Z-buffer algorithm

for each polygon in model

project vertices of polygon onto viewing plane

for each pixel inside the projected polygon

calculate pixel colour

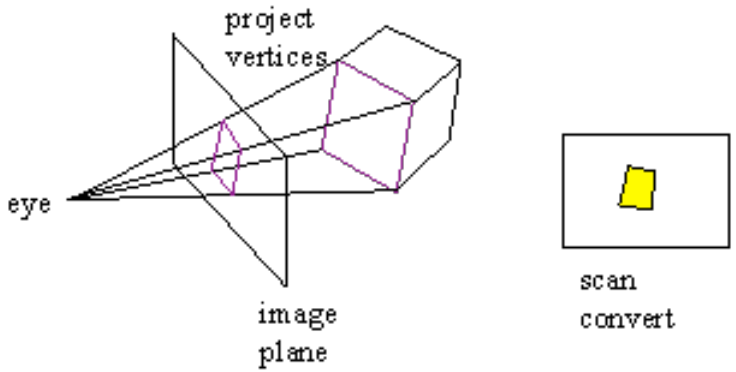
calculate pixel z-value

compare pixel z-value to value stored for pixel in z-buffer

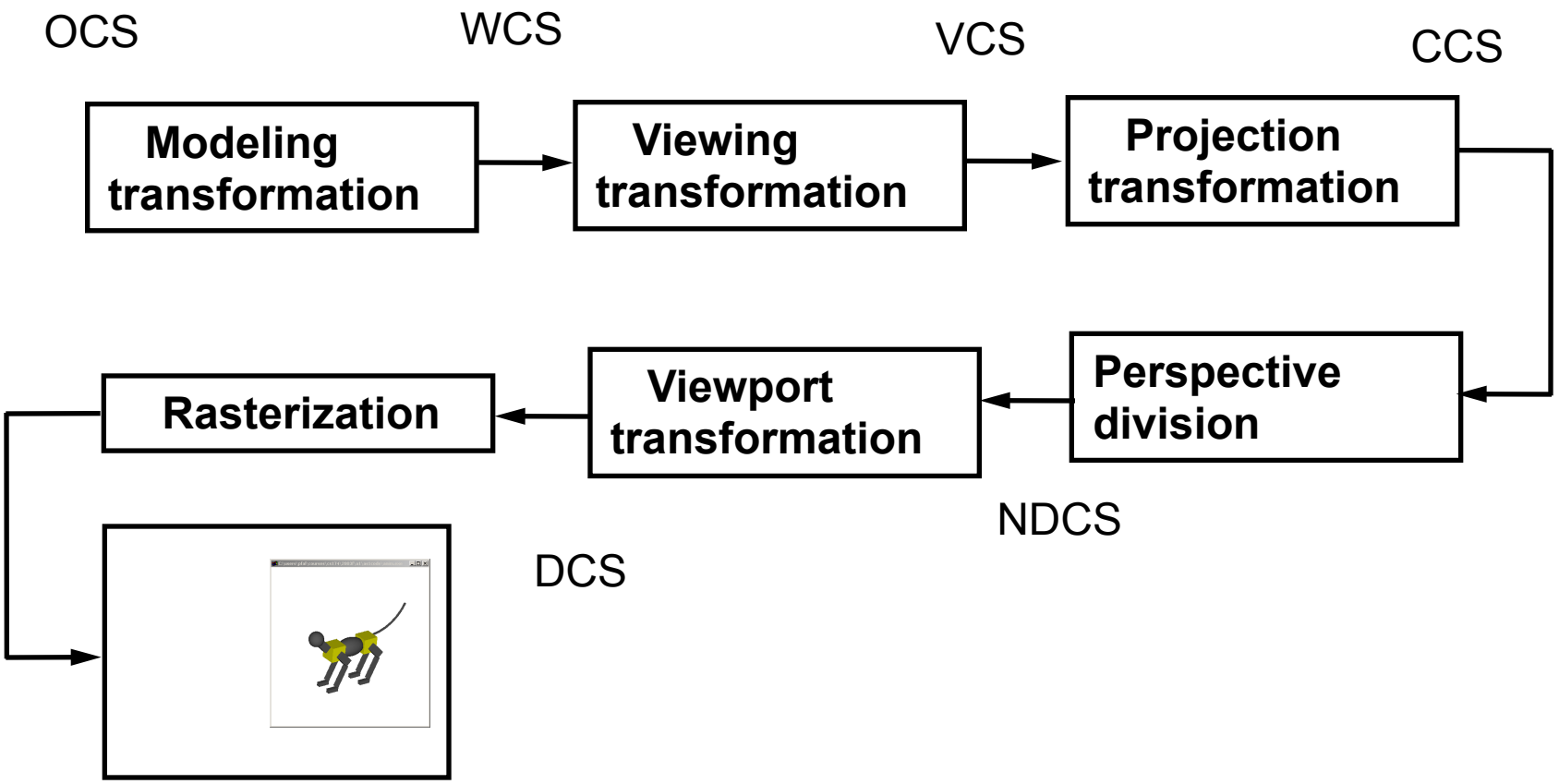
if pixel is closer, draw it in frame-buffer and z-buffer

end

end



Z-buffer Graphics Pipeline



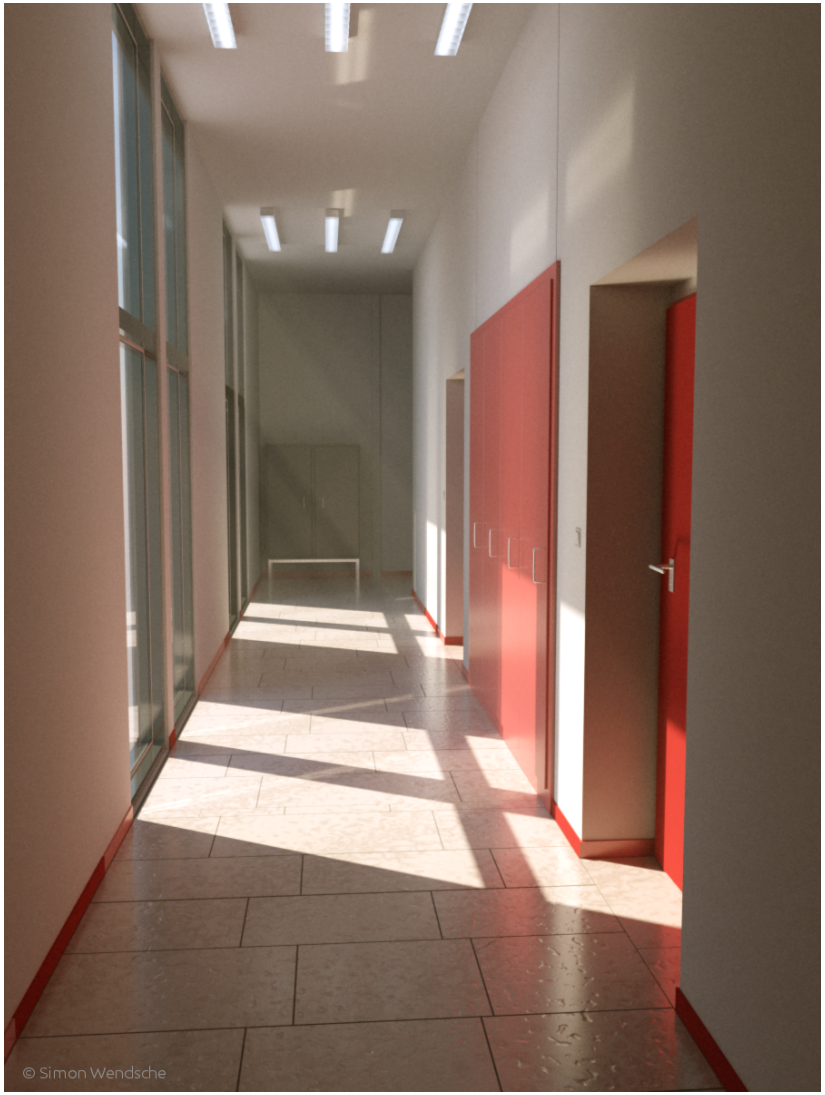
Global Illumination

Rendered by

LuxRender

Copyright

Simon Wendsche



Physics-based Rendering

Light Transport Equation (LTE)

$$L_o(p, \omega_0) = L_e(p, \omega_o) + \int_{\mathcal{S}^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\omega_i \cdot n| d\omega_i$$

L_o : Outgoing radiance at p in direction ω_o

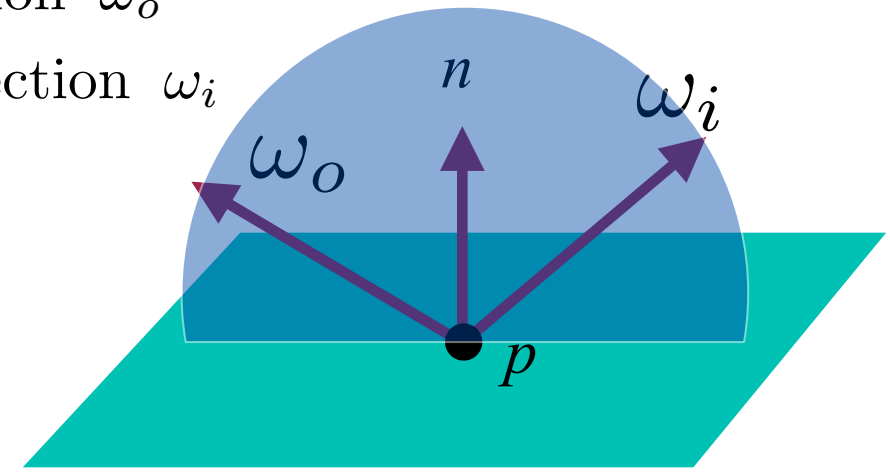
L_e : Emitted radiance at p in direction ω_o

L_i : Incident radiance at p from direction ω_i

n : Unit normal at p

f : Bidirectional scattering function

$\mathcal{S}^2$: The unit sphere centered at p



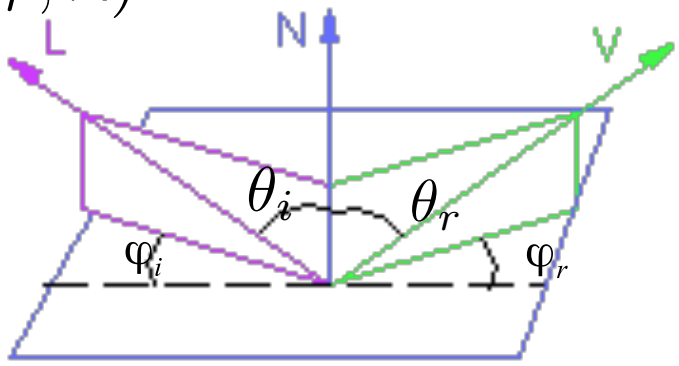
- Integral over directions

BSDF

Bidirectional Scattering Distribution Function

- BRDF: Bidirectional Reflectance Distribution Function

$$f_r(p, \omega_o, \omega_i, \lambda) = f_r(p, \phi_i, \theta_i, \phi_r, \theta_r, \lambda)$$



- BTDF: Bidirectional Transmittance Distribution function

Examples of a few simple diffuse BRDFs (wikipedia)

