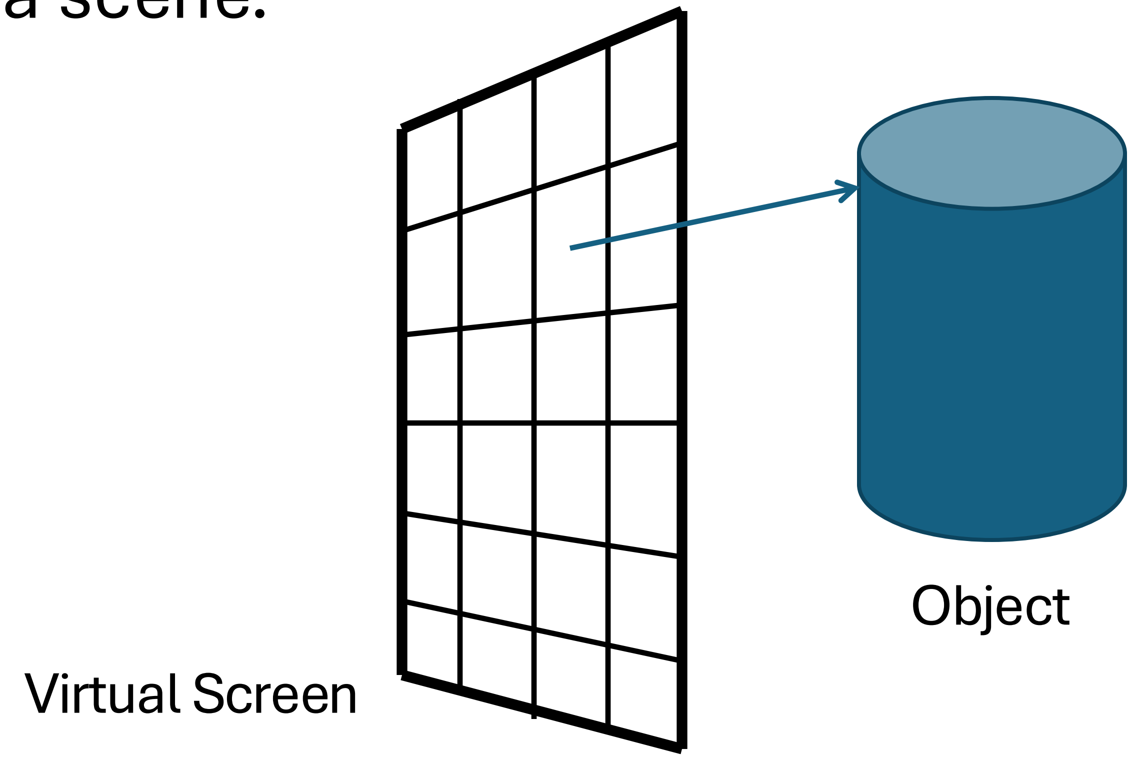
Ray Tracing

CSU44052 Computer Graphics

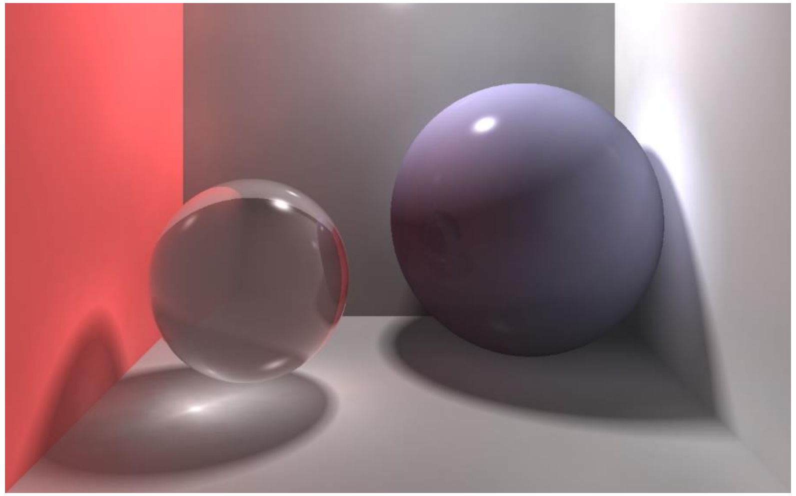
Binh-Son Hua

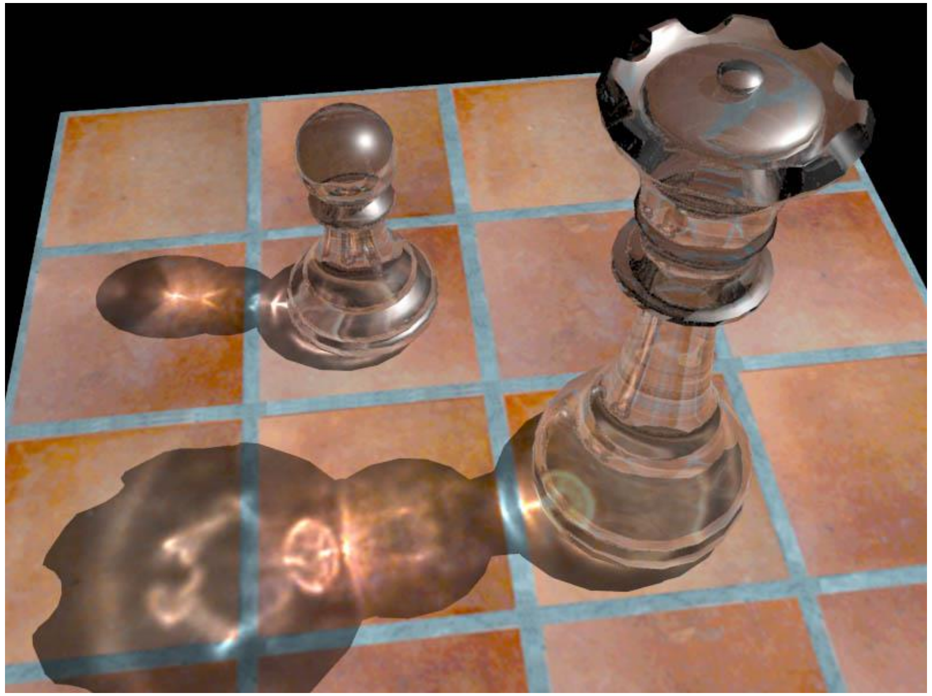
Rendering

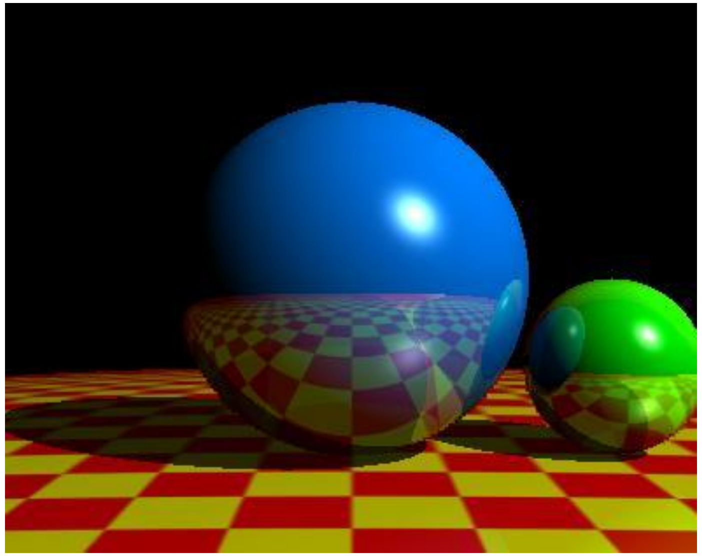
- Rendering is fundamentally concerned with determining the most appropriate colour (i.e. RGB tuple) to assign to a pixel associated with an object in a scene.



