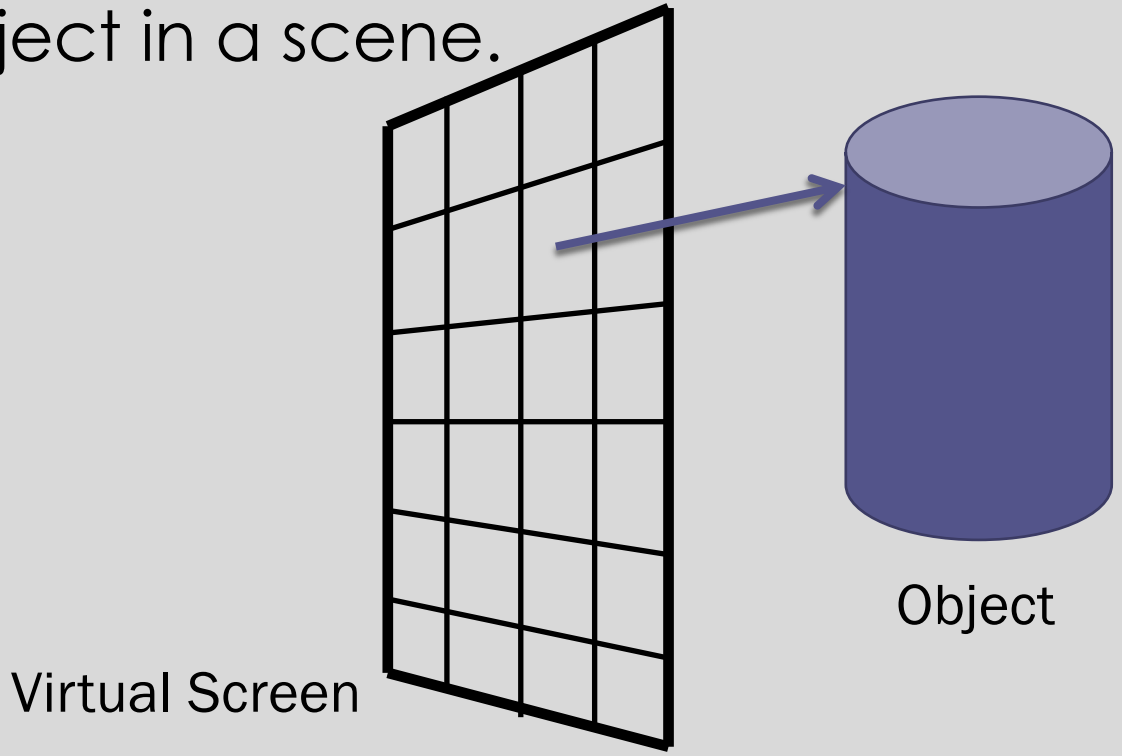
Ray-tracing

Lecturer: Carol O'Sullivan

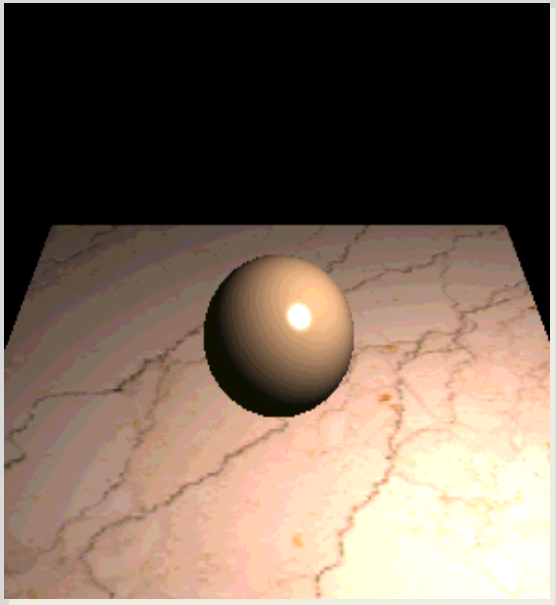
Credit: some slides from Rachel McDonnell

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.

