Ray Tracing

CS3241 Computer Graphics

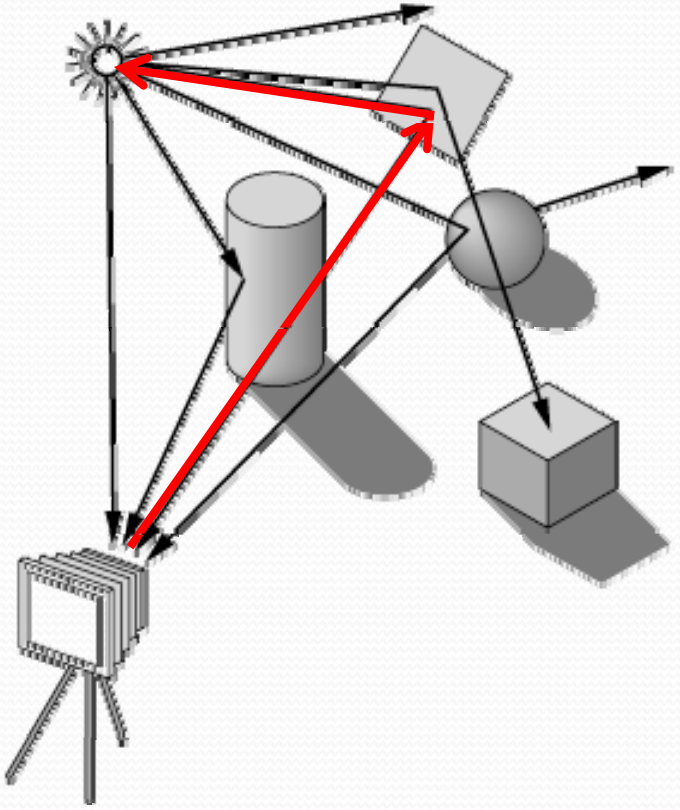
Ray Tracing

- A powerful (but simple) technique for photorealistic CG
  - global reflection, transmission and shadow effects
- But require considerable computation time to generate.

In real world, how does a camera work when you take a photo?

1. Infinitely many light rays coming from a light source
2. After a lot of reflection/refraction, some of them go into the camera and form an image
3. We can simulate this with the computer and produce some photos
   - However, a lot of light beams are wasted.

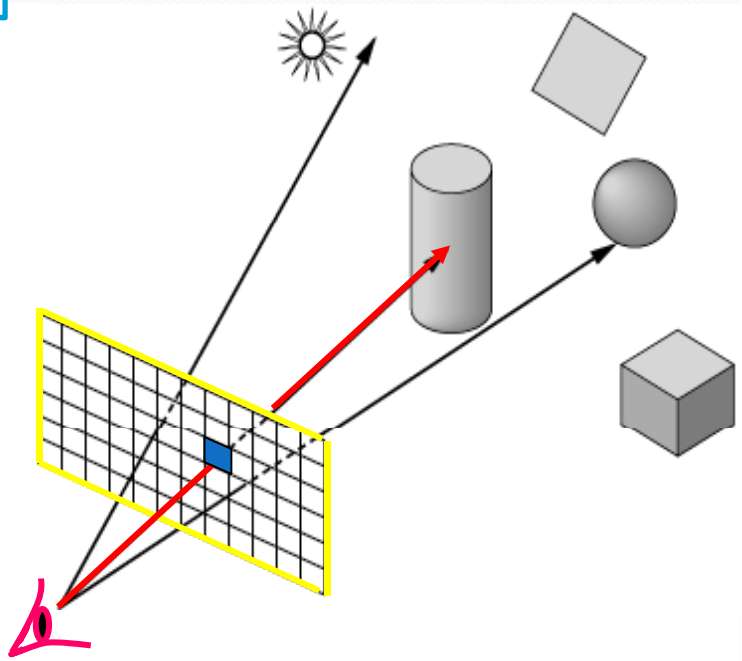
- Instead, we trace the light rays FROM the camera to the objects



Not ray-tracing yet

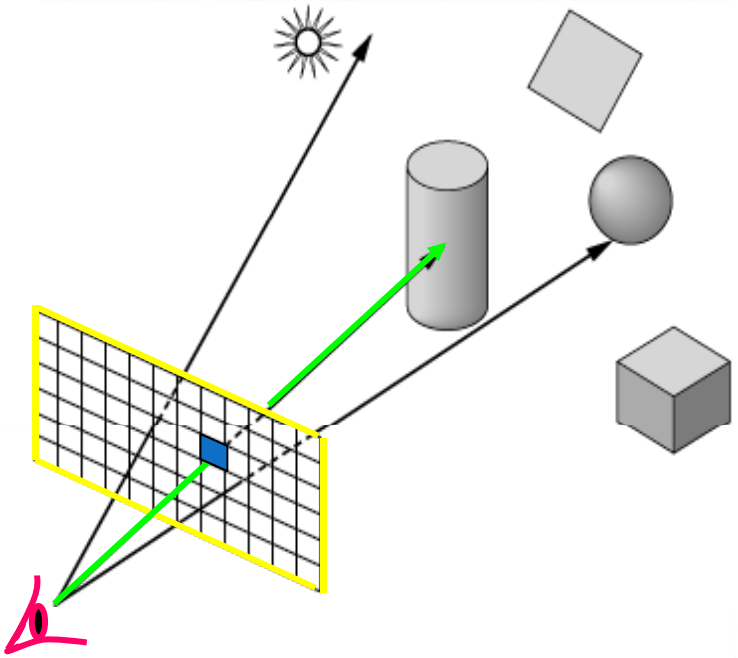
Ray-casting Method

- Set the eye and the image plane position in space
  - The image plane is divided into pixel-sized areas.
- Cast a ray from the eye through the center of each pixel into the scene
- The ray will either hit
  - An object
  - Or the “background”



Ray-casting Method

- If the ray hits an object
  - Compute the color of the surface at the intersection point using *Phong illumination model* and assign the pixel with the color
- If the ray hits nothing, namely, hits the background
  - Assign the background color to the pixel
  - E.g. black for the Universe

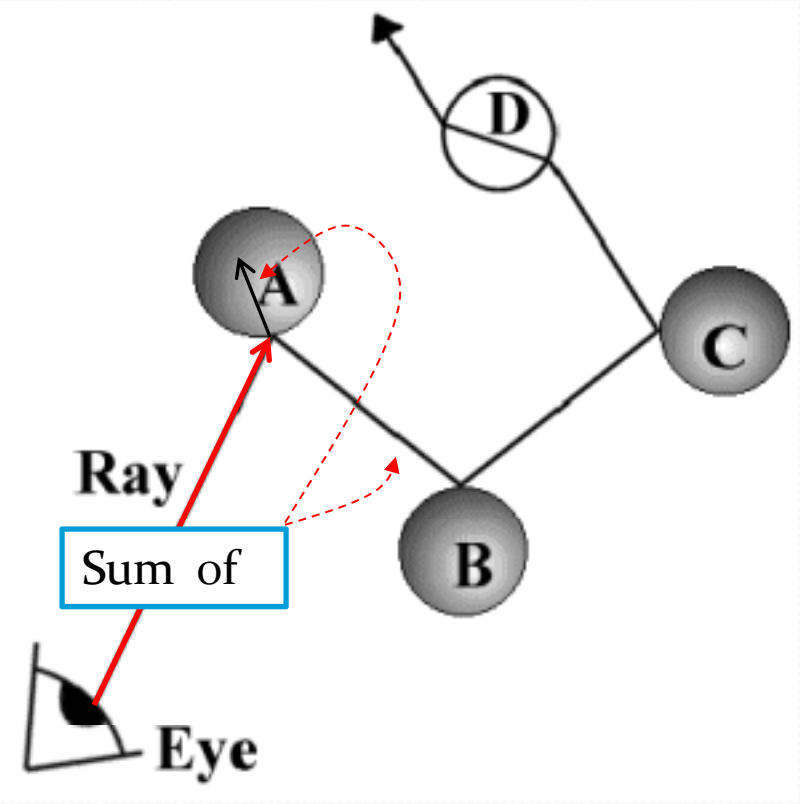


Ray-casting Method

- Result image
  - Same as z-buffer renderer + PIE
- Then what's new? ….. Nothing, except it's a lot slower
- Ray-casting is free of
  - Scanline conversion algorithm
  - Complicated projections and camera view transformation