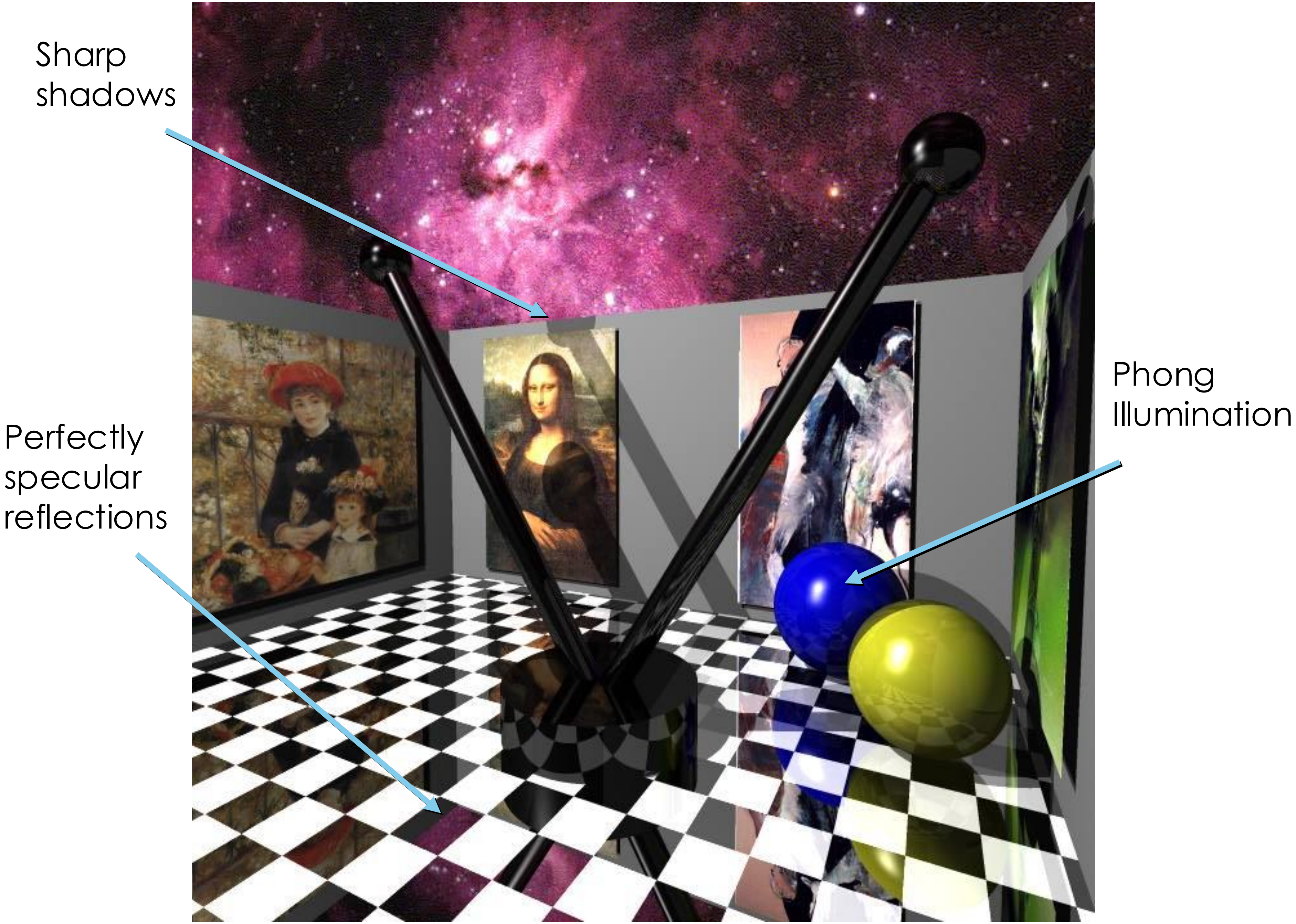
Global Illumination



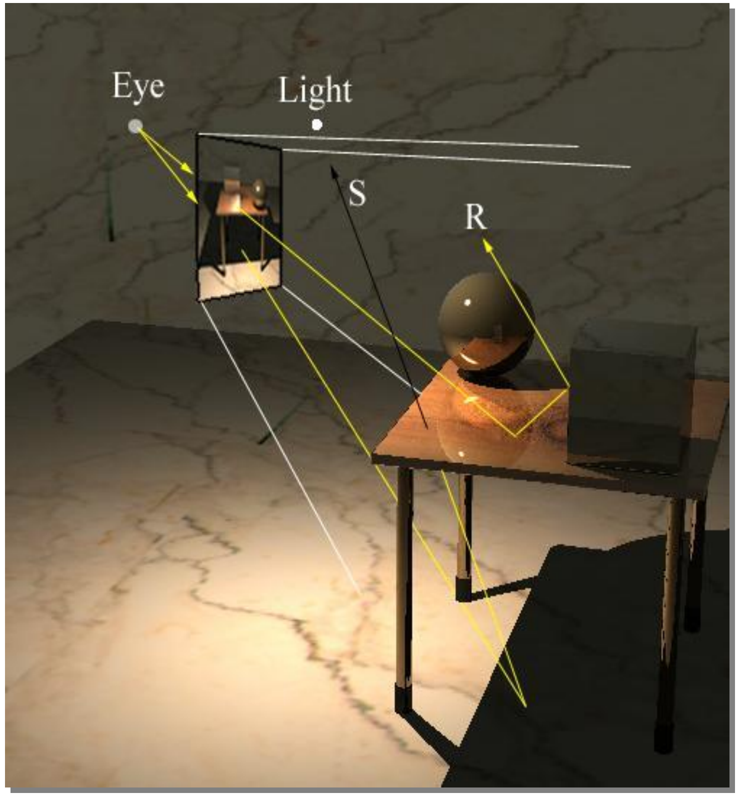




Ray Tracing


Sharp shadows
Perfectly specular reflections
Phong Illumination

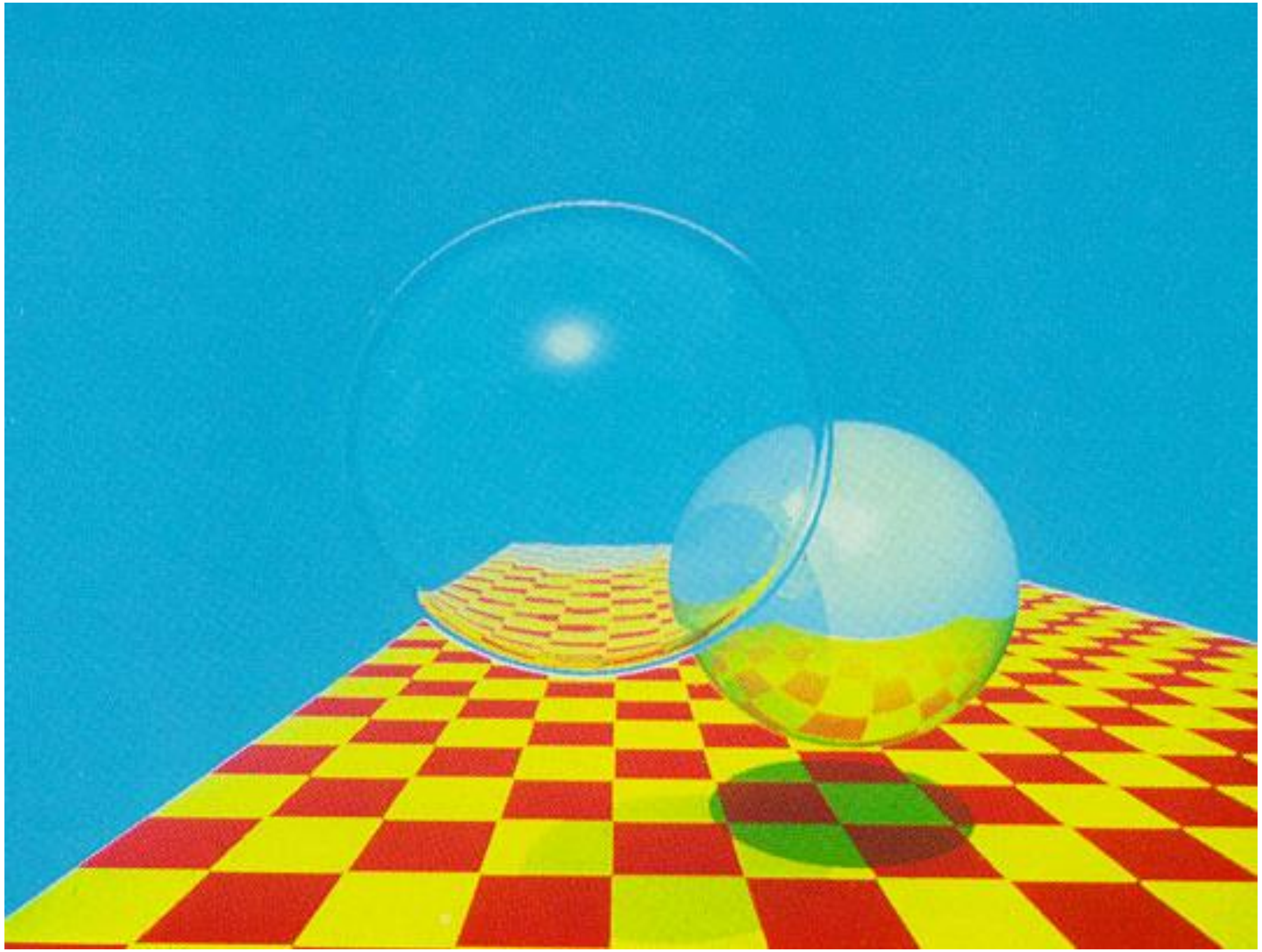
Ray Tracing is a View-Dependent Solution



Scene Geometry

Solution determined only for directions through pixels in the viewport

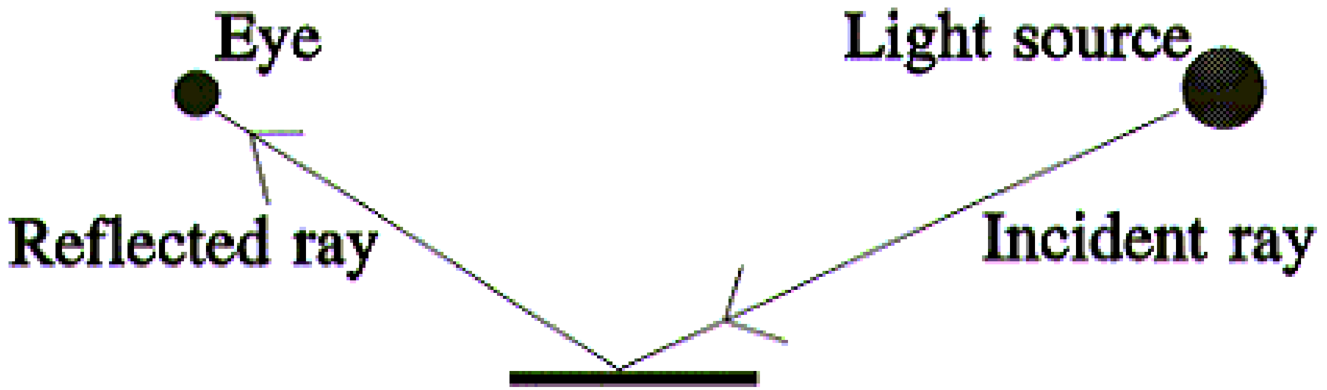
First ray-traced image



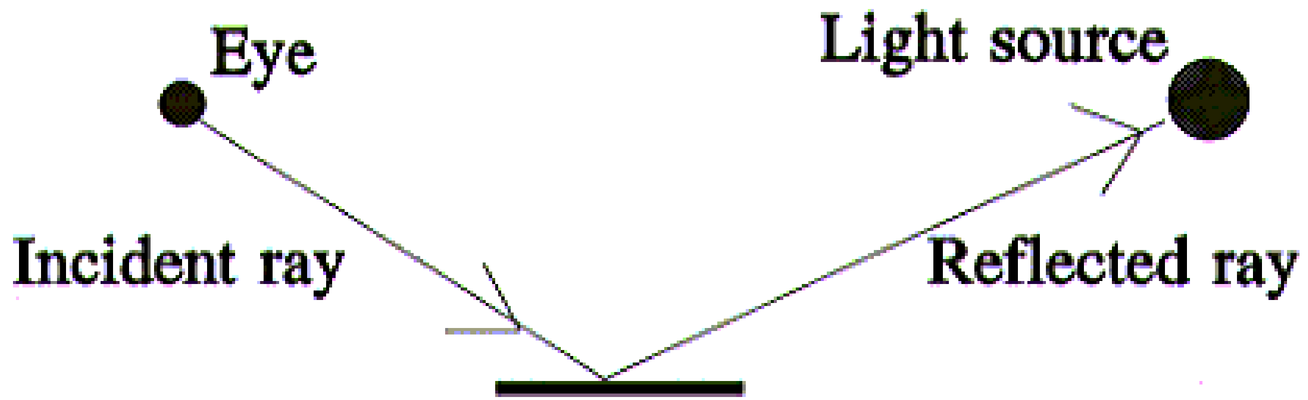
Turner Whitted, 1979

Ray-tracing history

- First used in computer graphics in 1980
- Integrated reflection, refraction, hidden surface removal, shadows in a single model
- Rays usually considered to be infinitely thin
  - Reflection & refraction occur without any spreading
  - Perfectly smooth surfaces
  - Not real-world – like a wall of mirrors
- “Super-real” images at a high cost

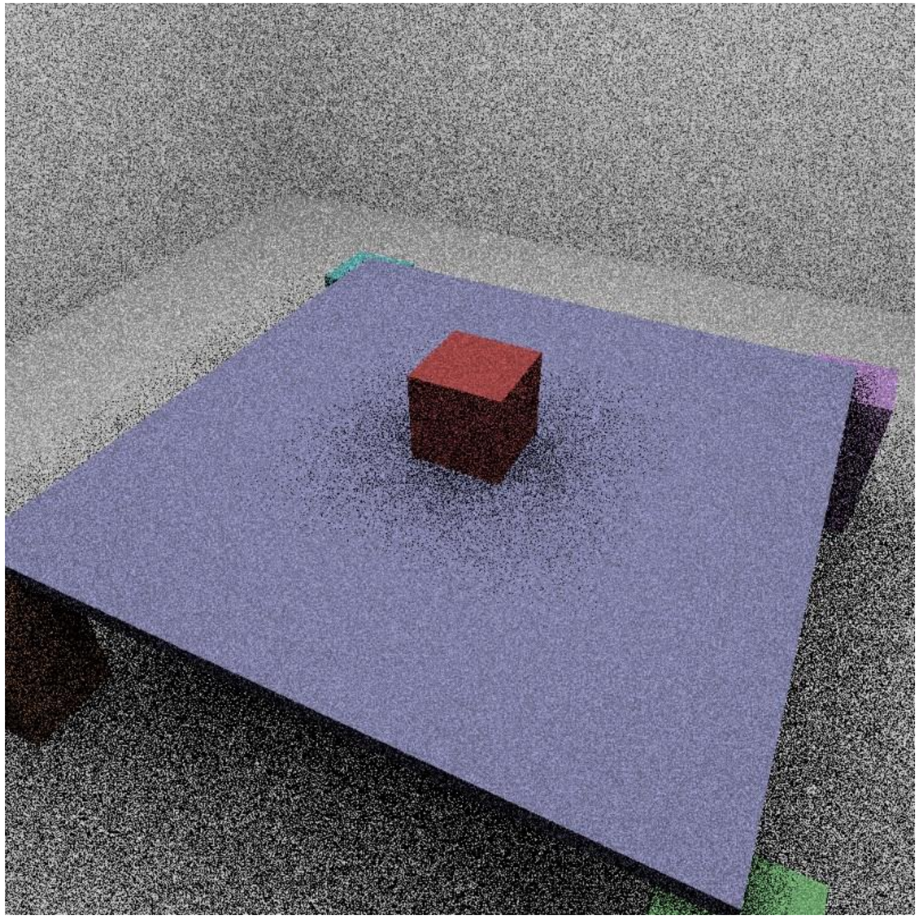


Forward Ray Tracing



Backward Ray Tracing

Forward Ray tracing

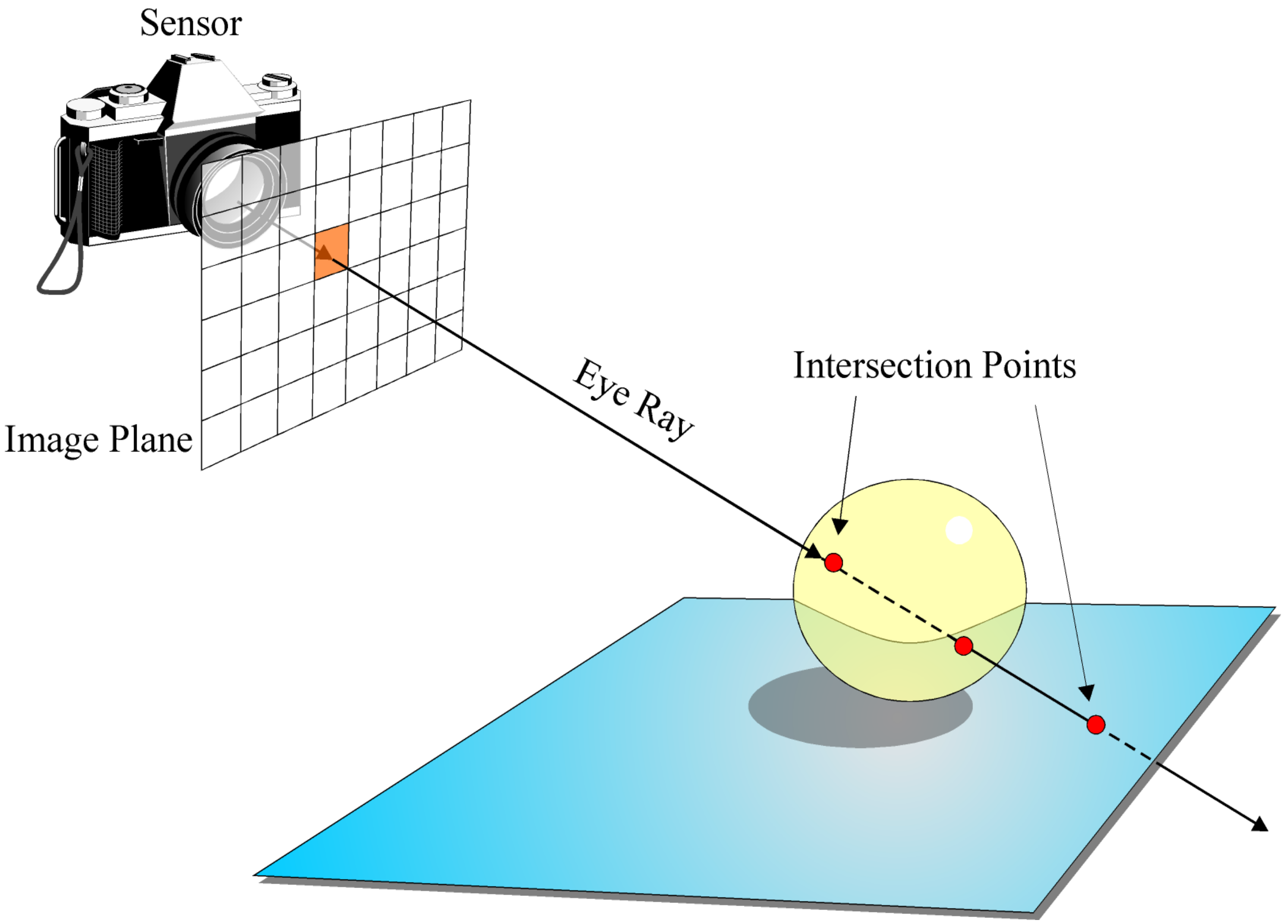


- Only a fraction of rays reach the image
- Many, many rays are required to get a value for each pixel.

