Ray Tracing- CSE160

- What ray tracing looks like
- Basic algorithm
- Rays
- CSG
- Design a raytracer
- Distributed ray tracing
- Photon Mapping
- Convolution Theorem
- Administrative
- Q&A

What ray tracing looks like

Today: Ray Tracing

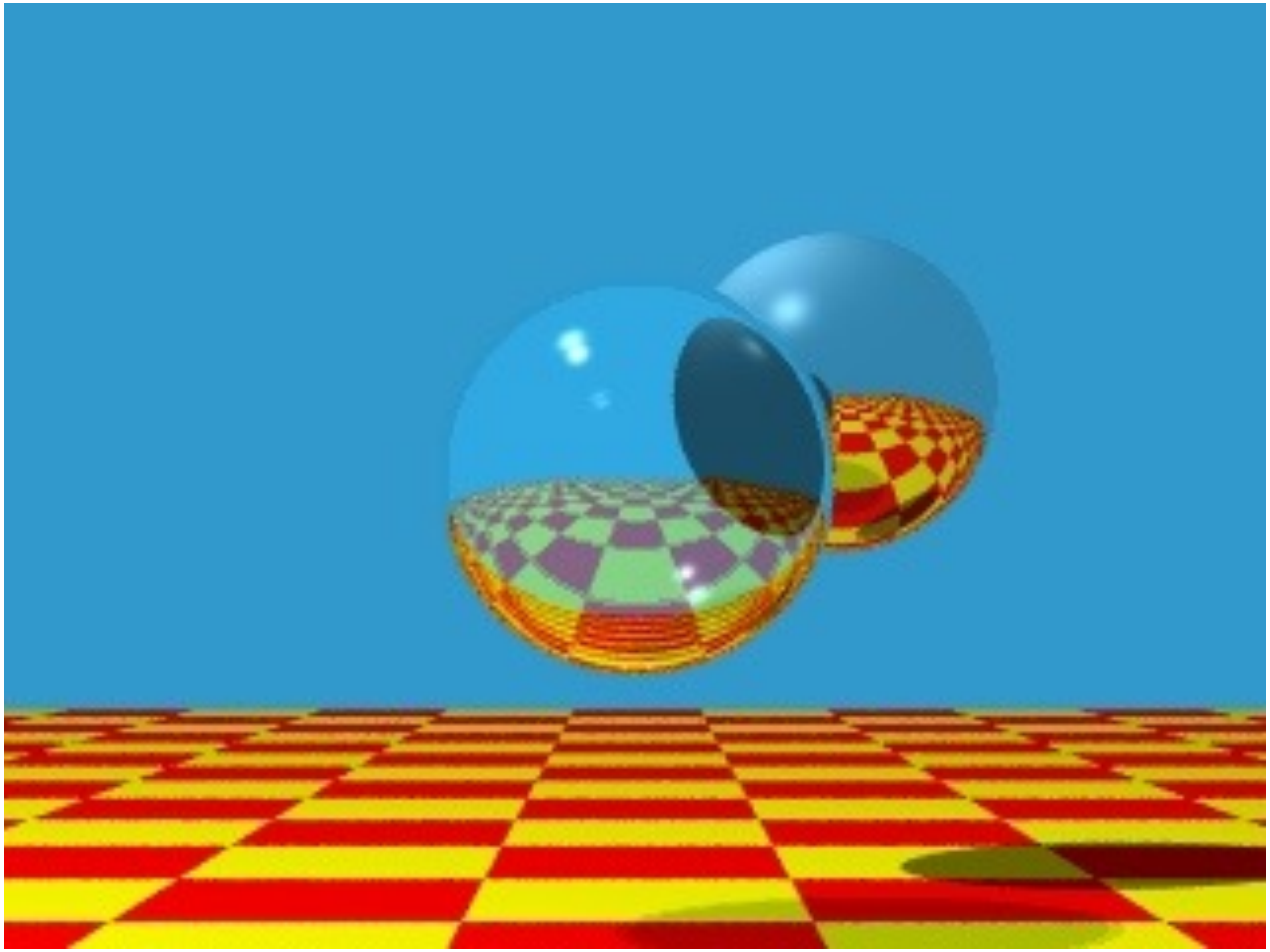
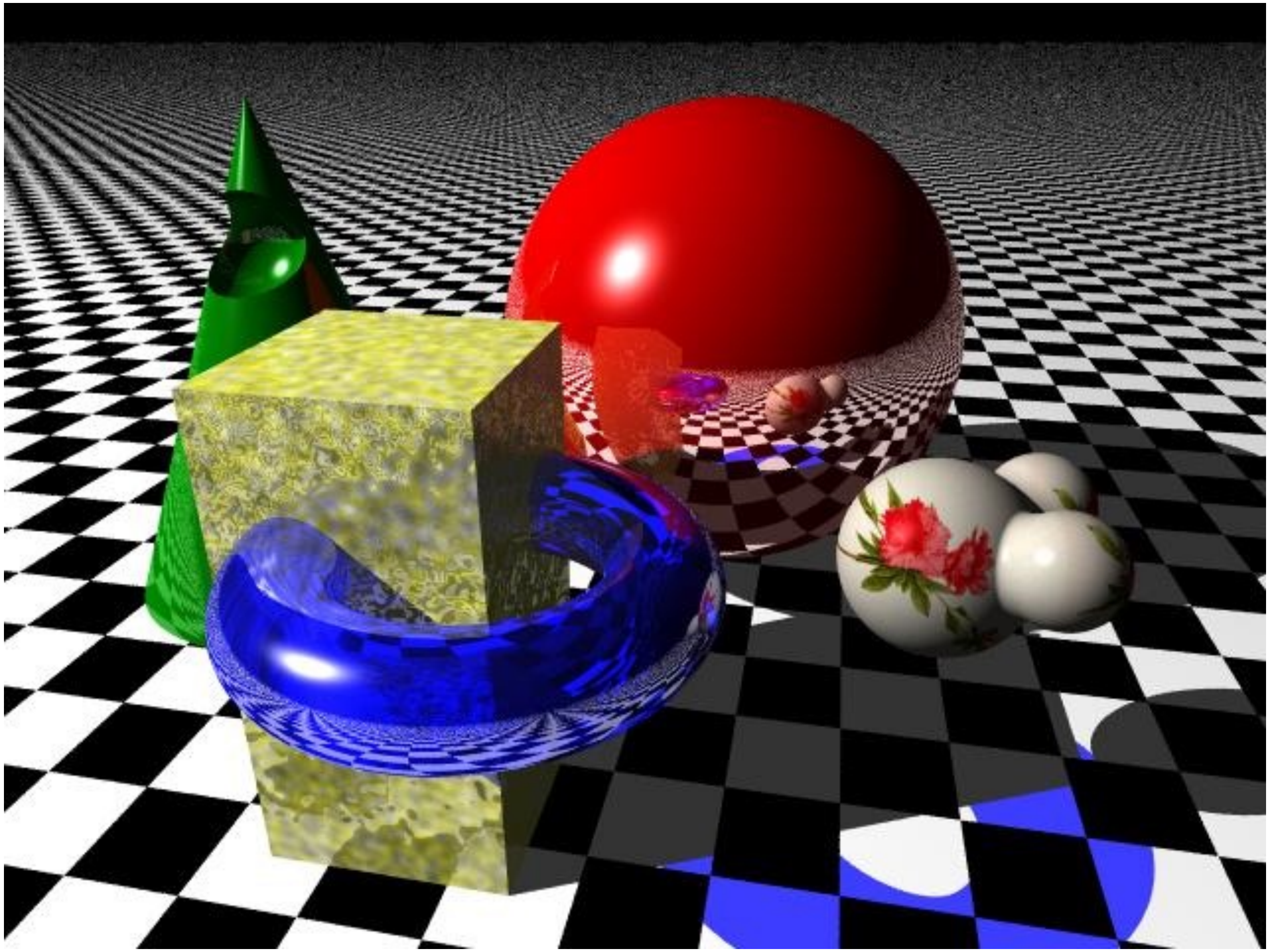
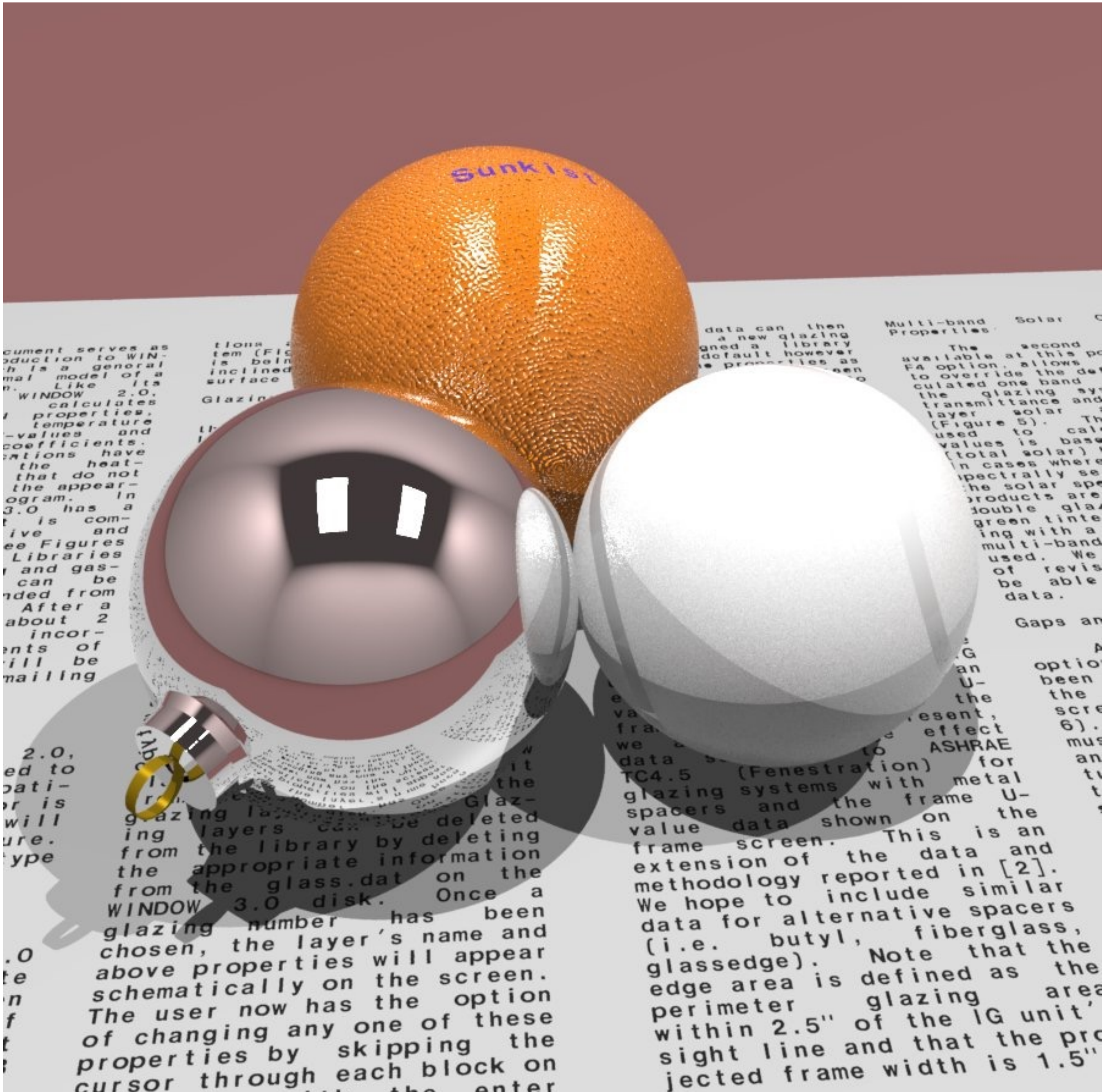


Image by Turner Whitted



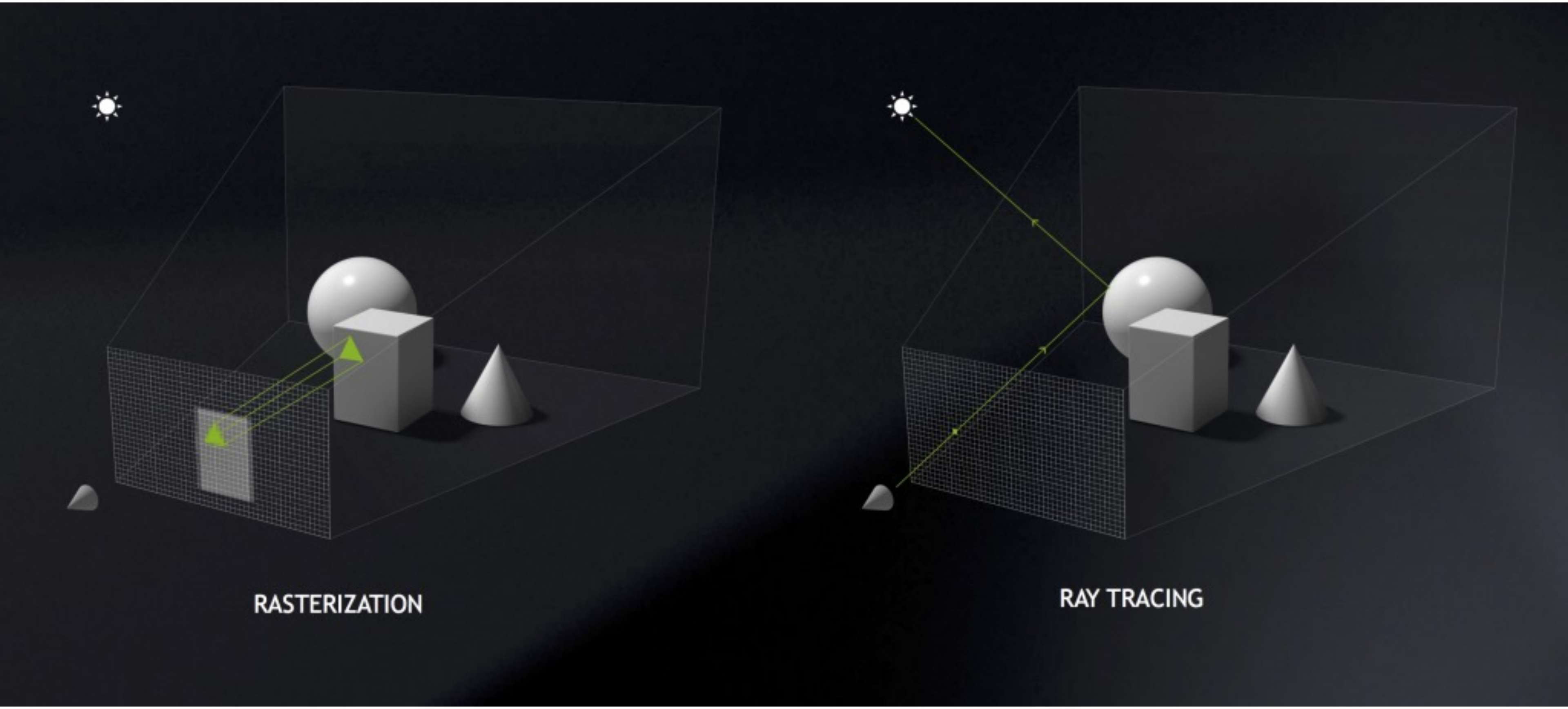

Sunkist
from the library by deleting
the appropriate information
from the glass.dat on the
WINDOW 3.0 disk. Once a
glazing number has been
chosen, the layer's name and
above properties will appear
schematically on the screen.
The user now has the option
of changing any one of these
properties by skipping the
cursor through each block on
TC4.5 (Fenestration) for
glazing systems with metal
spacers and the frame U-
value data shown on the
frame screen. This is an
extension of the data and
methodology reported in [2].
We hope to include similar
data for alternative spacers
(i.e. butyl, fiberglass,
glassedge). Note that the
edge area is defined as the
perimeter glazing area
within 2.5" of the IG unit'
jected frame width is 1.5"


NEXT-GEN
EXPLAINED
ON
OFF

RAY TRACING TECHNIQUES

Reflections	Reflections	Shadows	Global Illumination	Reflections	Reflections	Reflections
Shadows	Shadows		Ambient Occlusion		Shadows	Shadows
Ambient Occlusion	Caustics					

Basic Algorithm


RASTERIZATION
RAY TRACING

Basic rasterization algorithm

Sample = 2D point

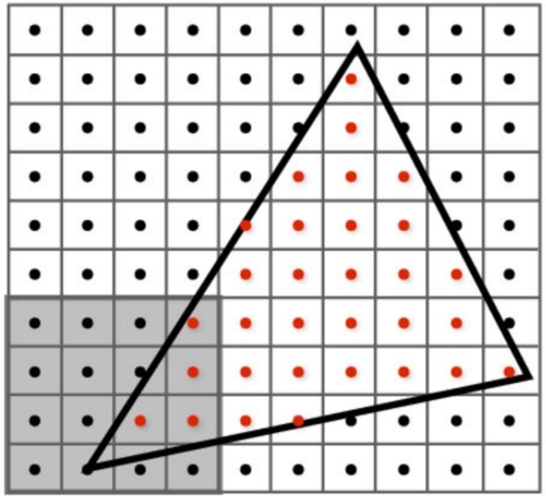
Coverage: 2D triangle/sample tests (does projected triangle cover 2D sample point)