Ray Tracing

- Extending from ray-casting method, ray tracing trace the lights after it hits an object
- Tracing the
  - Reflected ray, and
  - Refracted ray
- So, the final color for that pixel will be a **weighted summation** of the colors of all the deflected rays

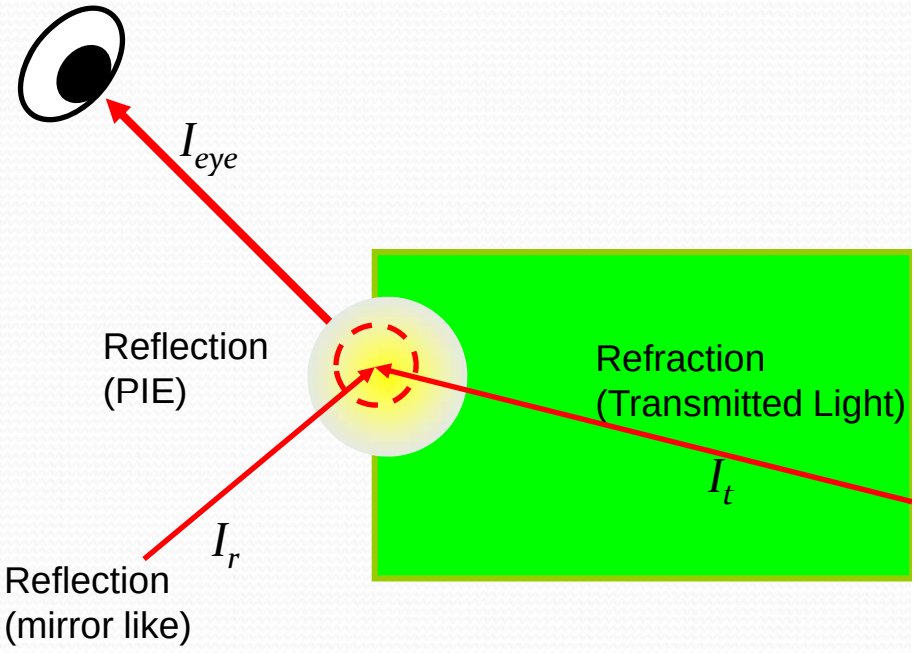


The Intensity on a Surface Point

- What the eye receives is the summation of
  1. Phone Illumination Model
  2. Reflection
  3. Refraction
- Namely

$$I_{eye} = I_{Phong} + k_s I_r + k_t I_t$$

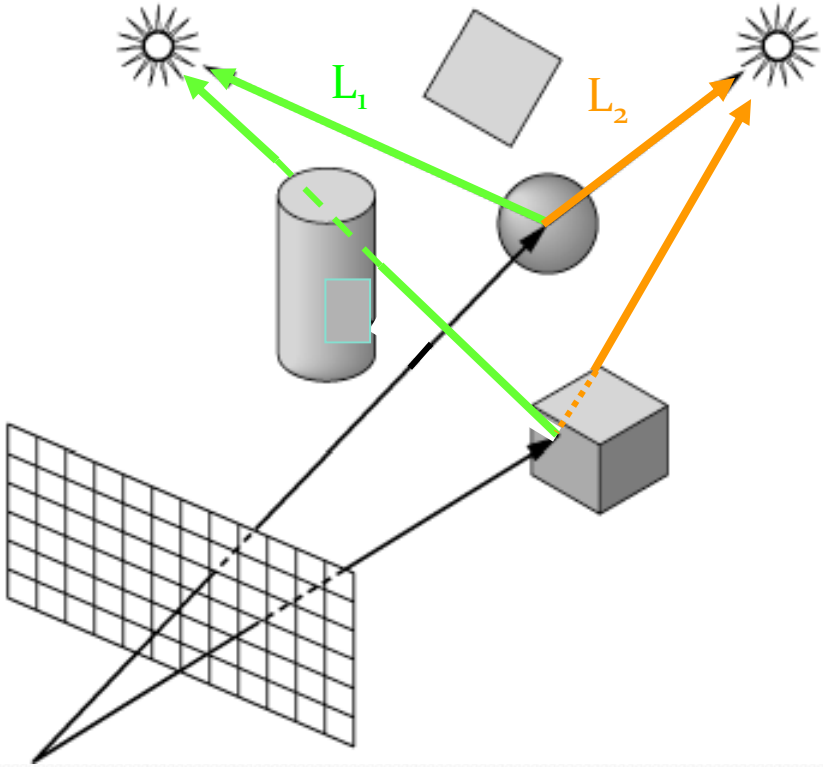
- And each of the Refraction and Reflection rays are also a summation of others



First term: PIE

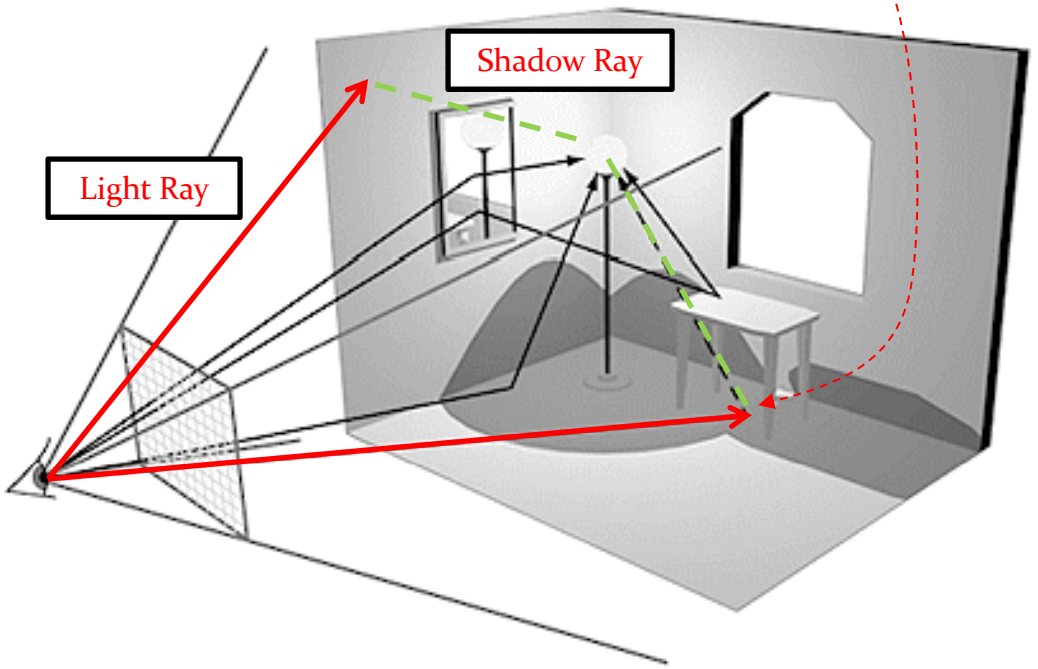
$$I_{eye} = I_{Phong} + k_s I_r + k_t I_t$$

- When the light ray hits a point on a surface, we could calculate its PIE to determine its color.
- However, BEFORE that, we first check the **shadow ray**
  - Draw a shadow ray from the surface point to each of the light source to check if there is anything in between the light and the surface point