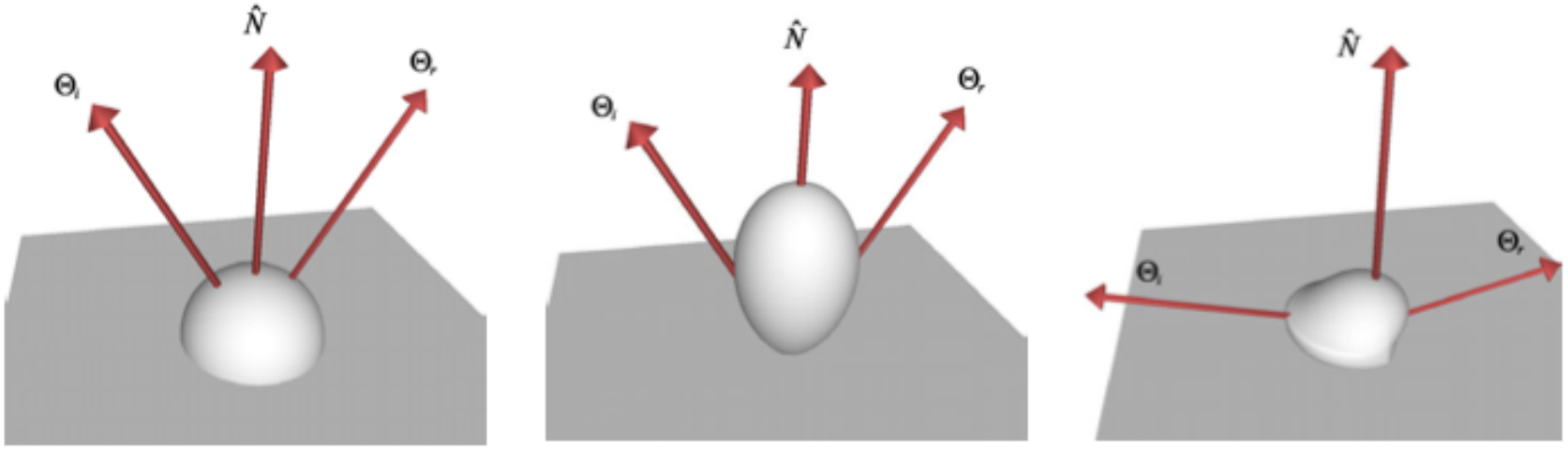


Fig. 2. Diffuse BRDF models: Lambert, Minnaert and Oren-Nayar.

BSSRF

- Bidirectional scattering-surface distribution function S()

- We now need to integrate over surface area as well

$$L(p, \omega_0) = \int_A \int_{S^2} S(p', \omega_i, p, \omega_o) \cos\theta_i d\omega_i dA$$

- Examples: http://graphics.ucsd.edu/~henrik/images/subsurf.html

Rendering Equation

Simple but difficult to solve

- High dimensionality
  - *and we have not even used a variable for color (wavelength of light)*

Solutions based on

- simplifying assumptions
- stochastic sampling
- FEM

One solution

Ray-tracing

Open source ray tracers

- Povray: www.povray.org
- LuxRender, http://www.luxrender.net
- MegaPov

Raytracing

Offline high quality rendering

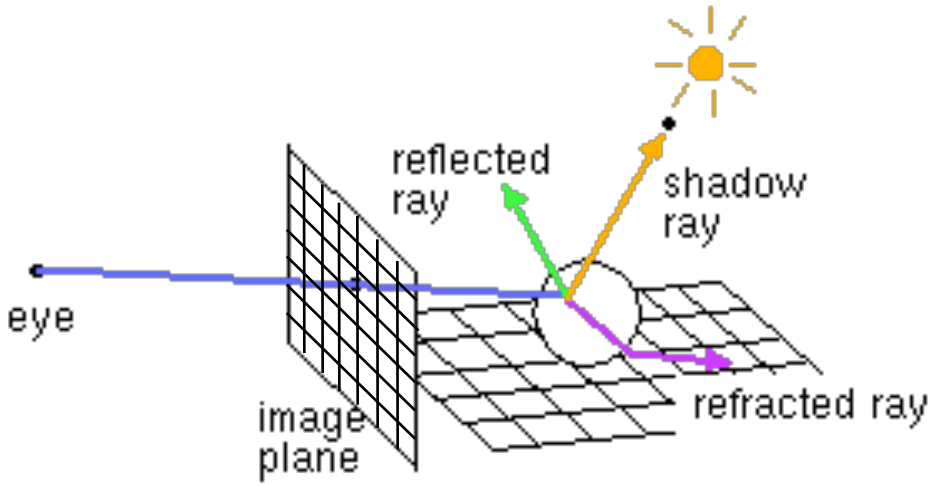
- Real-time versions for limited complexity scenes

Tuned for specular and transparent objects

- Partly physics, optics

Main idea:

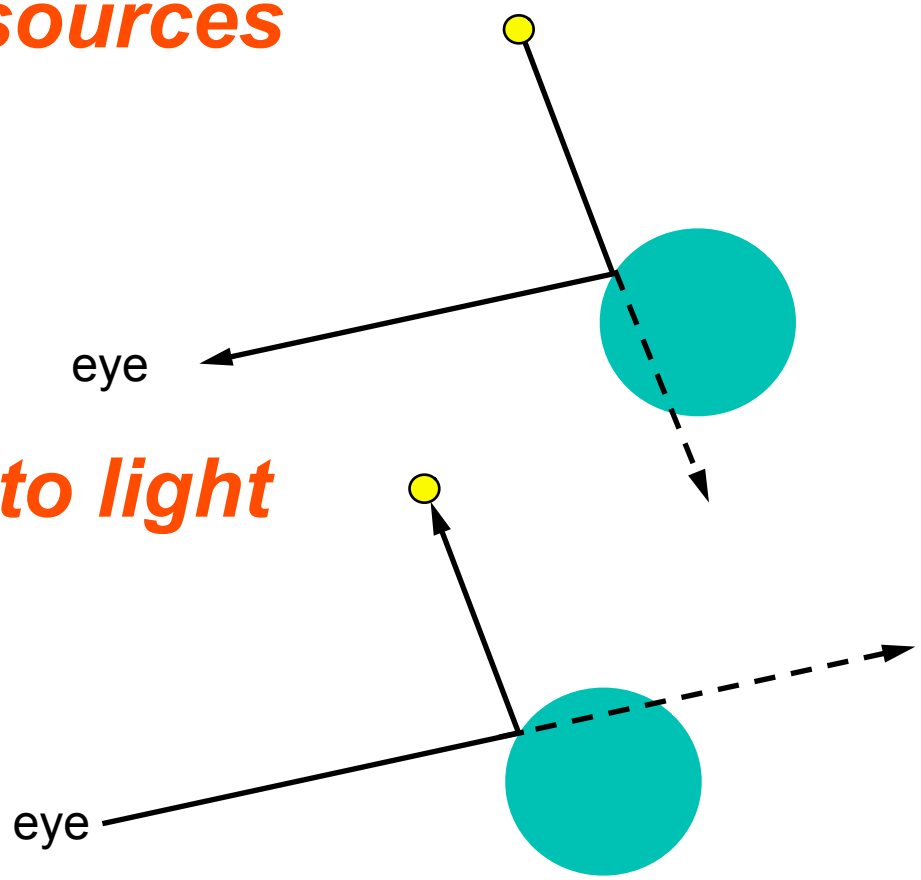
- A pixel should have the color of the object point that projects to it.



Forward and Backward methods

Forward: from light sources to eye

Backward: from eye to light sources



Our version

There are different variants of the back wards RT method that make different assumptions and simplifications

We will study a specific version that is

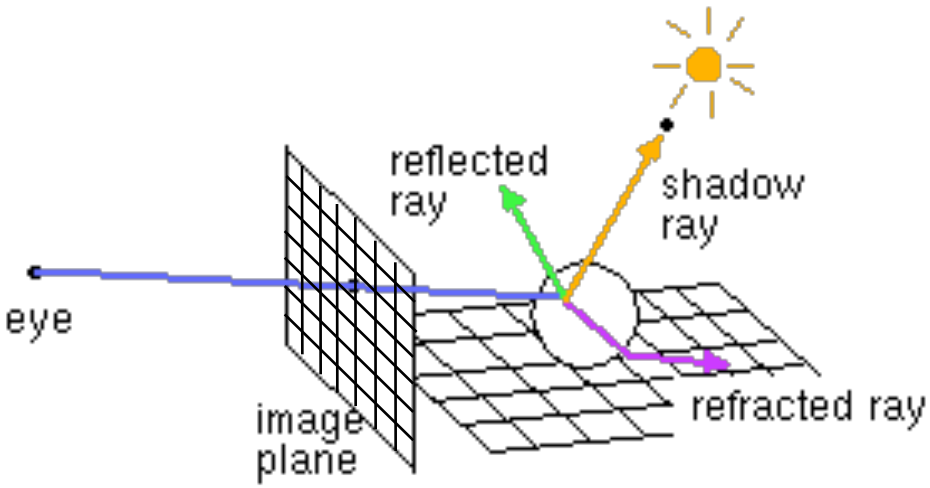
- Efficient
- Simple to implement
- Educational

Original Backwards Raytracing

Turner Whitted 1979

Assumptions

- Reflection and Refraction only on ideal direction cones
- Lambertian diffuse surfaces (no custom BRDFs)
- Diffuse objects do not receive reflected light



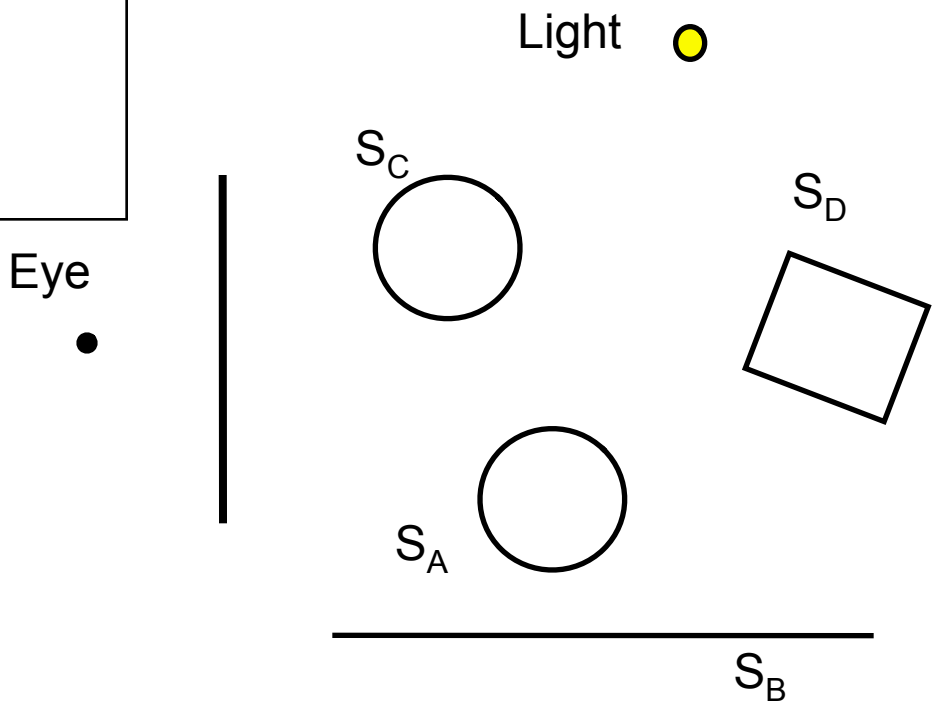
for each pixel on screen

- determine ray from eye through pixel
- find closest intersection of ray with an object
- compute direct illumination (shadow ray)
- trace reflected and refracted ray, recursively
- calculate pixel colour, draw pixel

