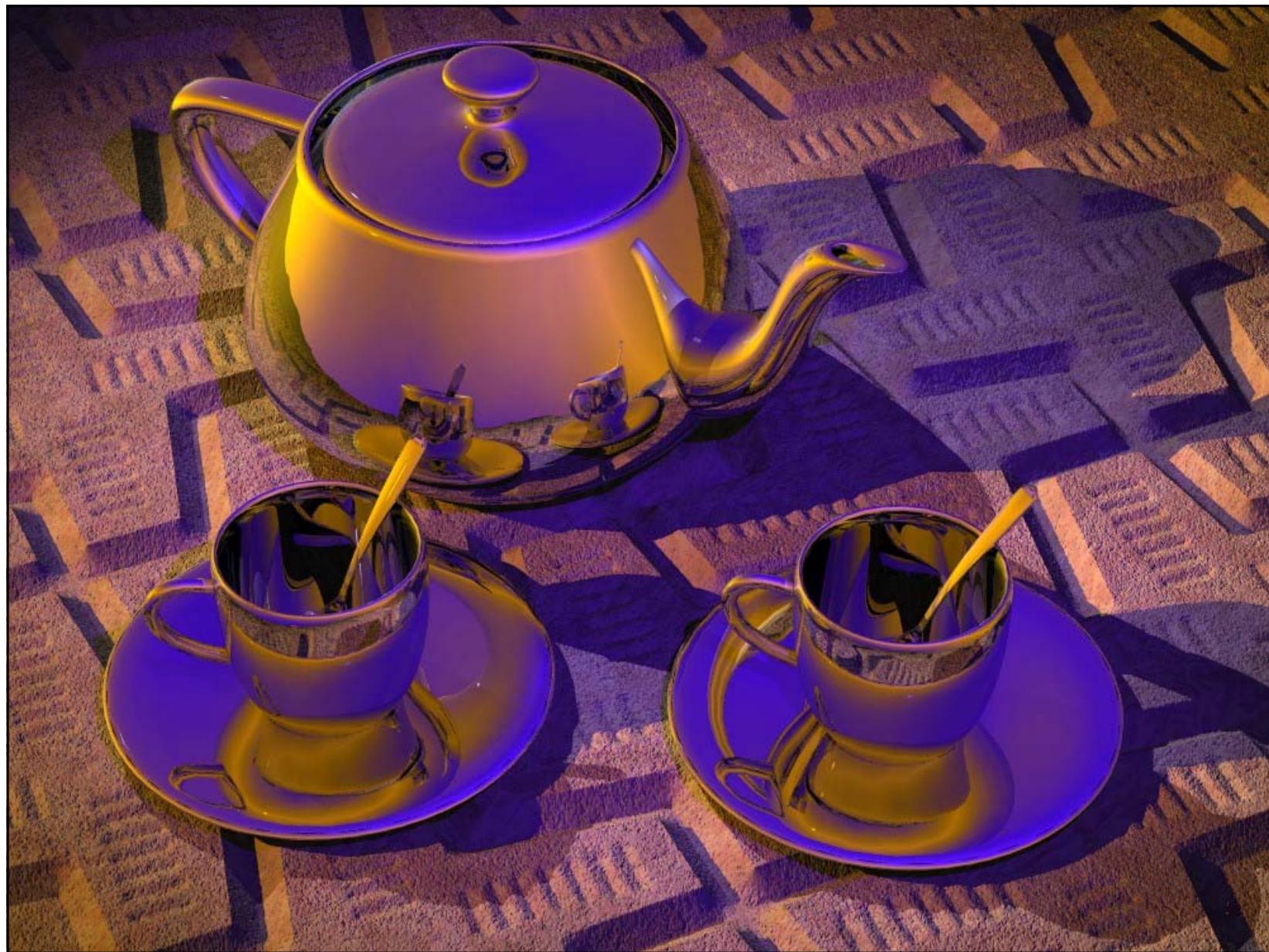


# *Interactive Computer Graphics: Lecture 10*

Ray tracing



2

2

© Steve Anger 1993

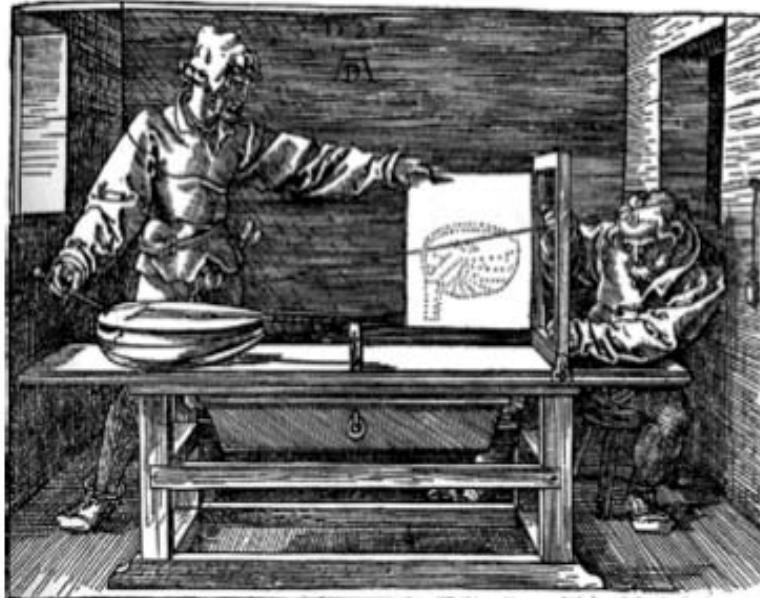


## *Direct and Global Illumination*

- **Direct illumination:** A surface point receives light directly from all light sources in the scene.
  - Computed by the direct illumination model.
- **Global illumination:** A surface point receives light after the light rays interact with other objects in the scene.
  - Points may be in shadow.
  - Rays may refract through transparent material.
  - Computed by reflection and transmission rays.

# *Albrecht Dürer's Ray Casting Machine*

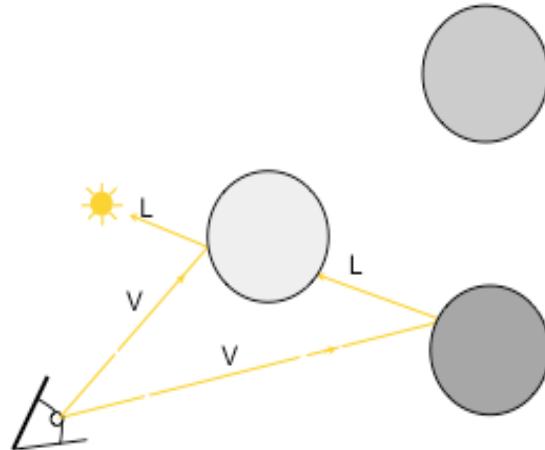
- Albrecht Dürer, 16th century



Graphics

## *Arthur Appel, 1968*

- On calculating the illusion of reality, 1968
- Cast one ray per pixel (ray casting).
  - For each intersection, trace one ray to the light to check for shadows
  - Only a local illumination model
- Developed for pen-plotters



# *Ray casting*

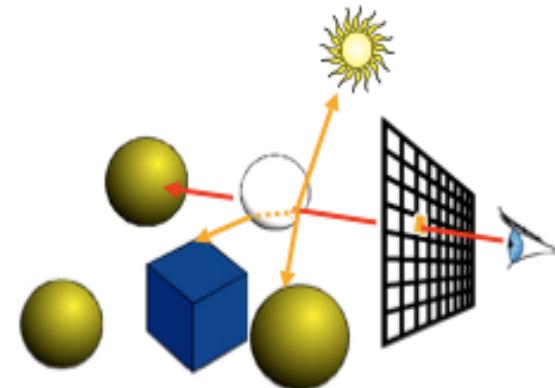
cast ray

    Intersect all objects

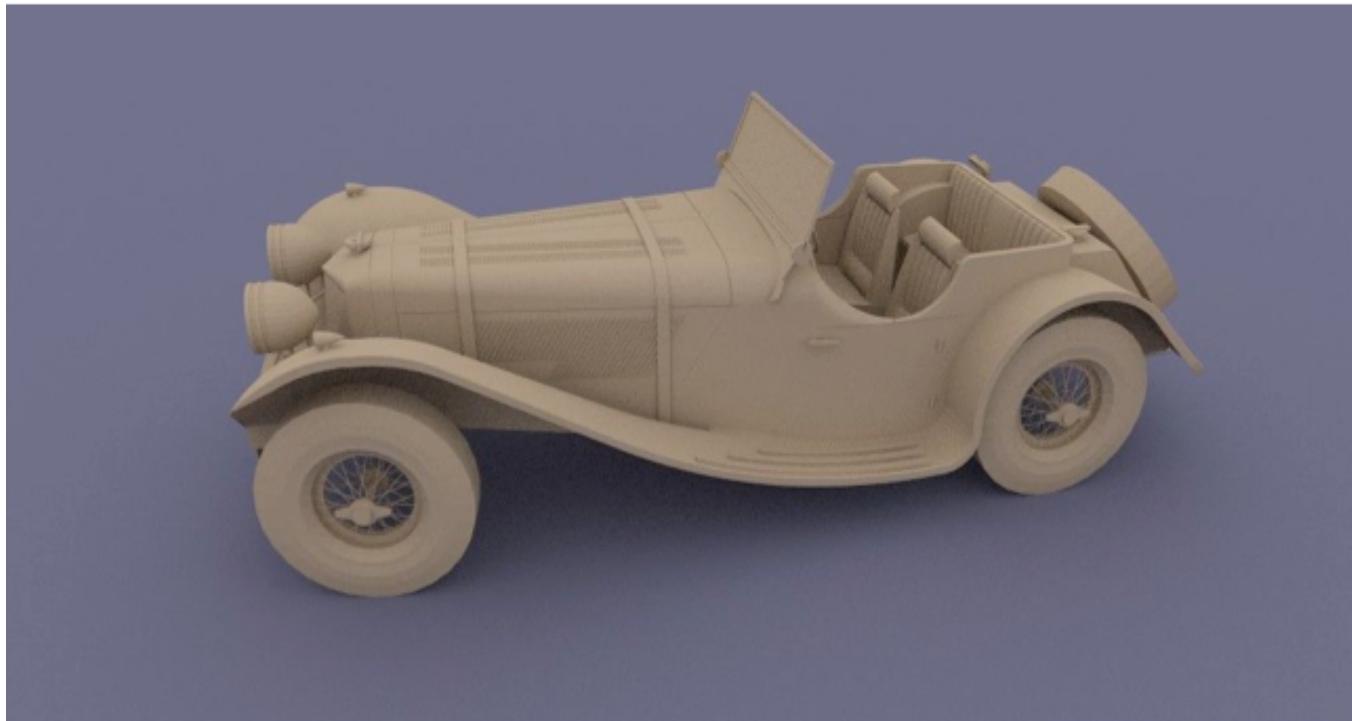
    color = ambient term

    For every light cast shadow ray

        col += local shading term



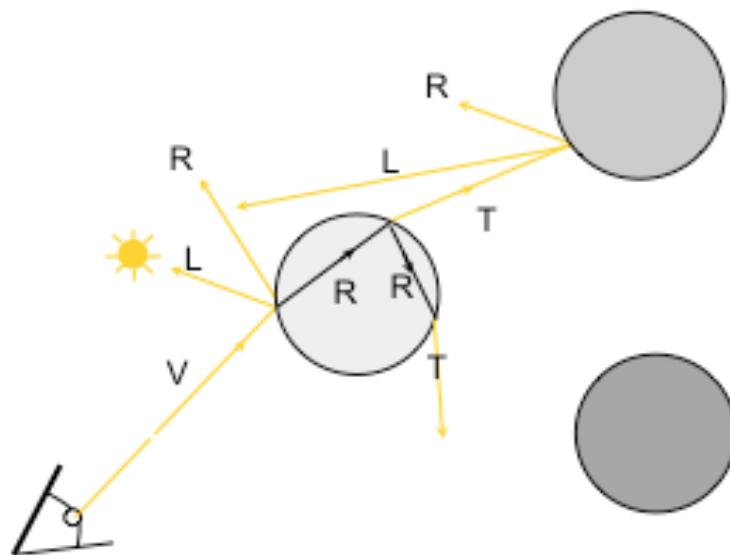
## *Ray casting*



Graphics Lecture 10: Slide 8

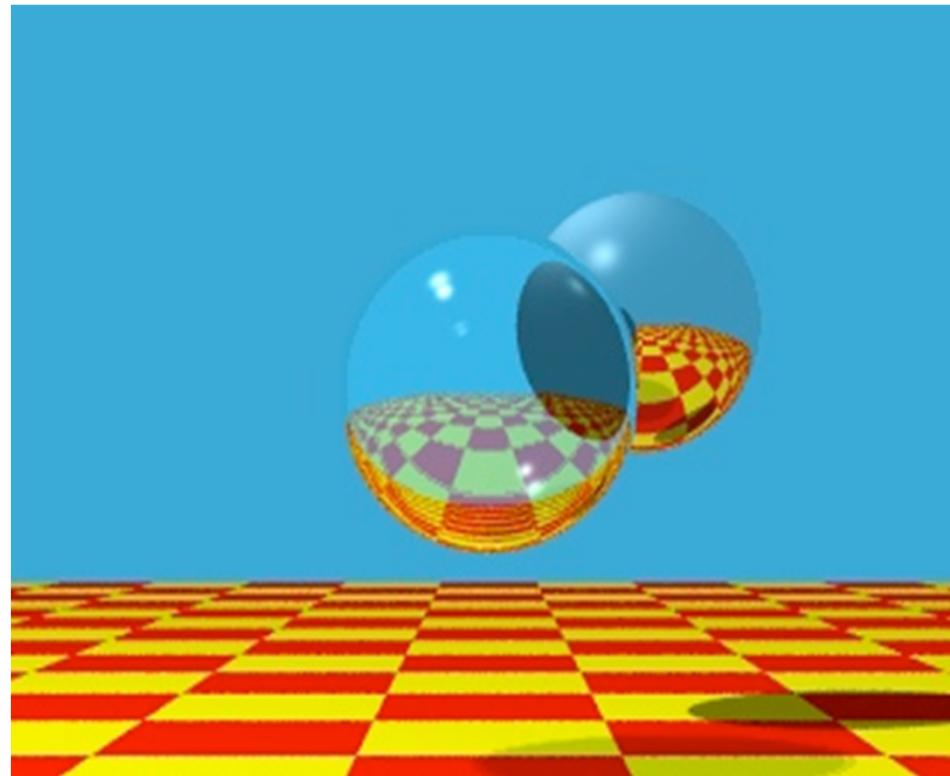
## *Turner Whitted, 1980*

- An Improved Illumination Model for Shaded Display, 1980
- First global illumination model:
  - An object's color is influenced by lights and other objects in the scene
  - Simulates specular reflection and refractive transmission



Graphics Lecture 10: Slide 9

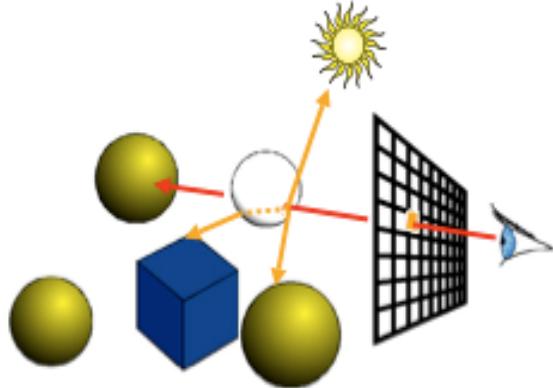
*Turner Whitted, 1980*



Graphics Lecture 10: Slide 10

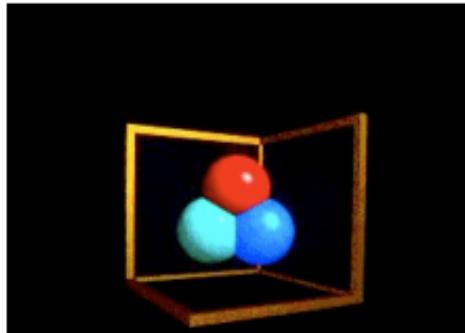
## *Recursive ray casting*

```
trace ray
    Intersect all objects
    color = ambient term
    For every light
        cast shadow ray
        col += local shading term
    If mirror
        col += k_refl * trace reflected ray
    If transparent
        col += k_trans * trace transmitted ray
```



## *Does it ever end?*

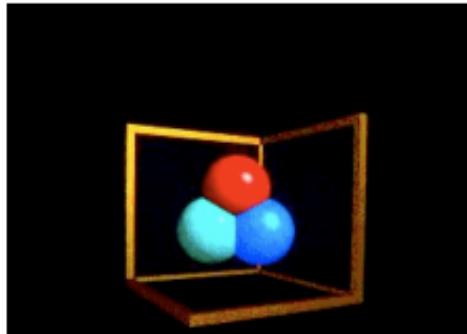
- Stopping criteria:
  - Recursion depth: Stop after a number of bounces
  - Ray contribution: Stop if reflected / transmitted contribution becomes too small



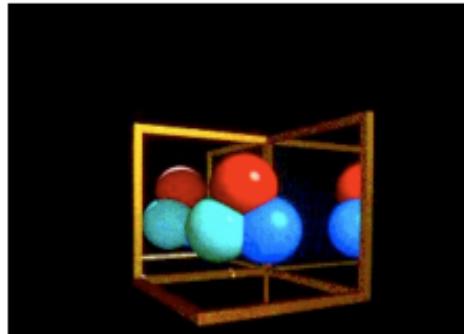
0 recursion

## *Does it ever end?*

- Stopping criteria:
  - Recursion depth: Stop after a number of bounces
  - Ray contribution: Stop if reflected / transmitted contribution becomes too small



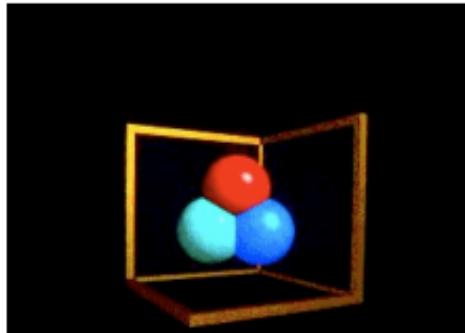
0 recursion



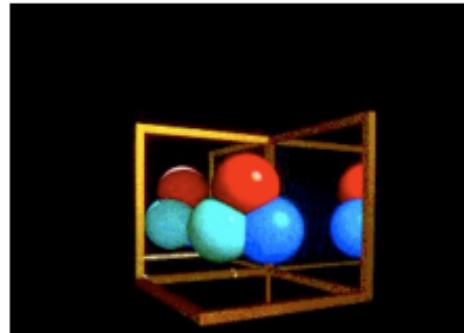
1 recursion

## *Does it ever end?*

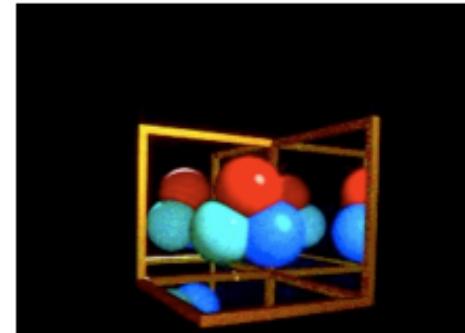
- Stopping criteria:
  - Recursion depth: Stop after a number of bounces
  - Ray contribution: Stop if reflected / transmitted contribution becomes too small



0 recursion



1 recursion



2 recursions

## *Ray tracing: Primary rays*

- For each ray we need to test which objects are intersecting the ray:
  - If the object has an intersection with the ray we calculate the distance between viewpoint and intersection
  - If the ray has more than one intersection, the smallest distance identifies the visible surface.
- Primary rays are rays from the view point to the nearest intersection point
- Local illumination is computed as before:

$$L = k_a + (k_d(\mathbf{n} \cdot \mathbf{l}) + k_s(\mathbf{v} \cdot \mathbf{r})^q)I_s$$

Graphics Lecture 10: Slide 15

## *Ray tracing: Secondary rays*

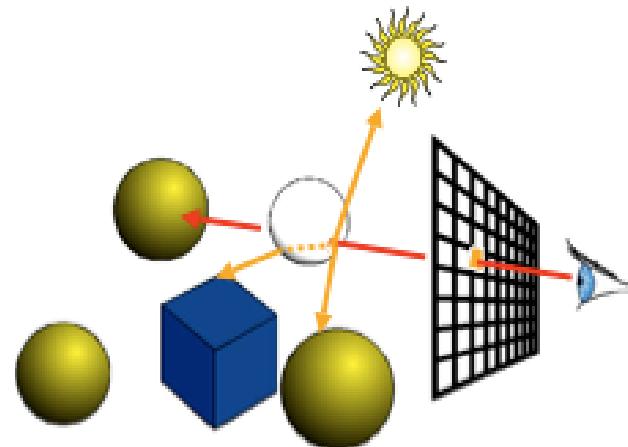
- Secondary rays are rays originating at the intersection points
- Secondary rays are caused by
  - rays reflected off the intersection point in the direction of reflection
  - rays transmitted through transparent materials in the direction of refraction
  - shadow rays

