Could use plane normals (flat shading)
- Better to interpolate from vertices

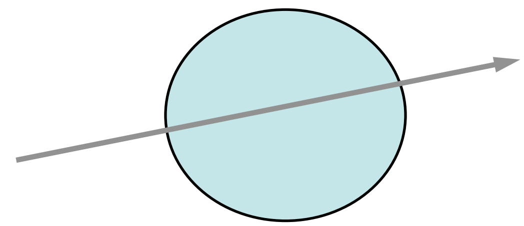



$$n = \frac{a n_{v0} + b n_{v1} + c n_{v2}}{\text{area}(v_0 v_1 v_2)}$$

# Ray-object intersections

- **Unit sphere at origin - ray intersection:**

$$ray(t) = S + \mathbf{c}t$$
$$Sphere(P) = |P| - 1 = 0$$



$$Sphere(ray(t)) = 0 \Rightarrow$$
$$|S + \mathbf{c}t| - 1 = 0 \Rightarrow (S + \mathbf{c}t)(S + \mathbf{c}t) - 1 = 0 \Rightarrow$$
$$\boxed{|\mathbf{c}|^2 t^2 + 2(S \cdot \mathbf{c})t + |S|^2 - 1 = 0}$$

- **That's a quadratic equation**

# Solving a quadratic equation

$$|\mathbf{c}|^2 t^2 + 2(S \cdot \mathbf{c})t + |S|^2 - 1 = 0$$
$$At^2 + 2Bt + C = 0$$

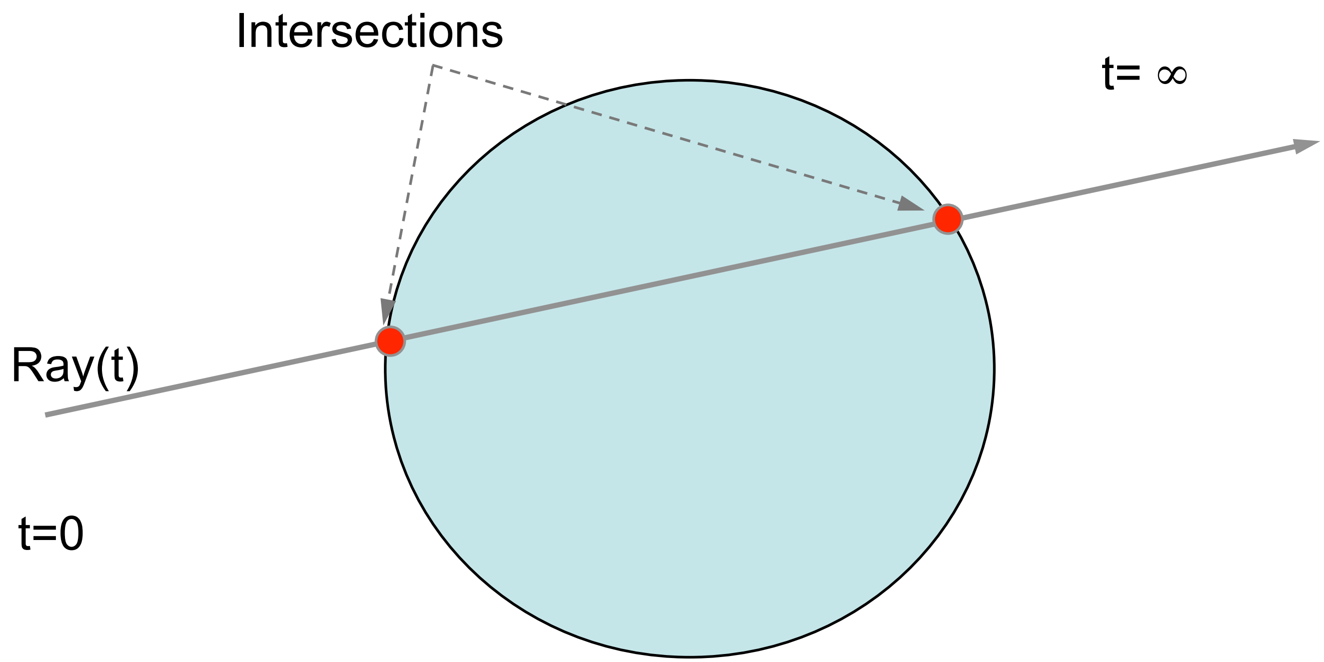
$$t_h = -\frac{B}{A} \pm \frac{\sqrt{B^2 - AC}}{A}$$
$$t_h = -\frac{S \cdot \mathbf{c}}{|\mathbf{c}|^2} \pm \frac{\sqrt{(S \cdot \mathbf{c})^2 - |\mathbf{c}|^2 (|S|^2 - 1)}}{|\mathbf{c}|^2}$$