end

Scene

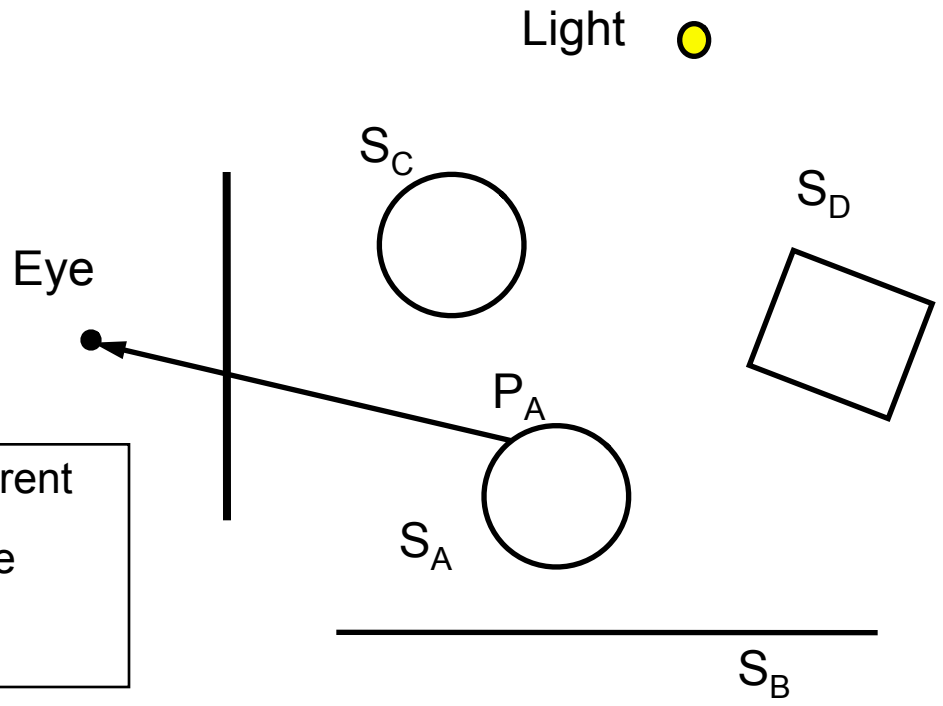
S_A	shiny, transparent
S_B,S_D	diffuse, opaque
S_C	shiny, opaque



Three sources of light

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

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

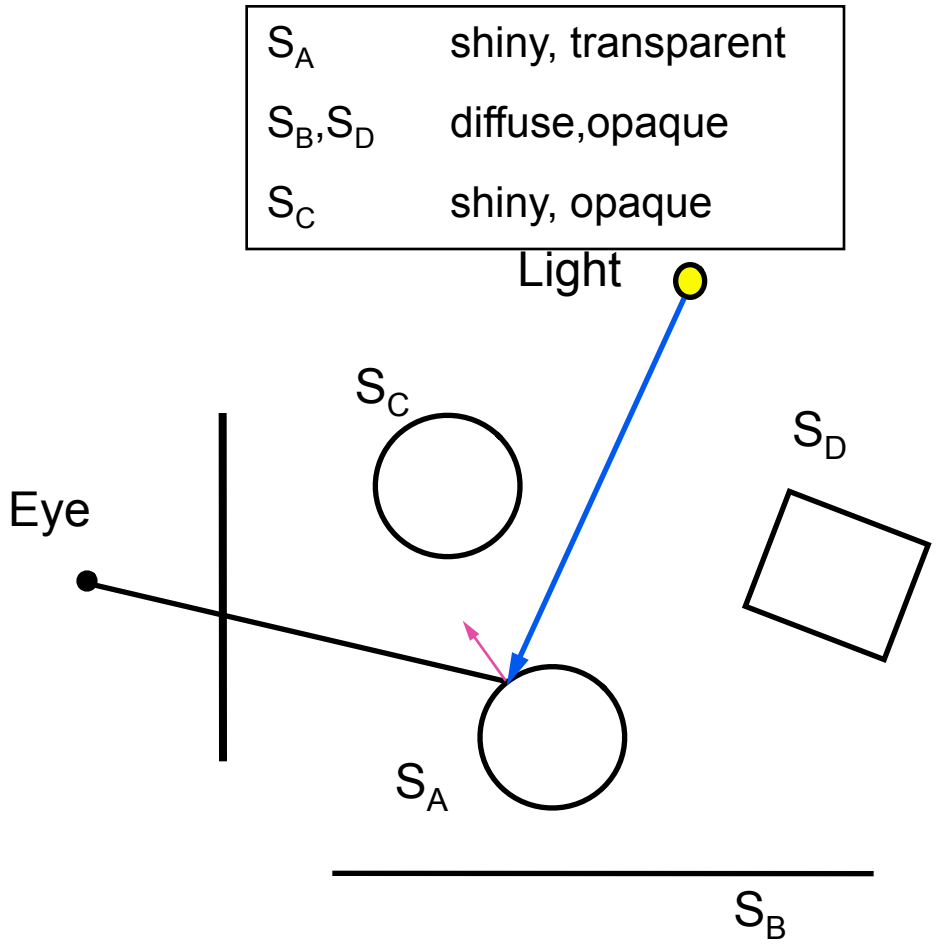


S_A	shiny, transparent
S_B,S_D	diffuse,opaque
S_C	shiny, opaque

Directly from a light source

Local illumination model:

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

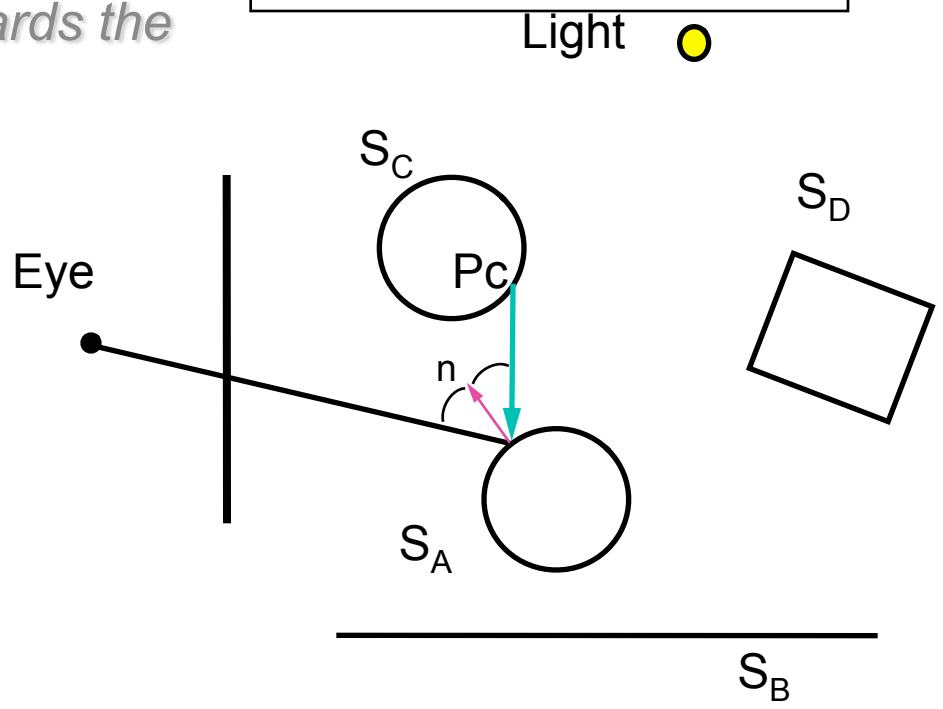


Reflection

What is the color that is reflected to P_A and that P_A reflects back to the eye?

What ever comes from the direction that eventually reflects towards the eye.

S_A	shiny, transparent
S_B,S_D	diffuse,opaque
S_C	shiny, opaque



Reflection

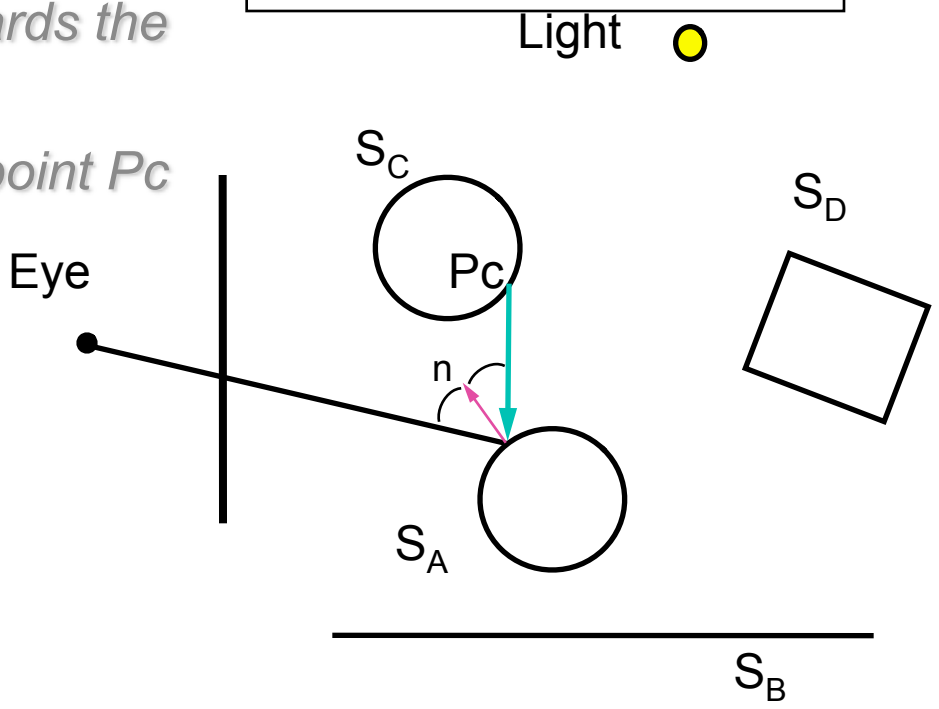
What is the color that is reflected to P_A and that P_A reflects back to the eye?

What ever comes from the direction that eventually reflects towards the eye.

That direction gets light from point Pc

What is the color of P_C ?

S_A	shiny, transparent
S_B,S_D	diffuse,opaque
S_C	shiny, opaque



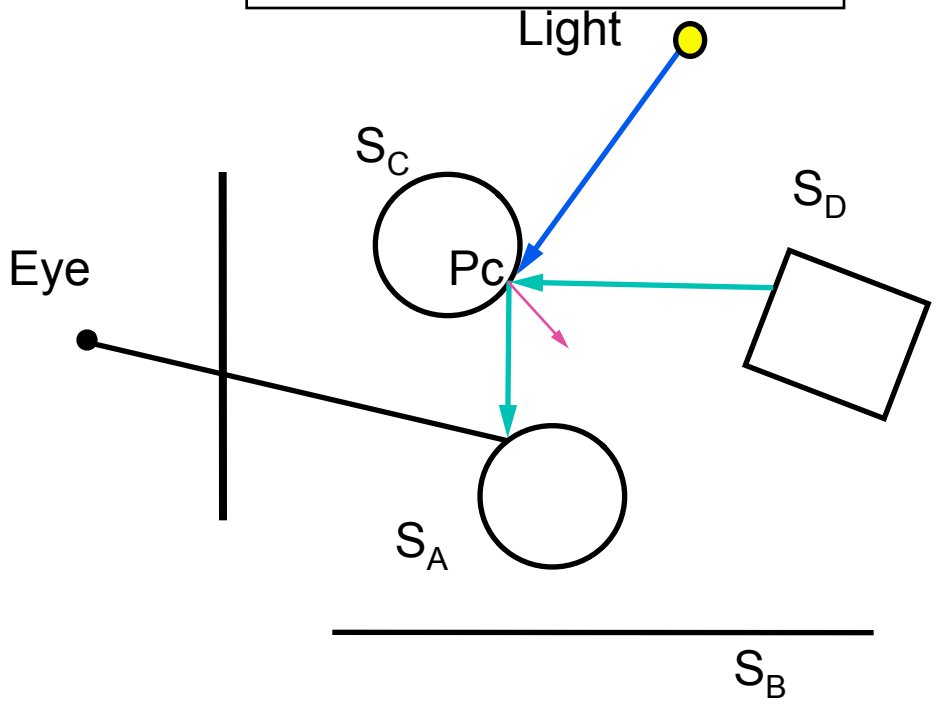
Reflection

What is the color of Pc?

