

Ray Tracing- CSE160 – Dec 1

- 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

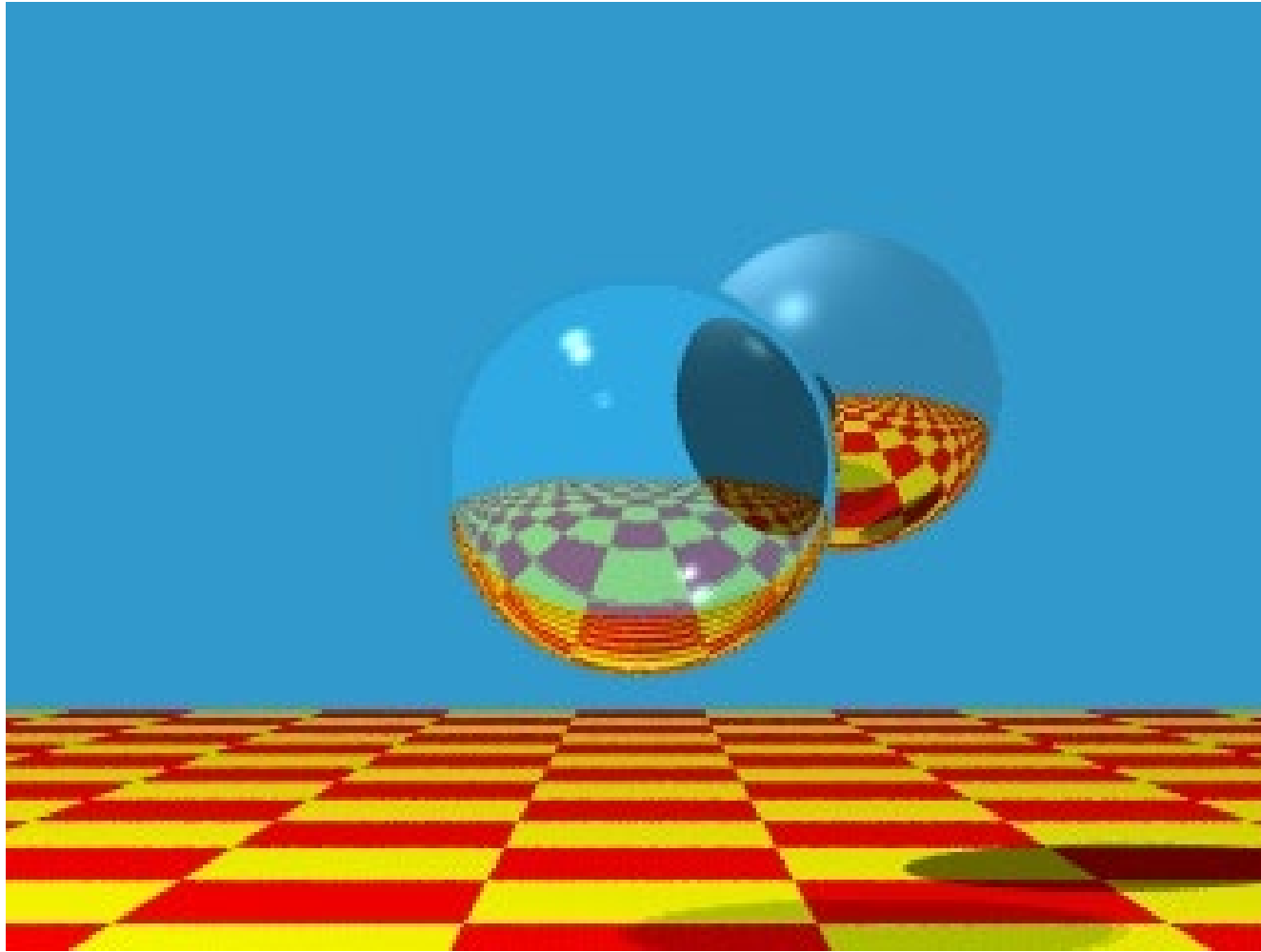
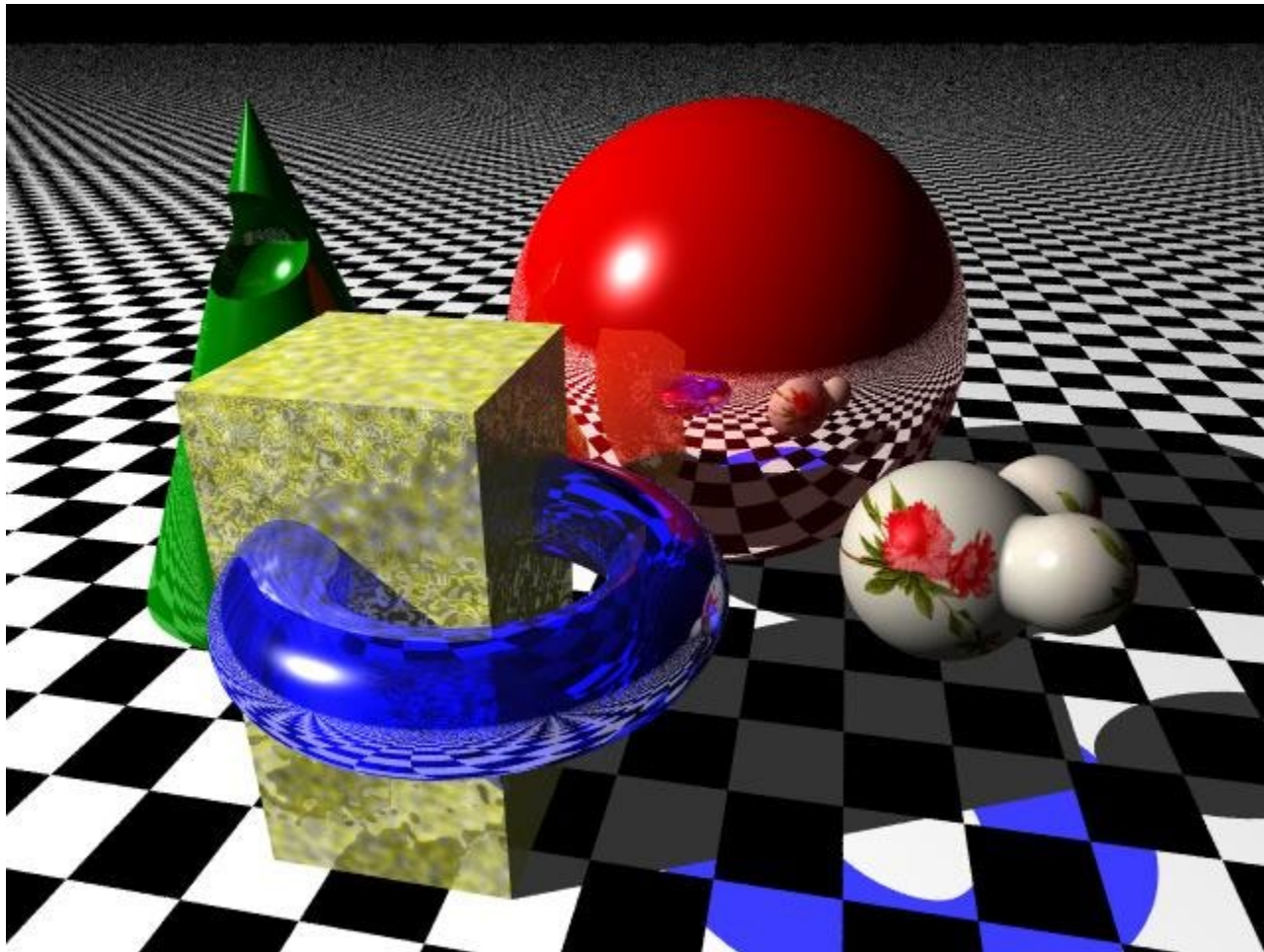
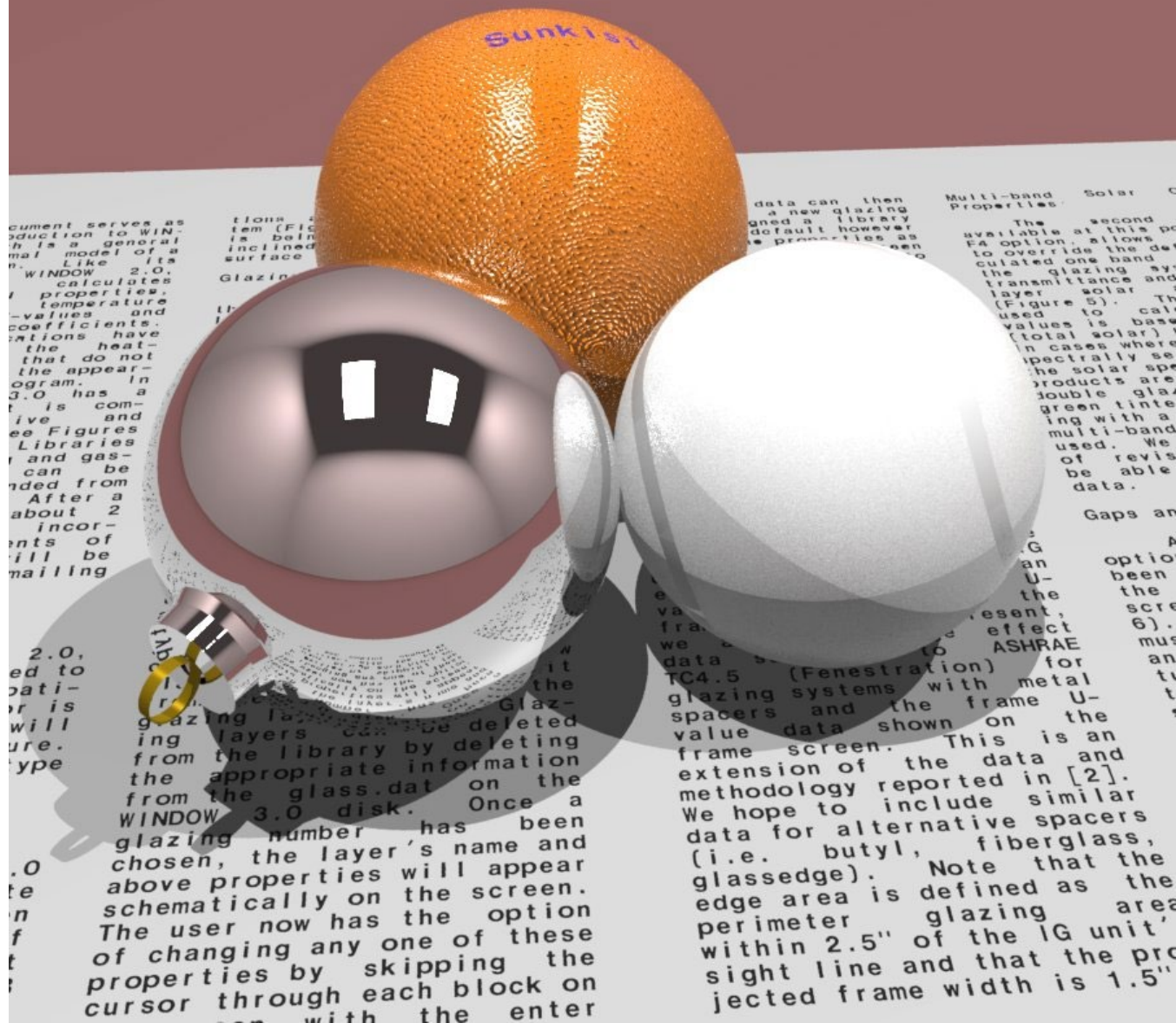


Image by
Turner
Whitted







ON



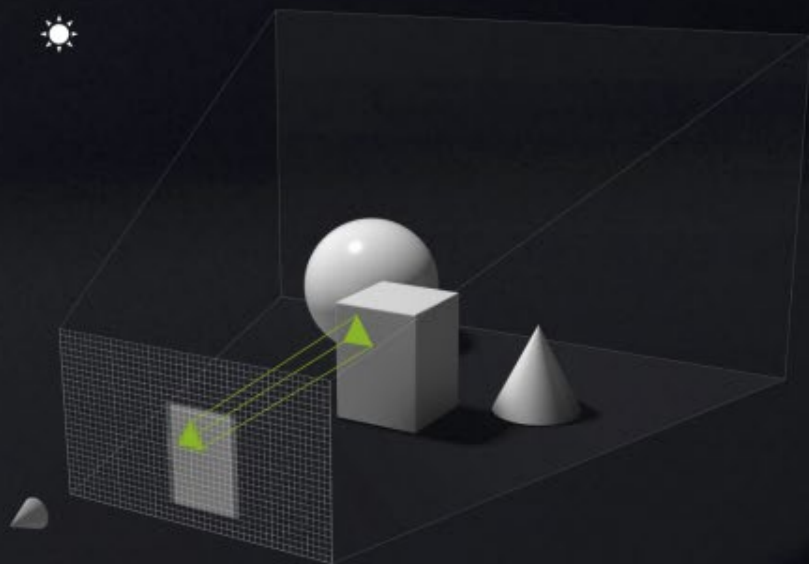
OFF



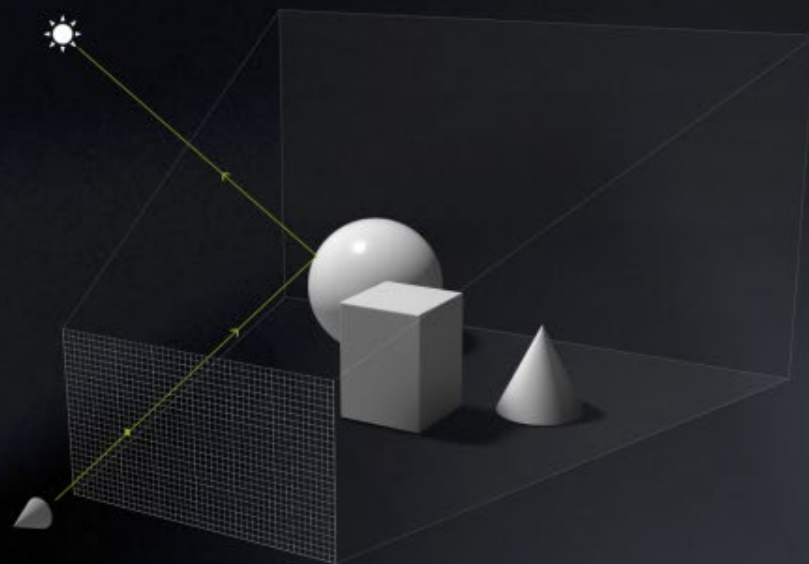
RAY TRACING TECHNIQUES



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]
```

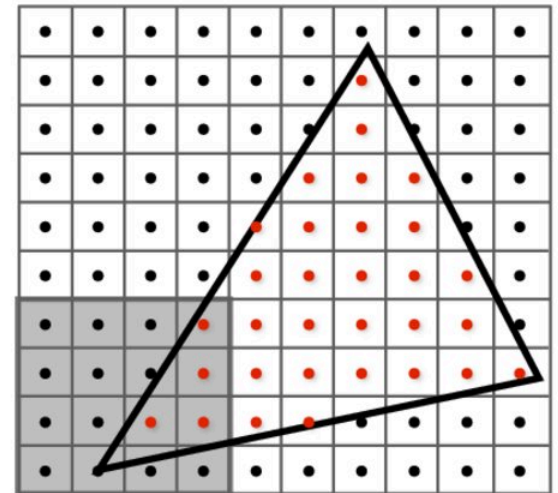
"Given a triangle, find the samples it covers"

(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)

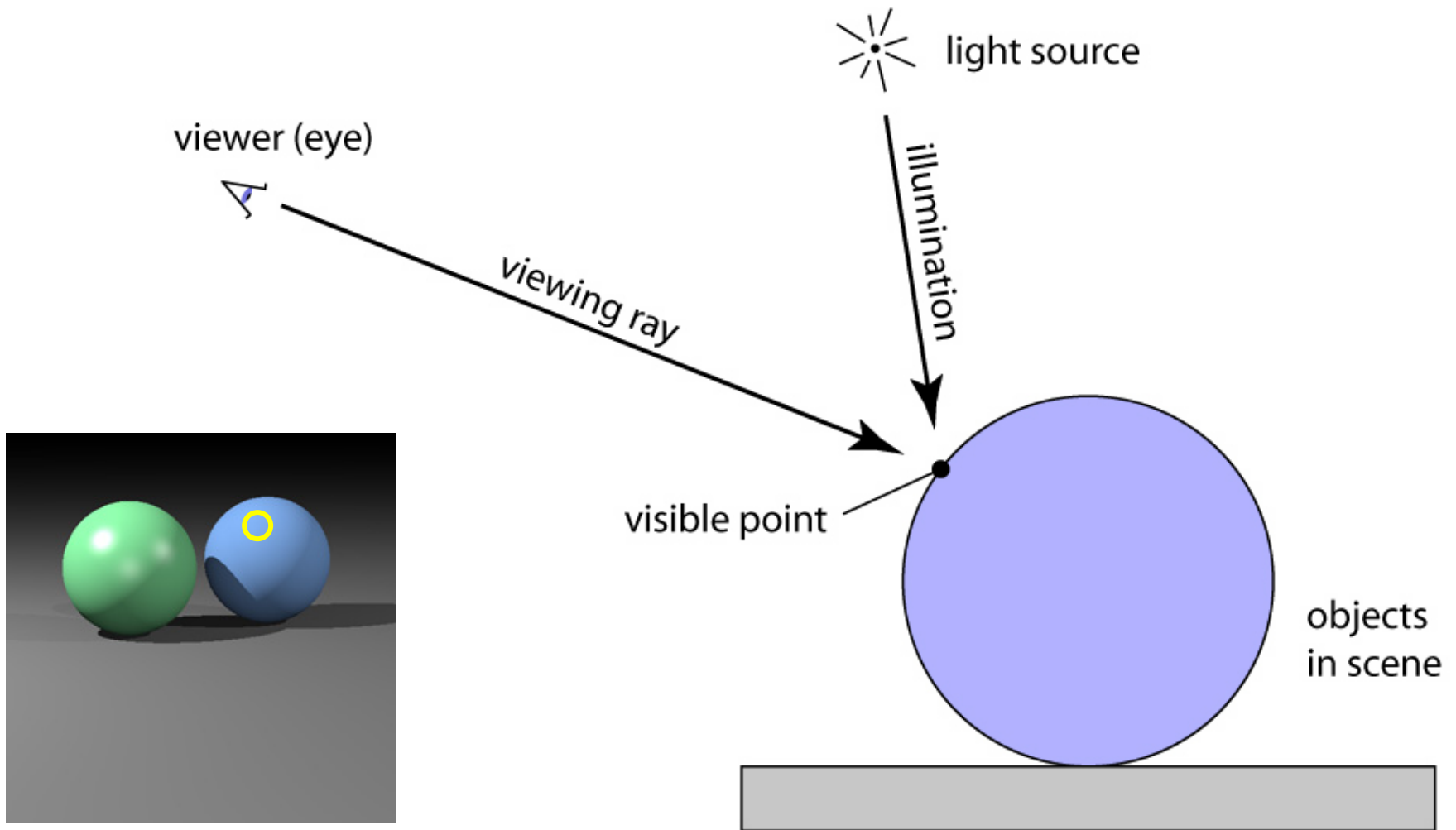
Occlusion: closest intersection along ray

```
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
```

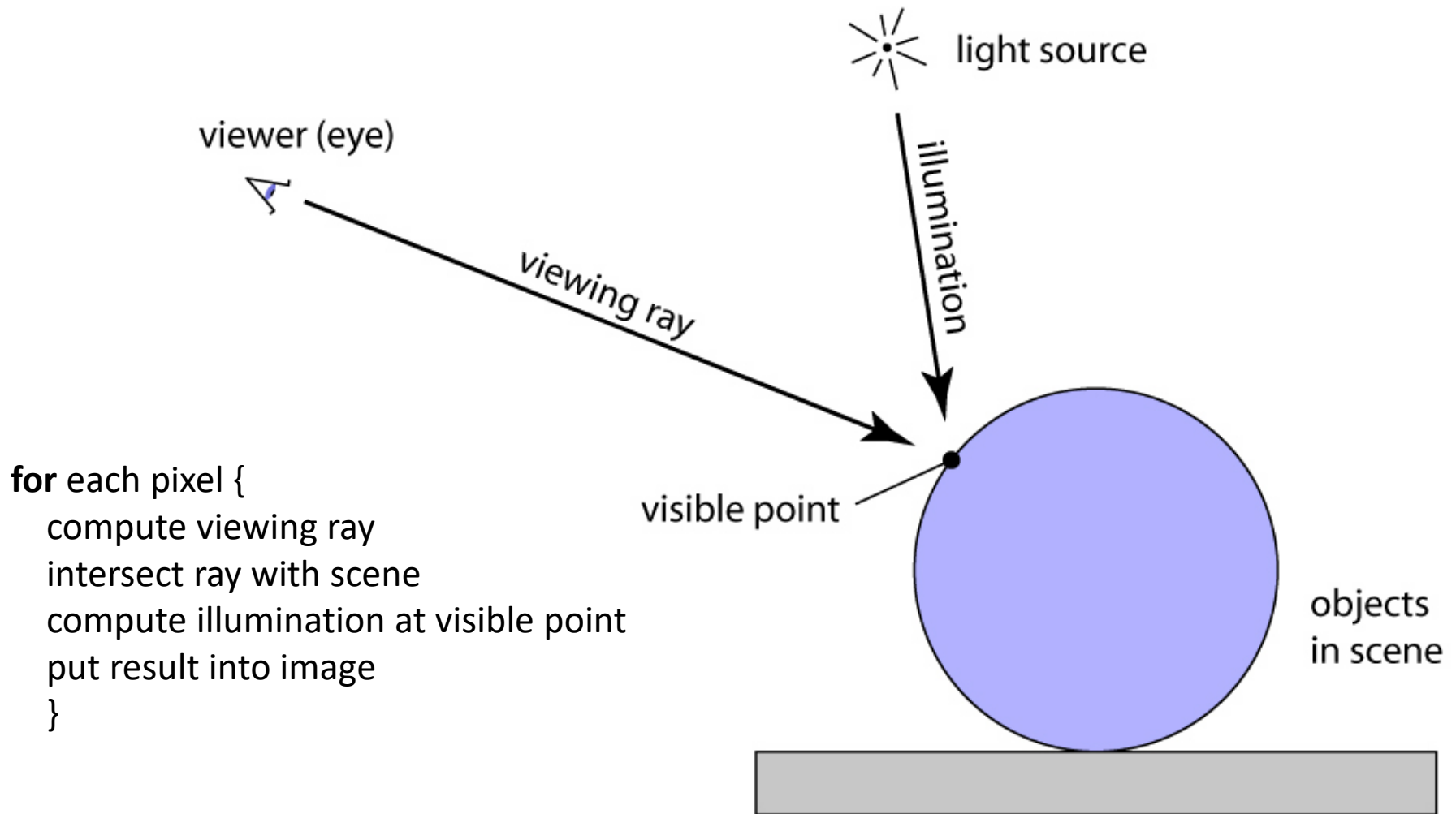
Compared to rasterization approach: just a reordering of the loops!

“Given a ray, find the closest triangle it hits.”

Ray tracing idea

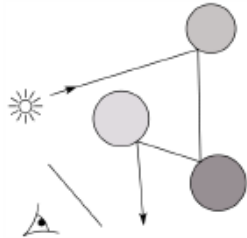


Ray tracing algorithm

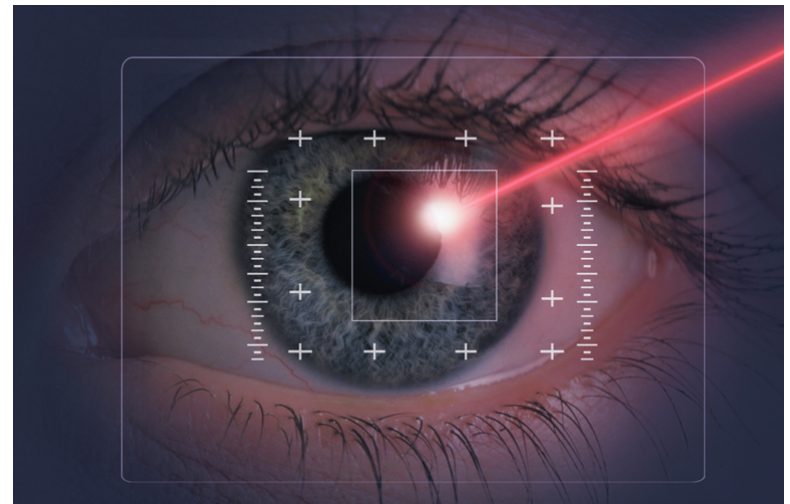
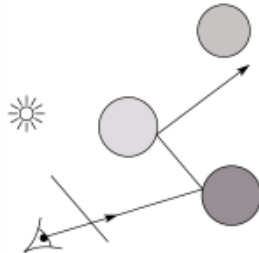


Eye vs. Light

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