If $(B^2 - AC) = 0$ one solution

If $(B^2 - AC) < 0$ no solution

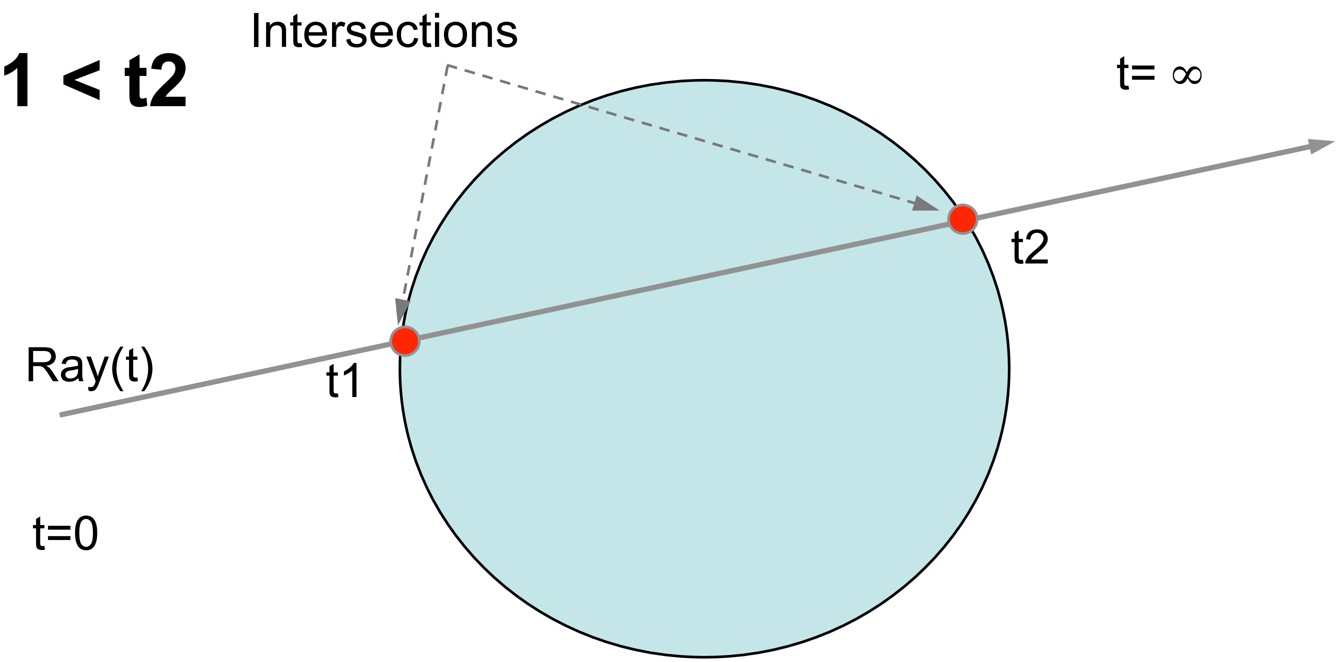
If $(B^2 - AC) > 0$ two solutions

# First intersection?

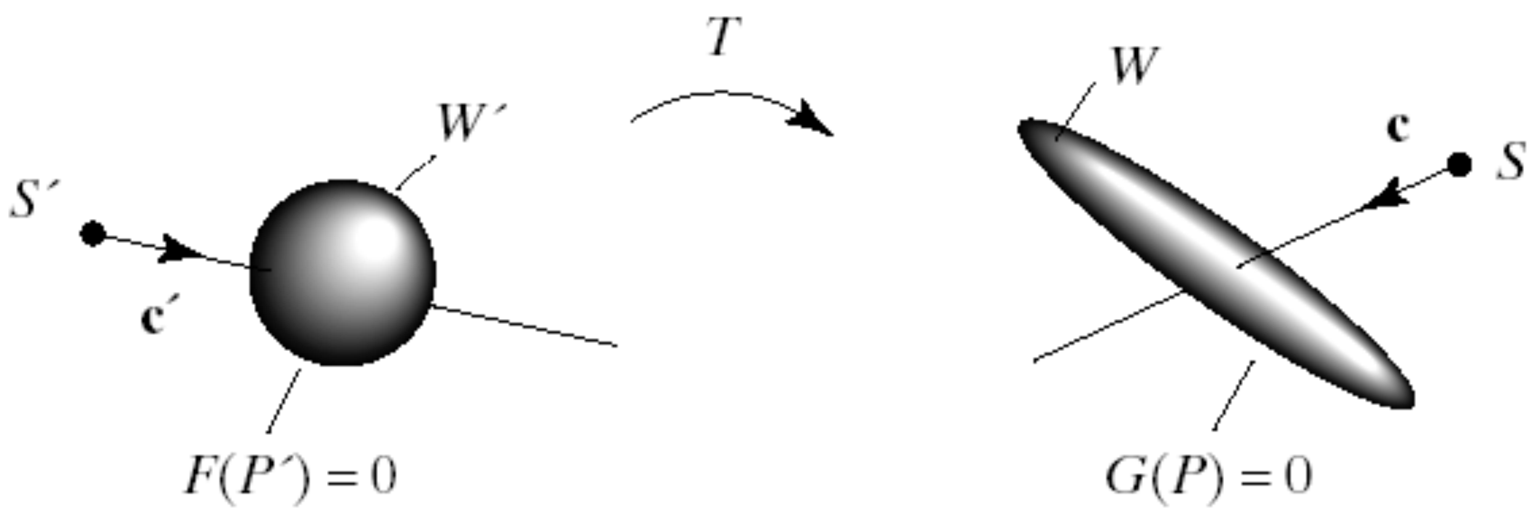


# First intersection?

- **t1 < t2**