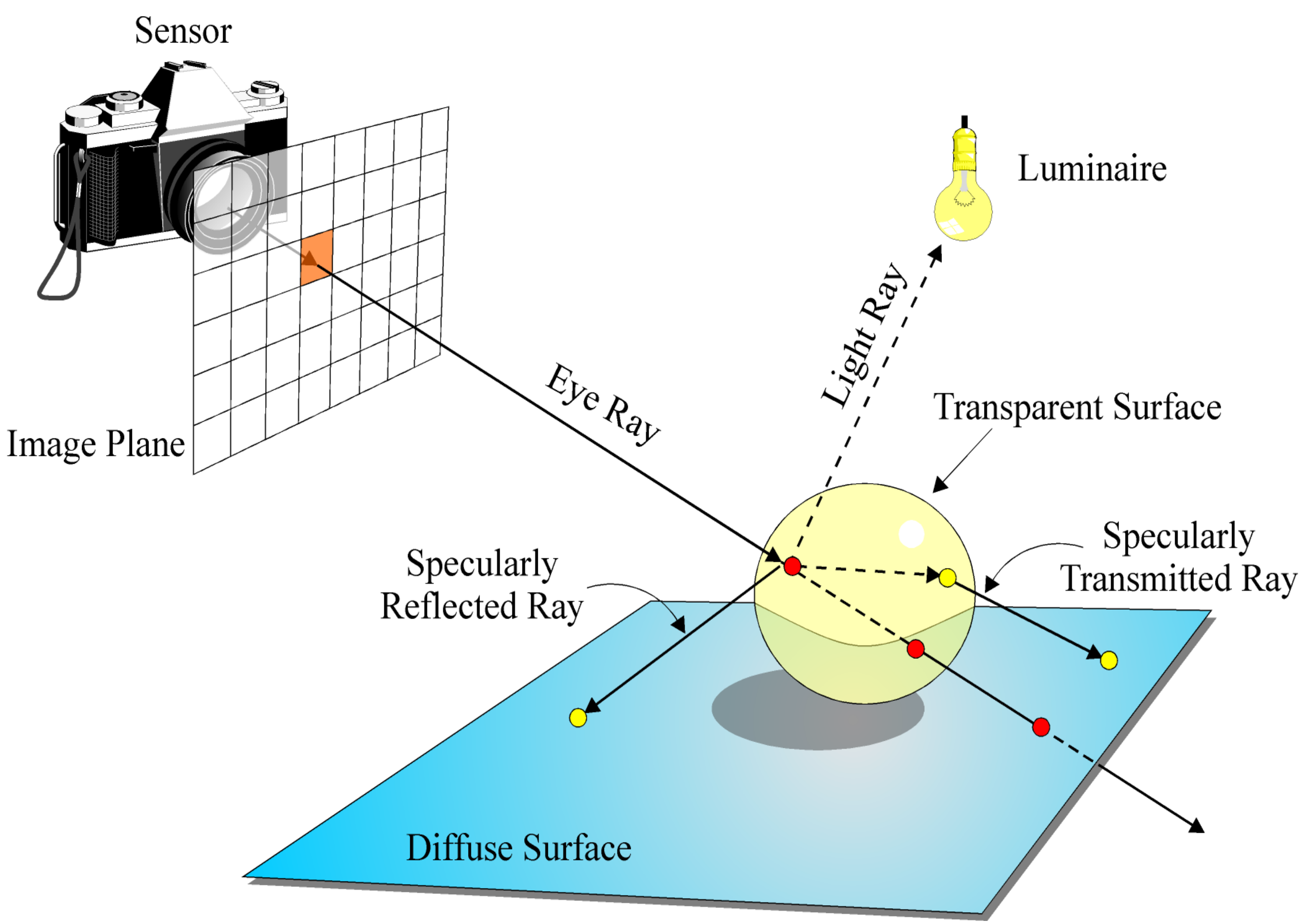
Backward Ray Tracing

For each pixel in the viewport:

- Trace a ray from the eye (called the eye ray) into the scene, through the pixel.
- Determine the first object hit by the ray $\Rightarrow$ ray casting.
- We then shade this using an extended form of the Phong model.



Ray Tracing


Sensor
Luminaire
Light Ray
Eye Ray
Transparent Surface
Image Plane
Specularly
Transmitted Ray
Specularly
Reflected Ray
Diffuse Surface

Ray Tracing

- The eye ray will typically intersect a number of objects, some more than once $\Rightarrow$ sort intersections to find the closest one.

- The Phong illumination model is then evaluated BUT:
  - we trace a reflected ray if the surface is specular
  - we trace a refracted ray if the surface is transparent
  - we trace shadow rays towards the light sources to determine which sources are visible to the point being shaded

- The reflected/refracted rays themselves will hit surfaces and we will **recursively** evaluate the illumination at these points.

- A very large number of rays must be traced to illuminate a single pixel.

Whitted Illumination Model

- Whitted Illumination Model

$$L_r(x,V) = \underbrace{L_{emitted}(x,V) + L_{Phong}(x,V)}_{\text{local contribution}} + \underbrace{L_{reflected}(x,V) + L_{refracted}(x,V)}_{\text{global contribution}}$$

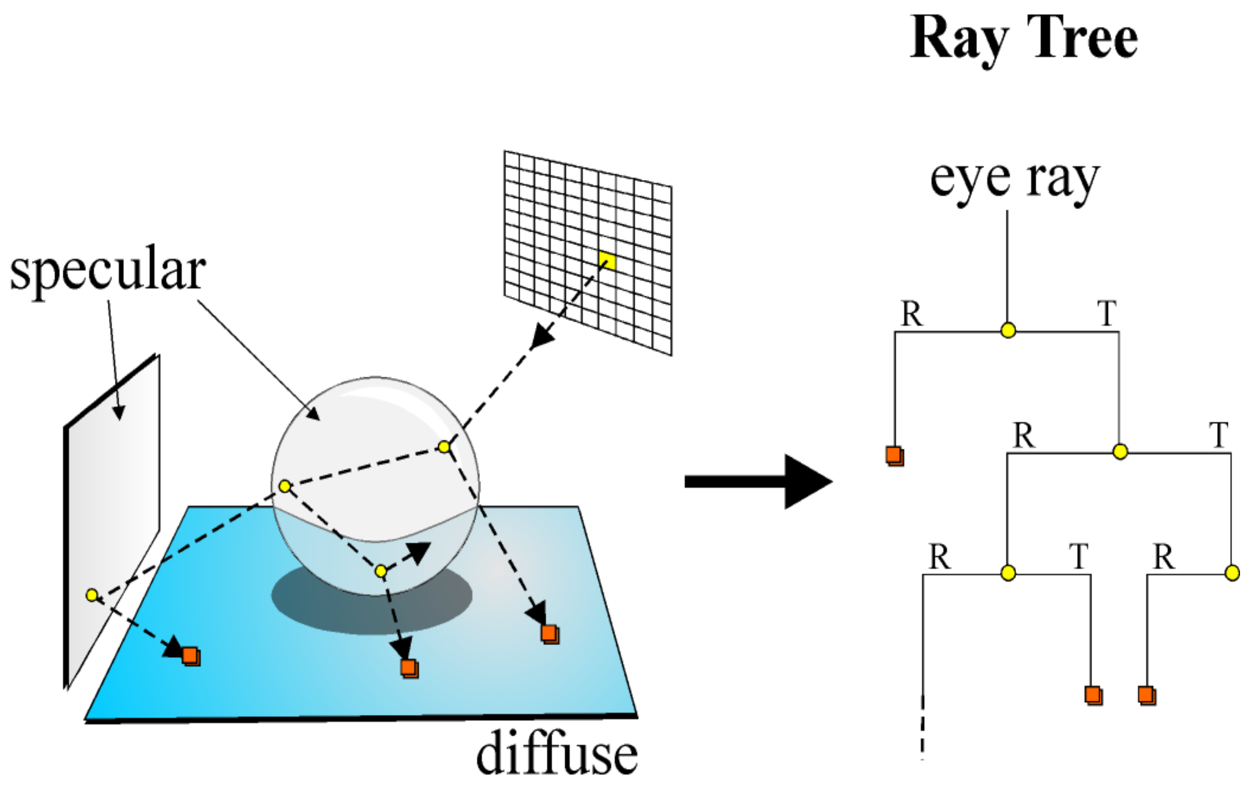
- Ray tracing is a hybrid local/global illumination algorithm

- We only consider global lighting effects from ideal specular directions

- Also, before adding each light's Phong contribution we determine if it is visible to point x, thus allowing shadows to be determined.

Recursive Ray Tracing

- The ray tracing algorithm is recursive, just as the radiance equation is recursive.
- At each intersection we trace a specularly reflected and transmitted ray (if the surface is specular) or terminate the ray if diffuse.
- Thus we trace a ray back in time to determine its history, beginning with the eye ray: this leads to a binary tree



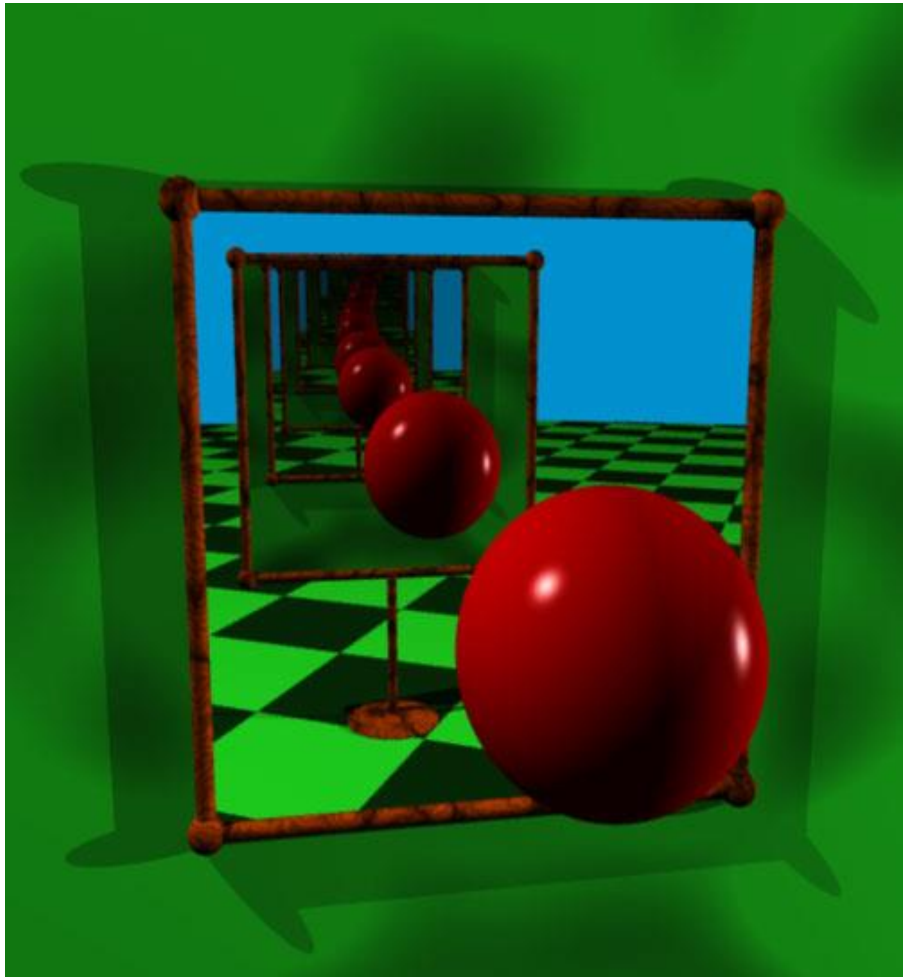
Recursive Ray Tracing

- In theory, this recursive process could continue indefinitely.

