# Ray Tracing

Forward & Backward Ray Tracing
Ray Tracing
Ray-Surface Intersection Testing
Shadows
Reflections
Transmission
Next time: efficient ray tracing

Shirley Chapter 10

# Ray Tracing

http://www.openrt.de/Gallery/2002_DynamicRayTracing/Images/teas_kitchen1.jpg

# Ray Tracing

eal-time Ray traced Bugatti Veyron with reflected reflections, adaptive FSAA and realistic shadows. [Source: Nvid

# Object-oriented vs. Pixel-oriented Rendering

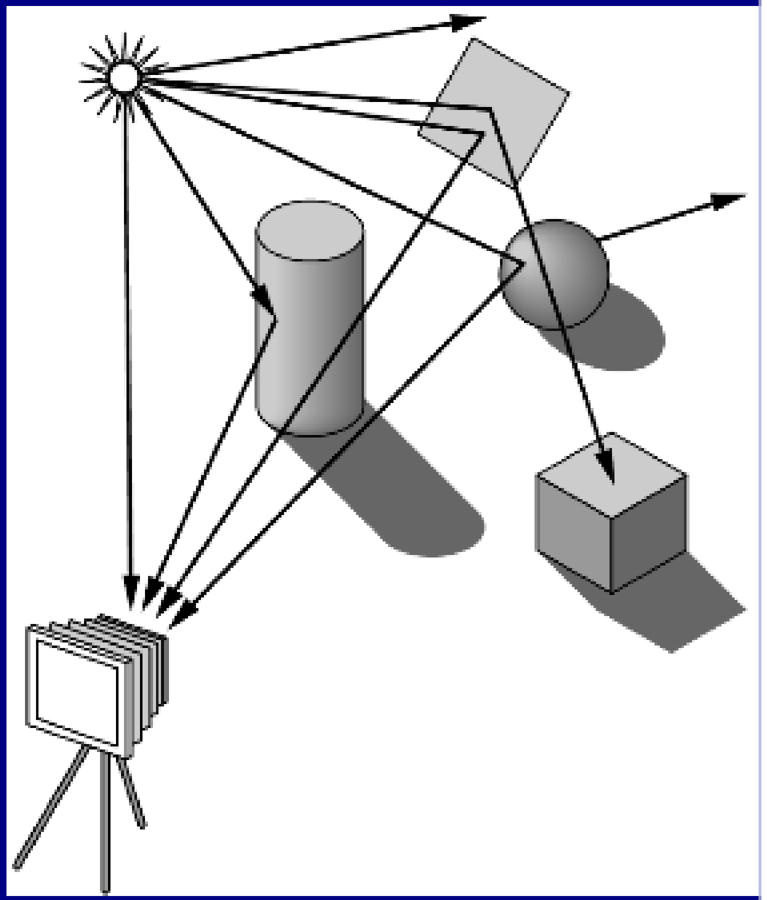
- OpenGL rendering:
  - walk through objects, transforming and then drawing each one unless the z buffer says that it is not in front
- Ray tracing
  - walk through each pixel looking for what object (if any) should be shown there

# Light is Bouncing Photons

- Light sources send off photons in all directions
  - Model these as particles that bounce off objects in the scene
  - Each photon has a wavelength and energy (color and intensity)
  - When photons bounce, some energy is absorbed, some reflected, some transmitted
- If we can model photon bounces we can generate images
- Technique: follow each photon from the light source until:
  - All of its energy is absorbed (after too many bounces)
  - It departs the known universe (not just the part of the world that is within the viewing volume!)
  - It strikes the image and its contribution is added to appropriate pixel

# Forward Ray Tracing

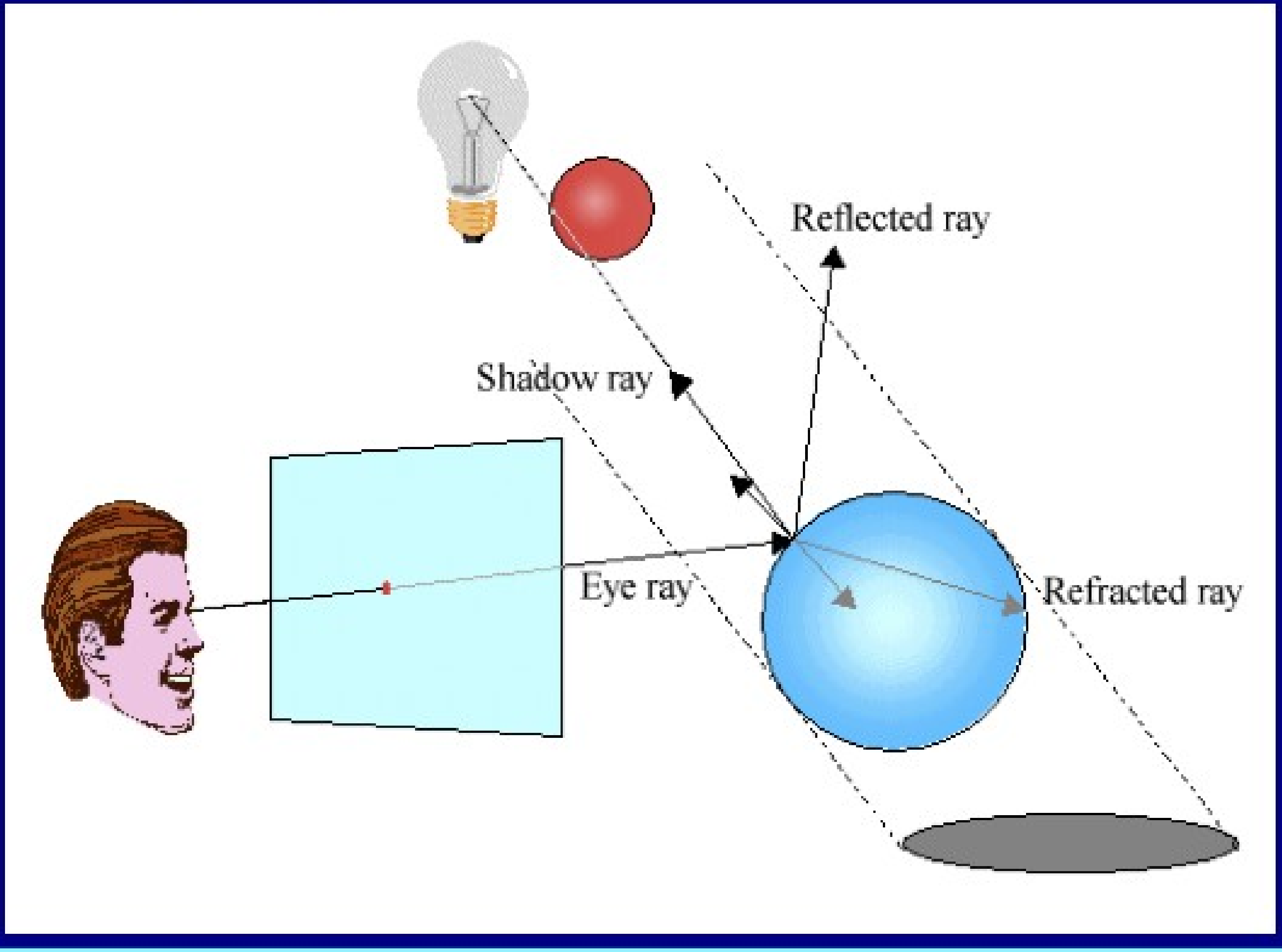
- Rays are the paths of these photons
- This method of rendering by following photon paths is called *ray tracing*
- *Forward* ray tracing follows the photon in direction that light travels (from the source)
- BIG problem with this approach:
  - Only a tiny fraction of rays reach the image
  - Many, many rays are required to get a value for each pixel
- Ideal Scenario:
  - We'd like to magically know which rays will eventually contribute to the image, and trace only those