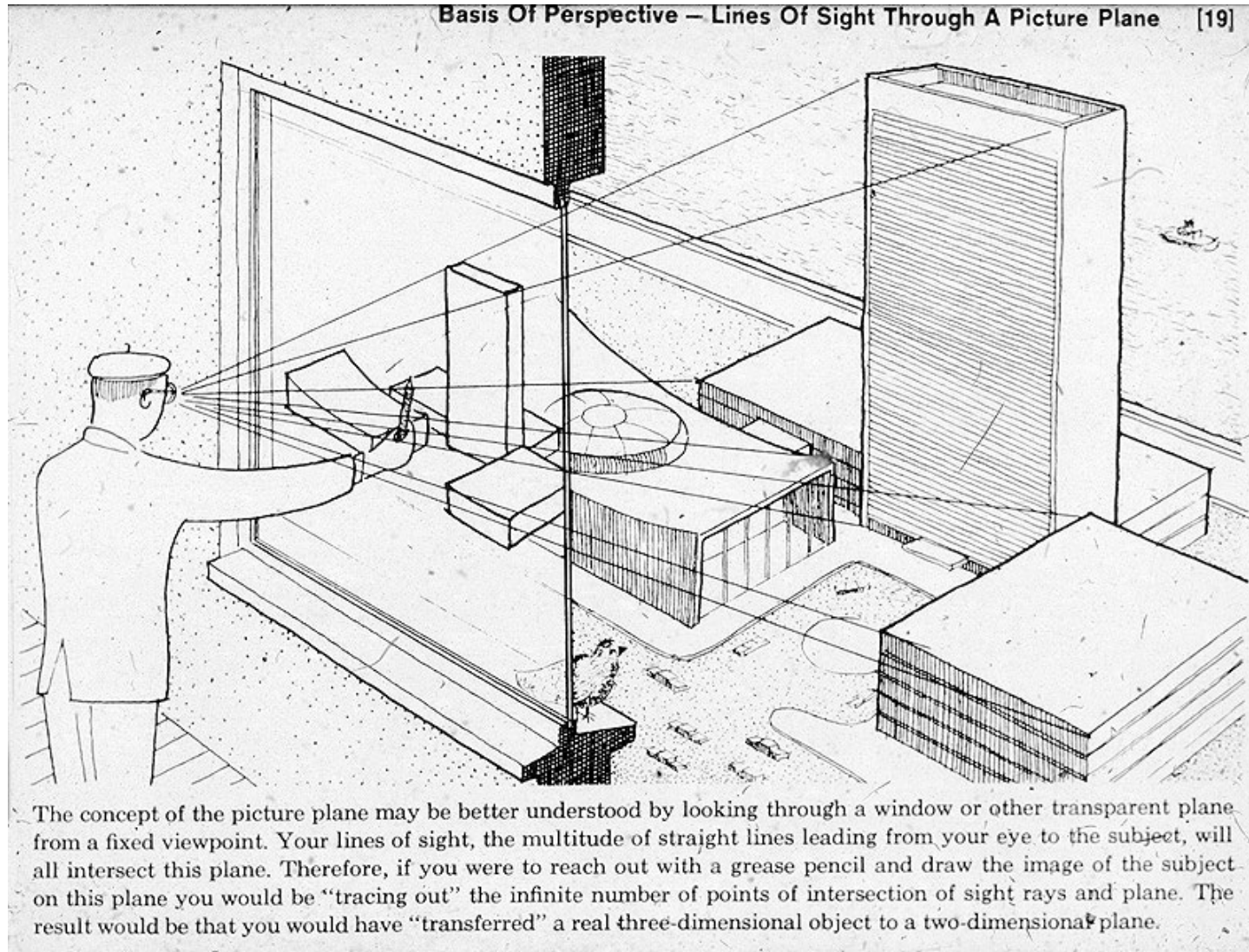


- Starting at the eye (a.k.a. backward ray tracing)



Rays

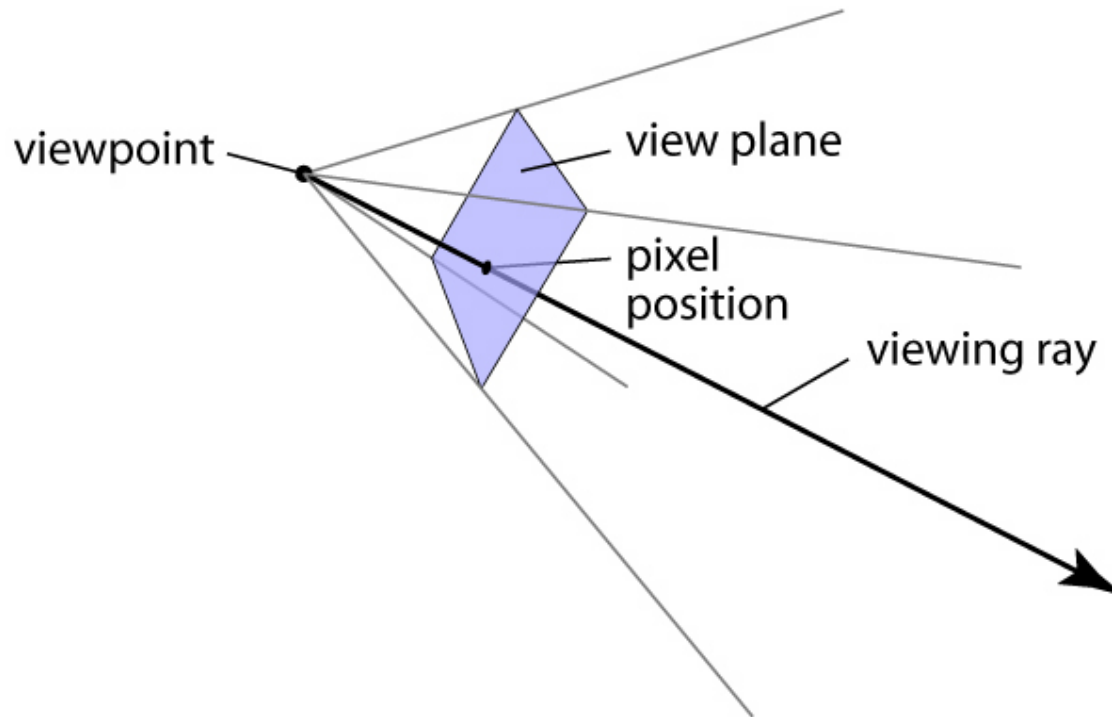
Analogy to drawing



[CS 417 Spring 2002]

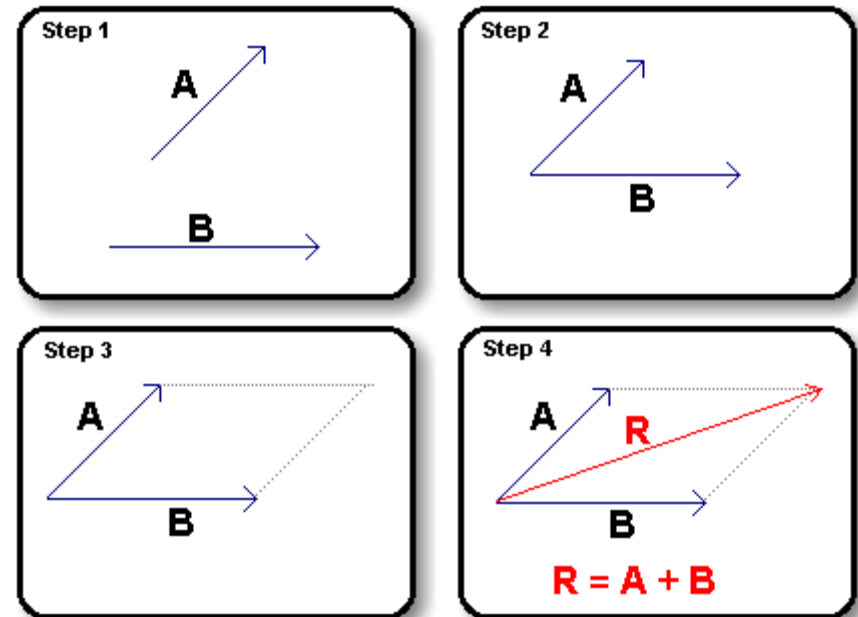
Generating eye rays

- Use window analogy directly

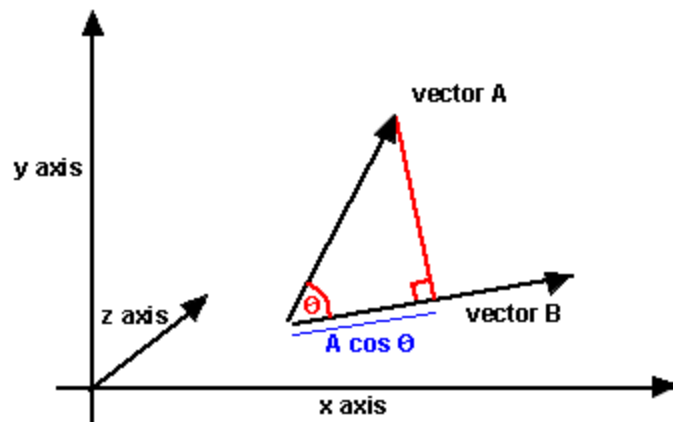


Vector math review

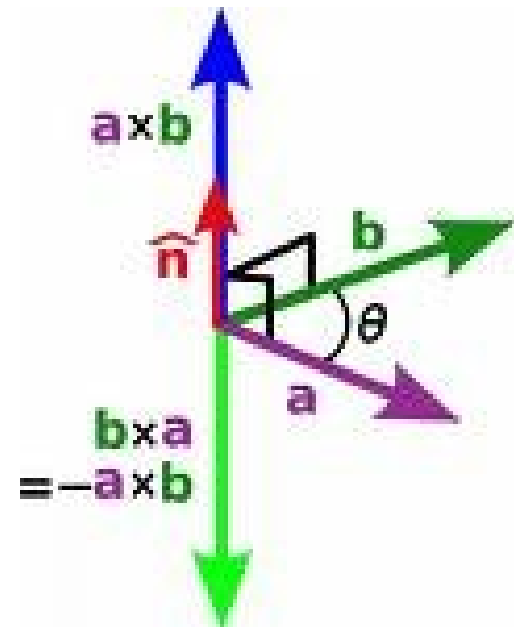
- Vectors and points
- Vector operations
 - addition
 - scalar product
- More products
 - dot product
 - cross product



Dot product



Cross product

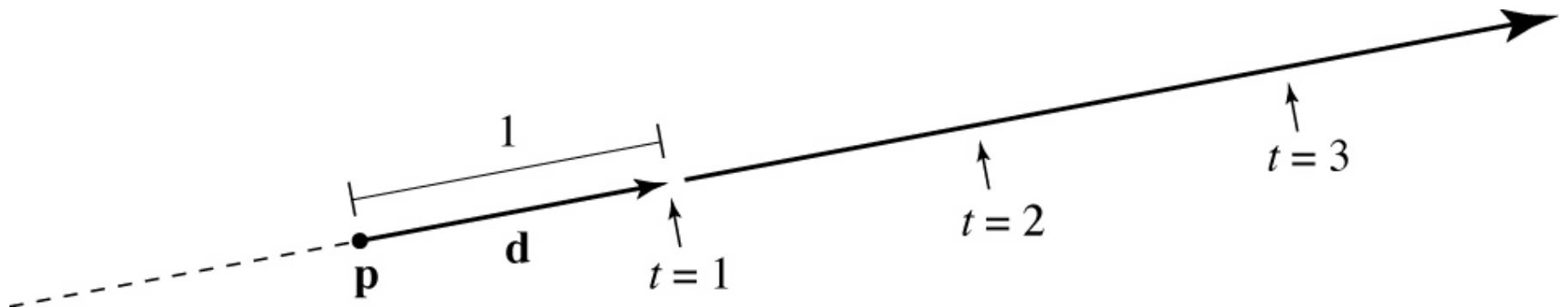


Ray: a half line

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

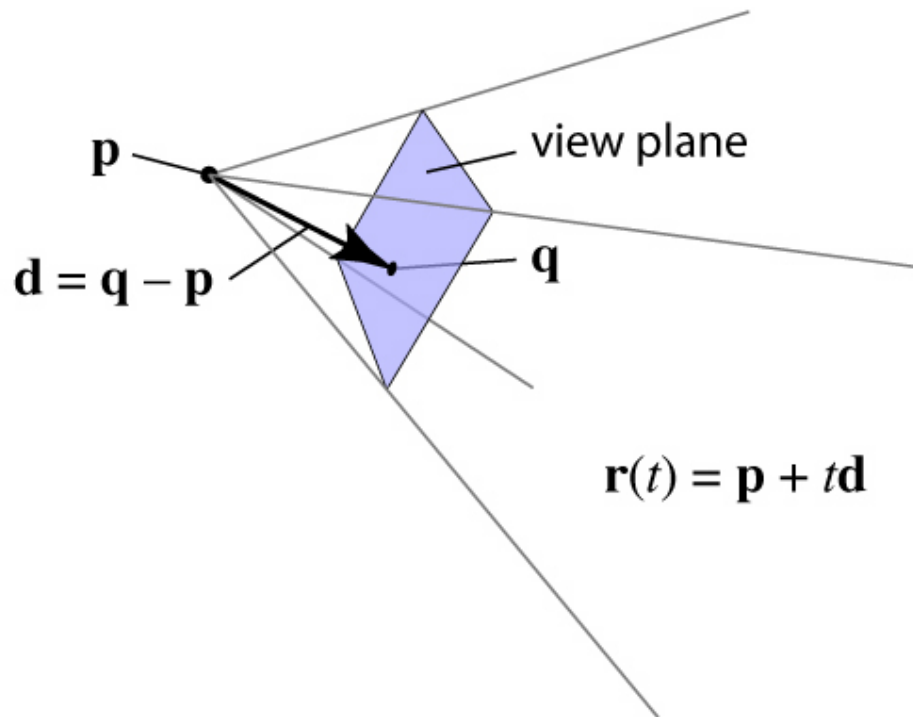
$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- 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 $a\mathbf{d}$ doesn't change ray ($a > 0$)



Generating eye rays

- Just need to compute the view plane point \mathbf{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

$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$

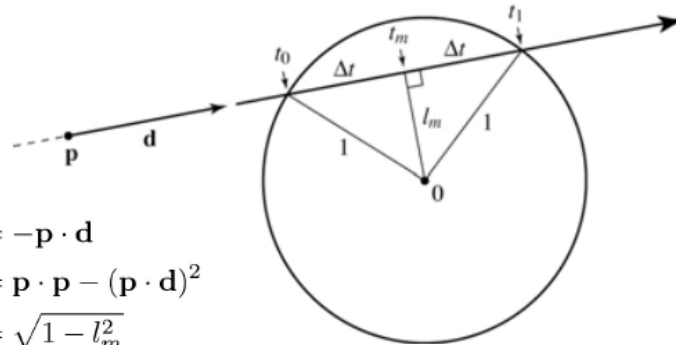
