Ray Tracing

Rendered by

PovRay 3.5

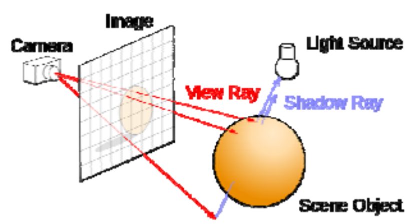
(Free open-source software)

Ray Tracing

Best for specular and transparent objects

Partly physics-based: geometric optics

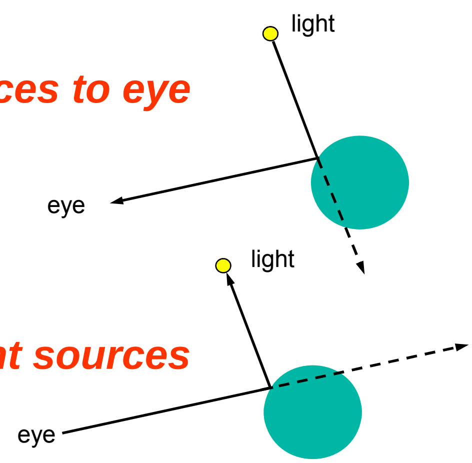
A pixel should have the color of the object point that projects to it



Light-Based and Eye-Based Methods

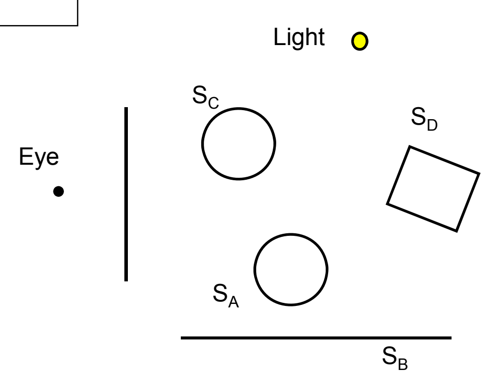
Light-based:
from light sources to eye

Eye-based:
from eye to light sources



Scene

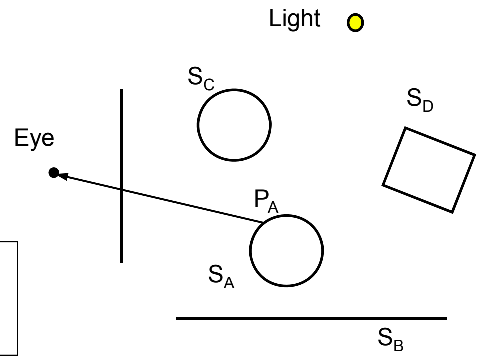
S_A	shiny, transparent
S_B,S_D	diffuse, opaque
S_C	shiny, opaque



Three Sources of Light

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

- Light sources
- Reflection from other objects
- Refraction from other objects

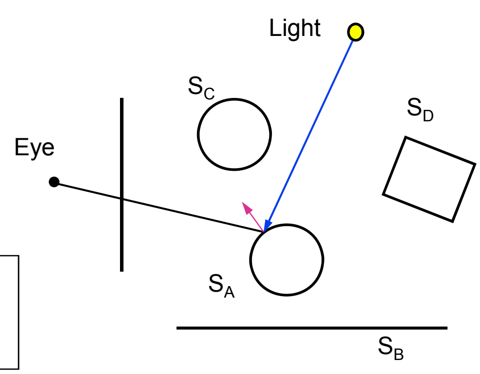


S_A	shiny, transparent
S_B,S_D	diffuse, opaque
S_C	shiny, opaque

Directly From Light Source

Local (Phong) shading model:

I = I-diffuse + I-specular + I-ambient



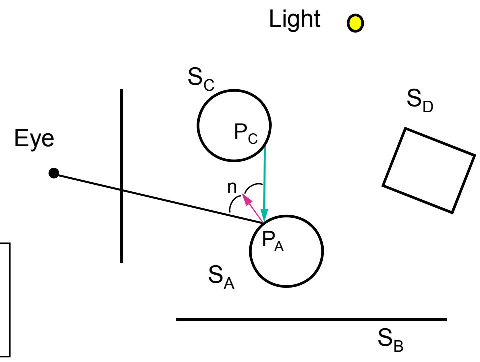
S_A	shiny, transparent
S_B,S_D	diffuse, opaque
S_C	shiny, opaque

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

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

What is the light that is reflected to P_A ?