Occlusion: depth buffer

```
initialize z_closest[] to INFINITY              // store closest-surface-so-far for all samples
initialize color[]                              // store scene color for all samples
for each triangle t in scene:                   // loop 1: over triangles
    t_proj = project_triangle(t)
    for each 2D sample s in frame buffer:       // loop 2: over visibility samples
        if (t_proj covers s)
            compute color of triangle at sample
            if (depth of t at s is closer than z_closest[s])
                update z_closest[s] and color[s]
```

"Given a triangle, find the samples it covers"

(finding the samples is relatively easy since they are distributed uniformly on screen)



More efficient hierarchical rasterization:

For each TILE of image

If triangle overlaps tile, check all samples in tile

Basic ray casting algorithm

Sample = a ray in 3D

Coverage: 3D ray-triangle intersection tests (does ray "hit" triangle)

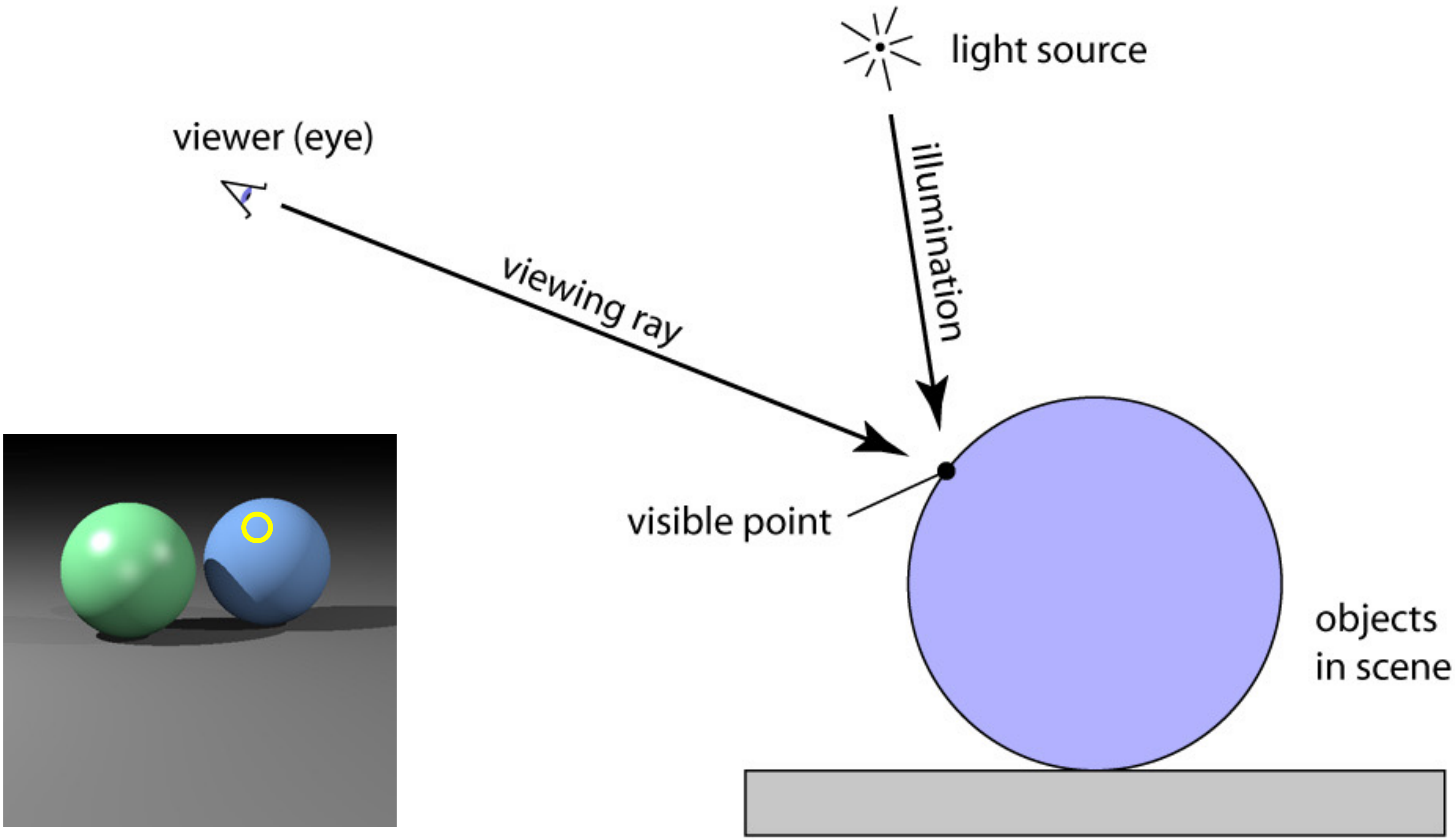
Occlusion: closest intersection along ray

```
initialize color[]                                     // store scene color for all samples
for each sample s in frame buffer:                     // loop 1: over visibility samples (rays)
    r = ray from s on sensor through pinhole aperture
    r.min_t = INFINITY                                 // only store closest-so-far for current ray
    r.tri = NULL;
    for each triangle tri in scene:                      // loop 2: over triangles
        if (intersects(r, tri)) {                        // 3D ray-triangle intersection test
            if (intersection distance along ray is closer than r.min_t)
                update r.min_t and r.tri = tri;
        }
    color[s] = compute surface color of triangle r.tri at hit point
```

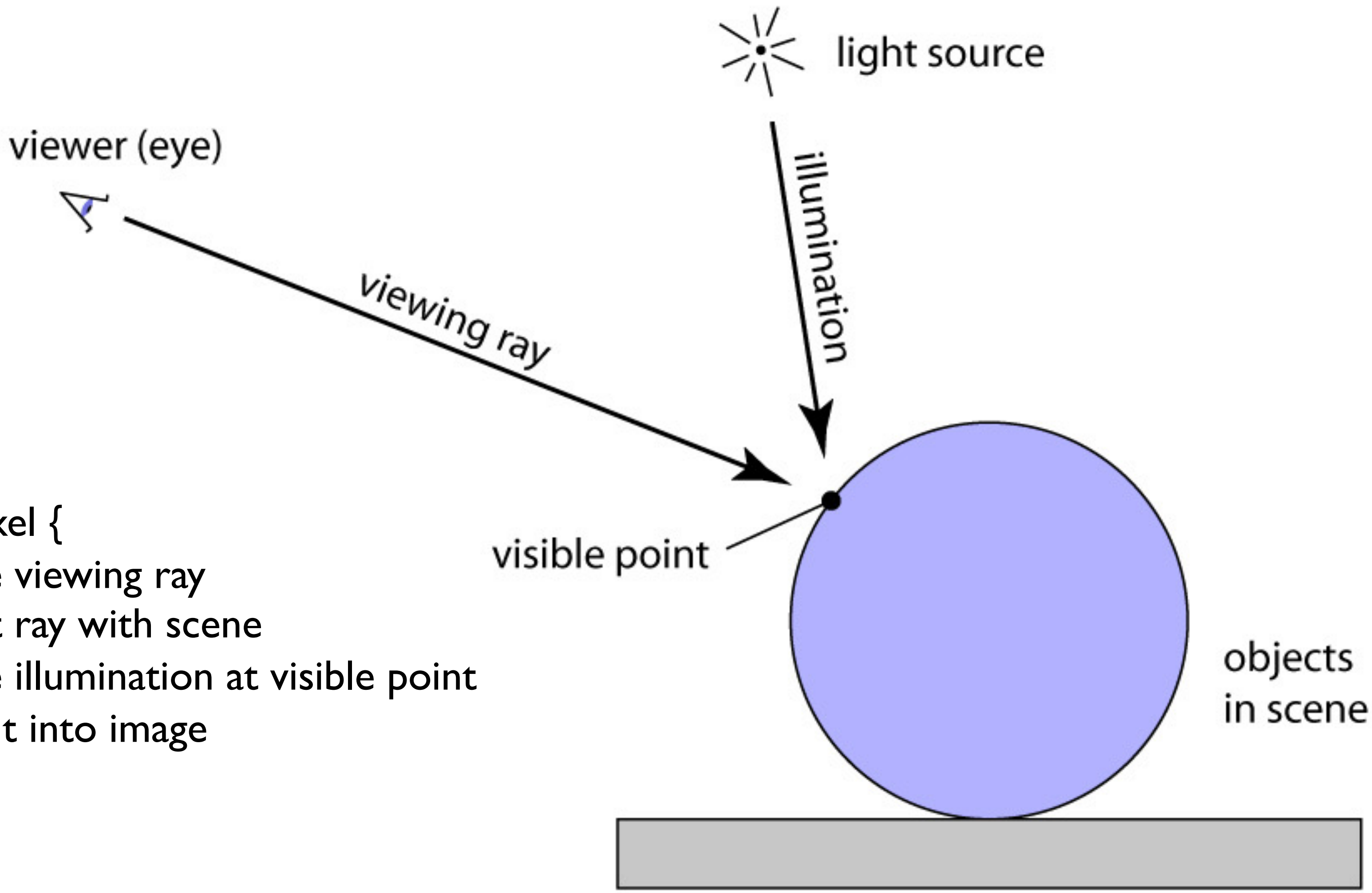
Compared to rasterization approach: just a reordering of the loops!

"Given a ray, find the closest triangle it hits."

Ray tracing idea



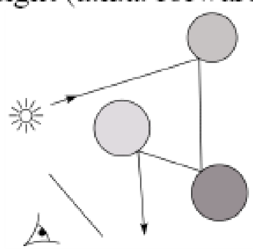
Ray tracing algorithm



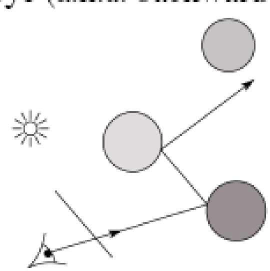
for each pixel {
 compute viewing ray
 intersect ray with scene
 compute illumination at visible point
 put result into image
 }

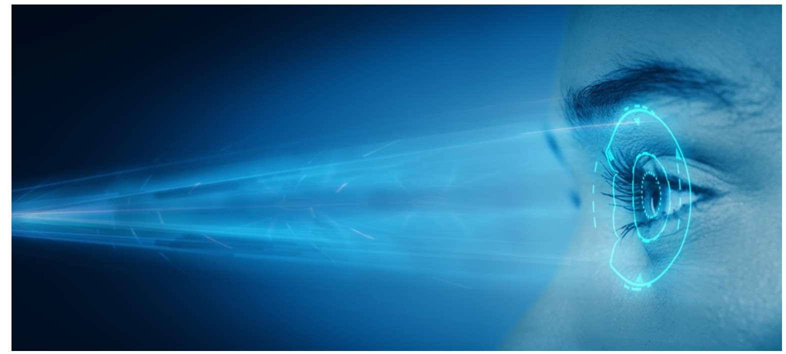
Eye vs. Light

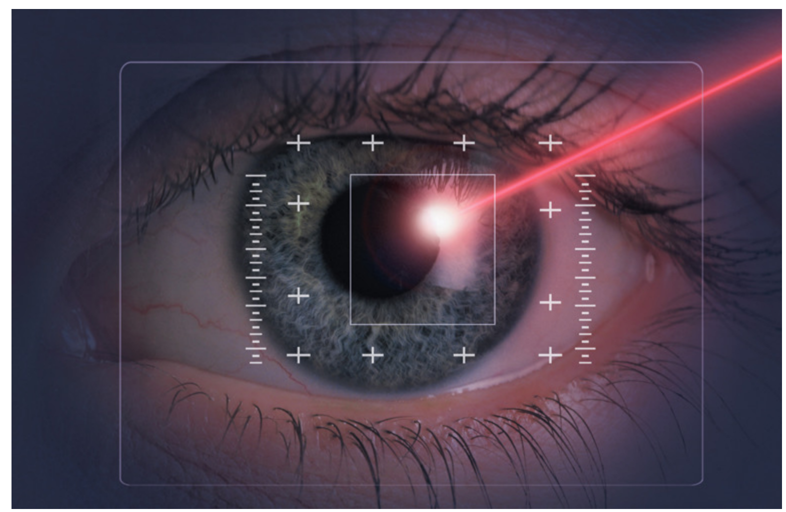
- Starting at the light (a.k.a. forward ray tracing, photon tracing)



- Starting at the eye (a.k.a. backward ray tracing)

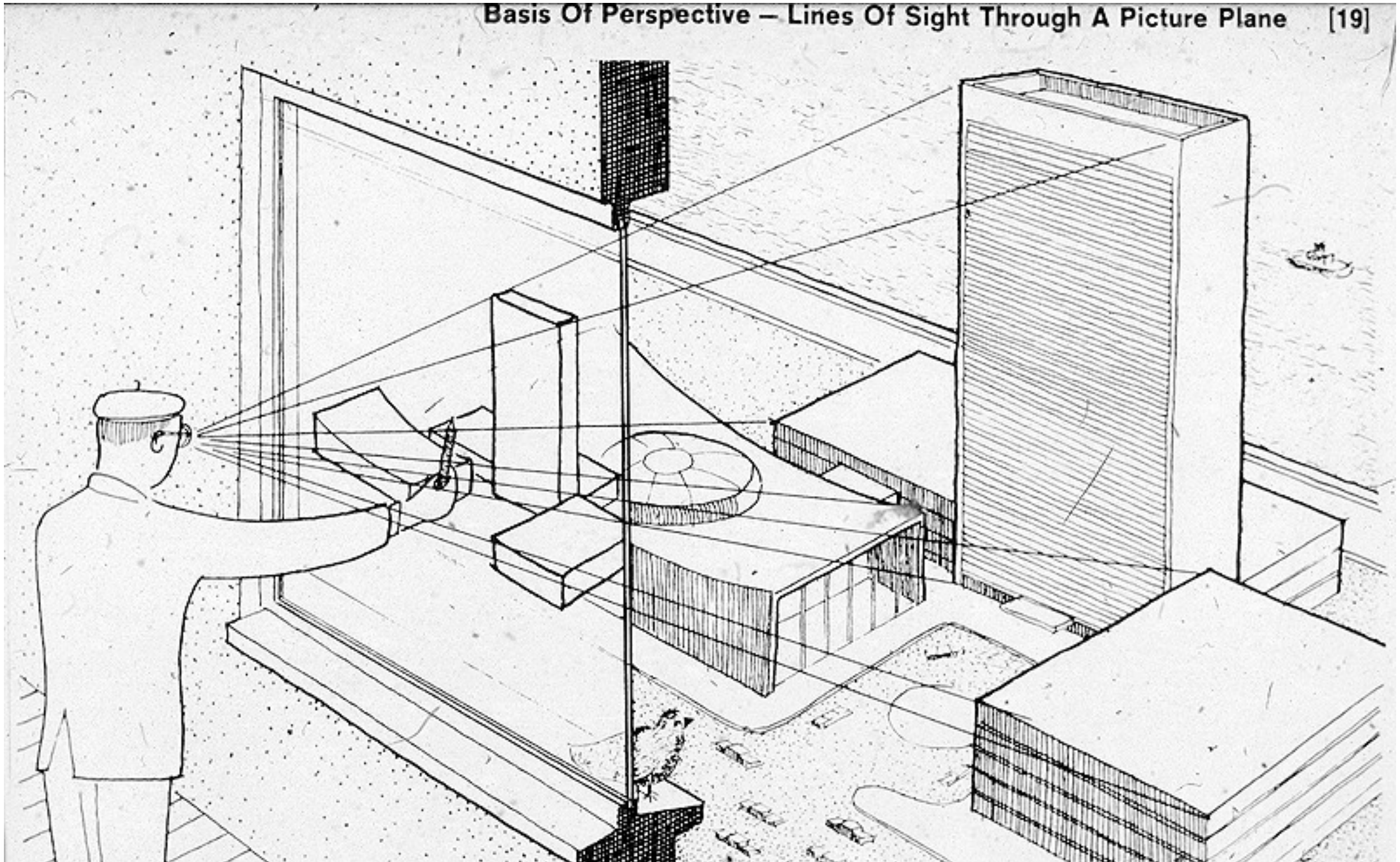






Rays

Analogy to drawing

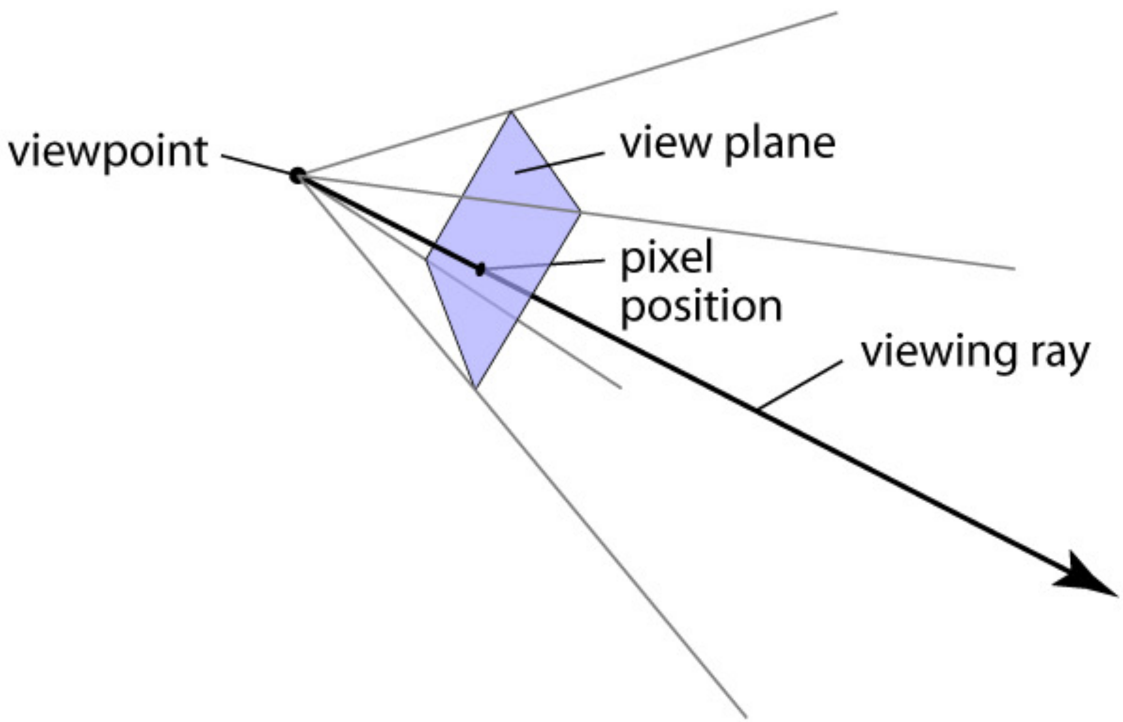


The concept of the picture plane may be better understood by looking through a window or other transparent plane from a fixed viewpoint. Your lines of sight, the multitude of straight lines leading from your eye to the subject, will all intersect this plane. Therefore, if you were to reach out with a grease pencil and draw the image of the subject on this plane you would be "tracing out" the infinite number of points of intersection of sight rays and plane. The result would be that you would have "transferred" a real three-dimensional object to a two-dimensional plane.