$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- 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)

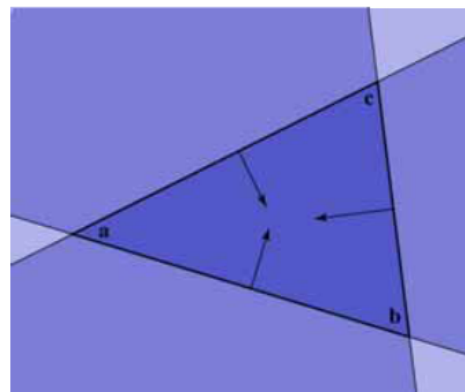
- substitute and solve for t :

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

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

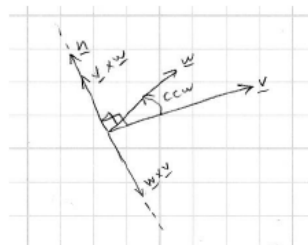
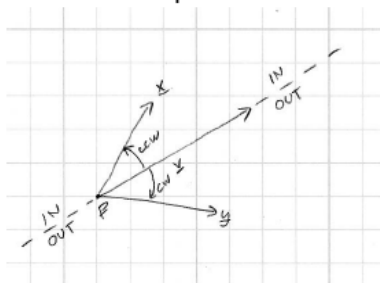
Ray-triangle intersection

- 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 \mathbf{x}
- Use cross product to decide

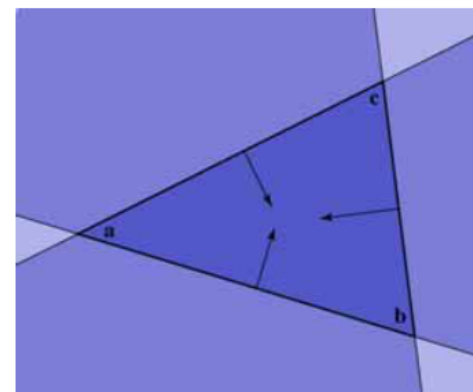


Ray-triangle intersection

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

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

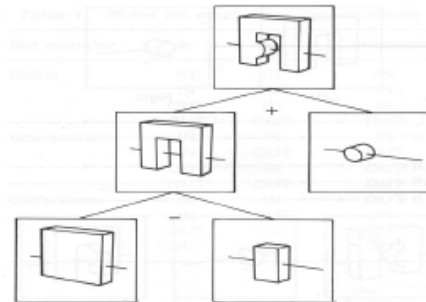
$$(\mathbf{a} - \mathbf{c}) \times (\mathbf{x} - \mathbf{c}) \cdot \mathbf{n} > 0$$



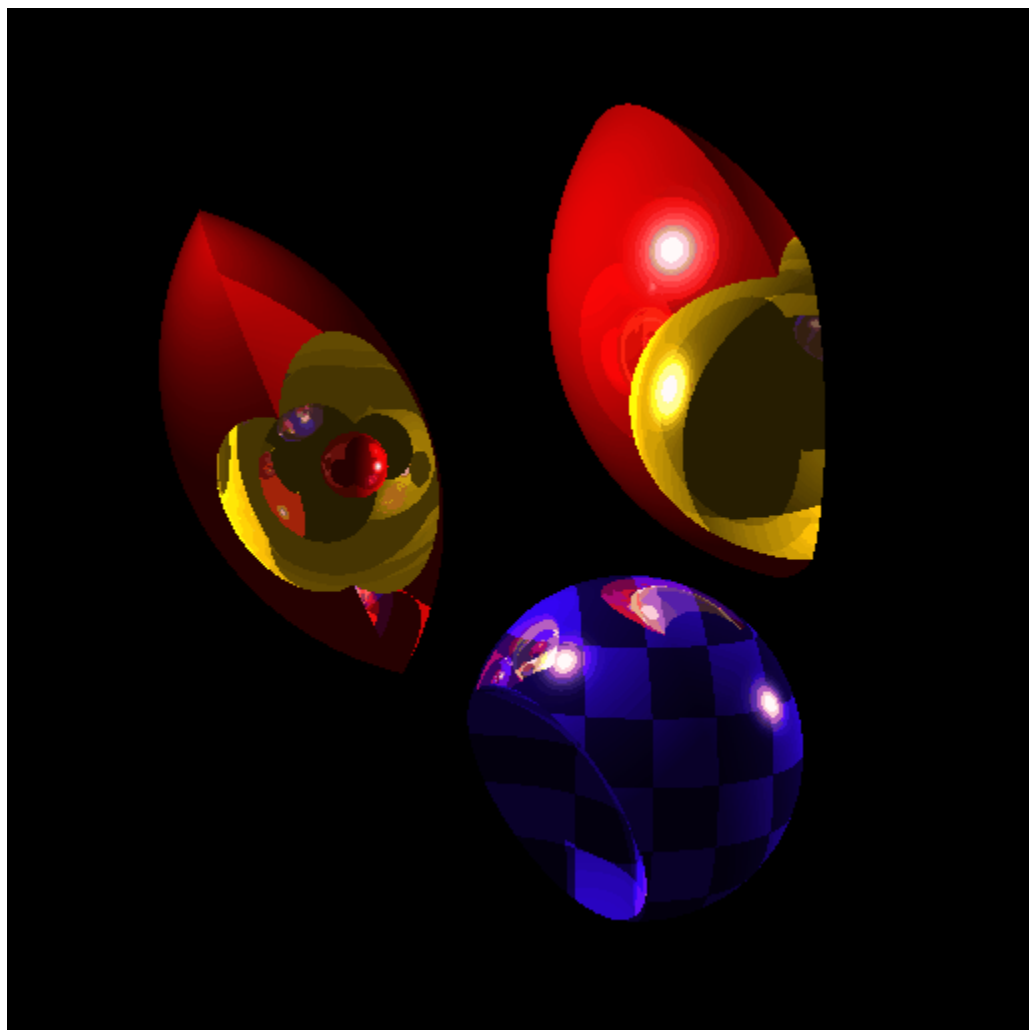
Constructive solid geometry

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)

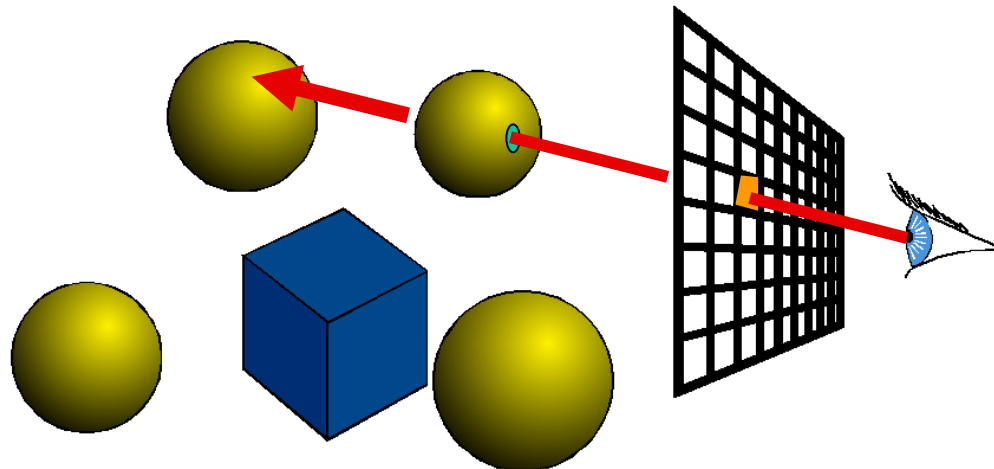
Ray Casting (a.k.a. Ray Shooting)

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

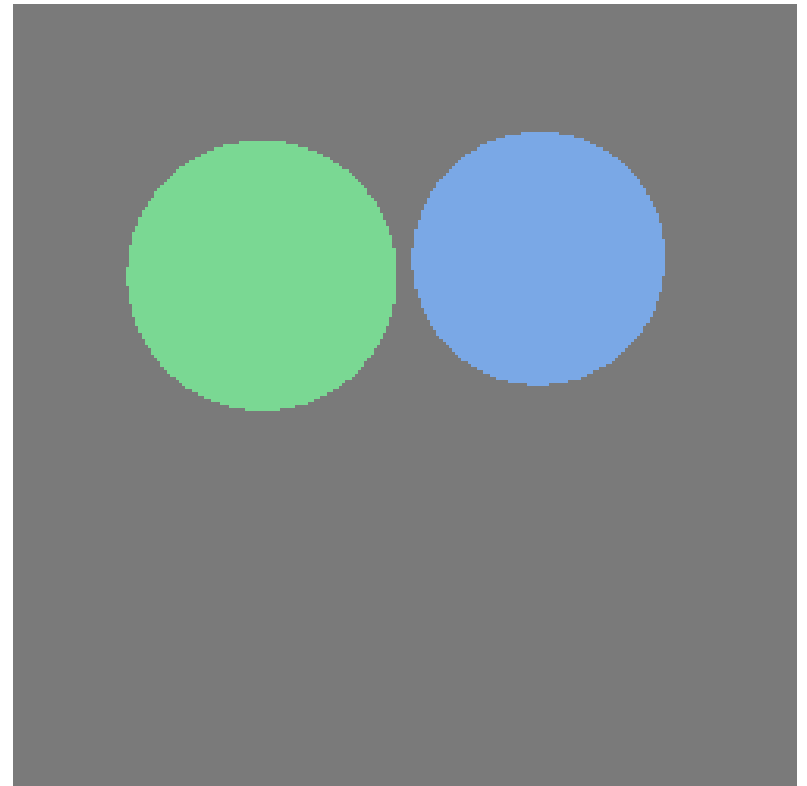
Complexity?