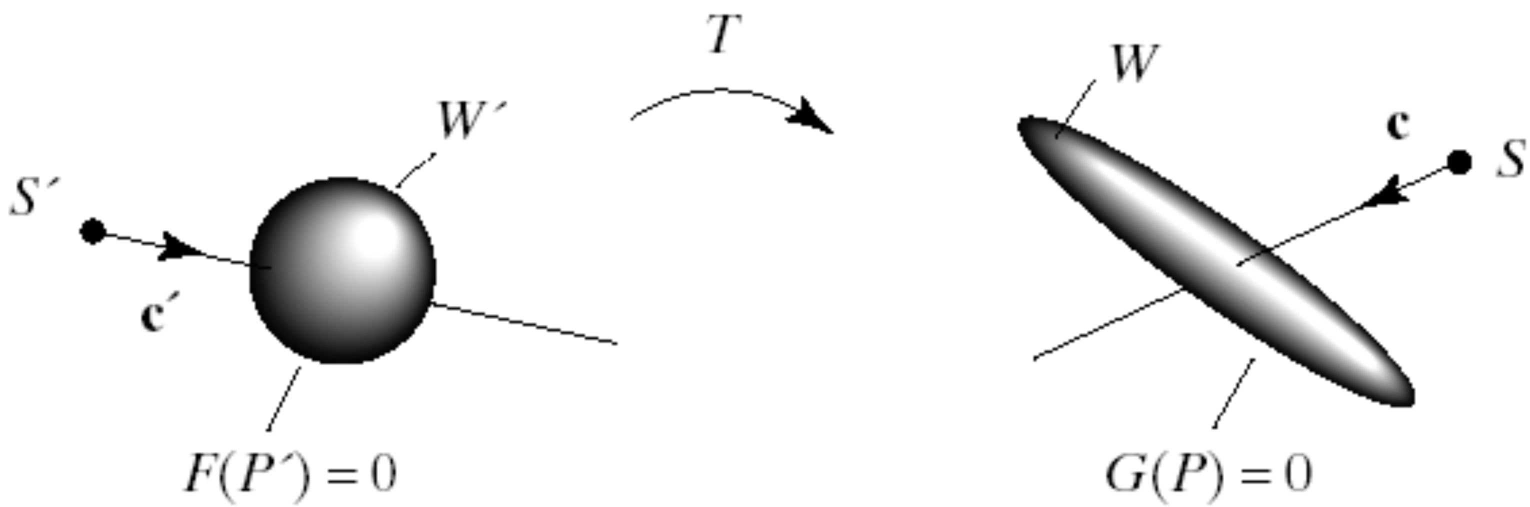


# Transformed primitives?




- **Where does S+ct hit the transformed sphere G ?**

# Linear transformation

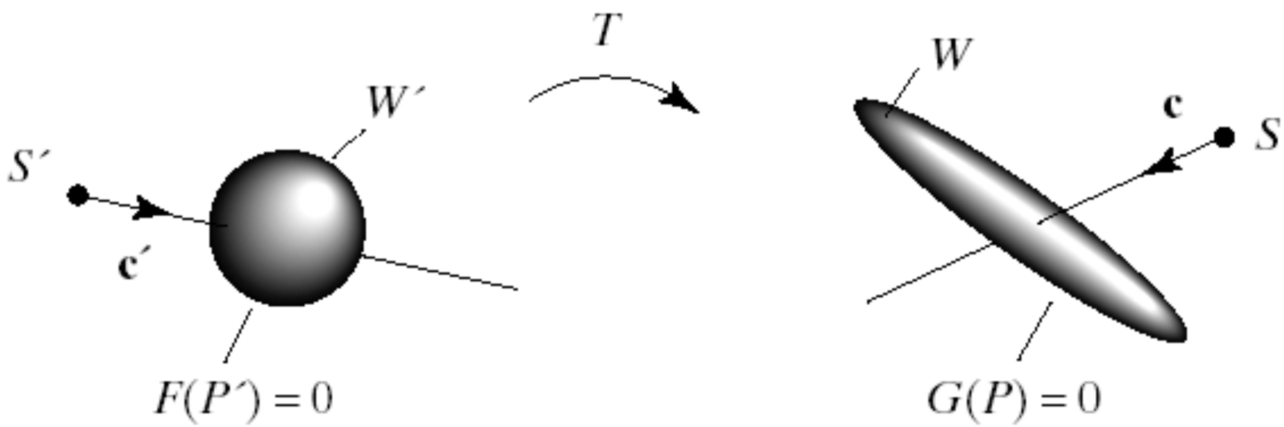



Implicit equation $G(P) = 0$.