Creating Shadow by Shadow Rays

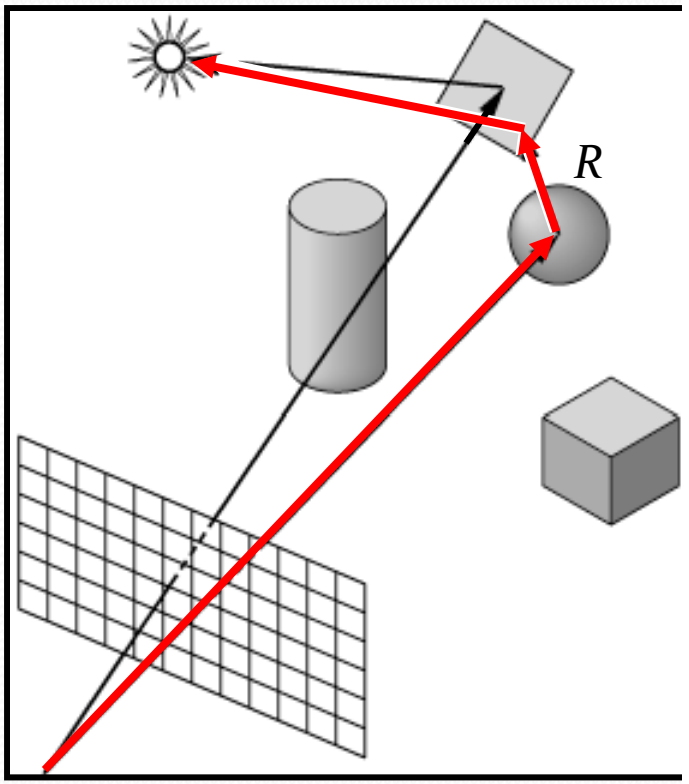
- If the shadow ray hits nothing but the light source, calculate the value PIE and us it as the color of that surface point
  - Else, only color it as the ambient color



$$I_{eye} = I_{Phong} + k_s I_r + k_t I_t$$

Second Term: Reflection

- The second contribution is the light ray following the reflecting direction R
  - The same as the reflection vector in the lecture of illumination and shading
- k_s is the weight of the reflection
- Follow the ray as it bounces from surface to surface until the reflected rays either passes out of the scene or intersects a source.

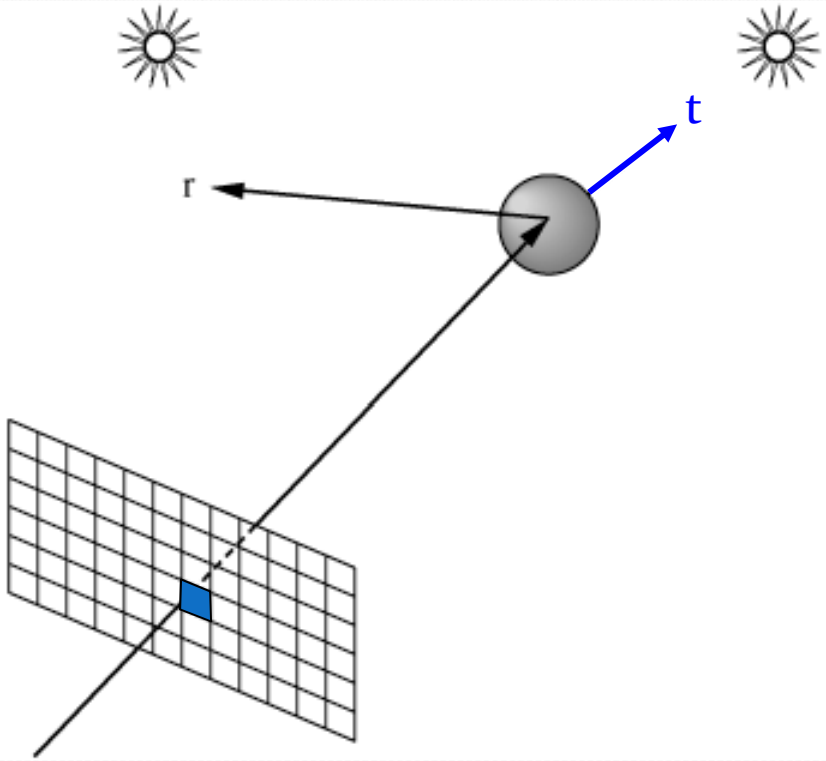


$$I_{eye} = I_{Phong} + k_s I_r + k_t I_t$$

Third Term: Refraction

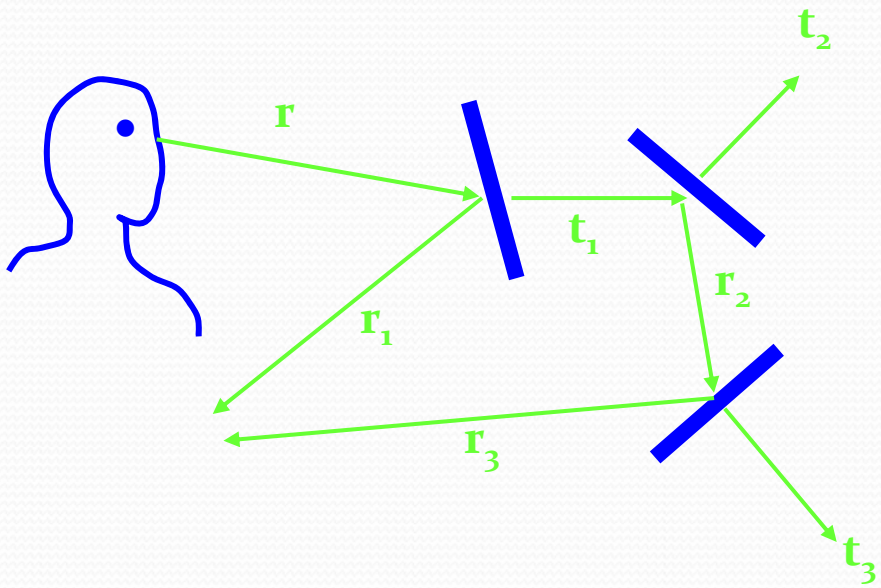
- There may be transparent objects

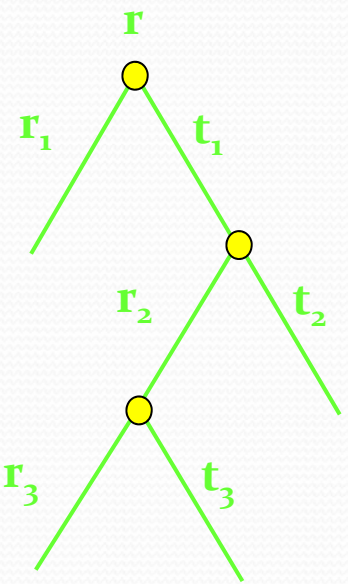
- If the surface is translucent, a refractive ray *t* will be generated, again according to the law of geometric optics .



Ray Tree

- When we trace the ray, the ray may split if the surface is from a transparent object
- Thus the easiest way to describe ray tracing is recursively, through a single function that traces a ray and calls itself for the reflected and refracted rays.

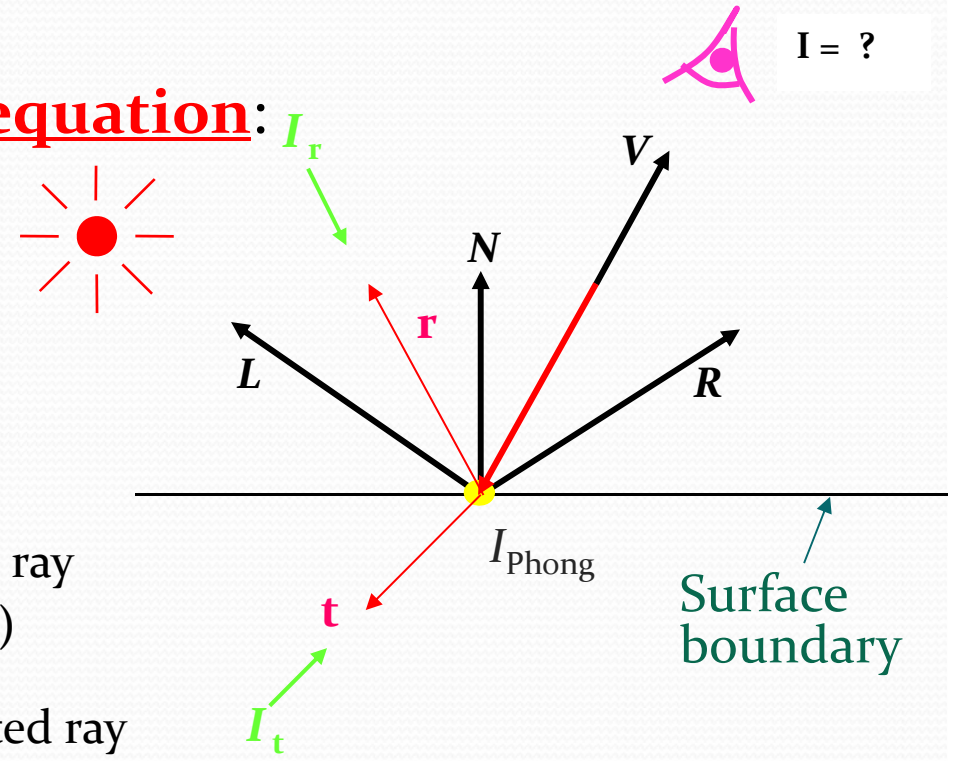




Intensity at Each "Rebound"