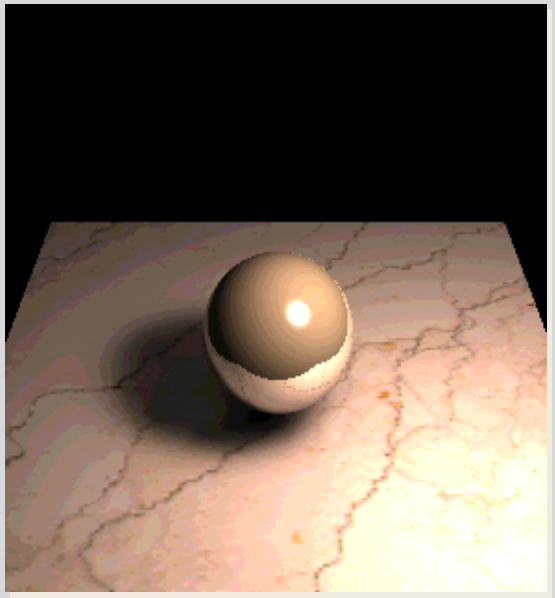


Local vs. Global Illumination



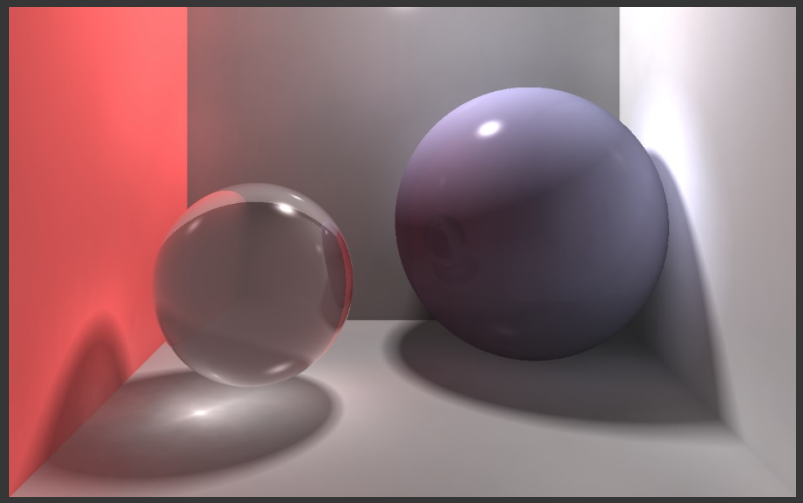
Local

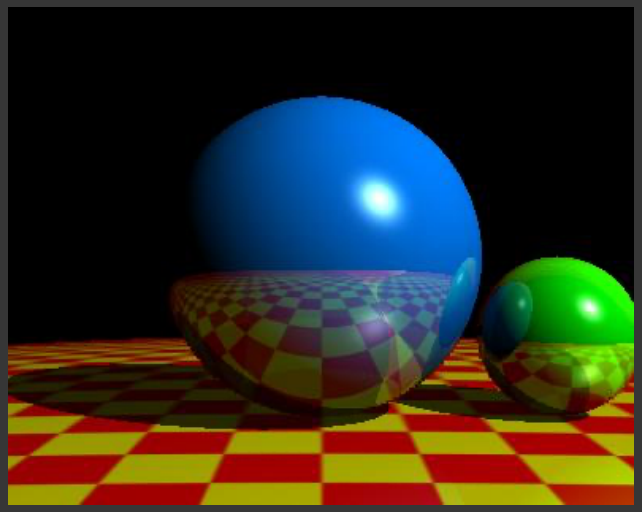
Illumination depends on local object & light sources only

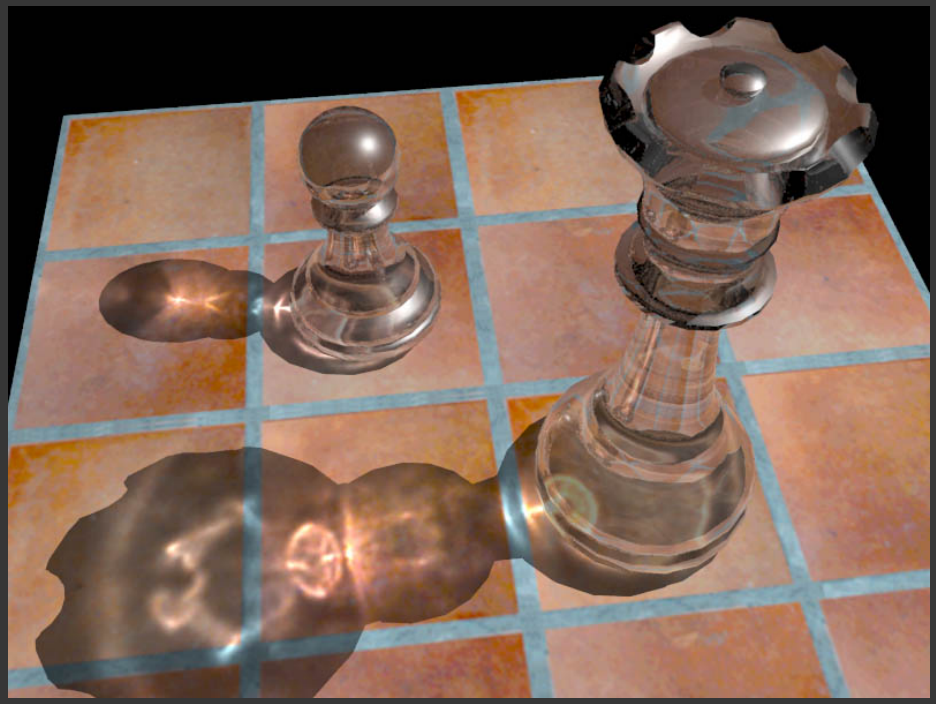


Global

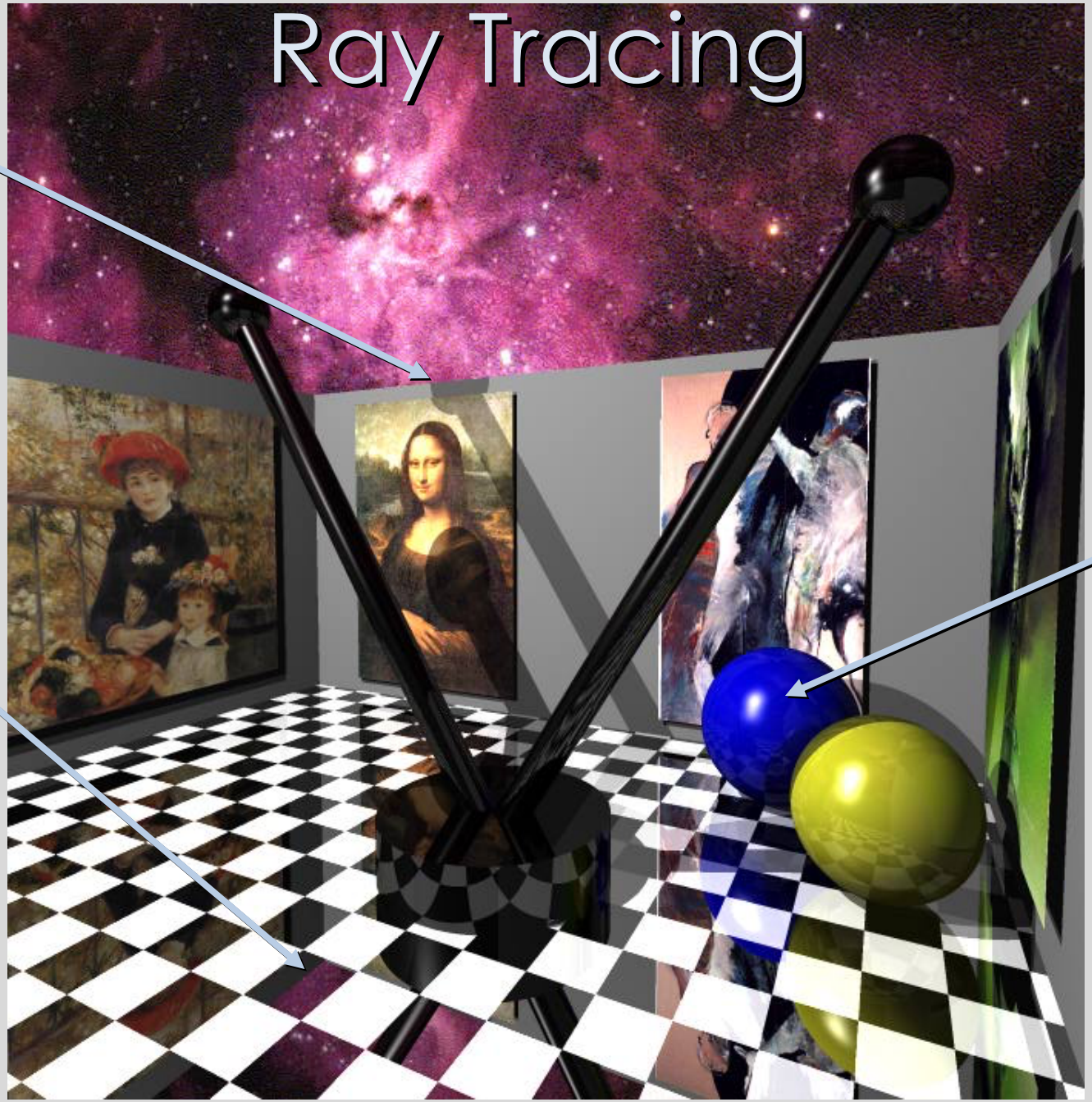
Illumination at a point can depend on any other point in the scene