$$O(n * m)$$

n = number of objects, m = number of pixels



Objects (no lighting)



Add lighting to your code

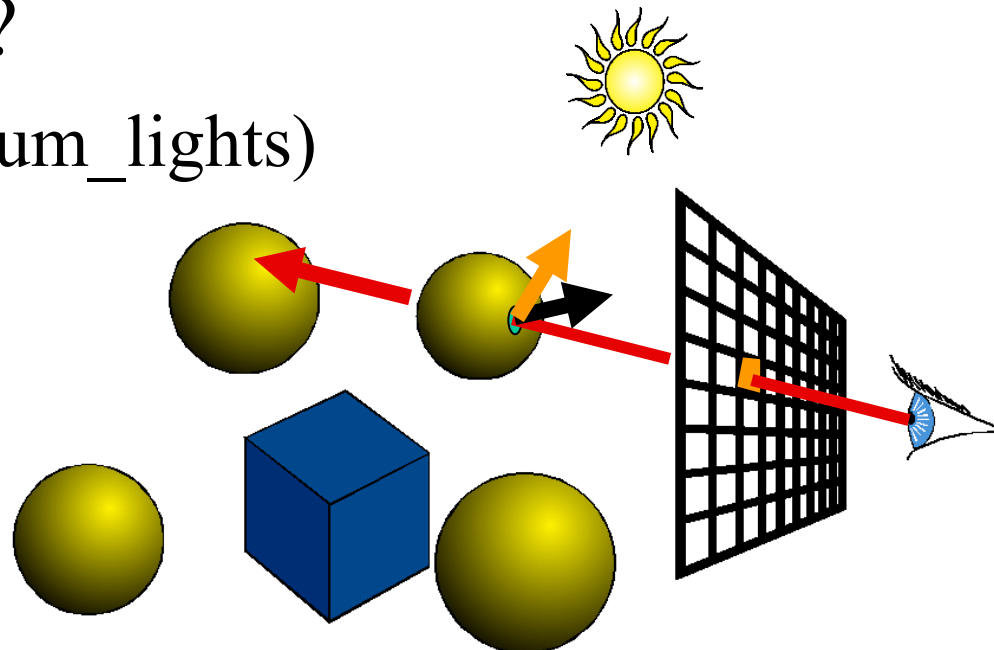
Ray Casting with Phong Shading

When you've found the closest intersection:

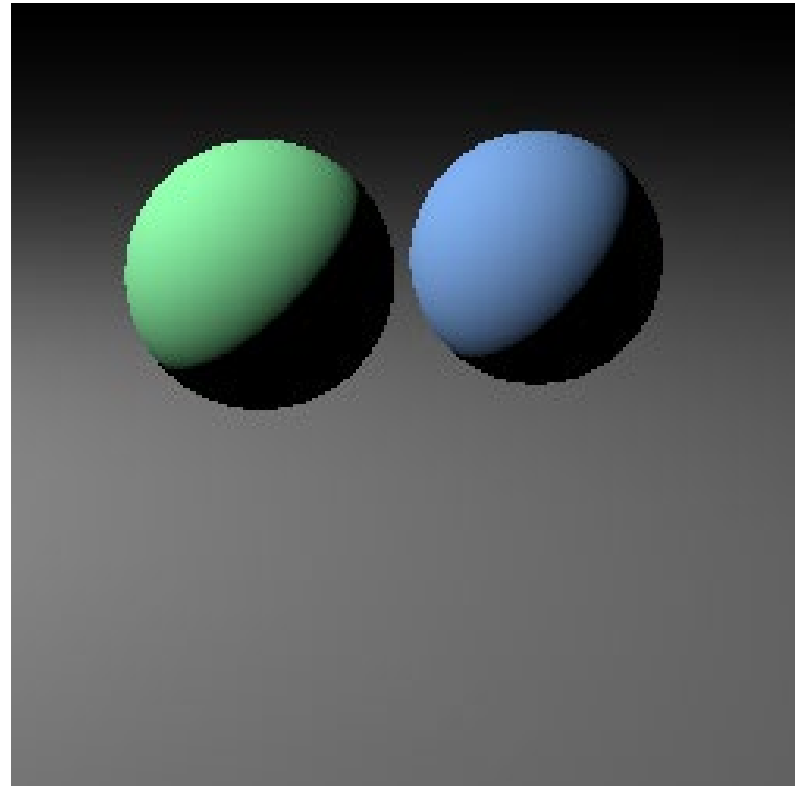
```
color = ambient*hit->getMaterial()->getDiffuseColor()  
for every light  
    color += hit->getMaterial()->Shade  
                (ray, hit, directionToLight, lightColor)  
return color
```

Complexity?

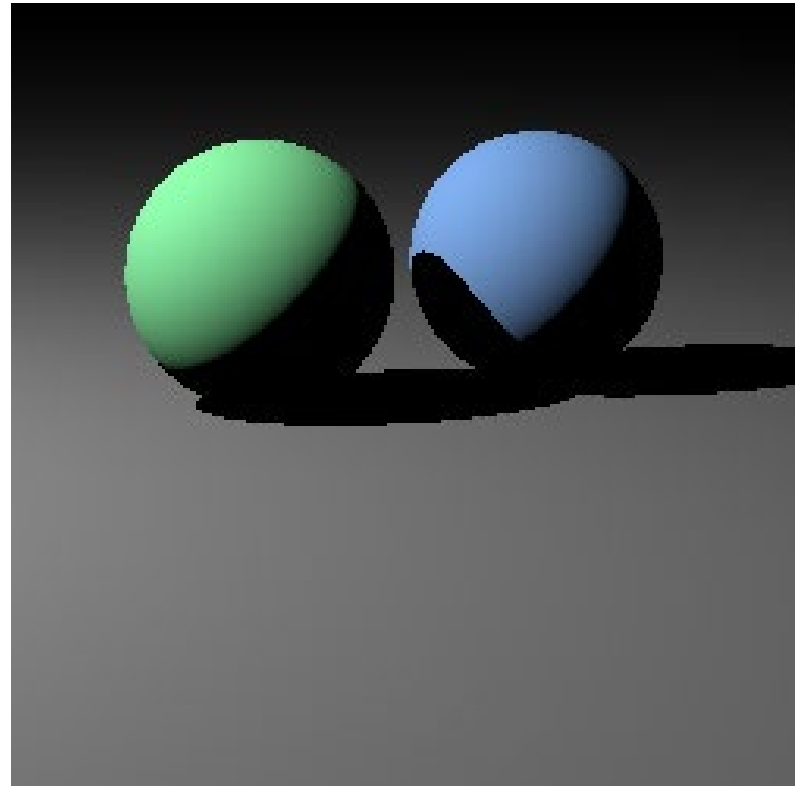
$O(n * m * \text{num_lights})$



Add lighting



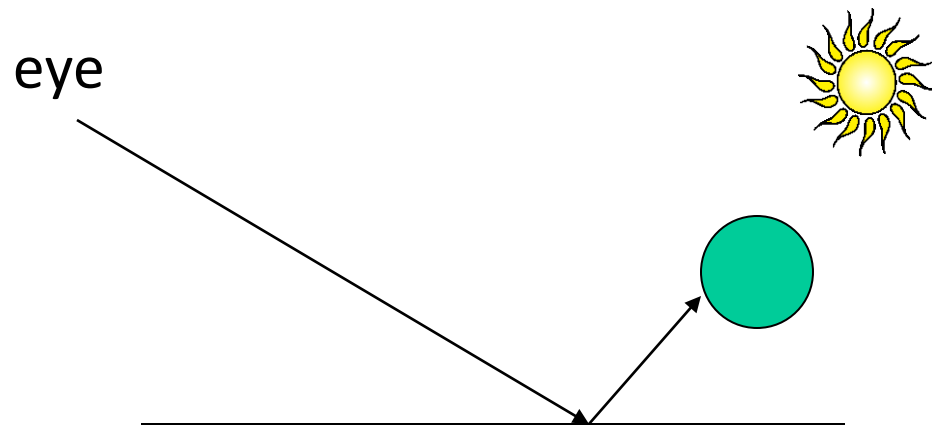
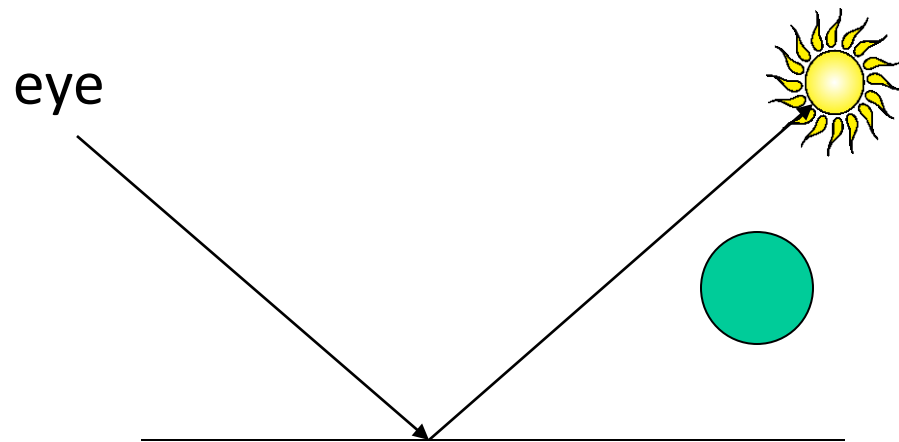
Add shadows



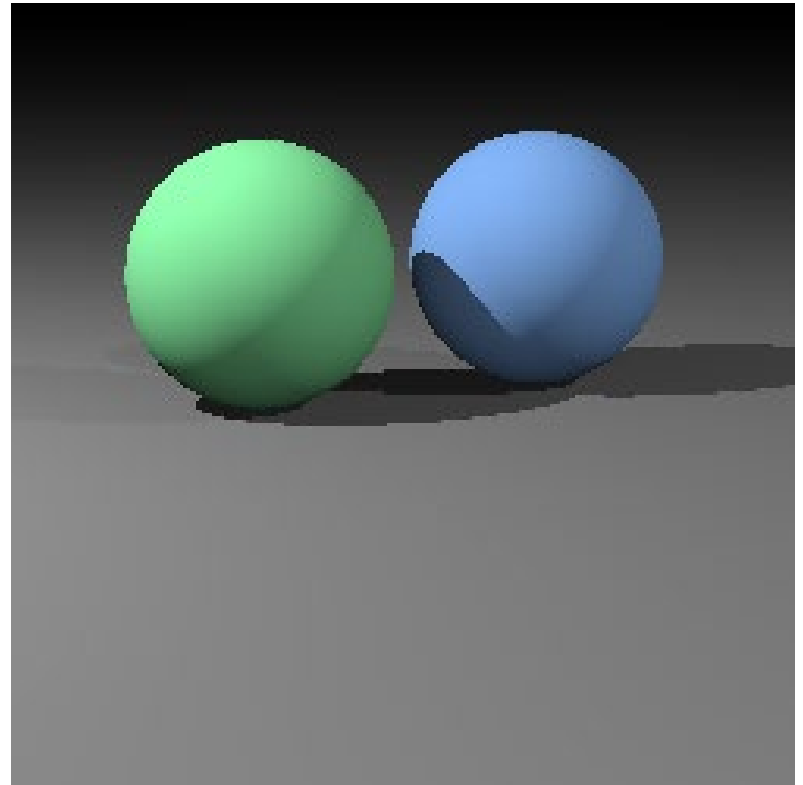
Add shadows to your code

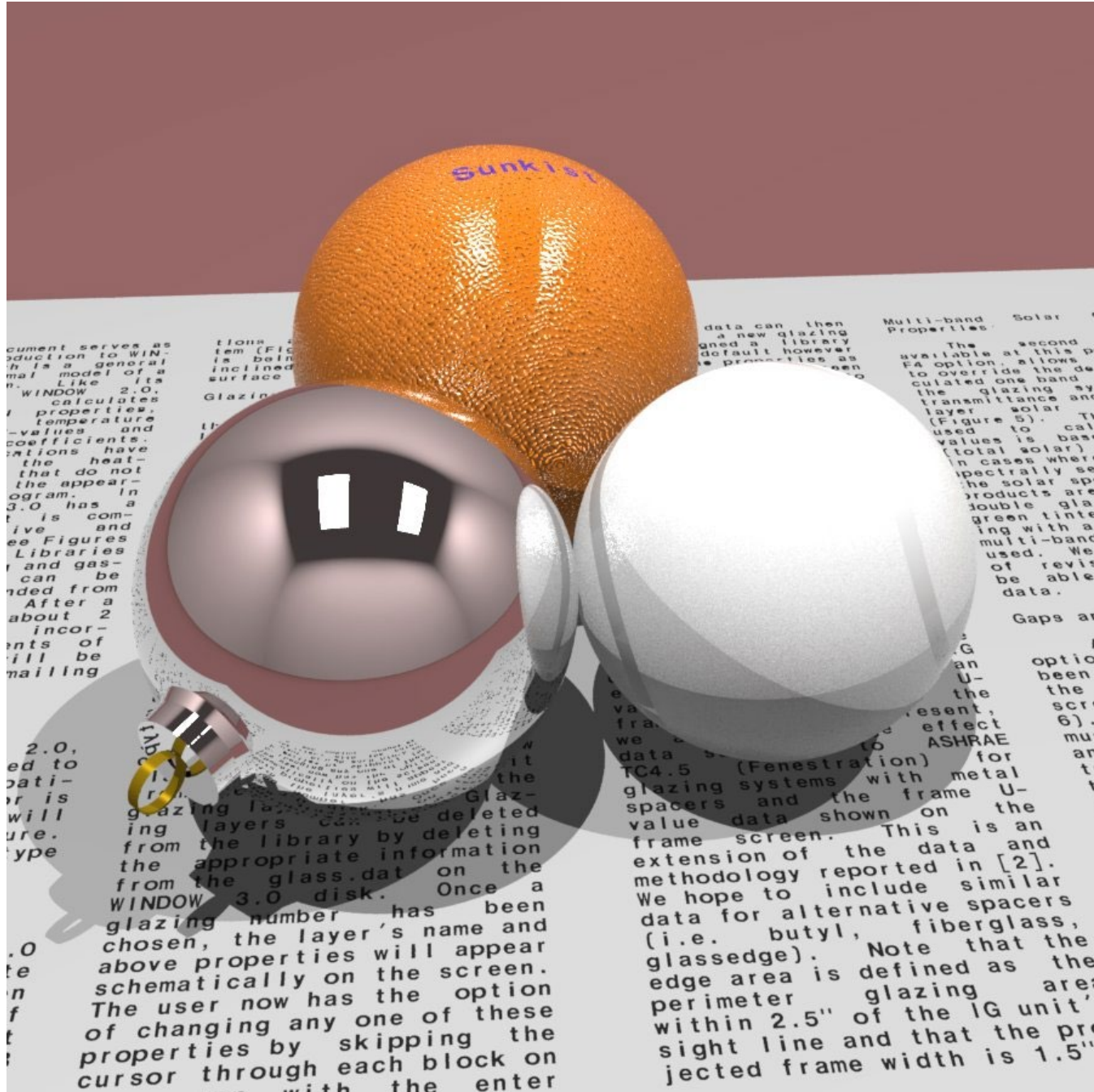
Q: How to calculate shadow

- 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



Multiple lights

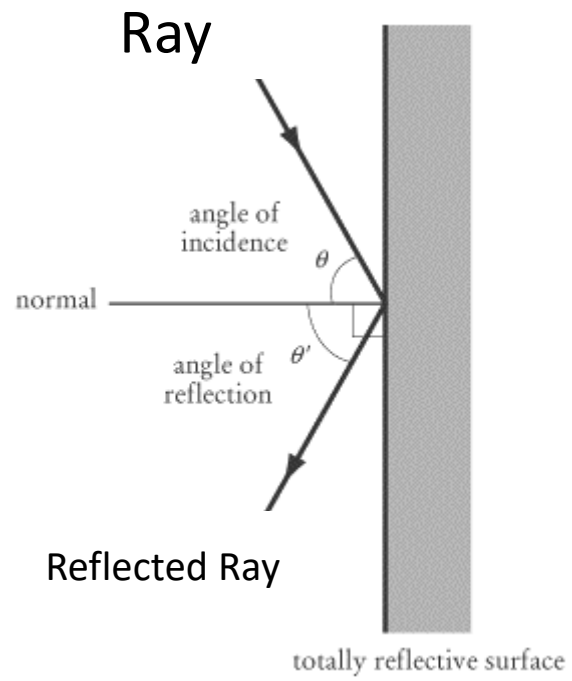




Add reflection to your code

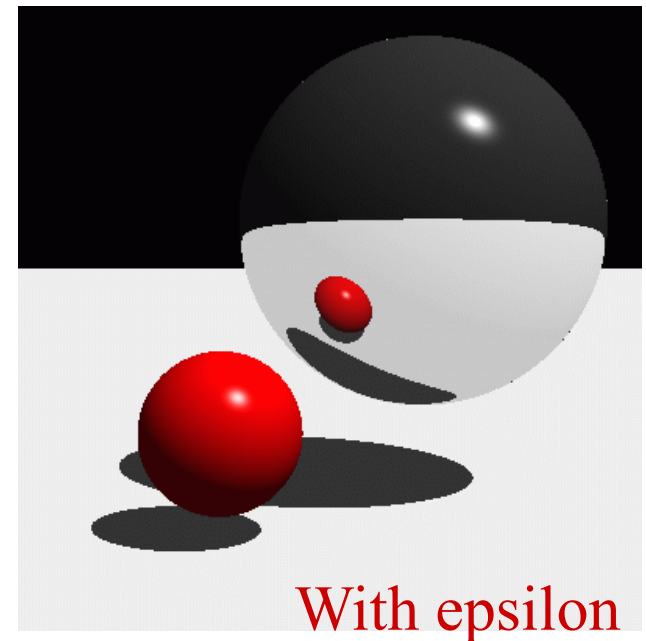