[CS 417 Spring 2002]

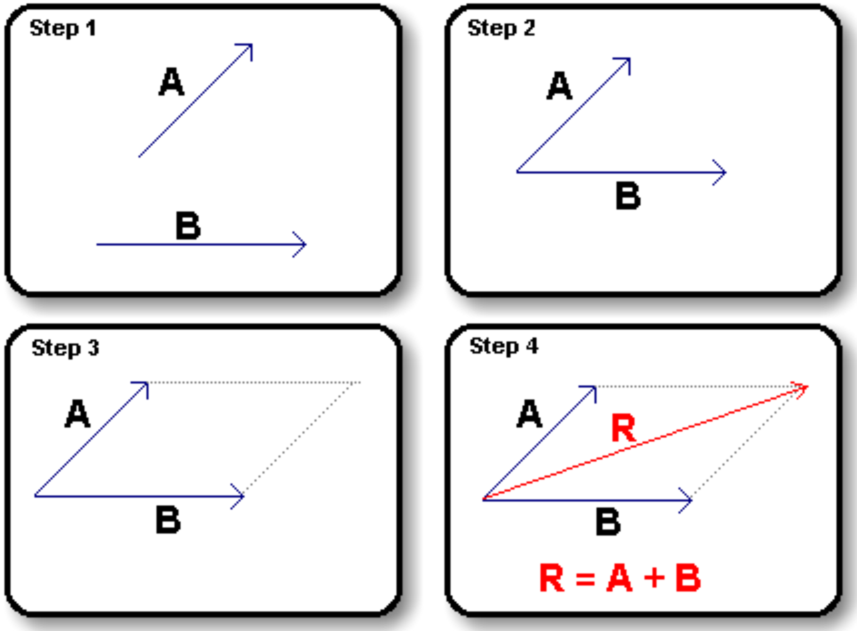
Generating eye rays

- Use window analogy directly

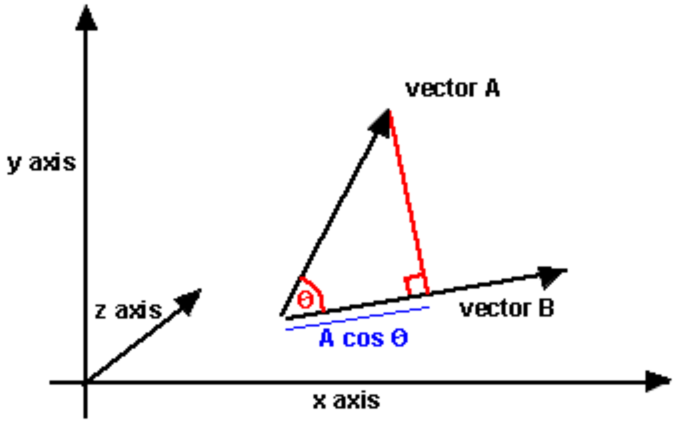


Vector math review

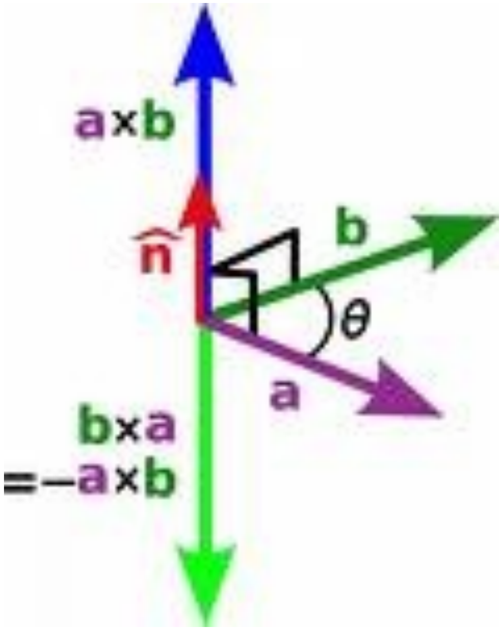
- Vectors and points
- Vector operations
  - addition
  - scalar product
- More products
  - dot product
  - cross product



Dot product



Cross product

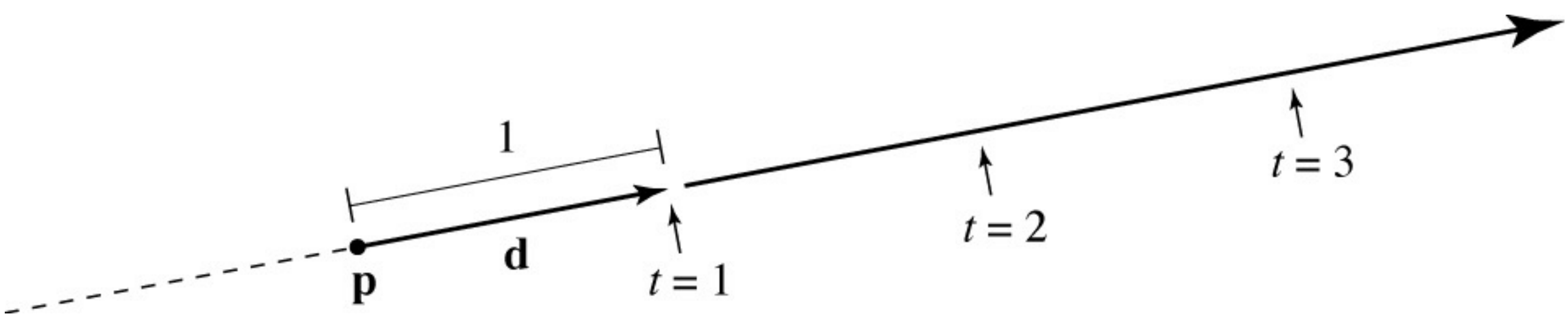


Ray: a half line

- Standard representation: point **p** and direction **d**

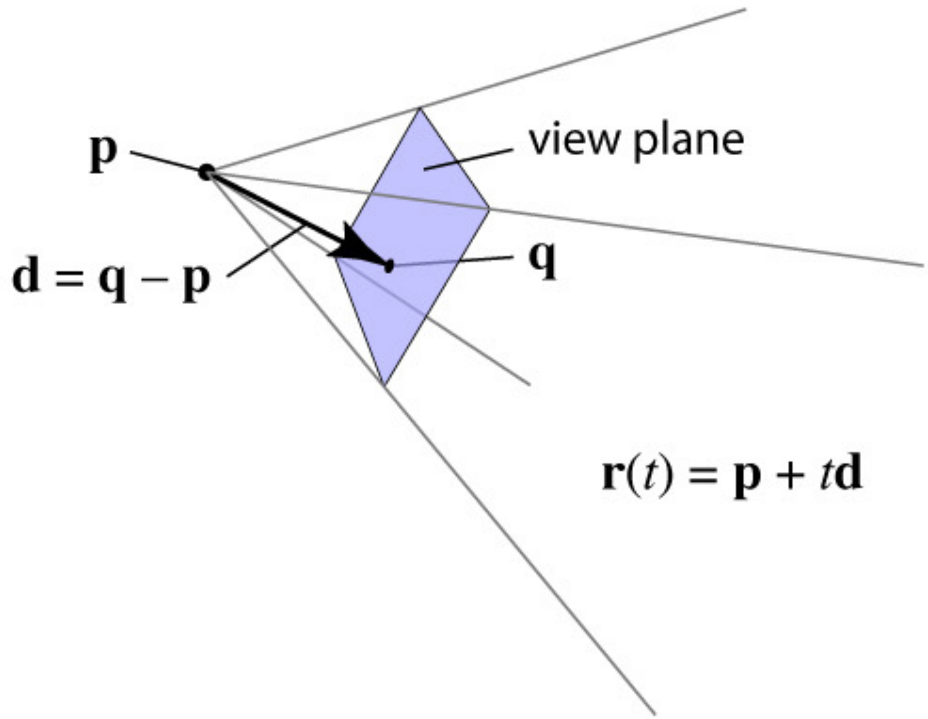
  $$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

  - this is a *parametric equation* for the line
  - lets us directly generate the points on the line
  - if we restrict to $t > 0$ then we have a ray
  - note replacing **d** with a**d** doesn't change ray ($a > 0$)



Generating eye rays

- Just need to compute the view plane point **q**:



  – we won't worry about the details for now

Ray-sphere intersection: algebraic

- Condition 1: point is on ray

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

- Condition 2: point is on sphere
  - assume unit sphere; see Shirley or notes for general

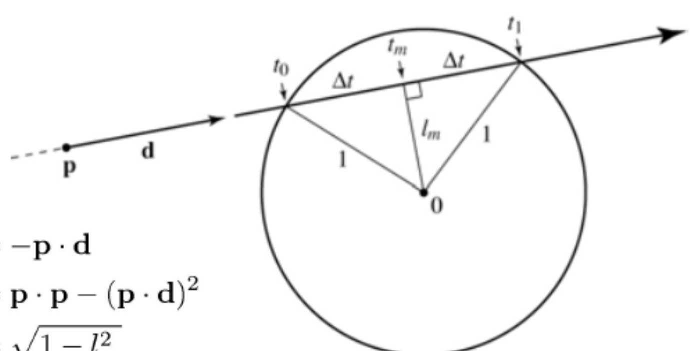
$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$
$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- Substitute:

$$(\mathbf{p} + t\mathbf{d}) \cdot (\mathbf{p} + t\mathbf{d}) - 1 = 0$$

  - this is a quadratic equation in *t*

Ray-sphere intersection: geometric



$$t_m = -\mathbf{p} \cdot \mathbf{d}$$
$$l_m^2 = \mathbf{p} \cdot \mathbf{p} - (\mathbf{p} \cdot \mathbf{d})^2$$
$$\Delta t = \sqrt{1 - l_m^2}$$
$$= \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$
$$t_{0,1} = t_m \pm \Delta t = -\mathbf{p} \cdot \mathbf{d} \pm \sqrt{(\mathbf{p} \cdot \mathbf{d})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

Ray-triangle intersection

- Condition 1: point is on ray

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

- Condition 2: point is on plane

$$(\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} = 0$$

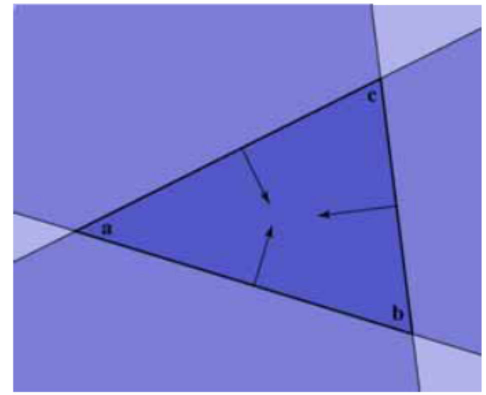
- Condition 3: point is on the inside of all three edges
- First solve 1&2 (ray–plane intersection)
  - substitute and solve for t:

$$(\mathbf{p} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

$$t = \frac{(\mathbf{a} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

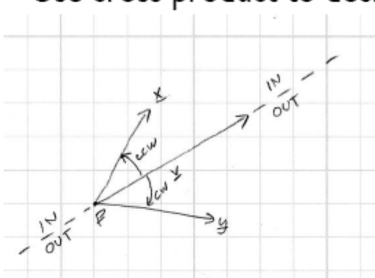
Ray-triangle intersection

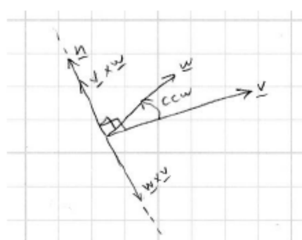
- In plane, triangle is the intersection of 3 half spaces



Inside-edge test

- Need outside vs. inside
- Reduce to clockwise vs. counterclockwise
  - vector of edge to vector to **x**
- Use cross product to decide



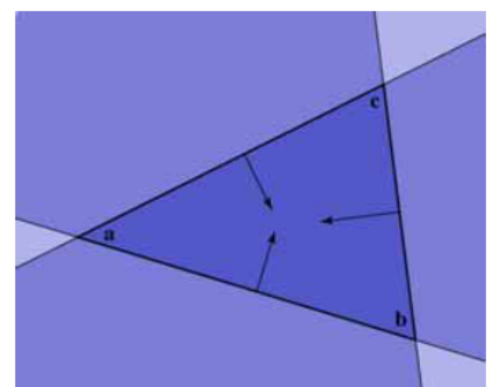


Ray-triangle intersection

$$(\mathbf{b} - \mathbf{a}) \times (\mathbf{x} - \mathbf{a}) \cdot \mathbf{n} > 0$$

$$(\mathbf{c} - \mathbf{b}) \times (\mathbf{x} - \mathbf{b}) \cdot \mathbf{n} > 0$$

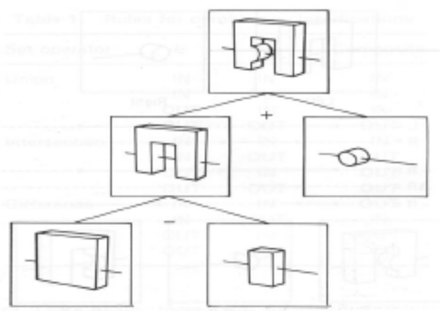
$$(\mathbf{a} - \mathbf{c}) \times (\mathbf{x} - \mathbf{c}) \cdot \mathbf{n} > 0$$



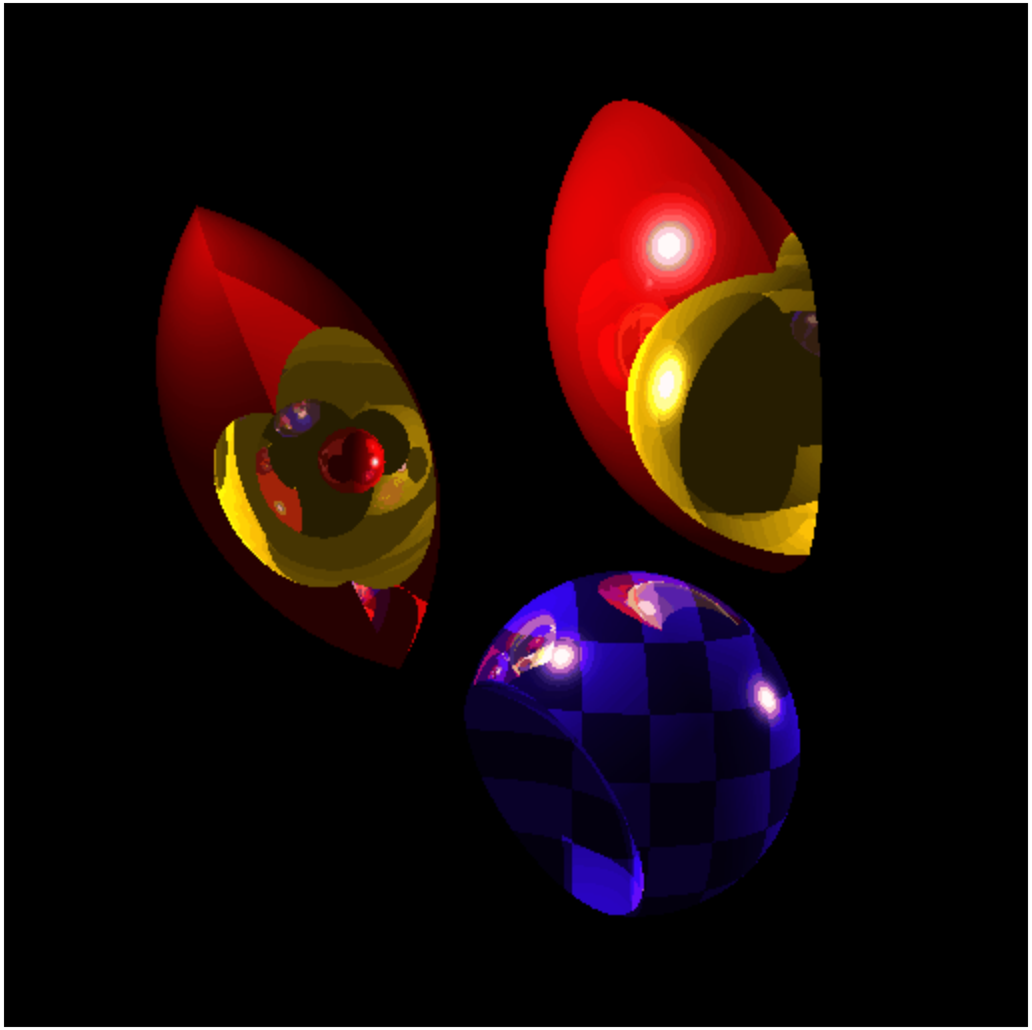
Constructive solid geometry

CSG

- CSG (constructive solid geometry) is an incredibly powerful way to create complex scenes from simple primitives.