Untransformed implicit equation F(P') = 0.

$$P = MP' \Rightarrow P' = M^{-1}P$$

# Linear transformation




$$P = MP' \Rightarrow P' = M^{-1}P$$

$$F(P') = F(T^{-1}(P)) = 0 \Rightarrow F(T^{-1}(P)) = 0$$
$$F(T^{-1}(S + \mathbf{c}t)) = 0 \Rightarrow$$
$$F(T^{-1}(S) + T^{-1}(\mathbf{c}t)) = 0$$

Which means that we can intersect the inverse transformed ray with the untransformed primitive.

# Final Intersection

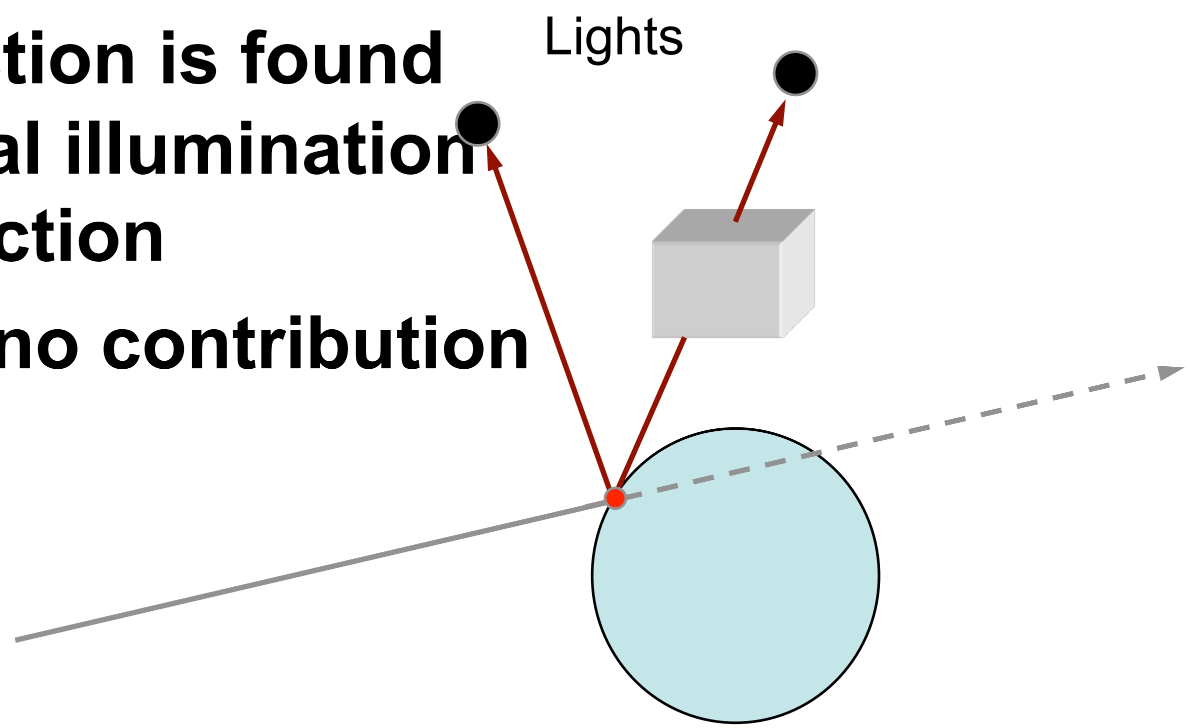
- **Inverse transformed ray**

$$\tilde{r}(t) = M^{-1}\begin{pmatrix} S_x \\ S_y \\ S_z \\ 1 \end{pmatrix} + M^{-1}\begin{pmatrix} c_x \\ c_y \\ c_z \\ 0 \end{pmatrix} = \tilde{S}' + \tilde{\mathbf{c}}'t$$

  - Drop 1 and 0 to get S'+c't
- **For each object**
  - Inverse transform ray getting S'+c't
  - Find intersection $t_{hit}$
  - Use $t_{hit}$ in the untransformed ray S+ct to find the intersection

# Shadow ray

- For each light intersect shadow ray with all objects.
- If no intersection is found apply local illumination at intersection
- If in shadow no contribution