# Backward Ray Tracing

The solution is to start from the image and trace backwards—*backward* ray tracing

Start from the image and follow the ray until the ray finds (or fails to find) a light source



# Backward Ray Tracing

Basic idea:

- Each pixel gets light from just one direction—the line through the image point and focal point
- Any photon contributing to that pixel's color has to come from this direction
- So head in that direction and see what is sending light
  - If we hit a light source—done
  - If we find nothing—done
  - If we hit a surface—see where that surface is lit from

At the end we've done forward ray tracing, but ONLY for the rays that contribute to the image

# Ray Tracing

The basic algorithm is

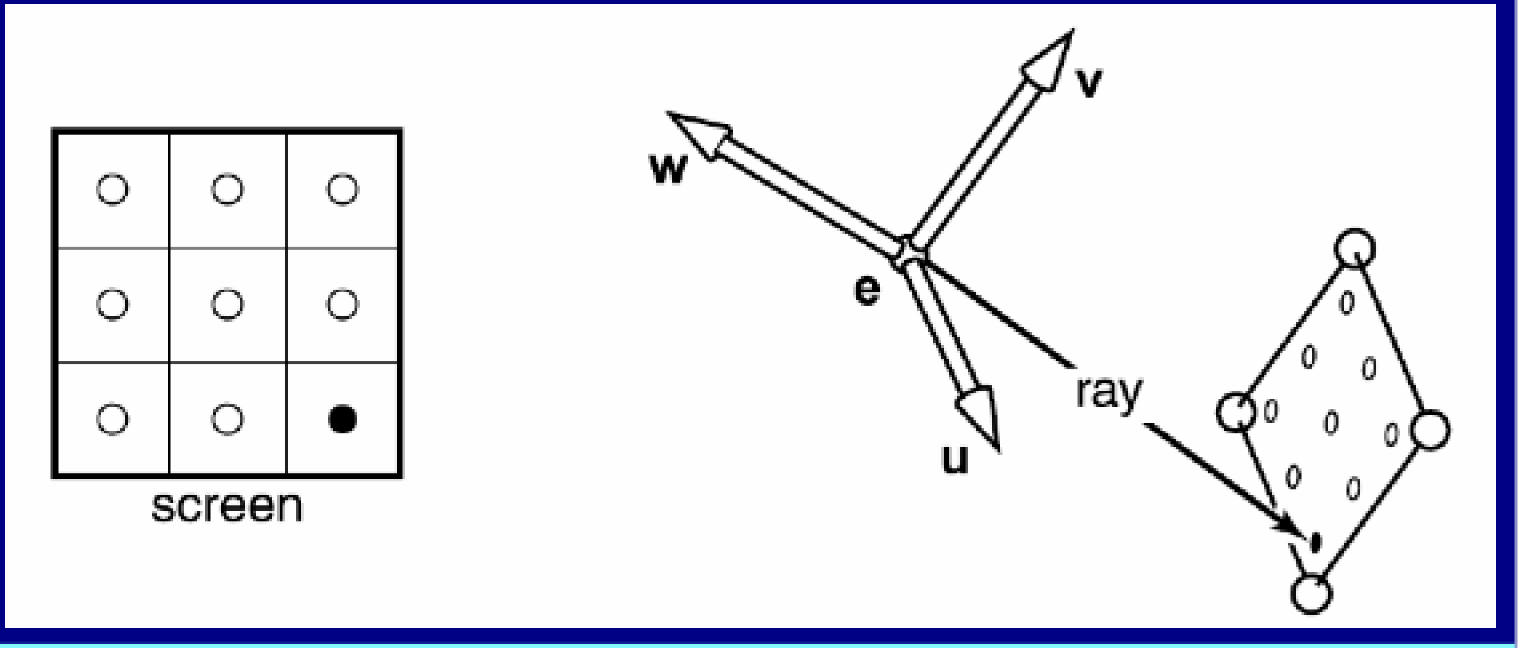
compute u, v, w basis vectors

for each pixel do

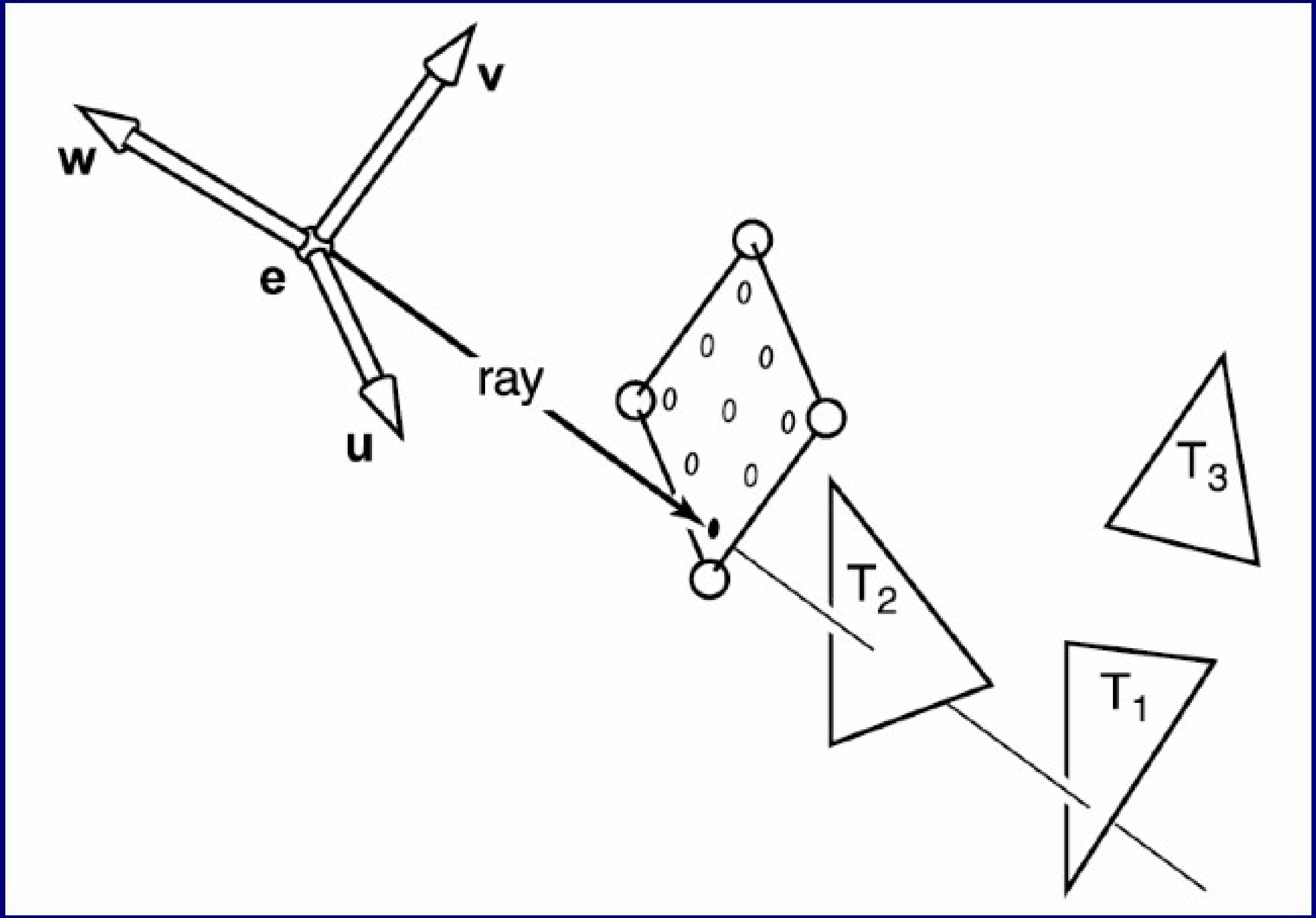
shoot ray from eye point through pixel (x,y) into scene