- Intensity at an intersection (each node of the tree) is given by the
  - **Whitted's illumination equation**:

- $I = I_{Phong} + k_s I_r + k_t I_t$



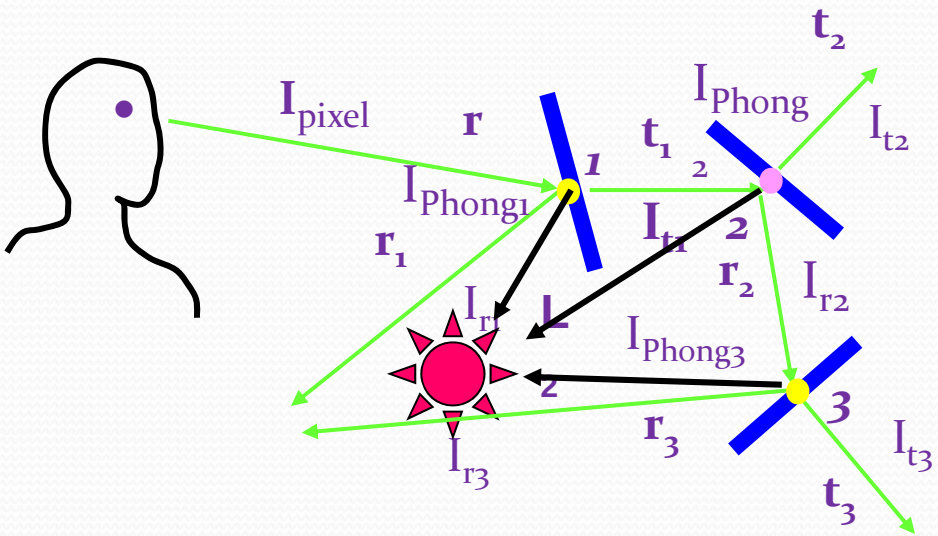
I_r : intensity collected from the reflected ray

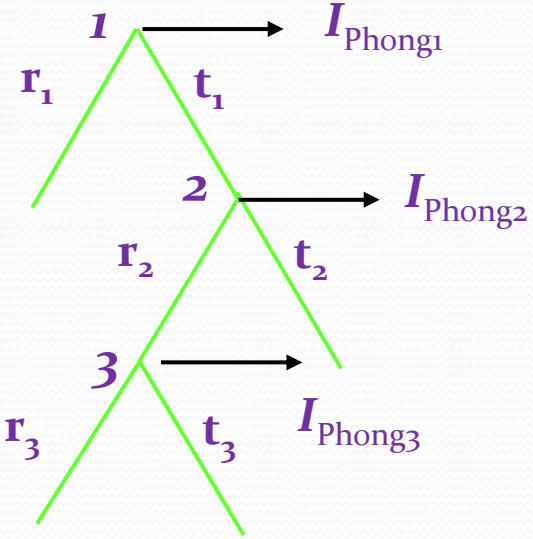
k_s = specular coefficient ($1 \geq k_s \geq 0$)

I_t : intensity collected from the transmitted ray

k_t = transmission coefficient ($1 \geq k_t \geq 0$)

Final Pixel Color





$$I_{r2} = I_{\text{Phong3}} + k_{t3} I_{t3} + k_{s3} I_{r3} .$$

$$I_{t1} = I_{\text{Phong2}} + k_{t2} I_{t2} + k_{s2} I_{r2} .$$

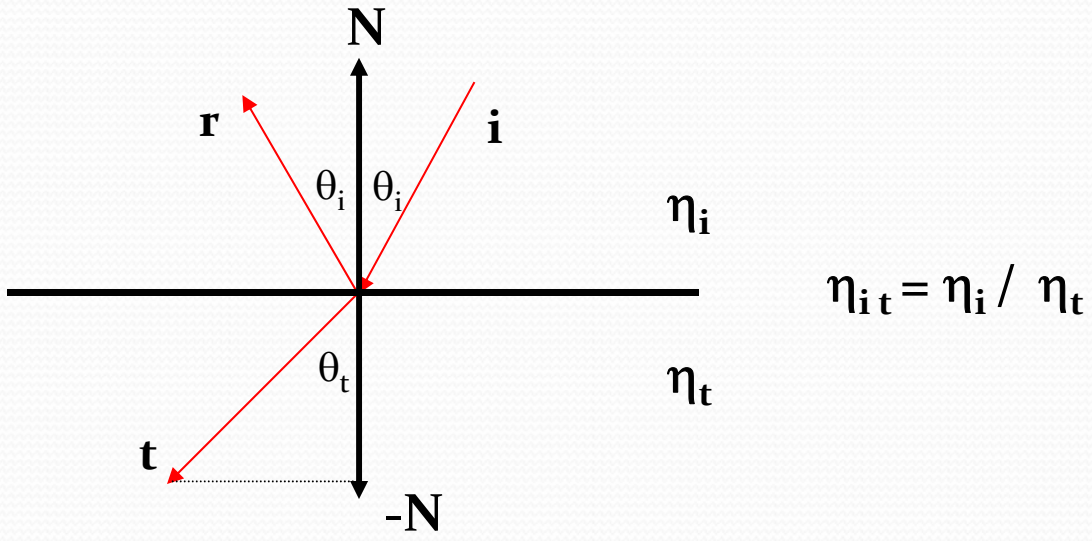
$$I_{\text{pixel}} = I_{\text{Phong1}} + k_{t1} I_{t1} + k_{s1} I_{r1} .$$

Complete ray tree

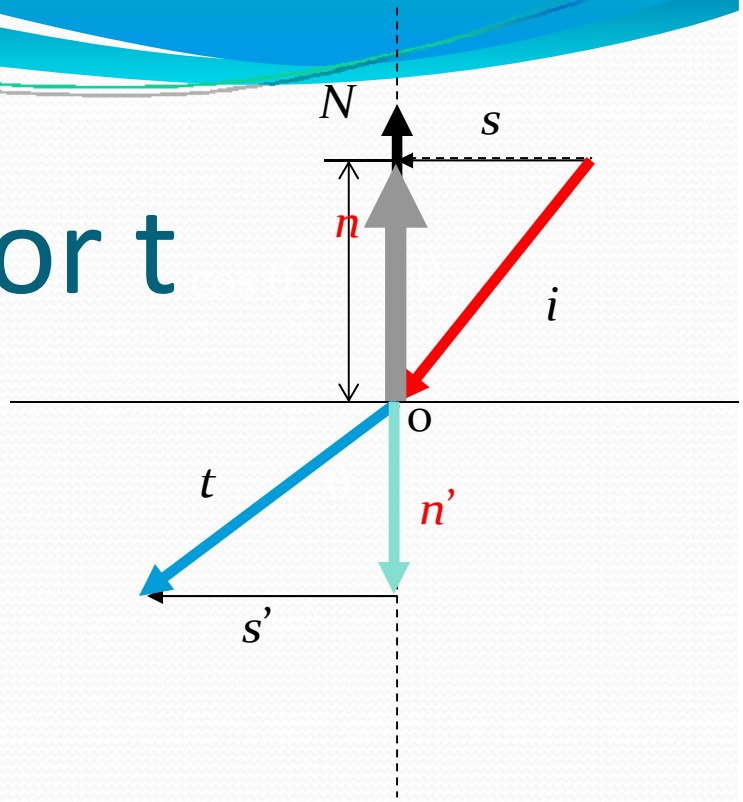
Direction of the Refracted Ray

- Transmitted (or refracted) ray direction vector, t, is given by Snell's law:

$$\sin\theta_i / \sin\theta_t = \eta_t / \eta_i = 1/\eta_{it}$$



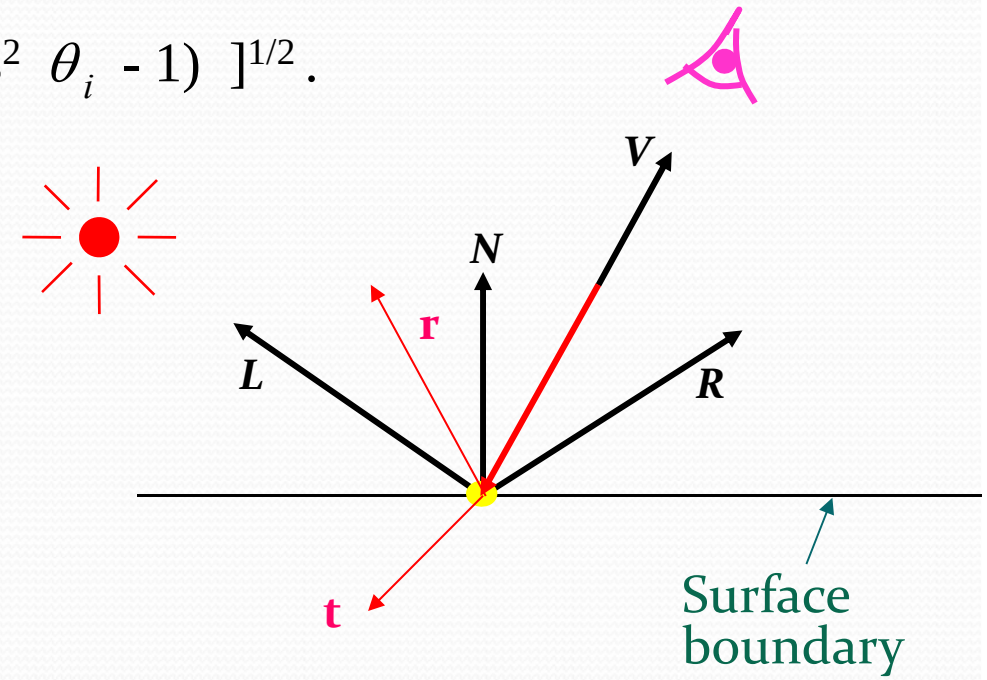
Computing the Vector t



- $\cos\theta_i = N \cdot (-i)$
- $n = N \cos\theta_i$
- $= N\,(N \cdot (-i))$
- $s = i + n$
- $= i + N(N \cdot (-i))$
- $|s| = \sin\theta_i$
- $|s'| = \sin\theta_t$
- $s' = (\sin\theta_t / \sin\theta_i)\, s$
- $= \eta_{it}\ (i + N(N \cdot (-i)))$
- $n' = -N \cos\theta_t$
- $t = n' + s'$
- $= \eta_{it}\, i + (\eta_{it}\,(N \cdot (-i)) - \cos\theta_t)N$

Refraction/Reflection Vectors

- In vector form:
  - $t = \eta_{it}\, i + [\, \eta_{it}\, (-N \cdot i\,) - \cos\theta_t\,]\, N$
  - Where
    - $\cos\theta_i = -N \cdot I$
    - $\cos\theta_t = [\, 1 + \eta_{it}^2\, (\cos^2\theta_i - 1)\,]^{1/2}$.
- $r = 2\,(N \cdot (-i)\,)\, N + i$

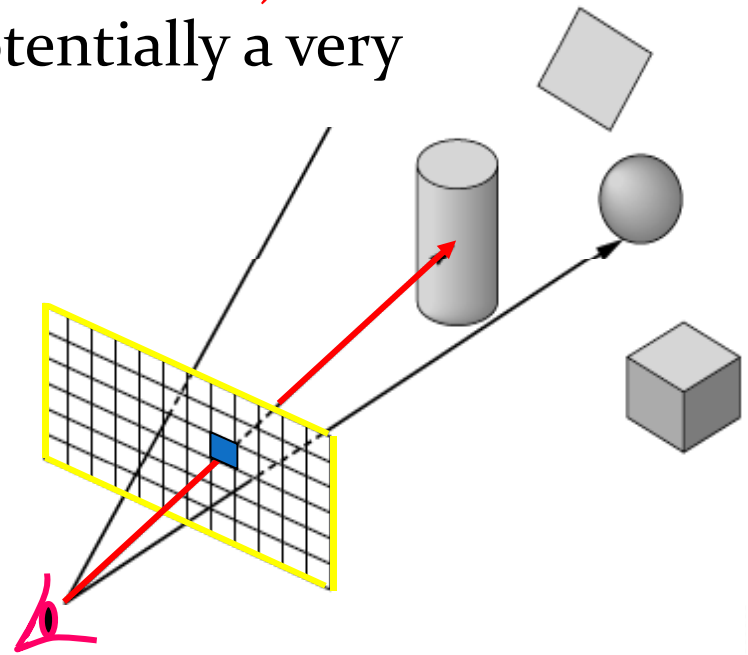


Ray-Object Intersection

For all the rays, how do I know if a ray hit an object

Ray-objection Intersection

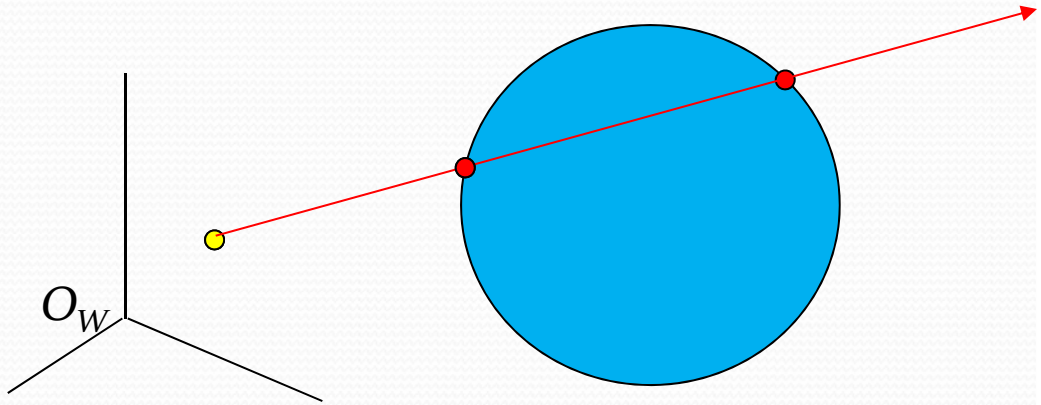
- For the red ray below, how do we know which object will the ray intersect?
- Intersection testing
  - One single ray must be tested against all objects in the scene for intersections. This is potentially a very expensive calculation
    - Check if it hits the cylinder
    - Check if it hits the ball
    - Check if it hits the cube
    - ..... Etc.