Just like P_A : raytrace P_C i.e compute the three contributions from

- *Light sources*
- *Reflection*
- *Refraction*

S_A	shiny, transparent
S_B,S_D	diffuse,opaque
S_C	shiny, opaque



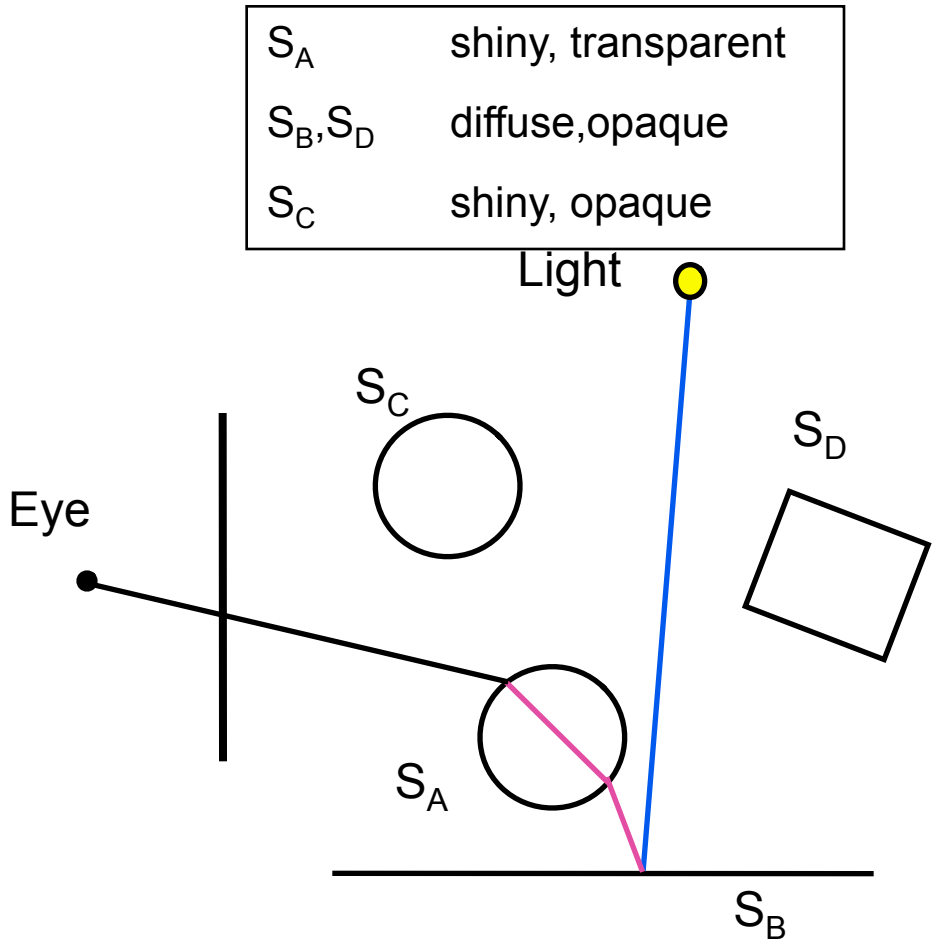
Refraction

Transparent materials

How do we compute the refracted contribution?

We raytrace the refracted ray.

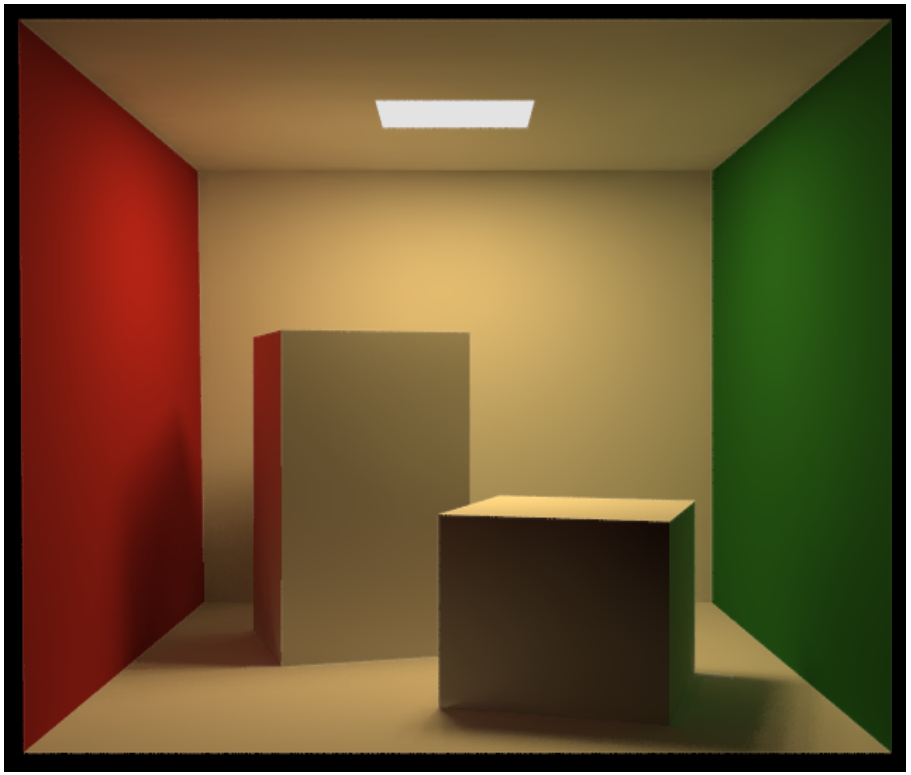
1. *Lights*
2. *Reflection*
3. *Refraction*



What have we ignored ?

What have we ignored ?

Diffuse objects do not receive light from other objects.

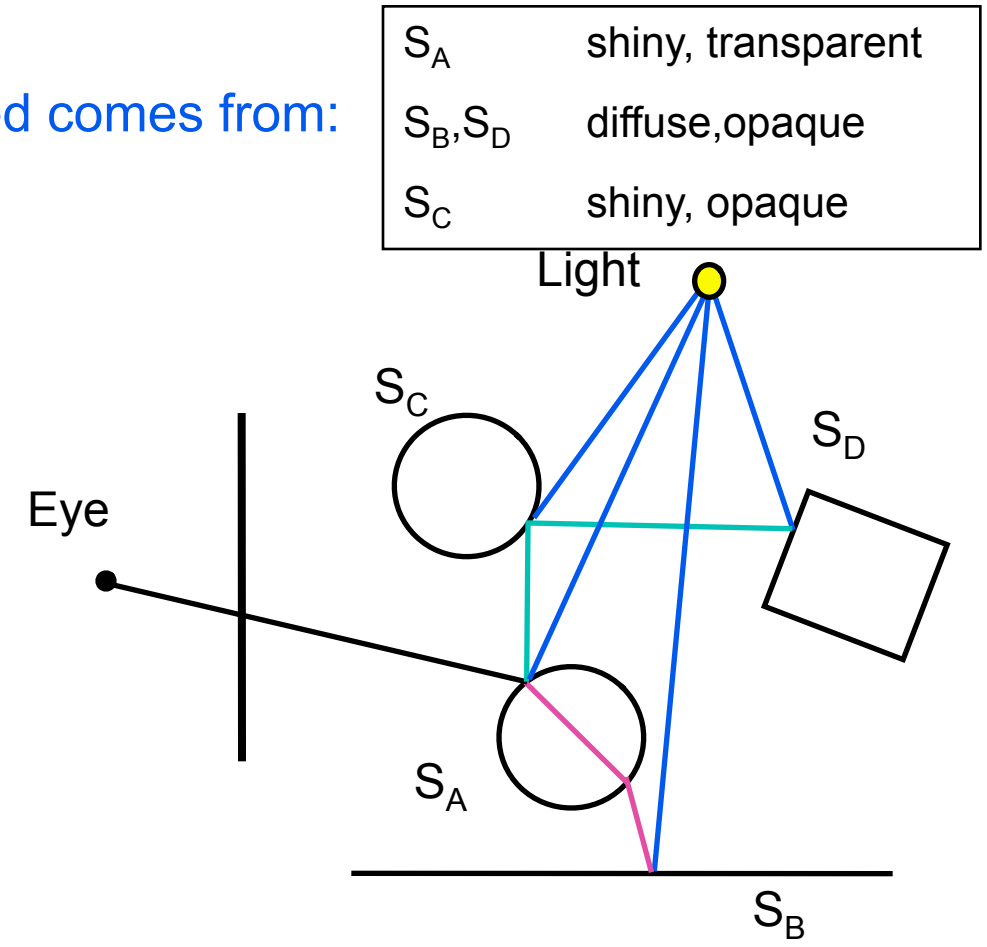


Three sources of light together

The color that the pixel is assigned comes from:

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

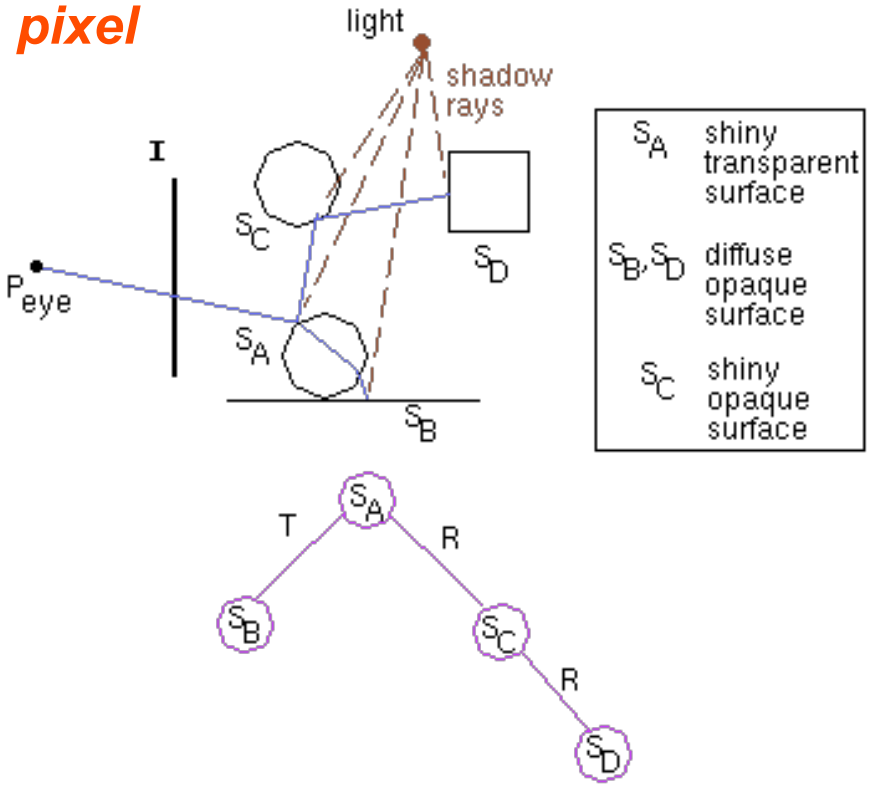
It is more convenient to trace the rays from the eye to the scene (backwards)



Backwards Raytracing Algoritm

For each pixel construct a ray: eye→ pixel

```
raytrace( ray )
    P = compute_closest_intersection(ray)
    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 )
```



How many levels of recursion do we use?

The more the better.