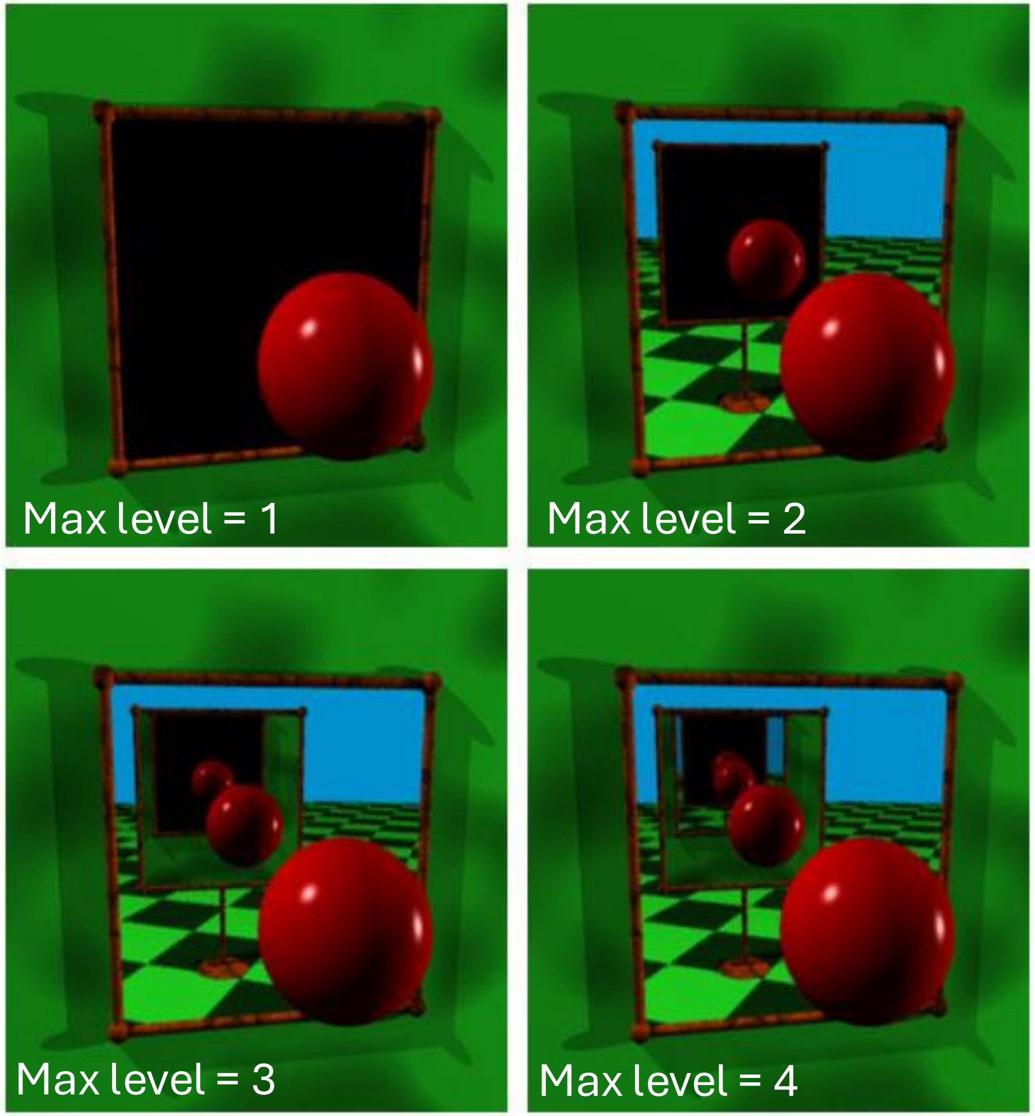
- In practice, at each intersection the ray loses some of its contribution to the pixel (i.e. its importance decreases).
  - if the eye ray hits a specularly reflecting surface with reflectivity of 50%, then only 50% of the energy hitting the surface from the reflected direction is reflected towards the pixel.
  - if the next surface hit is of the same material, the reflected ray will have its contribution reduced to 25%.

- We terminate the recursion if:
  - the current recursive depth > a pre-determined maximum depth or
  - if the ray's contribution to the pixel < some pre-determined threshold ε

Recursion Clipping



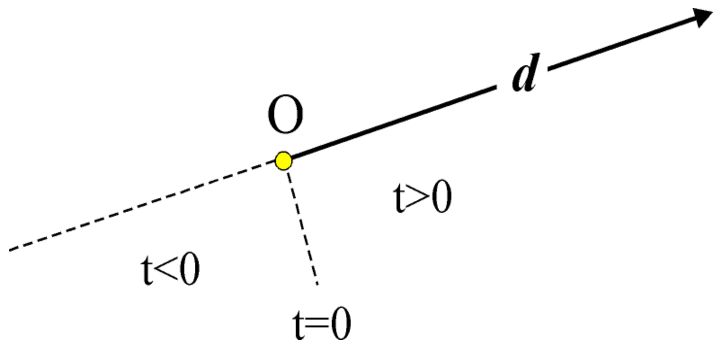
Very high maximum recursion level



The Ray

- Mathematically, a ray is the *affine half-space* defined by:

$$\mathbf{r} = O + t\vec{d} \quad t \geq 0$$

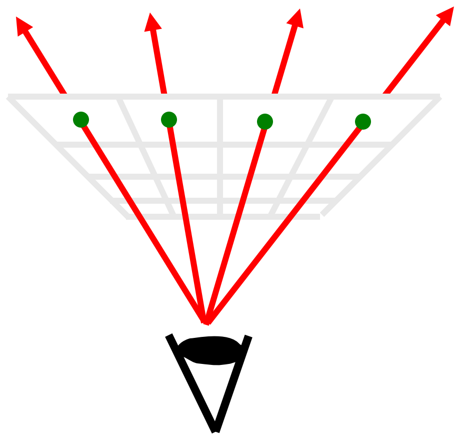


- All points on the ray correspond to some positive value of *t*, the *parametric distance* along the ray. If ***d*** is *normalised* then t is the length along the ray of the point.

Ray Tracing Algorithm

```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```

```
trace(ray, objects)
{
   for each object in scene
       intersect(ray, object)
   sort intersections
   return closest intersection
}
```



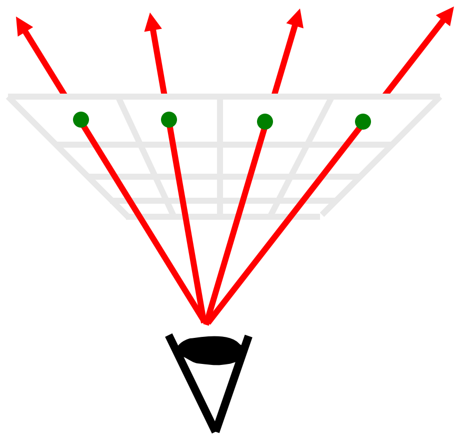
Ray Tracing Algorithm

```
colour shade(ray, intersection)
{
    if no intersection
        return background colour

    for each light source
        if(visible)
            colour += Phong contribution

    if(recursion level < maxlevel and surface not diffuse)
    {
        ray = reflected ray
        intersection = trace(ray, objects)
        colour += ρrefl*shade(ray, intersection)
      }
    return colour
}
```

Ray Casting

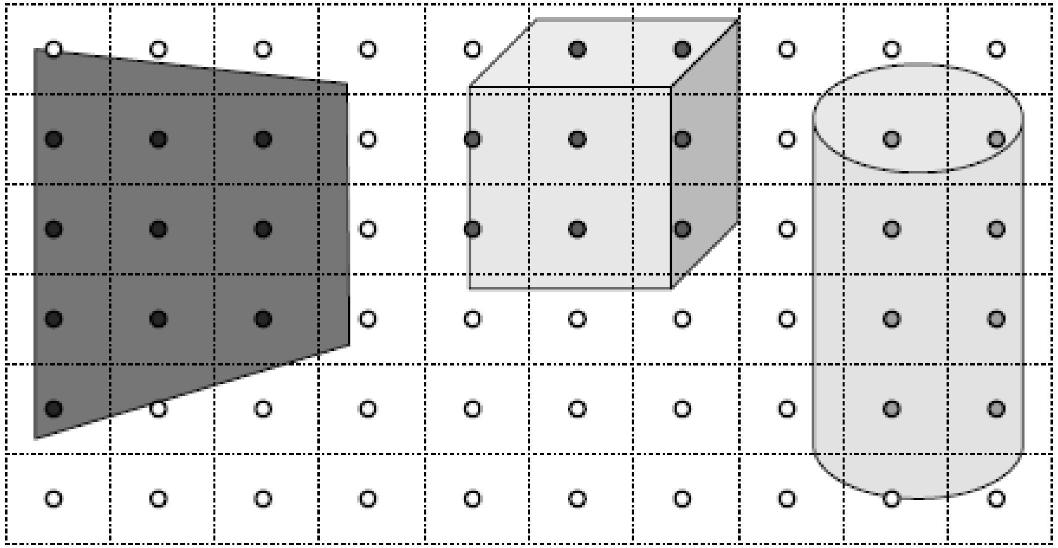
```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```

```
trace(ray, objects)
{
   for each object in scene
       intersect(ray, object)
   sort intersections
   return closest intersection
}
```



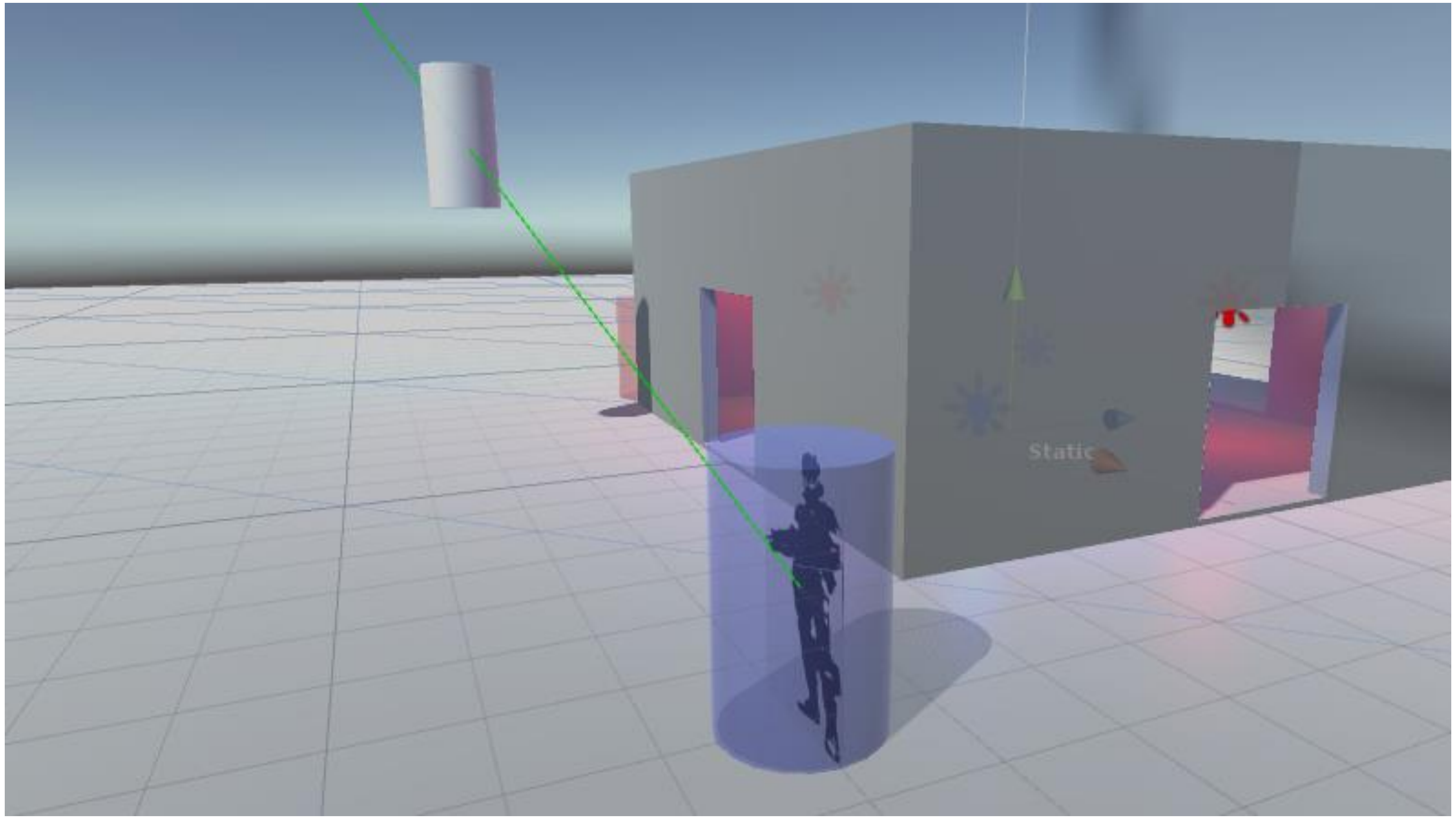
Ray Casting

- For each sample
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute colour sample based on surface radiance



Ray Casting

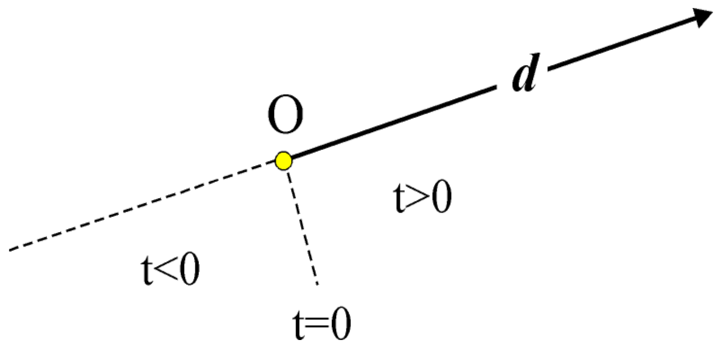
- Collision detection
- Object picking
- Line of sight
- Occlusion culling



The Ray

- Mathematically, a ray is the *affine half-space* defined by:

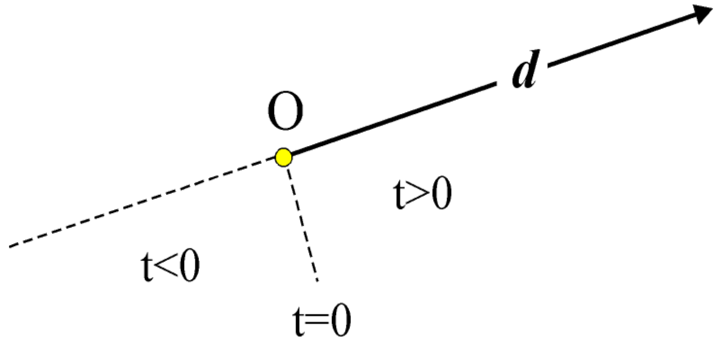
$$\mathbf{r} = O + t\vec{d} \quad t \geq 0$$



- All points on the ray correspond to some positive value of *t*, the *parametric distance* along the ray. If ***d*** is *normalised* then t is the length along the ray of the point.