intersect with all surfaces, find first one the ray hits

shade that point to compute pixel (x,y)'s color

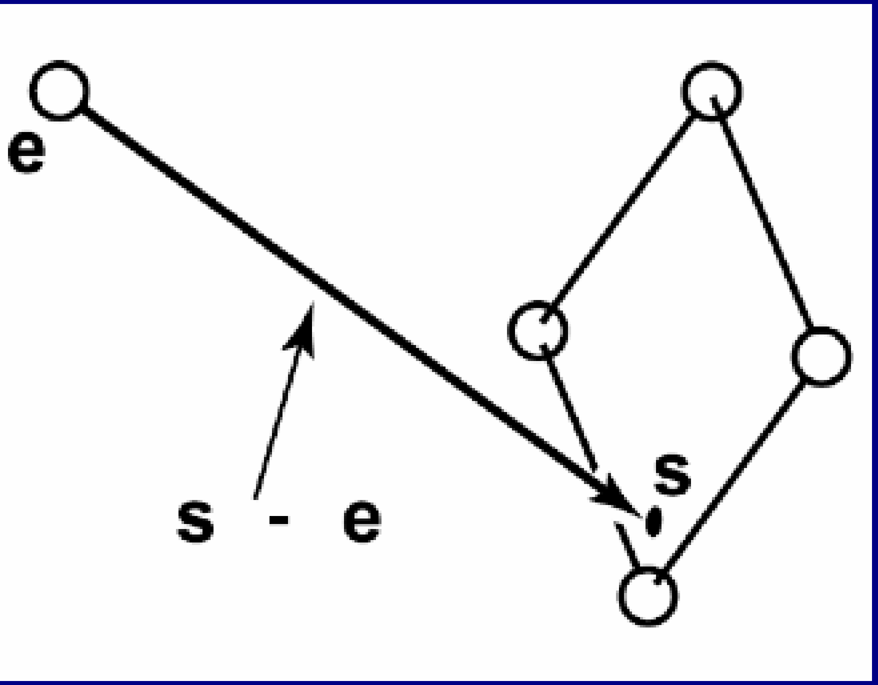


# Ray Tracing



# Computing Rays

$$p(t) = e + t(s - e)$$




$t = 0$ origin of the ray

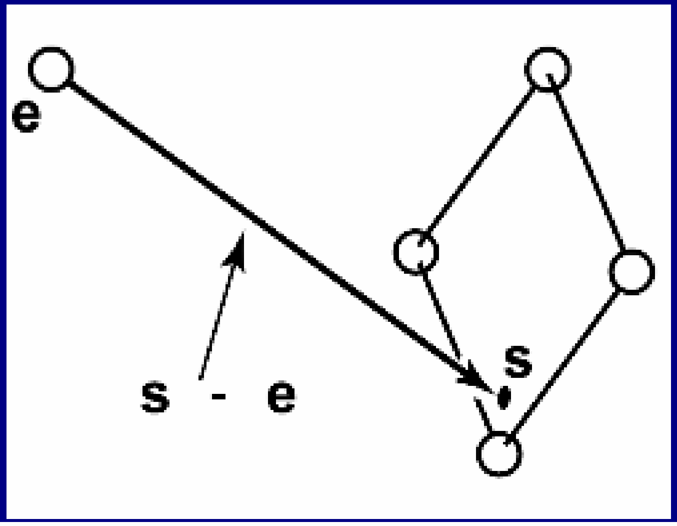
$t > 0$ in positive direction of ray

$t < 0 \Rightarrow$ then $p(t)$ is behind the eye

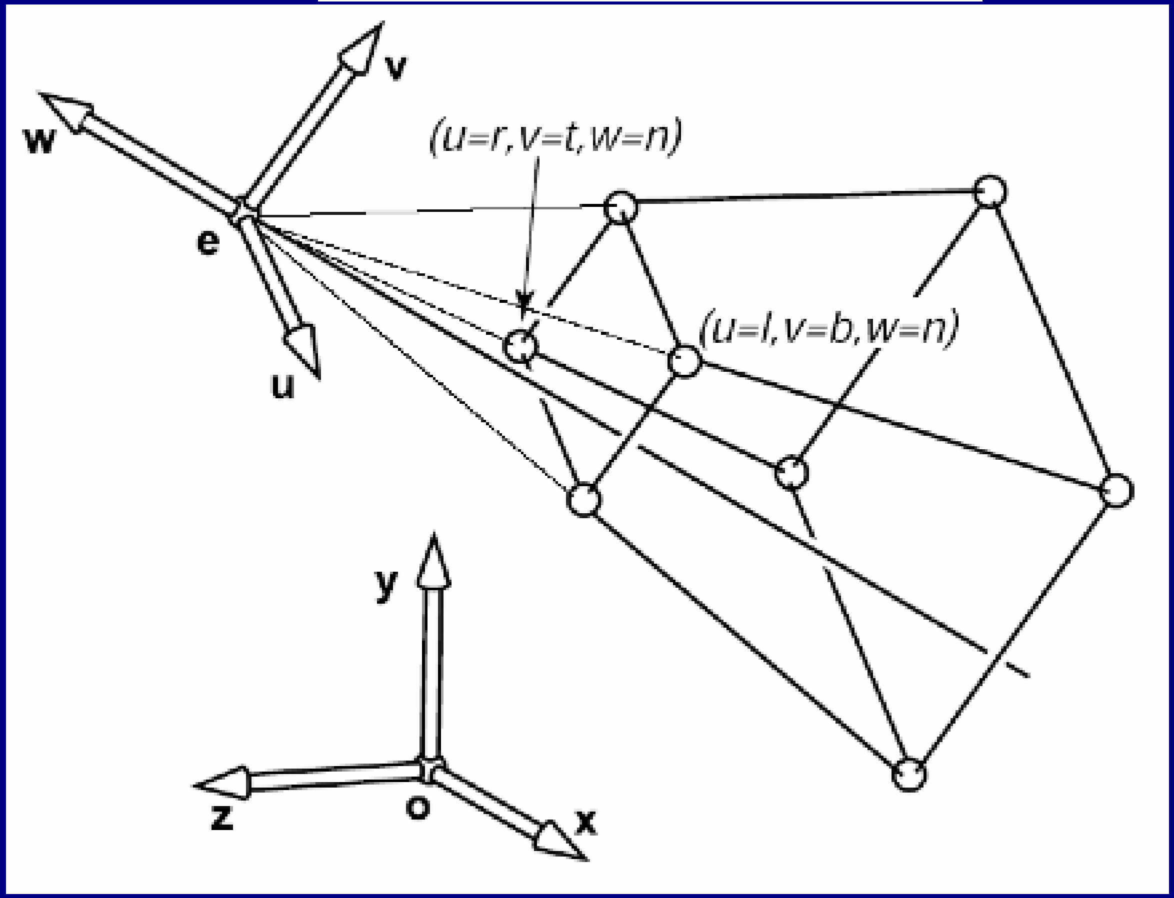
$t_1 < t_2 \Rightarrow p(t_1)$ is closer to the eye than $p(t_2)$