Ray-objection Intersection

Check if the ray hits a single object

- Step 1:
  - Setup the light ray equation in a parametric form in t
- Step 2:
  - Substitute the above equation into the implicit equation of an object and solve for t.
- The intersection will be the smallest positive value of t

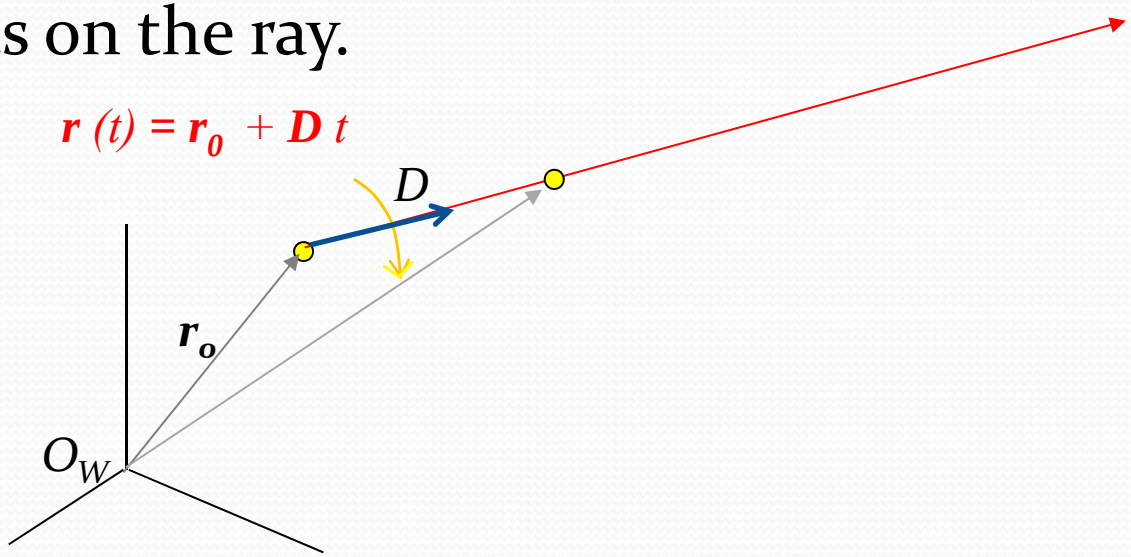


Step 1: Light Ray Equation

- Express the ray using a parameter t :

$$r(t) = r_0 + D\,t$$

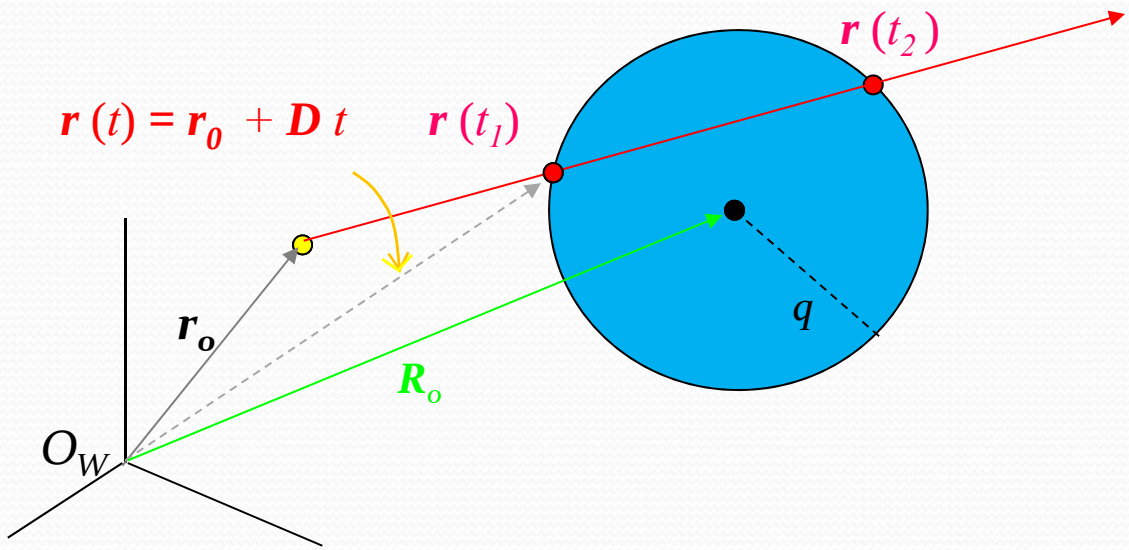
- $r_0 = r(0) = (x_0, y_0, z_0)$: origin of the ray,
- D : direction vector of the ray, typically normalized,
- $r(t)$: a set of points on the ray.



Example: Intersect with a Sphere

- Define a sphere with center $R_0 = (X_0, Y_0, Z_0)$ and radius q.
- Points lying on the sphere is given by:

$$(x - X_0)^2 + (y - Y_0)^2 + (z - Z_0)^2 = q^2 \qquad (2)$$



Solving the Equation

- Substituting (1) into (2), the values of t at the points of intersection are obtained by solving the quadratic equation:

$$\alpha t^2 + \beta t + \gamma = 0 \qquad (3)$$

- Where
  - $\alpha = D \cdot D$
  - $\beta = 2 D \cdot (r_0 - R_0)$
  - $\gamma = (r_0 - R_0) \cdot (r_0 - R_0) - q^2$
    $= R_0^2 + r_0^2 - 2 r_0 \cdot R_0 - q^2$

Possibilities of Intersection

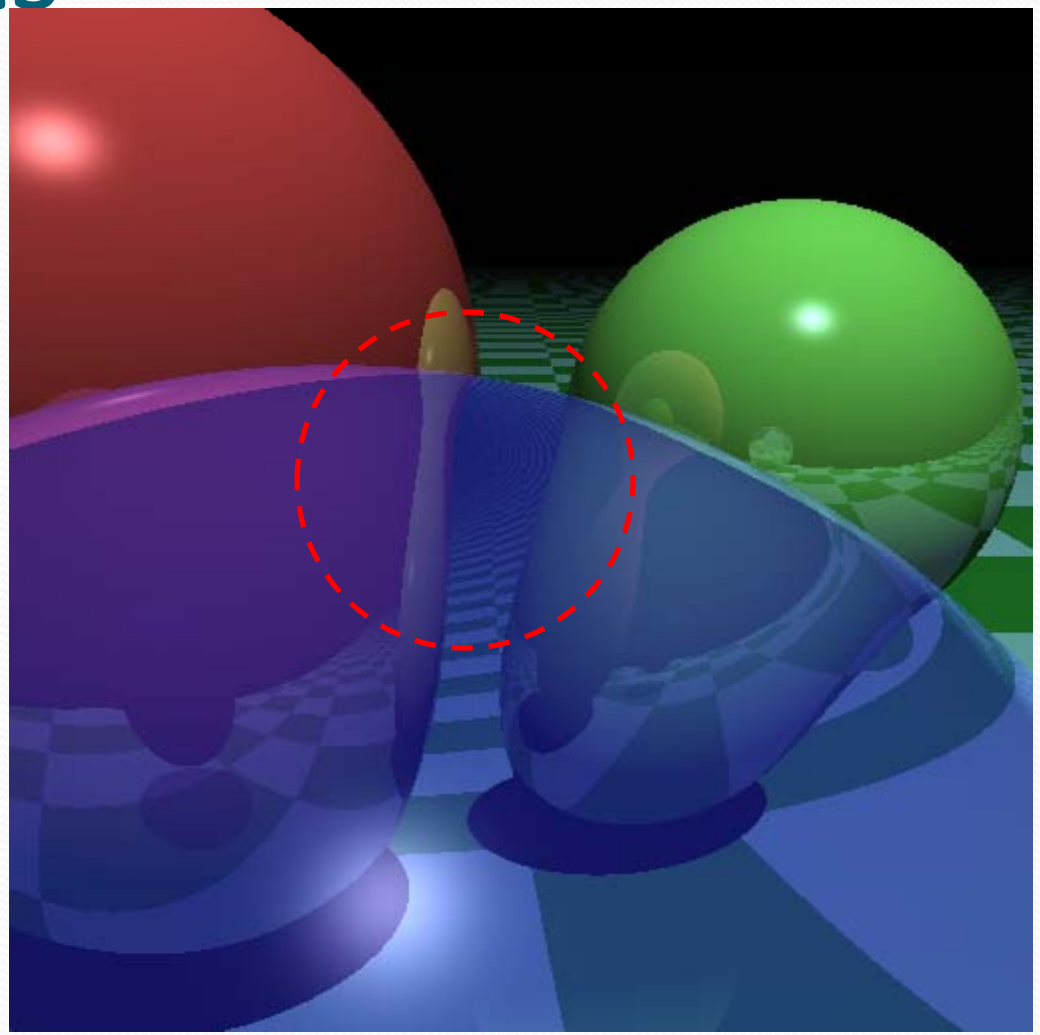
- If Eqn (3) has
  - 2 roots : smaller positive root gives the nearer intersection.
  - 1 root : the ray is tangential to the sphere. Consider no intersection.
  - Imaginary roots : no intersection.
  - All negative root(s): intersection(s) are behind ray origin. (Also considered as no intersection.)