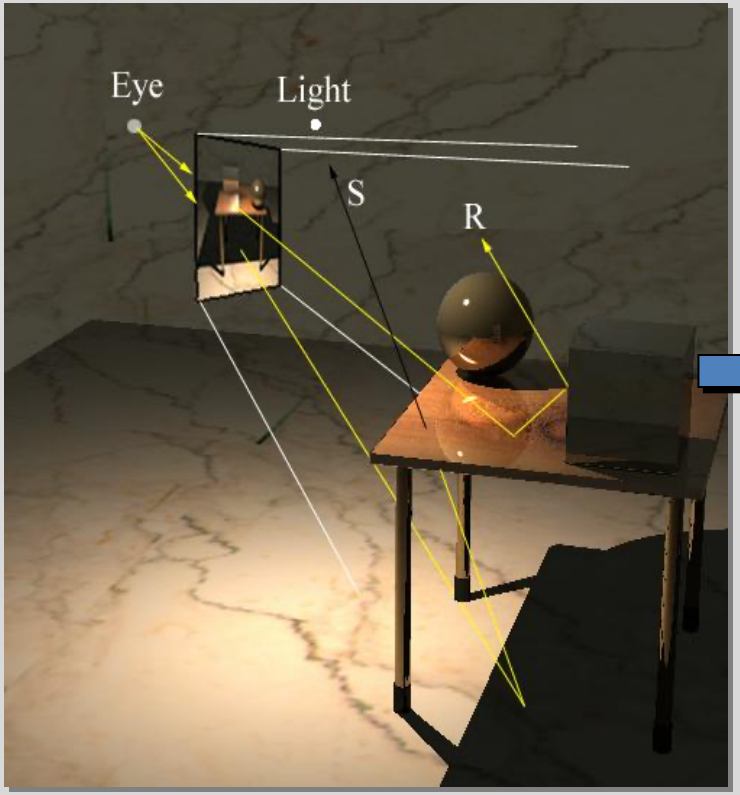




Ray Tracing


Sharp shadows
Perfectly specular reflections
Phong Illumination

View Dependent Solution (Ray Traced)