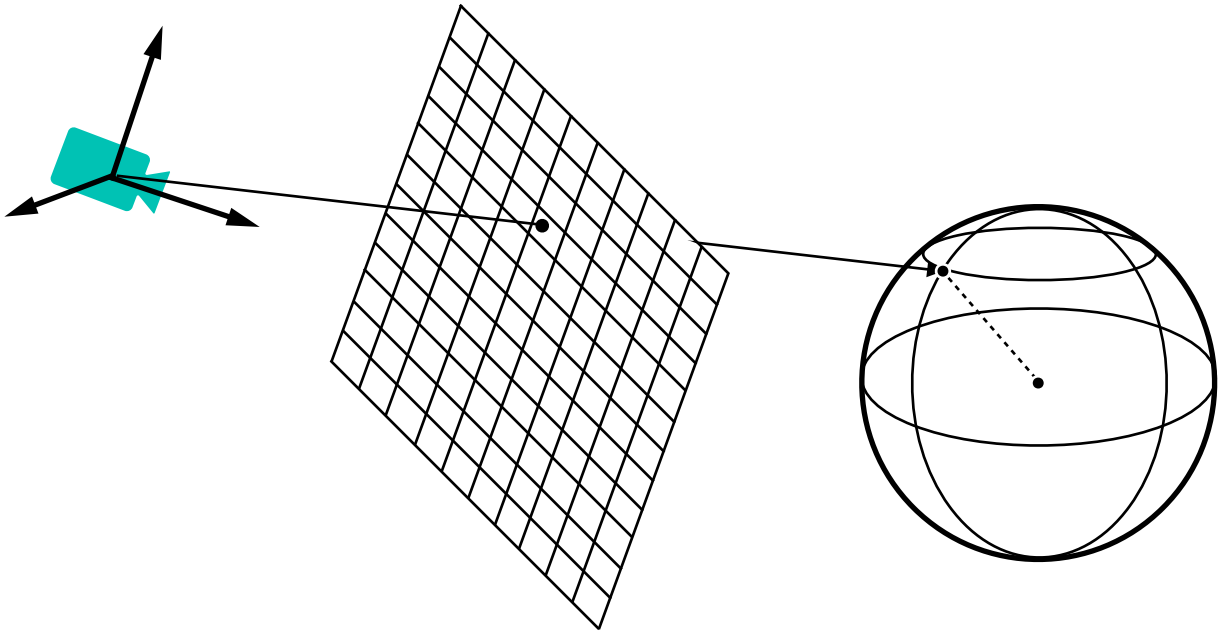
Infinite reflections at the limit.

Stages of raytracing

We need to do the following:

- Set the camera and the image plane
- Compute a ray from the eye to every pixel and trace it in the scene
- Compute object-ray intersections
- Cast shadow, reflected and refracted rays at each intersection

Setting up the camera



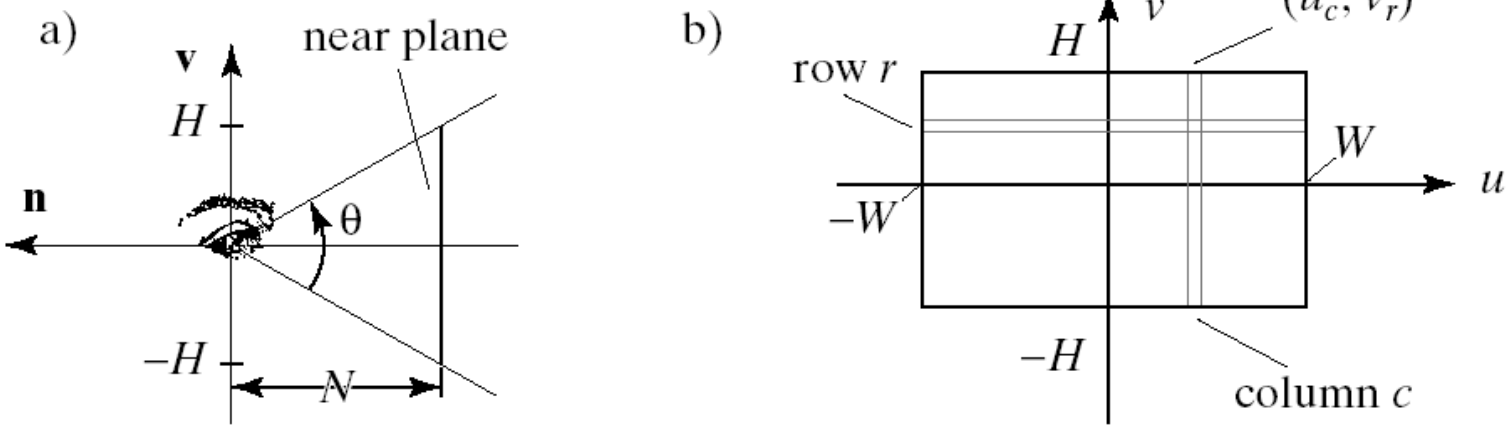
Set the viewport and resolution in Camera Coordinates

Image parameters

- Width 2W, Height 2H
- Number of pixels nCols x nRows

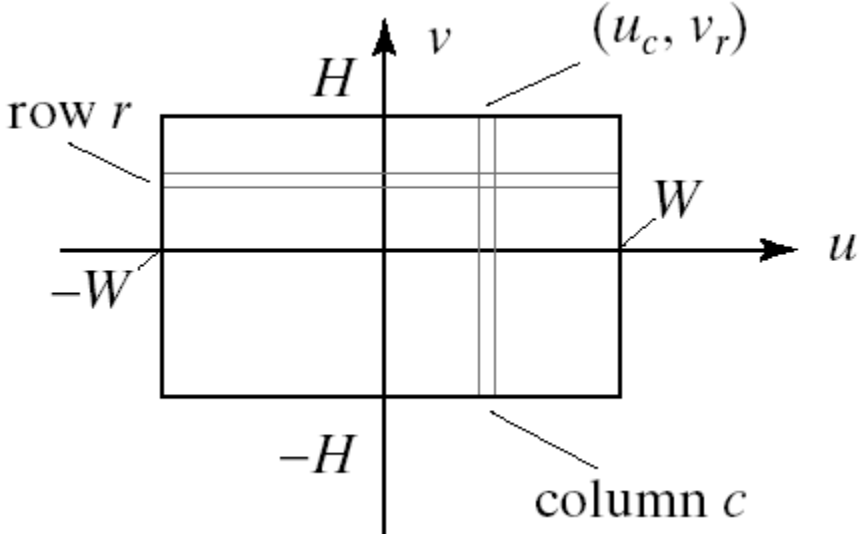
Camera parameters (view point)

- Camera coordinate system (**eye**, **u**,**v**,**n**)
- Image plane at n = -N



Pixel coordinates in camera coordinate system

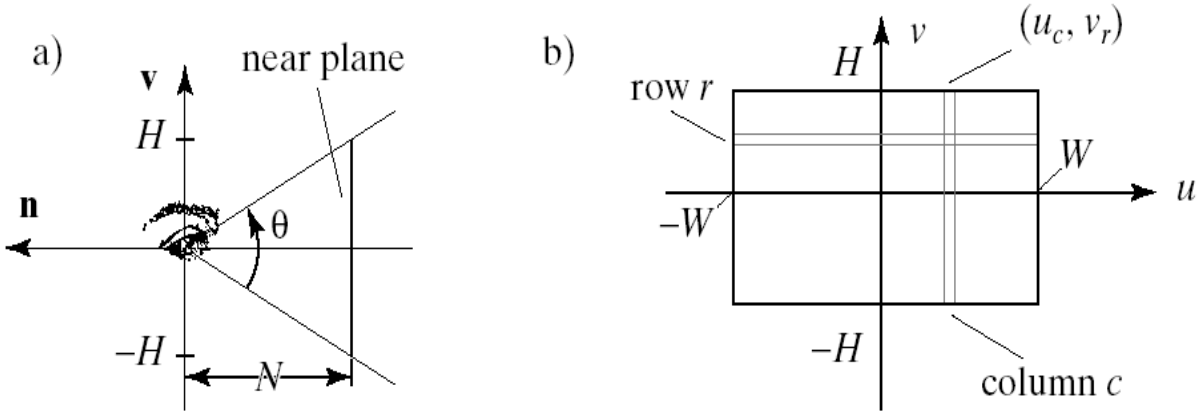
Lower left corner of pixel P(r,c) has coordinates in camera space:



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

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

Ray through pixel



Lower left corner

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

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

Ray through pixel:

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

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

Ray-object intersections

Objects

- Spheres, cylinders, surfaces, etc
- Any object that we can mathematically compute intersections with lines

For us

- Only affine transformed spheres

Sphere

Implicit

$$f(x,y,z) = (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = R^2$$

Parametric

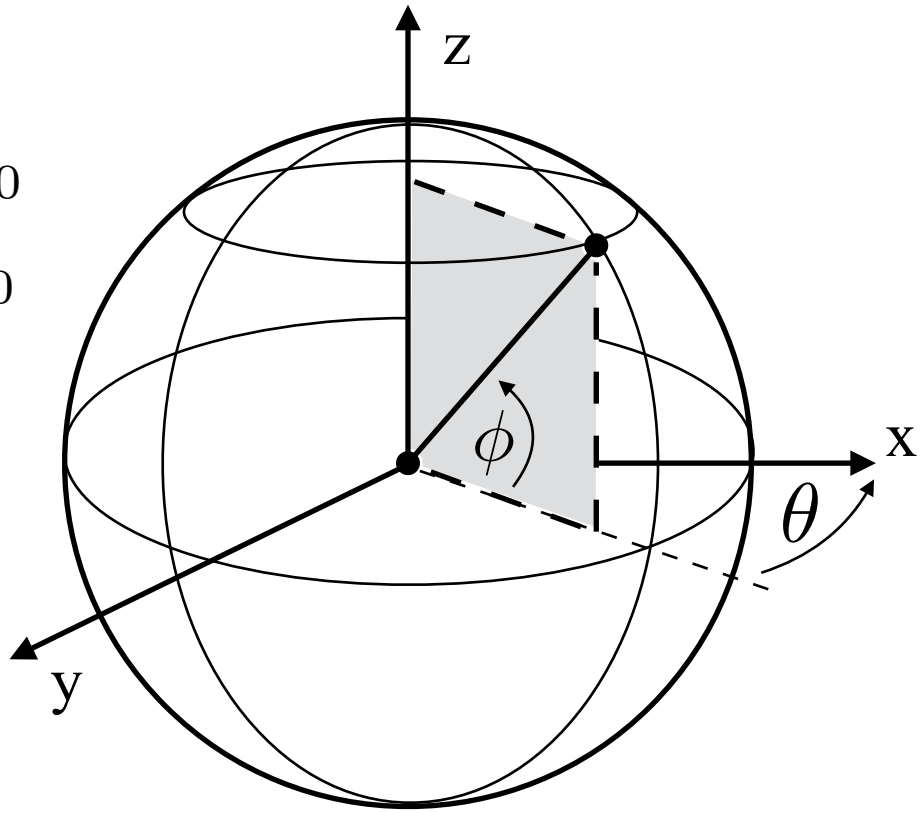
$$x(\phi,\theta) = Rcos(\phi)cos(\theta) + x_0$$

$$y(\phi,\theta) = Rcos(\phi)sin(\theta) + y_0$$

$$z(\phi,\theta) = Rsin(\phi) + z_0$$

$$-\pi/2 \leq \phi \leq \pi/2$$

$$-\pi \leq \theta \leq \pi$$



Canonical Sphere

- Radius 1 centered around the origin
- Equivalent forms

$$\sqrt{x^2 + y^2 + z^2} = 1$$

$$x^2 + y^2 + z^2 = 1$$

$$|P| - 1 = 0$$

$$|P| = 1 \rightarrow \sqrt{P \cdot P} = 1 \rightarrow P \cdot P = 1$$

Normal on a sphere

Normal

$$\mathbf{N}(x, y, z) = \nabla f(x, y, z) = 2(x - x_o, y - y_0, z - z_0)$$

Unit Normal

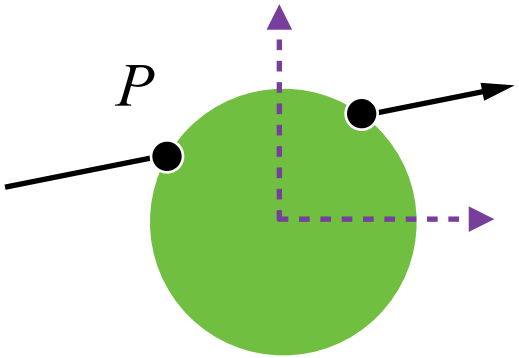
$$\mathbf{n}(x, y, z) = \mathbf{N}/||\mathbf{N}||$$