Conclusion of Ray Tracing

- Ray tracing is a simple powerful rendering technique for obtaining highly realistic images that exhibits
  - Global reflection, transmission and shadow effects.
- But its drawbacks are:
  - requires considerable computation time to generate.
    - (mainly spent in ray-object intersection tests)
    - We will “experience” it in our tutorials
  - Images have severe aliasing effect, just like z-buffer algorithm.
  - Too clear
    - We could use motion blurring, i.e. shooting rays at a slighting different time and angle

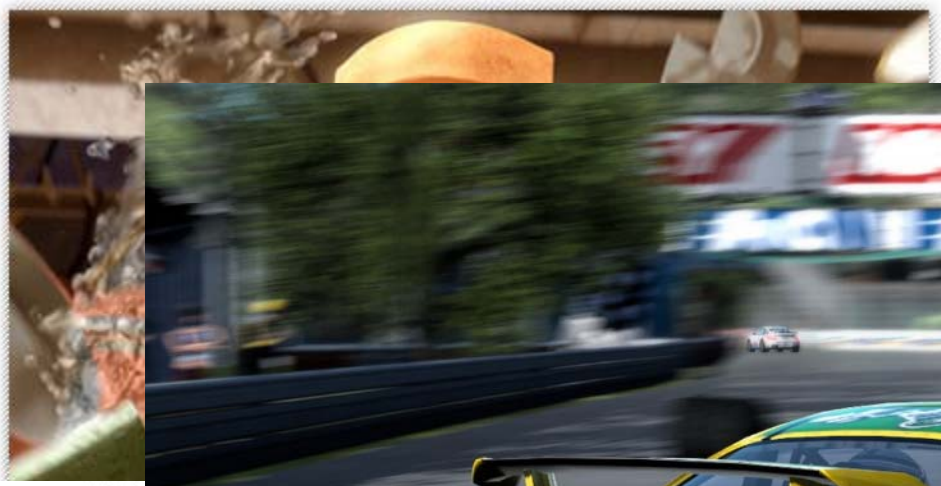
Aliasing Effects

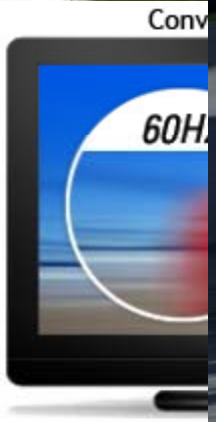
- (Discussed in the second last lecture)



Motion Blurring

- “Clear” vs “blur”, a choice to be made





LG’s TruMotion 2
detail even in fast moving sports and movie scenes.

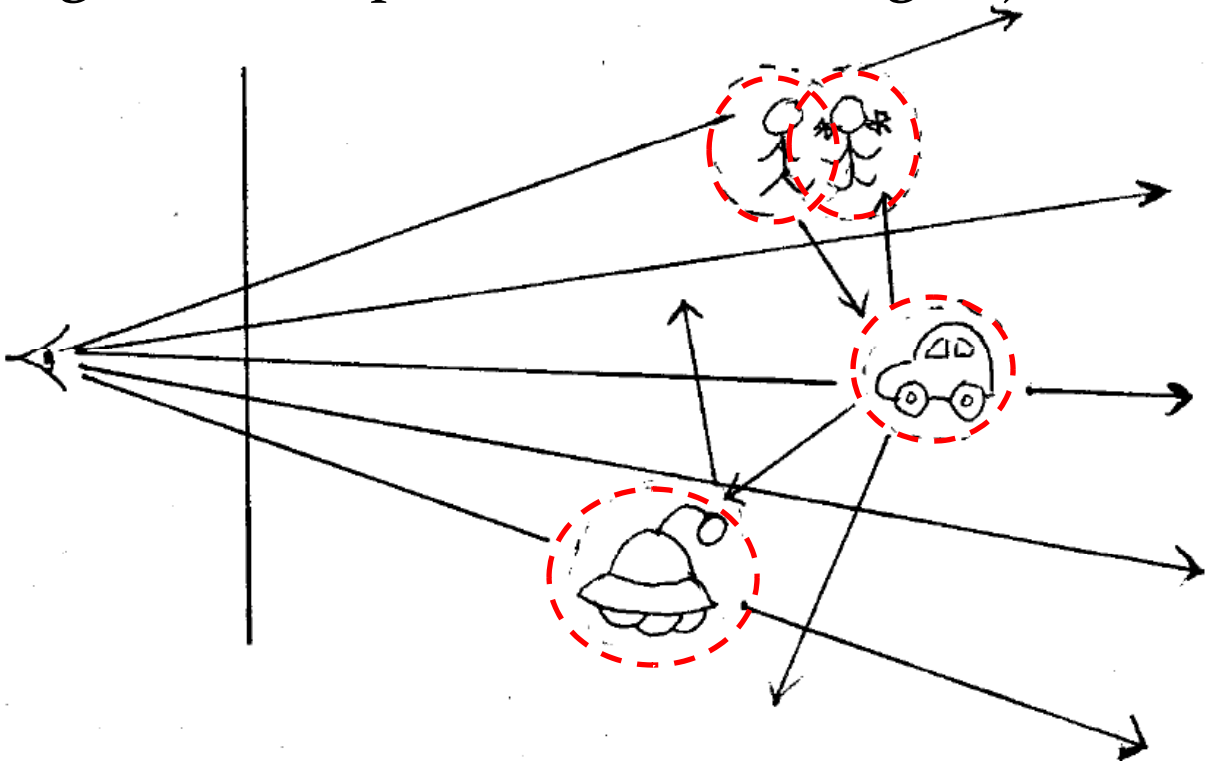
Speed Improvement

Speed Improvement

- Speed:
  - Avoiding intersection calculations
  - Hierarchies
  - Spatial partitioning
  - Grids, oct-tree