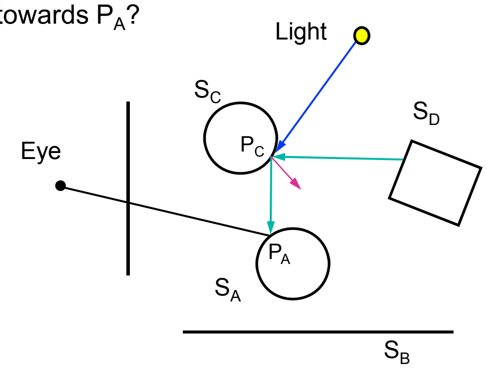
The color of P_C as viewed by P_A

What is the color of P_C reflected towards P_A?

Just like P_A :

Raytrace P_C ; i.e., compute the contributions from

1. Light sources
2. Reflection



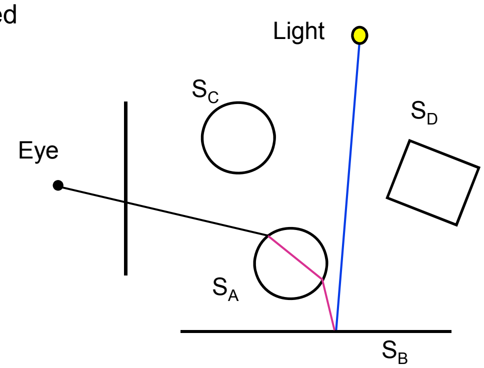
Refraction

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

Transparent materials

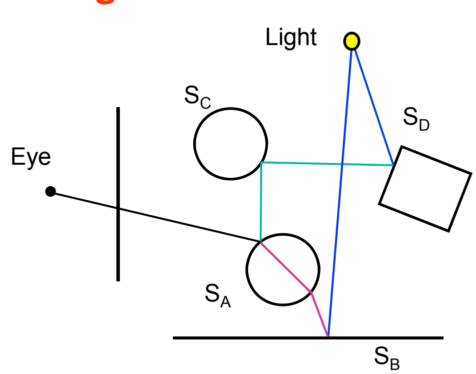
How do you compute the refracted contribution?

You raytrace the refracted ray



What Are We Missing?

Diffuse objects do not receive light from other objects, only from light sources

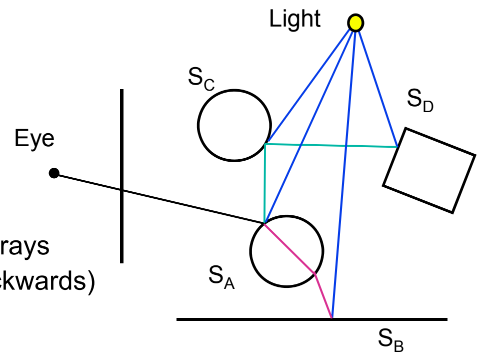


S_A	shiny, transparent
S_B,S_D	diffuse,opaque
S_C	shiny, opaque

Three Contributions Together

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

The color that the pixel is assigned comes from:
- Light sources
- Reflection from other objects
- Refraction from other objects



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

Ray Tracing

for each pixel on screen

determine ray from eye through pixel

find closest intersection of ray with an object

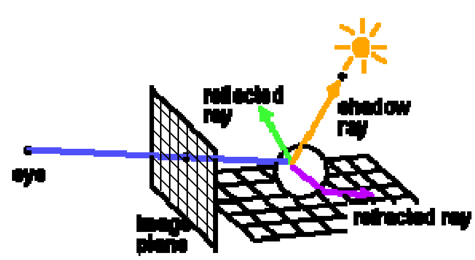
cast shadow ray(s) to the light source(s)

recursively cast reflected and refracted ray

calculate pixel color

paint pixel

end



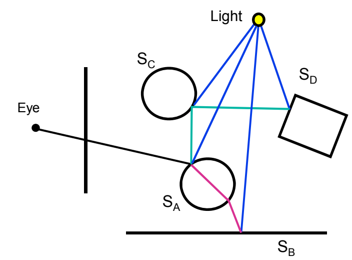
Backwards Ray Tracing Algorithm

For each pixel construct a ray: eye → pixel

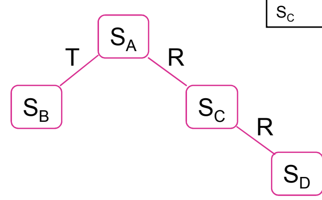
raytrace(ray)

P = closest intersection
color_local = ShadowRay(light$_1$, P) + …
+ ShadowRay(light$_N$, P)
color_reflect = raytrace(reflected_ray)
color_refract = raytrace(refracted_ray)
color = color_local +
+ k_{rfl}* color_reflect
+ k_{rfa}* color_refract

return(color)



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



How Many Levels of Recursion Should We Use?