# Computing Rays

Where is s? (x,y of image)
Intersection of ray with image plane

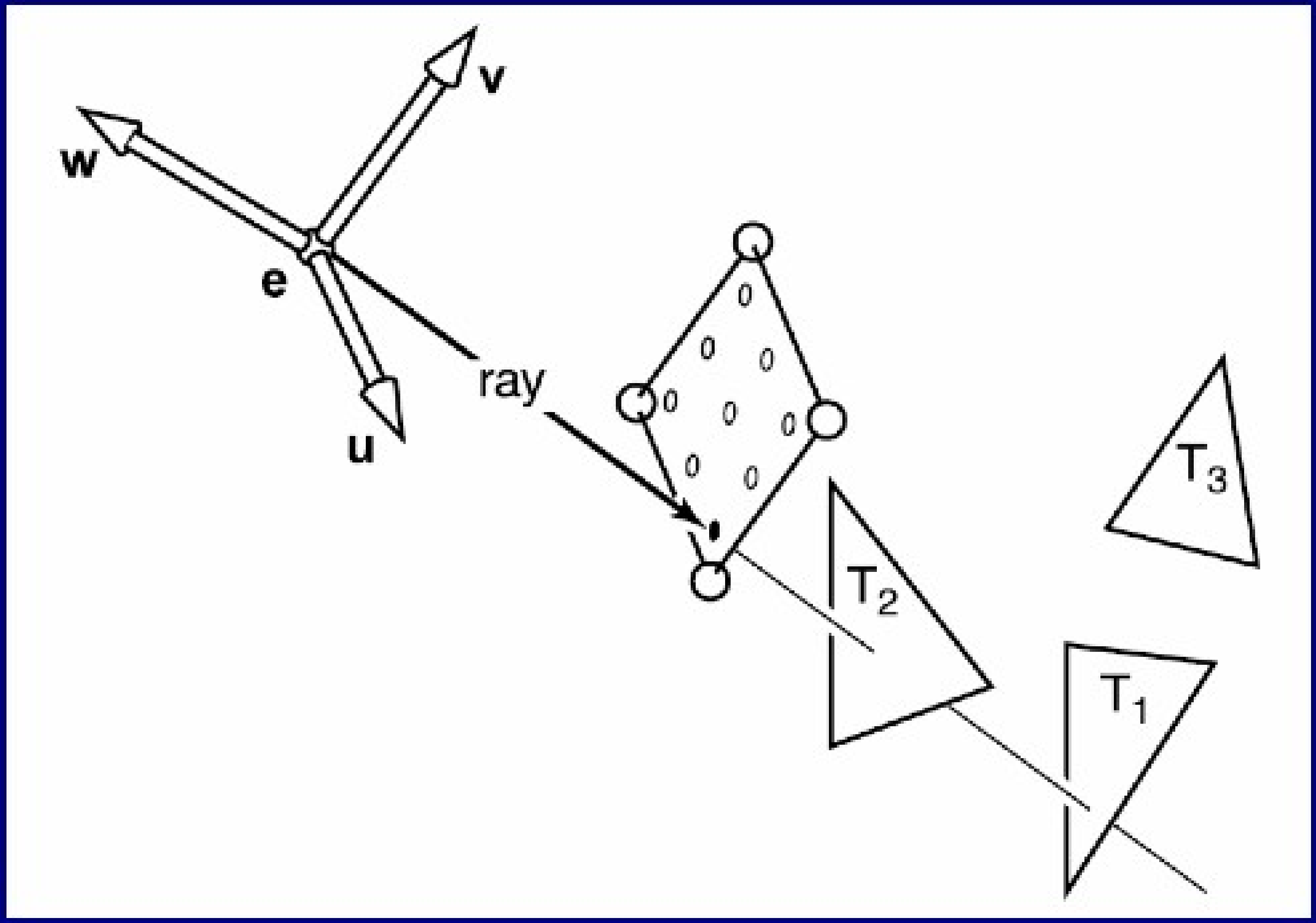



Details in book.
Derived using viewing transformations



# Ray Object Intersection

Sphere
Triangle
Polygon

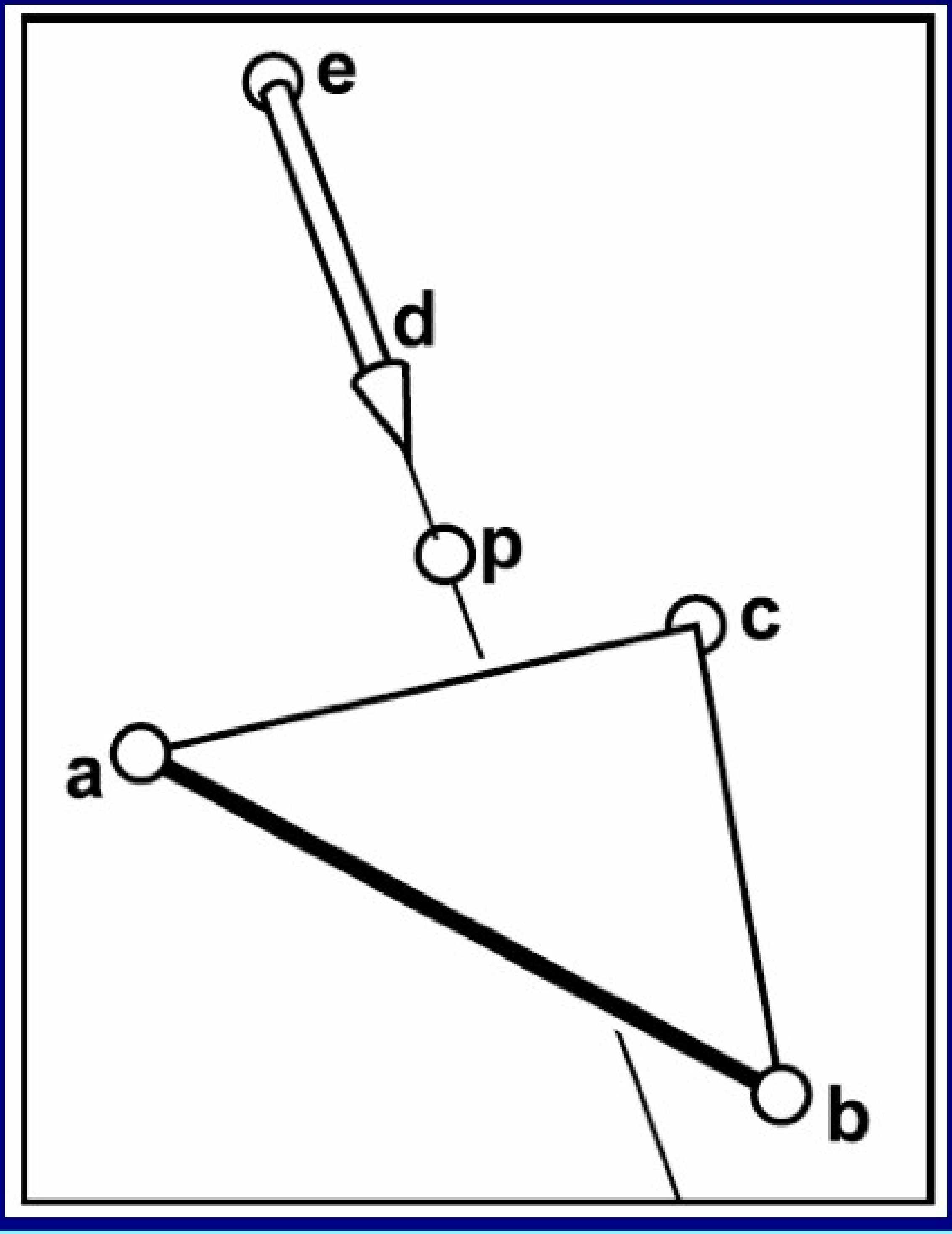



blackboard

# Ray Object Intersection

Sphere
Triangle
Polygon

**blackboard**



# Ray Object Intersection

Sphere
Triangle
Polygon

Ray-polygon—in book
- Intersection with plane of polygon
- in/outside of polygon determination