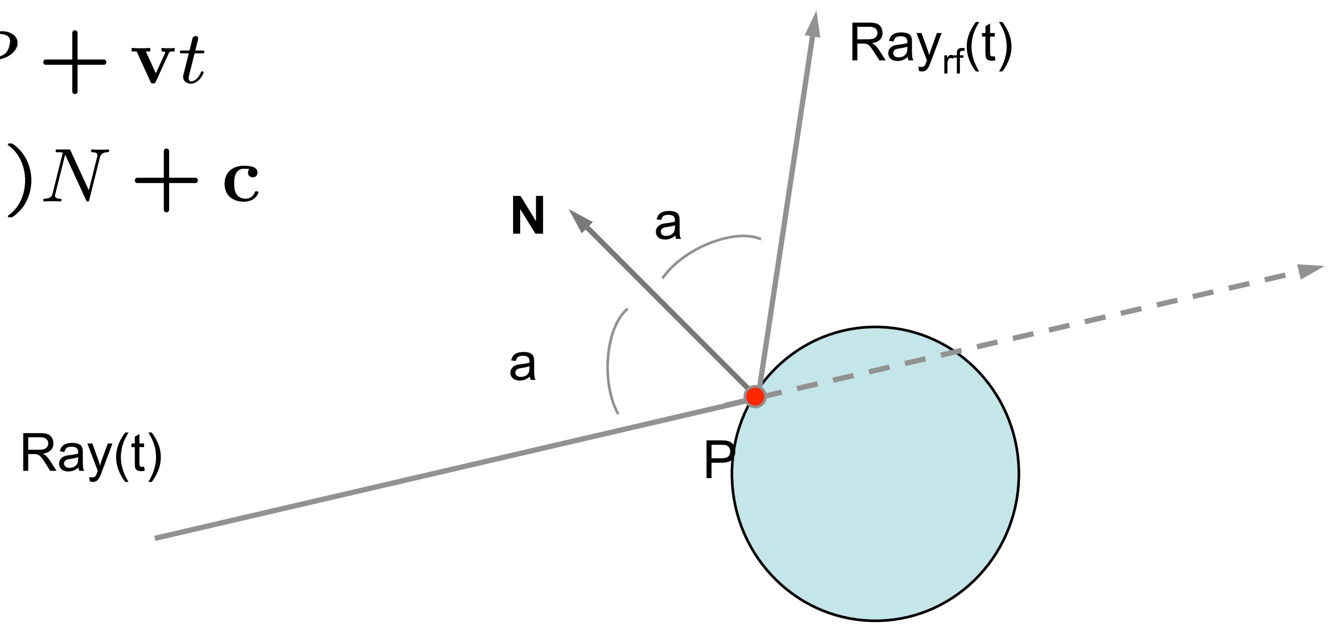
# Reflected ray

- **Raytrace the reflected ray**

$$Ray(t) = A + \mathbf{c}t$$

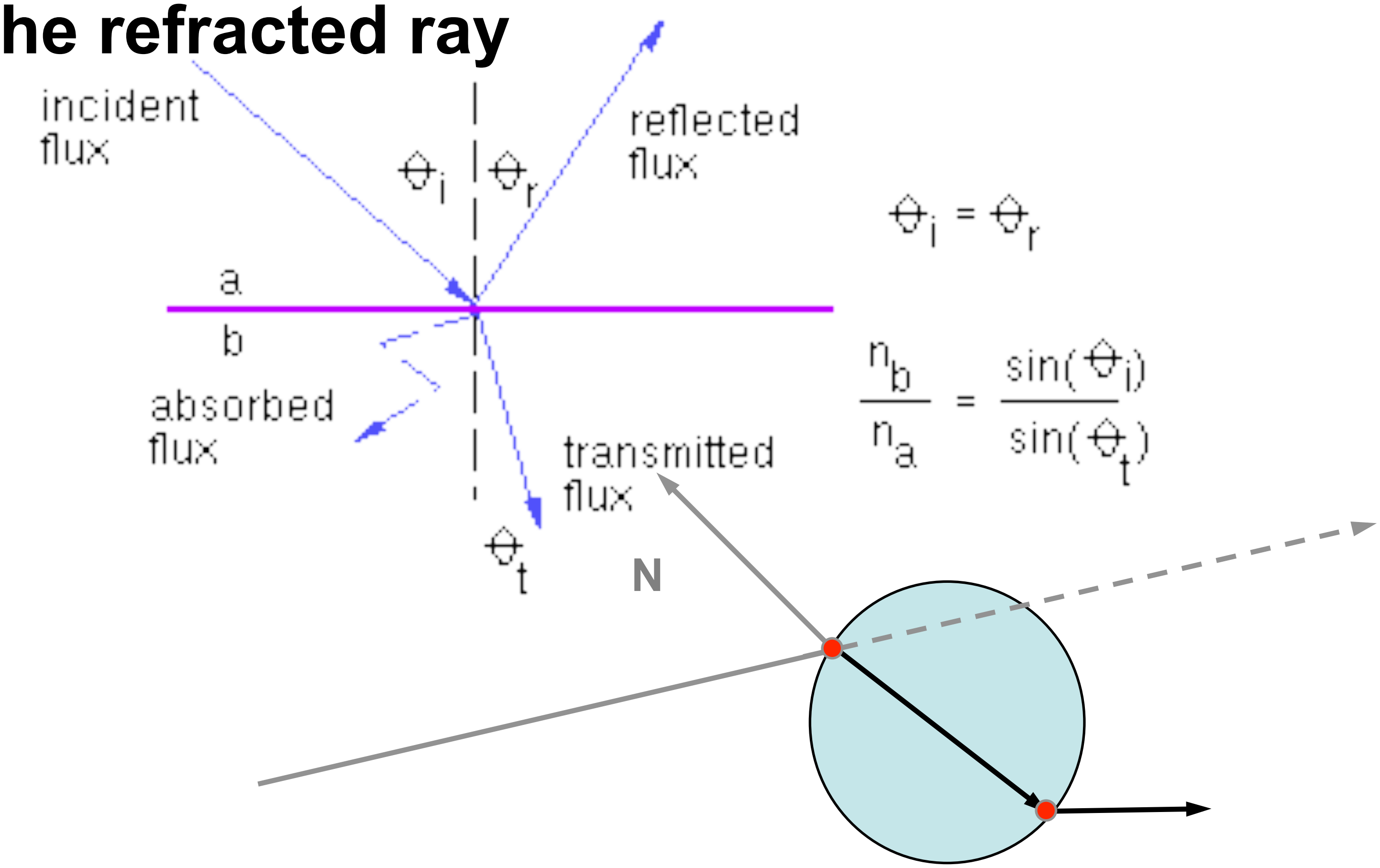
$$Ray_{rf}(t) = P + \mathbf{v}t$$

$$\mathbf{v} = -2(N \cdot \mathbf{c})N + \mathbf{c}$$



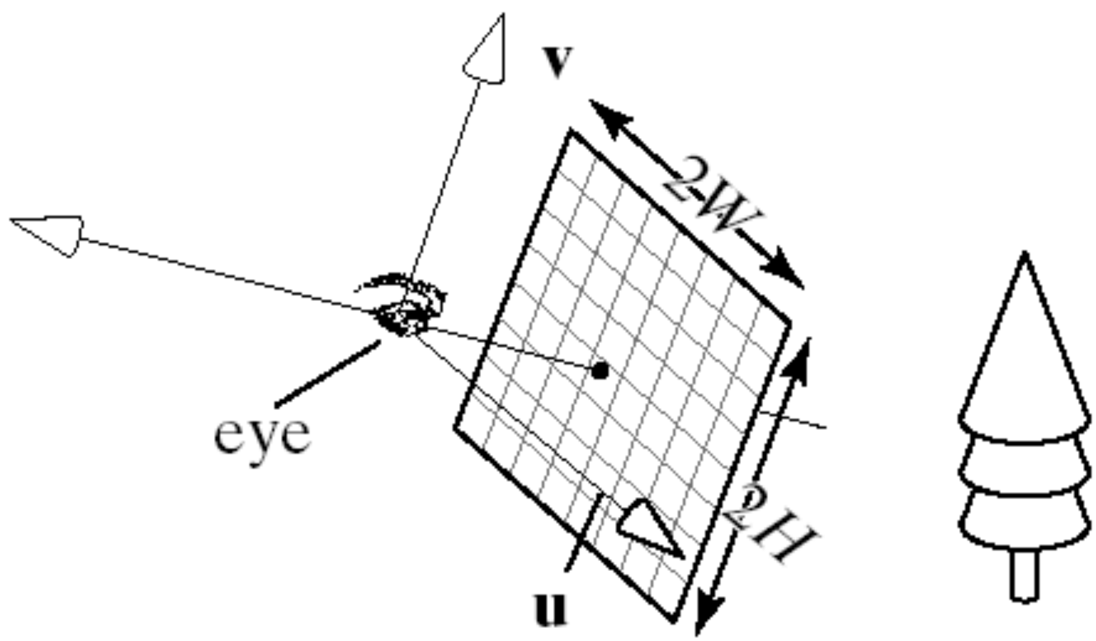
# Refracted ray

- **Raytrace the refracted ray**