Ray - Canonical sphere intersection

Can I use homogeneous coordinates?

$$ray(t) = S + \mathbf{c}t$$

$$Sphere(P) = |P| - 1 = 0$$



$$Sphere(ray(t)) = 0 \Rightarrow$$

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

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

This is a quadratic equation in t

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

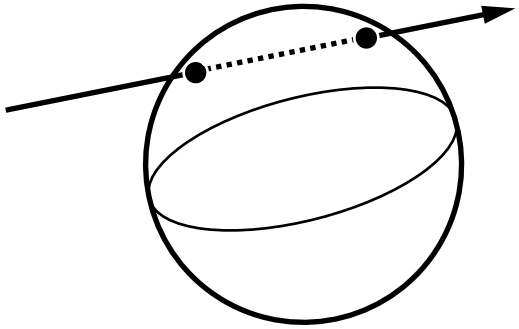
If $(B^2 - AC) > 0$ two solutions

Note!

Can I use homogeneous coordinates?

$$ray(t) = S + \mathbf{c}t$$

$$Sphere(P) = |P| - 1 = 0$$



$$Sphere(ray(t)) = 0 \Rightarrow$$

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

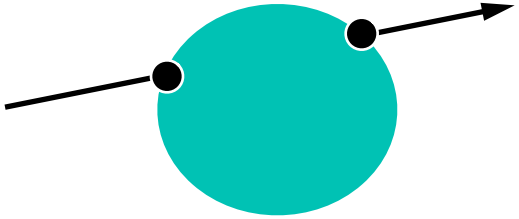
$$\boxed{|\mathbf{c}|^2 t^2 + 2(S \cdot \mathbf{c})t + |S|^2 - 1 = 0}$$

Note!

Can I use homogeneous coordinates?

$$ray(t) = S + \mathbf{c}t$$

$$Sphere(P) = |P| - 1 = 0$$



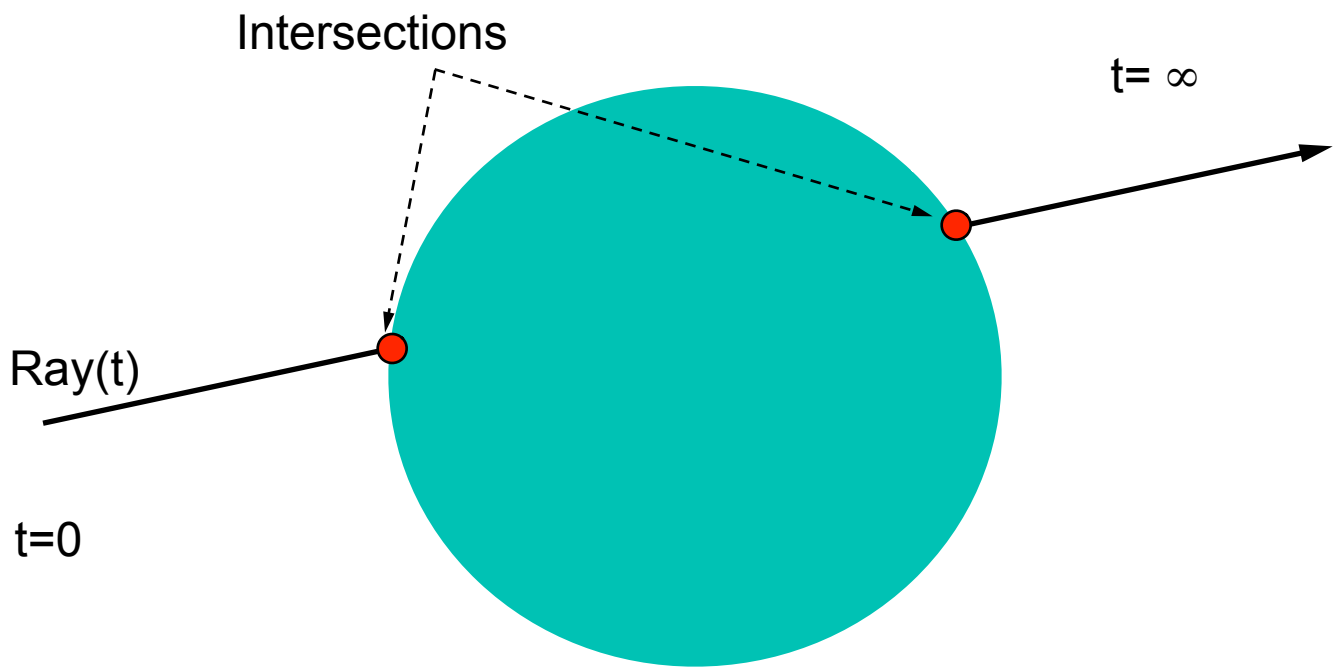
$$Sphere(ray(t)) = 0 \Rightarrow$$

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

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

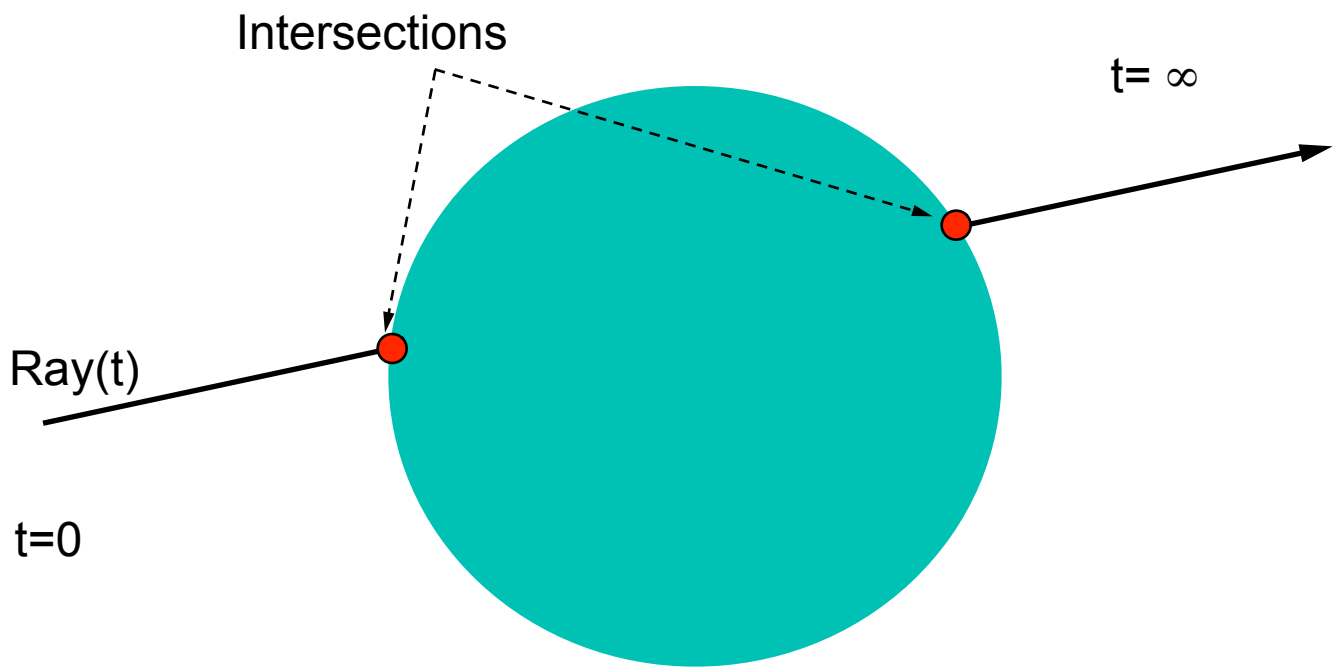
Must solve the equation in cartesian space

First intersection?



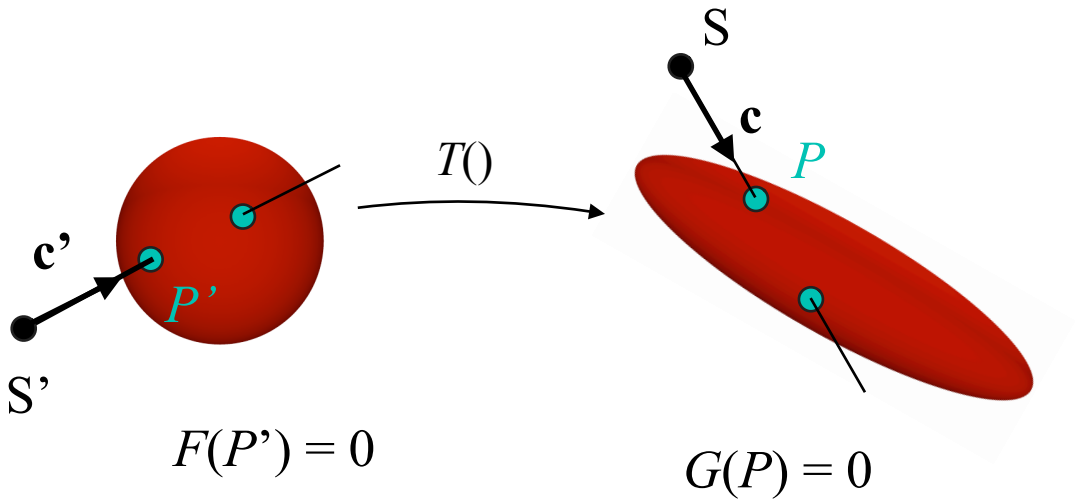
First intersection?

t1 < t2

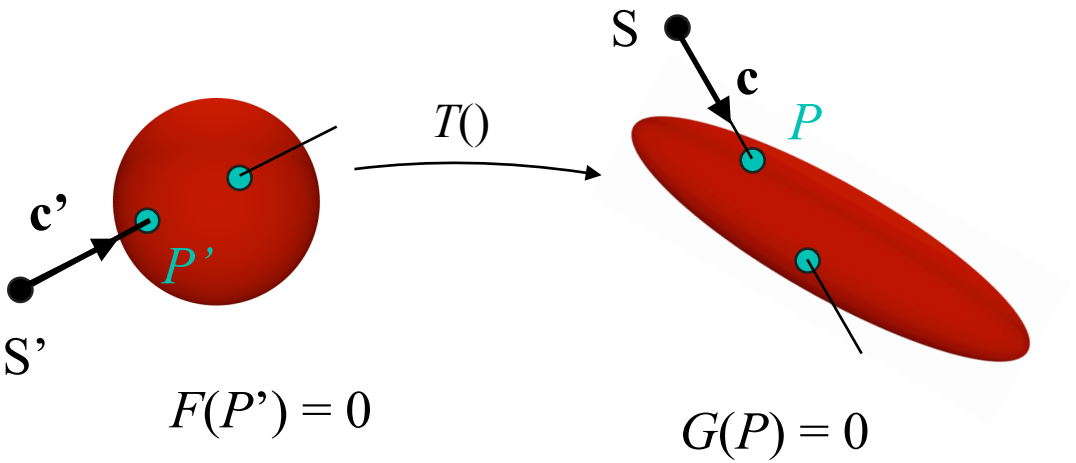


Transformed primitives?