Ray-triangle—3D models composed of triangles

Ray-sphere—early models for raytracing, and now bounding volumes

# Global vs. Local Rendering Models

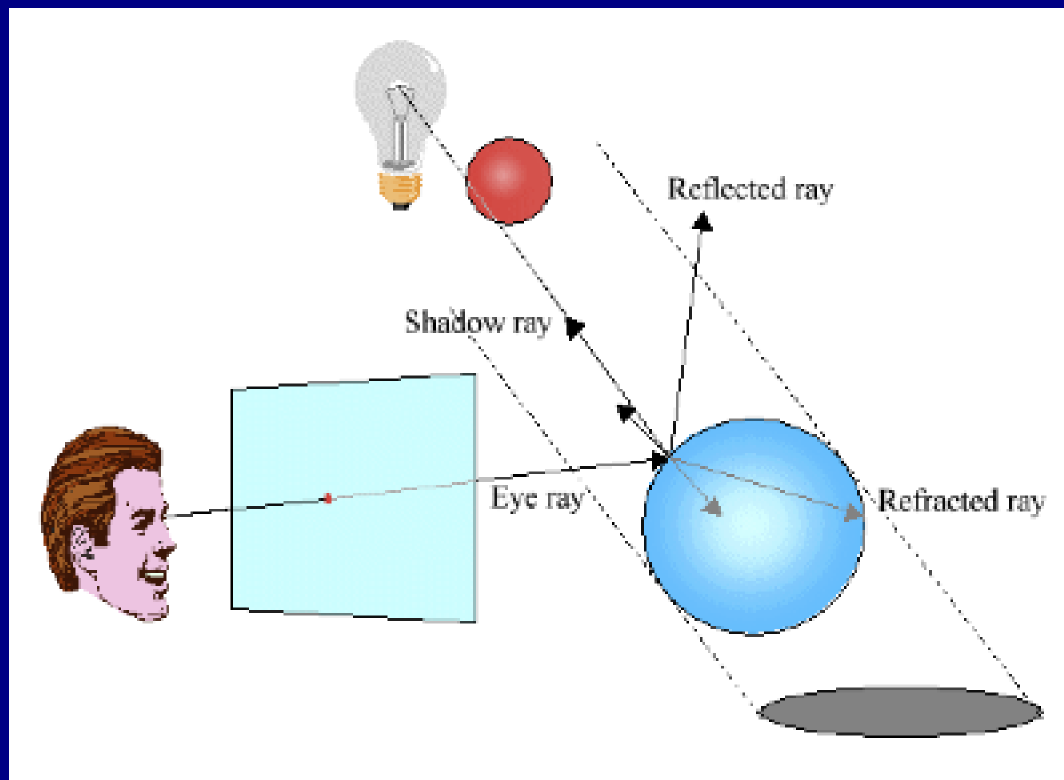
Local rendering models: the color of one object is independent of its neighbors (except for shadows)

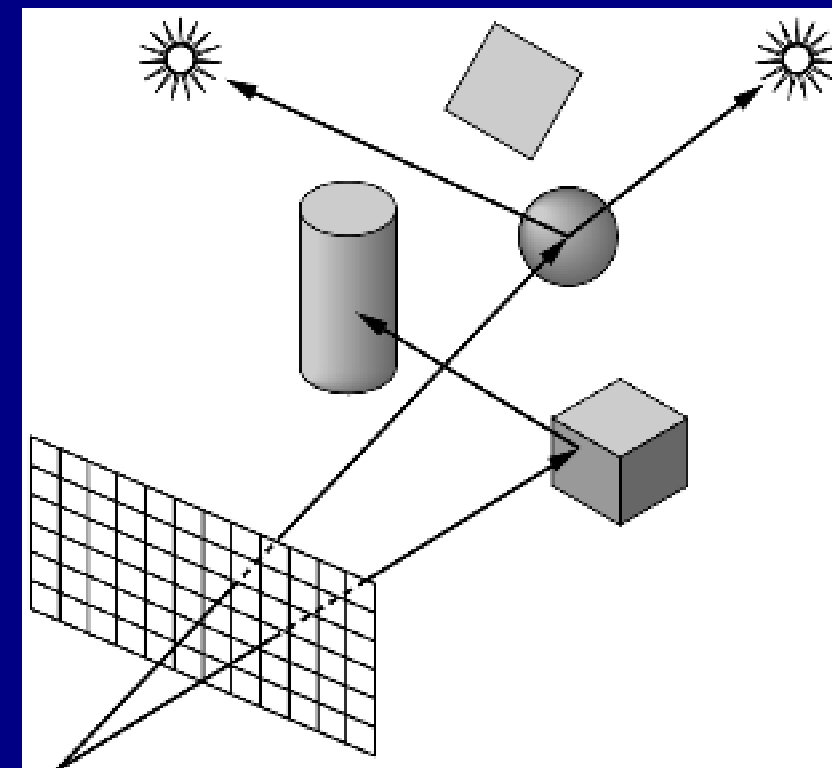
Missing scattering of light between objects, real shadowing

Global Rendering Models

- Raytracing—specular highlights
- Radiosity—diffuse surfaces, closed environments

# Recursive Ray Tracing



Four ray types:

- Eye rays: originate at the eye
- Shadow rays: from surface point toward light source
- Reflection rays: from surface point in mirror direction
- Transmission rays: from surface point in refracted direction

# Writing a Simple Ray Caster (no bounces)

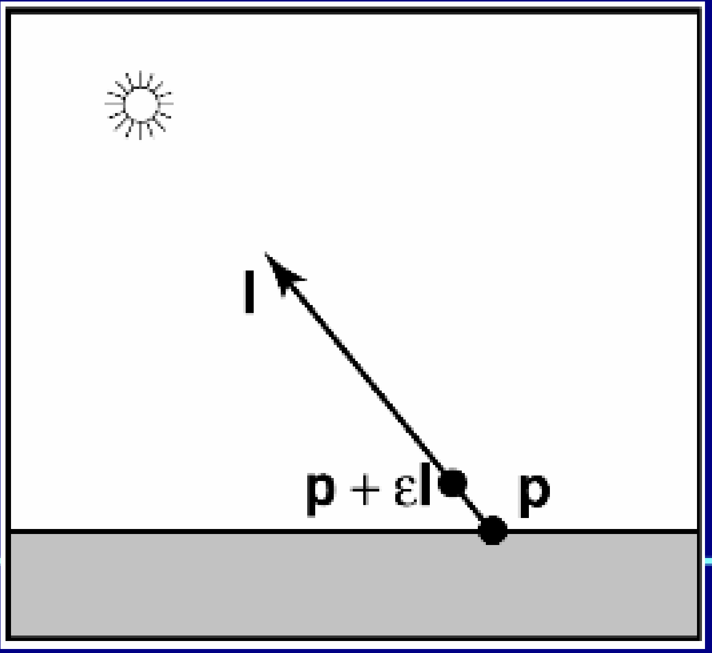
```
Raycast()                   // generate a picture
   for each pixel x,y
      color(pixel) = Trace(ray_through_pixel(x,y))

Trace(ray)                  // fire a ray, return RGB radiance
                            // of light traveling backward along it
   object_point = Closest_intersection(ray)
   if object_point return Shade(object_point, ray)
   else return Background_Color

Closest_intersection(ray)
   for each surface in scene
        calc_intersection(ray, surface)
   return the closest point of intersection to viewer
   (also return other info about that point, e.g., surface
     normal, material properties, etc.)

Shade(point, ray)  // return radiance of light leaving
                            // point in opposite of ray direction
   calculate surface normal vector
   use Phong illumination formula (or something similar)
   to calculate contributions of each light source
```