Graphics Lecture 10: Slide 16

## *Recursive ray tracing: Putting it all together*

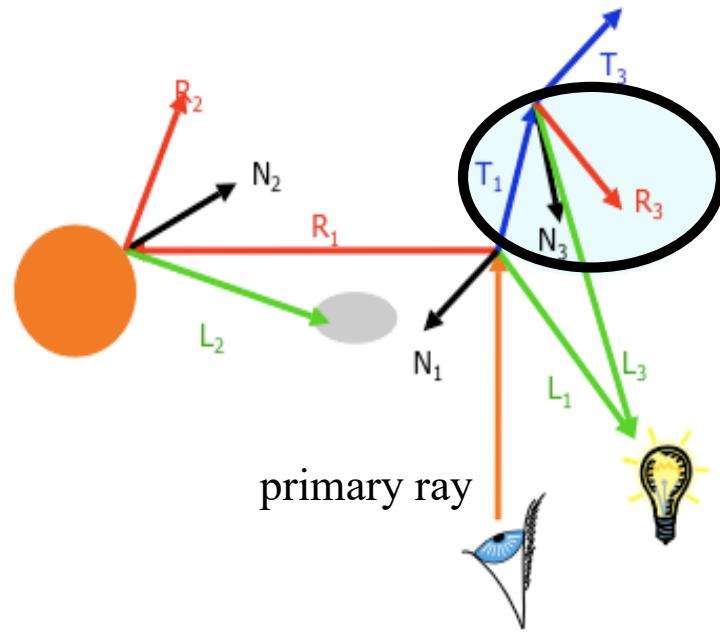
- Illumination can be expressed as

$$L = k_a + (k_d(\mathbf{n} \cdot \mathbf{l}) + k_s(\mathbf{v} \cdot \mathbf{r})^q)I_s + k_{reflected}L_{reflected} + k_{refracted}L_{refracted}$$

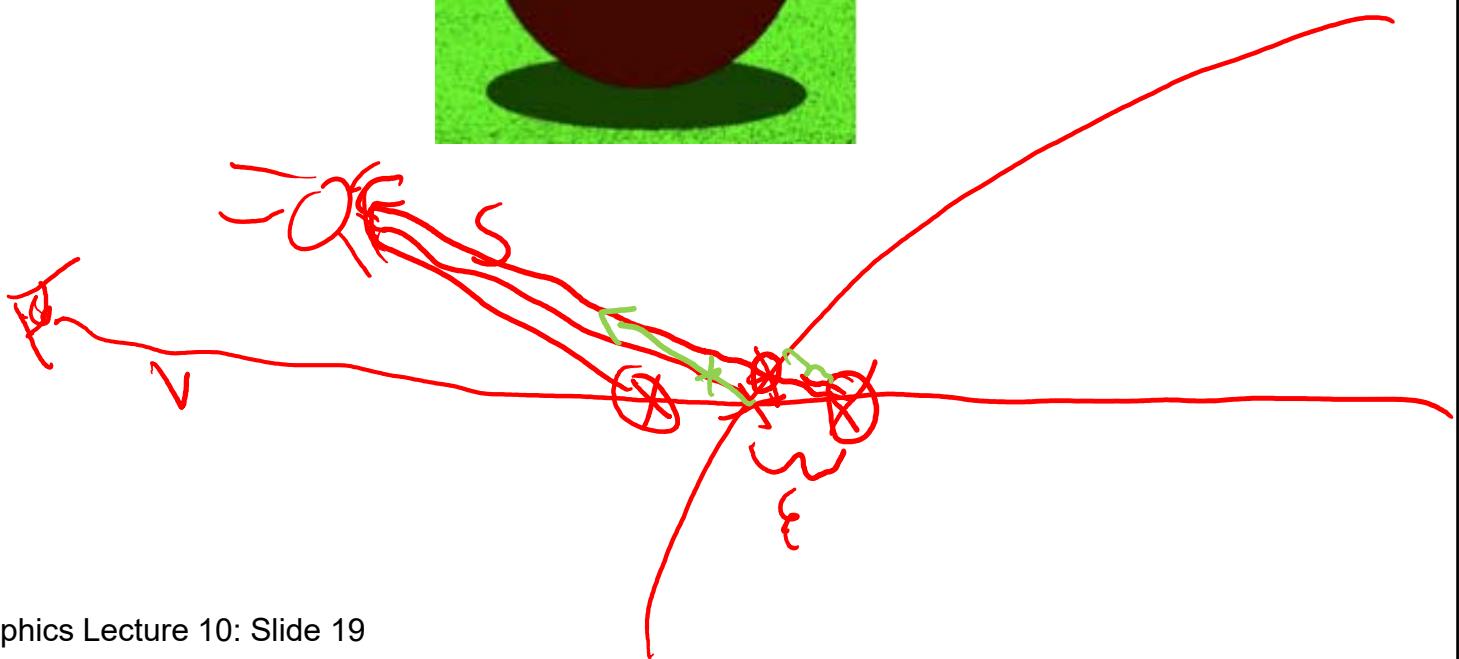
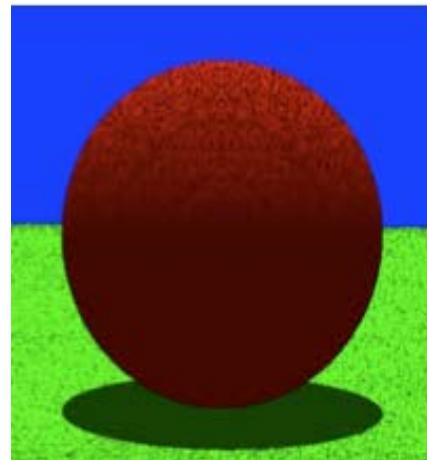


Graphics Lecture 10: Slide 17

## *Recursive Ray Tracing: Ray Tree*

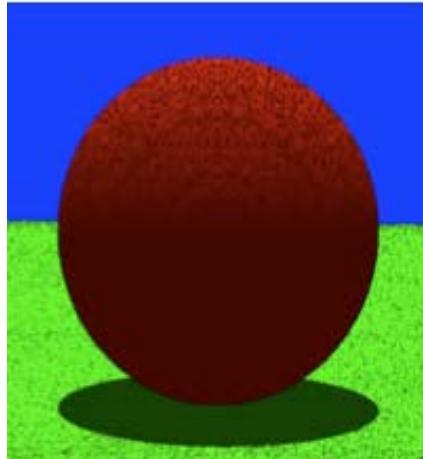


## Precision Problems



Graphics Lecture 10: Slide 19

## *Precision Problems*



- In ray tracing, the origin of (secondary) rays is often below the surface of objects
  - Theoretically, the intersection point should be on the surface
  - Practically, calculation imprecision creeps in, and the origin of the new ray is slightly beneath the surface
- Result: the surface area is shadowing itself or the ray continues on the wrong side

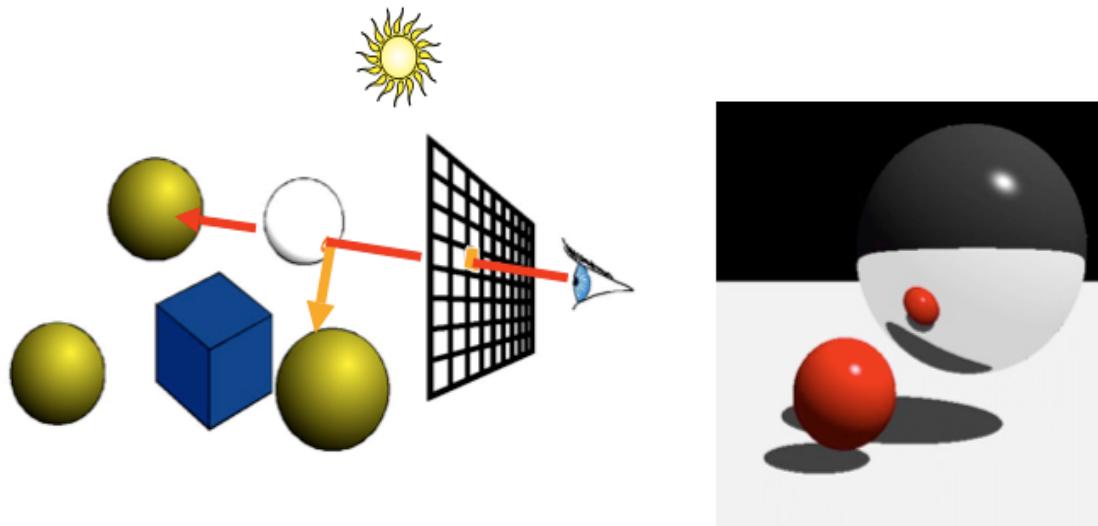
Graphics Lecture 10: Slide 20

## $\varepsilon$ to the rescue ...

- Check if  $t$  is within some epsilon tolerance:
  - if  $\text{abs}(\mu) < \varepsilon$ 
    - point is on the surface
  - else
    - point is inside/outside
  - Choose the  $\varepsilon$  tolerance empirically
- Move the intersection point by epsilon along the surface normal so it is outside of the object
- Check if point is inside/outside surface by checking the sign of the implicit (sphere etc.) equation

## *Mirror reflection*

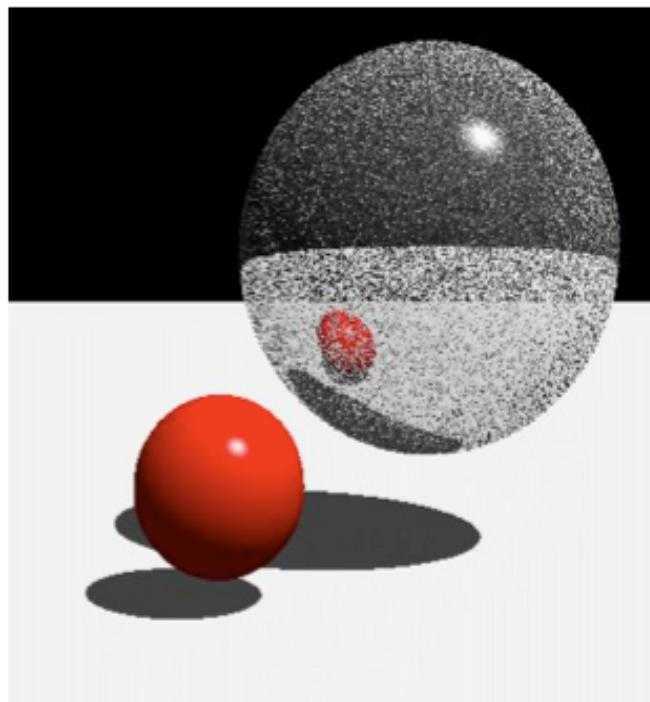
- Compute mirror contribution
- Cast ray in direction symmetric wrt. normal
- Multiply by reflection coefficient (color)



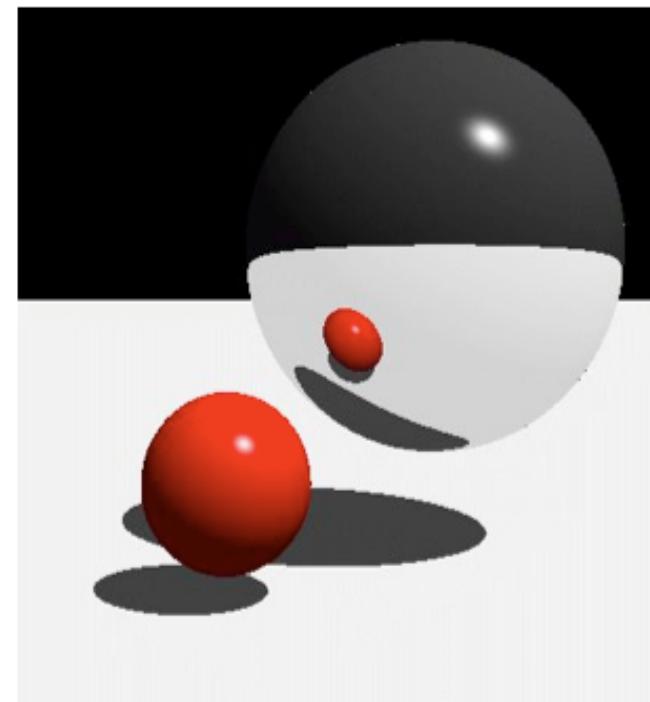
Graphics Lecture 10: Slide 22

## *Mirror reflection*

- Don't forget to add epsilon to the ray



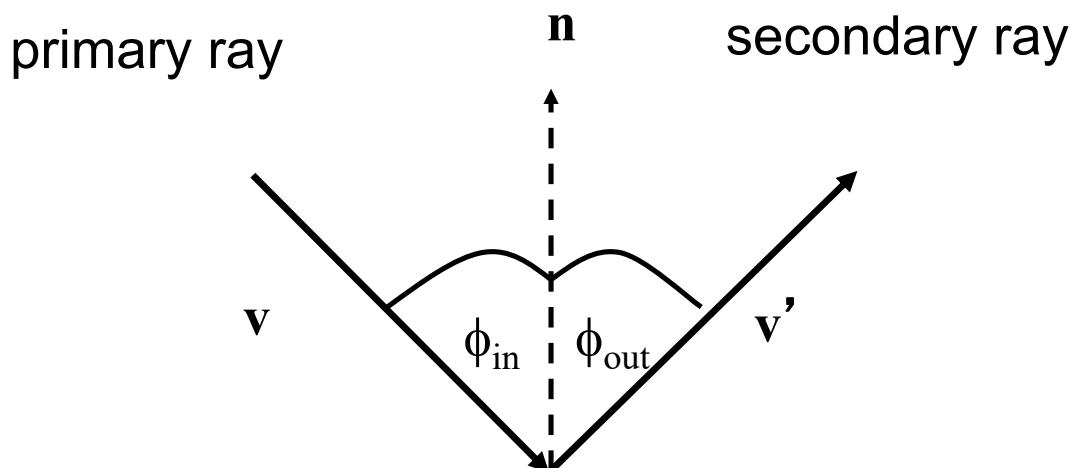
Without epsilon



With epsilon

Graphics Lecture 10: Slide 23

## *Mirror reflection*

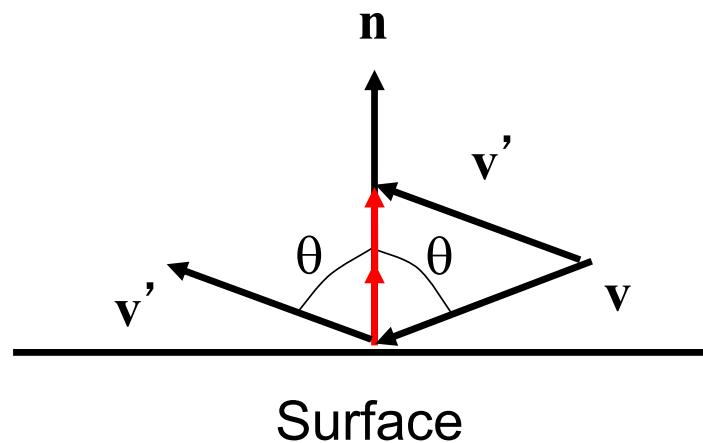


## *Mirror reflection*

- To calculate illumination as a result of reflections
    - calculate the direction of the secondary ray at the intersection of the primary ray with the object.
- given that
- $\mathbf{n}$  is the unit surface normal
  - $\mathbf{v}$  is the direction of the primary ray
  - $\mathbf{v}'$  is the direction of the secondary ray as a result of reflections

$$\mathbf{v}' = \mathbf{v} - (2\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$

## *Mirror reflection*

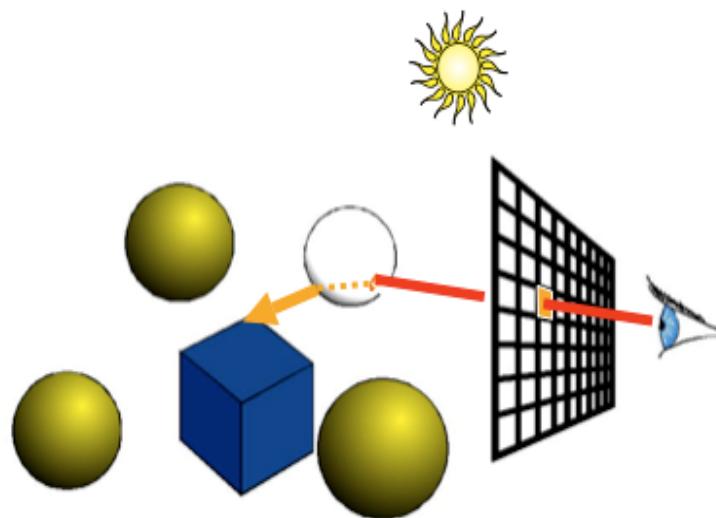


$$\mathbf{v}' = \mathbf{v} - (2\mathbf{v} \cdot \mathbf{n})\mathbf{n}$$

Graphics Lecture 10: Slide 26

## *Transparency*

- Compute transmitted contribution
- Cast ray in refracted direction
- Multiply by transparency coefficient



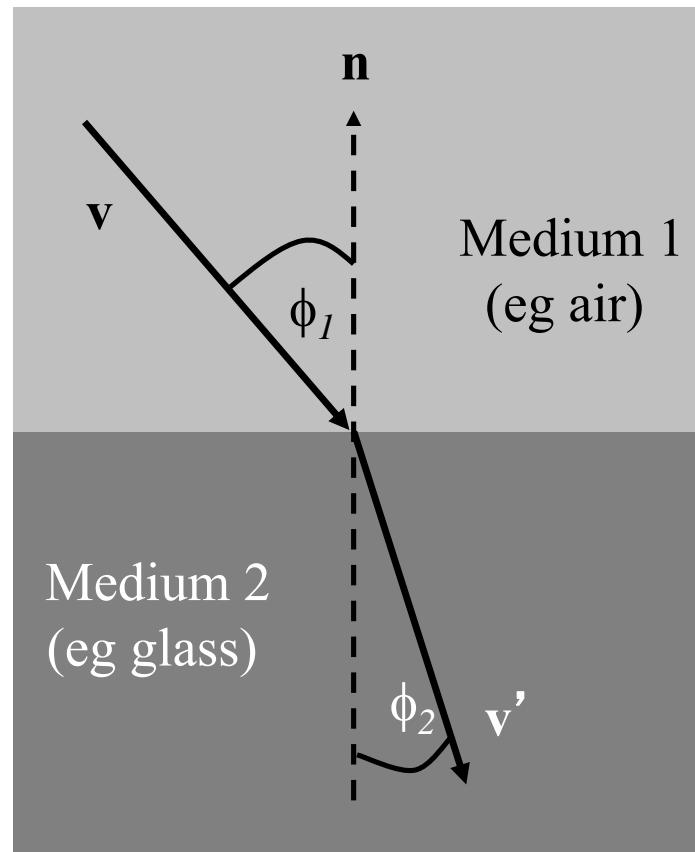
Graphics Lecture 10: Slide 28

## Refraction

- The angle of the refracted ray can be determined by Snell's law:

$$\eta_1 \sin(\phi_1) = \eta_2 \sin(\phi_2)$$

- $\eta_1$  is a constant for medium 1
- $\eta_2$  is a constant for medium 2
- $\phi_1$  is the angle between the incident ray and the surface normal
- $\phi_2$  is the angle between the refracted ray and the surface normal



## Refraction

- In vector notation Snell's law can be written:

$$k_1(v \cdot n) = k_2(v' \cdot n)$$

- The direction of the refracted ray is

$$\mathbf{v}' = \frac{\eta_1}{\eta_2} \left( \sqrt{\left(\mathbf{n} \cdot \mathbf{v}\right)^2 + \left(\frac{\eta_2}{\eta_1}\right)^2} - 1 - \mathbf{n} \cdot \mathbf{v} \right) \cdot \mathbf{n} + \mathbf{v}$$

## Refraction

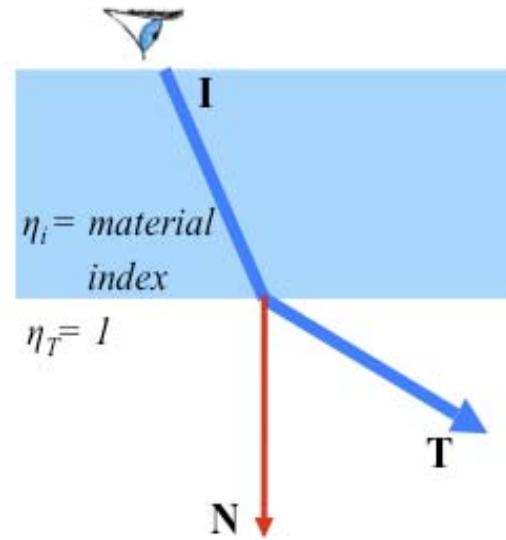
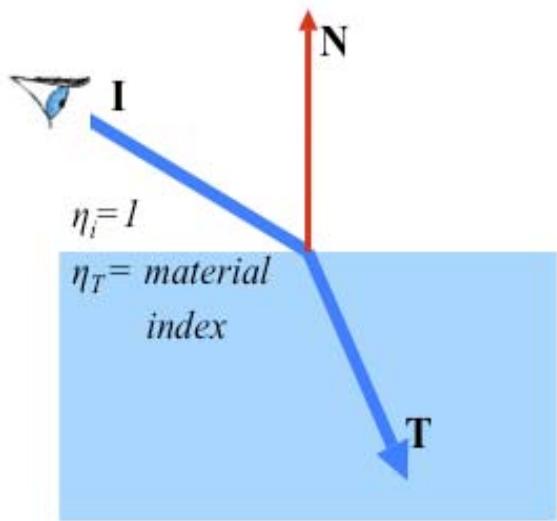
- This equation only has a solution if

$$(\mathbf{n} \cdot \mathbf{v})^2 > 1 - \left( \frac{\eta_2}{\eta_1} \right)^2$$

- This illustrates the physical phenomenon of the limiting angle:
  - if light passes from one medium to another medium whose index of refraction is low, the angle of the refracted ray is greater than the angle of the incident ray
  - if the angle of the incident ray is large, the angle of the refracted ray is larger than 90°
    - ➔ the ray is reflected rather than refracted