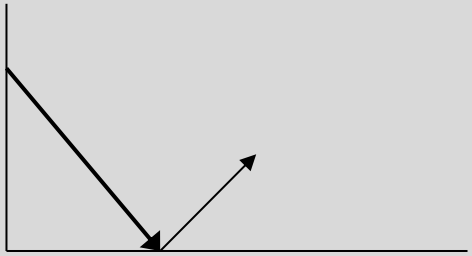


Scene Geometry

Solution determined only for directions through pixels in the viewport

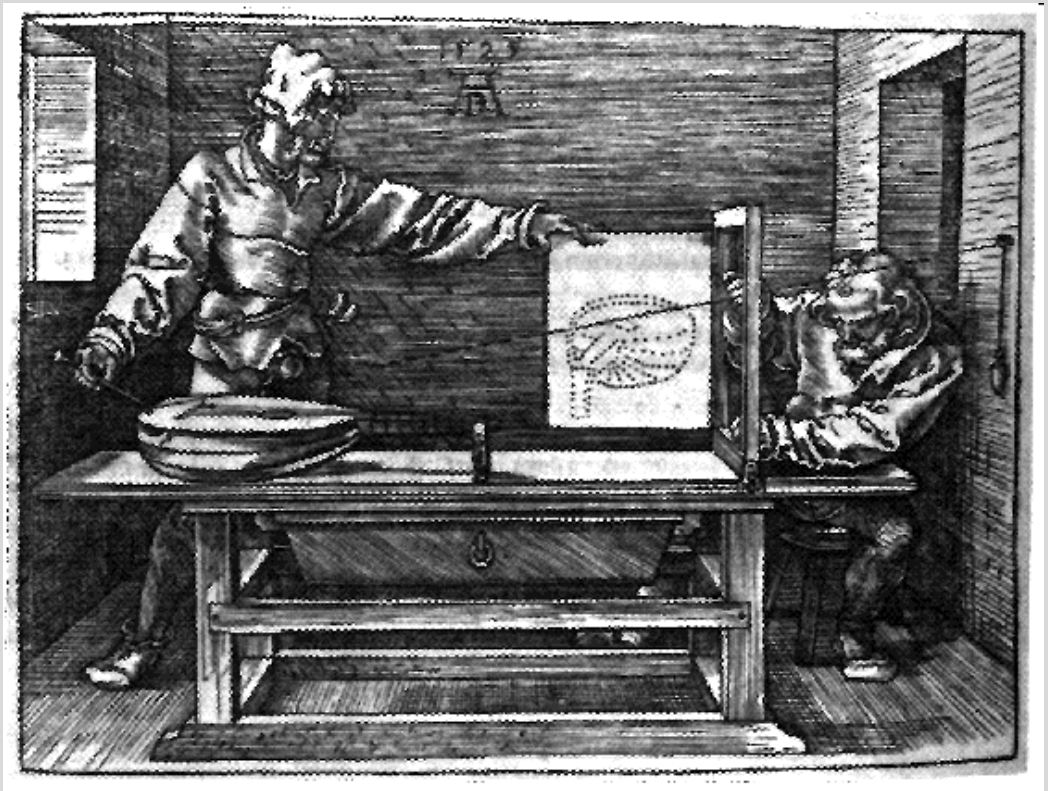
Ray-tracing

- Simplifies Light Transport
- specular to specular only, without the spread



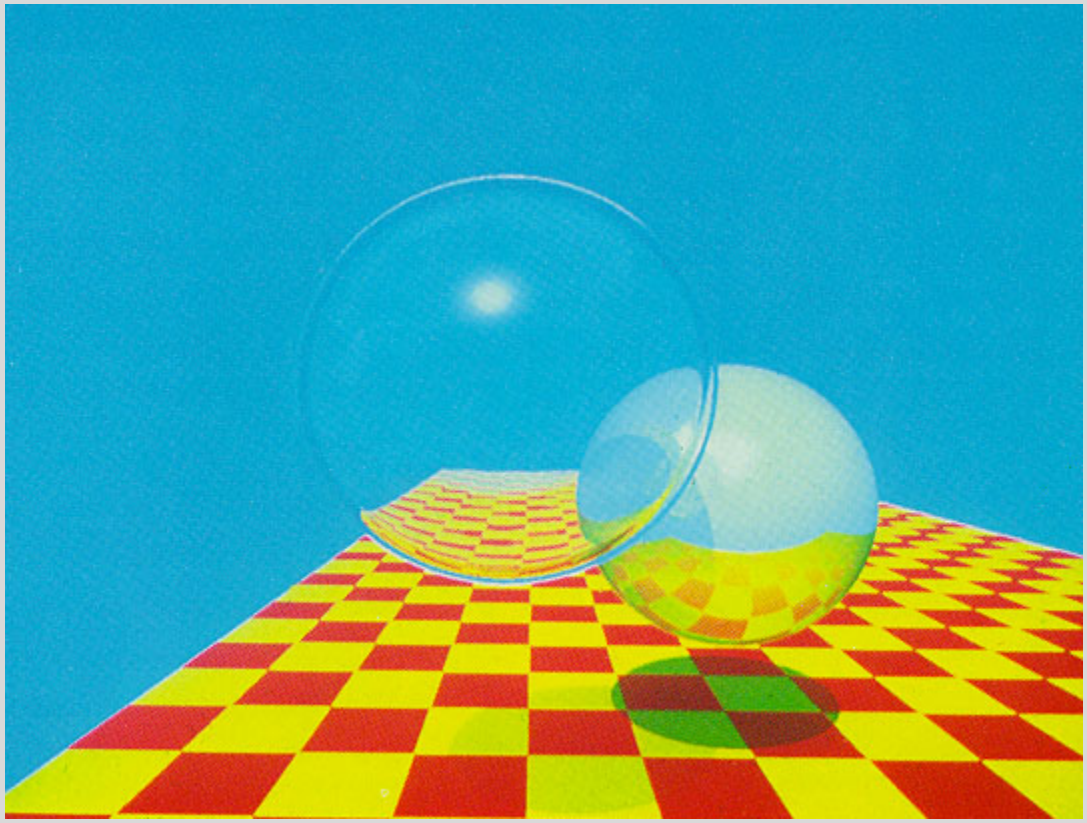
- local illumination rays are spread (empirically), but eye rays are not
  - reflections on an object, from other items in the scene, appear as if that object was a perfect mirror
  - rays traced through objects appear as if the object was a perfect transmitter
- Typical of "quick fixes" used in CG to achieve results that look acceptable.

Albrecht Dürer (1471-1528)



Ray-tracing based on ideas employed since early Renaissance artists e.g. daVinci & Dürer