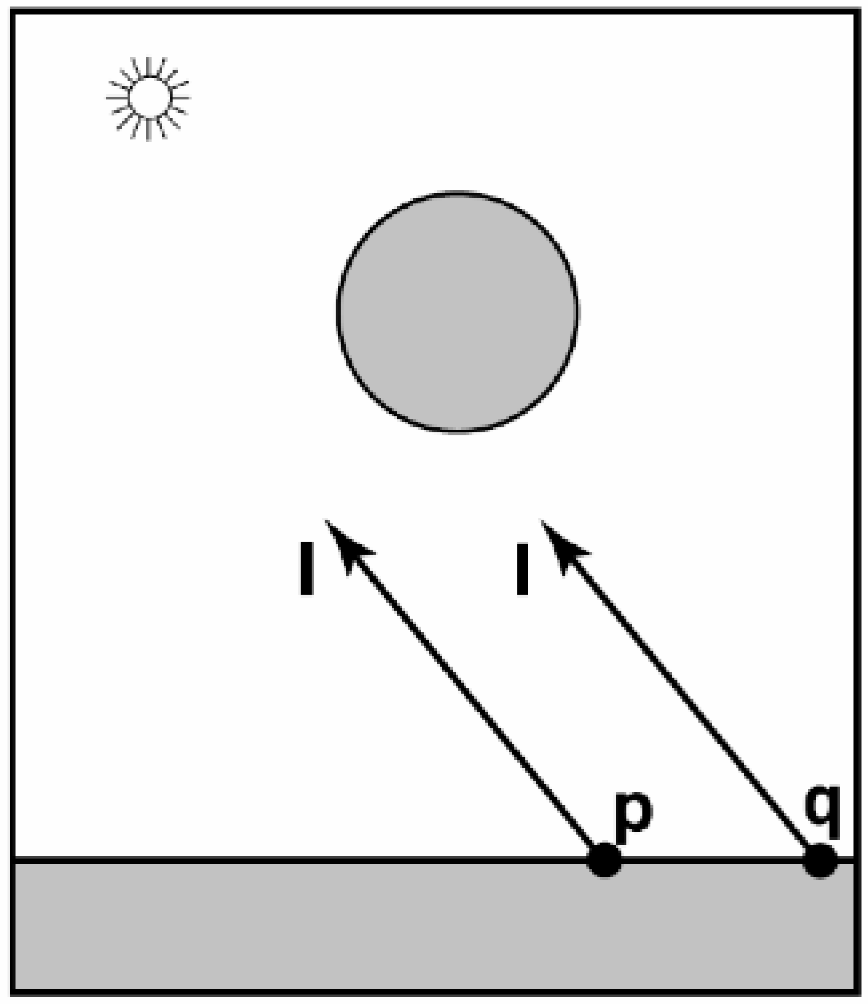
# Shadow Rays

$\mathbf{p} + t\,\mathbf{l}$ does not hit any objects

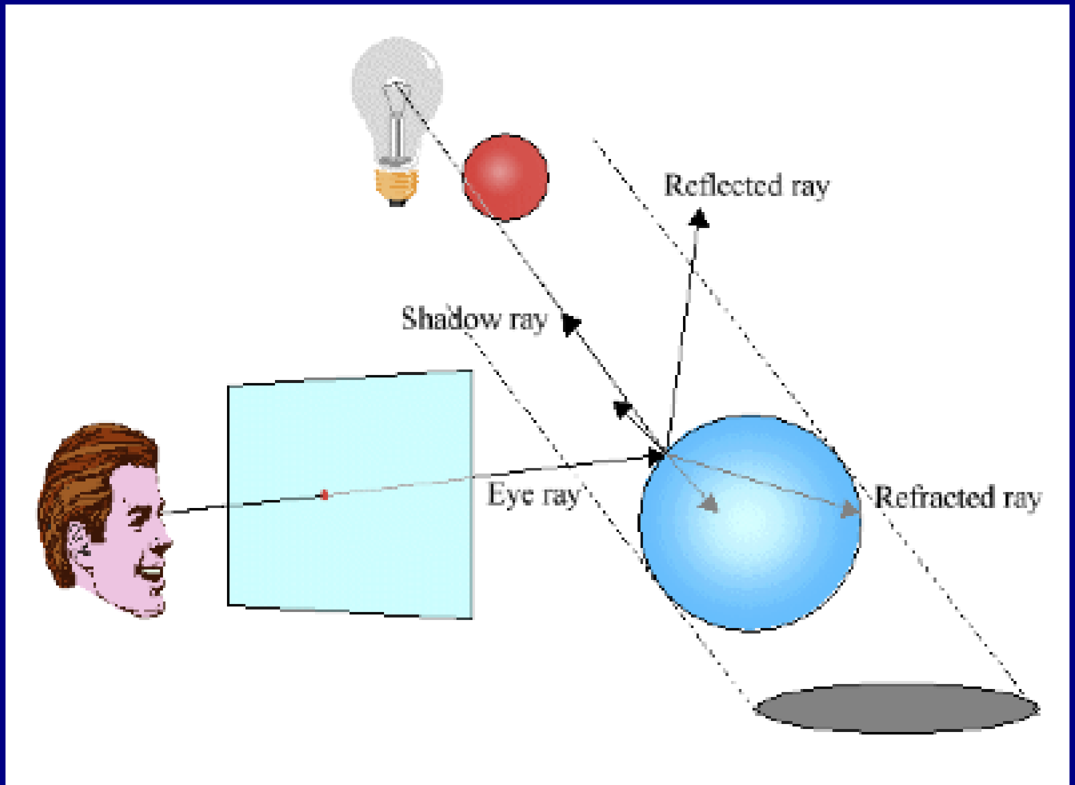
$\mathbf{q} + t\,\mathbf{l}$ does hit an object and is shadowed

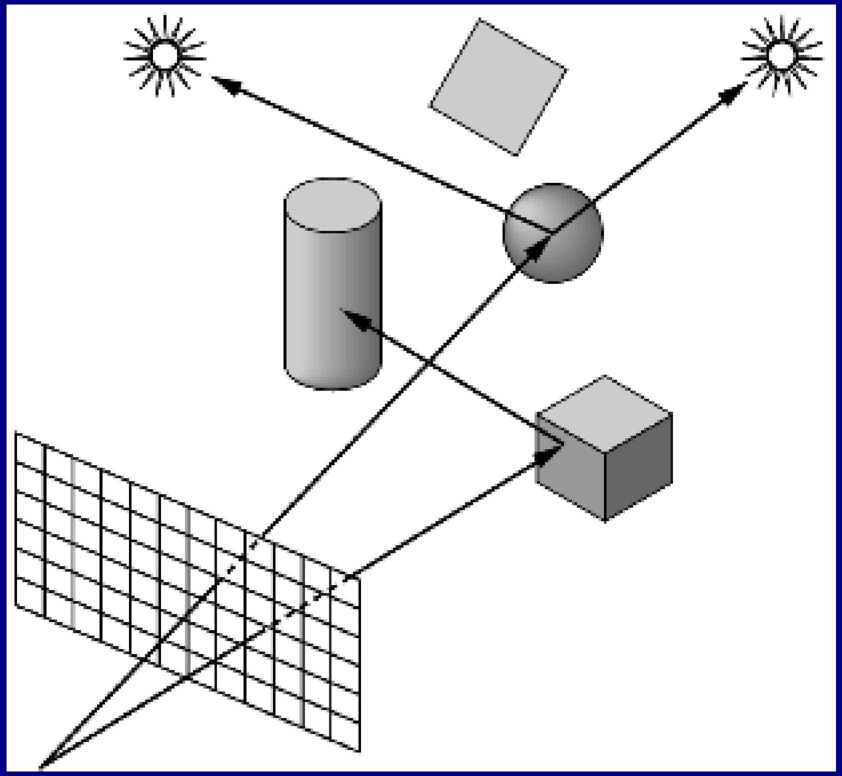
$\mathbf{l}$ the same for both points because this is a directional light (infinitely far away)