Ray-Object Intersection Testing

- Once we’ve constructed the eye rays we need to determine the intersections of these rays and the objects in the scene.
- Upon intersection we need the **normal** vector at the point of intersection in order to perform shading calculations.

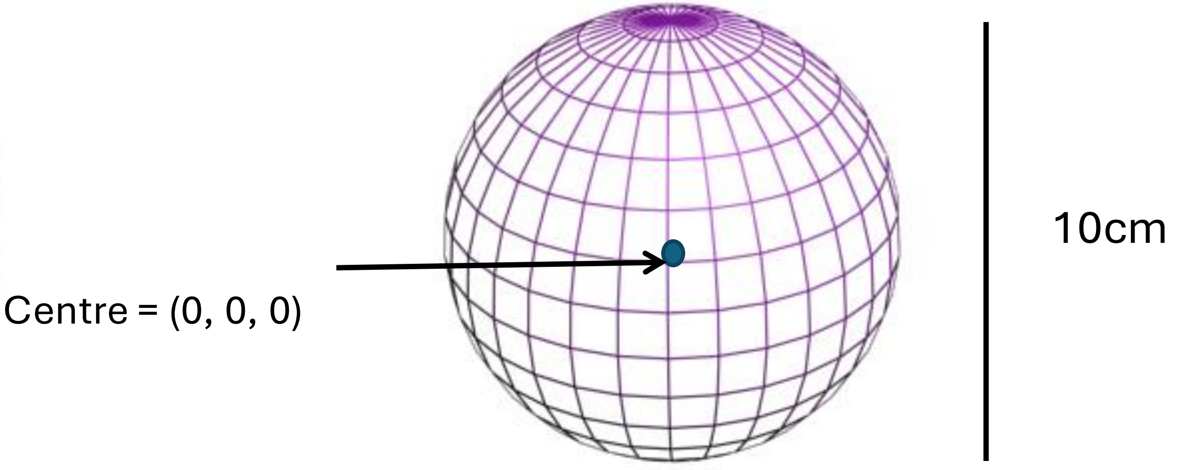


The Sphere

- A sphere of center (Cx, Cy, Cz) with radius r is given by:

$$f(x, y, z) = (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$$

- Question: is the point (5,1,0) on this sphere?



The Sphere

- A sphere object is defined by its center C and its radius r.
- Implicit Form:

$$f(\vec{v}) = |\vec{v} - C|^2 - r^2 = 0$$

$$f(x, y, z) = (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$$

- Explicit Form:

$$x = f_x(\theta, \phi) = C_x + r \sin\theta \cos\phi$$

$$y = f_y(\theta, \phi) = C_y + r \cos\theta$$

$$z = f_z(\theta, \phi) = C_z + r \sin\theta \sin\phi$$

- We can use either form to determine the intersection; we will choose the implicit form.

Ray Sphere Intersection

- All points on the ray are of the form: $\mathbf{r}ay = O + t\vec{d} \quad t \geq 0$
- All points on the sphere satisfy:

$$(x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - r^2 = 0$$

- Any intersection points (= points shared by both) must satisfy both, so substitute the ray equation into the sphere equation and solve for t:

$$\left(\left[\underbrace{O_x + td_x}\right] - C_x\right)^2 + \left(\left[\underbrace{O_y + td_y}\right] - C_y\right)^2 + \left(\left[\underbrace{O_z + td_z}\right] - C_z\right)^2 - r^2 = 0$$

ray equation

Problem

$$\left(\left[O_x + td_x\right] - C_x\right)^2 + \left(\left[O_y + td_y\right] - C_y\right)^2 + \left(\left[O_z + td_z\right] - C_z\right)^2 - r^2 = 0$$

ray equation

- Expand the first term
  - remember $(a-b)^2 = a^2 - 2ab + b^2$
- Rearrange into: $$At^2 + Bt + C = 0$$

Ray Sphere Intersection

- Rearrange and solving for t leads to a quadratic form (which is to be expected as the sphere is a quadratic surface):

$$
\begin{aligned}
&At^2 + Bt + C = 0 \\
&A = \left(d_x^2 + d_y^2 + d_z^2\right) = 1 \\
&B = 2d_x\left(O_x - C_x\right) + 2d_y\left(O_y - C_y\right) + 2d_z\left(O_z - C_z\right) \\
&C = \left(O_x - C_x\right)^2 + \left(O_y - C_y\right)^2 + \left(O_z - C_z\right)^2 - r^2
\end{aligned}
$$

- We employ the classic quadratic formula to determine the 2 possible values of t:

$$
t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} = \frac{-B \pm \sqrt{B^2 - 4C}}{2}
$$

Intersection Classification

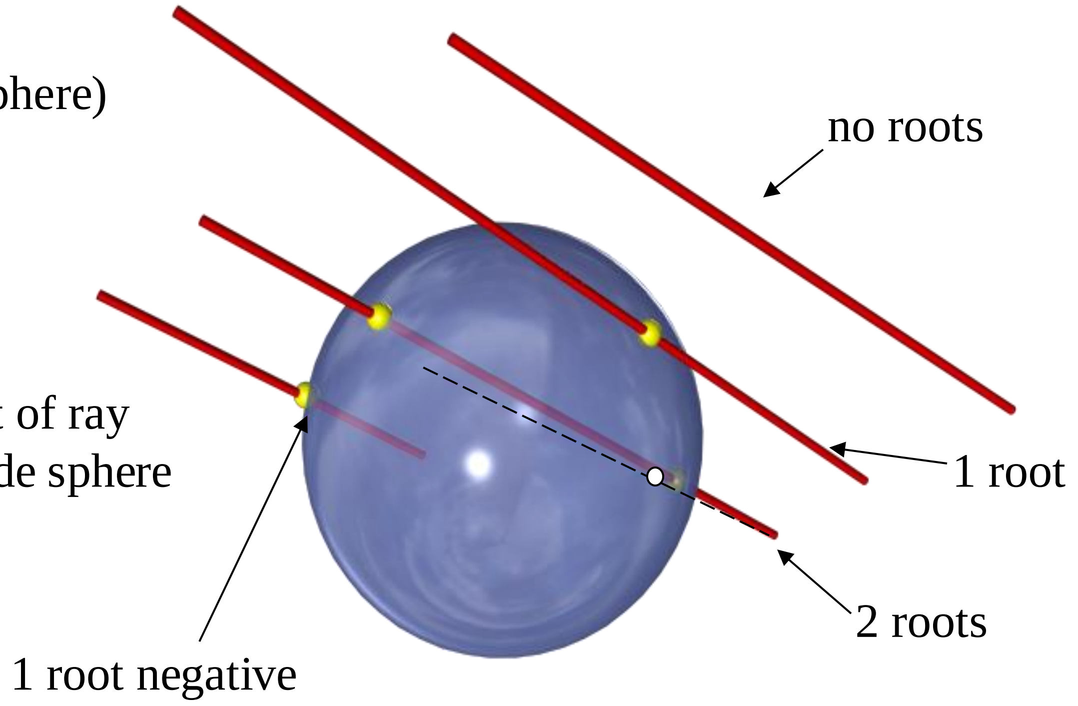
- Depending on the number of real roots we have a number of outcomes which have nice geometric interpretations
- We use the discriminant: $d = B^2 - 4C$

$d = 0 \Rightarrow$ 1 root (ray is tangent to sphere)

$d < 0 \Rightarrow$ no real roots (ray misses)

$d > 0 \Rightarrow$ 2 real roots:

Both positive $\Rightarrow$ sphere in front of ray
One negative $\Rightarrow$ ray origin inside sphere



Ray Sphere Intersection Test

- If we have 2 positive values of t we use the **smallest** (i.e. the nearest to the origin of the ray).

- t is substituted back into the ray equation yielding the point of intersection: $P_{int} = O_{ray} + t_{int} d_{ray}$

- We then evaluate the Phong model at this point. To do so we need the <u>normal to the surface of the sphere</u> at the point of intersection.

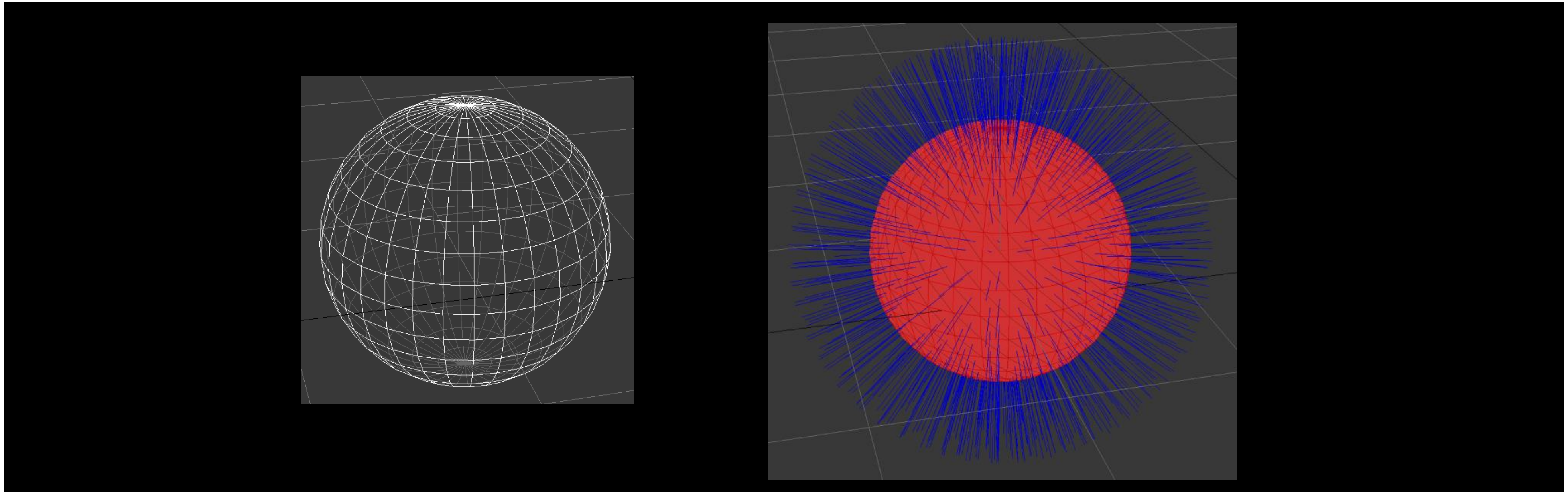
- The normal and the original ray direction are then used to determine the directions of reflected and refracted rays.