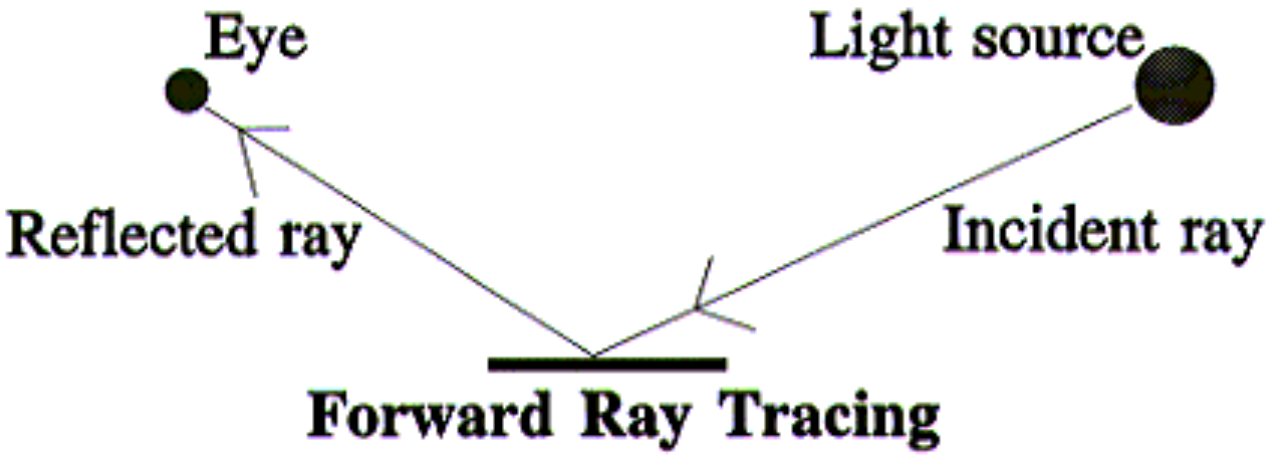
First ray-traced image

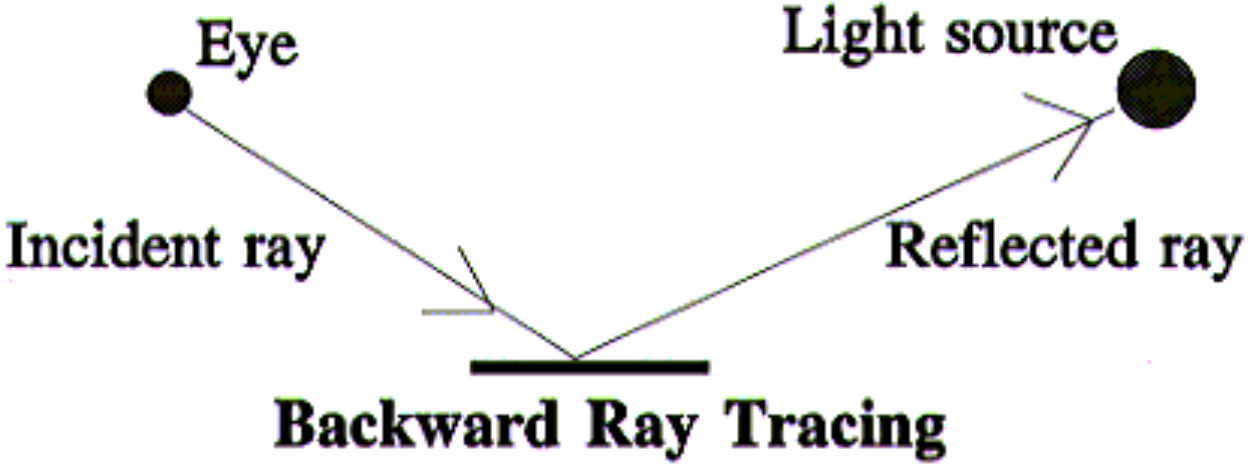


By Turner Whitted in 1979

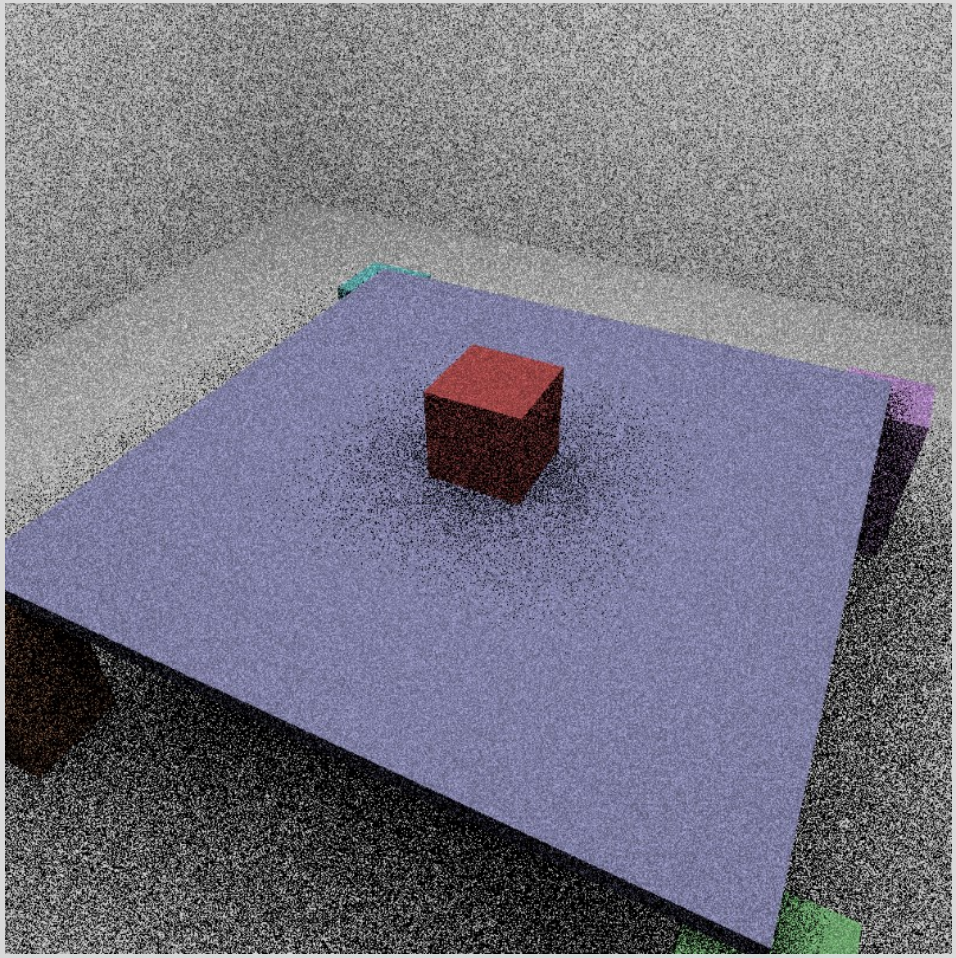
Ray-tracing history

- First used in computer graphics in 1980
  - "super-real" images at a high cost
  - 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