  - Snell's law



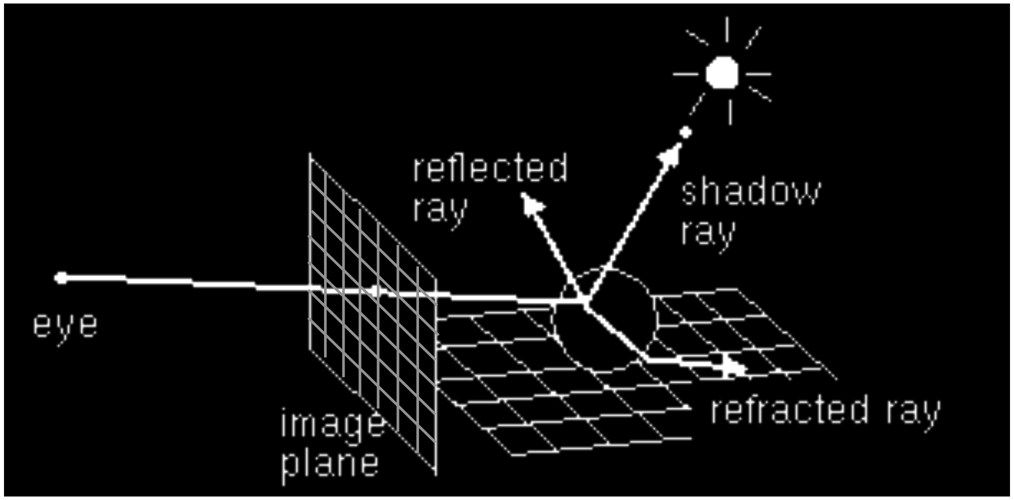
# Add all together

- color(r,c) = color_shadow_ray + $K_f$*$color_{rf}$ + $K_r$*$color_{rfa}$



# Raytracing

```
for each pixel on screen
  1. determine ray from eye through pixel
  2. find closest intersection of ray with an object