Normal to Sphere

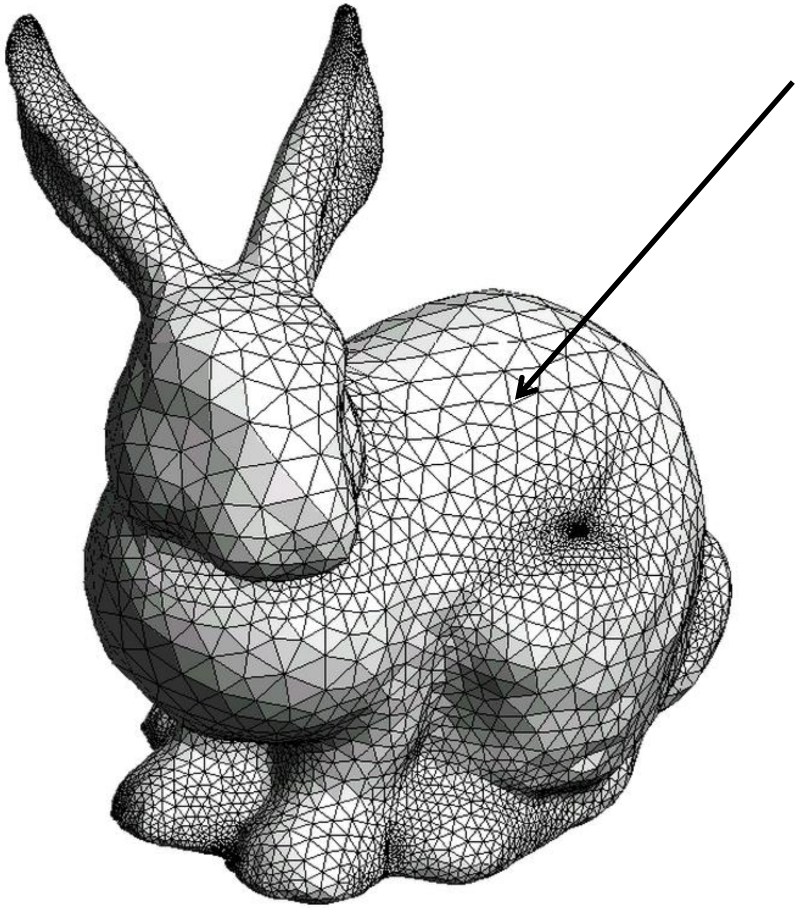
- We can compute the normal to a sphere with centre C at a point x as:

$$N = \frac{\vec{x} - C}{\left|\vec{x} - C\right|}$$

i.e. the normal to the sphere is the normalised vector associated with the point.



Ray Polygon Intersection

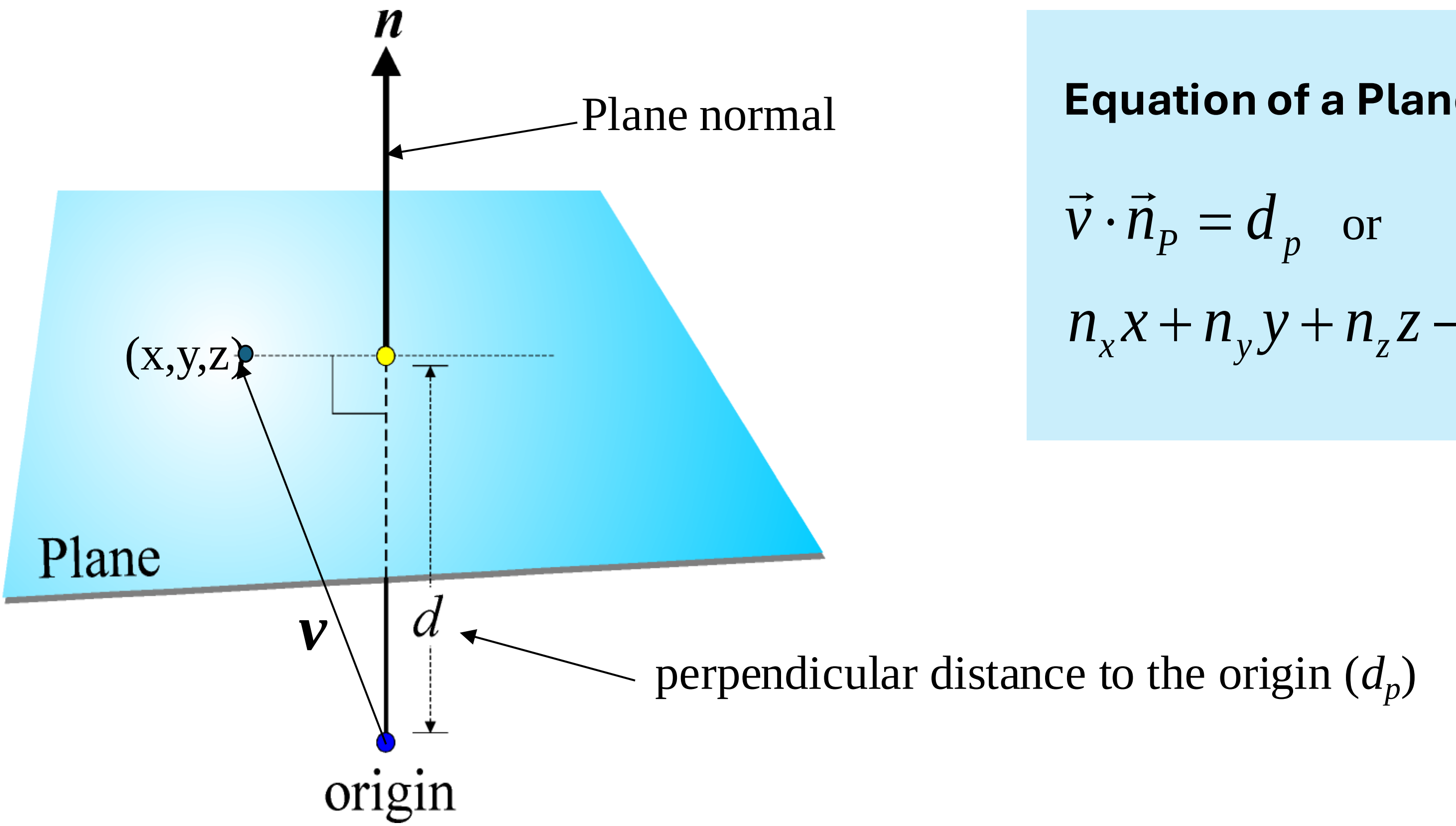


To test if the ray intersects the polygon:

- Assume <u>planar</u> polygons
- First see if the ray intersects the polygon's plane
- If it does, see if the intersection point is inside or outside the polygon

Plane Definition

- A plane may be defined by its normal and the perpendicular distance of the plane to the origin:



Equation of a Plane

$$\vec{v} \cdot \vec{n}_P = d_p \quad \text{or}$$

$$n_x x + n_y y + n_z z - d_P = 0$$

Ray Plane Intersection

1. Does ray intersect the polygon's plane?

$$\vec{n}_P . \vec{d}_r$$

- First compute the dot product between the normalised ray and the polygon's normal (i.e., the plane's normal):
  - If dot product == 0 it => ray & normal perpendicular
    - No intersection!
  - If > 0 => ray and normal are in the same direction
    - Back face intersection
  - if < 0 => plane facing direction of ray
    - There is an intersection!

- Note there can only be one intersection. The normal to the plane at each point is the same, i.e., the plane normal n_P

Ray Plane Intersection

2. Now find intersection point of ray with the plane

To intersect a ray with a plane we substitute in the ray equation and solve for t

$$(O_r + t d_r) \cdot \vec{n}_P = d_P$$

- *O* is origin of the ray and d_r is the direction of the ray. Solve for *t* and that will give you the point on the plane

$$\Rightarrow t = \frac{d_P - \vec{n}_P \cdot O_r}{\vec{n}_P \cdot \vec{d}_r}$$

- *t* is substituted back into the ray equation yielding the point of intersection:

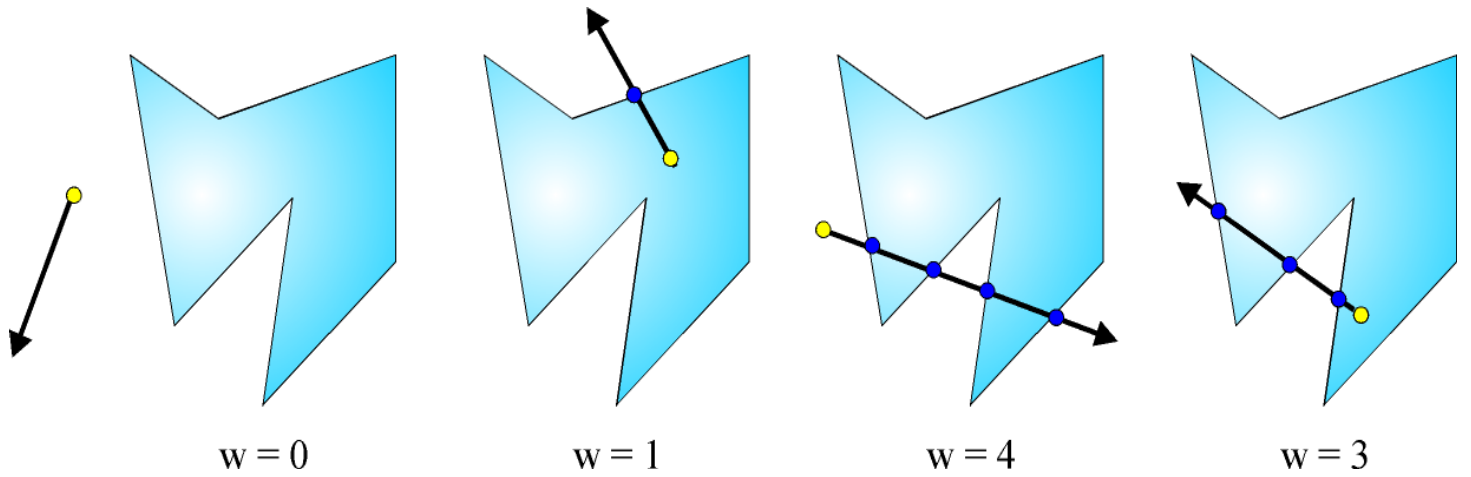
$$P_{int} = O_{ray} + t_{int} d_{ray}$$

Ray Polygon Intersections

3. Next, determine if the intersection point is within the polygon interior.

The *Jordan Curve Theorem*:

- construct any ray with the intersection point as an origin
- count the number of polygon edges this ray crosses (= *winding number*)
- if *w* is odd then the point is in the interior



Ray Triangle Intersections

3. Next, determine if the intersection point is within the **triangle** interior.

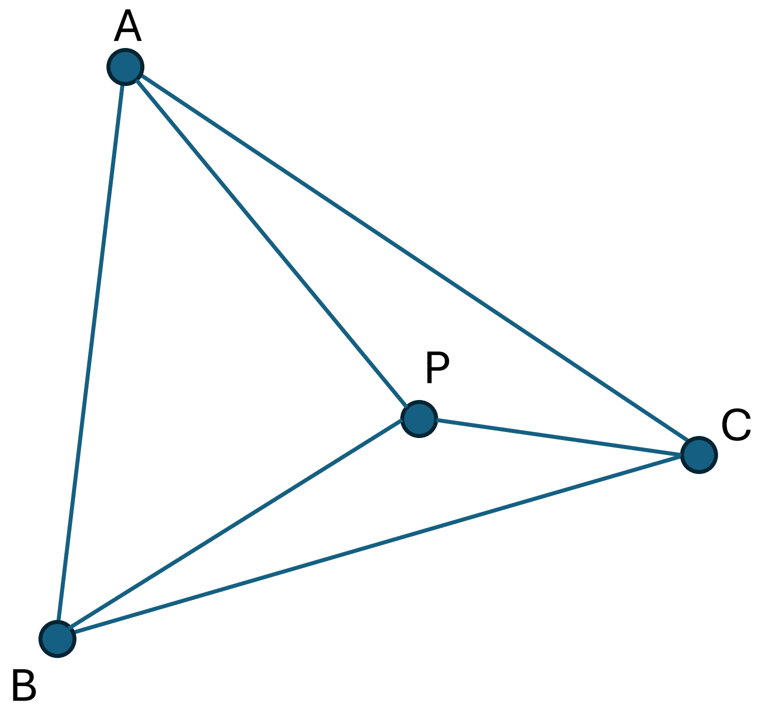
Use barycentric coordinates!

$$P = aA + bB + cC$$

with the constraint

$$a + b + c = 1$$

If 0 <= a, b, c <= 1: point P is in the triangle.