Eye
Light source
Reflected ray
Incident ray
Forward Ray Tracing


Eye
Light source
Incident ray
Reflected ray
Backward Ray Tracing

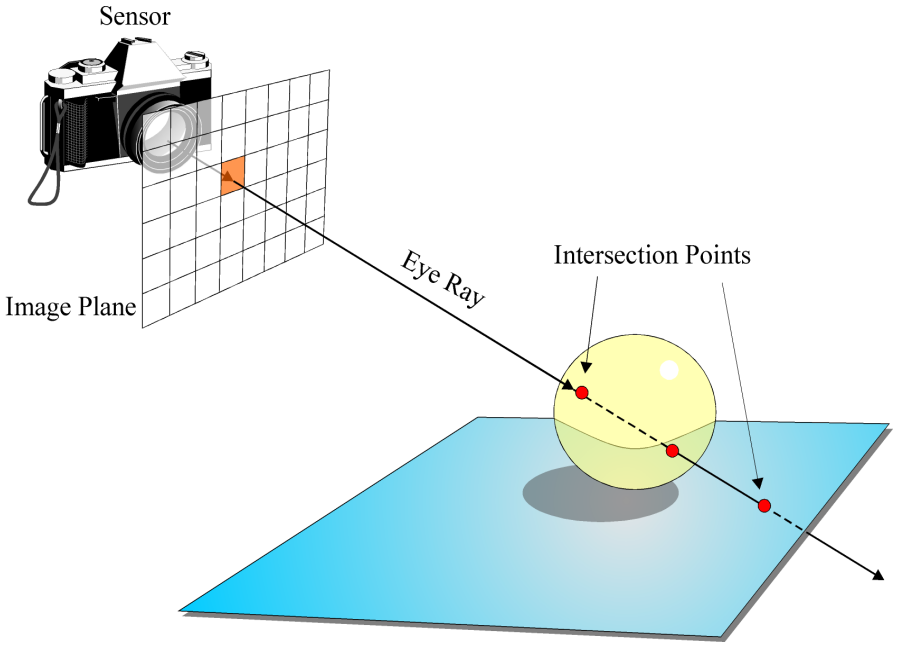
Forward Ray tracing



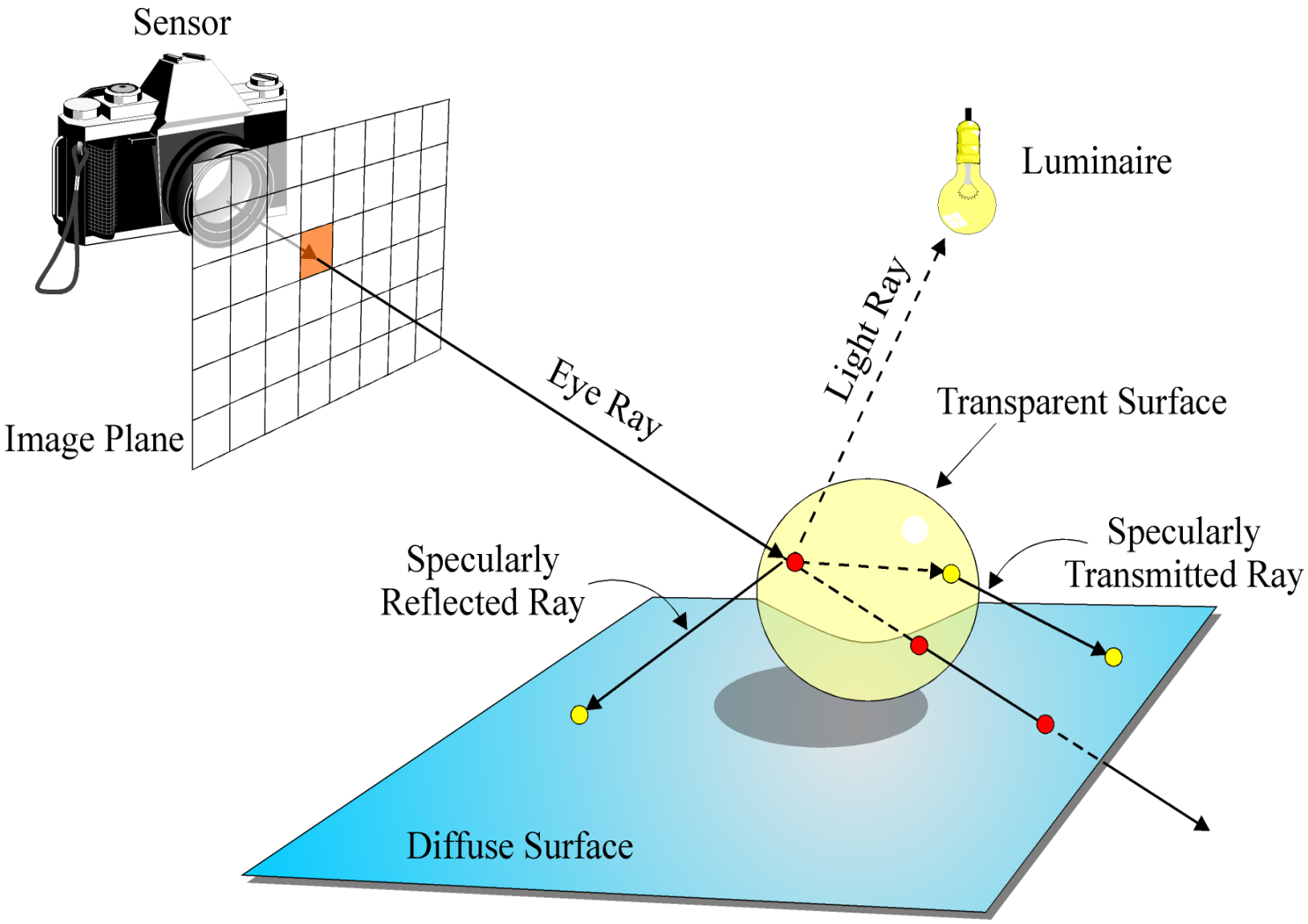
- Only a fraction of rays reach the image
- Many, many rays are required to get a value for each pixel

Backward Ray Tracing

- For each *pixel* in the viewport:
  - we trace a ray from the eye (called the *eye ray*) into the scene, through the pixel and
  - determine the first object hit by the ray ⇒ *ray casting*.
- We then shade this using an extended form of the Phong Model.



Ray Tracing


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

Ray Tracing

- The eye ray will typically intersect a number of objects, some more than once ⇒ *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.