## Refraction

- Make sure you know whether you are entering or leaving the transmissive material

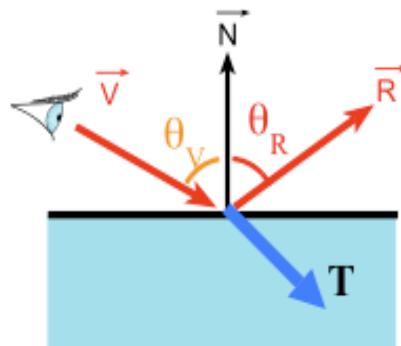


Graphics Lecture 10: Slide 32

## *Amount of reflection and refraction*

- Traditional (hacky) ray tracing
  - Constant coefficient reflection
  - Component per component multiplication
- Better: Mix reflected and refracted light according to the Fresnel factor.

$$L = k_{fresnel} L_{reflected} + (1 - k_{fresnel}) L_{refracted}$$



Graphics Lecture 10: Slide 33

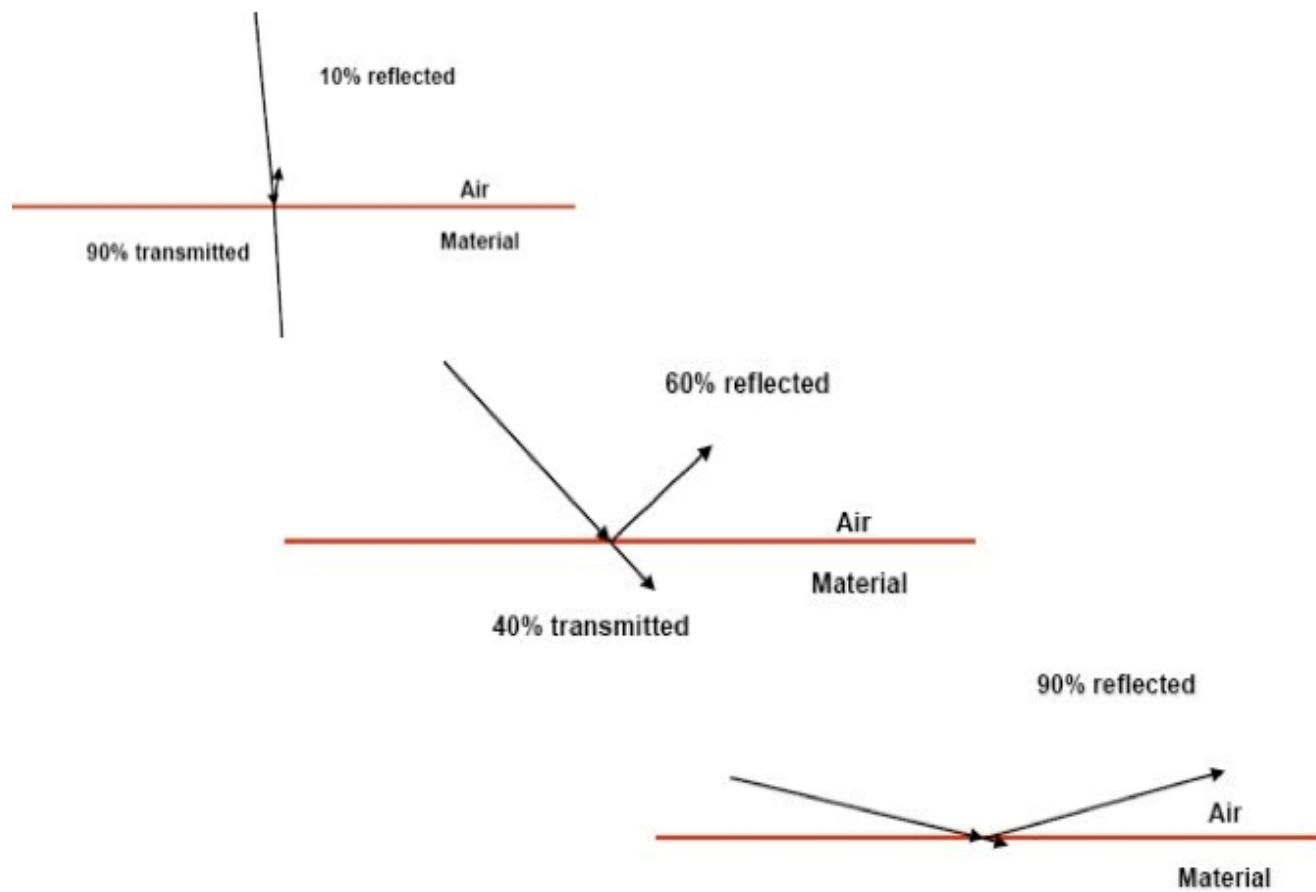
## *Fresnel factor*

- More reflection at grazing angle



Graphics Lecture 10: Slide 34

## Fresnel factor



Graphics Lecture 10: Slide 35

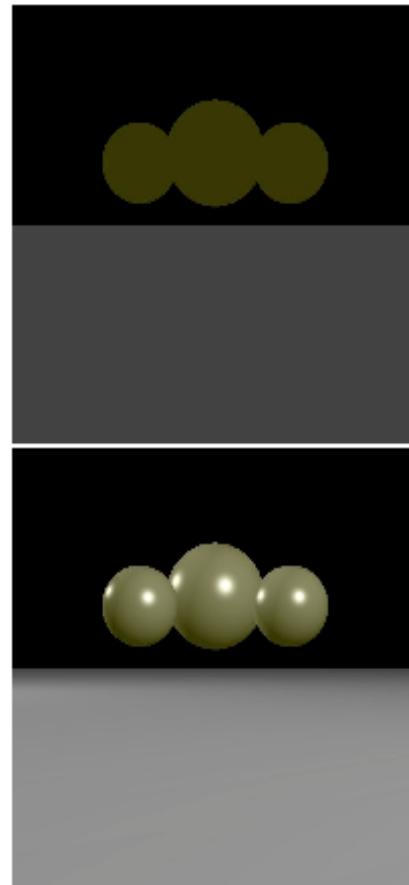
## *Schlick's Approximation*

- Schlick' s approximation

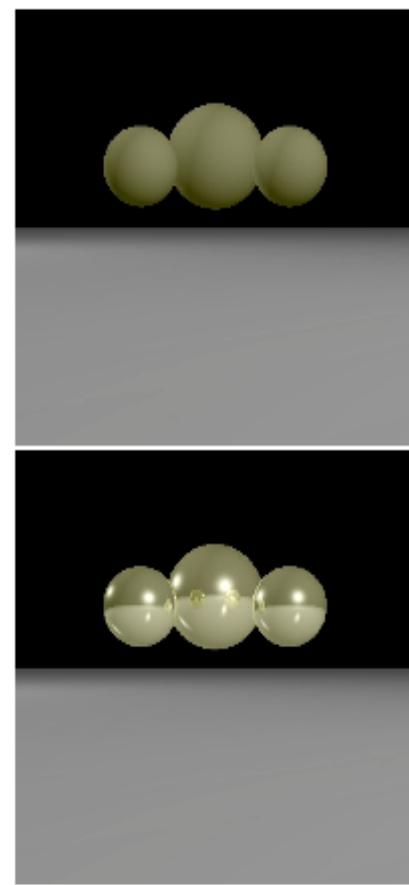
$$k_{fresnel}(\theta) = k_{fresnel}(0) + (1 - k_{fresnel}(0))(1 - (\mathbf{n} \cdot \mathbf{l}))^5$$

- $k_{fresnel}(0)$  = Fresnel factor at zero degrees
- Choose  $k_{fresnel}(0) = 0.8$ , this will look like stainless steel
- Fresnel factor at zero degrees has low value  $\sim 0.05$  for dielectric materials like plastic.

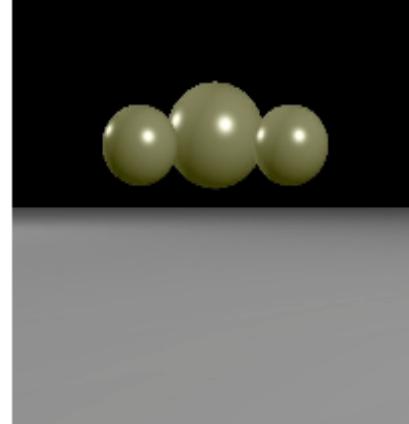
## *Example*



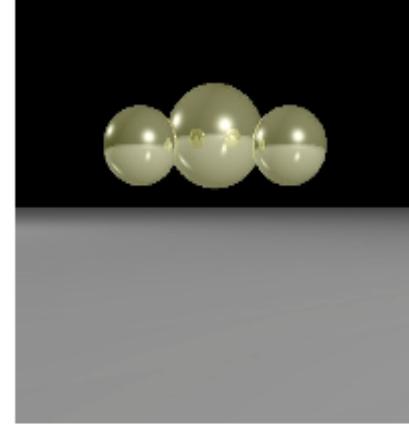
Ambient



+ Diffuse



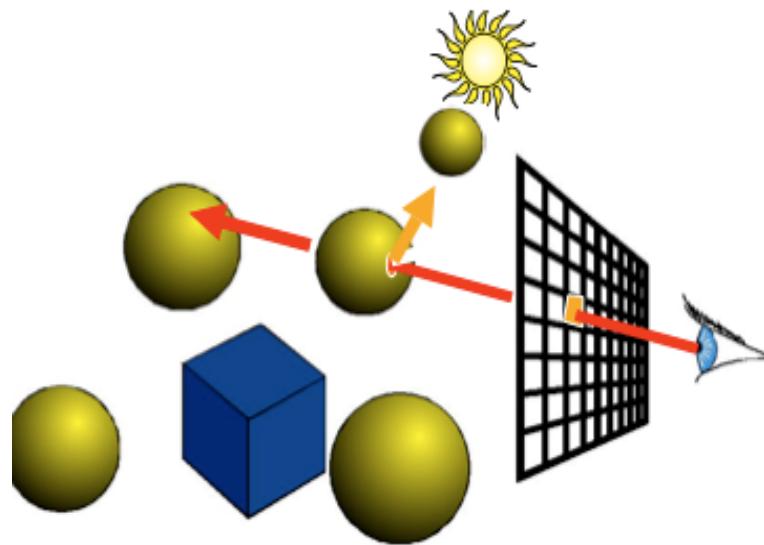
+ Specular



+  $I_{\text{reflected}}$

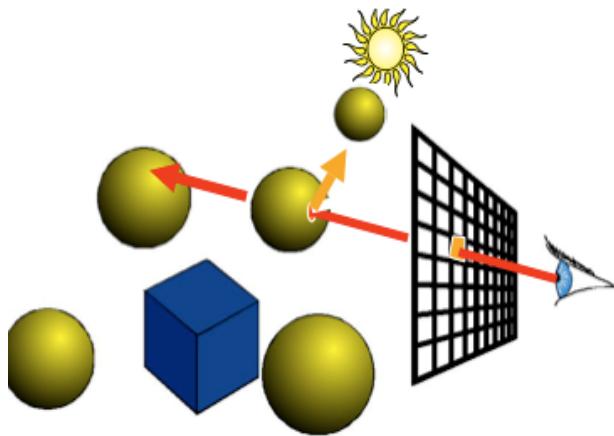
Graphics Lecture 10: Slide 37

## *How do we add shadows?*



Graphics Lecture 10: Slide 38

## How do we add shadows?

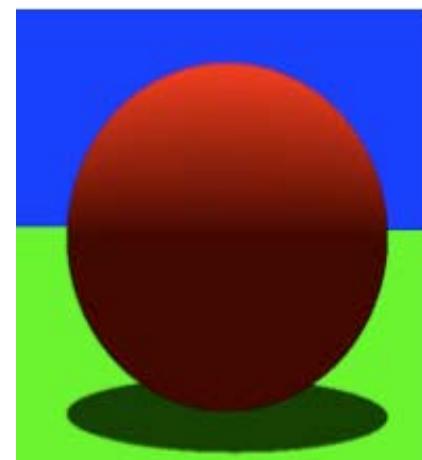
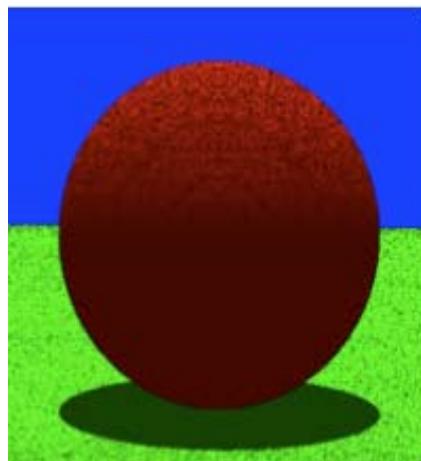


$$L = k_a + s(k_d(\mathbf{n} \cdot \mathbf{l}) + k_s(\mathbf{v} \cdot \mathbf{r})^q)I_s + k_{reflected}L_{reflected} + k_{refracted}L_{refracted}$$

$$s = \begin{cases} 0 & \text{if light source is obscured} \\ 1 & \text{if light source is not obscured} \end{cases}$$