# From Last time: Recursive Ray Tracing

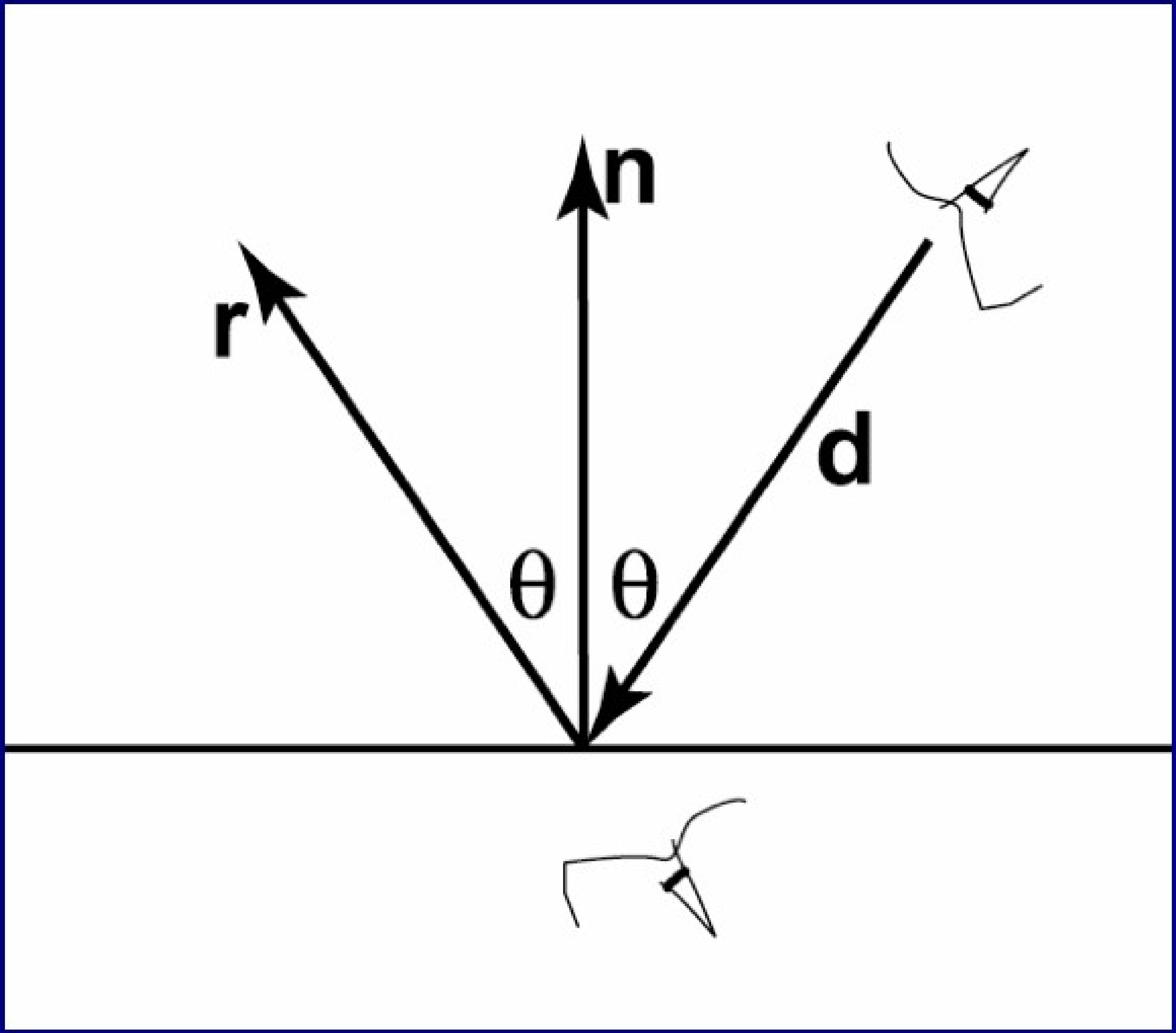


Four ray types:

- Eye rays: originate at the eye
- Shadow rays: from surface point toward light source
- Reflection rays: from surface point in mirror direction
- Transmission rays: from surface point in refracted direction

# Specular Reflection Rays




blackboard

# Transmission Rays

Dielelectrics—transparent material that refracts (and filters) light. Diamonds, glass, water, and air.

Light bends by the physics *principle of least time*

- light travels from point A to point B by the fastest path
- when passing from a material of one index to another *Snell's law* gives the angle of refraction

When traveling into a denser material (larger $n$), light bends to be more perpendicular (eg air to water) and vice versa

| MATERIAL | INDEX OF REFRACTION |
|---|---|
| air/vacuum | 1 |
| water | 1.33 |
| glass | about 1.5 |
| diamond | 2.4 |

# Refraction

# Transmission Rays

Dielelectrics—transparent material that refracts (and filters) light. Diamonds, glass, water, and air.

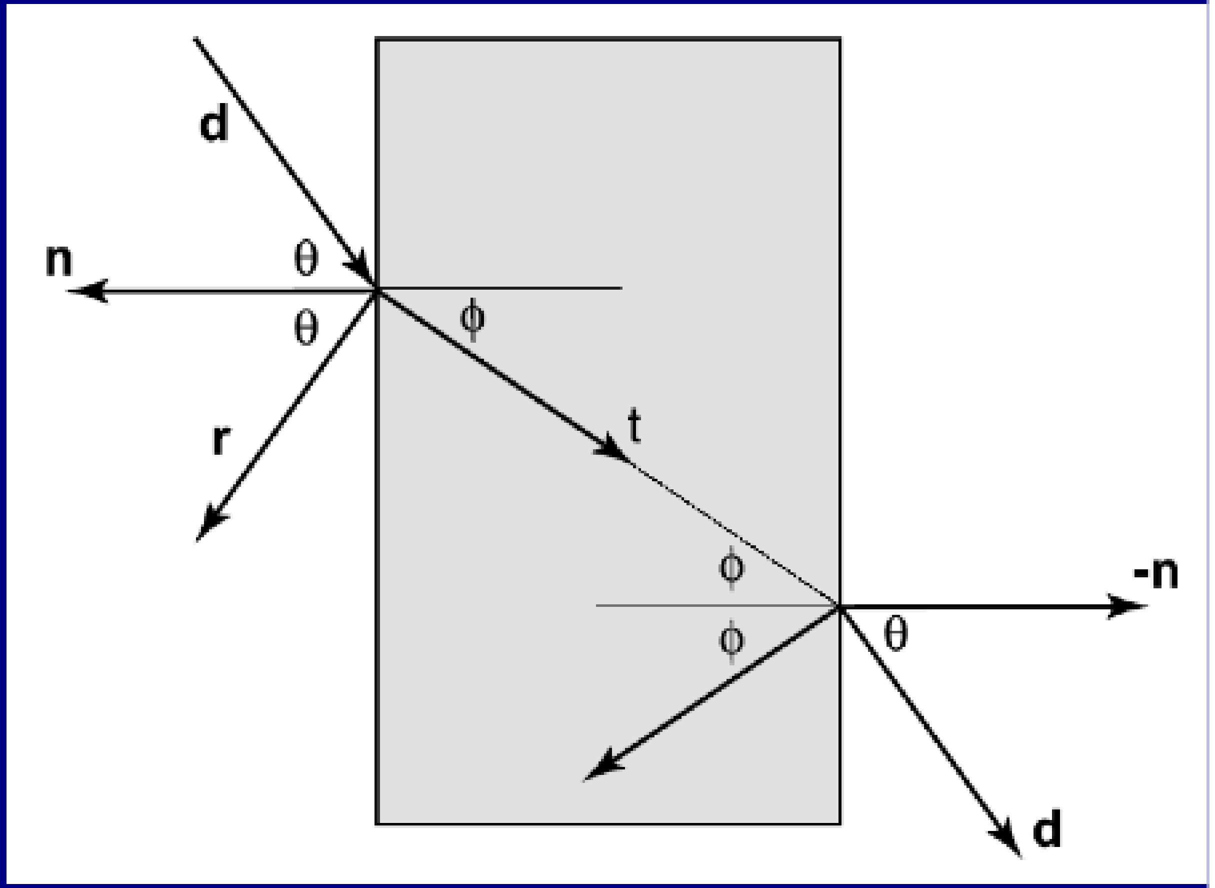
Snell's law:

$$n \sin\theta = n_t \sin\phi$$

$n$ is the refractive index of the first material.