Ray Tracing

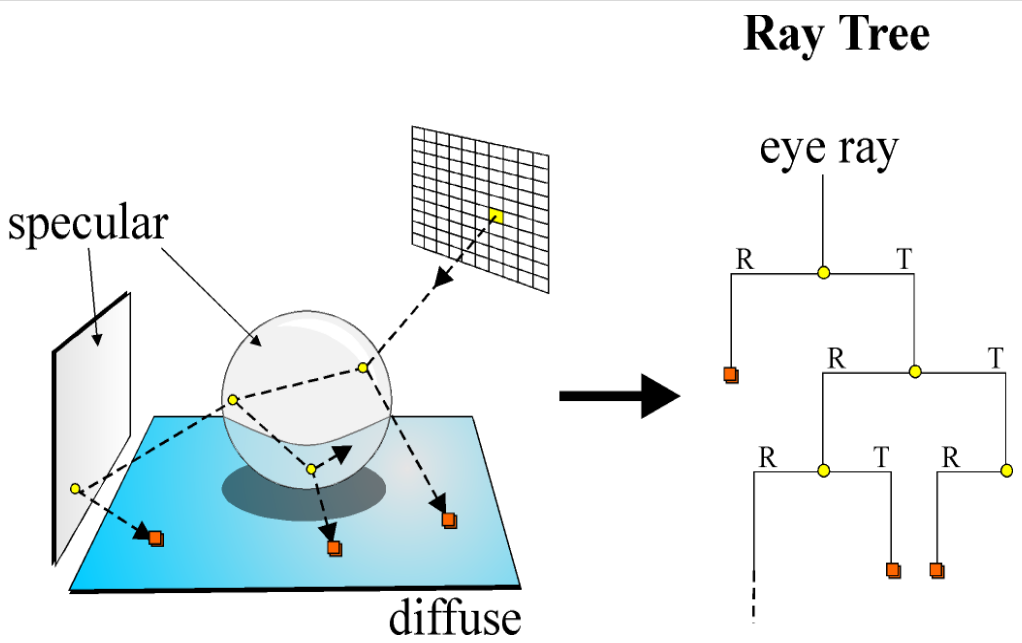
- *Whitted Illumination Model*

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

- Therefore, 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

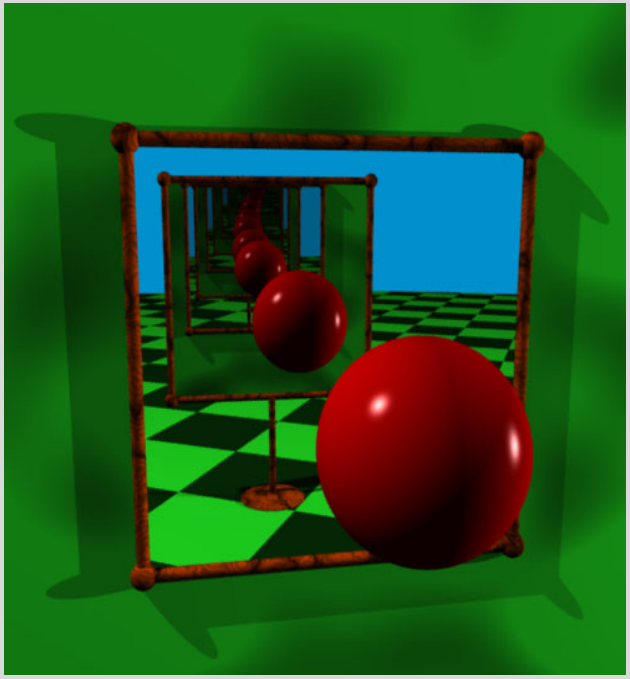
- The ray tracing algorithm is *recursive*, just as the *radiance equation is recursive*.
  - At each intersection we trace a specularly reflected and transmitted ray (if the surface is specular) or terminate the ray if diffuse.
- Thus we trace a ray *back in time* to determine its history, beginning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

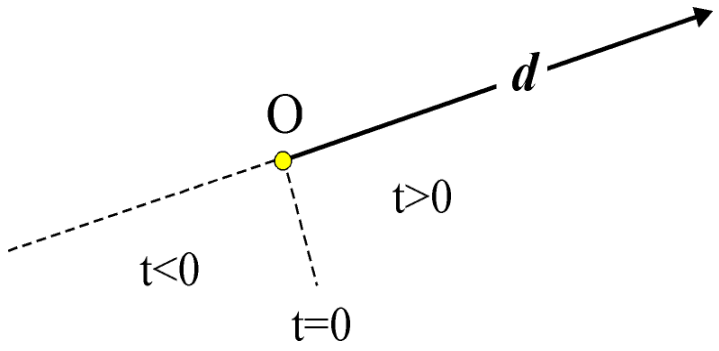


Very high maximum recursion level

The Ray

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

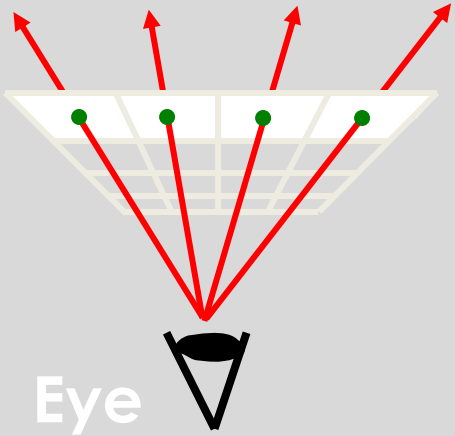
$$\mathbf{r} = O + t\vec{d} \quad t \geq 0$$



- 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.

The 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

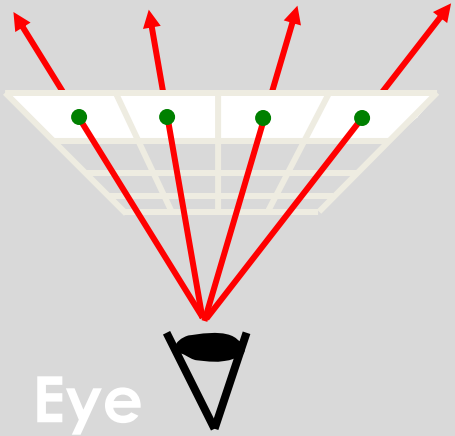
```
colour shade(ray, intersection)
{
    if no intersection
        return background colour

    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 += ρ_refl*shade(ray, intersection)
      }
    return colour
}
```