The more the better

Infinite reflections at the limit

But at increasing computational expense

Stages of Ray Tracing

Setting the camera and the image plane

Computing a ray from the eye to every pixel and trace it in the scene

Computing object-ray intersections

Computing shadow, reflected, and refracted rays at each intersection

Setting Up the Camera

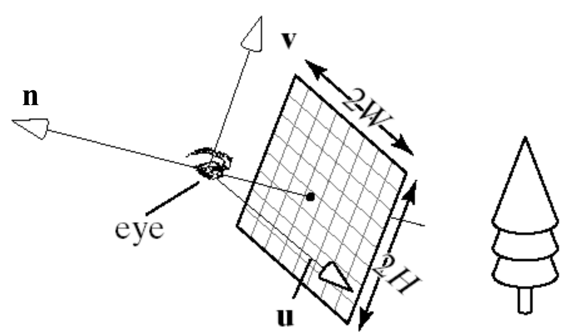
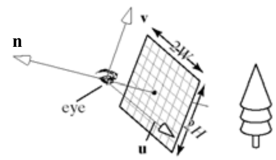


Image Parameters

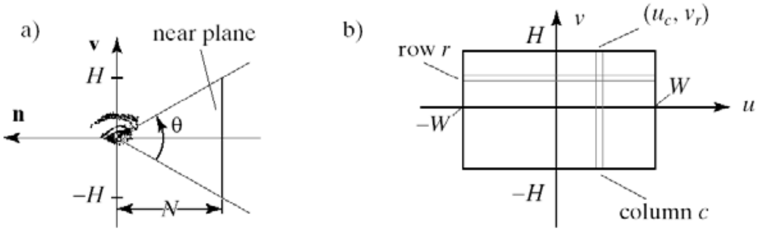


Width 2W, Height 2H

Number of pixels N_c x N_r

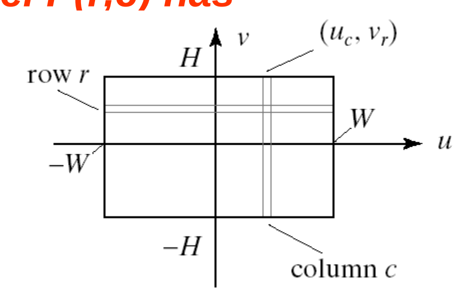
Camera coordinate system (eye, u,v,n)

Image plane at n = -N



Pixel Coordinates in Camera Coordinate System

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

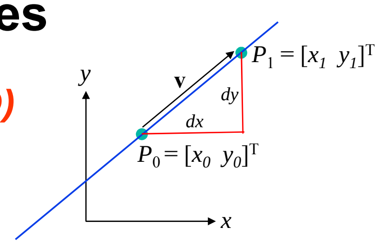


$$u_c = -W + W\frac{2c}{N_c - 1}, \quad c = 0, 1, \ldots, N_c - 1,$$

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

Reminder: Lines

Representations of a line (in 2D)



- Explicit $y = \alpha x + \beta$

  $y = m(x - x_0) + y_0; \quad m = \dfrac{dy}{dx} = \dfrac{y_1 - y_0}{x_1 - x_0}$

- Implicit $f(x, y) = (x - x_0)dy - (y - y_0)dx$

  if $f(x, y) = 0$ then (x, y) is **on** the line

  $f(x, y) > 0$ then (x, y) is **below** the line

  $f(x, y) < 0$ then (x, y) is **above** the line

- Parametric $x(t) = x_0 + t(x_1 - x_0)$

  $y(t) = y_0 + t(y_1 - y_0)$

  $t \in [0,1]$ for line segment, or $t \in [-\infty, \infty]$ for infinite line

  $\boxed{P(t) = P_0 + t(P_1 - P_0) \quad \text{or} \quad P(t) = P_0 + t\,\mathbf{v}}$

  $P(t) = (1 - t)P_0 + tP_1$

Ray Through Pixel

Lower left corner of pixel

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

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

Ray through pixel:

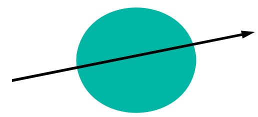
$$\text{ray}(r, c, t) = \text{eye} + t(P(r, c) - \text{eye})$$

$$= \text{eye} + t\left(-N\mathbf{n} + W\left(\frac{2c}{N_c - 1} - 1\right)\mathbf{u} + H\left(\frac{2r}{N_r - 1} - 1\right)\mathbf{v}\right)$$

Ray-Object Intersections

Intersection of ray with unit sphere at origin:

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



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

This is a quadratic equation

Solving the Quadratic Equation

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

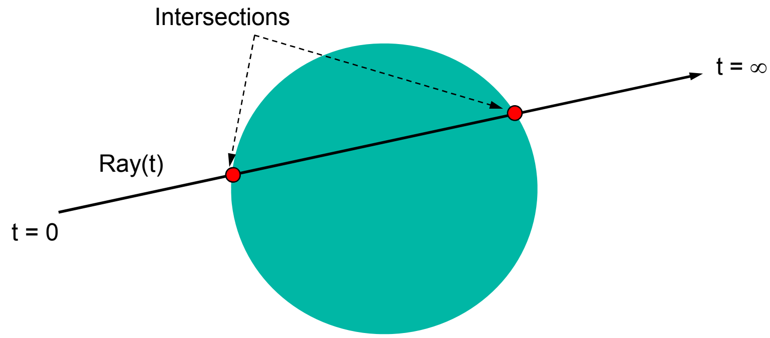
$$t_h = -\frac{B}{A} \pm \frac{\sqrt{B^2 - AC}}{A}$$
$$= -\frac{S \cdot \mathbf{c}}{|\mathbf{c}|^2} \pm \frac{\sqrt{(S \cdot \mathbf{c})^2 - |\mathbf{c}|^2 (|S|^2 - 1)}}{|\mathbf{c}|^2}$$

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

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

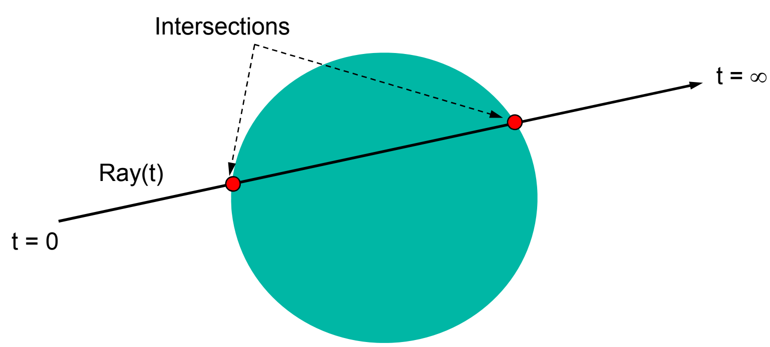
If $(B^2 - AC) > 0$ two solutions

First Intersection?

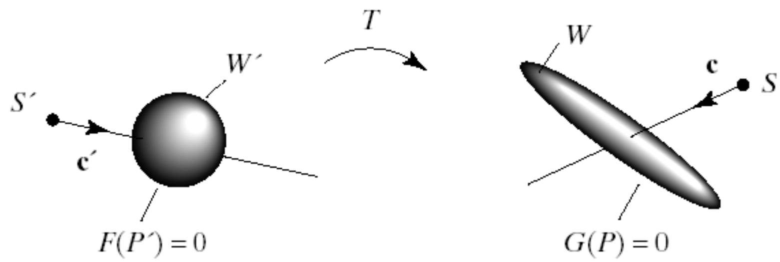


First Intersection?

$t_1 < t_2$

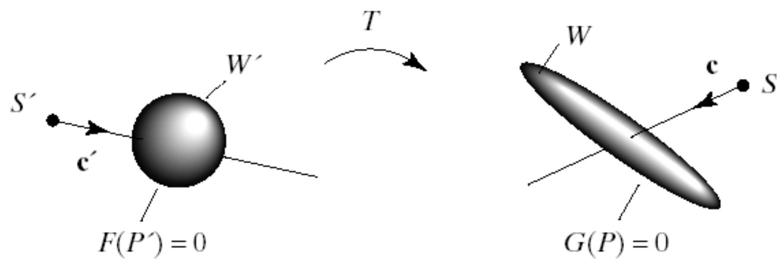


How Do We Deal With Transformed Primitives?



Where does $S + t\mathbf{c}$ intersect the transformed sphere G ?