  3. cast off reflected and refracted ray, recursively
  4. calculate pixel color, draw pixel
end
```



# Acceleration

1280x1024 image with 10 rays/pixel

1000 objects (triangle, CSG, NURBS)

3 levels recursion

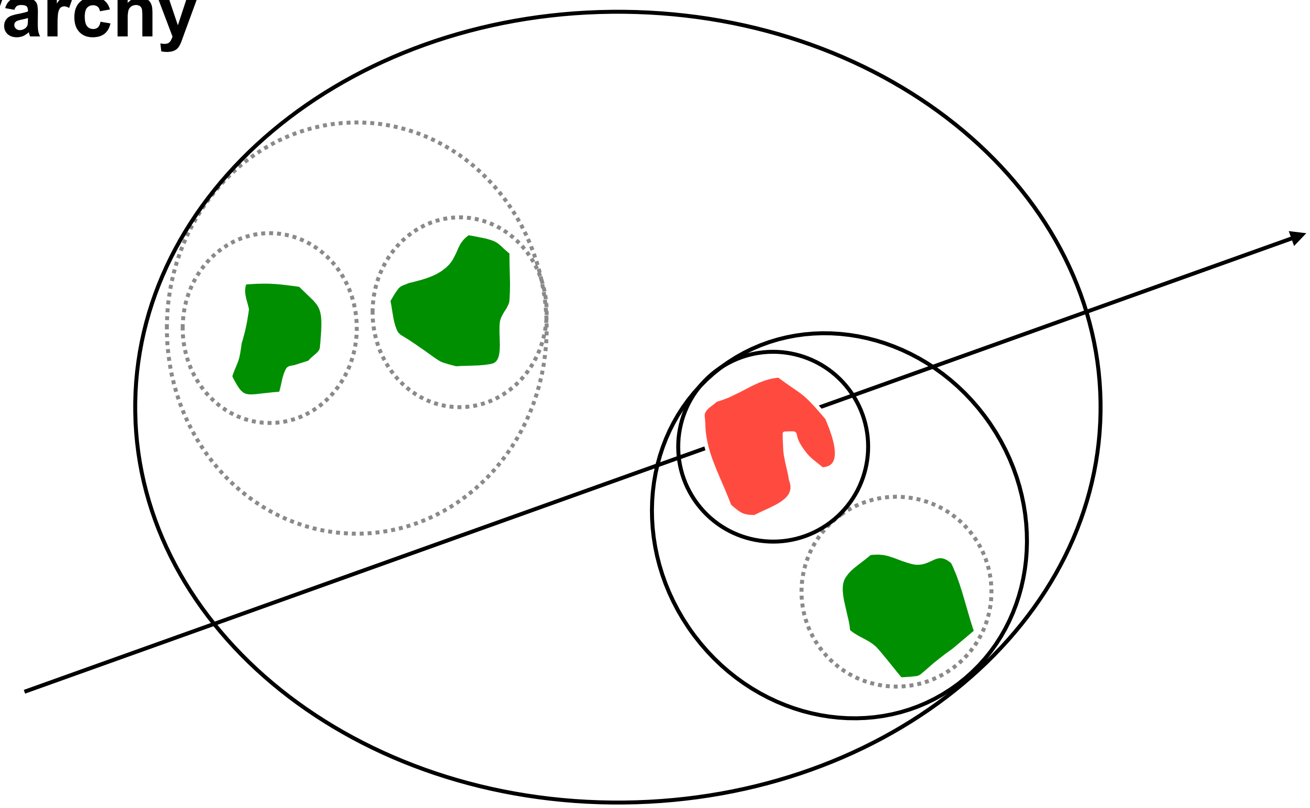
39321600000 intersection tests

100000 tests/second -> 109 days!

Must use an acceleration method!