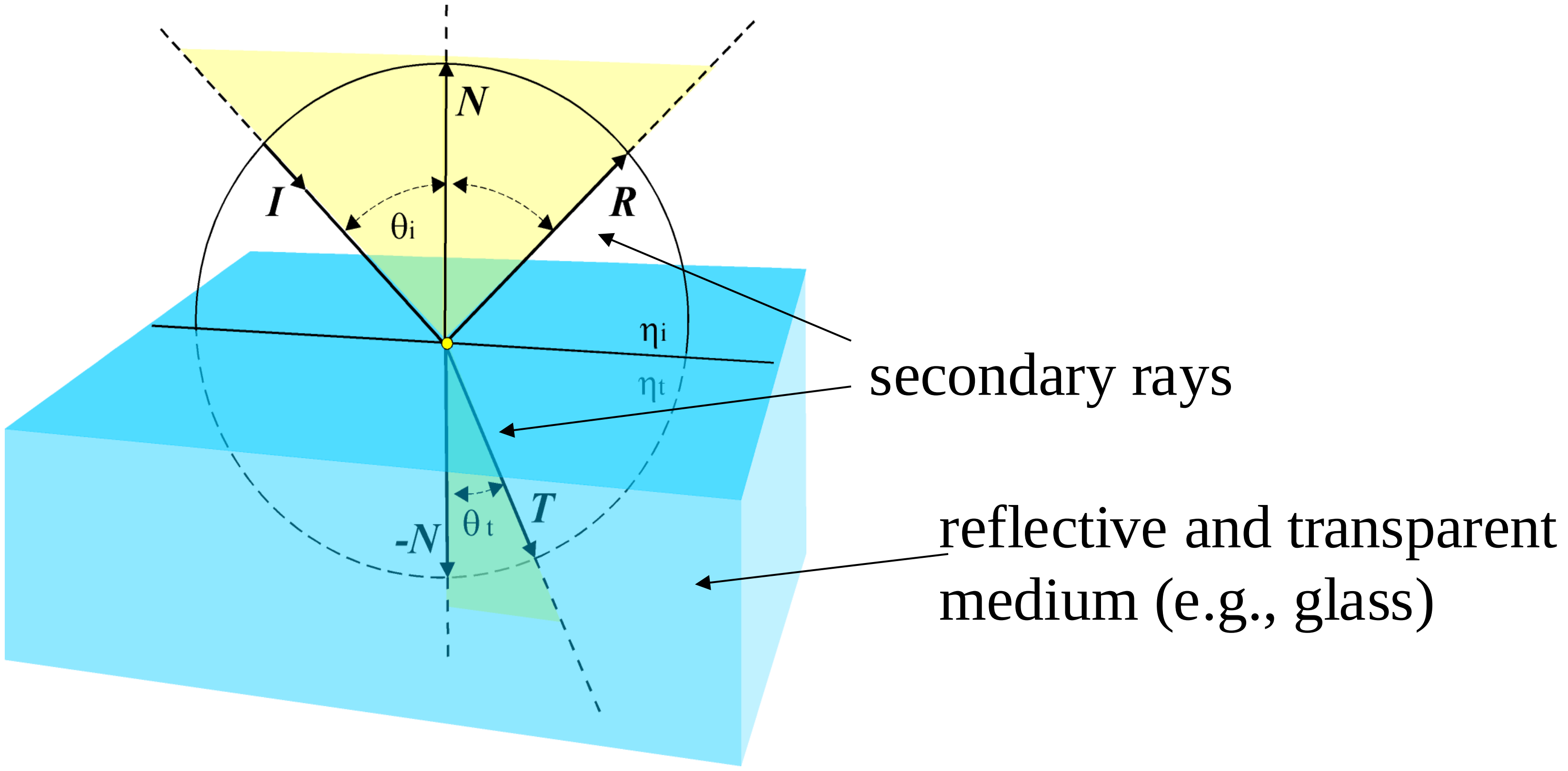


Secondary Rays

- Having determined the closest intersection point x along a ray path we then evaluate the *Whitted illumination model* at this point:
  - Construct a shadow ray to each light source
  - Compute the local illumination at x
  - Construct a *reflected* ray from x and recurse
  - Construct a *refracted* ray from x and recurse

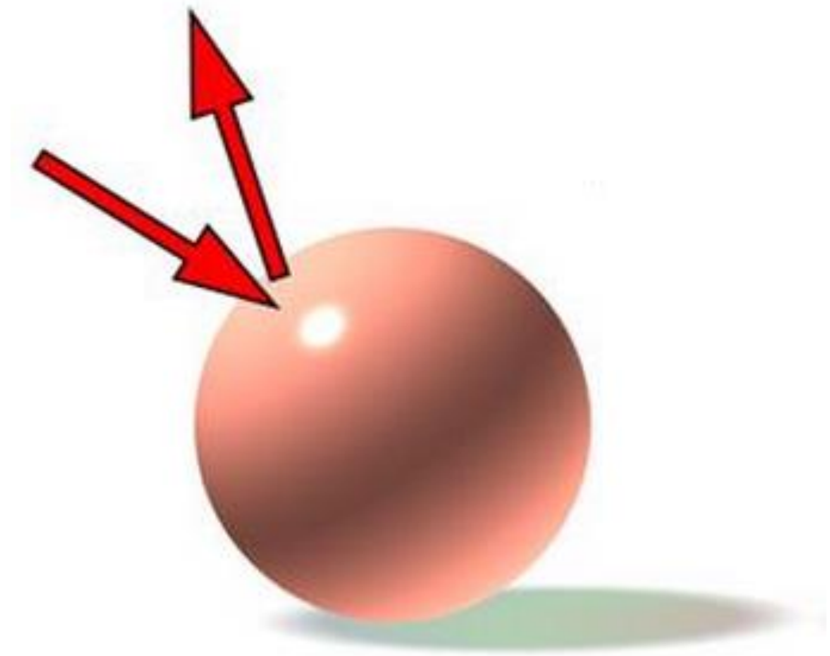
Secondary Rays

N, ***I***, ***R*** and ***T*** all lie in the same plane

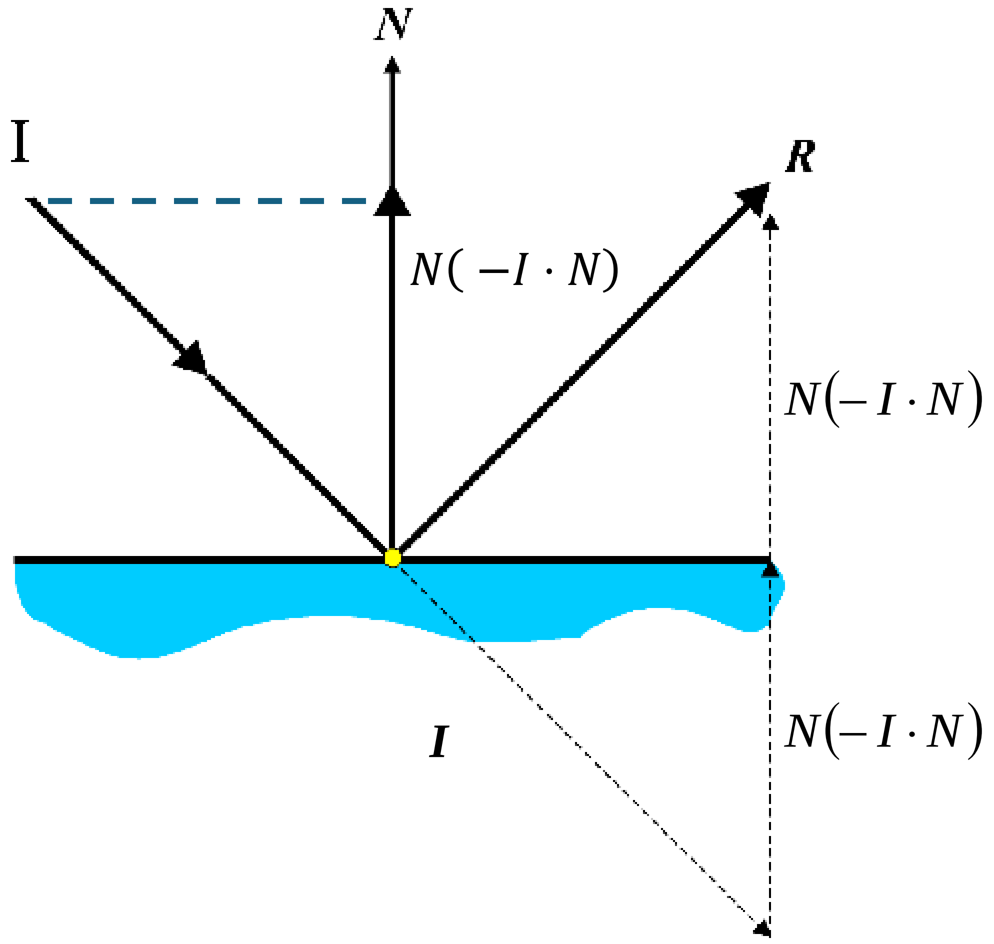


Reflection

- The law of reflection says that for specular reflection, the angle at which the wave is incident on the surface equals the angle at which it is reflected.



Reflected Ray



$$R = I + 2N(-I \cdot N)$$
$$= I - 2N(I \cdot N)$$

(-I . N) is the length of *I* projected onto *N*

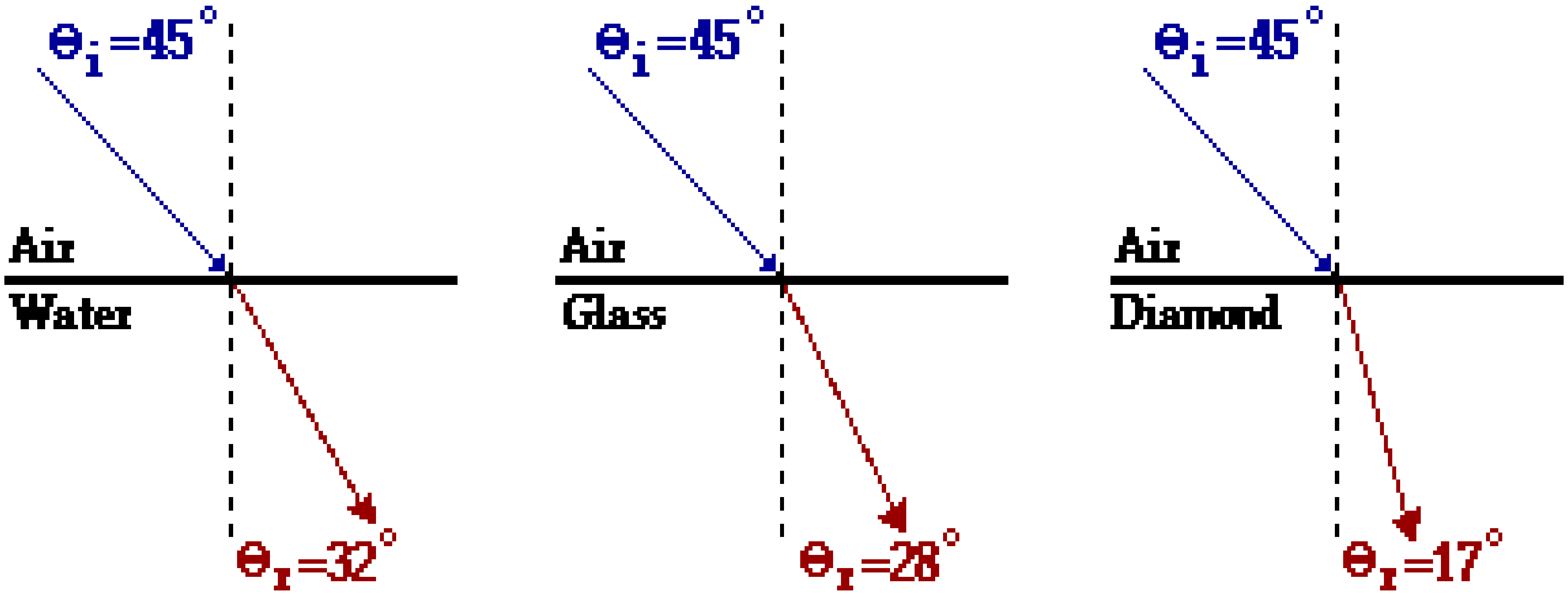
Refraction

- Refraction is the bending of the path of a light wave as it passes from one material to another material.
- The refraction occurs at the boundary and is caused by a change in the speed of the light wave upon crossing the boundary.
- The tendency of a ray of light to bend one direction or another is dependent upon whether the light wave speeds up or slows down upon crossing the boundary.

Credit: Google

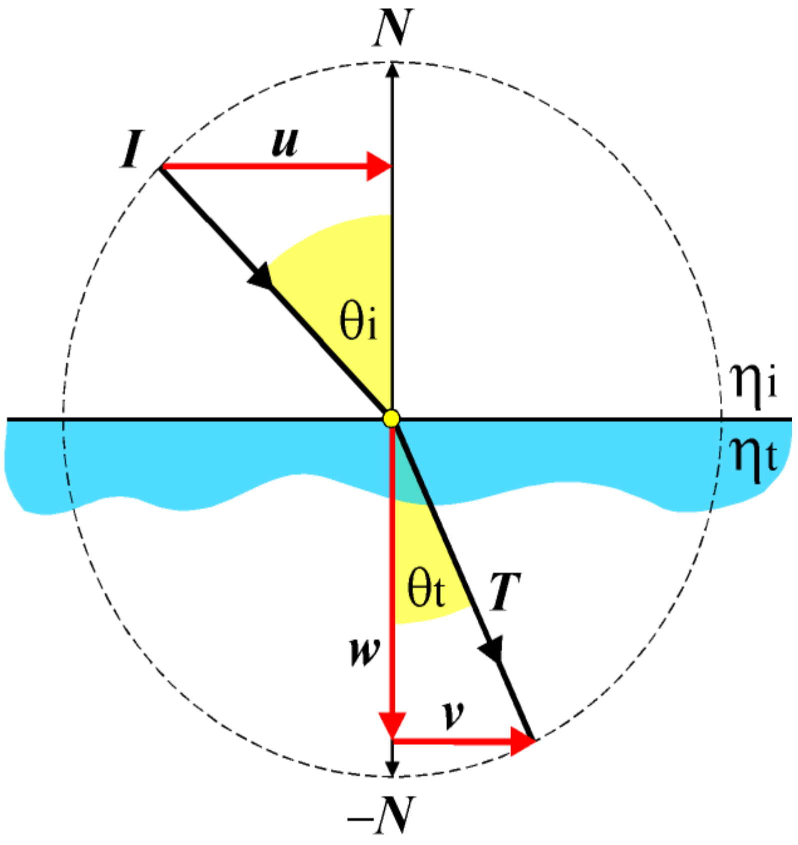
Refraction

- A comparison of the angle of refraction to the angle of incidence provides a good measure of the refractive ability of any given boundary.