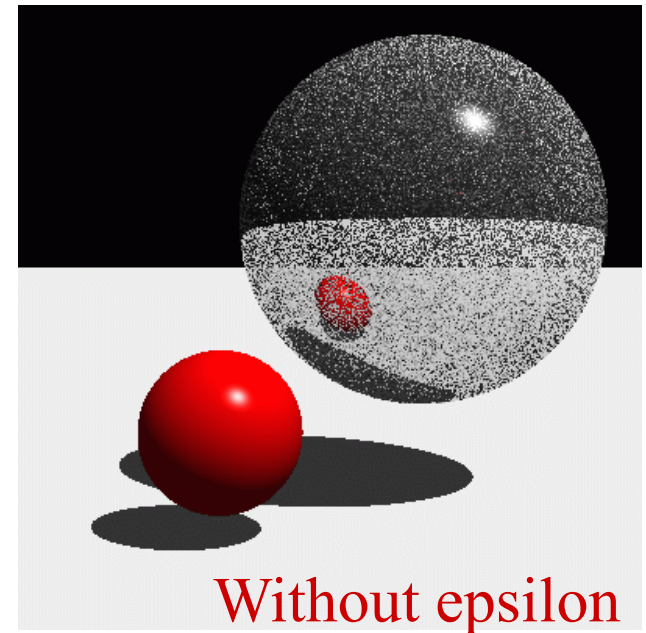
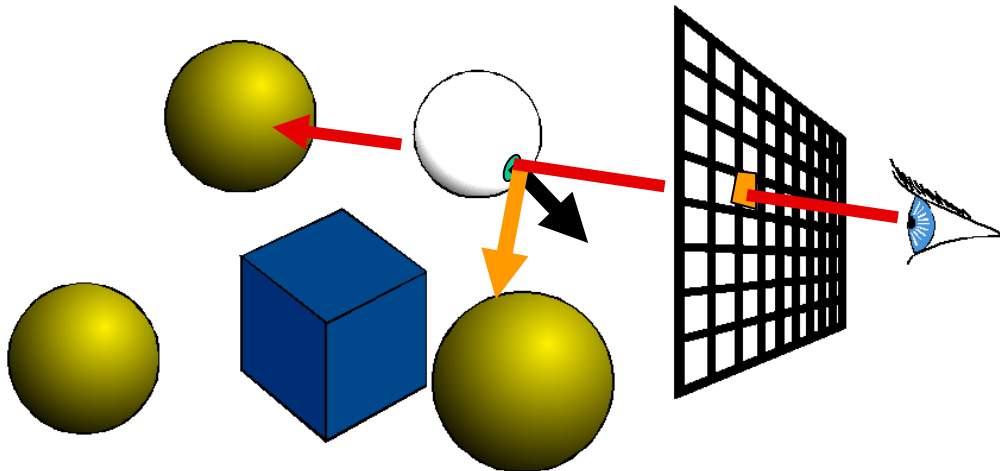
Q: How to calculate reflection

- 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



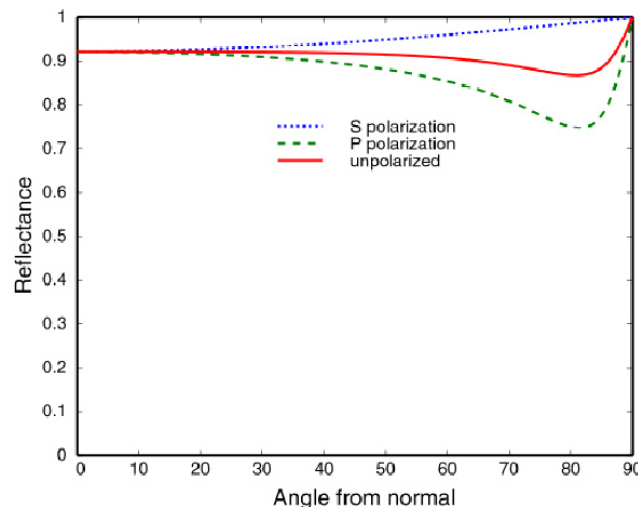
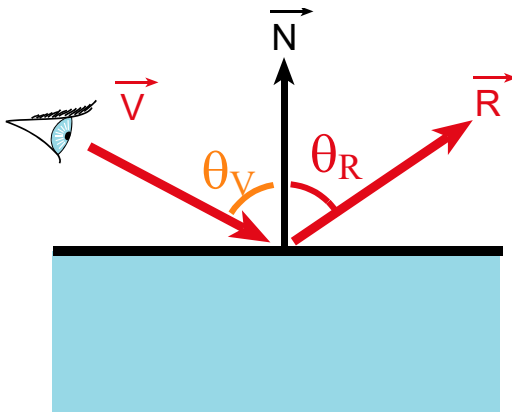
Mirror Reflection

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

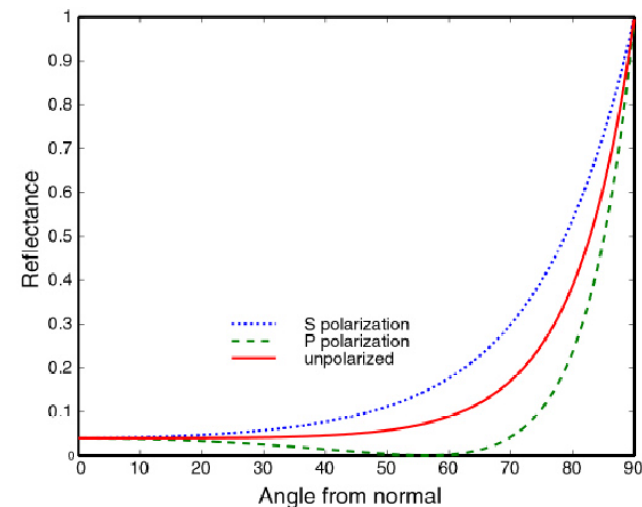


Amount of Reflection

- 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



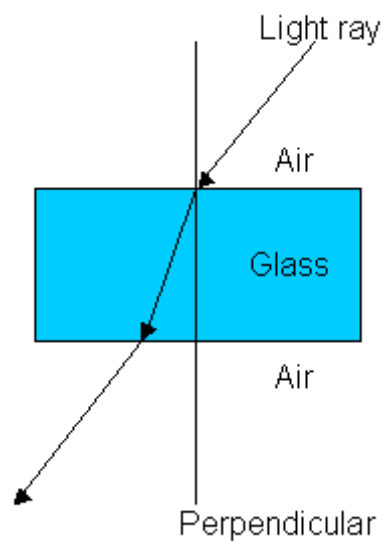
Dielectric (glass)



Add refraction to your code

Q: How to calculate refraction

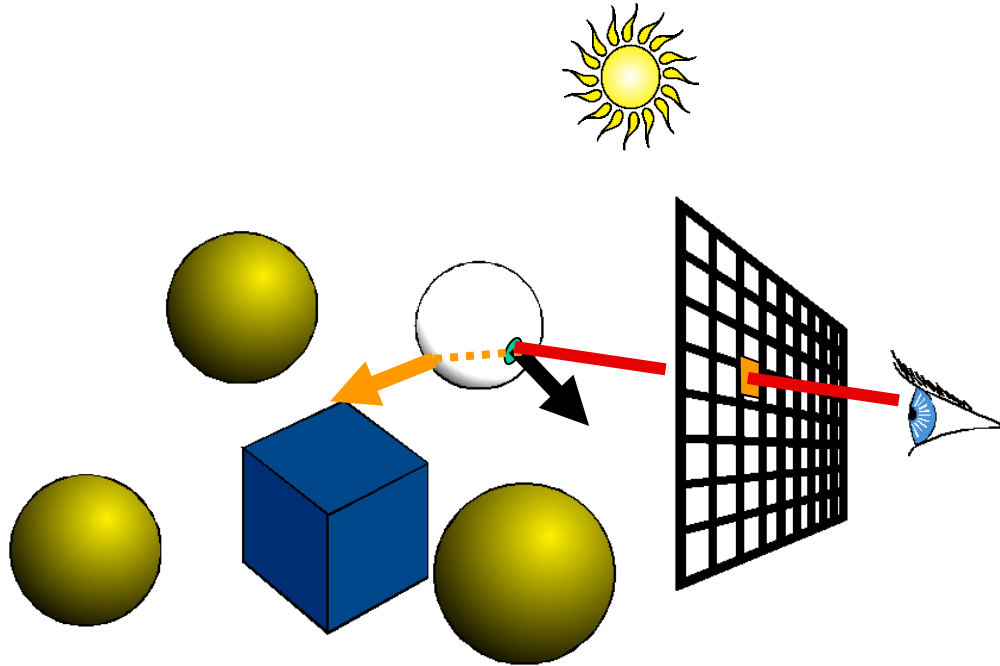
- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions

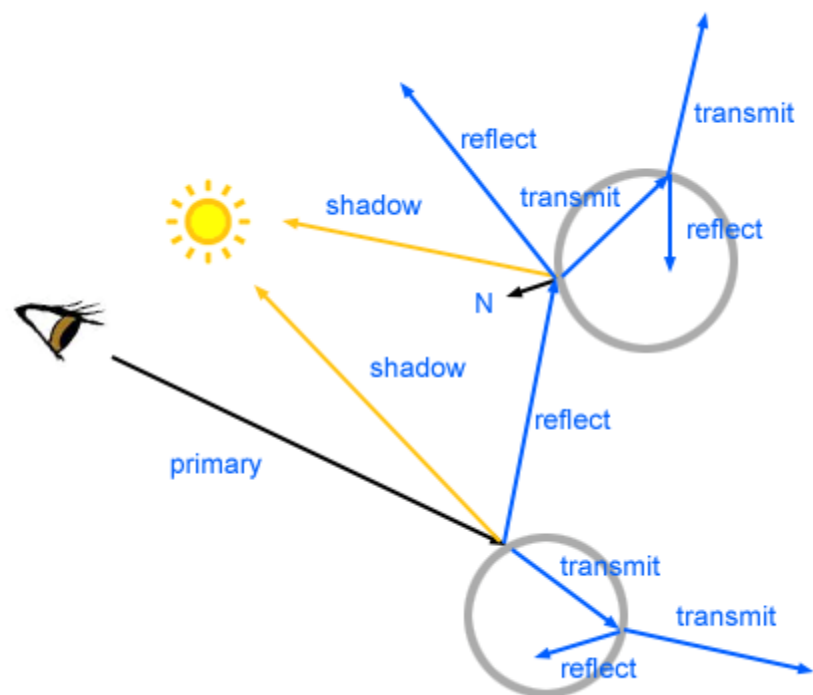


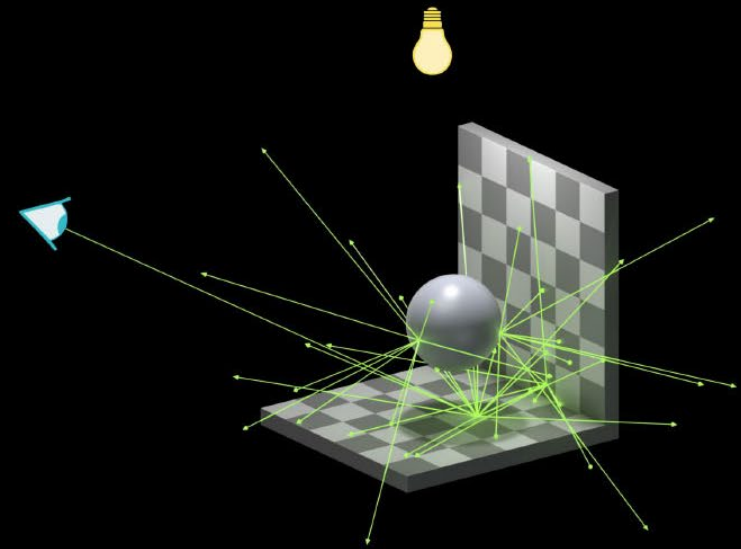


Transparency

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







Rays Needed	Many
BVH Search	Global
Bounces	Optional

- <http://tiny.cc/160-1201>

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...