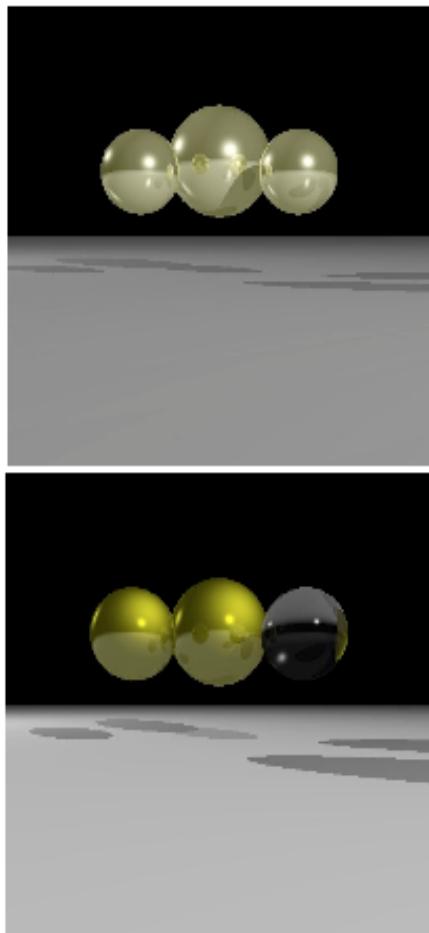
## *Shadows: Problems?*

- Make sure to avoid self-shadowing

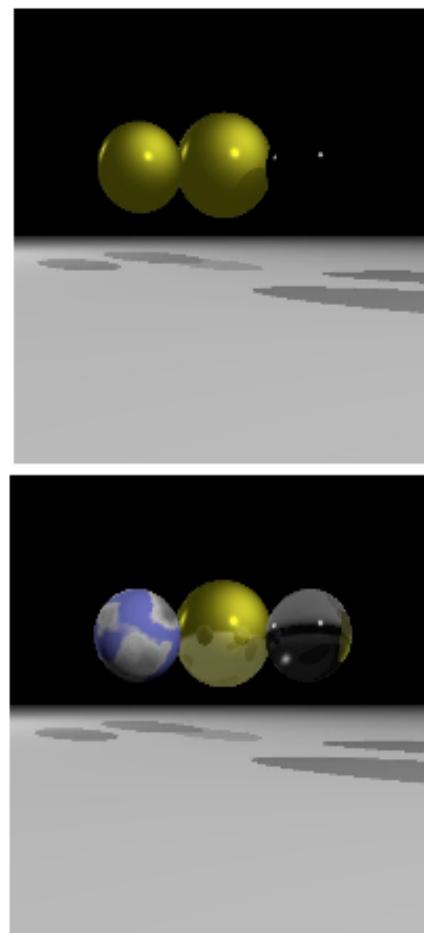


Graphics Lecture 10: Slide 40

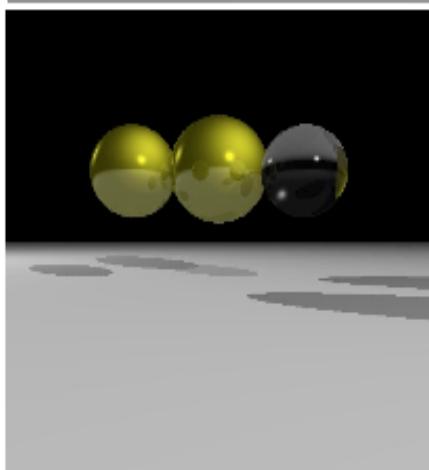
## *Example*



+ Shadows



-  $I_{\text{reflected}}$   
+ transmitted



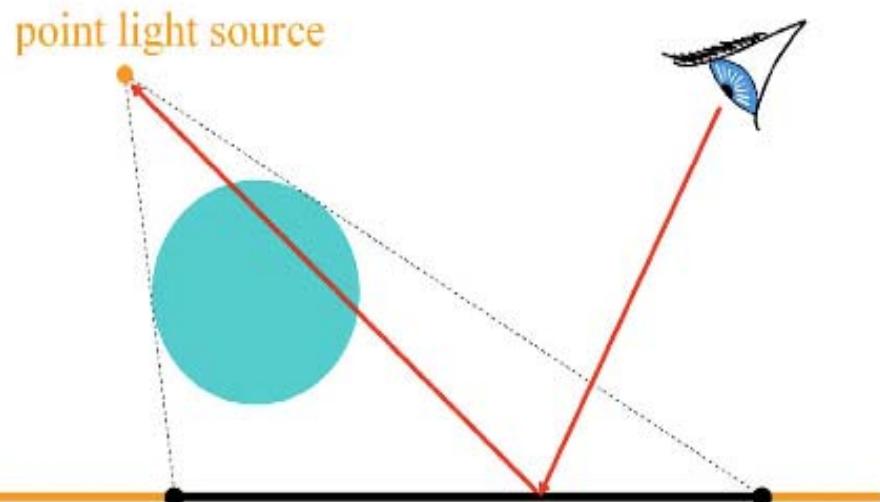
+  $I_{\text{reflected}}$

+ textures

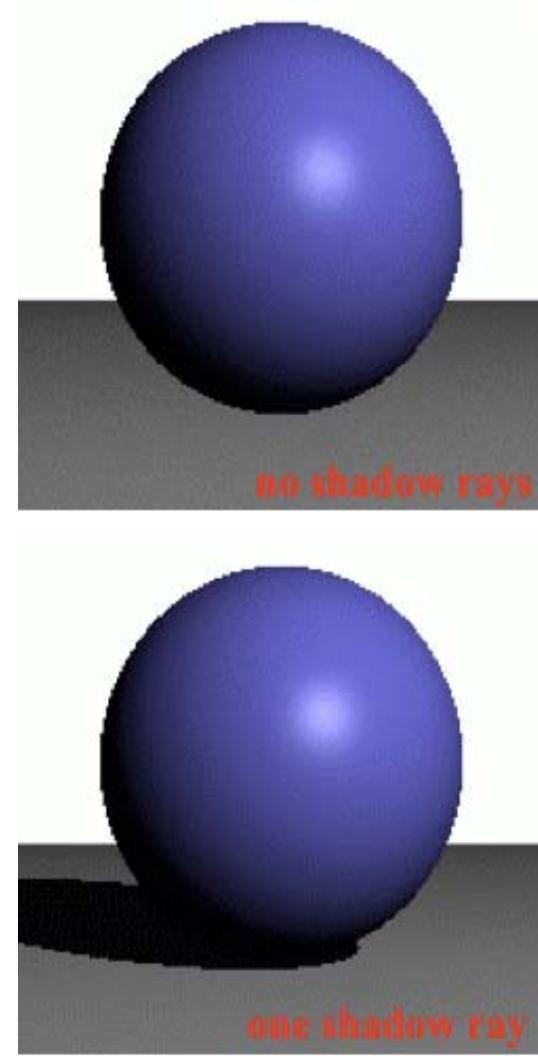
Graphics Lecture 10: Slide 41

## Shadows

- One shadow ray per intersection per point light source

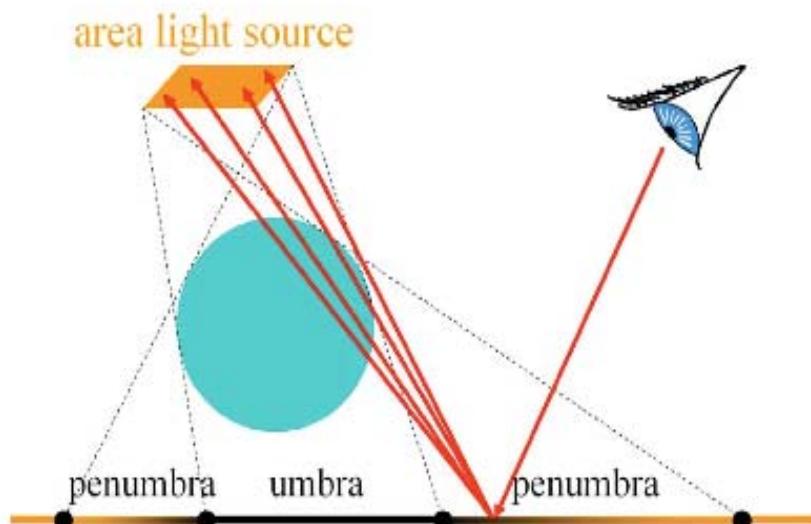


Graphics Lecture 10: Slide 42

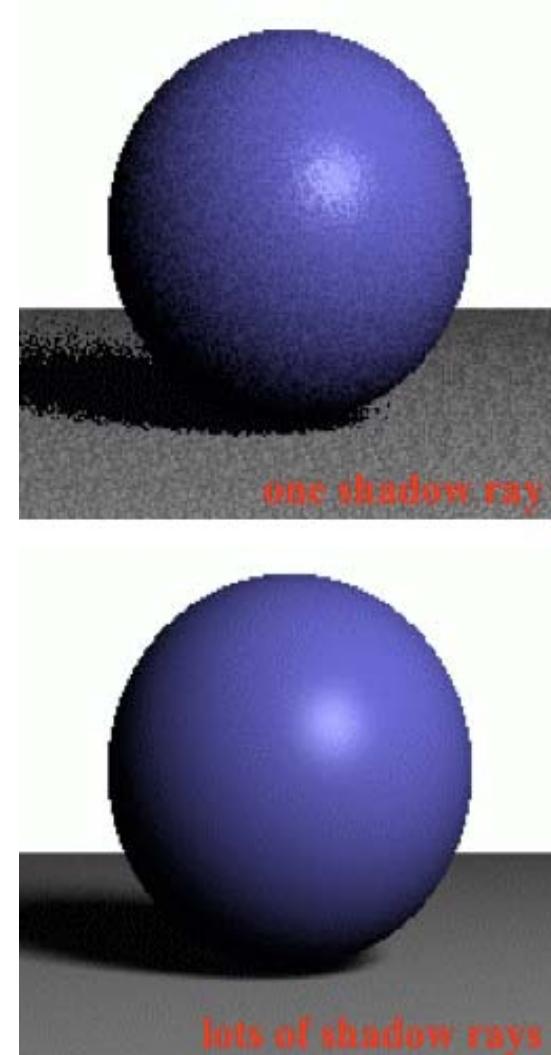


## Soft shadows

- Multiple shadow rays to sample area light source

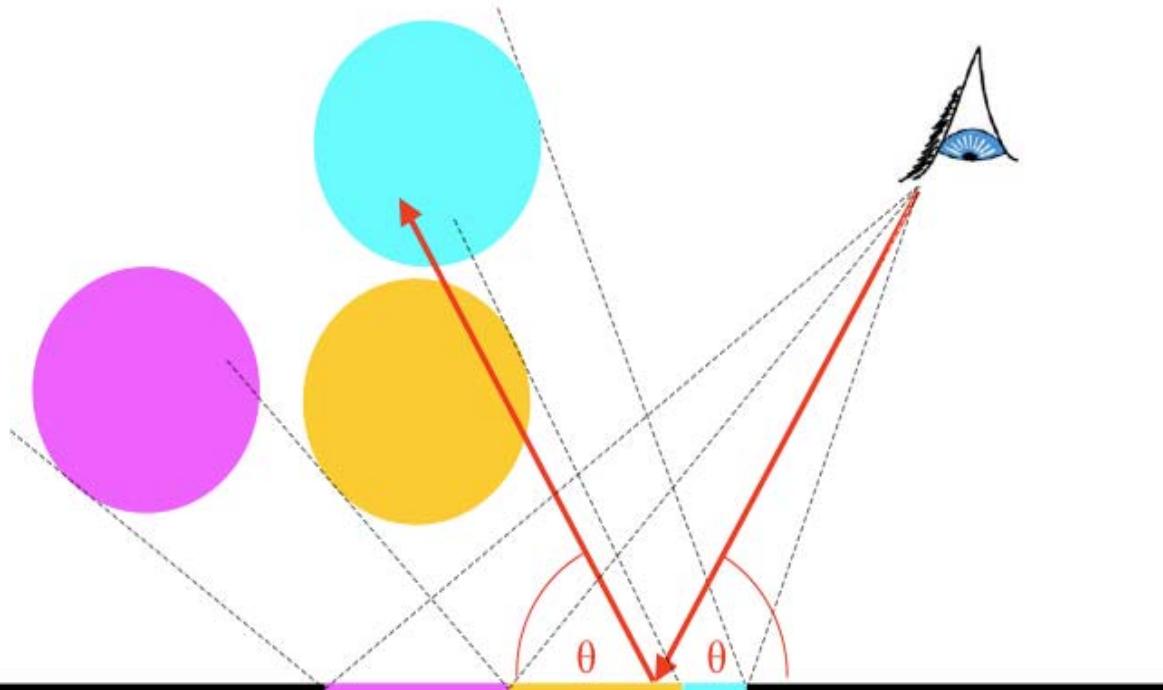


Graphics Lecture 10: Slide 43



## *Reflection: Conventional ray tracing*

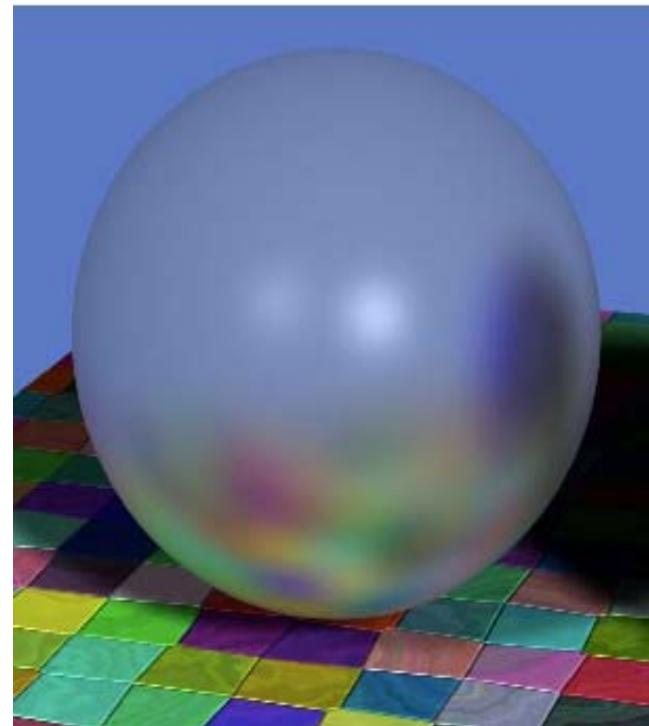
- One reflection per intersection



Graphics Lecture 10: Slide 44

## *Reflection: Conventional ray tracing*

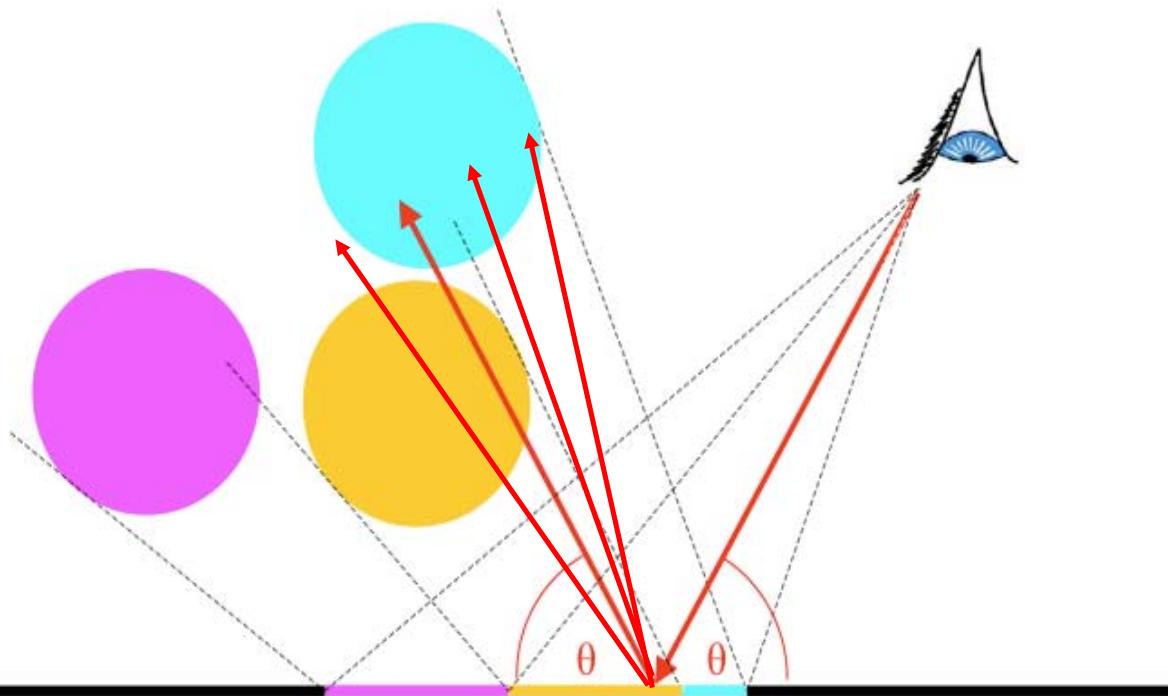
- How can we create effects like this?



Graphics Lecture 10: Slide 45

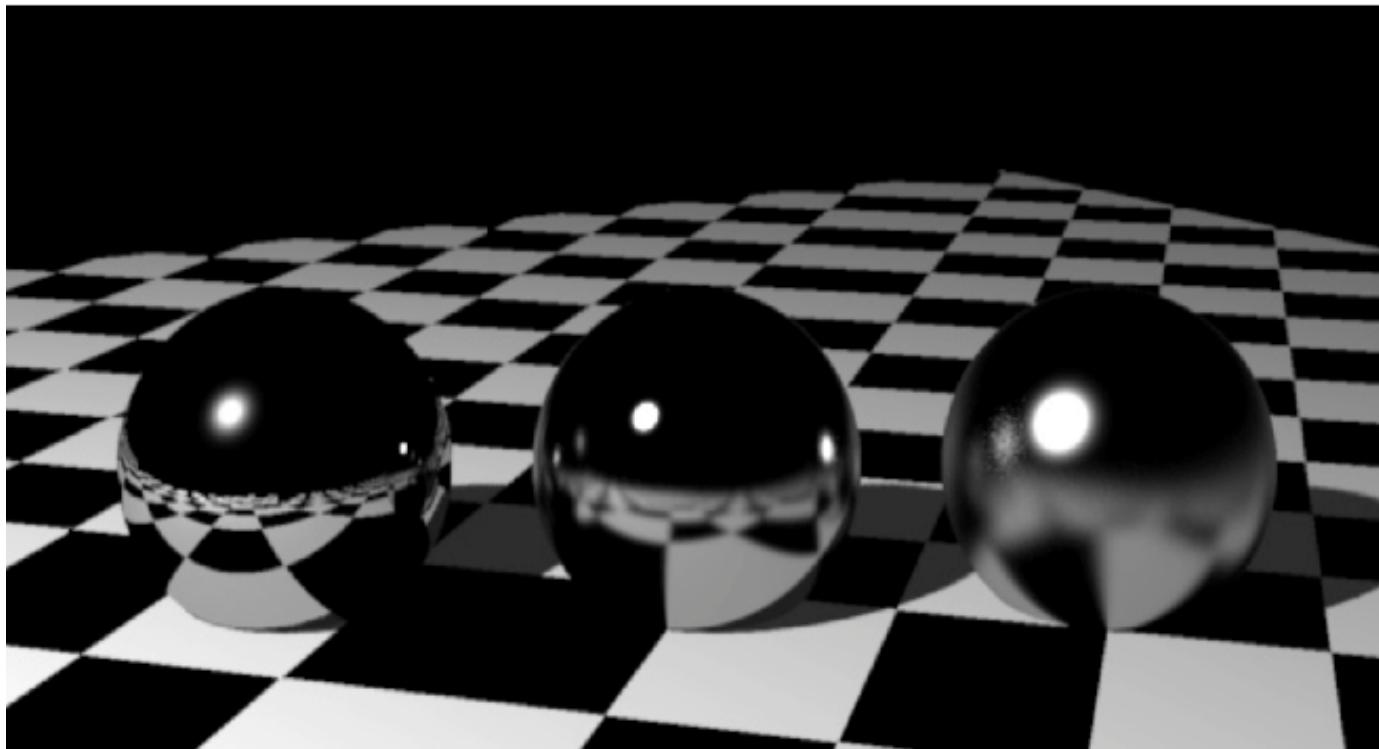
## *Reflection: Monte Carlo ray tracing*

- Random reflection rays around mirror direction



Graphics Lecture 10: Slide 46

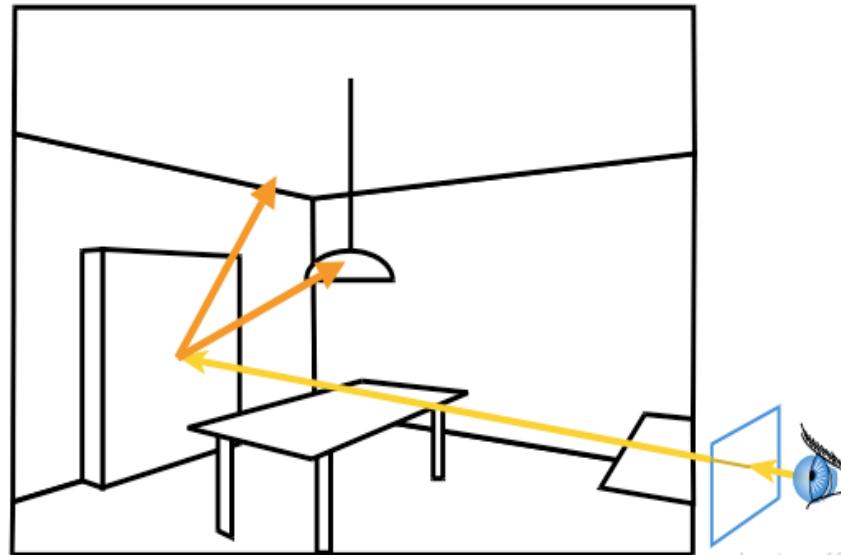
## *Glossy surfaces*



Graphics Lecture 10: Slide 47

## *Ray tracing*

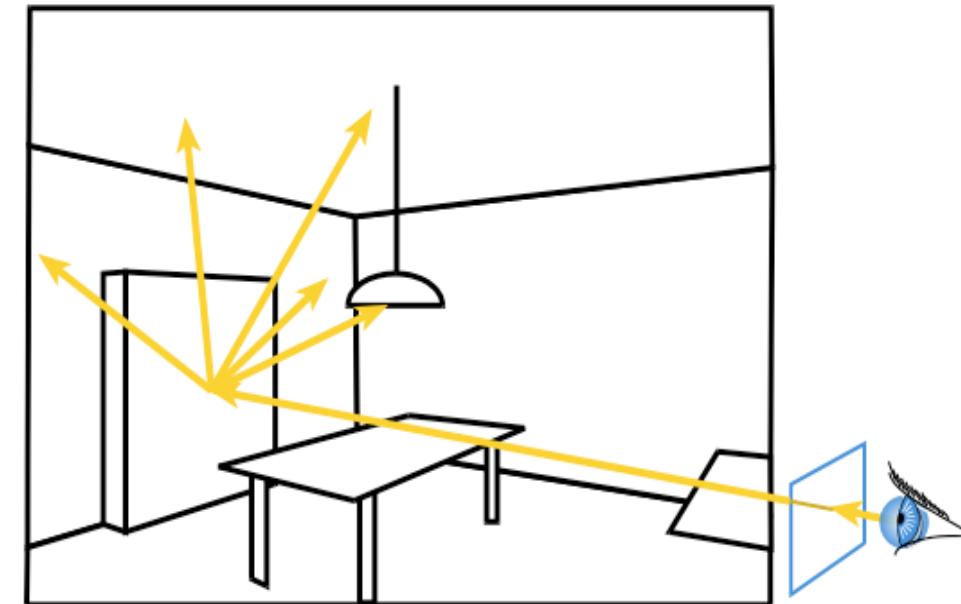
- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)



Graphics Lecture 10: Slide 48

## *Monte-Carlo Ray Tracing*

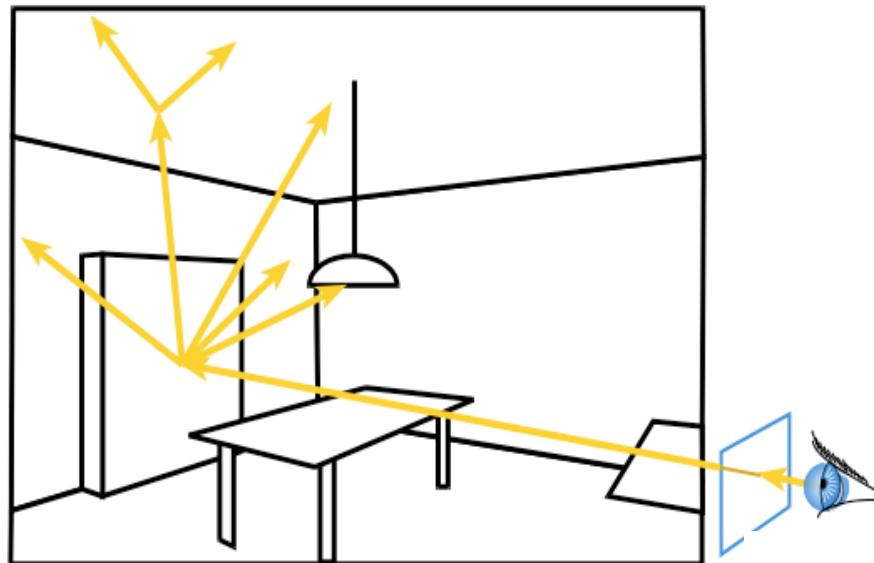
- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
  - Accumulate radiance contribution



Graphics Lecture 10: Slide 49

## *Monte-Carlo Ray Tracing*

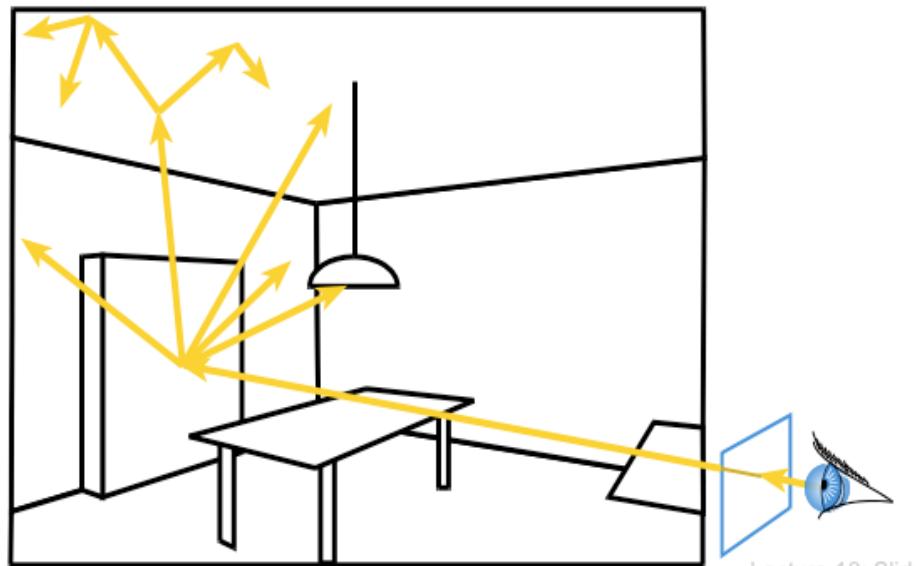
- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