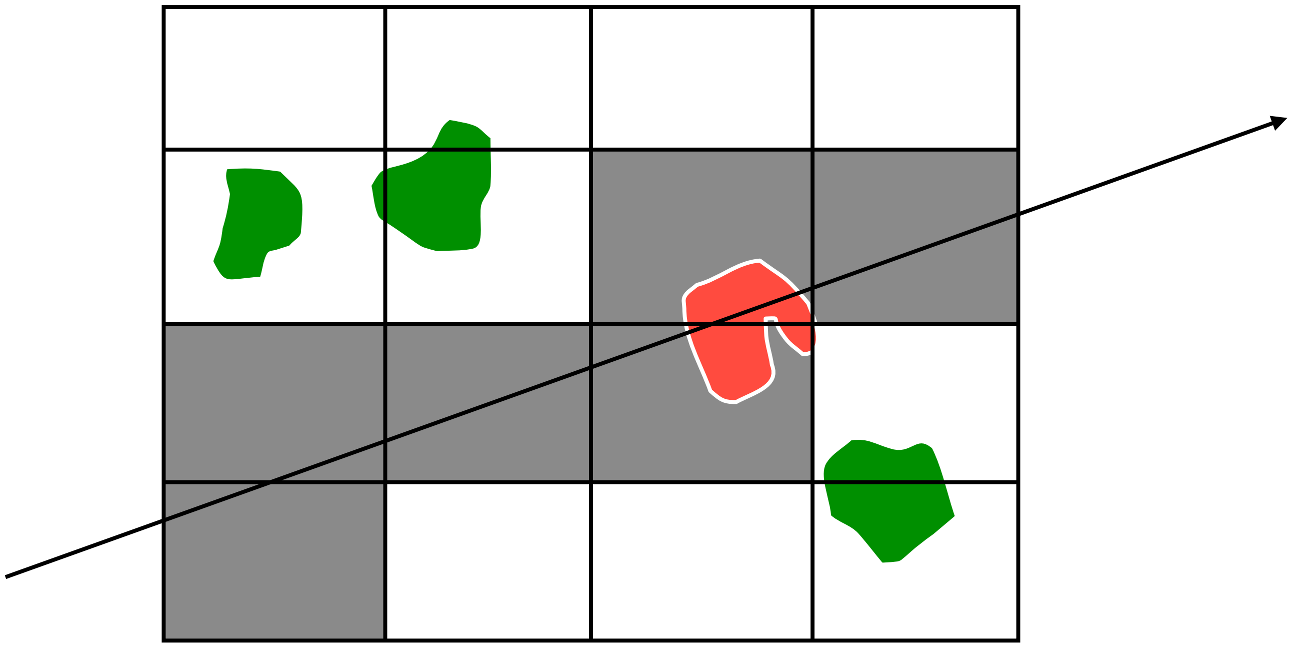
# Bounding volumes

- Use simple shape for quick test, keep a hierarchy



# Space Subdivision

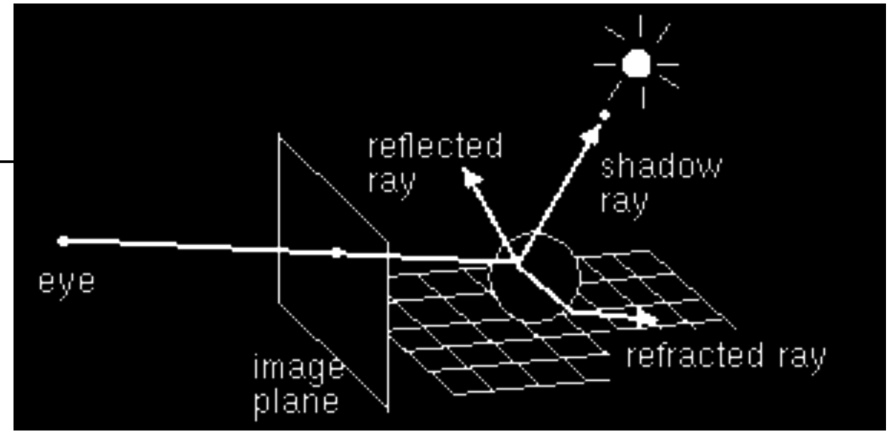
- Break your space into pieces
- Search the structure linearly



# Parallel Processing

- **You can always throw more processors at it.**

# Summary: Raytracing




**Recursive algorithm**

Function **main**()

for each pixel (c,r) on screen

determine ray $r_{c,r}$ from eye through pixel

color(c,r) = **raytrace**($r_{c,r}$ )

end for

End

Function **raytrace**(r)

find closest intersection P of ray with objects

$c_{local}$ = Sum(shadowRays(P,$Light_i$))

$c_{re}$ = **raytrace**($r_{re}$)

$c_{ra}$ = **raytrace**($r_{ra}$)

return c = $c_{local}+k_{re}*c_{re}+k_{ra}*c_{ra}$

end

# Advanced concepts

- Participating media
- Transculency
- Sub-surface scattering (e.g. Human skin)
- Photon mapping

# Raytracing summary

- View dependent
- Computationally expensive
- Good for refraction and reflection effects