Affine Transformation

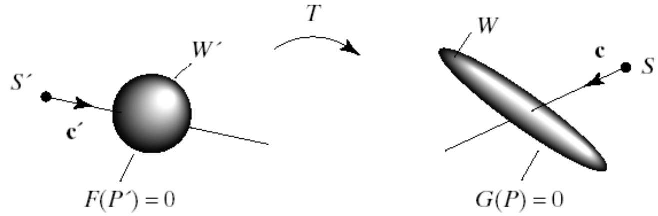


Implicit equation $G(P) = 0$

Untransformed implicit equation F(P') = 0

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

Affine Transformation



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

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

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

Final Intersection

Inverse transformed ray

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

- Drop 1 and 0 to get **r**'(t) in 3D space

For each object

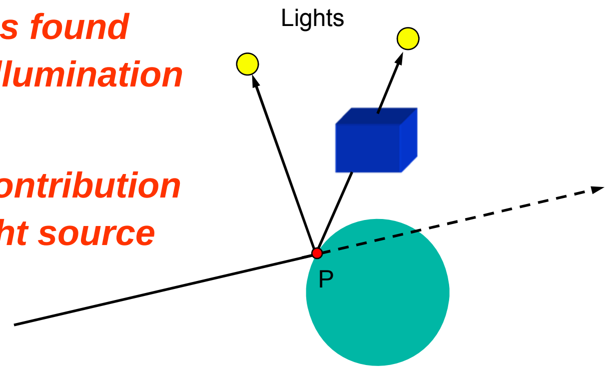
- Inverse transform ray, getting $S' + t\mathbf{c}'$
- Find t_h for intersection with the untransformed object
- Use t_h in the **untransformed ray** $S + t\mathbf{c}$ to find the point of intersection with the transformed object

Shadow Ray

For each light source, intersect shadow ray (from point P towards light source) with all objects

If no intersection is found
apply local illumination
at point P

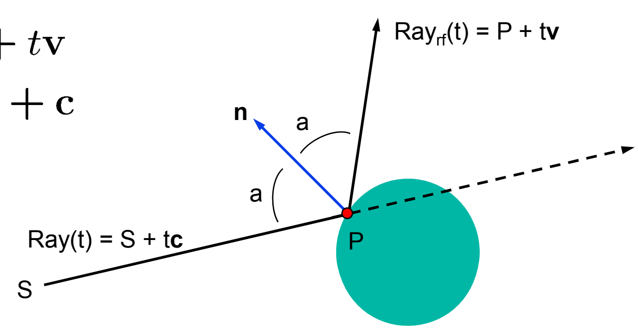
If in shadow, no contribution
from that light source



Reflected Ray

Raytrace the reflected ray

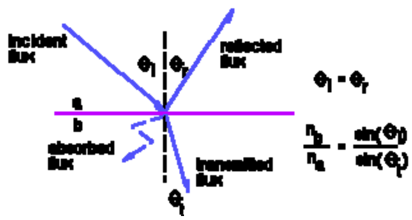
$$\text{Ray}(t) = S + t\mathbf{c}$$
$$\text{Ray}_{\text{rf}}(t) = P + t\mathbf{v}$$
$$\mathbf{v} = -2(\mathbf{n} \cdot \mathbf{c})\mathbf{n} + \mathbf{c}$$

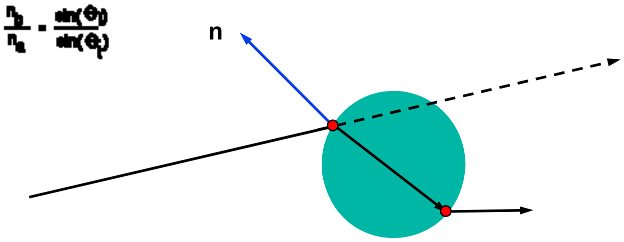


Refracted Ray

Raytrace the refracted ray

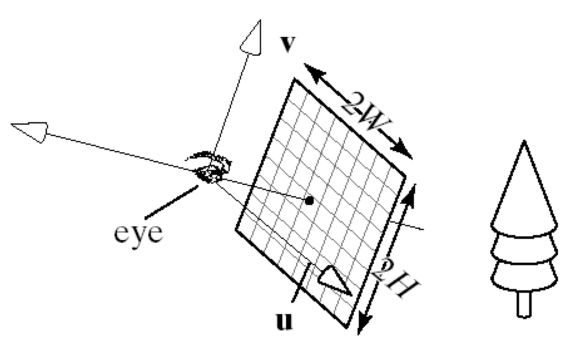
Snell's law





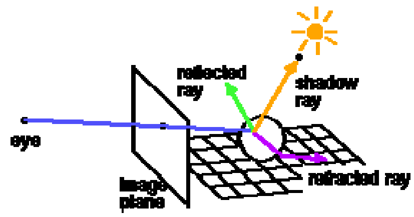
All Together

color(r,c) = color_shadow_ray +
*k_{rfl} * color_reflected +*
*k_{rfa} * color_refracted*