Graphics Lecture 10: Slide 50

## *Monte-Carlo Ray Tracing*

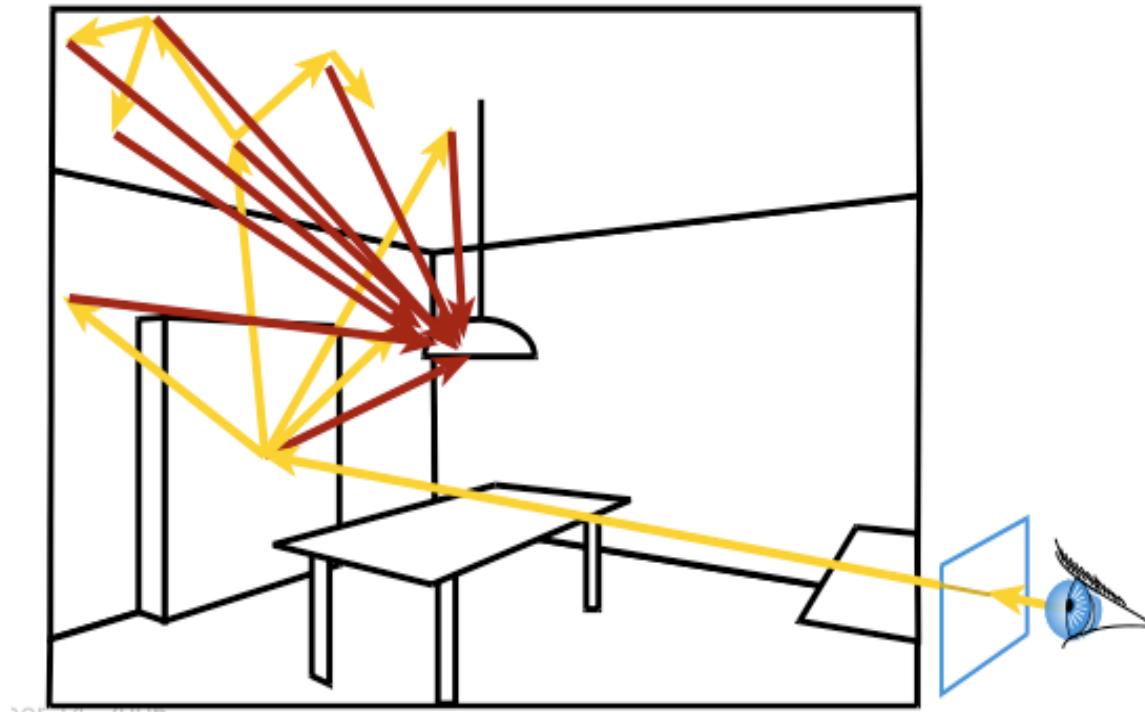
- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



Graphics Lecture 10: Slide 51

## *Monte-Carlo Ray Tracing*

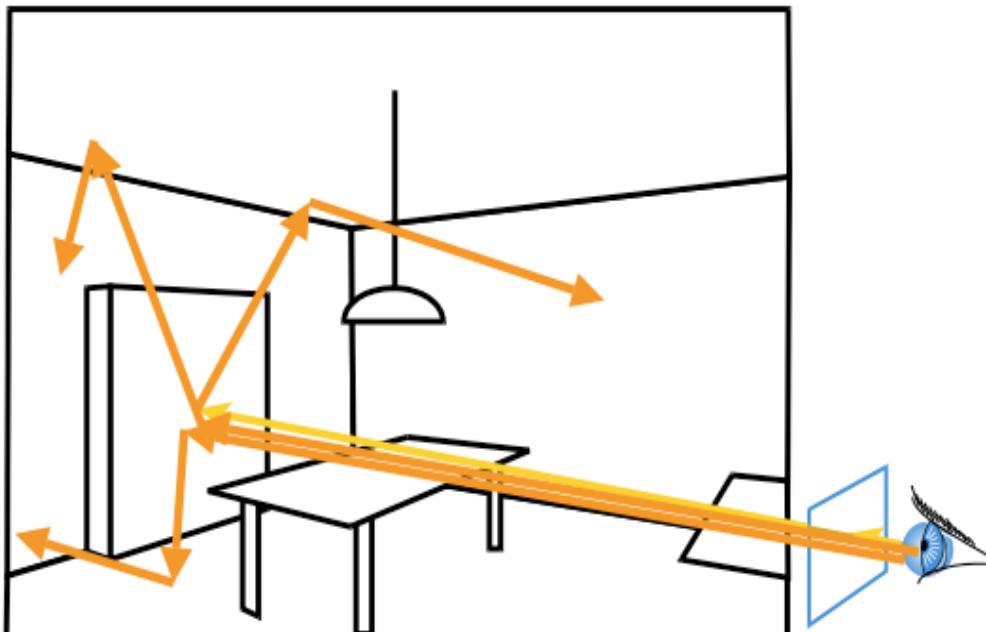
- Send rays to light



Graphics Lecture 10: Slide 52

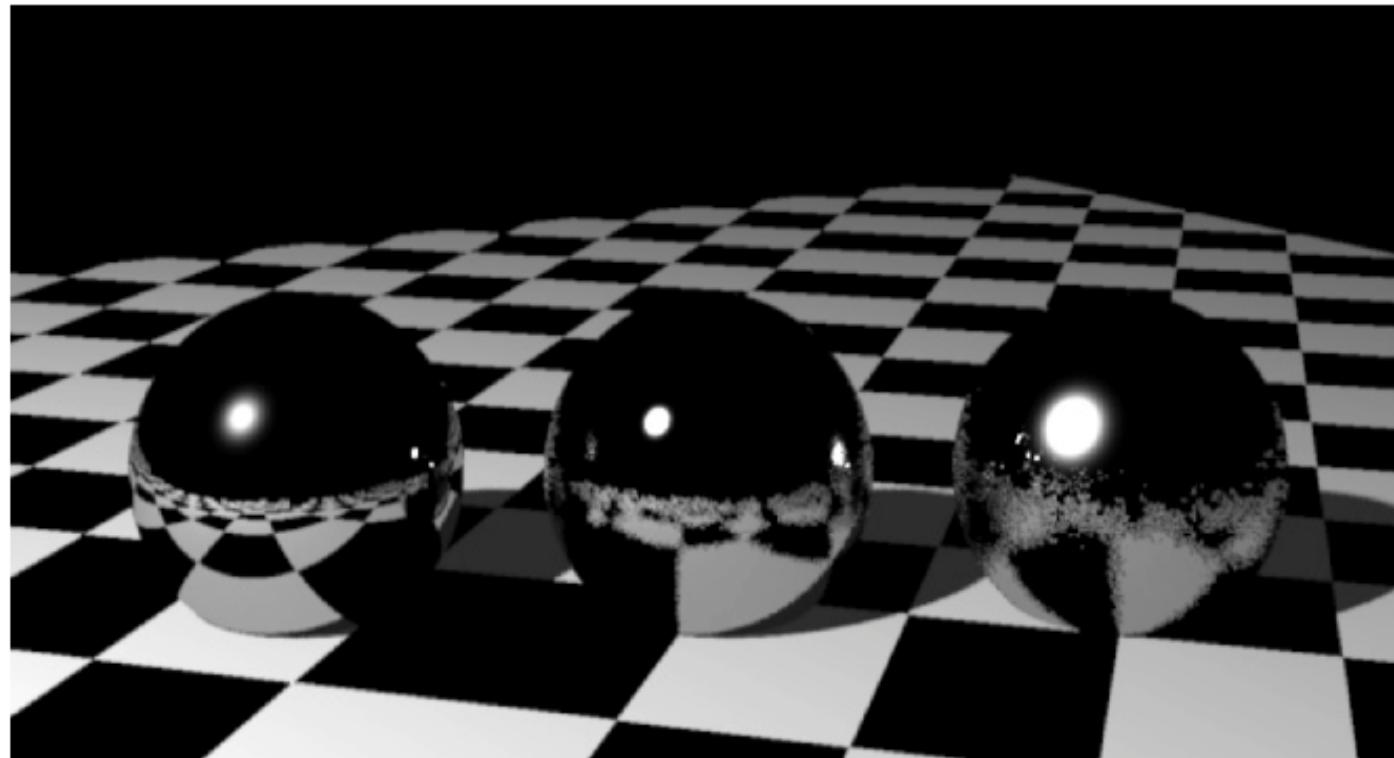
## *Monte-Carlo Path Tracing*

- Trace only one secondary ray per recursion
- But send many primary rays per pixel



Graphics Lecture 10: Slide 53

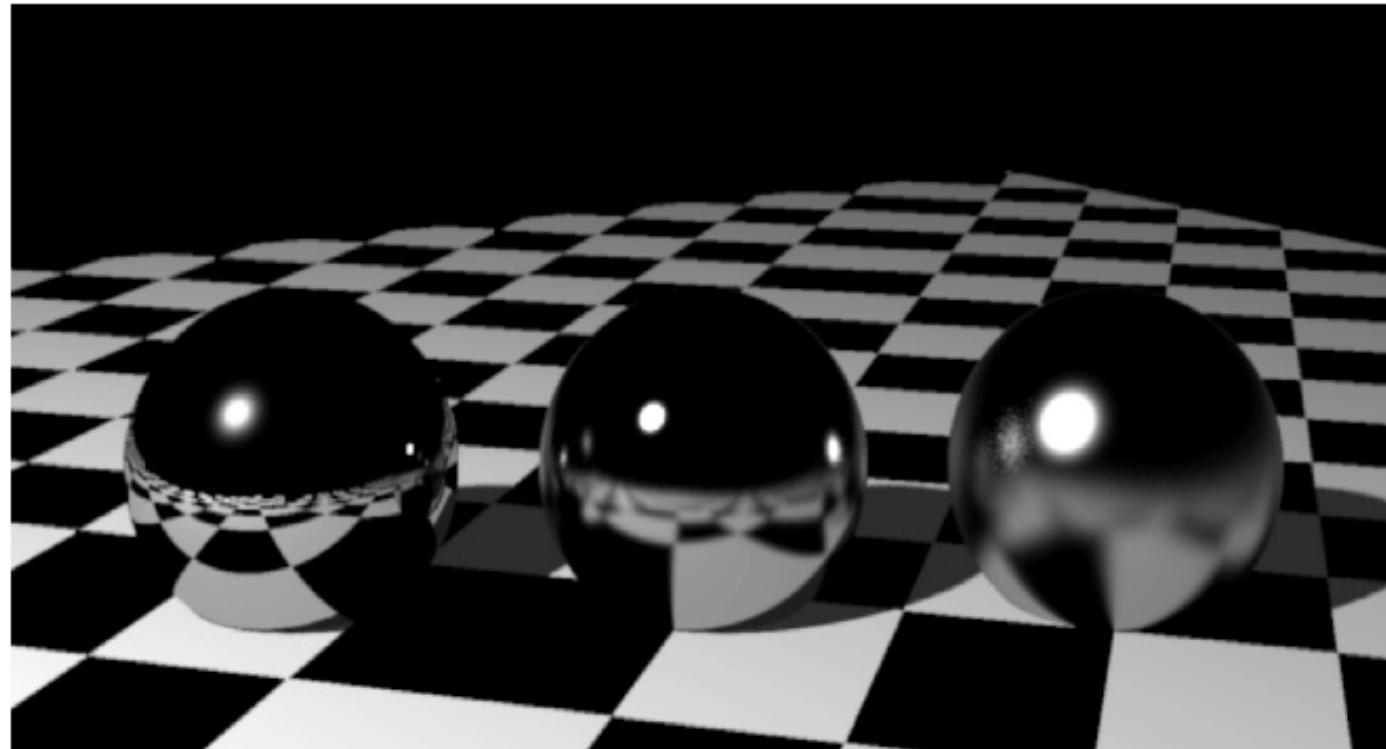
## *Example*



1 sample per ray

Graphics Lecture 10: Slide 54

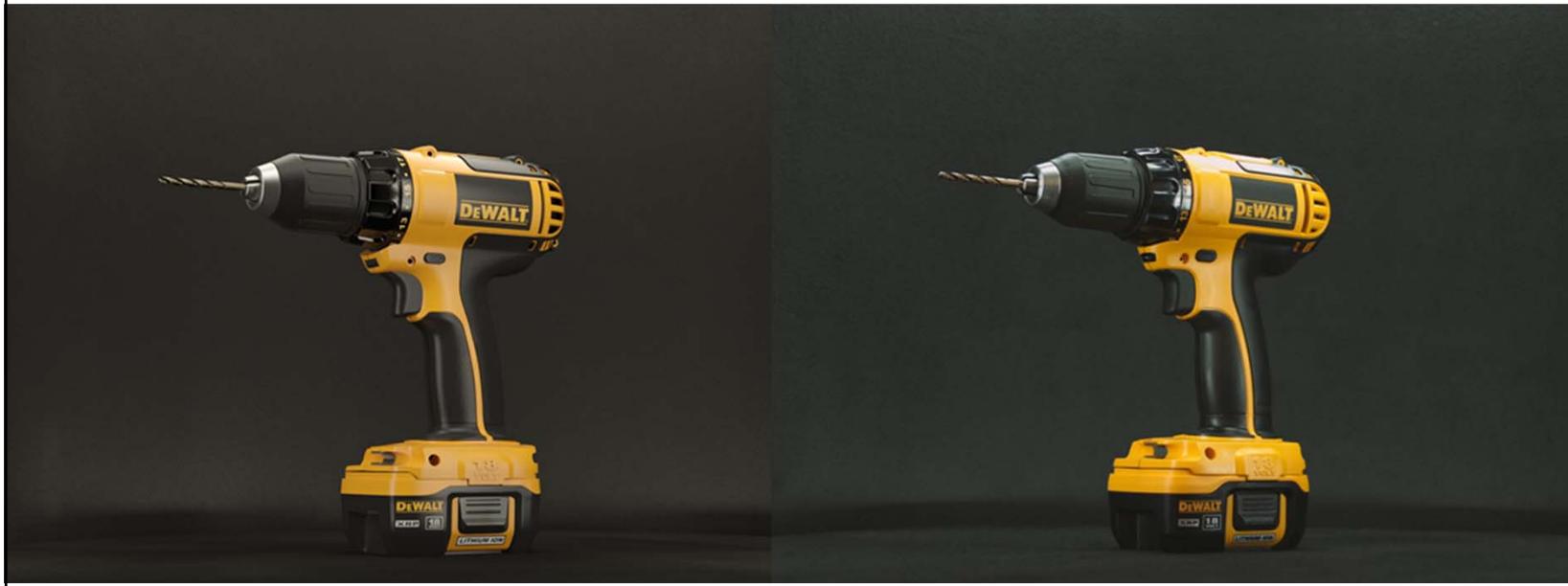
## *Example*



256 samples per ray

Graphics Lecture 10: Slide 55

## *Some cool pictures*



Graphics Lecture 10: Slide 56

## *Some cool pictures*

"Fukaya House" Waro Kishi



Graphics Lecture 10: Slide 57

## *Some cool pictures*



Graphics Lecture 10: Slide 58

## *Some cool pictures*



Graphics Lecture 10: Slide 59

## *Some cool pictures*



Copyright 2000 Gilles Tran

Graphics Lecture 10: Slide 6

## *Some cool pictures*

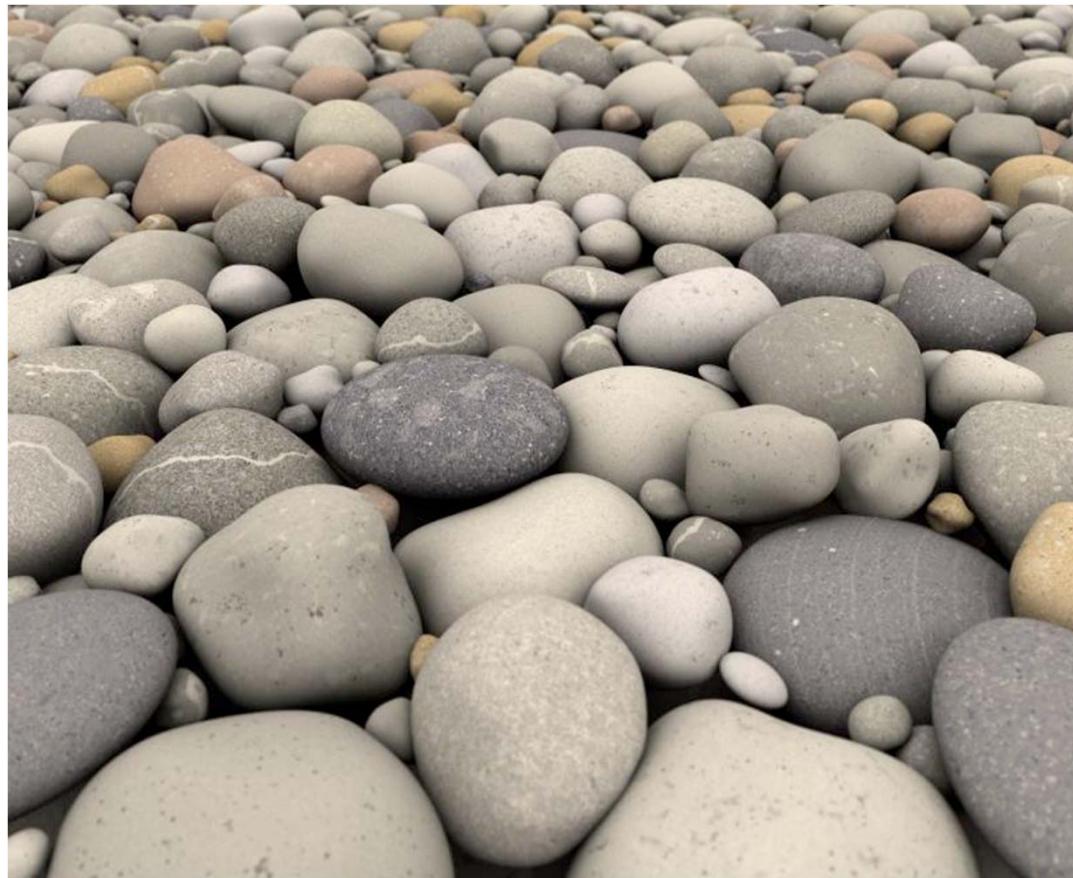


## *Some cool pictures*



Graphics Lecture 10: Slide 62

## *Some cool pictures*



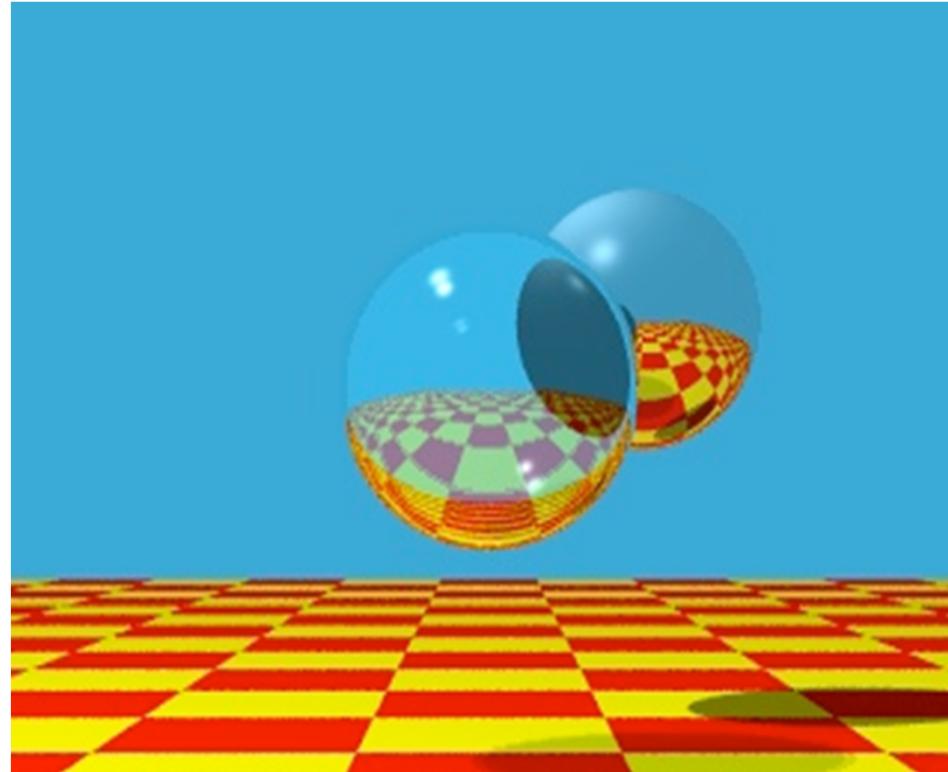
Graphics Lecture 10: Slide 63

took 4.5 days to render! (10 years ago)

# *Interactive Computer Graphics: Lecture 11*

Ray tracing (cont.)