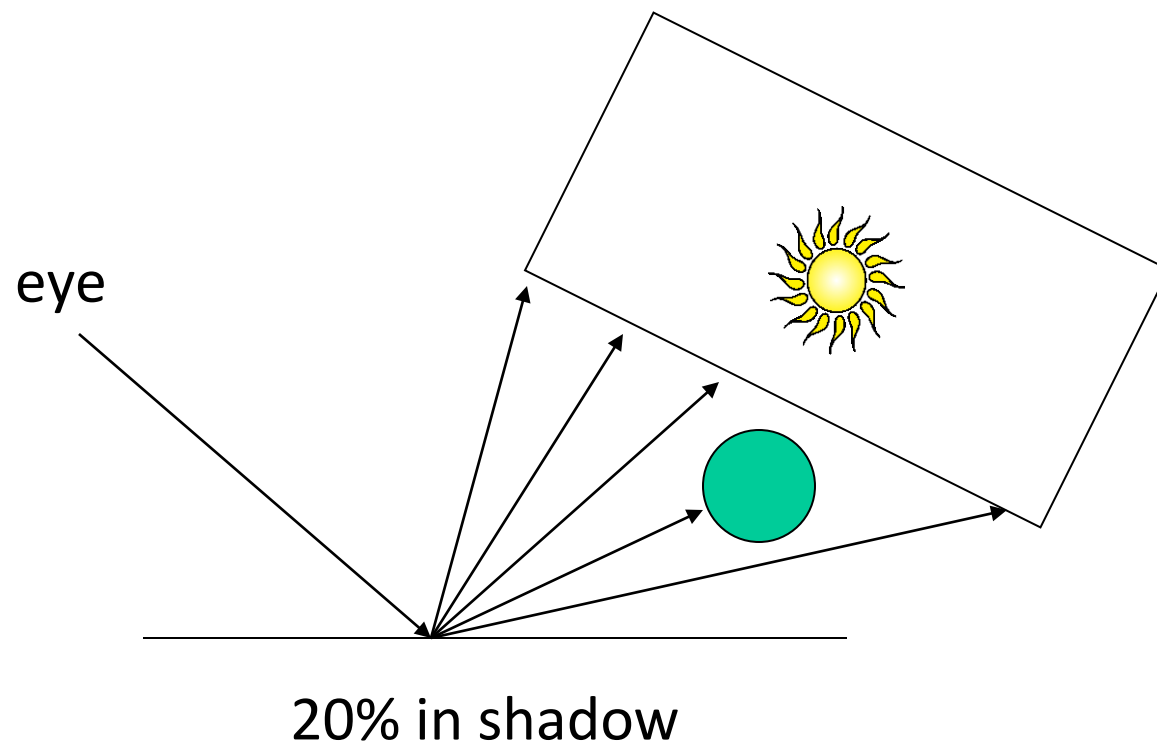
I want to go to grad school:

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

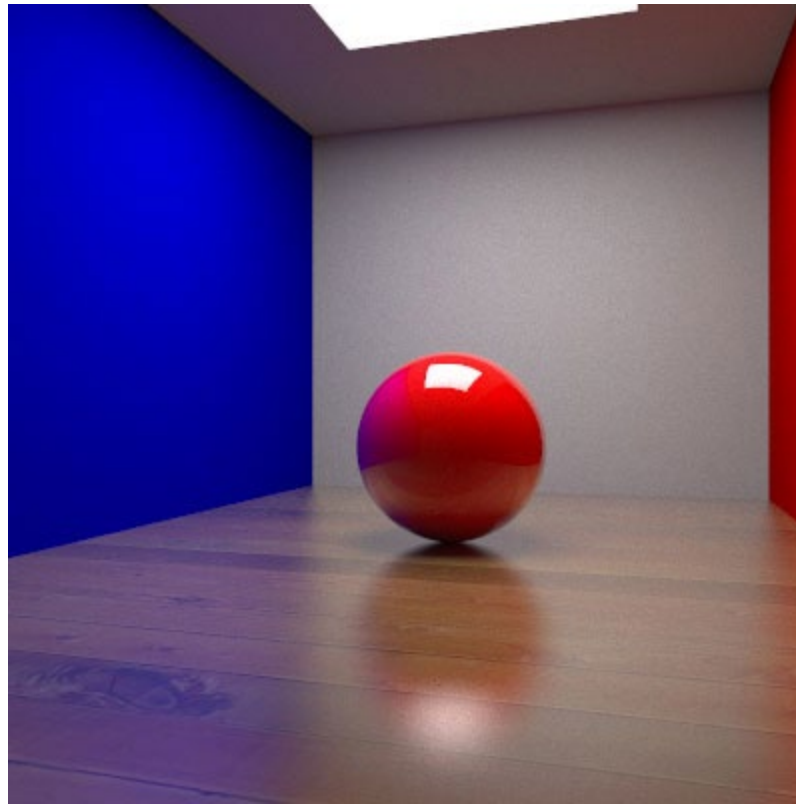
Distributed raytracing

Soft shadows?



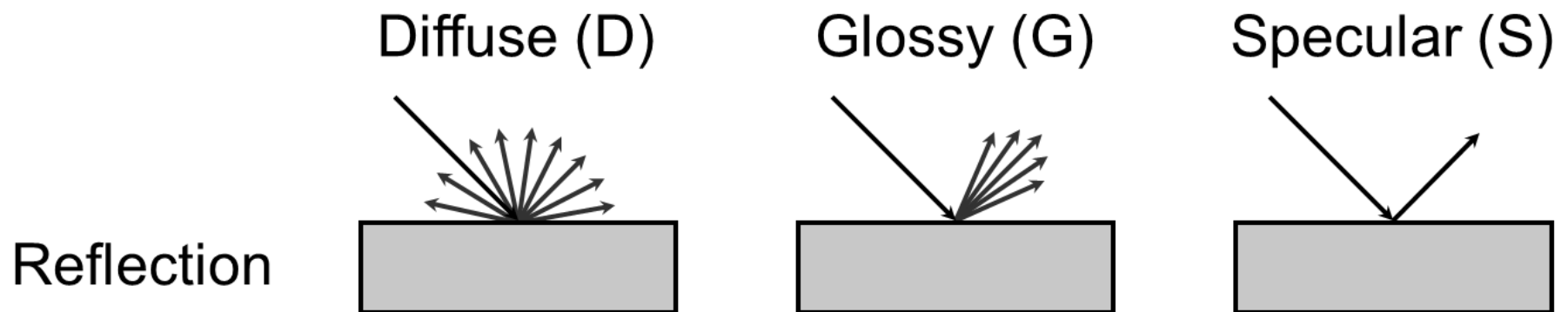
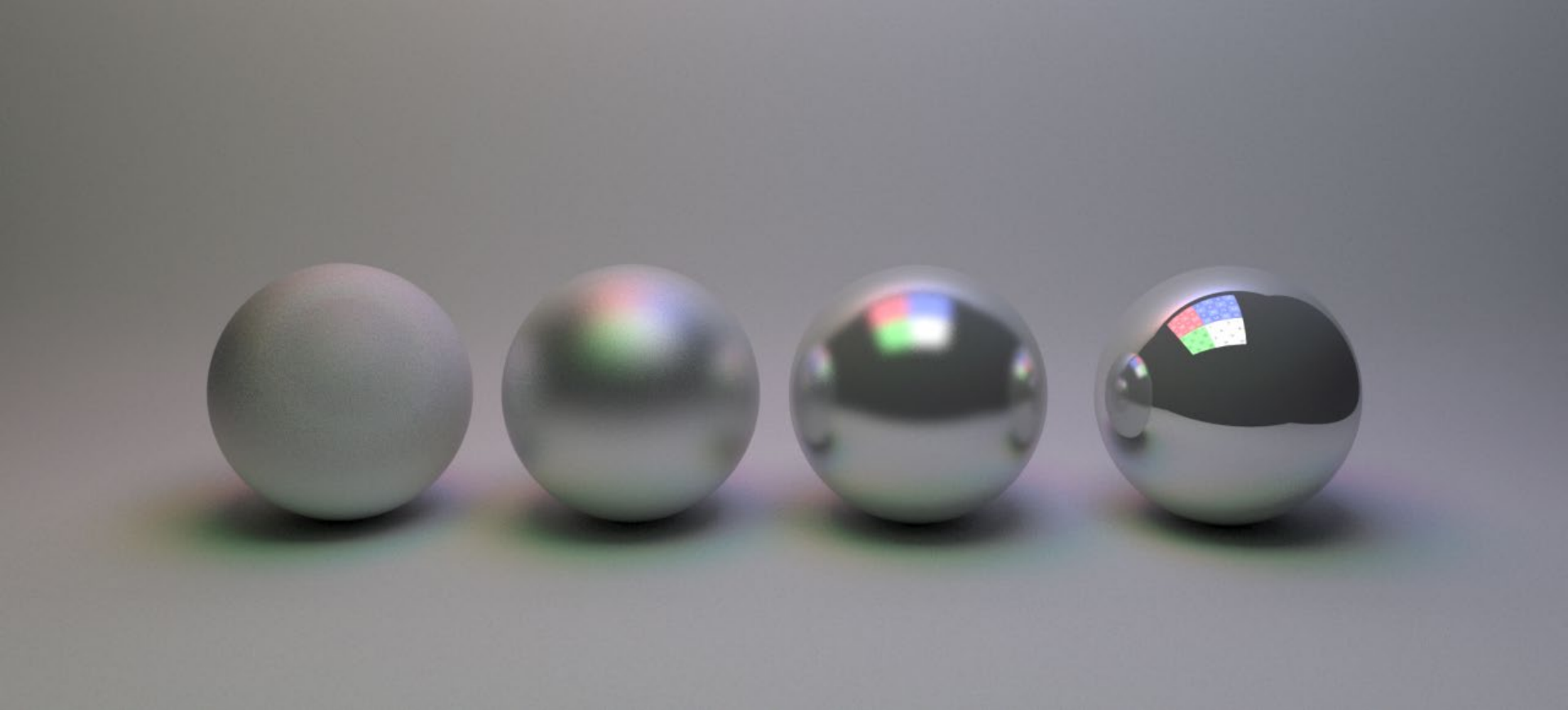


Glossy reflections?

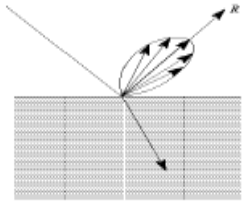


Q: How to calculate glossy surface

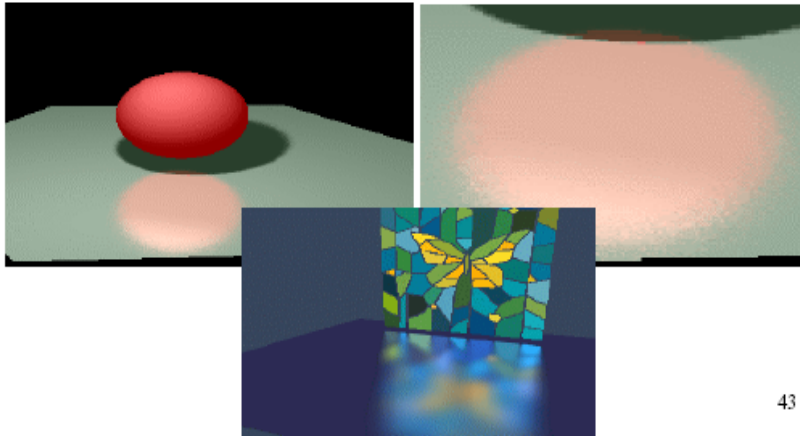
- A) Send a ray to the eye
- B) Send a ray through the surface to other side
- C) Send a ray to the light
- D) Send a ray in the reflection direction
- E) Send lots of rays in all directions



Distributing Reflections



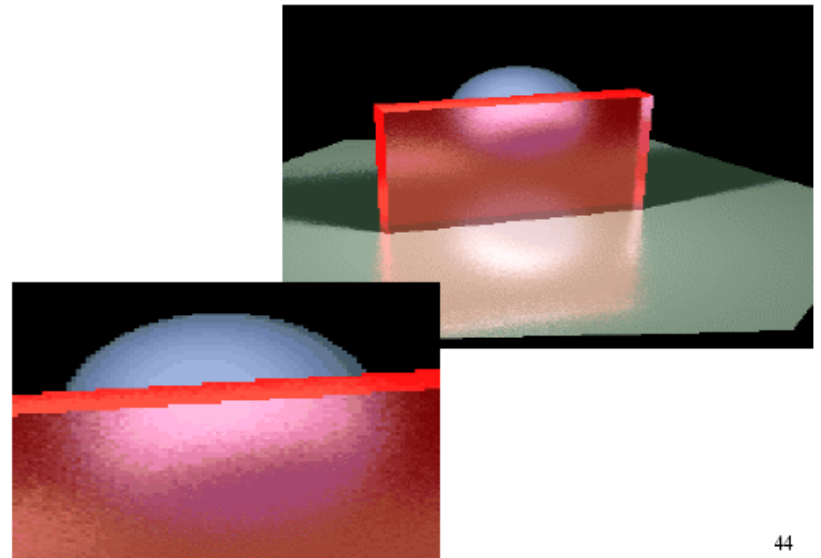
- Distributing rays over reflection direction gives:



43

Distributing Refractions

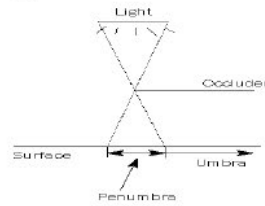
- Distributing rays over transmission direction gives:



44

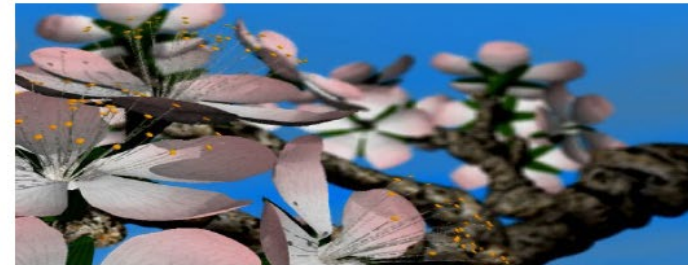
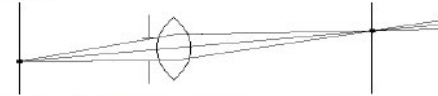
Distributing Over Light Area

- 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".