- That was a canonical sphere.
- Where does S+**c**t hit the transformed sphere G = T(F) ?



Linear transformation

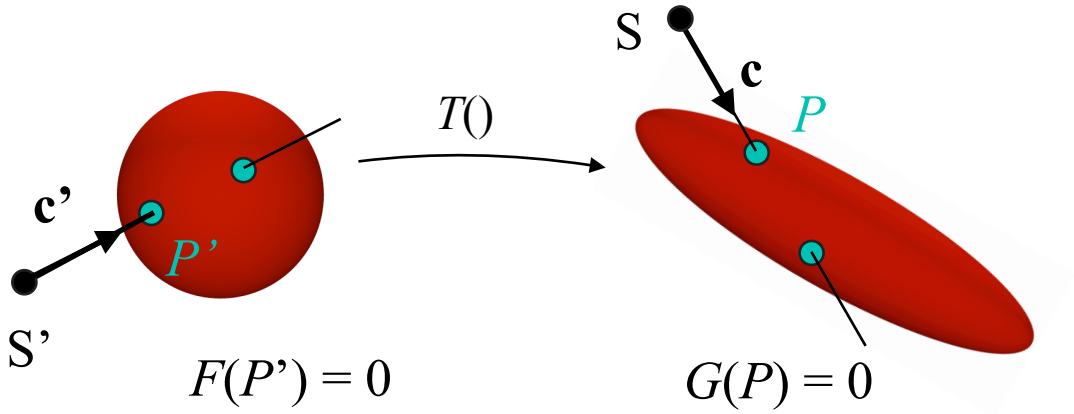


Implicit equation $G(P) = 0$.

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

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

- Drop 1 and 0 to get *S'*+c'*t.*

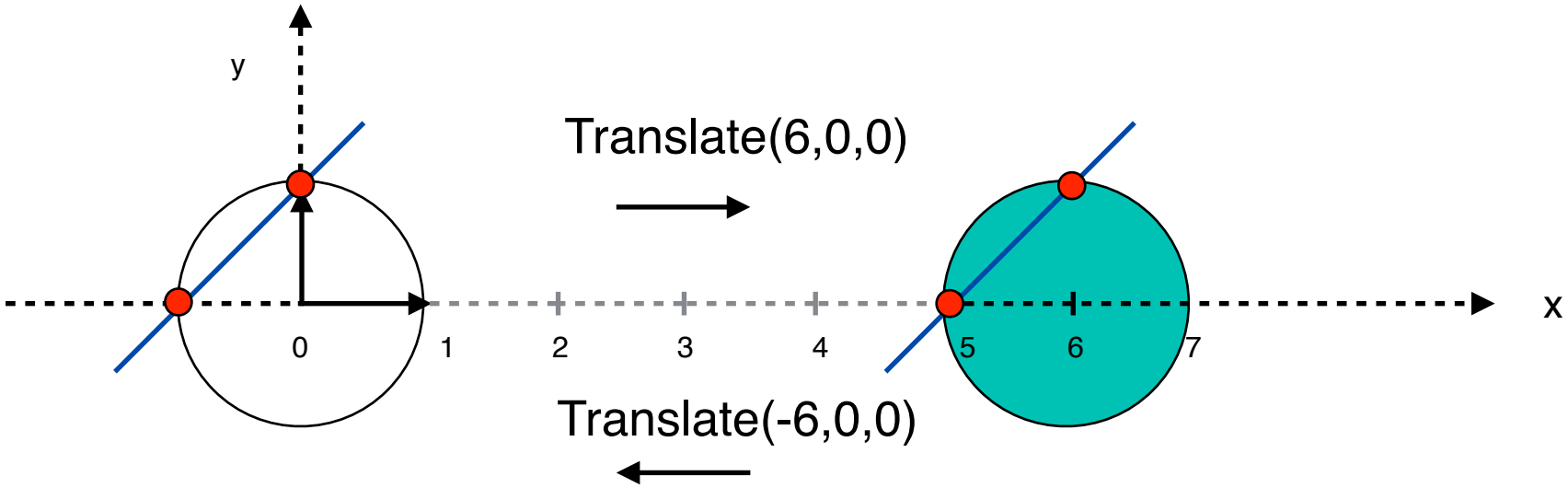
So ..for each object

- Inverse transform ray and get *S'*+c*'t.*
- Find the intersection *t*, $t_{h,}$ between inv-ray and canonical object.
- Use t_h in the untransformed ray *S*+c*t* to find the intersection.
- **For the normal, use the inverse transpose**

Example

Canonical sphere, radius 1
|(x,y,z)| - 1 = 0

Translated sphere
|(x-6,y,z)| - 1 = 0



Ray'(t) = (-1,0,0) + t(1,1,0)

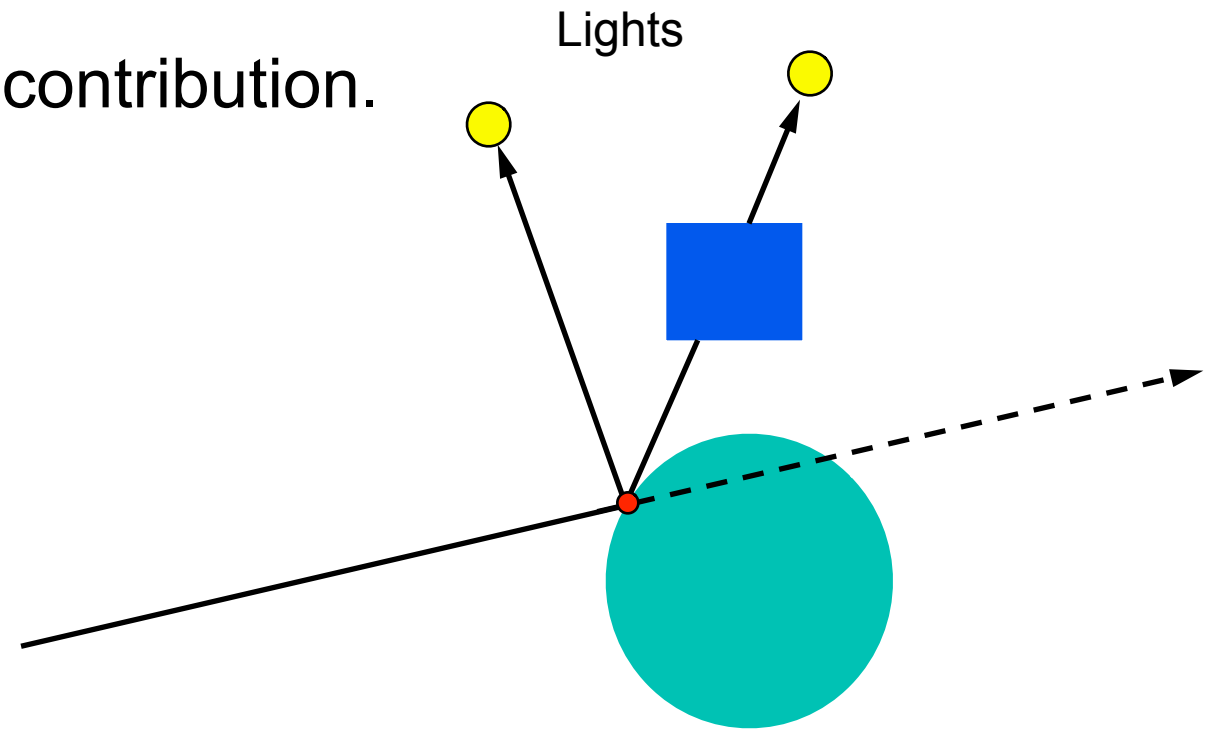
Intersections
(-1,0,0) and (0,1,0)

Ray(t) = (5,0,0)+ t(1,1,0)

Intersections:
(5,0,0) and (6,1,0)

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.



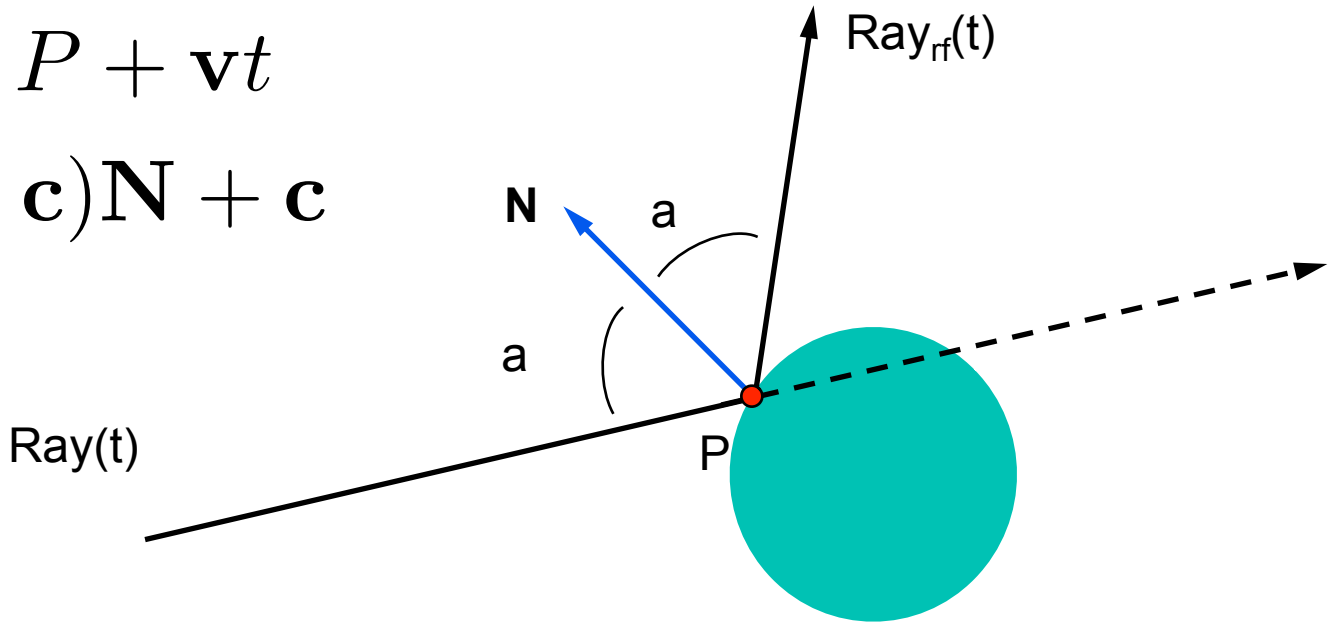
Reflected ray

Raytrace the reflected ray

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

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

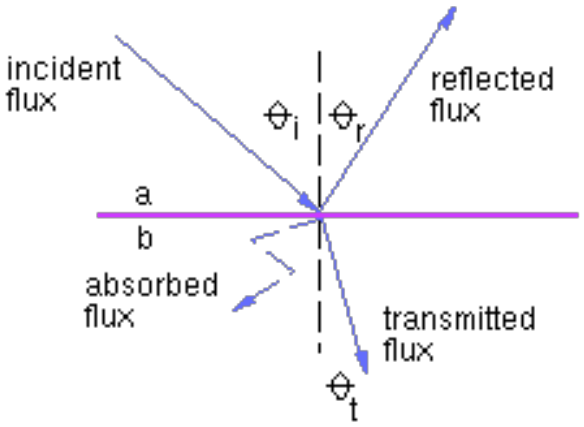
$$\mathbf{v} = -2(\mathbf{N} \cdot \mathbf{c})\mathbf{N} + \mathbf{c}$$