## *Ray tracing - Summary*



Graphics Lecture 11: Slide 2

## *Ray tracing - Summary*

trace ray

    Intersect all objects

    color = ambient term

    For every light

        cast shadow ray

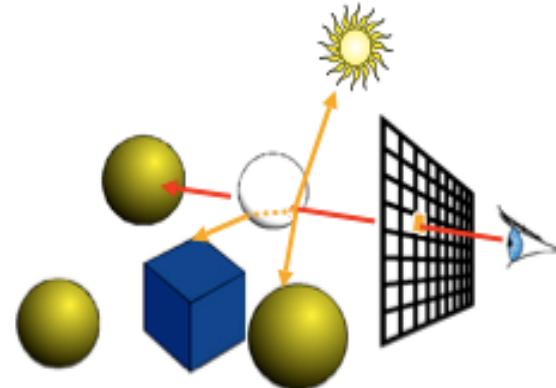
        col += local shading term

    If mirror

        col += k\_refl \* trace reflected ray

    If transparent

        col += k\_trans \* trace transmitted ray



## *Ray tracing - Summary*

trace ray

Intersect all objects

color = ambient term

For every light

    cast shadow ray

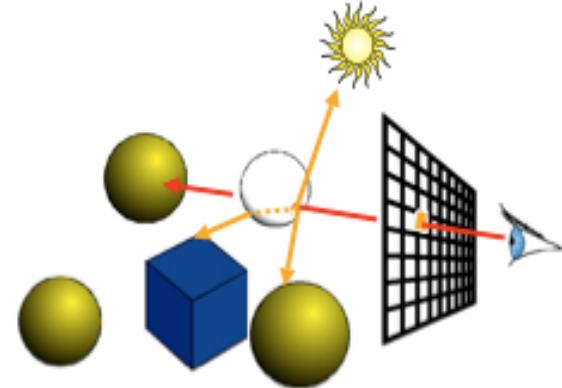
        col += local shading term

If mirror

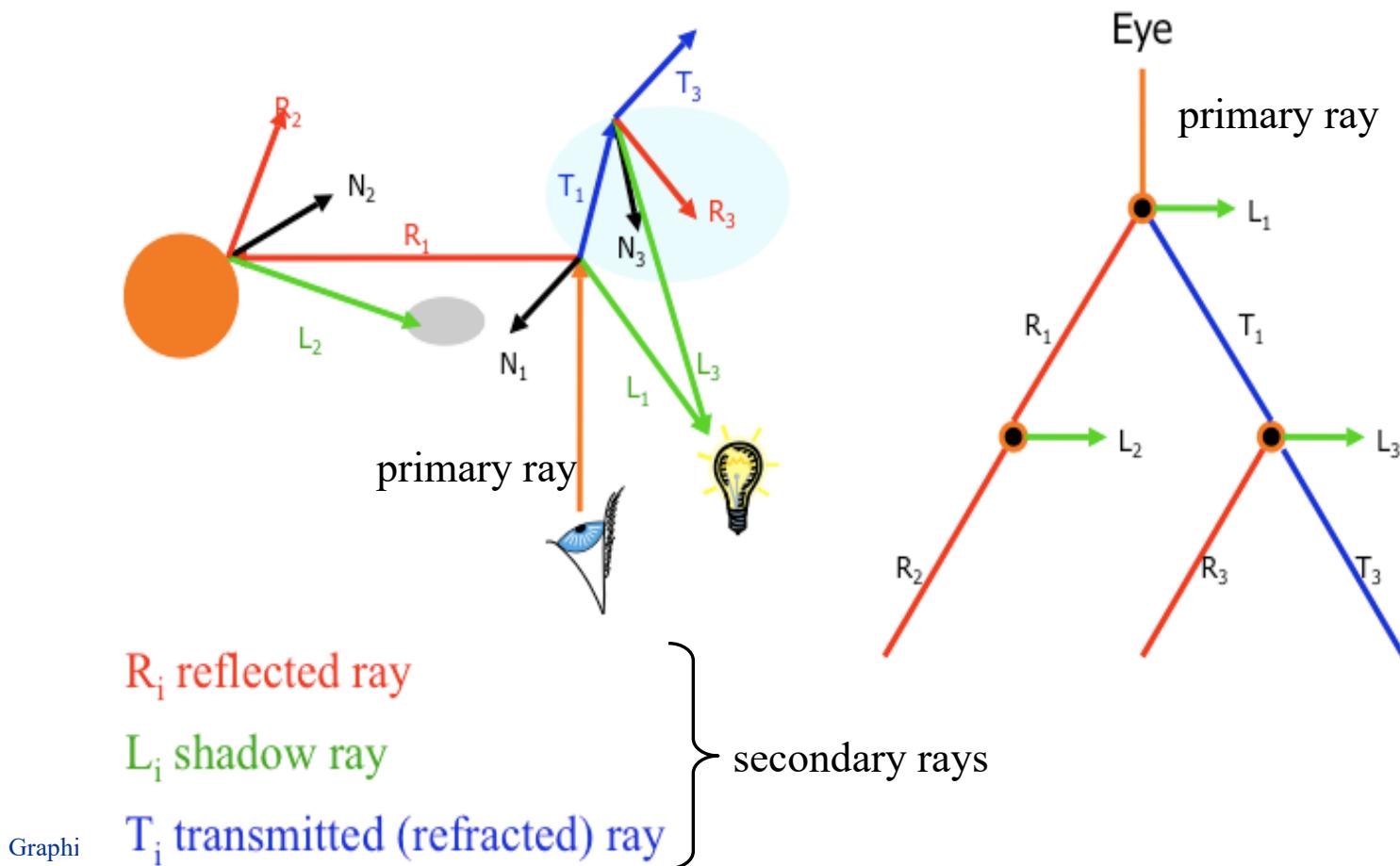
    col += k\_refl \* trace reflected ray

If transparent

    col += k\_trans \* trace transmitted ray



## Ray tracing - Summary



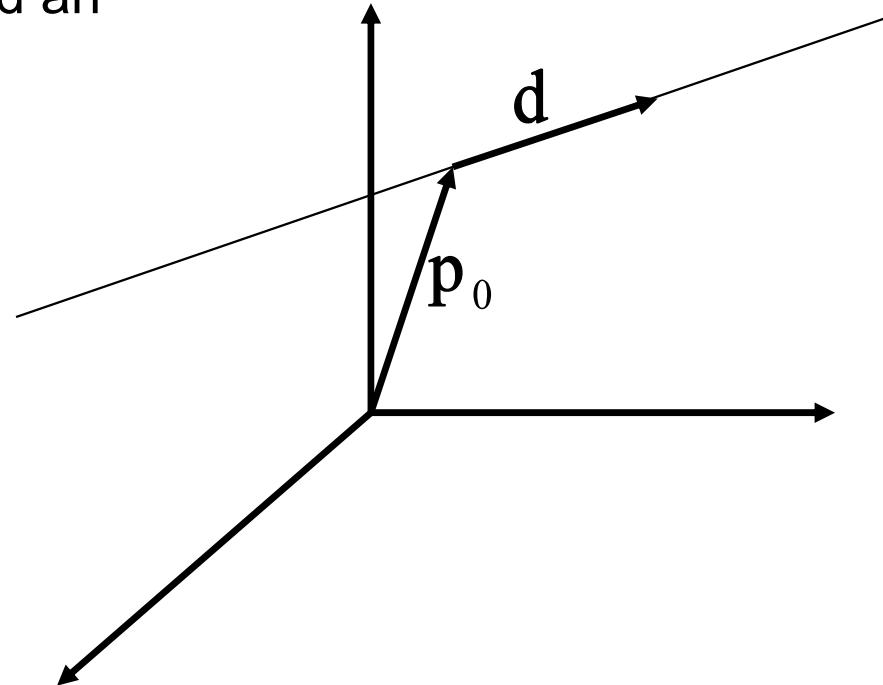
## *Intersection calculations*

- For each ray we must calculate all possible intersections with each object inside the viewing volume
- For each ray we must find the nearest intersection point
- We can define our scene using
  - Solid models
    - sphere
    - cylinder
  - Surface models
    - plane
    - triangle
    - polygon

## Rays

- Rays are parametric lines
- Rays can be defined as
  - origin  $p_0$
  - direction  $d$
- Equation of ray:

$$p(\mu) = p_0 + \mu d$$



Graphics Lecture 11: Slide 7

## *Ray tracing: Intersection calculations*

- The coordinates of any point along each primary ray are given by:

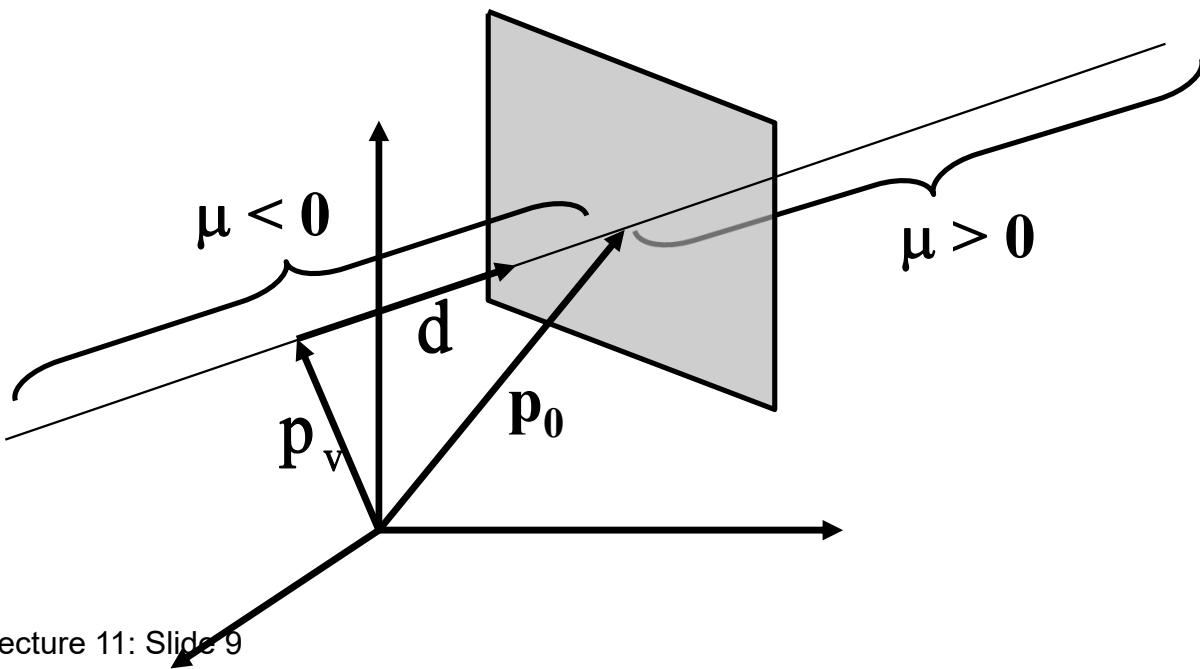
$$\mathbf{p} = \mathbf{p}_0 + \mu \mathbf{d}$$

- $\mathbf{p}_0$  is the current pixel on the viewing plane.
- $\mathbf{d}$  is the direction vector and can be obtained from the position of the pixel on the viewing plane  $\mathbf{p}_0$  and the viewpoint  $\mathbf{p}_v$ :

$$\mathbf{d} = \frac{\mathbf{p}_0 - \mathbf{p}_v}{|\mathbf{p}_0 - \mathbf{p}_v|}$$

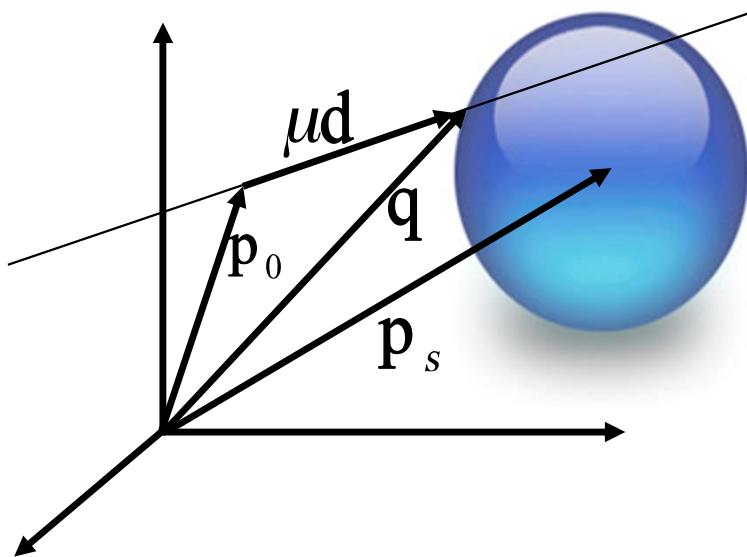
## *Ray tracing: Intersection calculations*

- The viewing ray can be parameterized by  $\mu$ :
  - $\mu > 0$  denotes the part of the ray behind the viewing plane
  - $\mu < 0$  denotes the part of the ray in front of the viewing plane
  - For any visible intersection point  $\mu > 0$



Graphics Lecture 11: Slide 9

## *Intersection calculations: Spheres*



- For any point on the surface of the sphere

$$|q - p_s|^2 - r^2 = 0$$

- where  $r$  is the radius of the sphere

Graphics Lecture 11: Slide 10

## *Intersection calculations: Spheres*

- To test whether a ray intersects a surface we can substitute for  $q$  using the ray equation:

$$|\mathbf{p}_0 + \mu \mathbf{d} - \mathbf{p}_s|^2 - r^2 = 0$$

- Setting  $\Delta\mathbf{p} = \mathbf{p}_0 - \mathbf{p}_s$  and expanding the dot product produces the following quadratic equation:

$$\mu^2 + 2\mu(\mathbf{d} \cdot \Delta\mathbf{p}) + |\Delta\mathbf{p}|^2 - r^2 = 0$$

## *Intersection calculations: Spheres*

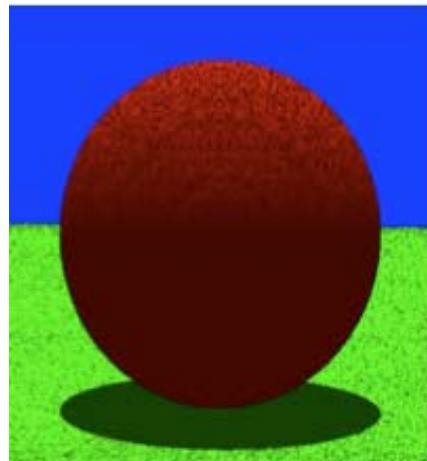
- The quadratic equation has the following solution:

$$\mu = -\mathbf{d} \cdot \Delta\mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta\mathbf{p})^2 - |\Delta\mathbf{p}|^2 + r^2}$$

- Solutions:
  - if the quadratic equation has no solution, the ray does not intersect the sphere
  - if the quadratic equation has two solutions ( $\mu_1 < \mu_2$ ):
    - $\mu_1$  corresponds to the point at which the rays enters the sphere
    - $\mu_2$  corresponds to the point at which the rays leaves the sphere

## Precision Problems

- In ray tracing, the origin of (secondary) rays is often on the surface of objects
  - Theoretically,  $\mu = 0$  for these rays
  - Practically, calculation imprecision creeps in, and the origin of the new ray is slightly beneath the surface
- Result: the surface area is shadowing itself



Graphics Lecture 11: Slide 13

## $\varepsilon$ to the rescue ...

- Check if  $t$  is within some epsilon tolerance:
  - if  $\text{abs}(\mu) < \varepsilon$ 
    - point is on the sphere
  - else
    - point is inside/outside
  - Choose the  $\varepsilon$  tolerance empirically
- Move the intersection point by epsilon along the surface normal so it is outside of the object
- Check if point is inside/outside surface by checking the sign of the implicit (sphere etc.) equation

## *Intersection calculations: Cylinders*

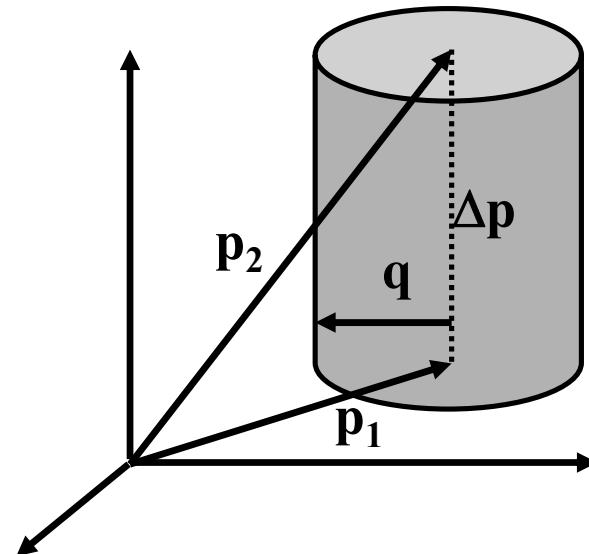
- A cylinder can be described by
  - a position vector  $p_1$  describing the first end point of the long axis of the cylinder
  - a position vector  $p_2$  describing the second end point of the long axis of the cylinder
  - a radius  $r$
- The axis of the cylinder can be written as  $\Delta p = p_1 - p_2$  and can be parameterized by  $0 \leq \alpha \leq 1$

## *Intersection calculations: Cylinders*

- To calculate the intersection of the cylinder with the ray:

$$p_1 + \alpha \Delta p + q = p_0 + \mu d$$

- Since  $q \cdot \Delta p = 0$  we can write



$$\alpha(\Delta p \cdot \Delta p) = p_0 \cdot \Delta p + \mu d \cdot \Delta p - p_1 \cdot \Delta p$$

## *Intersection calculations: Cylinders*

- Solving for  $\alpha$  yields:

$$\alpha = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

- Substituting we obtain:

$$\mathbf{q} = \mathbf{p}_0 + \mu\mathbf{d} - \mathbf{p}_1 - \left( \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}} \right) \Delta\mathbf{p}$$

## *Intersection calculations: Cylinders*

- Using the fact that  $\mathbf{q} \cdot \mathbf{q} = r^2$  we can use the same approach as before to the quadratic equation for  $\mu$ :

$$r^2 = \left( \mathbf{p}_0 + \mu \mathbf{d} - \mathbf{p}_1 - \left( \frac{\mathbf{p}_0 \cdot \Delta \mathbf{p} + \mu \mathbf{d} \cdot \Delta \mathbf{p} - \mathbf{p}_1 \cdot \Delta \mathbf{p}}{\Delta \mathbf{p} \cdot \Delta \mathbf{p}} \right) \Delta \mathbf{p} \right)^2$$

- If the quadratic equation has no solution:  
**no intersection**
- If the quadratic equation has two solutions:  
**intersection**

## *Intersection calculations: Cylinders*

- Assuming that  $\mu_1 \leq \mu_2$  we can determine two solutions:

$$\alpha_1 = \frac{p_0 \cdot \Delta p + \mu_1 d \cdot \Delta p - p_1 \cdot \Delta p}{\Delta p \cdot \Delta p}$$

$$\alpha_2 = \frac{p_0 \cdot \Delta p + \mu_2 d \cdot \Delta p - p_1 \cdot \Delta p}{\Delta p \cdot \Delta p}$$

- If the value of  $\alpha_1$  is between 0 and 1 the intersection is on the outside surface of the cylinder
- If the value of  $\alpha_2$  is between 0 and 1 the intersection is on the inside surface of the cylinder

## *Intersection calculations: Plane*

- Objects are often described by geometric primitives such as
  - triangles
  - planar quads
  - planar polygons
- To test intersections of the ray with these primitives we must test whether the ray will intersect the plane defined by the primitive

## *Intersection calculations: Plane*

- The intersection of a ray with a plane is given by

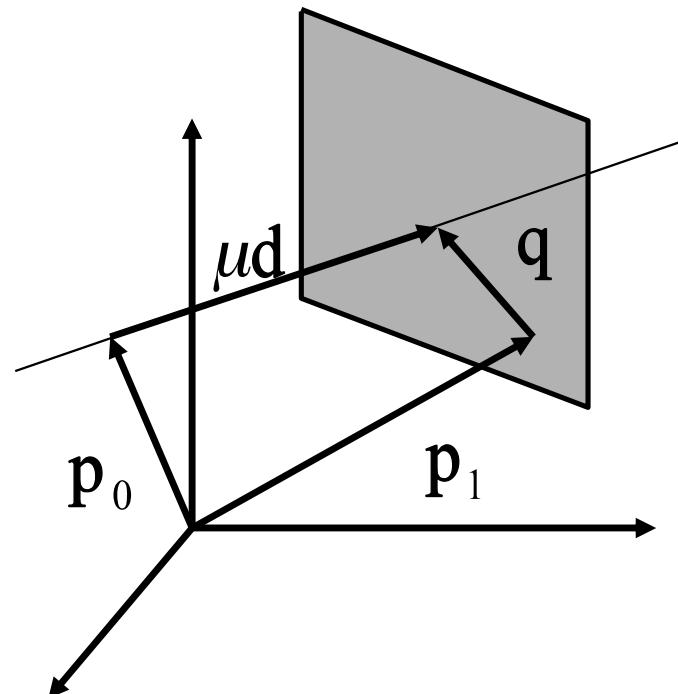
$$\mathbf{p}_1 + \mathbf{q} = \mathbf{p}_0 + \mu \mathbf{d}$$

- where  $\mathbf{p}_1$  is a point in the plane.  
Subtracting  $\mathbf{p}_1$  and multiplying with the normal of the plane  $\mathbf{n}$  yields:

$$\mathbf{q} \cdot \mathbf{n} = 0 = (\mathbf{p}_0 - \mathbf{p}_1) \cdot \mathbf{n} + \mu \mathbf{d} \cdot \mathbf{n}$$

- Solving for  $\mu$  yields:

$$\mu = -\frac{(\mathbf{p}_0 - \mathbf{p}_1) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