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 r with objects

clocal = Sum(shadowRays(P,Light$_i$))

c_{rfl} = raytrace(r_{rfl})

c_{rfa} = raytrace(r_{rfa})

return c = clocal + k_{rfl}*c_{rfl} + k_{rfa}*c_{rfa}

end

A Ray Tracer in Postscript!

```
%! Tiny RayTracing by HAYAKAWA,Takashi(h-takasi@isea.is.titech.ac.jp)
/p/floor/S/add/A/copy/n/exch/i/index/J/ifelse/r/roll/e/sqrt/H{count 2 idiv exch
repeat}def/q/gt/h/exp/t/and/C/neg/T/dup/Y/pop/d/mul/w/div/s/cvi/R/rlineto{load
def}H/c(j1idj2id42rd)/G(140N7)/Q(31C85d4)/B(V0R0VRVC0R)/K(WCVW)/U(4C577d7)300
T translate/I(3STinTinTinY)/l(993dC99Cc96raN)/k(X&E9!&1!J)/Z(blxC1SdC9n5dh)/j
(43r)/O(Y43d9rE3IaN96r63rvx2dcaN)/z(&93r6IQO2Z4o3AQYaNlxS2w!)/N(3A3Axe1nwc)/W
270 def/L(1i2A00053r45hNvQXz&vUX&UOvQXzFJ!FJ!J)/D(cjS5o32rS4oS3o)/v(6A)/b(7o)
/F(&vGYx4oGbxSd0nq&3IGbxSGY4Ixwca3AlvvUkbQkdbGYx4ofwnw!&vlx2w13wSb8Z4wS!J!)/X
(4I3Ax52r8Ia3A3Ax65rTdCS4iw5o5IxnwTTd32rCST0q&eCST0q&D1!&EYE0!J!&EYEY0!J0q)/V
1 def/x(jd5o32rd4odSS)/a(1CD)/E(YYY)/o(1r)/f(nY9wn7wpSps1t1S){[n{( )T 0 4 3 r
put T(/)q{T(9)q{cvn}{s}J}{($)q{[}{]}J}J cvx}forall]cvx def}H K{K{L setgray
moveto B fill}for Y}for showpage
```

Efficiency Issues

Computationally expensive

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

Advanced Concepts

Participating media

Translucency

Sub-surface scattering (e.g., human skin)

Aperture effects, depth of field

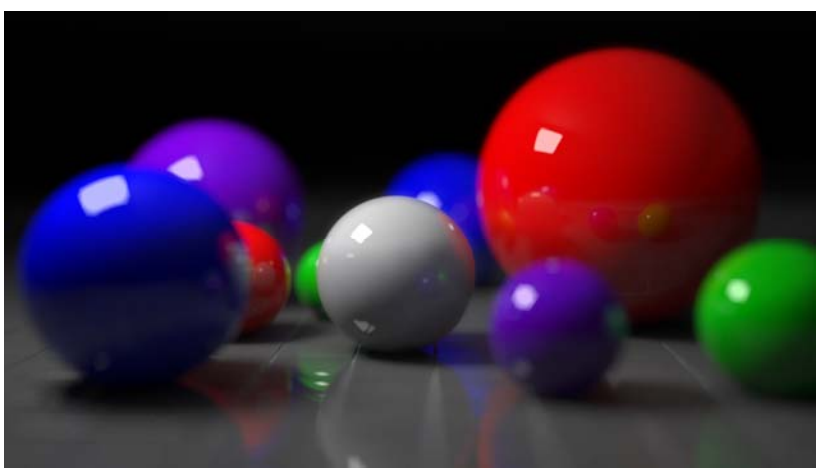
Photon mapping

- Combination of eye-based and light-based ray tracing
- Good for rendering caustic effects

Caustics

Depth of Field and Aperture Effects

Hexagonal aperture



Ray Tracing Summary

Recursive

Computationally expensive

Good for reflection and refraction effects

Comparison

Ray tracing *vs* *Radiosity*

Direct Lighting

Indirect Lighting