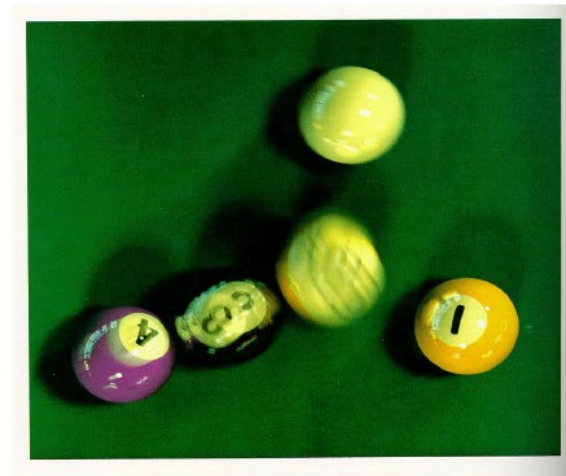


Distributing Over Time

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

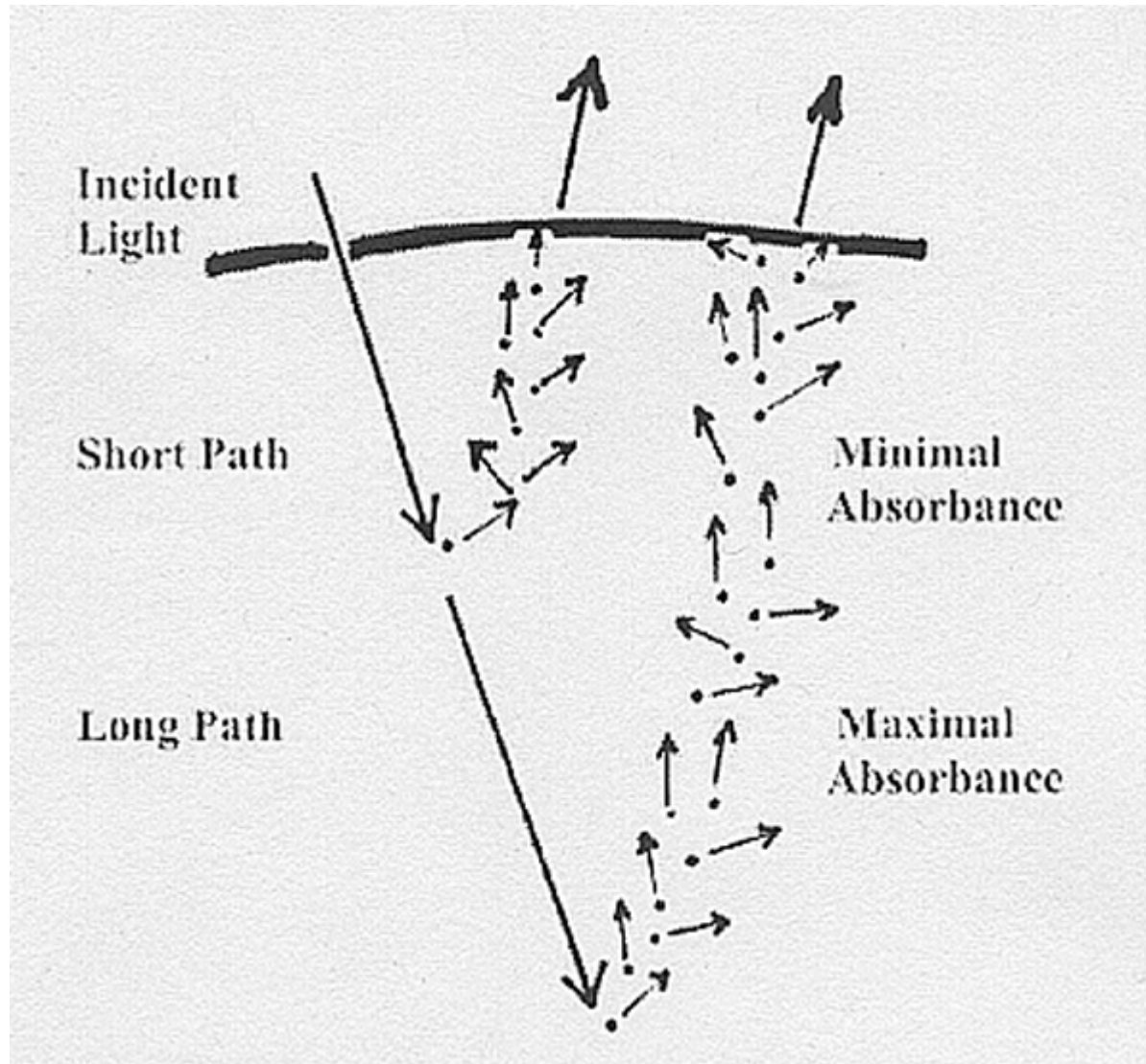


47



48

Subsurface scattering







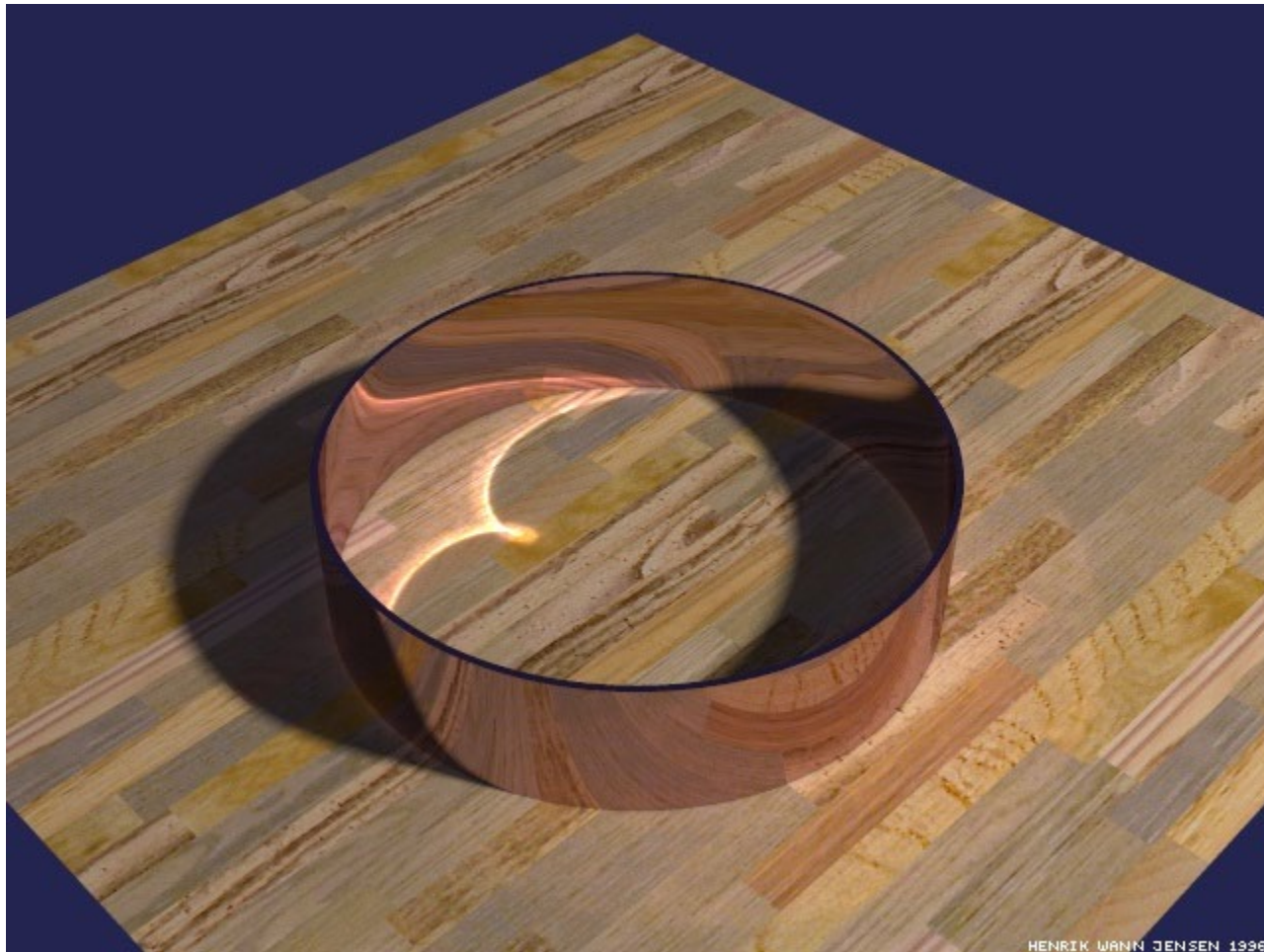
HENRIK WAPNIN ZEHSEN 2000



HENRIK WAPNIN ZEHSEN 2001

Photon Mapping

How do we get this effect? (caustics)

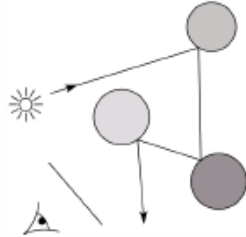




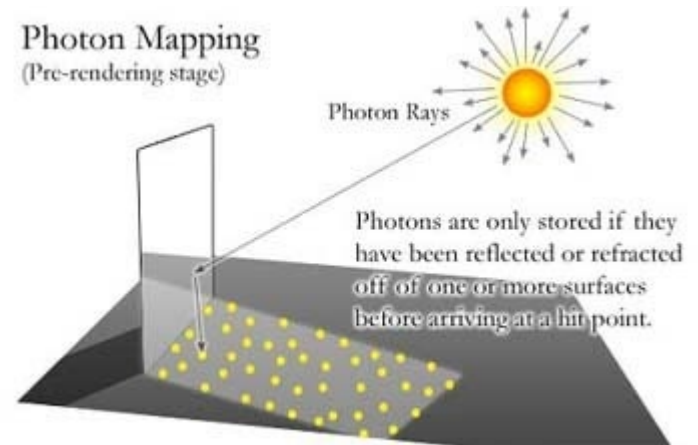
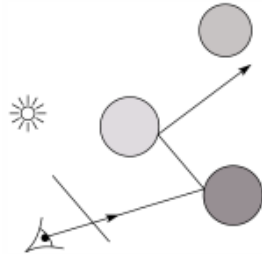
Do both directions. Deposit light in the scene from the light.

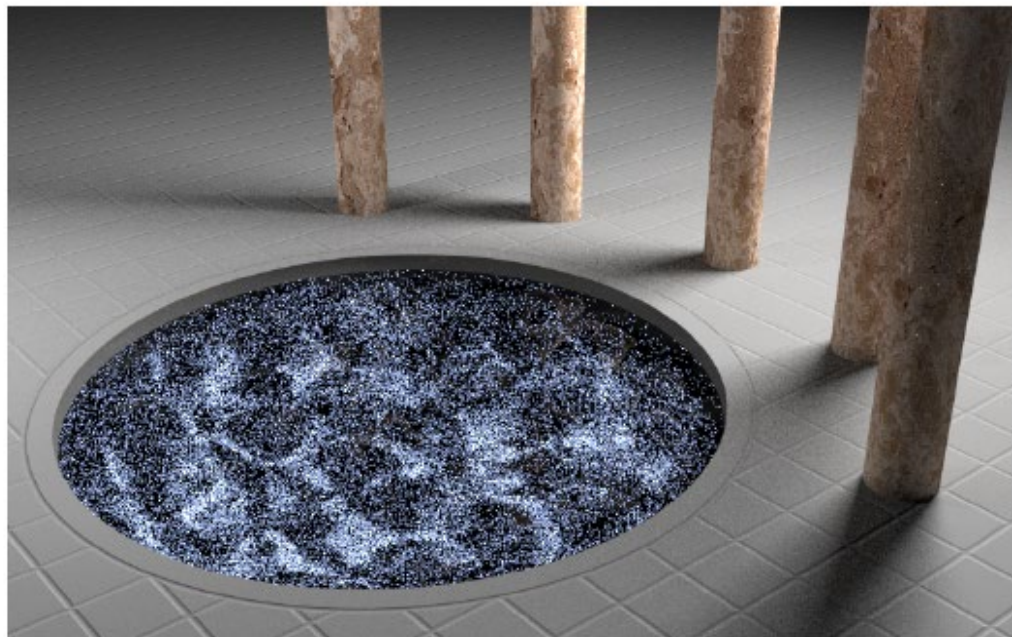
Eye vs. Light

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

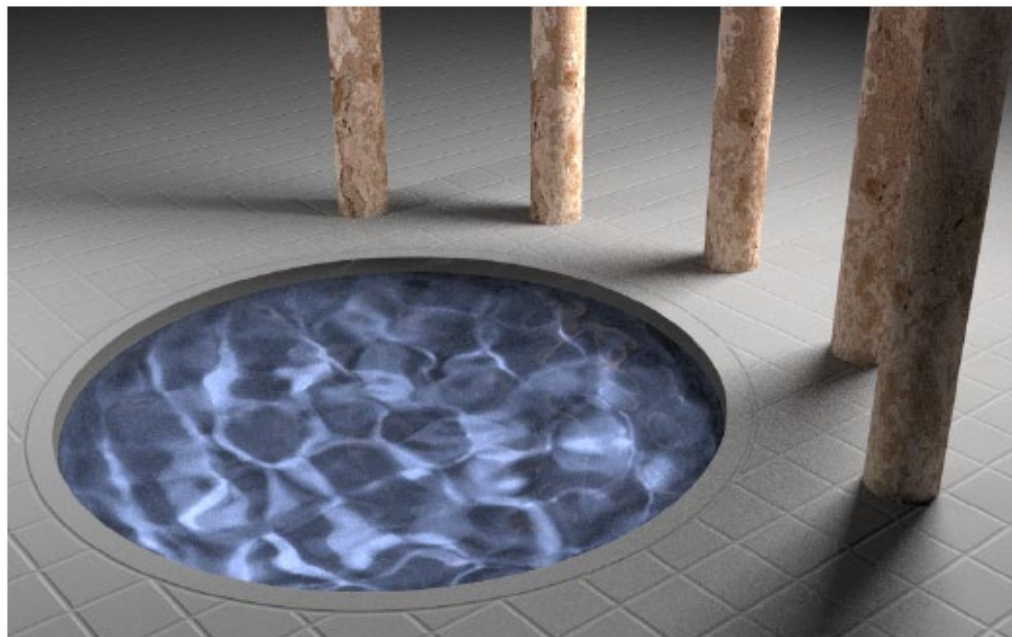


- Starting at the eye (a.k.a. backward ray tracing)





(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)

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.