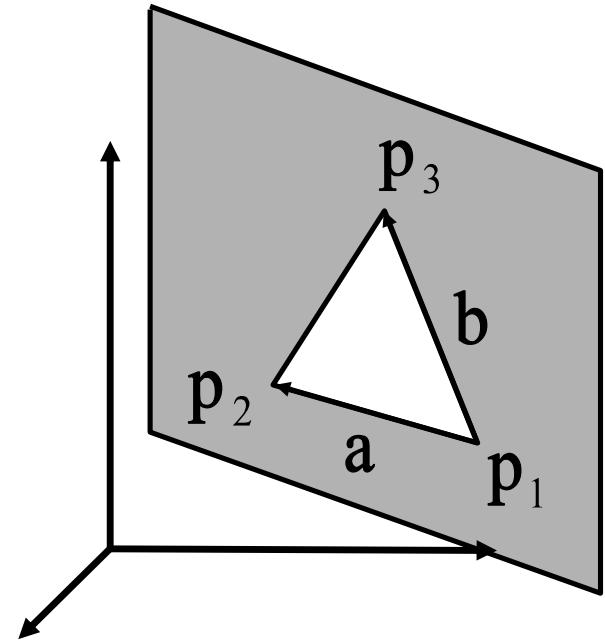


Graphics Lecture 11: Slide 21

## *Intersection calculations: Triangles*

- To calculate intersections:
  - test whether triangle is front facing
  - test whether plane of triangle intersects ray
  - test whether intersection point is inside triangle
- If the triangle is front facing:

$$d \cdot n < 0$$



Graphics Lecture 11: Slide 22

## *Intersection calculations: Triangles*

- To test whether plane of triangle intersects ray

– calculate equation of the plane using

$$\mathbf{p}_2 - \mathbf{p}_1 = \mathbf{a}$$

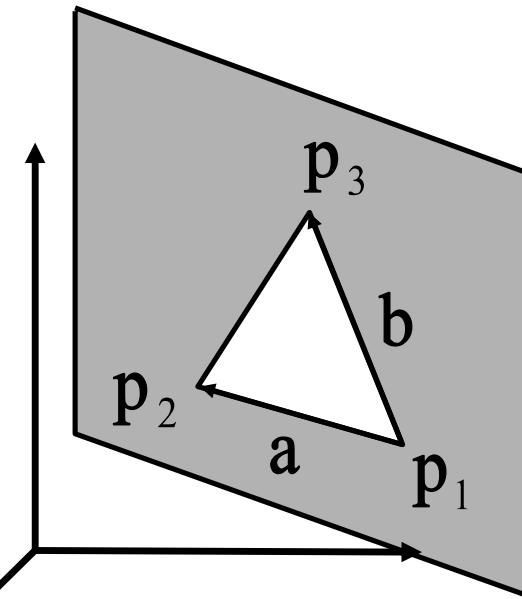
$$\mathbf{p}_3 - \mathbf{p}_1 = \mathbf{b}$$

– calculate intersections with plane as before

$$\mathbf{n} = \mathbf{a} \times \mathbf{b}$$

- To test whether intersection point is inside triangle:

$$\mathbf{q} = \alpha \mathbf{a} + \beta \mathbf{b}$$



## *Intersection calculations: Triangles*

- A point is inside the triangle if

$$0 \leq \alpha \leq 1$$

$$0 \leq \beta \leq 1$$

$$\alpha + \beta \leq 1$$

- Calculate  $\alpha$  and  $\beta$  by taking the dot product with  $a$  and  $b$ :

$$\alpha = \frac{(\mathbf{b} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{a}) - (\mathbf{a} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{b})}{(\mathbf{a} \cdot \mathbf{a})(\mathbf{b} \cdot \mathbf{b}) - (\mathbf{a} \cdot \mathbf{b})^2}$$

$$\beta = \frac{\mathbf{q} \cdot \mathbf{b} - \alpha(\mathbf{a} \cdot \mathbf{b})}{\mathbf{b} \cdot \mathbf{b}}$$

Graphics Lecture 11: Slide 24

## *Intersection calculations: Triangles*

- algorithm by **Möller and Trumbore**:
- translate the origin of the ray and then change the base of that vector which yields parameter vector  $(t, u, v)$
- $t$  is the distance to the plane in which the triangle lies
- $(u, v)$  are barycentric coordinates inside the triangle
- plane equation need not be computed on the fly nor be stored

Graphics Lecture 11: Slide 25

## Intersection calculations: Triangles

```
1 bool triangle_intersection( vec3 v1,
2                             vec3 v2,
3                             vec3 v3, // Triangle vertices
4                             vec3 origin, //Ray origin
5                             vec3 ray_dir, //Ray direction
6                             float* out )
7
8 //Find vectors for two edges sharing v1
9 vec3 edge1 = v2-v1;
10 vec3 edge2 = v3-v1;
11
12 //Begin calculating determinant - also used to calculate u parameter
13 vec3 p = cross(ray_dir, edge2)
14
15 //if determinant is near zero, ray lies in plane of triangle or ray
16 //is parallel to plane of triangle
17 float det = dot(edge1, p);
18
19 if(det > -EPSILON && det < EPSILON)
20 return false;
21
22 float inv_det = 1.f / det;
23
24 //calculate distance from v1 to ray origin
25 vec3 t = origin-v1;
```

## Intersection calculations: Triangles

```
27 //Calculate u parameter and test bound
28 float u = dot(t, p) * inv_det;
29
30 //The intersection lies outside of the triangle
31 if(u < 0.f || u > 1.f)
32 return false;
33
34 //Prepare to test v parameter
35 vec3 q = cross(t, edge1);
36
37 float v = dot(ray_dir, q) * inv_det;
38 //The intersection lies outside of the triangle
39 if(v < 0.f || u + v > 1.f)
40 return false;
41
42 float mu = dot(edge2, q) * inv_det;
43
44 if(t > EPSILON) { //ray intersection
45     *out = mu;
46     return true;
```

## *Ray tracing: Pros and cons*

- Pros:
  - Easy to implement
  - Extends well to global illumination
    - shadows
    - reflections / refractions
    - multiple light bounces
    - atmospheric effects
- Cons:
  - Speed! (seconds per frame, not frames per second)

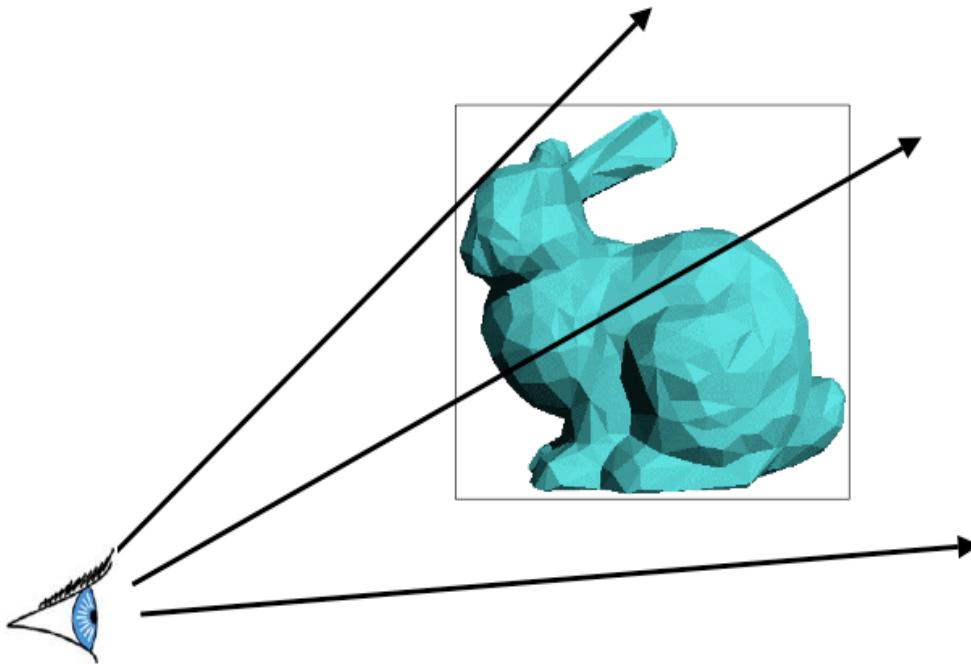
## *Speedup Techniques*

- Why is ray tracing slow? How to improve?
  - Too many objects, too many rays
  - Reduce ray-object intersection tests
  - Many techniques!

Graphics Lecture 11: Slide 29

## *Acceleration of Ray Casting*

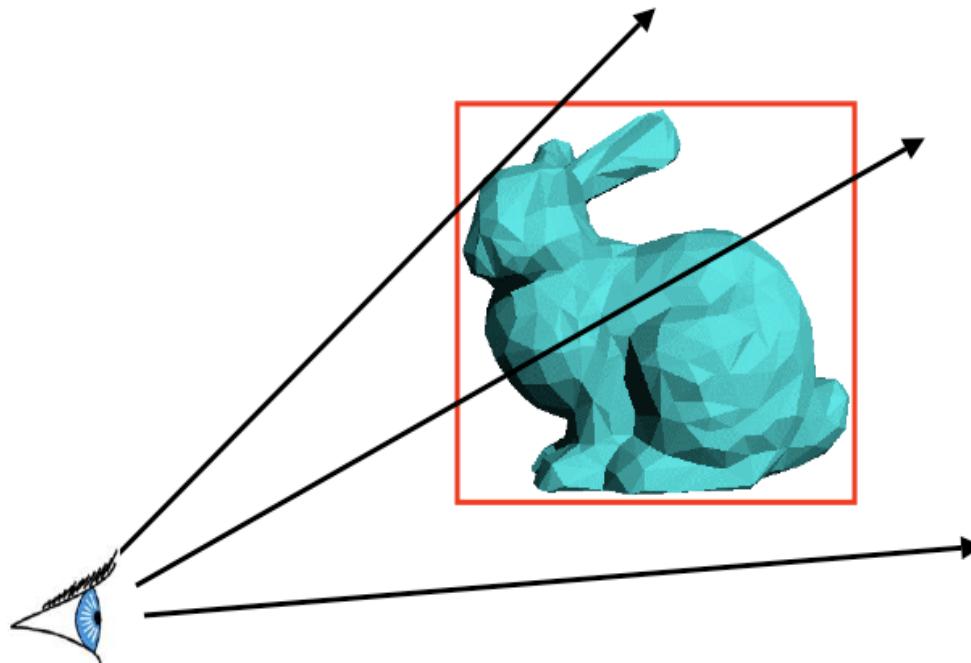
- Goal: Reduce the number of ray/primitive intersections



Graphics Lecture 11: Slide 30

## *Conservative Bounding Region*

- First check for an intersection with a conservative bounding region
- Early reject



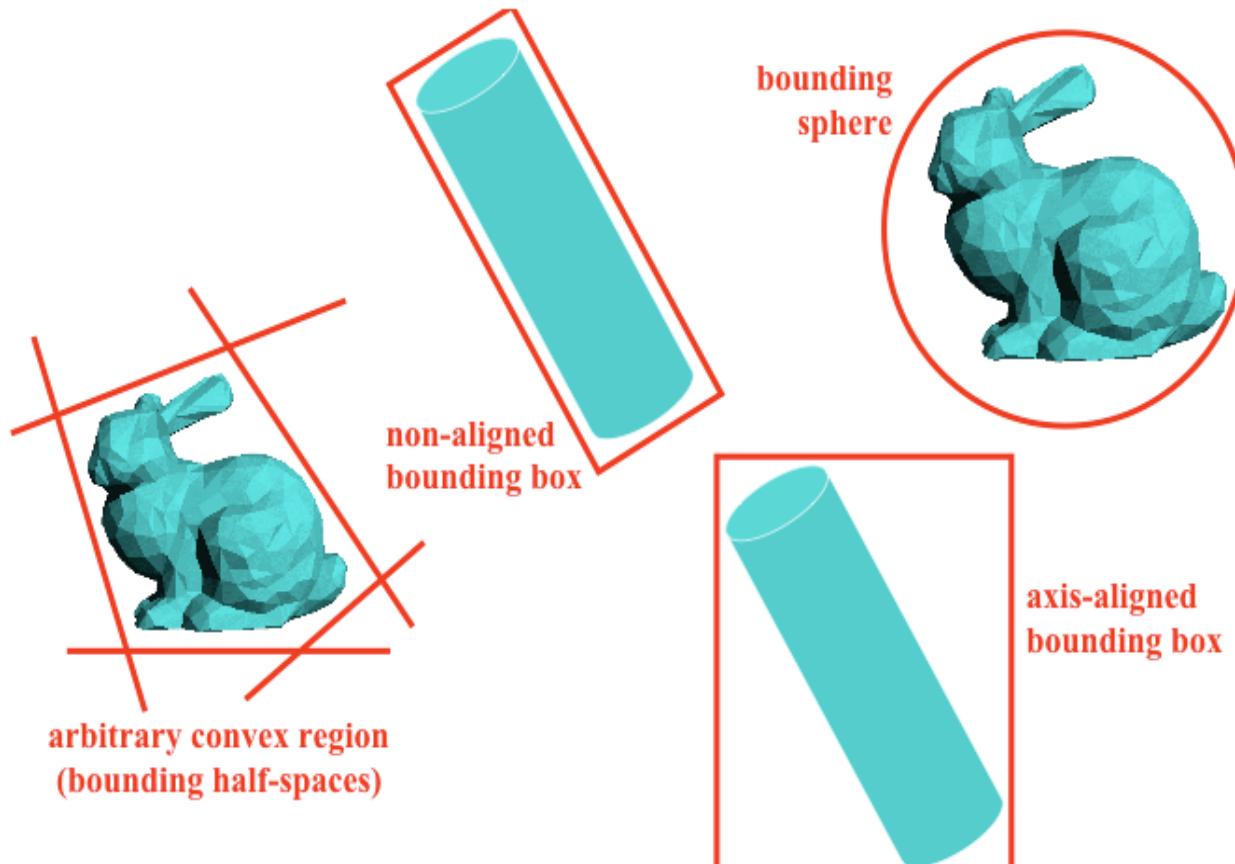
Graphics Lecture 11: Slide 31

## *Bounding Regions*

- What makes a good bounding region?

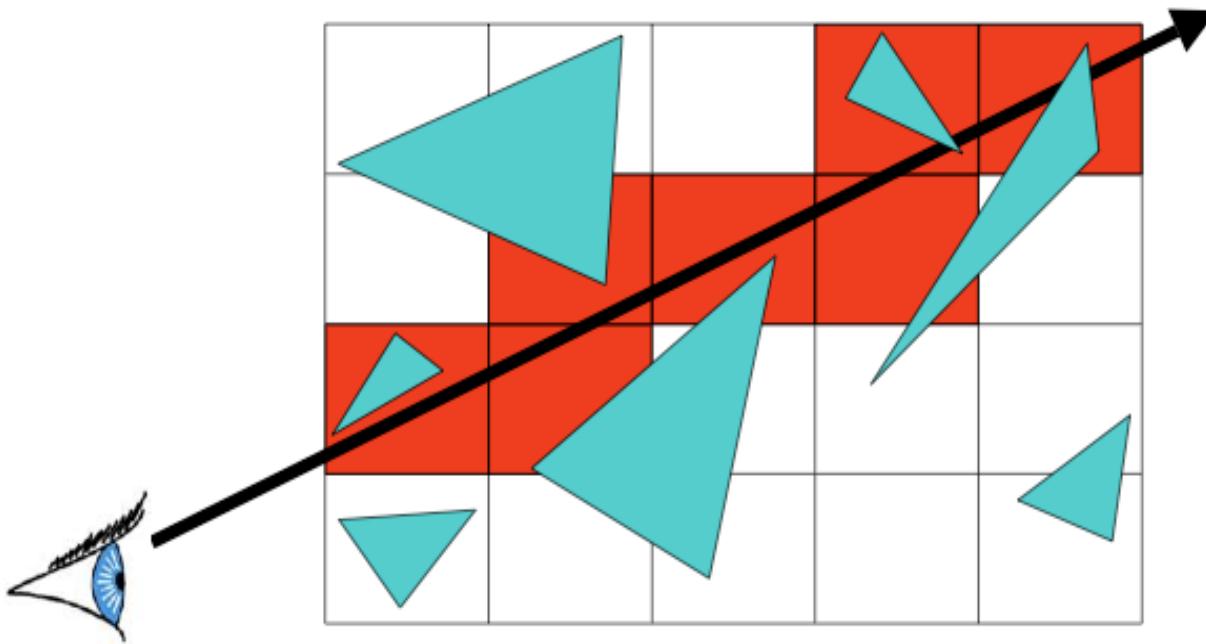
Graphics Lecture 11: Slide 32

## *Conservative Bounding Regions*



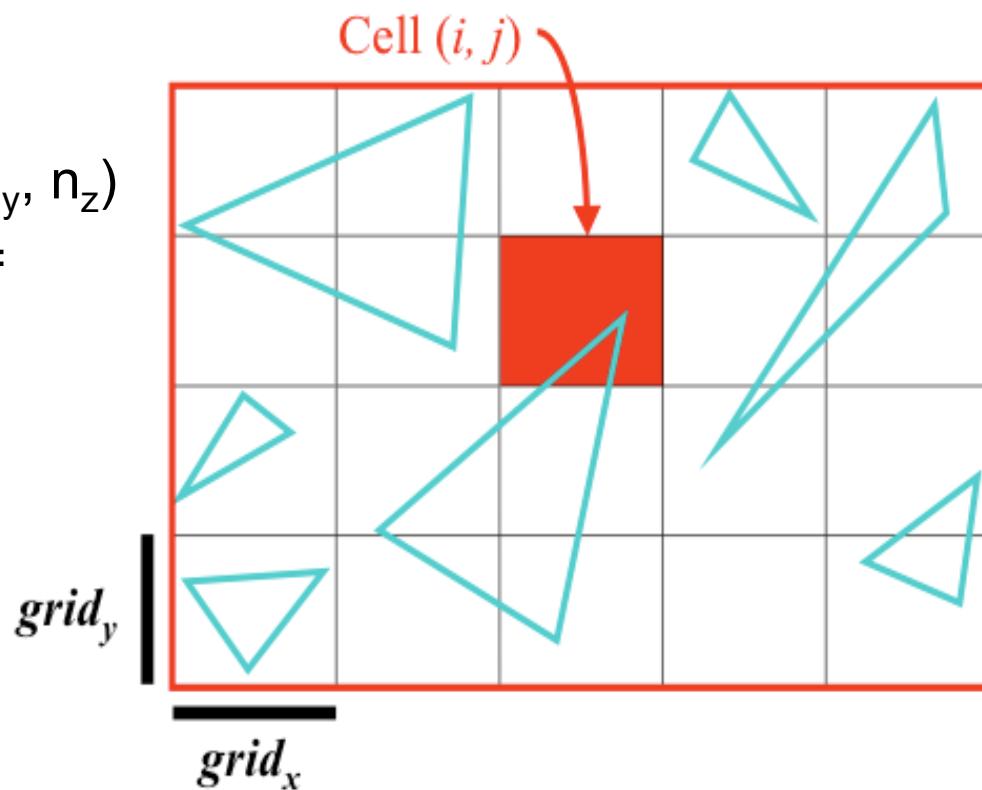
Graphics Lecture 11: Slide 33

## *Regular Grid*



## Create Grid

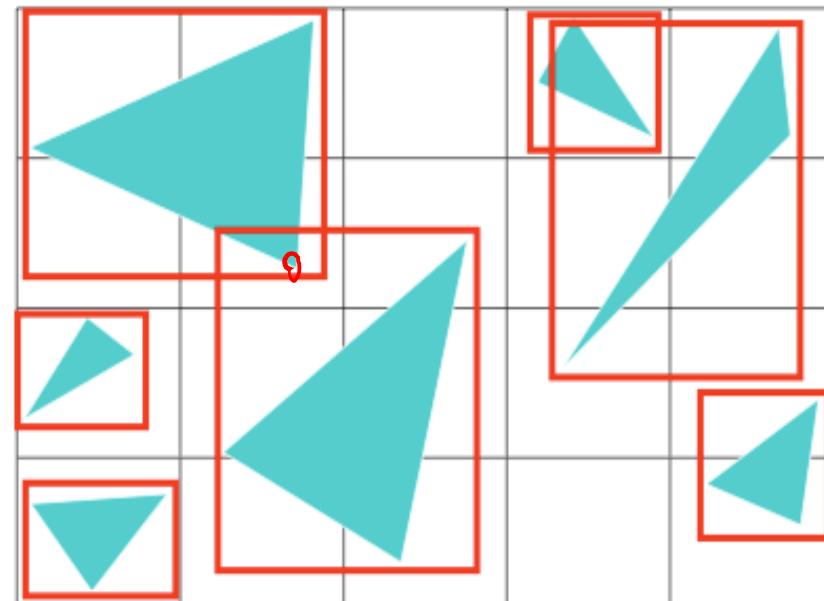
- Find bounding box of scene
- Choose grid resolution ( $n_x, n_y, n_z$ )
- $grid_x$  need not =  $grid_y$