- CSG is a modeling technique; basically, we only need to modify ray-object intersection.



Design a raytracer

Class designs a ray tracing algorithm

(Small group: write pseudo-code)

(Less than 10 lines code)

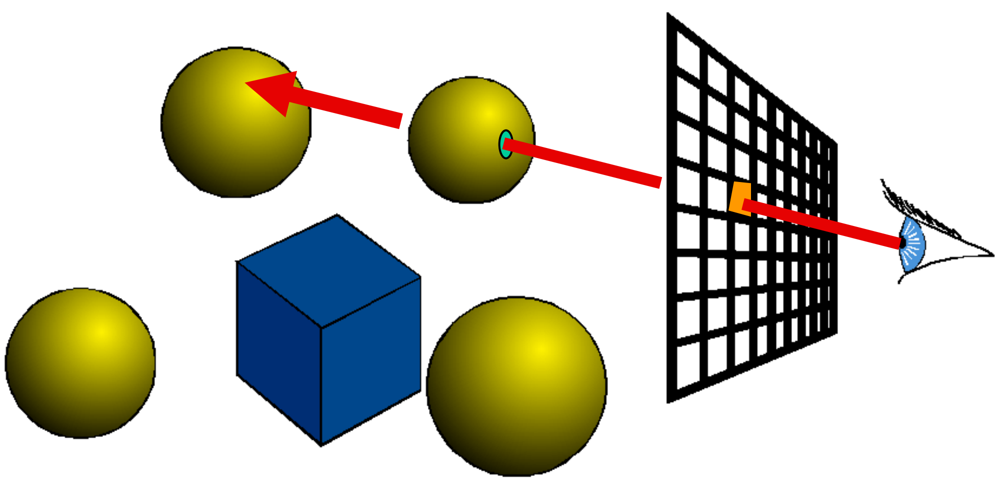
Ray Casting (a.k.a. Ray Shooting)

```
for every pixel
  construct a ray
  for every object
      intersect ray with object
```

Complexity?

$O(n * m)$

n = number of objects, m = number of pixels



Objects (no lighting)

Add lighting to your code

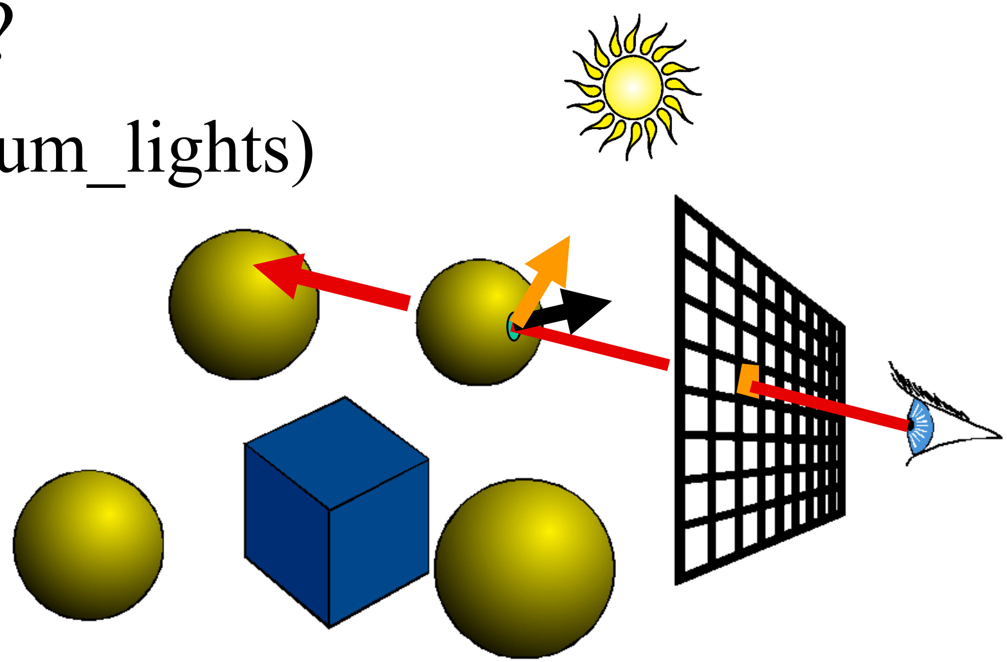
Ray Casting with Phong Shading

When you've found the closest intersection:

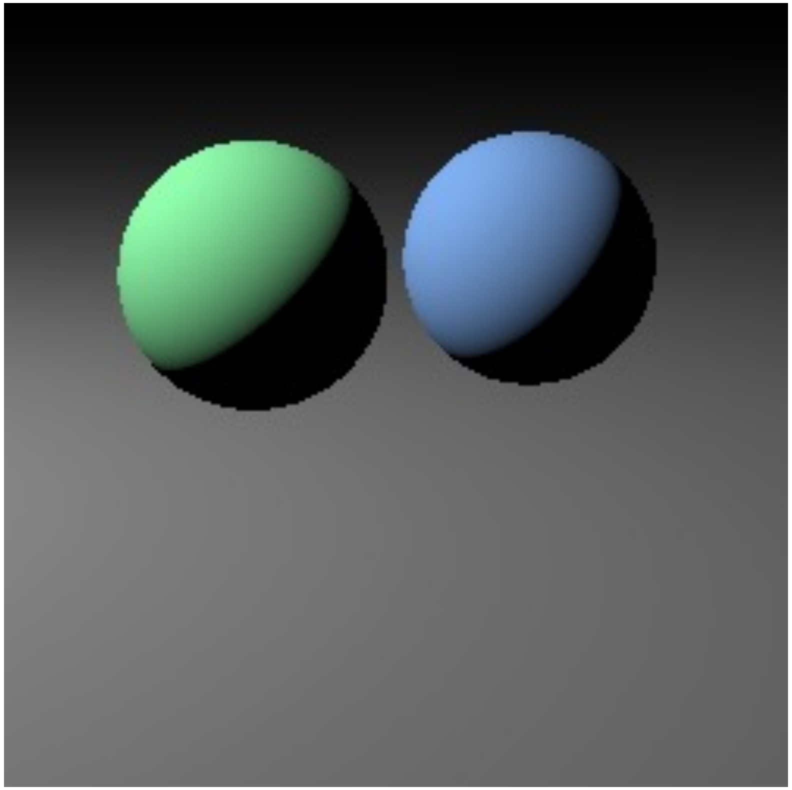
```
color = ambient*hit->getMaterial()->getDiffuseColor()
for every light
    color += hit->getMaterial()->Shade
                (ray, hit, directionToLight, lightColor)
return color
```

Complexity?

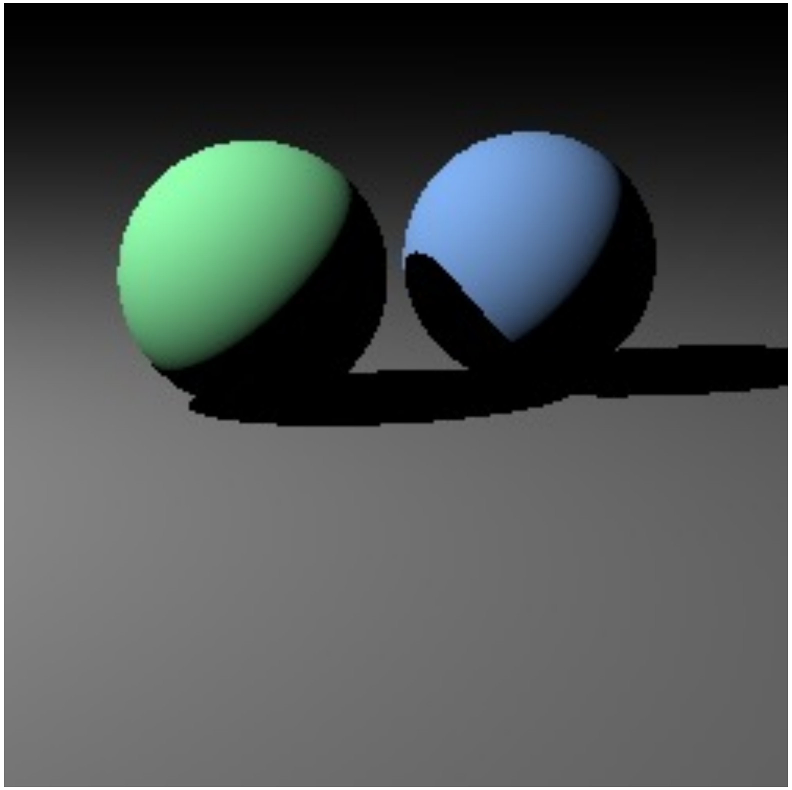
O(n * m * num_lights)



Add lighting



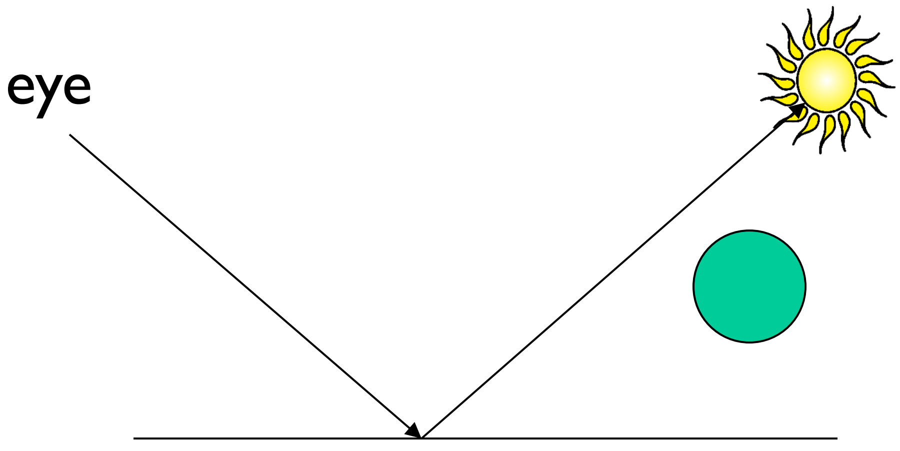
Add shadows

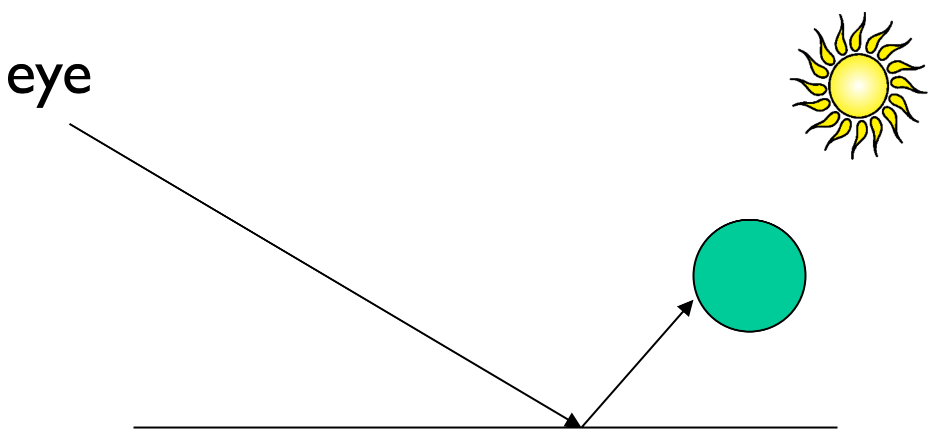


Add shadows to your code

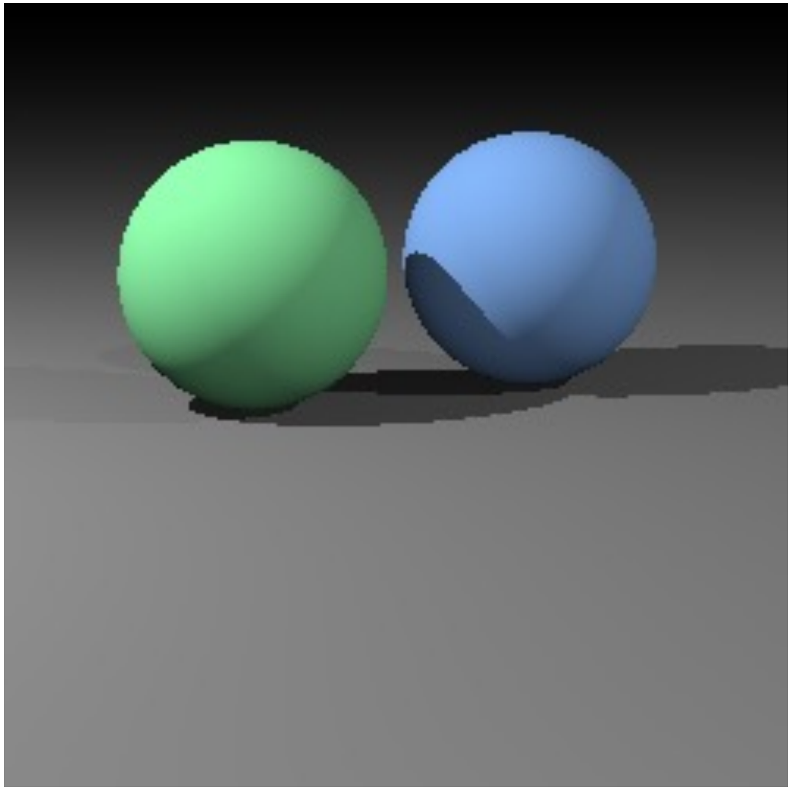
Q: How to calculate shadow

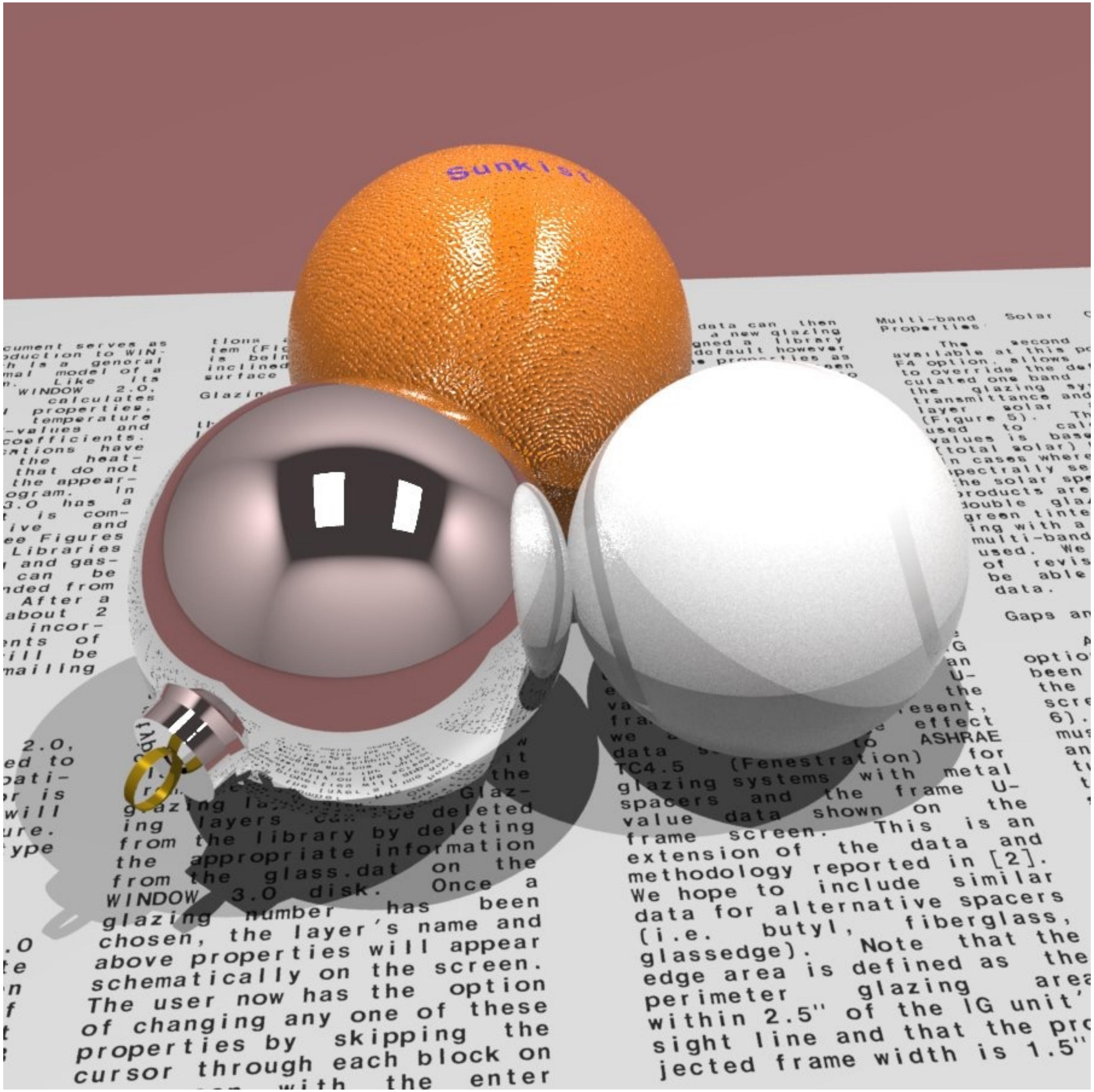
- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions


eye


eye

Multiple lights

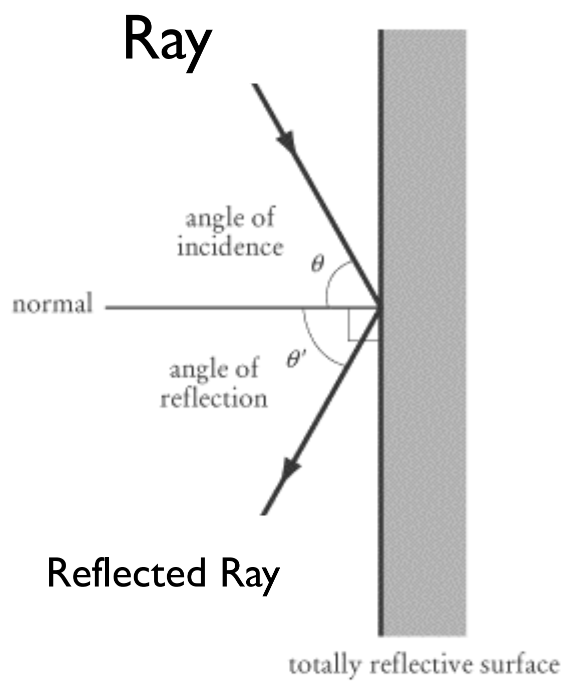



Sunkist
glazing layers can be deleted
from the library by deleting
the appropriate information
from the glass.dat on the
WINDOW 3.0 disk. Once a
glazing number has been
chosen, the layer's name and
above properties will appear
schematically on the screen.
The user now has the option
of changing any one of these
properties by skipping the
cursor through each block on
TC4.5 (Fenestration) for
glazing systems with metal
spacers and the frame U-
value data shown on the
frame screen. This is an
extension of the data and
methodology reported in [2].
We hope to include similar
data for alternative spacers
(i.e. butyl, fiberglass,
glassedge). Note that the
edge area is defined as the
perimeter glazing area
within 2.5" of the IG unit'
sight line and that the pro
jected frame width is 1.5"

Add reflection to your code

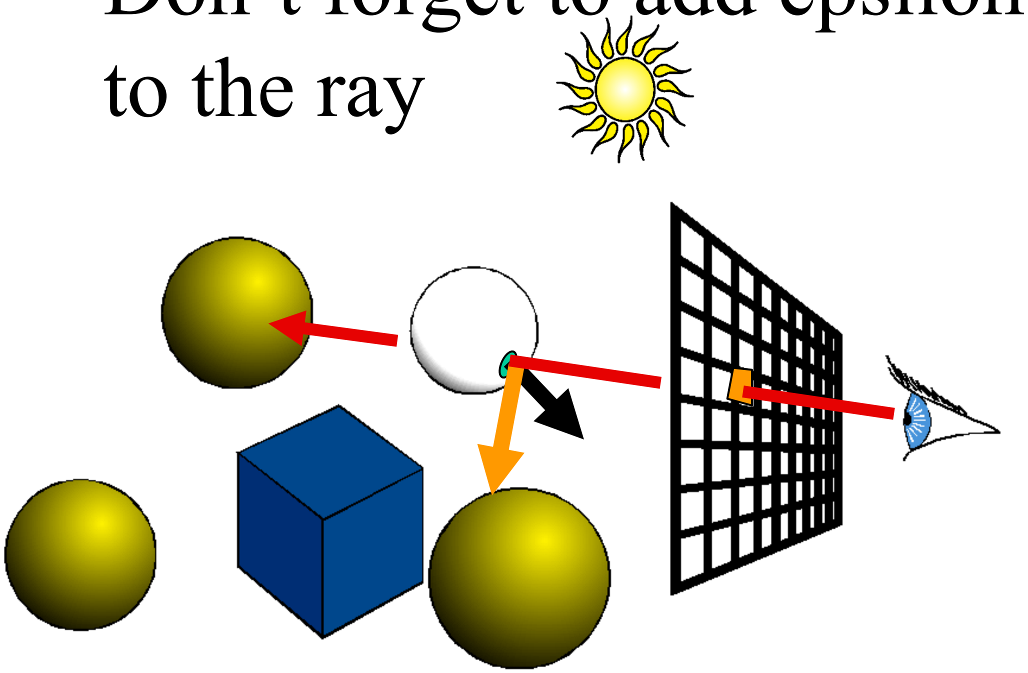
Q: How to calculate reflection

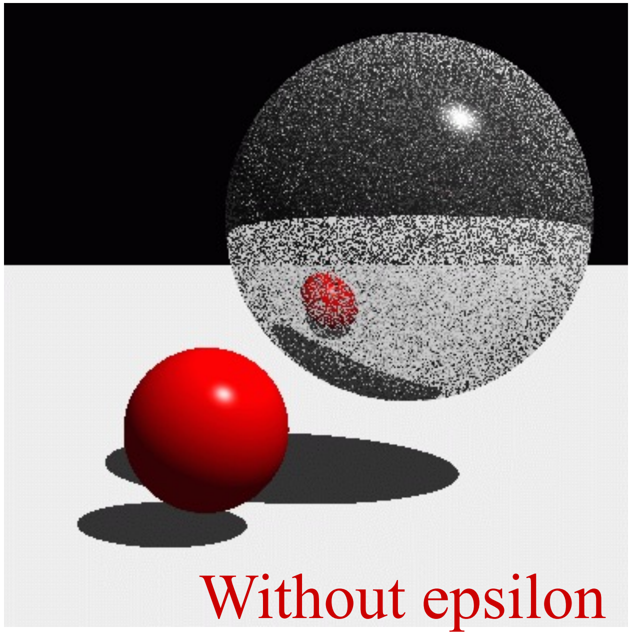
- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions

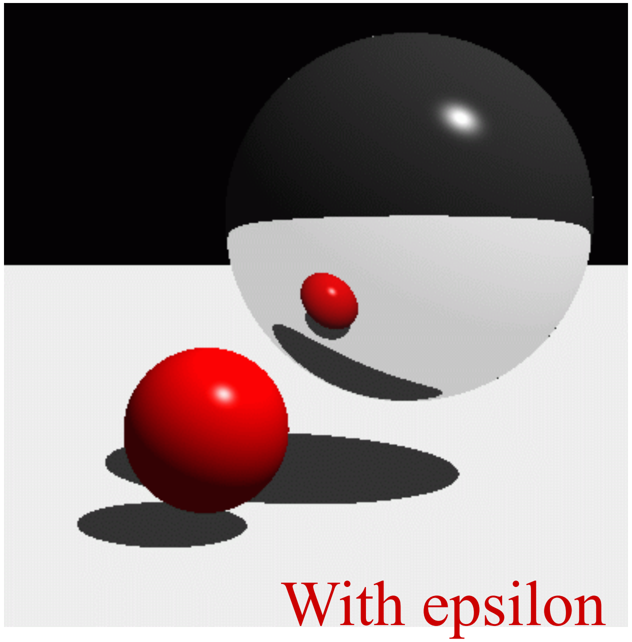

Ray
angle of incidence
θ
normal
θ'
angle of reflection
Reflected Ray
totally reflective surface

Mirror Reflection

- Cast ray symmetric with respect to the normal
- Multiply by reflection coefficient (color)
- Don’t forget to add epsilon to the ray

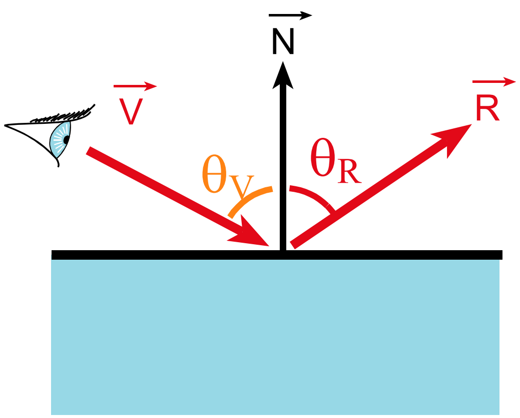


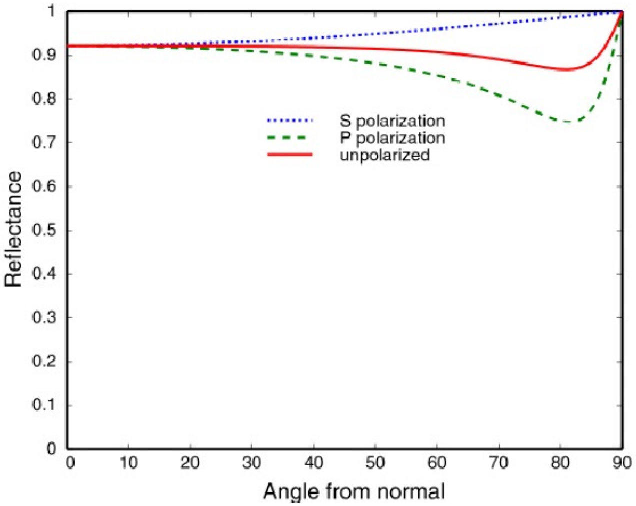




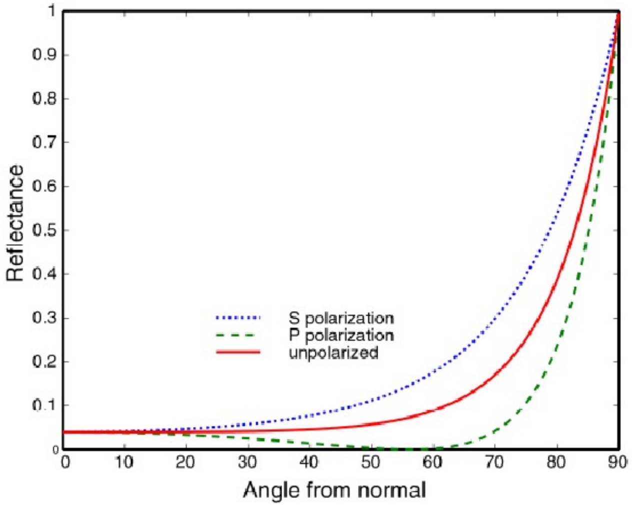
Amount of Reflection

- Traditional ray tracing (hack)
  - Constant **`reflectionColor`**
- More realistic:
  - Fresnel reflection term (more reflection at grazing angle)
  - Schlick's approximation: $R(\theta)=R_0+(1-R_0)(1-\cos\theta)^5$





metal

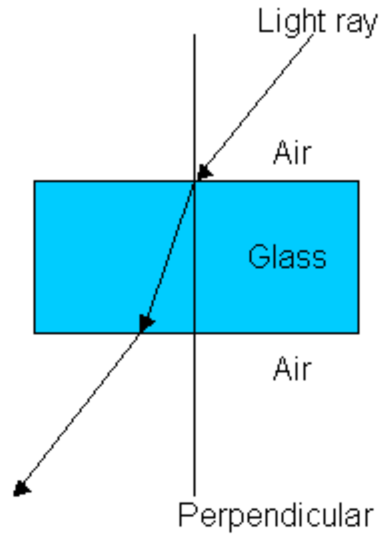


Dielectric (glass)

Add refraction to your code

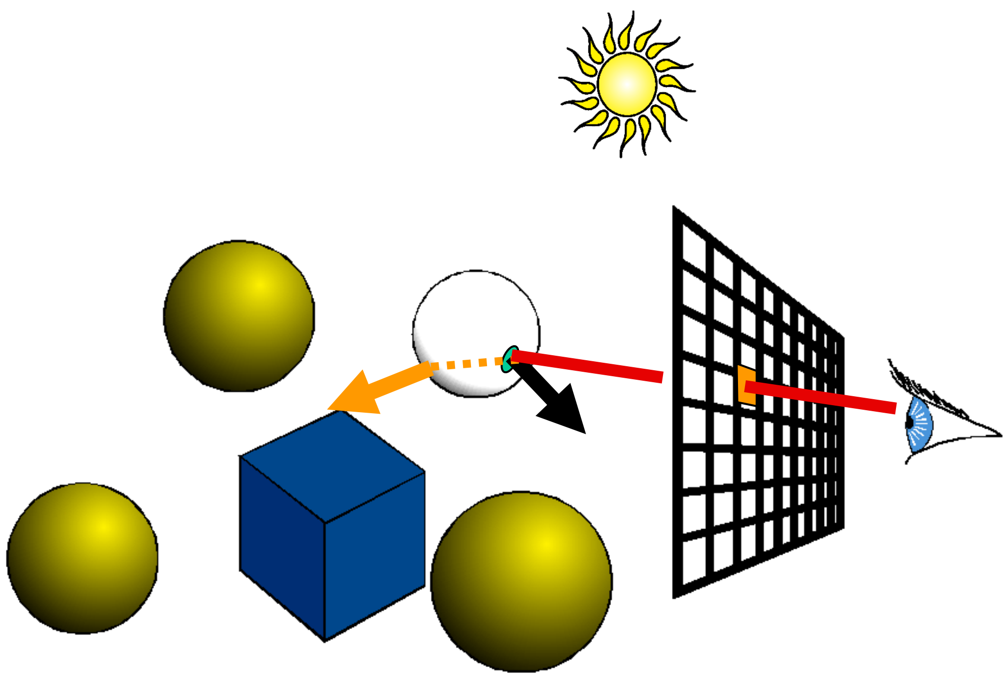
Q: How to calculate refraction

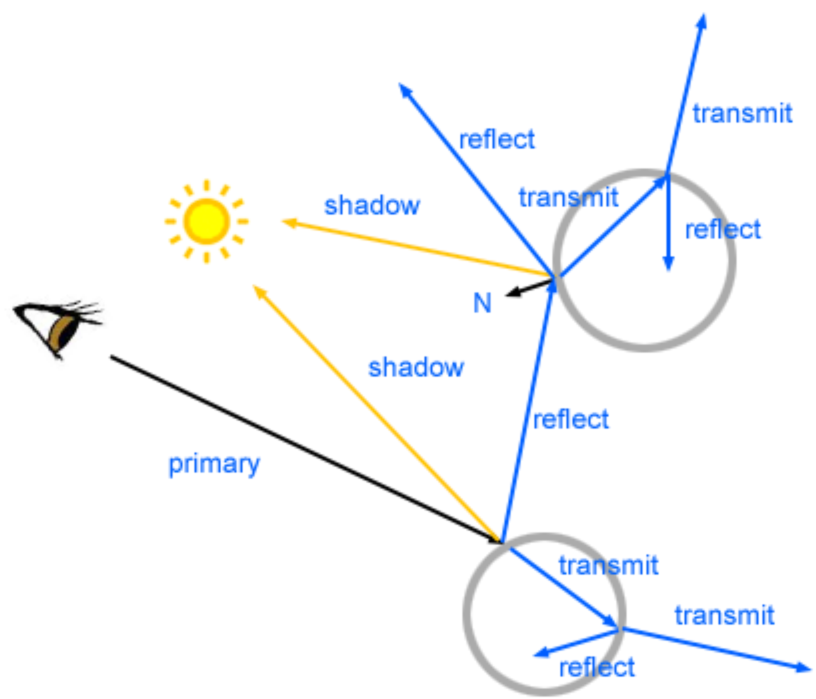
- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions


Light ray
Air
Glass
Air
Perpendicular

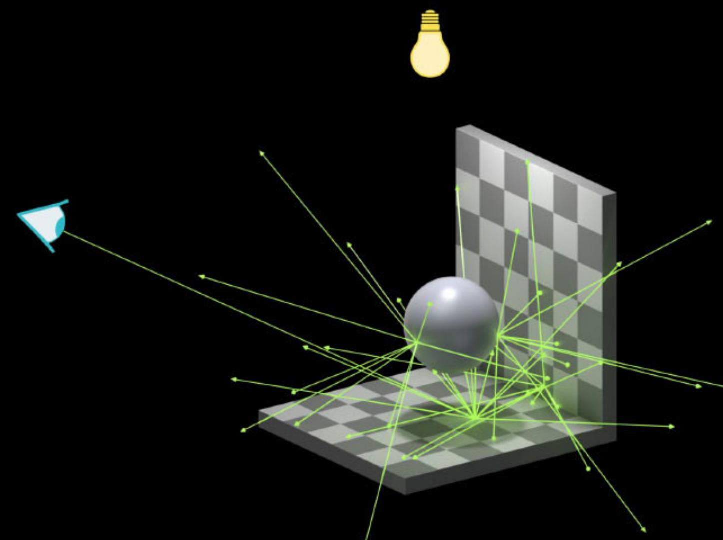
Transparency

- Cast ray in refracted direction
- Multiply by transparency coefficient (color)





reflect
transmit
shadow
transmit
reflect
N
shadow
reflect
primary
transmit
transmit
reflect



Rays Needed	Many
BVH Search	Global
Bounces	Optional

Participation May 26

Form description

This form is automatically collecting email addresses for UC Santa Cruz users. Change settings

I was in class May 26

- Yes
- No

I created an extra credit assignment to force you to do SETs. Was that a good idea?