Refracted Rays

- We use a geometric construction using *Snell's Law* to determine ***T***:

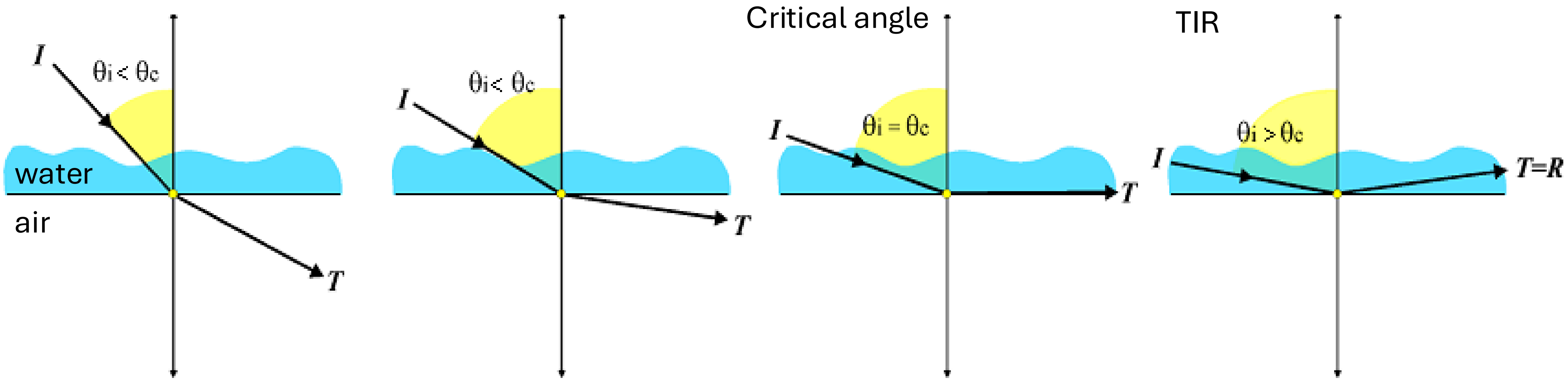


$$\eta = \frac{\eta_i}{\eta_t} = \frac{\sin\theta_t}{\sin\theta_i} = \frac{|\vec{v}|}{|\vec{u}|}$$

$$T = \vec{w} + \vec{v} = \vec{w} + \frac{|\vec{v}|}{|\vec{u}|}\vec{u} = \vec{w} + \eta\vec{u}$$

Total Internal Reflection

- *Total internal reflection* occurs when the incident angle exceeds the *critical angle* for the surface. This will only happen when passing from a *higher refractive index* to a *lower one,* e.g., from water to air.



Critical angle can be estimated when angle of refraction is 90 degrees, so $\frac{\eta_i}{\eta_t} = \frac{1}{\sin \theta_i}$

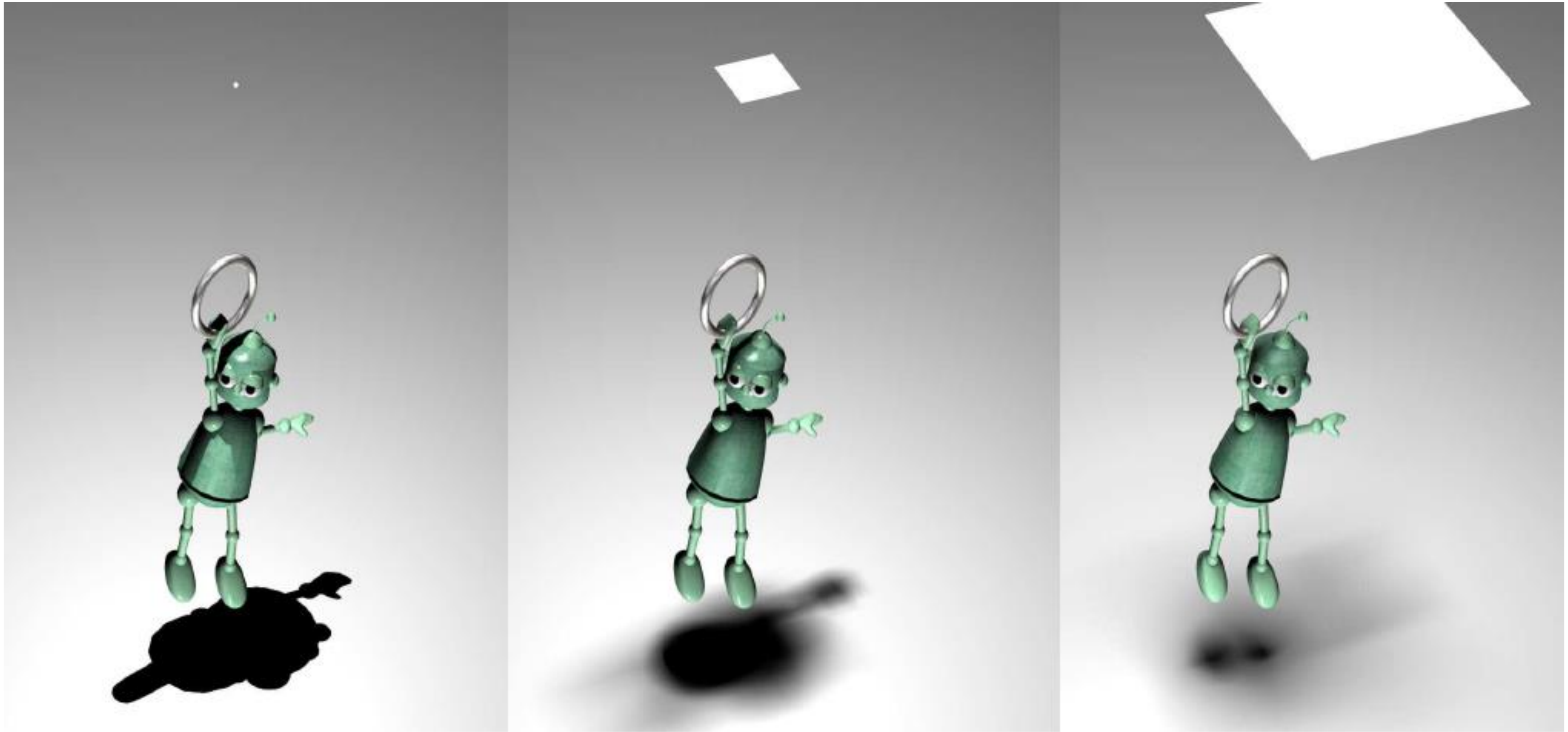
Shadows

- Whether a point is in shadow or not is determined by casting a ray from each intersection point to the light source

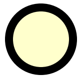
- If it intersects any object then the point of interest is deemed to be in shadow
    - Easier than ray/object intersections, as we only need to know if intersection has occurred (do not need to find the nearest object)

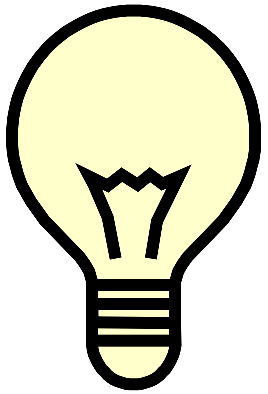
- Shadow calculations impose a computational overhead in ray tracing that increases rapidly as the number of light sources increases.

Soft Shadows



Point Light Source





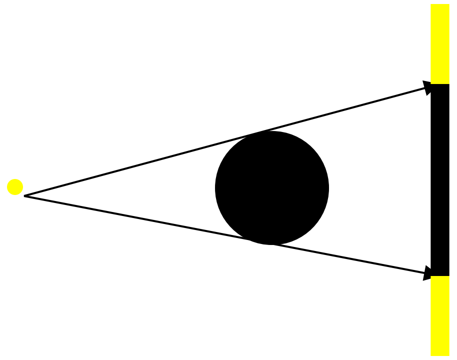
- A "point" light source is just a convenient model, and only allows for hard/sharp shadows.

- Consider a light bulb, for example. It is not an infinitesimally small point. It has volume.

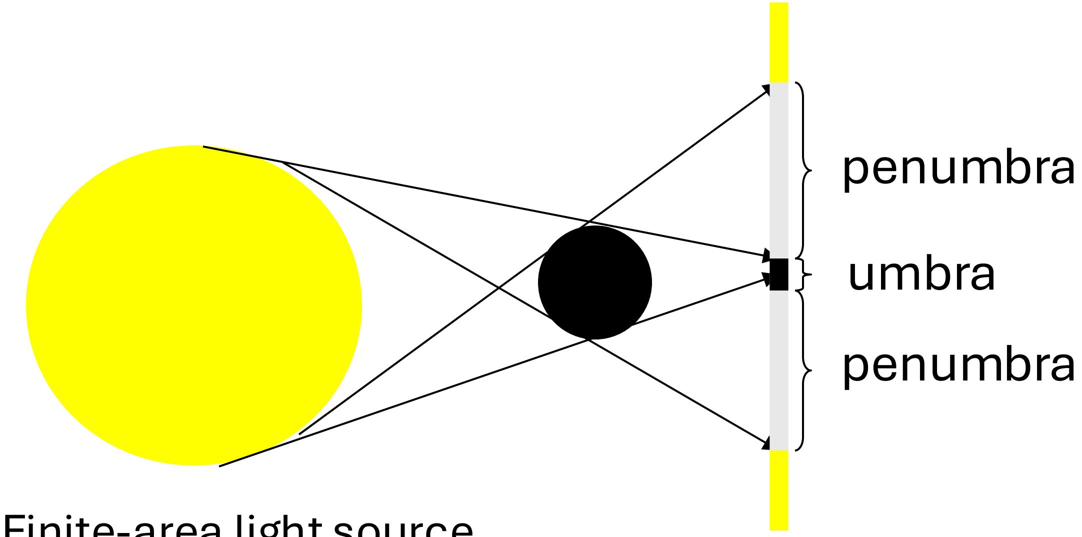
- Implication: Real light sources produce soft shadows.

Soft Shadows

- Shadows are not uniformly dark. The shadow is divided into two parts: the **umbra** and **penumbra**:
- **Umbra**: no light at all from the light source. Completely dark.
- **Penumbra**: some light from the light source. Partially dark.



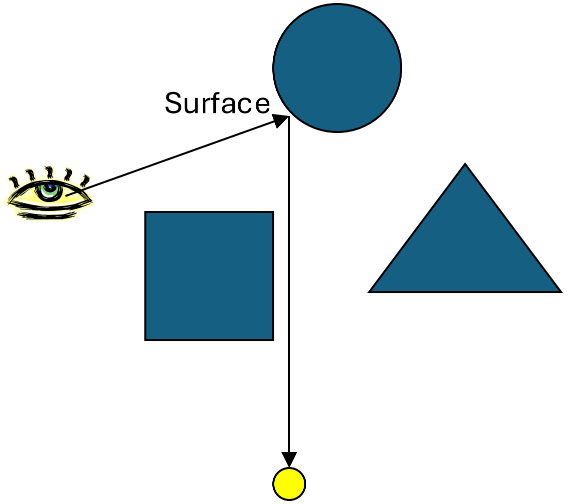
Point light source



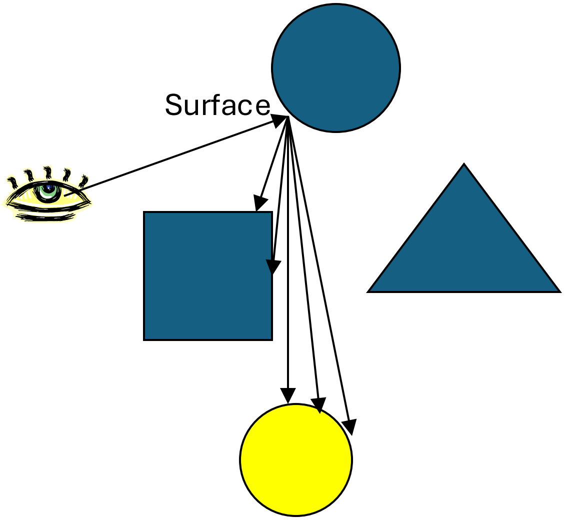
Finite-area light source

Soft Shadows in Ray Tracing

- For more realistic shadows, model light sources as **volumes** rather than points.
- Use a sphere to model a light source, rather than a point.
- This is often quite realistic because many light sources in real life are spherical, e.g. light bulbs and lanterns.
- When calculating lighting, shoot **several** rays to the light source, instead of just one ray.
- Calculate lighting for each ray, and take the average.



Point light source: The surface is completely lit by the light source.



Finite light source: 3/5 of the rays reach the light source. The surface is partially lighted.

Further Reading

- Physically Based Rendering: https://pbr-book.org/

Chapter 1.2: Photorealistic Rendering and the Ray-Tracing Algorithm.

Chapter 13: Light Transport I: Surface Reflection.