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

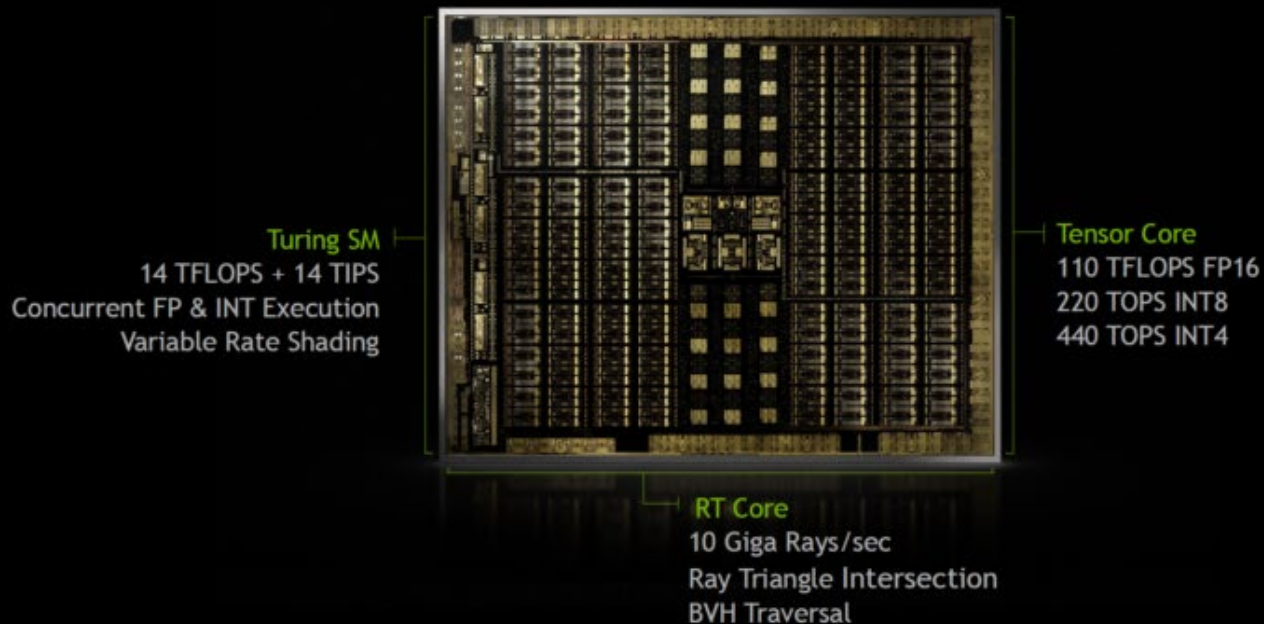
SHOP ALL

GEFORCE RTX
2060 SUPER
\$399.⁰⁰

SHOP ALL

TURING BUILT FOR RTX

GREATEST LEAP SINCE 2006 CUDA GPU

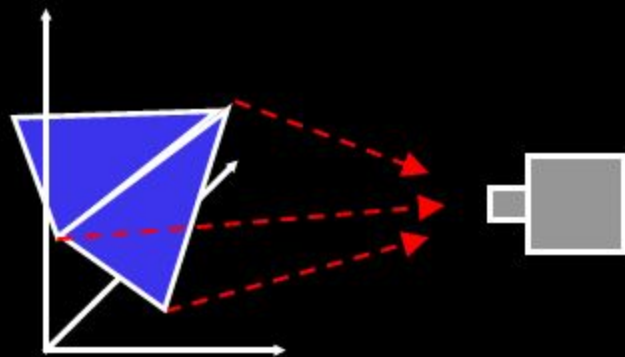


Introduction to Realtime Ray Tracing



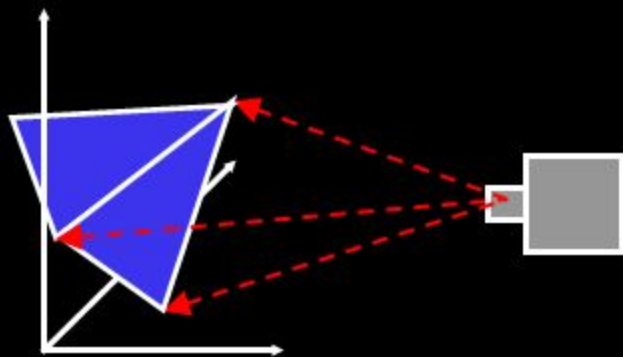
SIGGRAPH2005

Rendering in Computer Graphics



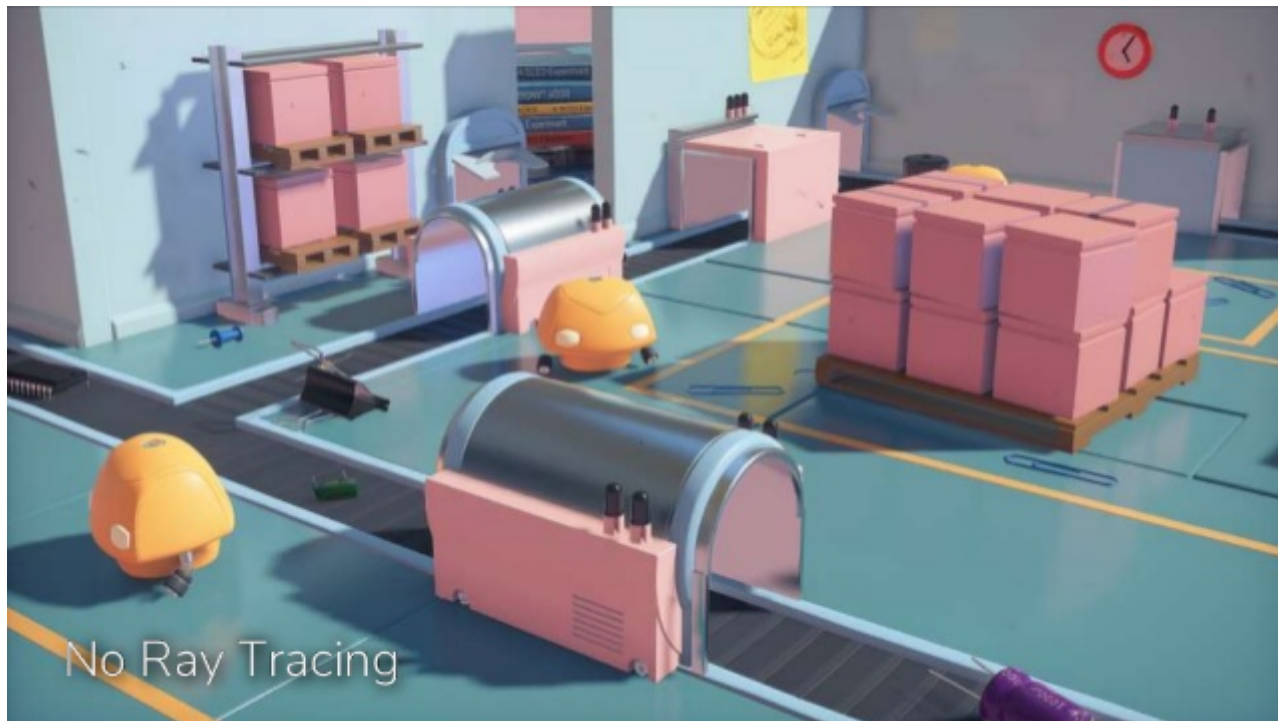
Rasterization:

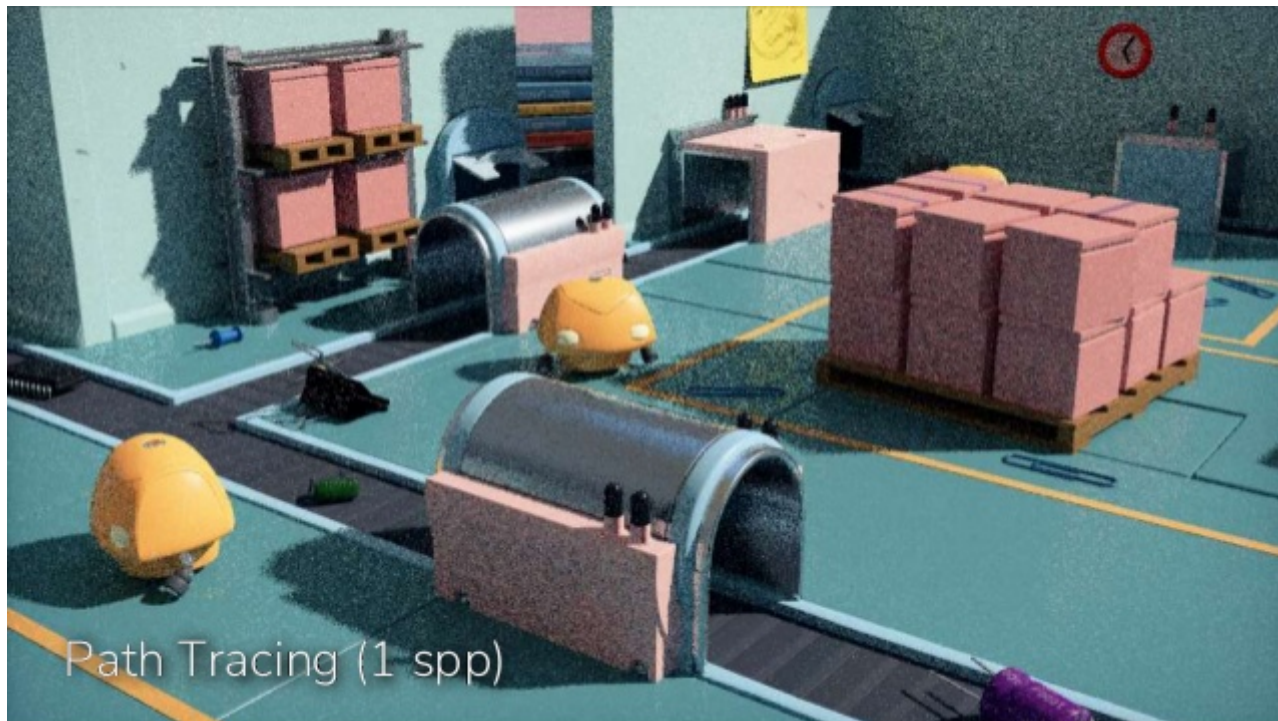
Projection geometry forward

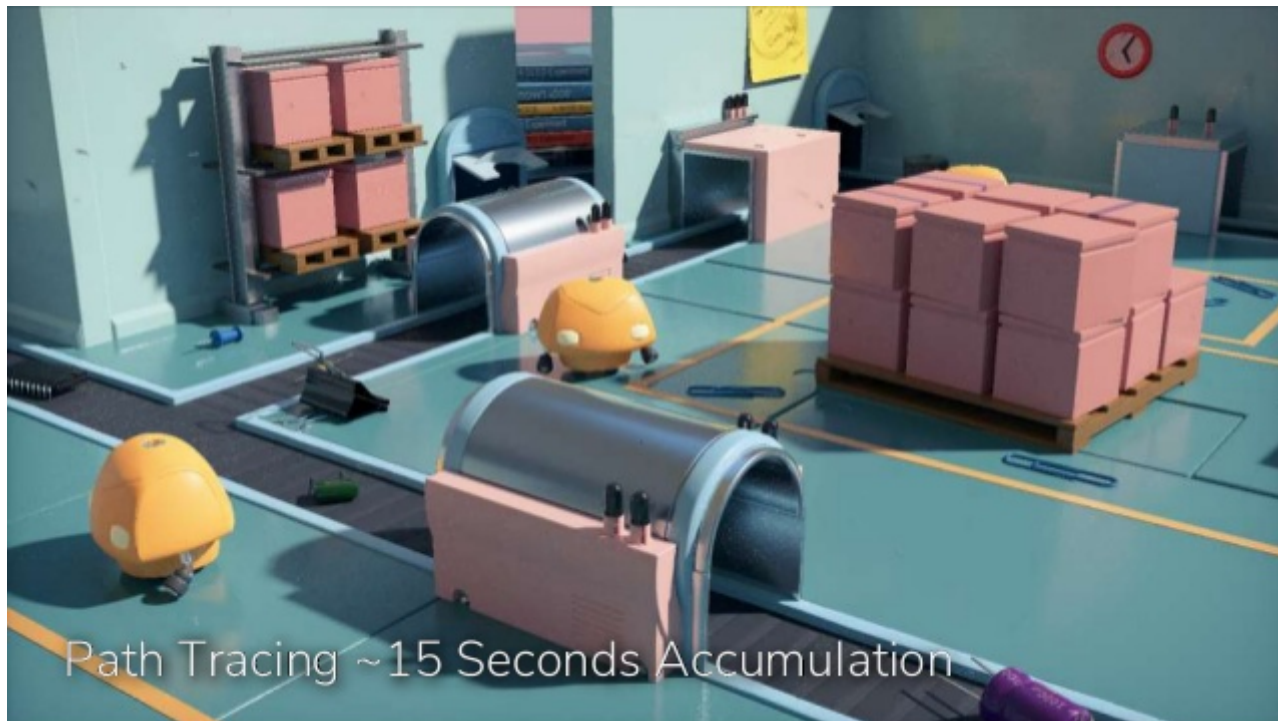


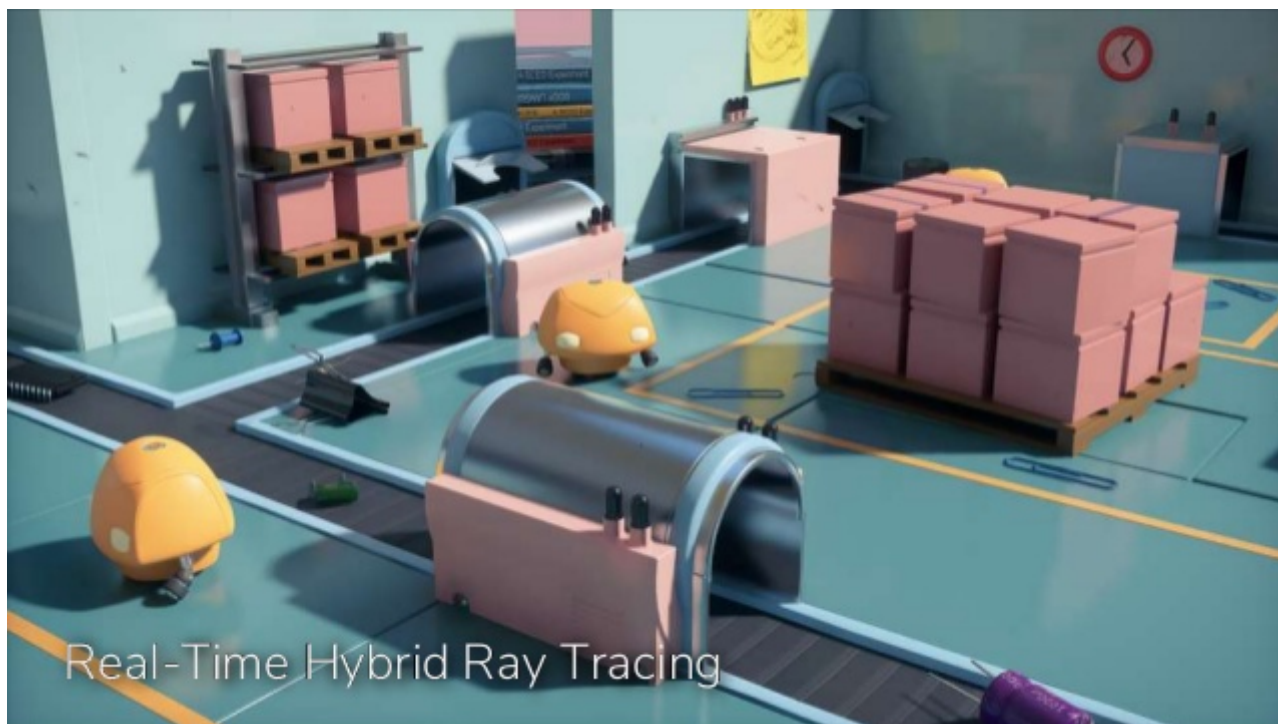
Ray Tracing:

Project image samples backwards

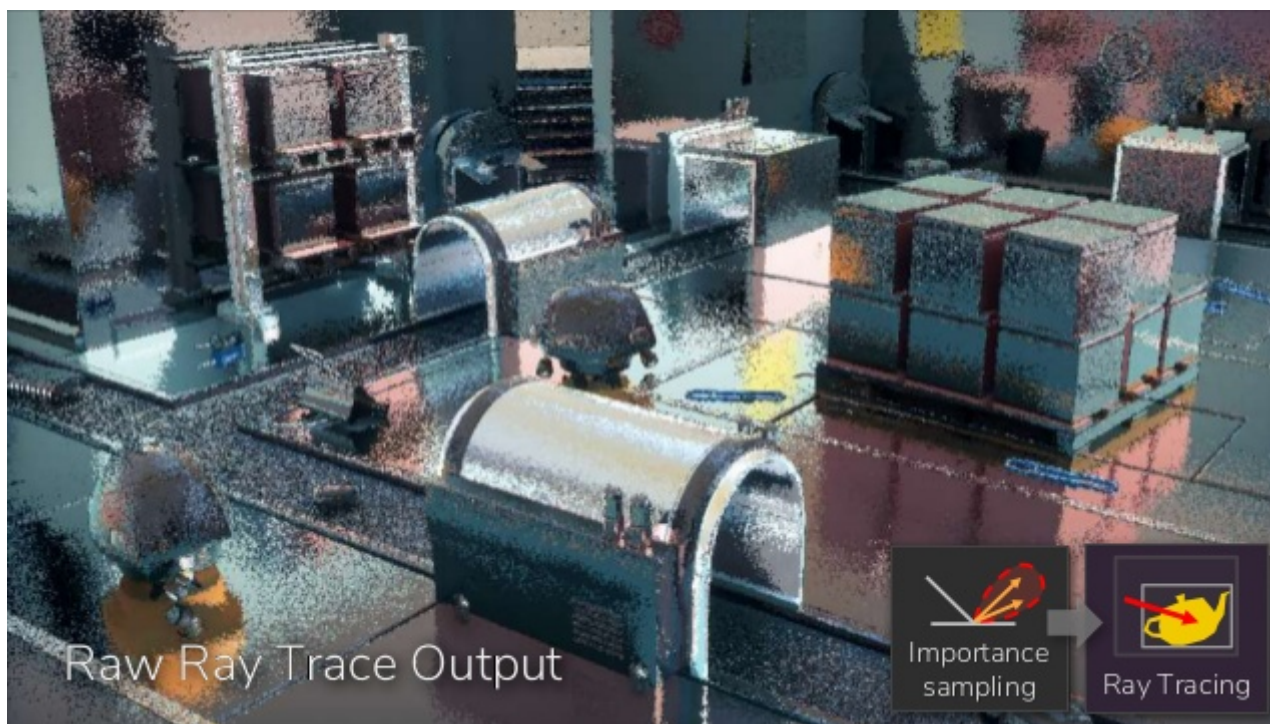




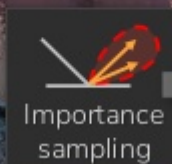




Real-Time Hybrid Ray Tracing



Raw Ray Trace Output



Administrative

Remaining Classes

- This Thur
 - Class summary – fast forward of class topics
 - About jobs and grad school (very popular topics)
- Next Tue
 - How to improve the class - discussion (I learn from you)
- Next Thur
 - Demo day Lab 5 (last class!)

Due Dates

- Due Yesterday
 - A4 Lighting
- Due next Wed
 - Quiz 5 (Cumulative)
- Due next Wed
 - Lab5 – Three.js

Q&A

End