- Yes. I always do SETs
- Yes. I skip unless its required like this.
- No. I dont like to be forced.
- No. Sets are a waste of time.
- Other...

My primary reason to be in college is:

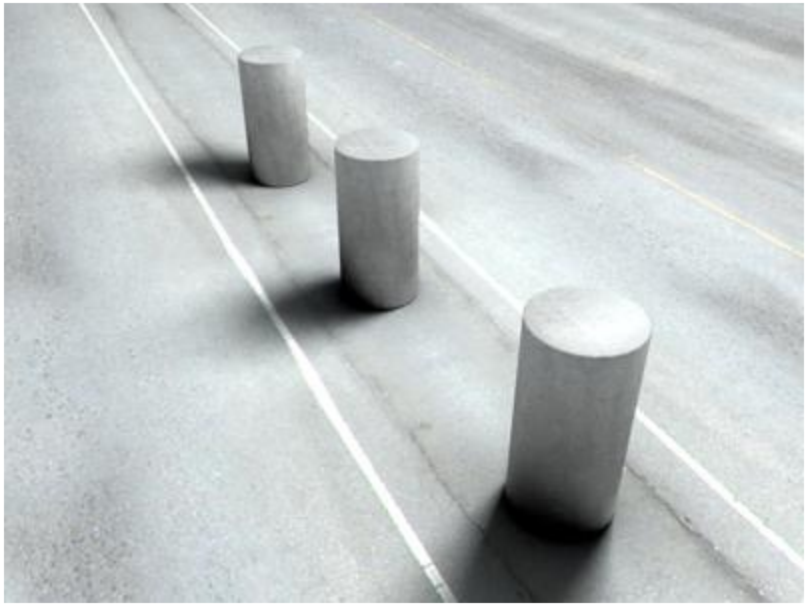
- To be educated because its fun
- To be educated because its a civic responsibility
- Better job, specific job requirement, higher salary, etc
- Other...

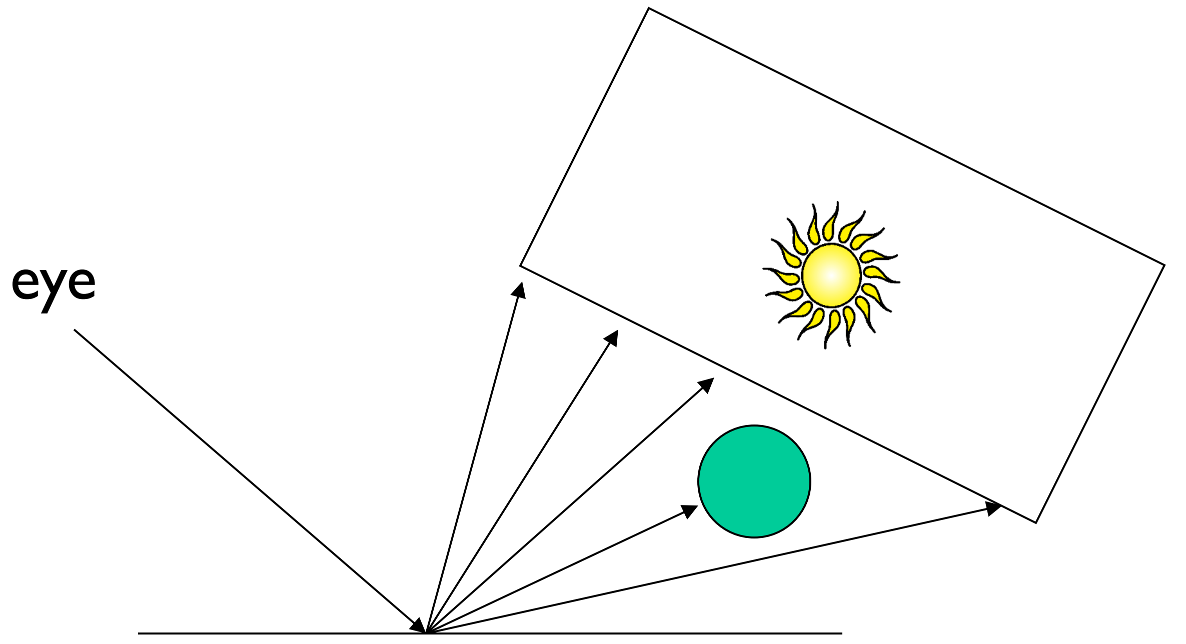
I want to go to grad school:

- Heck no! I am done with school
- Never thought about it

Distributed raytracing

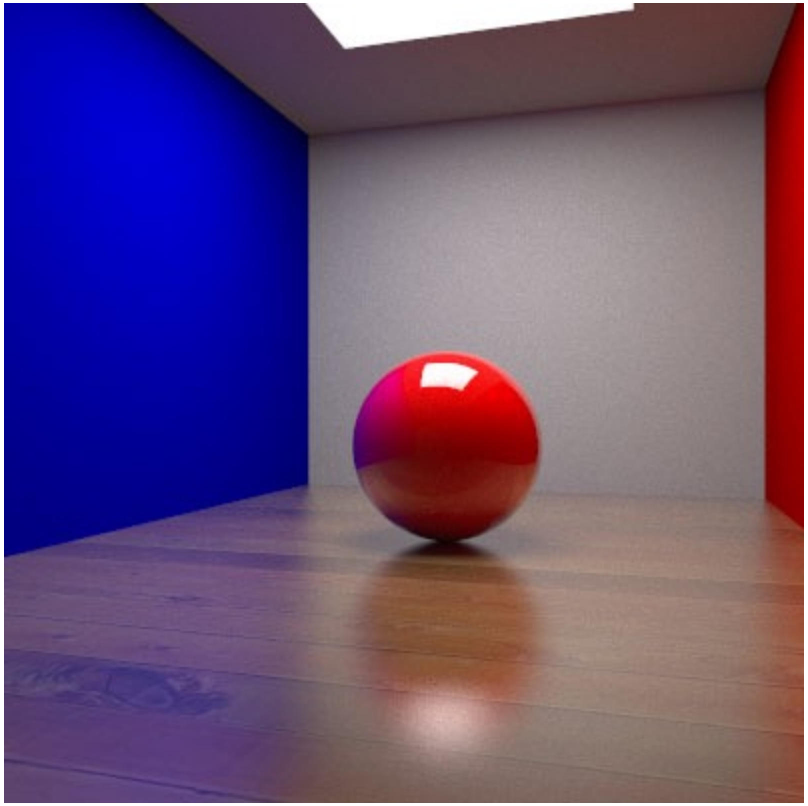
Soft shadows?





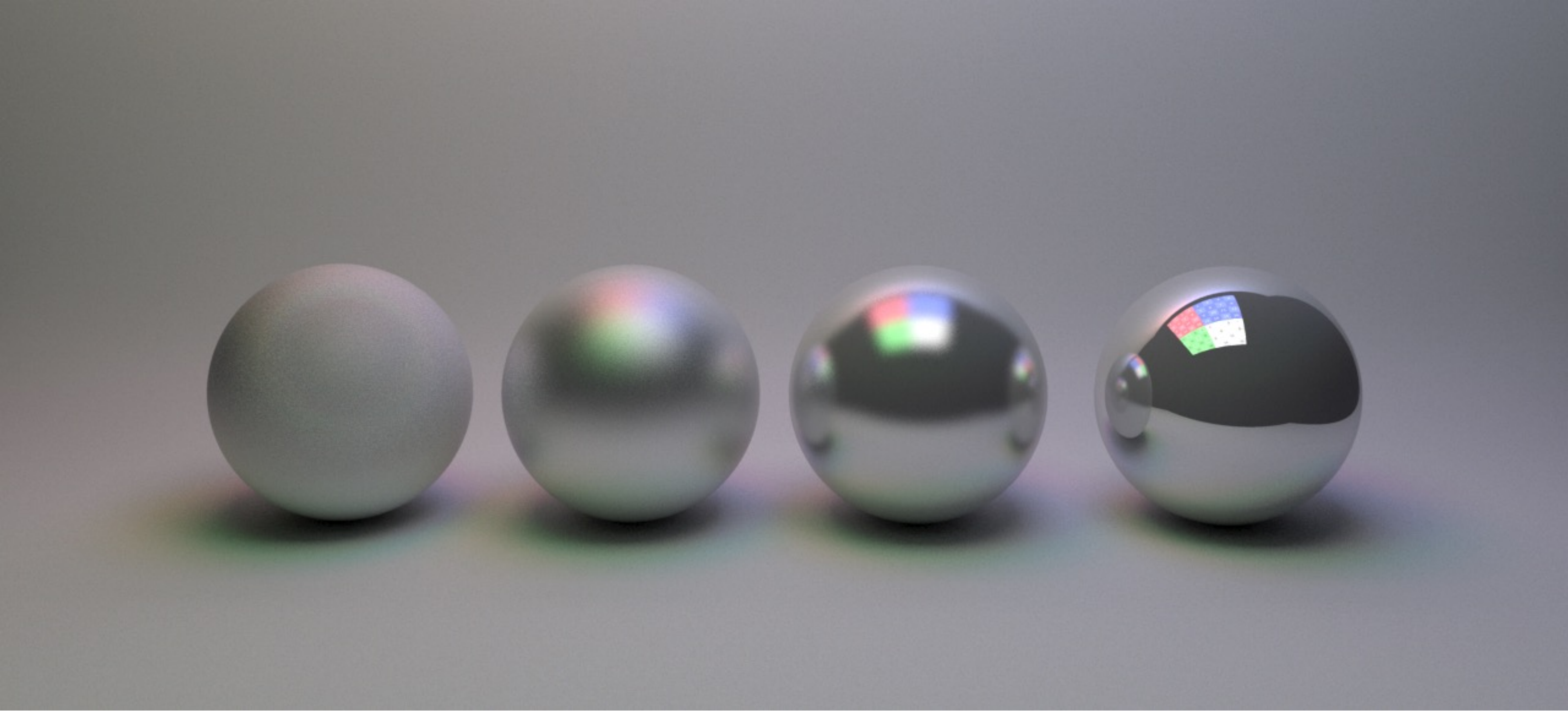
20% in shadow

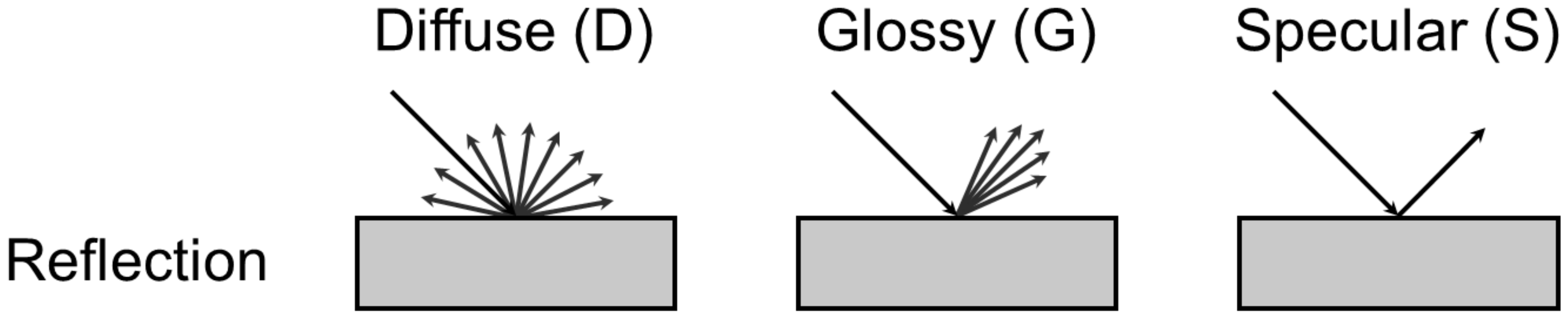
Glossy reflections?



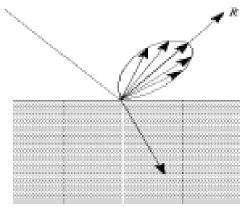
Q: How to calculate glossy surface

- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions

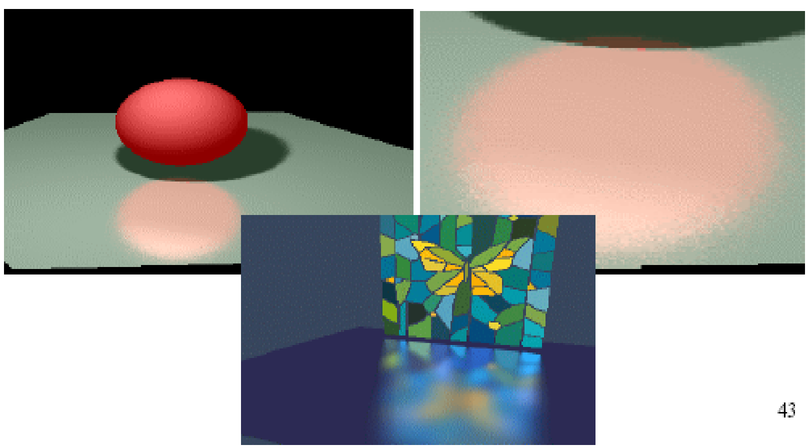



Diffuse (D)
Glossy (G)
Specular (S)
Reflection

Distributing Reflections

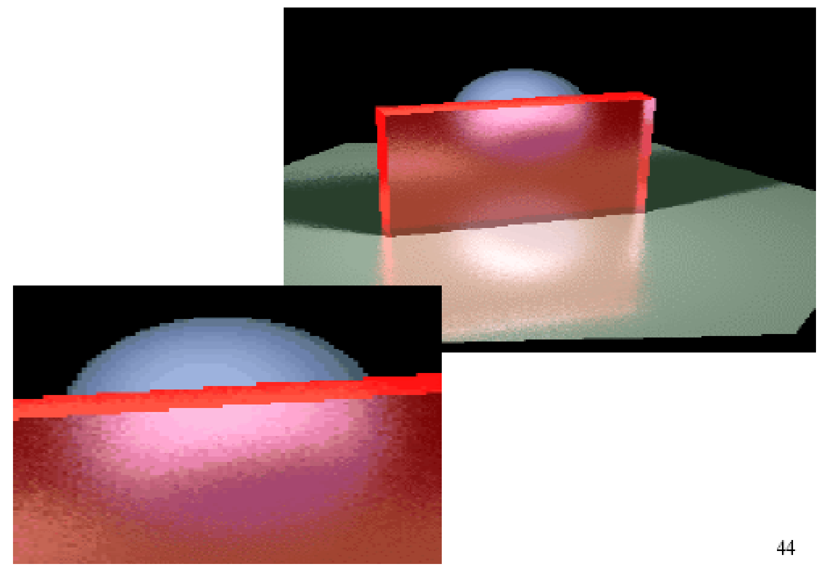


- Distributing rays over reflection direction gives:



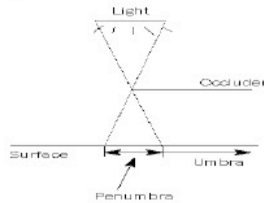
Distributing Refractions

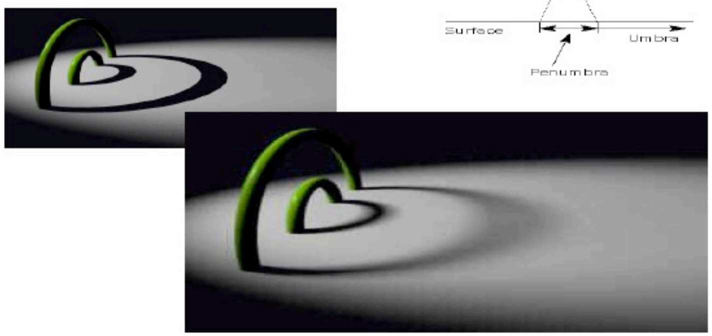
- Distributing rays over transmission direction gives:



Distributing Over Light Area

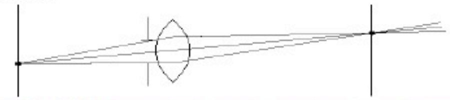
- Distributing over light area gives:

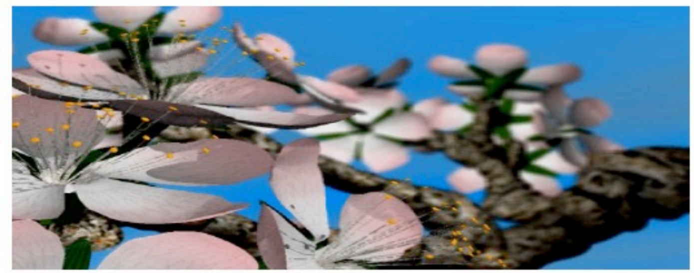




Distributing Over Aperature

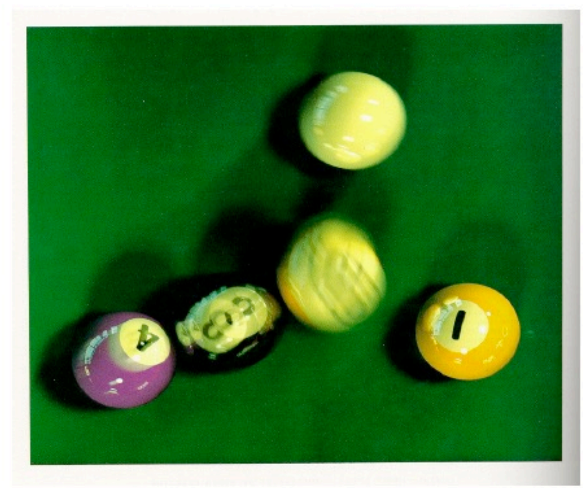
- We can fake distribution through a lens by choosing a point on a finite aperature and tracing through the "in-focus point".



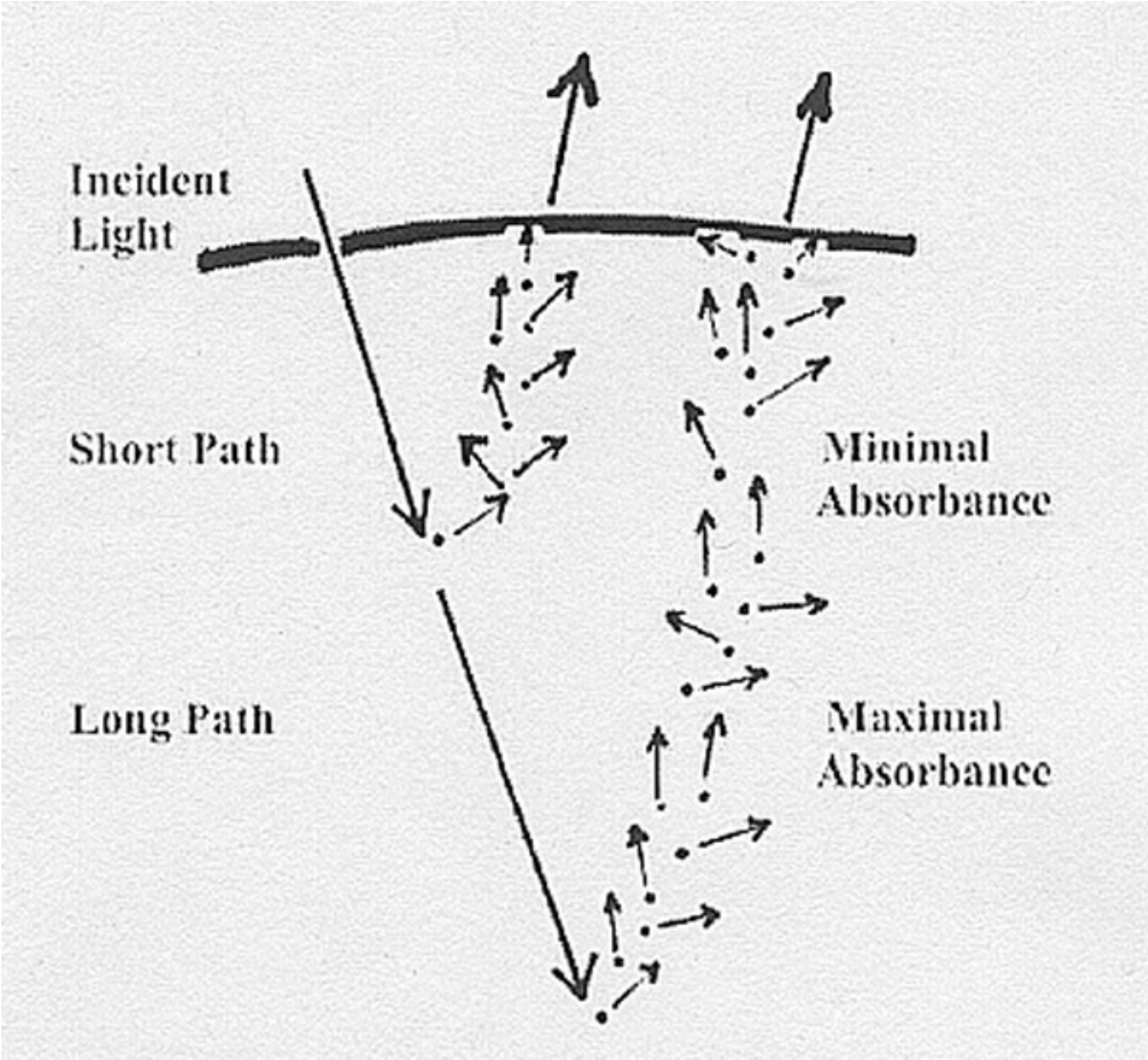


Distributing Over Time

- We can endow models with velocity vectors and distribute rays over *time*. this gives:



Subsurface scattering

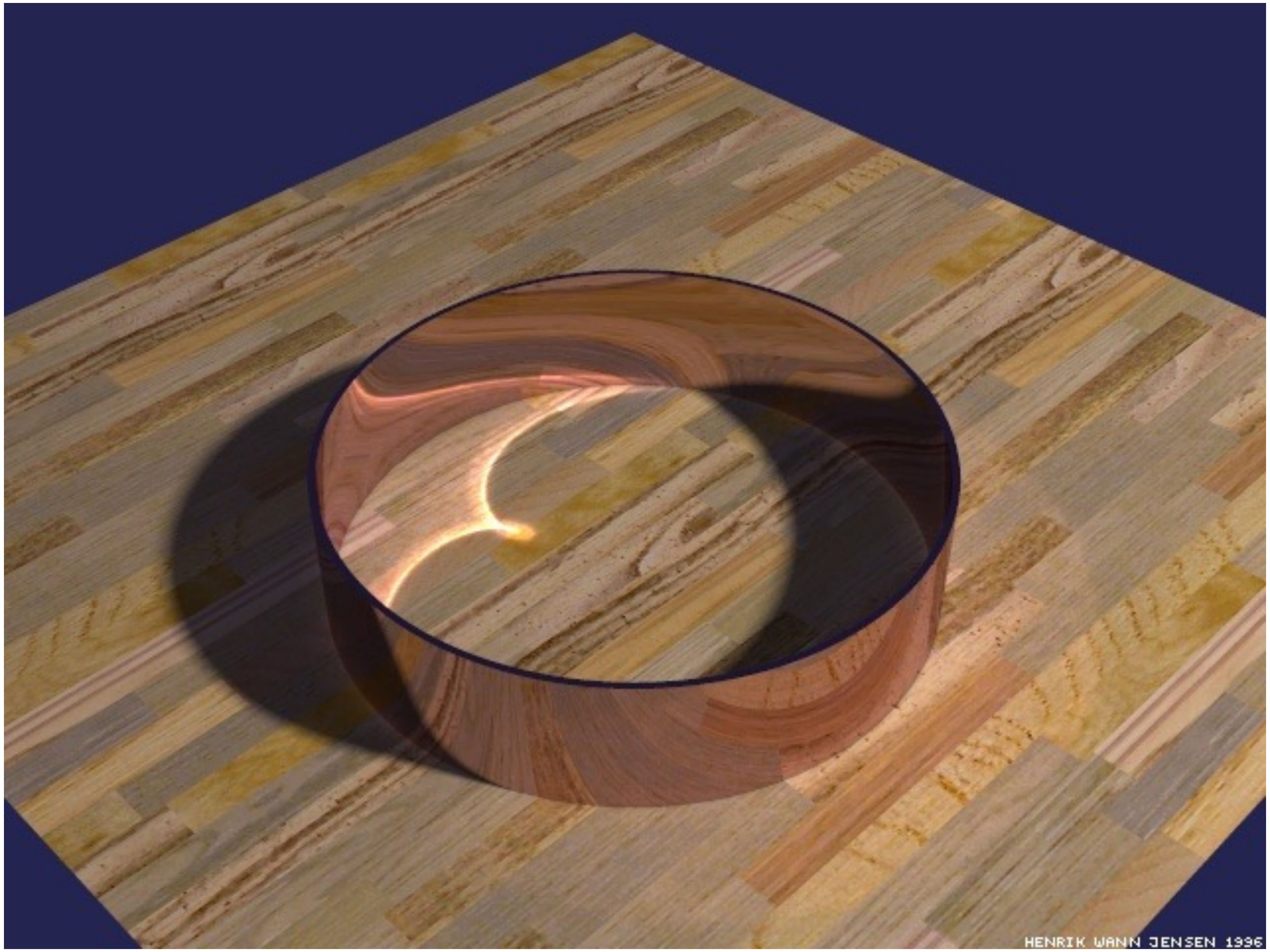



RENDERED USING DALI - HENRIK WANN JENSEN 2001


HENRIK WANN JENSEN 2000
HENRIK WANN JENSEN 2001

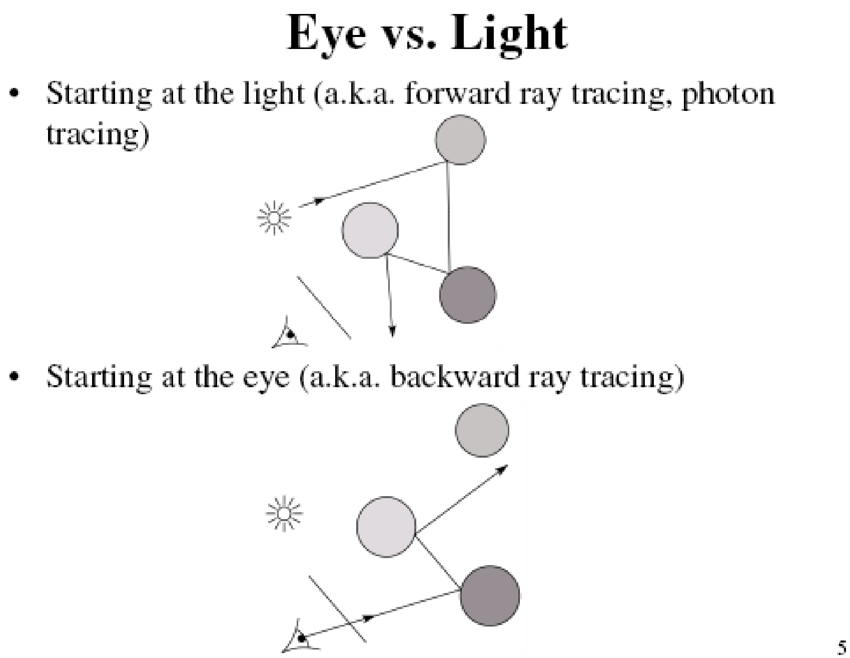
Photon Mapping

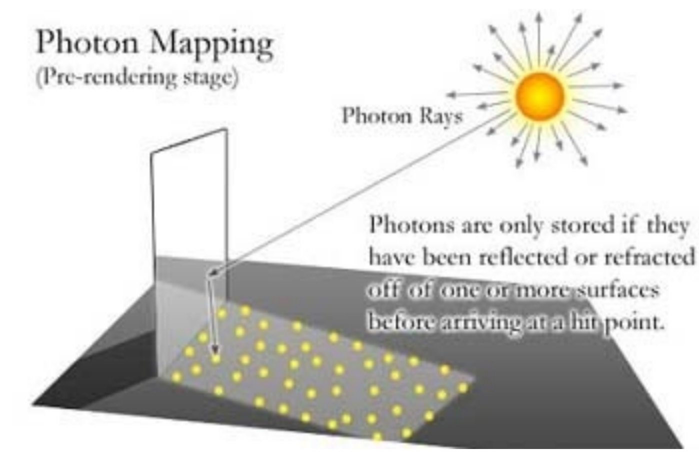
How do we get this effect? (caustics)

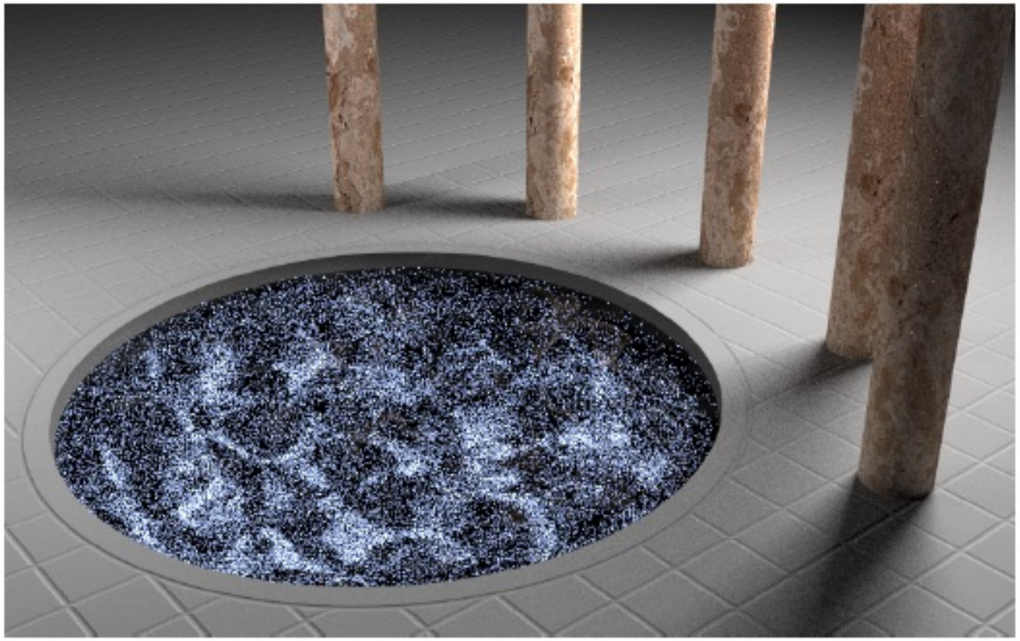



HENRIK WANN JENSEN 1995

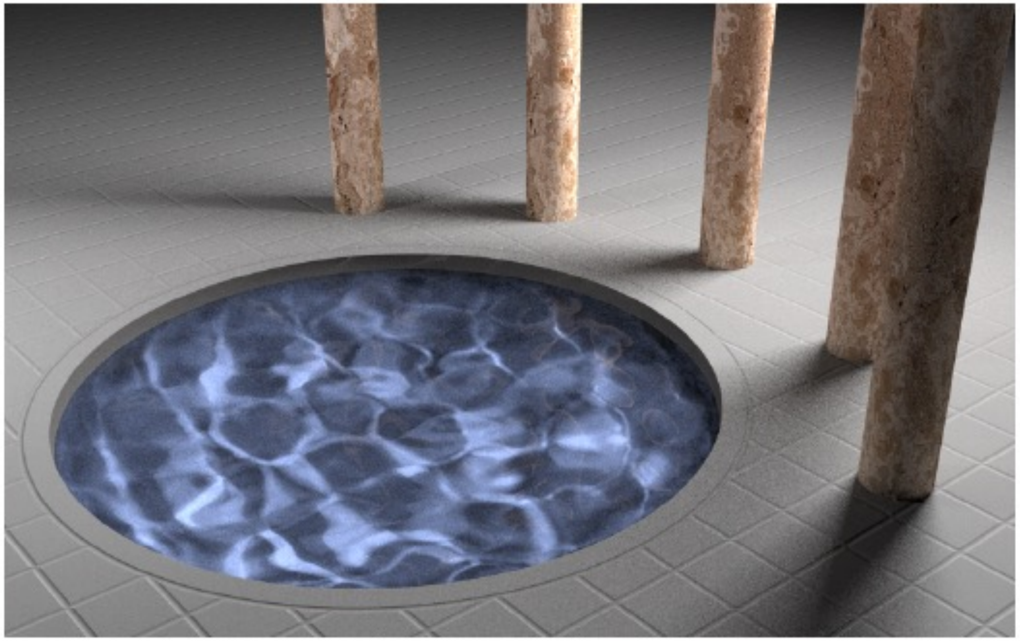
Do both directions. Deposit light in the scene from the light.