$n_t$ is the refractive index of the second material

**blackboard**



# Transmission Rays

Dielelectrics—transparent material that refracts (and filters) light.  Diamonds, glass, water, and air.

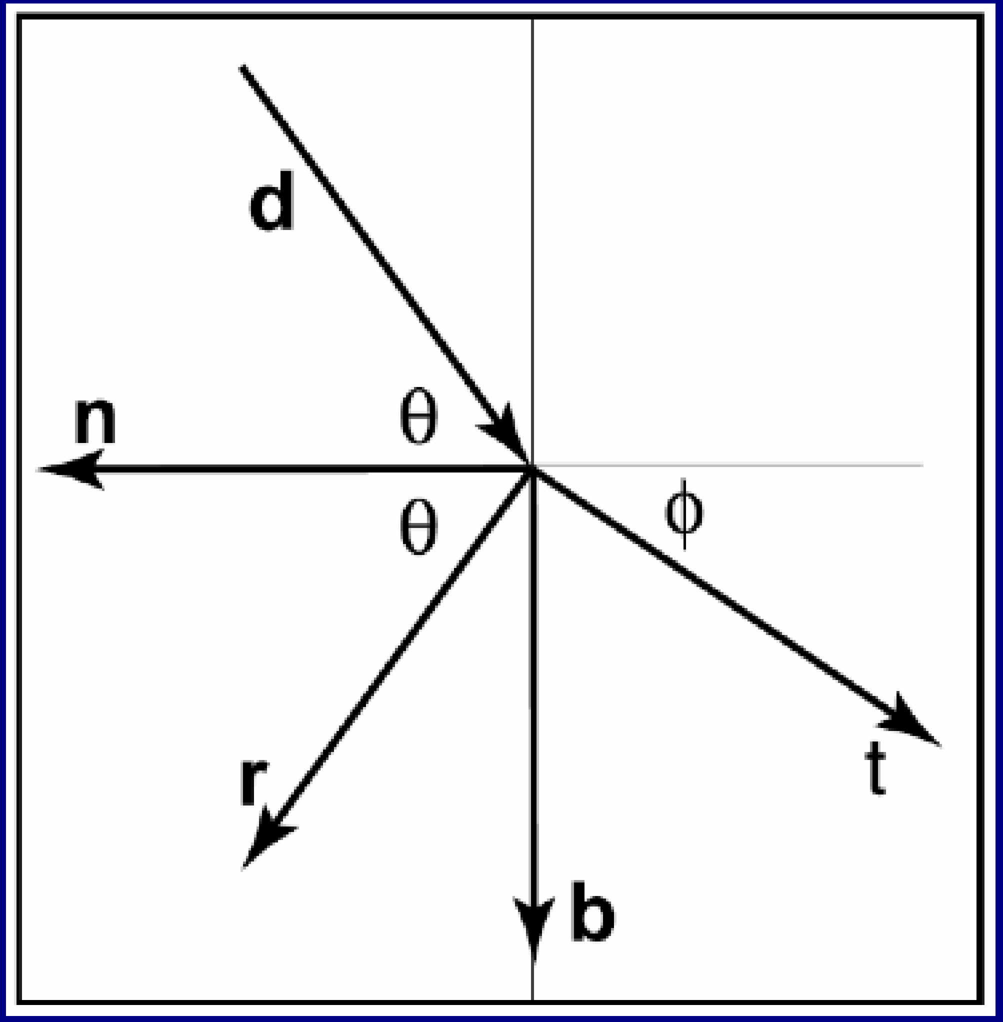
Snell's law:

$$n \sin\theta = n_t \sin\phi$$

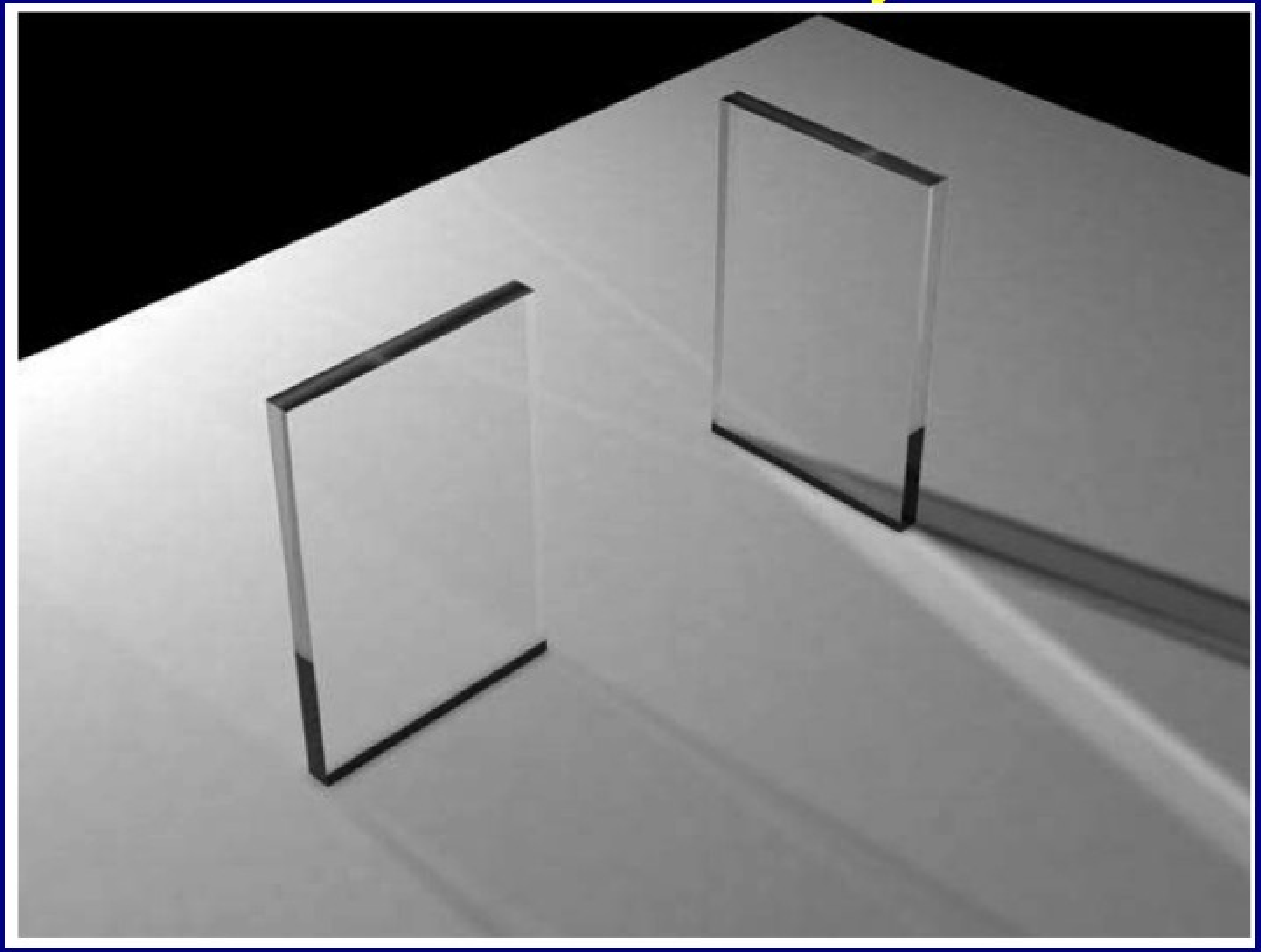
$n$ is the refractive index of the first material.

$n_t$ is the refractive index of the second material

blackboard



# Transmission Rays



# Total Internal Reflection

http://en.wikipedia.org/wiki/File:Total_internal_reflection_of_Chelonia_mydas_.jpg