Refracted ray

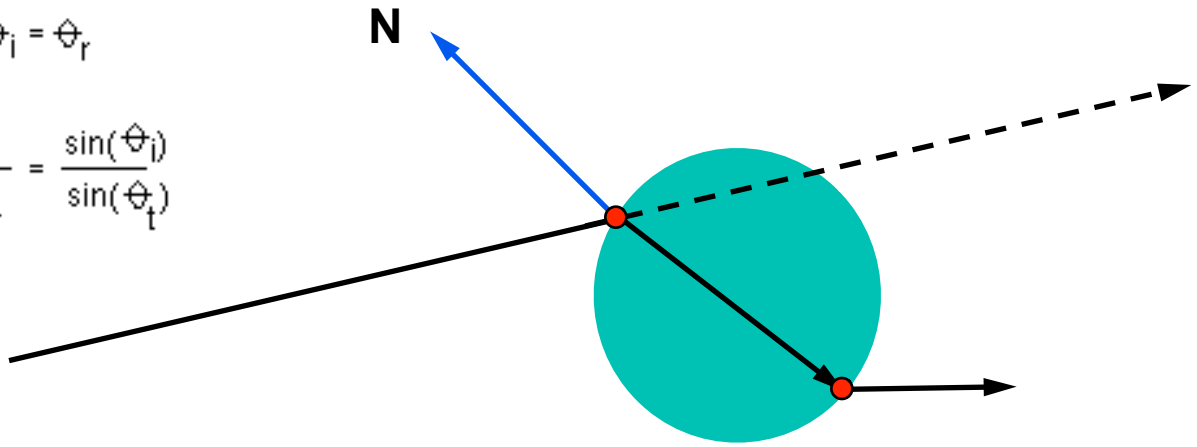
Raytrace the refracted ray

Snell's law



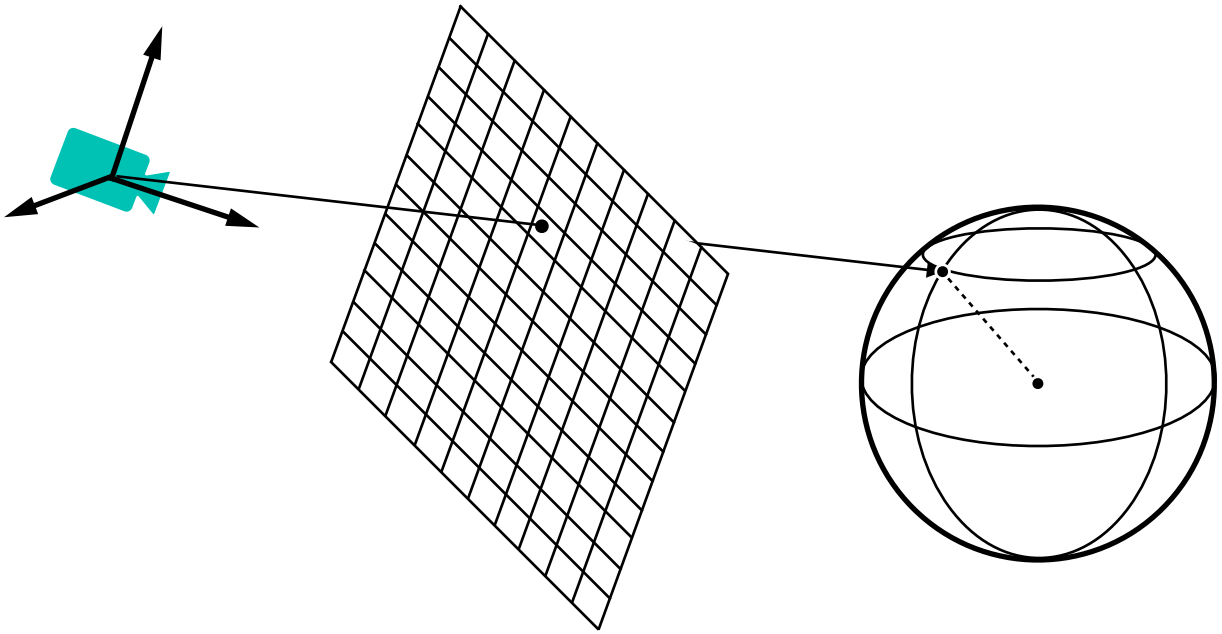
$$\theta_i = \theta_r$$

$$\frac{n_b}{n_a} = \frac{\sin(\theta_i)}{\sin(\theta_t)}$$



Add all together

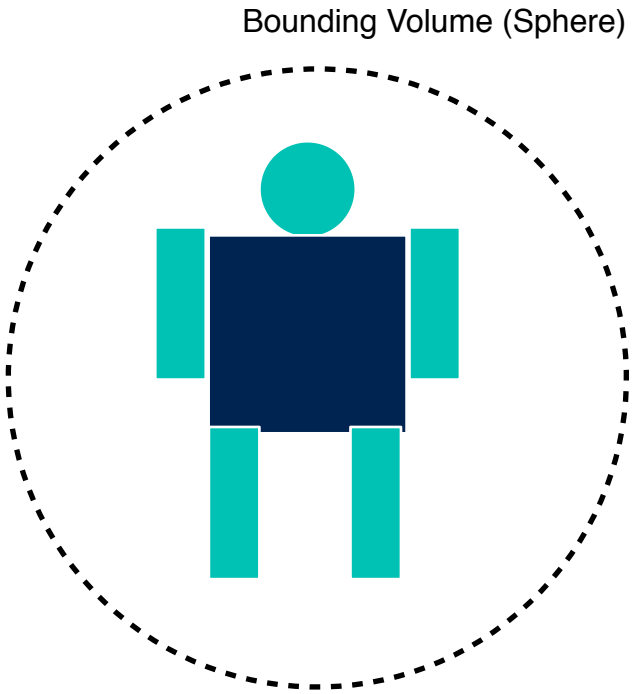
- color(r,c) = color_shadow_ray + K_{re}*$color_{re}$ + K_{ra}*$color_{ra}$



Efficiency issues

Computationally expensive

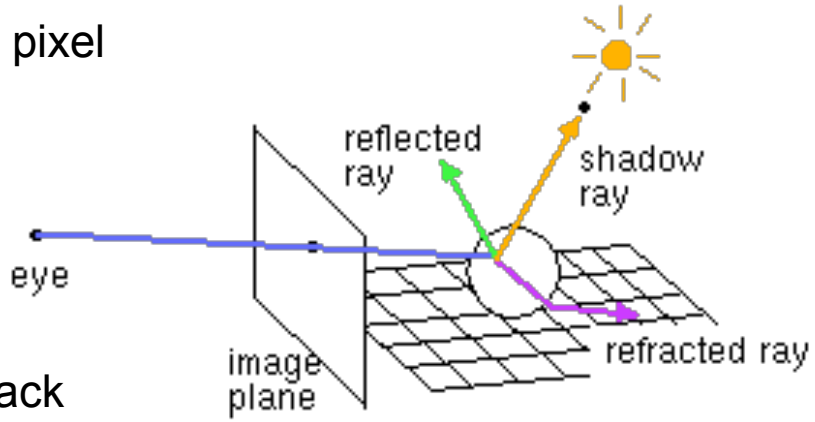
- avoid intersection calculations
  - *Voxel grids*
  - *BSP trees*
  - *Octrees*
  - *KD-Trees*
  - *Bounding volume trees*
- optimize intersection calculations
  - *try recent hit first*
  - *reuse info from numerical methods*



Summary: Raytracing

Recursive algorithm

Function Main
    for each pixel (c,r) on screen
        determine ray $r_{c,r}$ from eye through pixel
        ray.setDepth(1)
        color(c,r) = raytrace($r_{c,r}$)
    end for
end



function raytrace(r)
    if (ray.depth() > MAX_DEPTH) return black
    P = closest intersection of ray with all objects
    if ((no intersection) and (ray.depth ==1)) return backgroundColor
    clocal = Sum(shadowRays(P,Lighti))
    c_{re} = raytrace(r_{re})
    c_{ra} = raytrace(r_{ra})
    return (clocal+k_{re}*c_{re}+k_{ra}*c_{ra})
end

Advanced concepts

Participating media

Transculency

Sub-surface scattering (e.g. Human skin)

Photon mapping

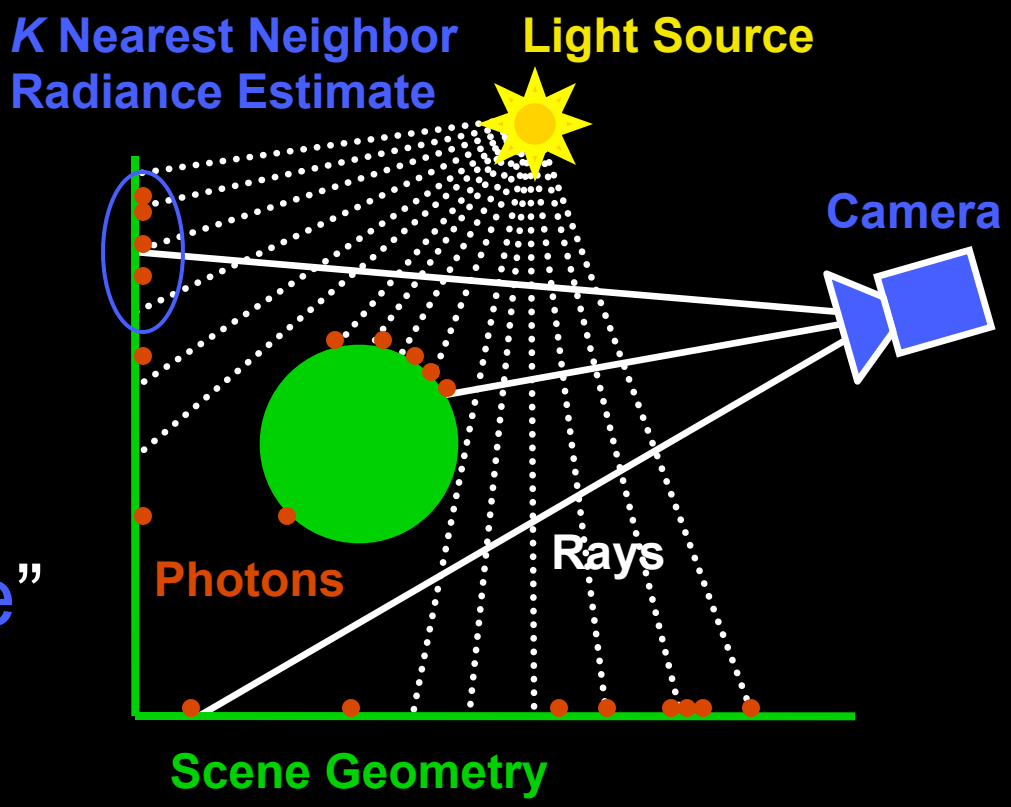
Photon Mapping
(slides courtesy of Shawn Singh)

Create a sampling of the illumination

- For scenes that do not change can be pre-computed
- View independent

Photon Mapping

- "Photons" are sample points traced from light sources
- For every ray, a "radiance estimate" is computed using photons



Resoures

- Real-time Ray Tracing
  Our implementation based on [Wald 2004]
- GPU Photon Mapping
  [Purcell et al 2003], [Larsen and Christensen 2004]
- Specialized Hardware
  [Steinhurst 2007], [Singh 2007]
- Improve performance of each k-nearest neighbor query
  [Ma and McCool 2002], [Gunther et al 2004], [Wald et al. 2004]
- Improve coherence of queries
  [Steinhurst et al. 2005], [Havran et al. 2005]
- Reverse photon mapping
  [Havran et al 2005]

Raytracing summary

View dependent

Computationally expensive

Good for refraction and reflection effects