(a) Path tracing with 210 samples per pixel.



(b) Metropolis light transport with an average of 100 mutations per pixel [the same computation time as (a)].

Real time raytracing

(I have no idea how this works)

NVIDIA

GEFORCE PRODUCTS GEFORCE EXPERIENCE DRIVERS GAMES NEWS

GEFORCE RTX™ SHOP ALL

RTX. IT'S ON.

RAY TRACING IS HERE

Experience today's biggest blockbusters like never before with the visual fidelity of real-time ray tracing and the ultimate performance of AI-powered DLSS 2.0. RTX. It's On.

NVIDIA

GEFORCE PRODUCTS GEFORCE EXPERIENCE DRIVERS GAMES NEWS

GEFORCE RTX™ SHOP ALL

SHOP NOW

GEFORCE RTX 2080 Ti
$1,199.00
NOTIFY ME

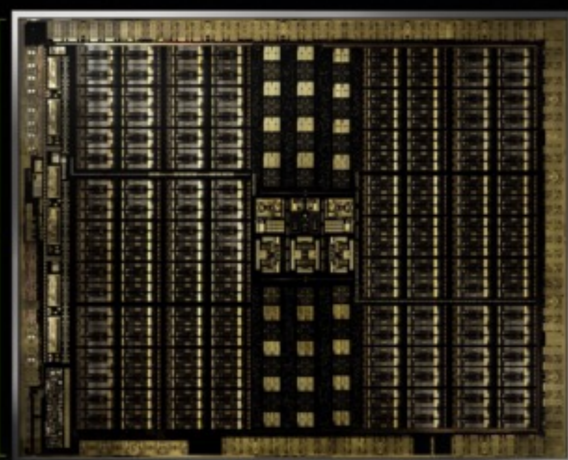
GEFORCE RTX 2080 SUPER
$699.00
SHOP ALL

GEFORCE RTX 2070 SUPER
$499.00
SHOP ALL

GEFORCE RTX 2060 SUPER
$399.00
SHOP ALL

TURING BUILT FOR RTX

GREATEST LEAP SINCE 2006 CUDA GPU



Turing SM
14 TFLOPS + 14 TIPS
Concurrent FP & INT Execution
Variable Rate Shading

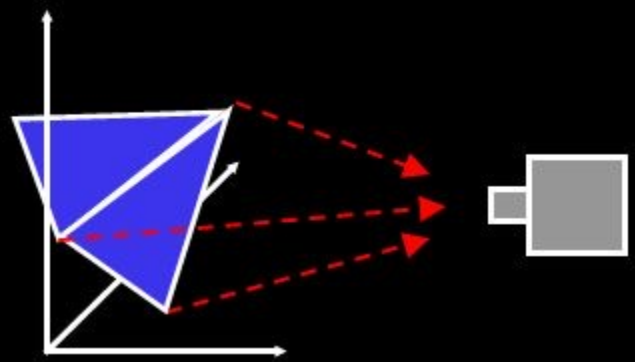
Tensor Core
110 TFLOPS FP16
220 TOPS INT8
440 TOPS INT4

RT Core
10 Giga Rays/sec
Ray Triangle Intersection
BVH Traversal

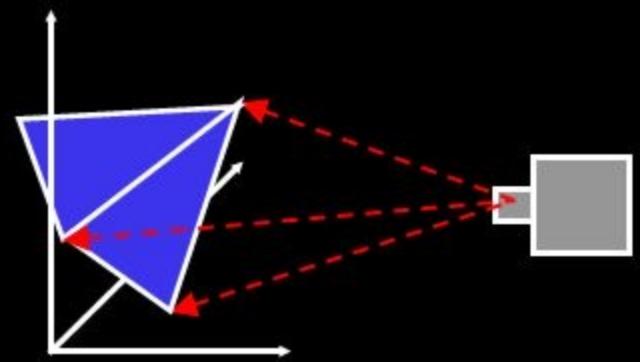
Introduction to Realtime Ray Tracing

SIGGRAPH2005

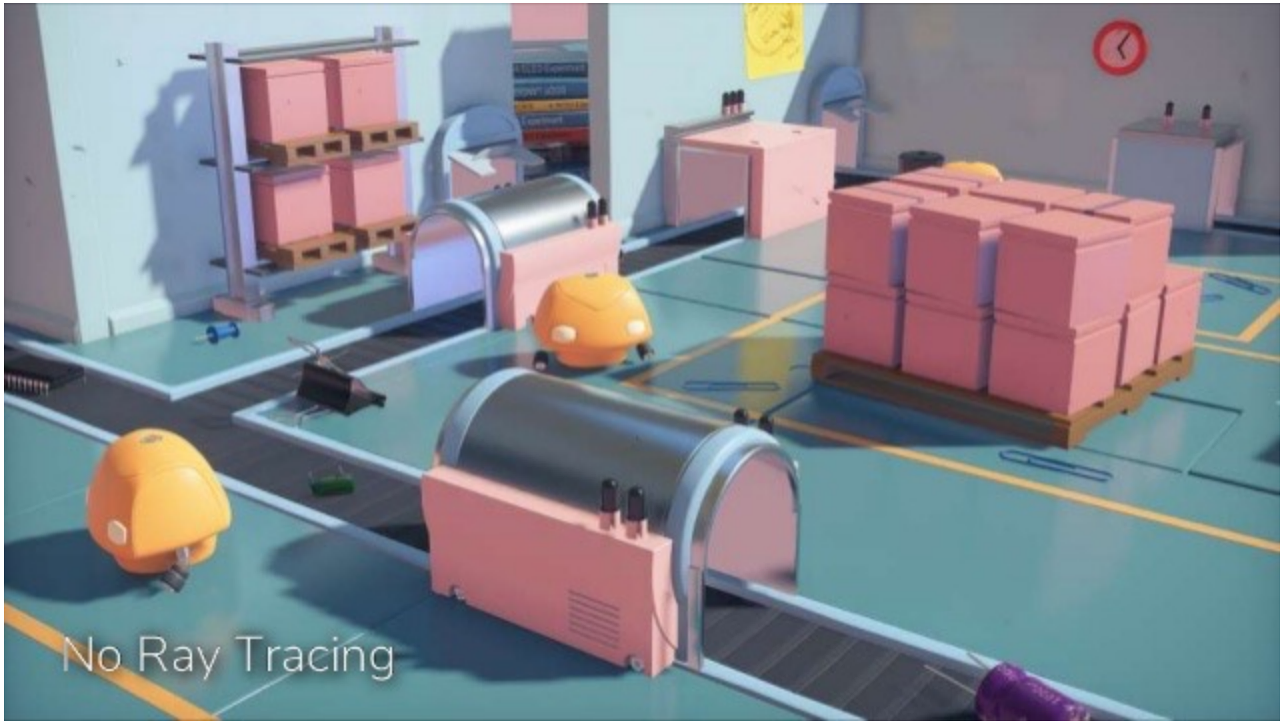
Rendering in Computer Graphics

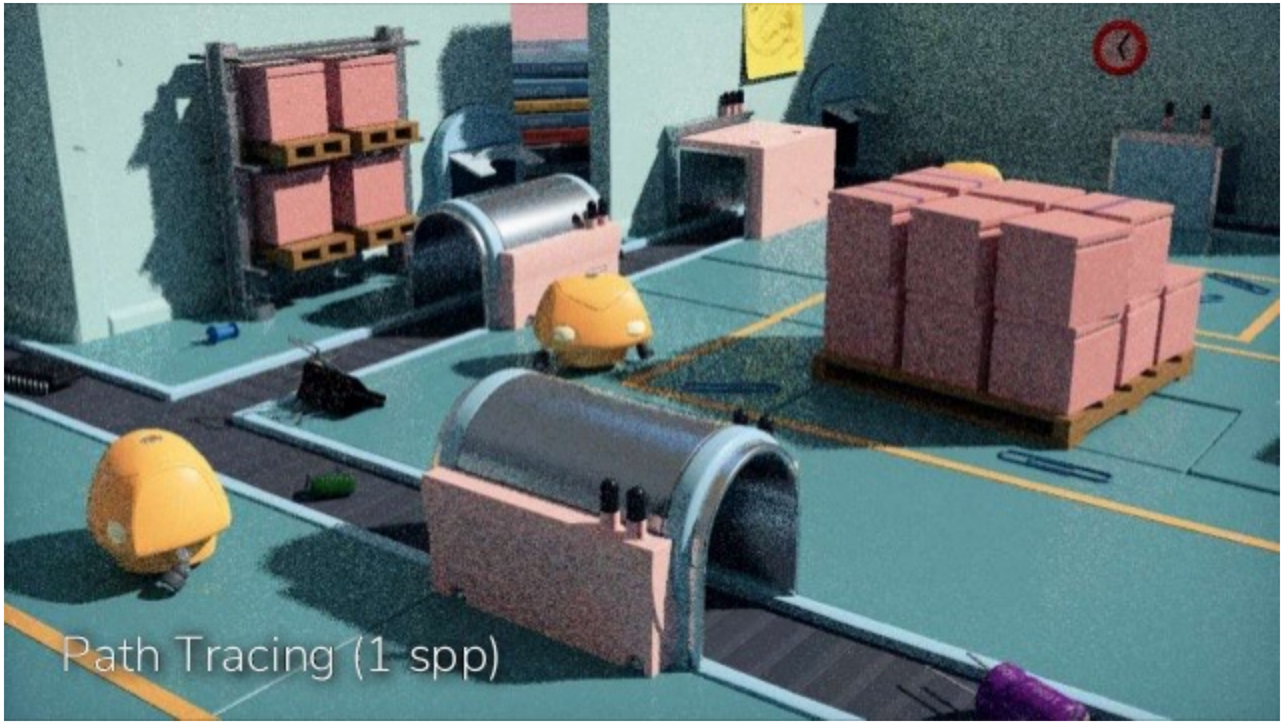


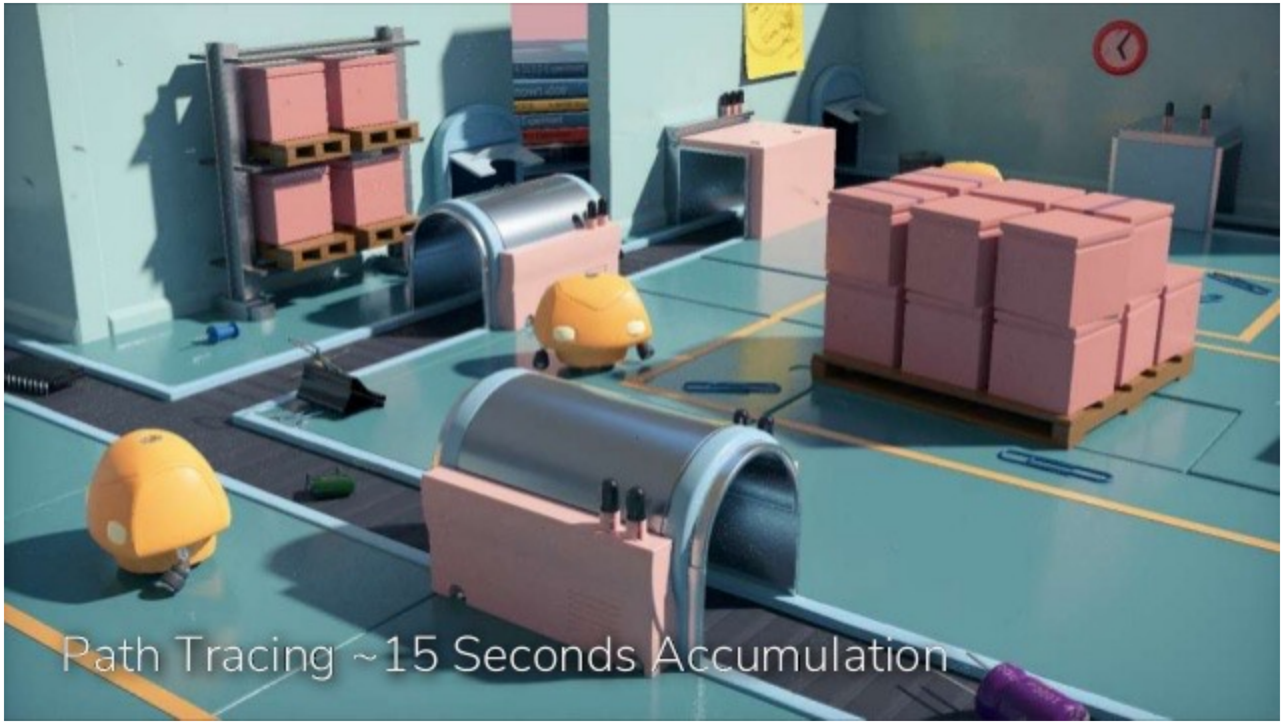
Rasterization:
Projection geometry forward



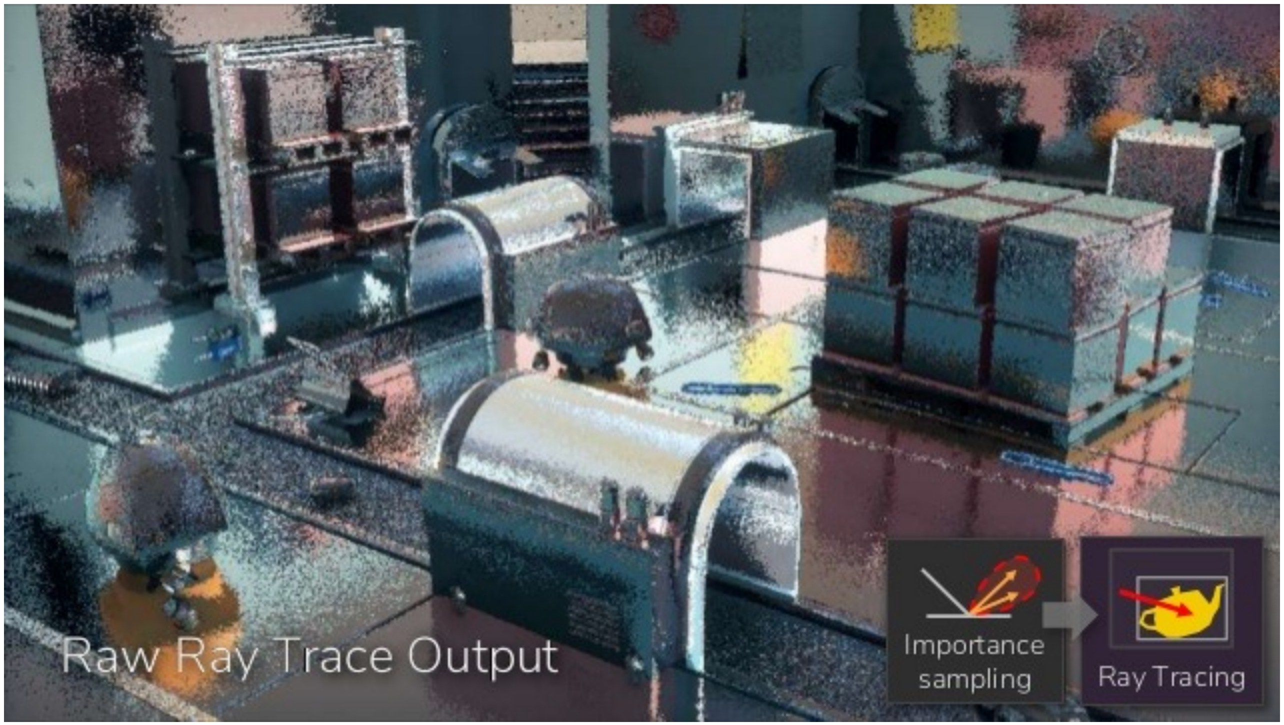
Ray Tracing:
Project image samples backwards


No Ray Tracing


Path Tracing (1 spp)


Path Tracing ~15 Seconds Accumulation


Real-Time Hybrid Ray Tracing


Raw Ray Trace Output
Importance sampling
Ray Tracing

Administrative

Q&A

End