Graphics Lecture 11: Slide 35

## *Insert Primitives into Grid*

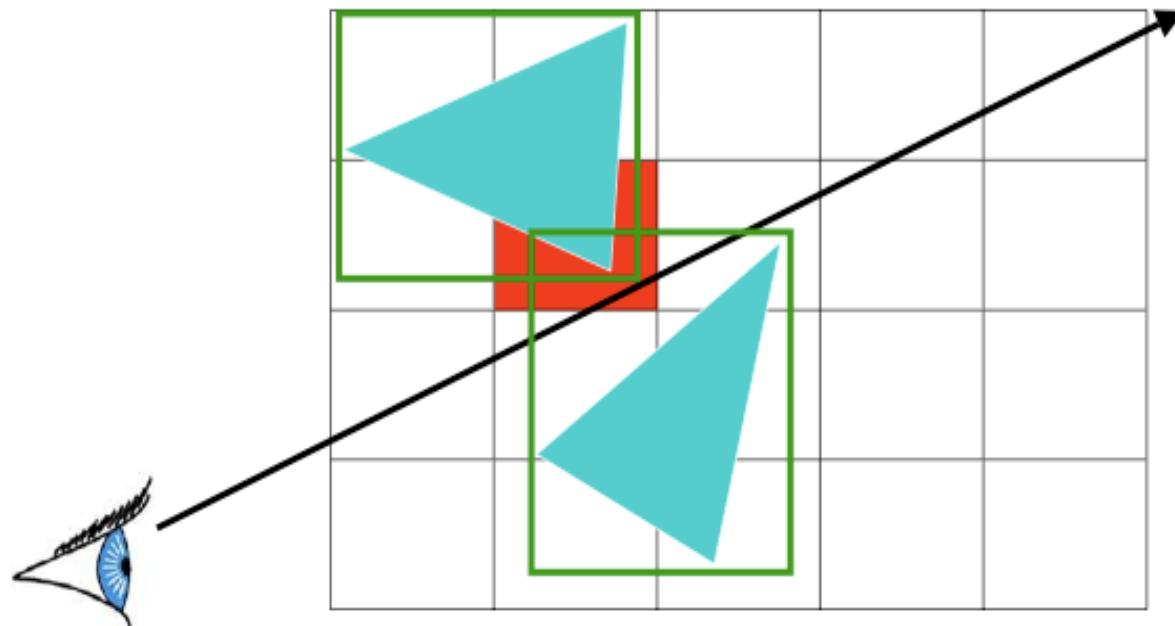
- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)



Graphics Lecture 11: Slide 36

## *For Each Cell Along a Ray*

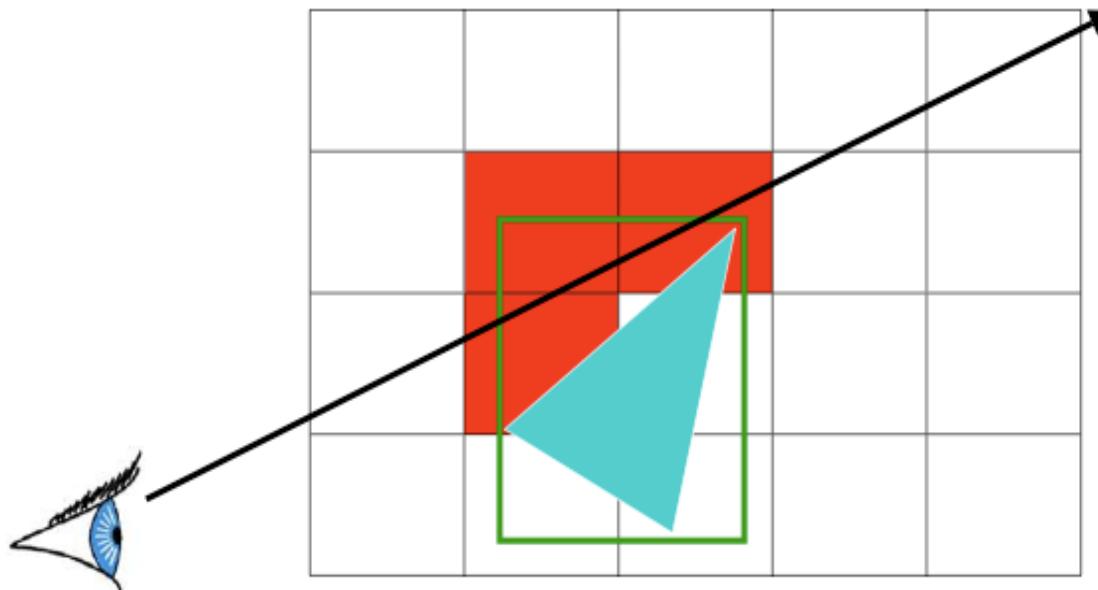
- Does the cell contain an intersection?
  - Yes: return closest intersection
  - No: continue



Graphics Lecture 11: Slide 37

## *Preventing Repeated Computation*

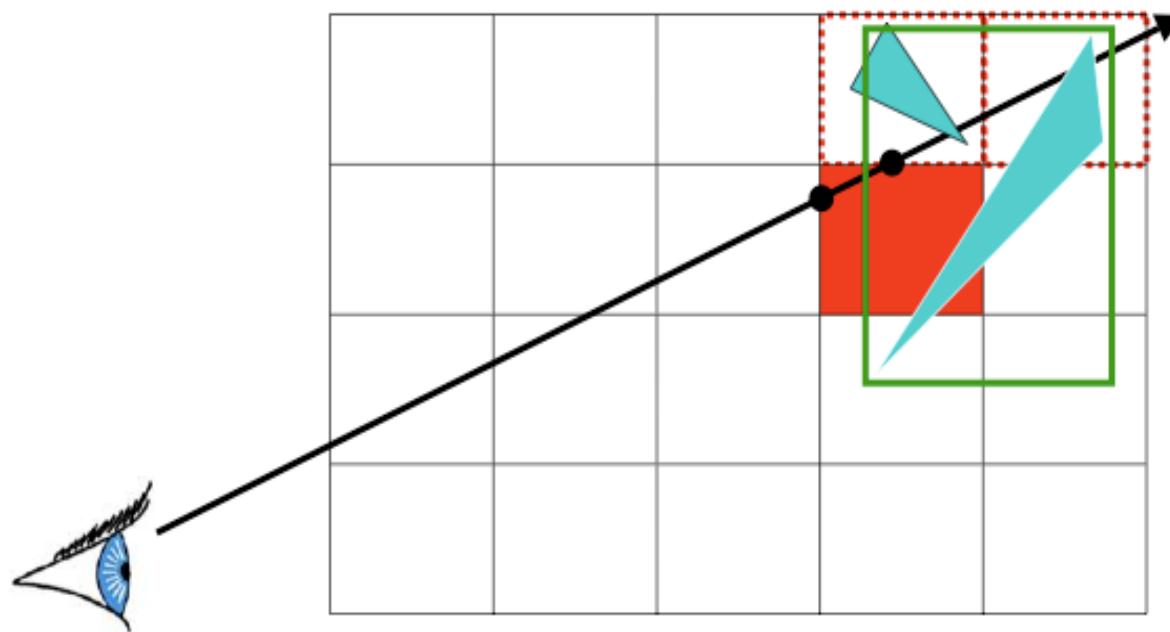
- Perform the computation once, "mark" the object
- Don't re-intersect marked objects



Graphics Lecture 11: Slide 38

## *Don't Return Distant Intersections*

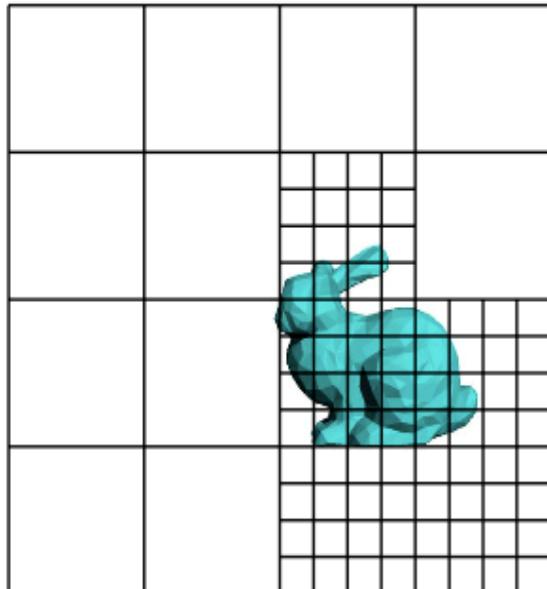
- If intersection  $t$  is not within the cell range, continue (there may be something closer)



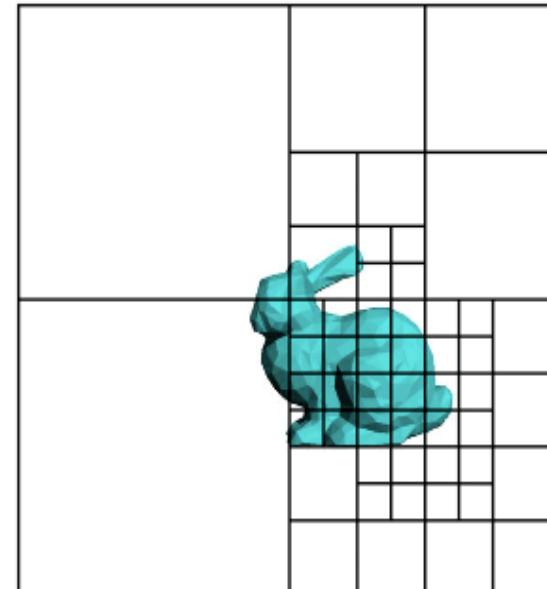
Graphics Lecture 11: Slide 39

## *Adaptive Grids*

- Subdivide until each cell contains no more than  $n$  elements, or maximum depth  $d$  is reached



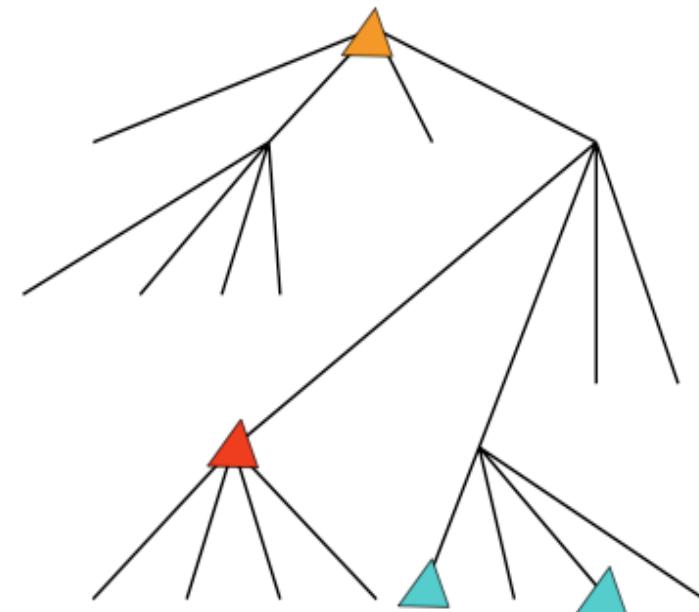
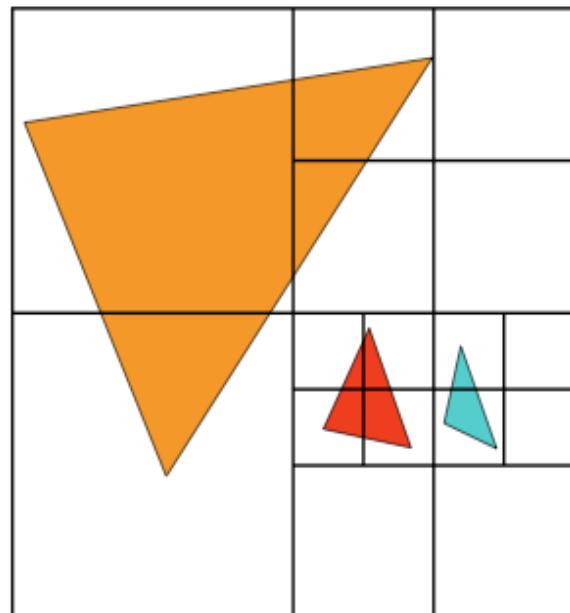
Nested Grids



Octree/(Quadtree)

## *Primitives in an Adaptive Grid*

- Can live at intermediate levels, or be pushed to lowest level of grid

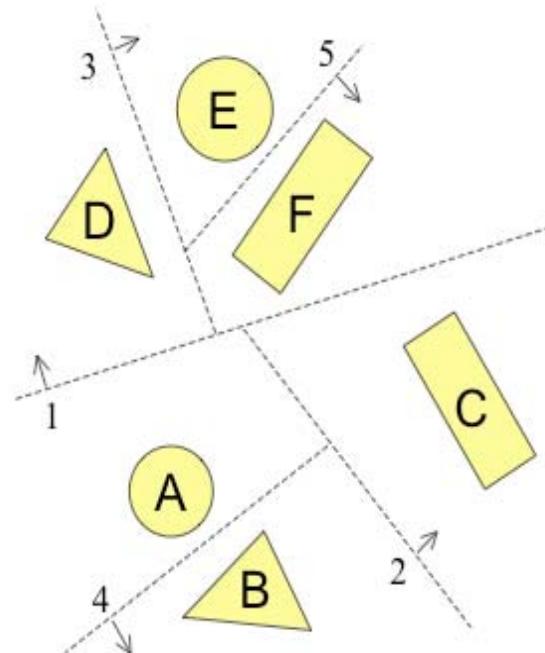
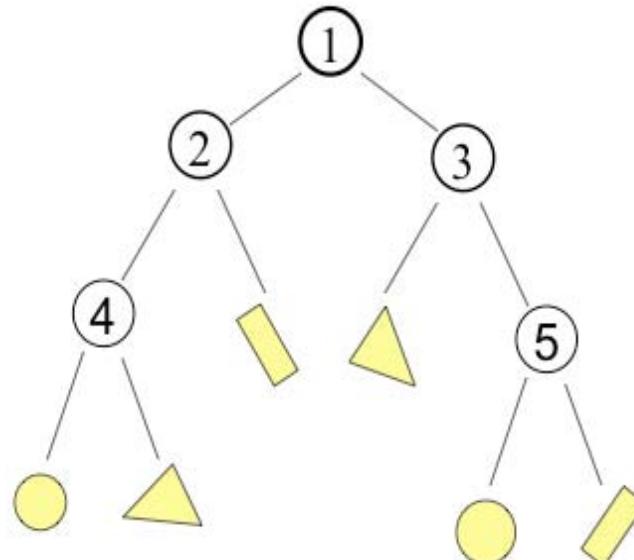


Octree/(Quadtree)

Graphics Lecture 11: Slide 41

## *Binary Space Partition (BSP) Tree*

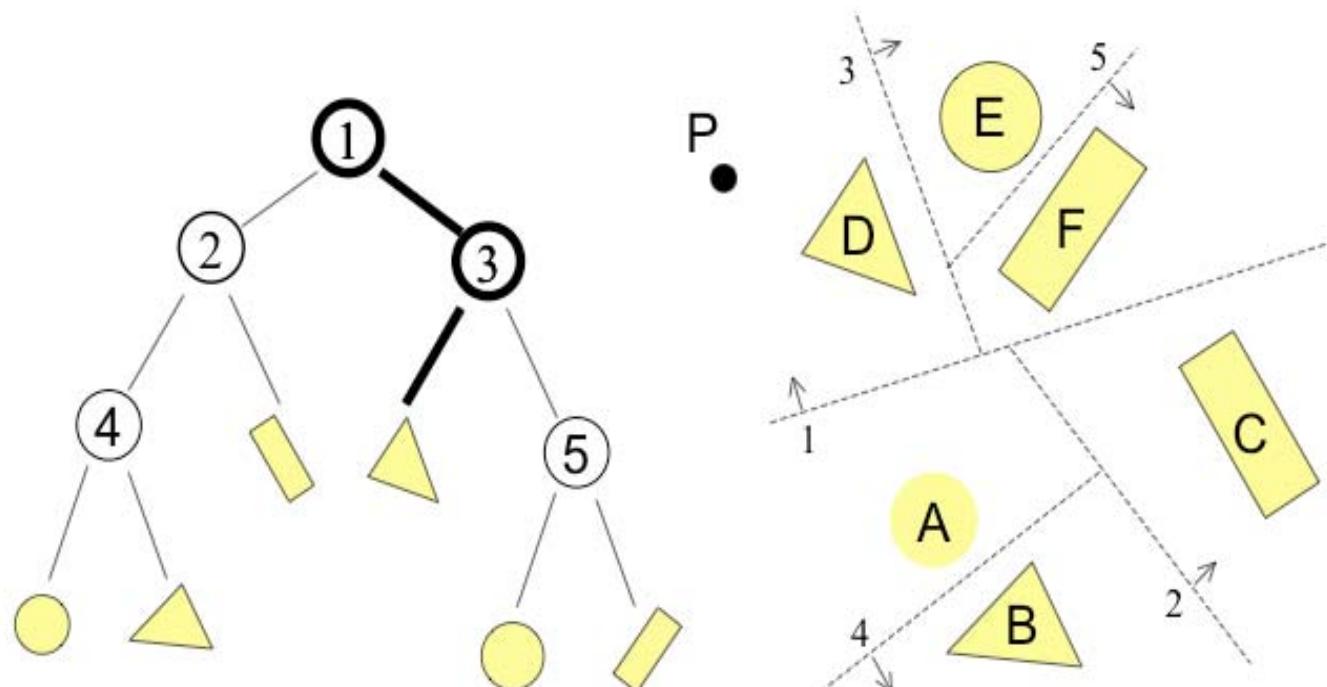
- Recursively partition space by planes
- Every cell is a convex polyhedron



Graphics Lecture 11: Slide 42

## *Binary Space Partition (BSP) Tree*

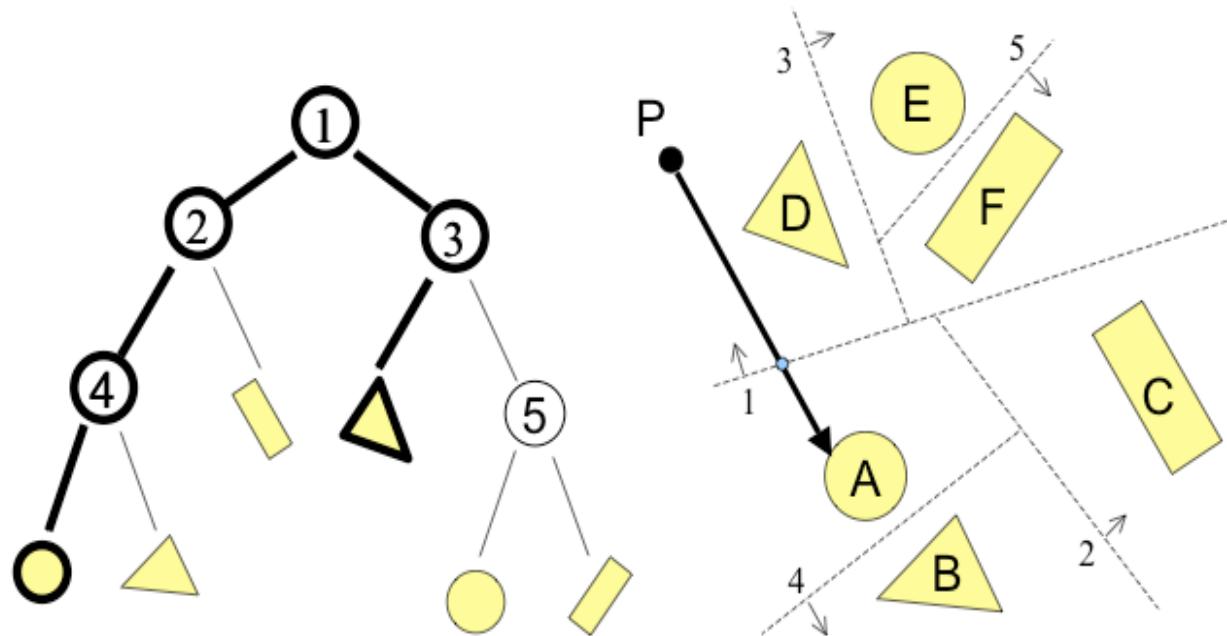
- Simple recursive algorithms
- Example: point finding



Graphics Lecture 11: Slide 43

## *Binary Space Partition (BSP) Tree*

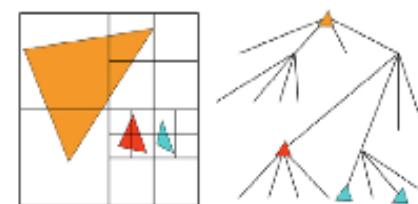
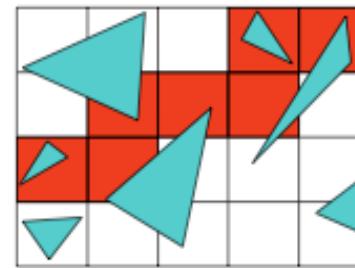
- Trace rays by recursion on tree
  - BSP construction enables simple front-to-back traversal



Graphics Lecture 11: Slide 44

## *Grid Discussion*

- Regular
  - + easy to construct
  - + easy to traverse
  - may be only sparsely filled
  - geometry may still be clumped
- Adaptive
  - + grid complexity matches geometric density
  - more expensive to traverse (especially BSP tree)



## *Example*

- <https://www.youtube.com/watch?v=aKqxonOrI4Q>
- <https://www.youtube.com/watch?v=qlyzo9ll9Vw>
- <https://www.youtube.com/watch?v=AV279wThmVU>
-