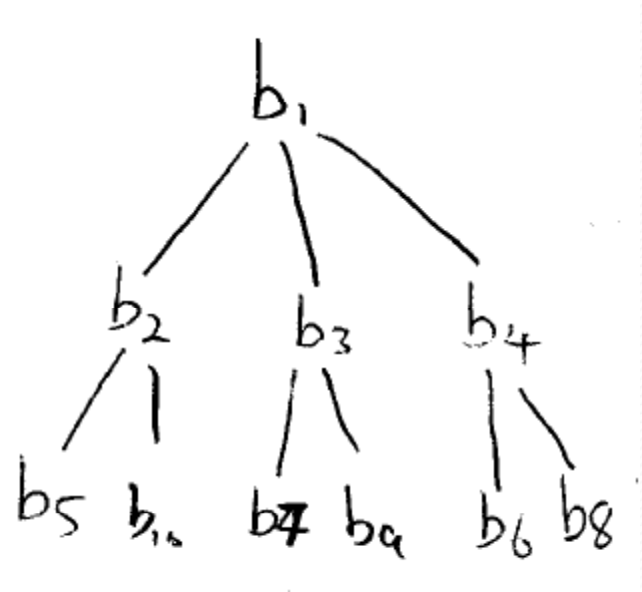
Speed Improvement

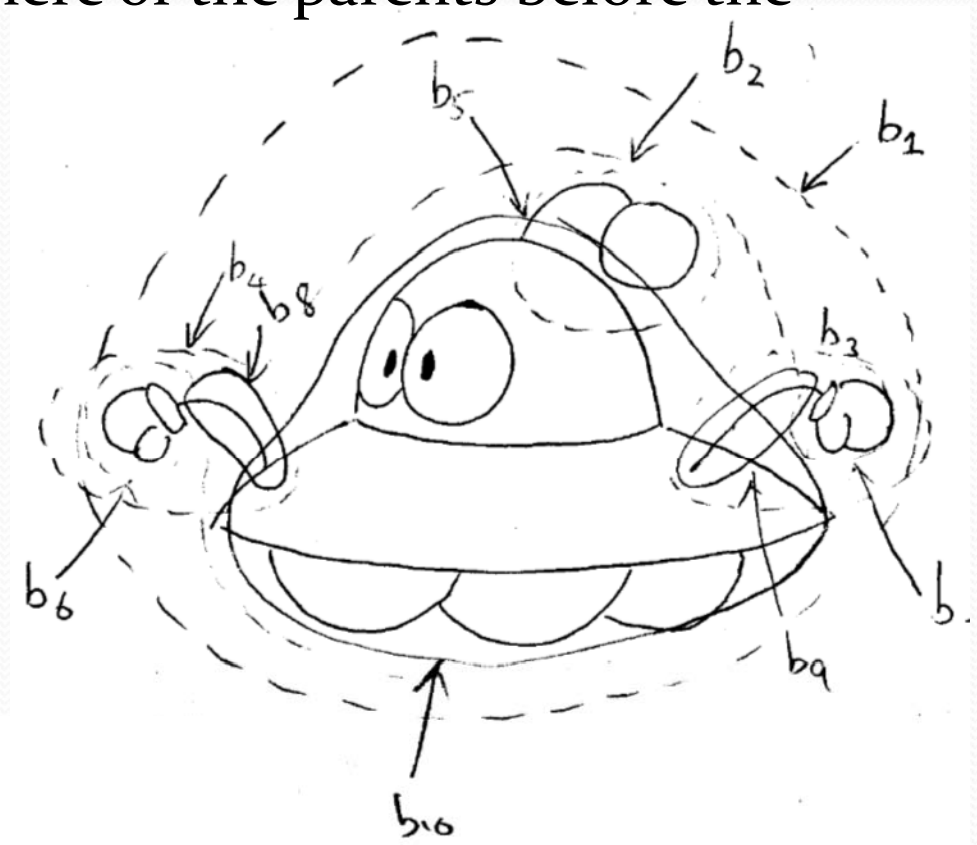
- Avoiding intersection calculation
  - Compute a bounding sphere for a group of objects
  - Test against the sphere before testing objects within



Speed Improvement

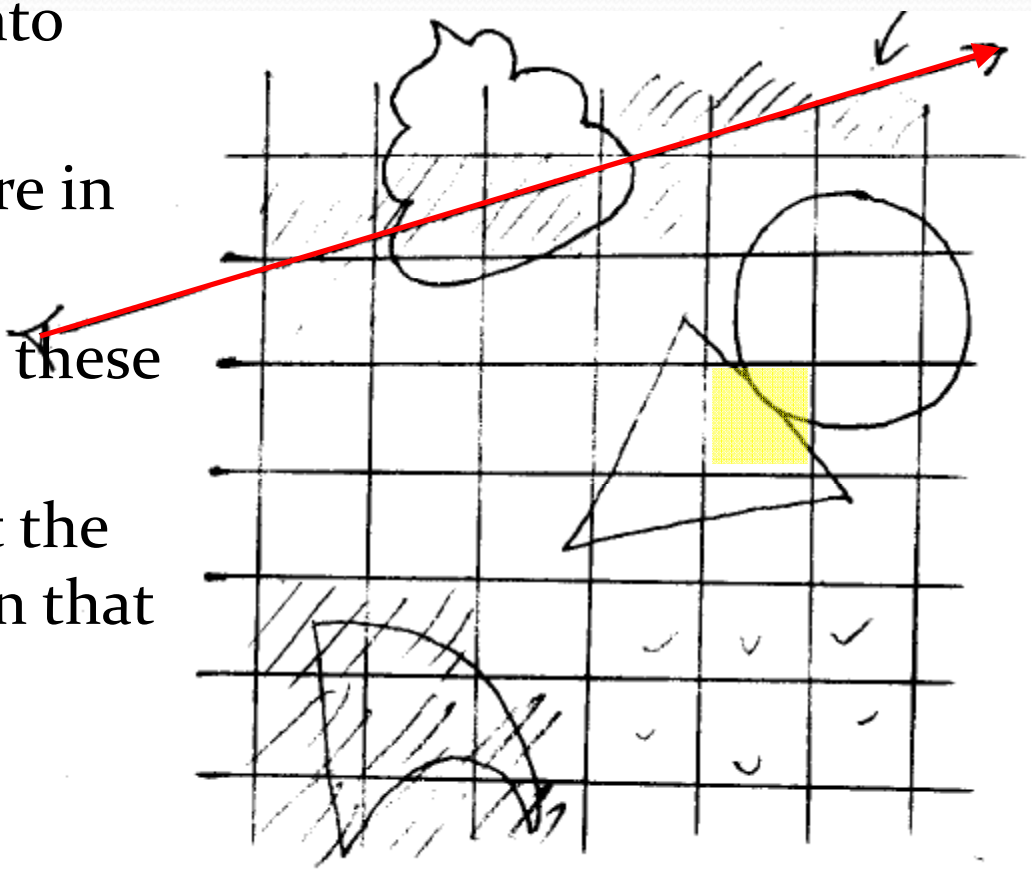
- Boundary spheres for object hierarchy
  - Check the boundary sphere of the parents before the children





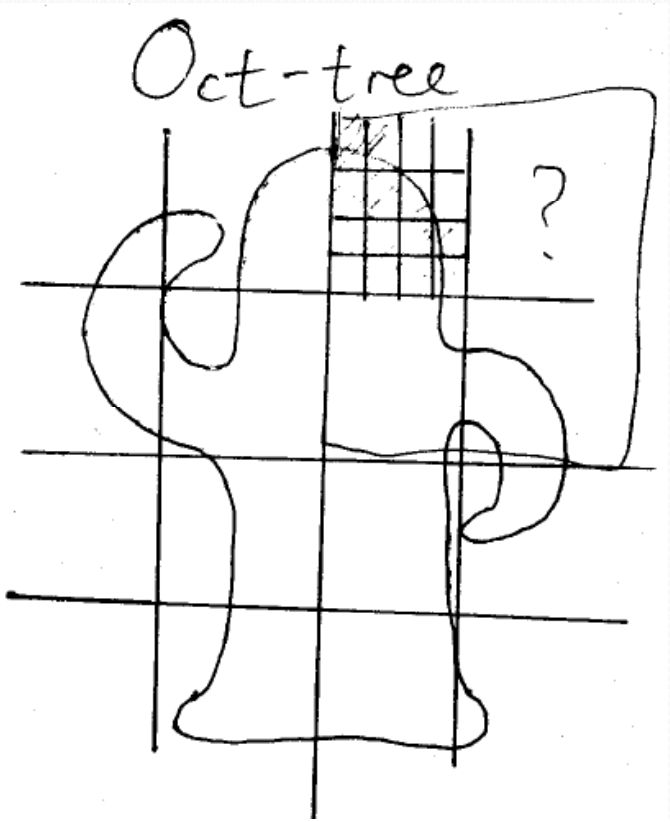
Speed Improvement

- Spatial partitioning
  - First divide the space into small cubes
  - Record which objects are in which cubes
  - A ray will travel though these cubes
  - For each cube, only test the ray with those objects in that cube



Speed Improvement

- Spatial partitioning
- Evenly distributed cubes are wasteful
  - A lot of empty cubes
- Use Oct-tree instead



The End of Rasterization

- If ray-tracing become realtime
  - Imagine no more SCA, projections, etc..
- Real-time ray tracing
  - OpenRT
  - http://igad.nhtv.nl/~bikker/
  - http://mpierce.pie2k.com/pages/108.php

Some History of Computer Display

- 1980's
  - 320 x 200, four colors
- 1987
  - VGA: 640 x 480, 256 colors
- 1990
  - UXGA: 1600 x 1200,
  - 10,000,000 colors
- 20xx?
  - Real time raytracing?

Admin

- Lab 4 deadline this Sat
  - No FB voting, but you can still post your creation onto FB
- Lab 5 next week
  - Will have FB voting