Ray Casting

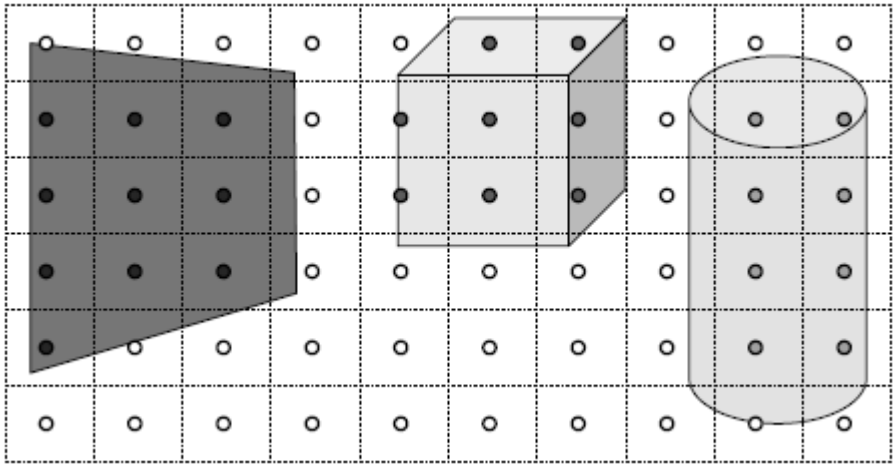
```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```




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

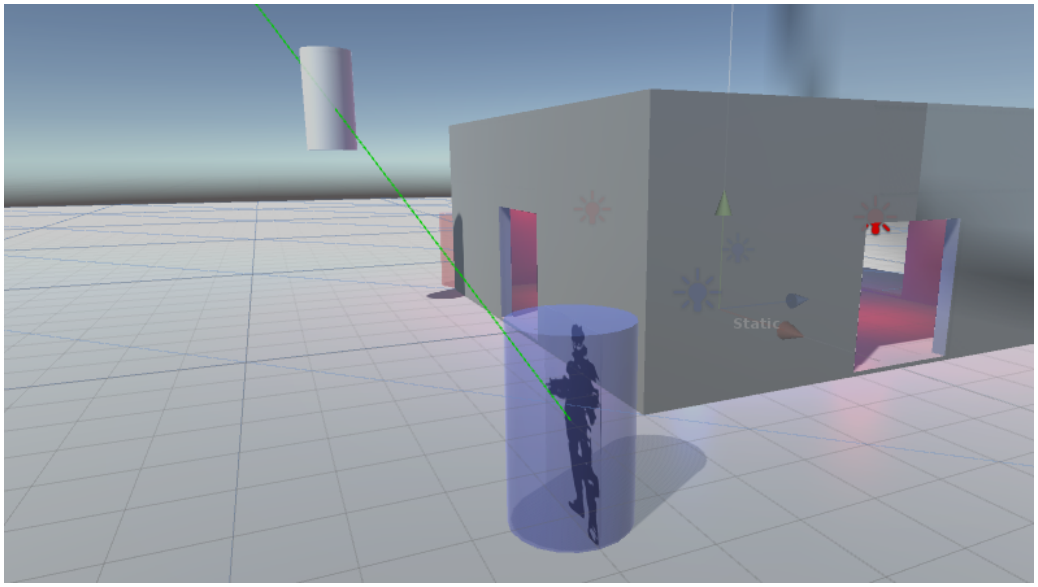
Ray Casting

- 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



Other uses of Ray Casting

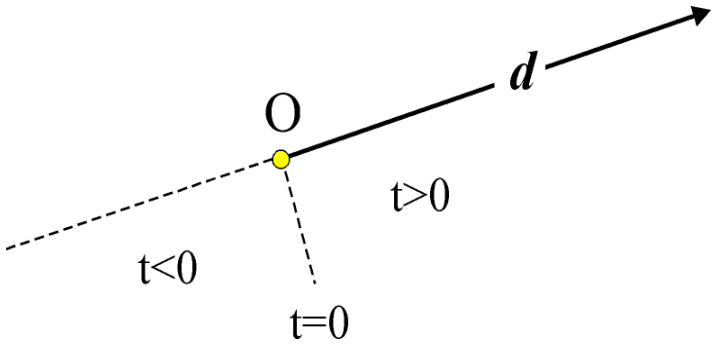
- Collision detection
- Shooting a gun - what does the bullet intersect?
- Line of sight - what's in front of me?
- Occlusion culling



The Ray

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

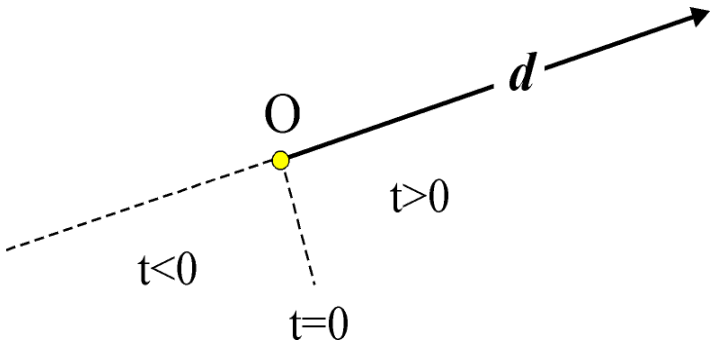
$$\mathbf{r} = O + t\vec{d} \quad t \geq 0$$



- 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* to the object at the point of intersection in order to perform shading calculations.



Ray Object Intersection Testing

- Usually objects are defined either *implicitly* or *explicitly* (*parametrically*)
- Implicit:

$$f(\vec{v}) = f(v_0, v_1, \ldots, v_n) \quad \begin{cases} < 0 \text{ if } \vec{v} \text{ inside surface} \\ = 0 \text{ if } \vec{v} \text{ on surface} \\ > 0 \text{ if } \vec{v} \text{ outside surface} \end{cases}$$

  - the set of points on the surface are the zeros of the function *f*.

- Explicit:

$$S^n = f(\alpha_0, \alpha_1, \ldots, \alpha_n)$$

  - α_n are the generating parameters and usually have infinite ranges
  - for surfaces in 3-space, S^2, there will be 2 generating parameters.
  - we iterate over all parameters to generate the set of all points on the surface.

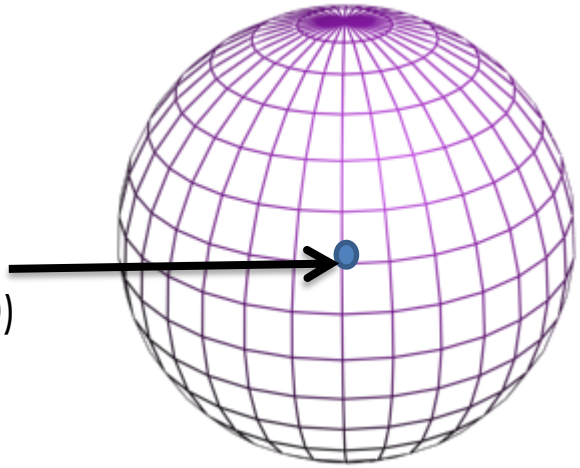
The Sphere

- 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?

Centre = (0, 0, 0)



10cm

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.

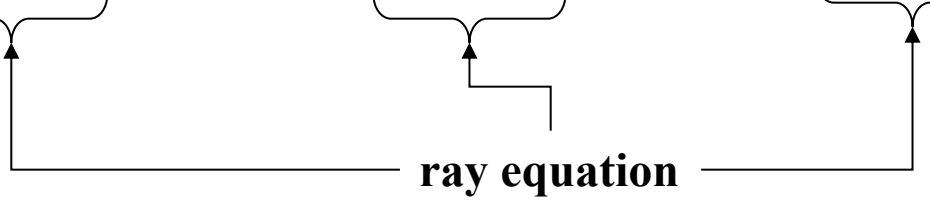
Ray Sphere Intersection

- 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*:

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



Problem

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

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):

$$At^2 + Bt + C = 0$$

$$A = \left(d_x^2 + d_y^2 + d_z^2\right) = 1$$

$$B = 2d_x\left(O_x - C_x\right) + 2d_y\left(O_y - C_y\right) + 2d_z\left(O_z - C_z\right)$$

$$C = \left(O_x - C_x\right)^2 + \left(O_y - C_y\right)^2 + \left(O_z - C_z\right)^2 - r^2$$

- 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

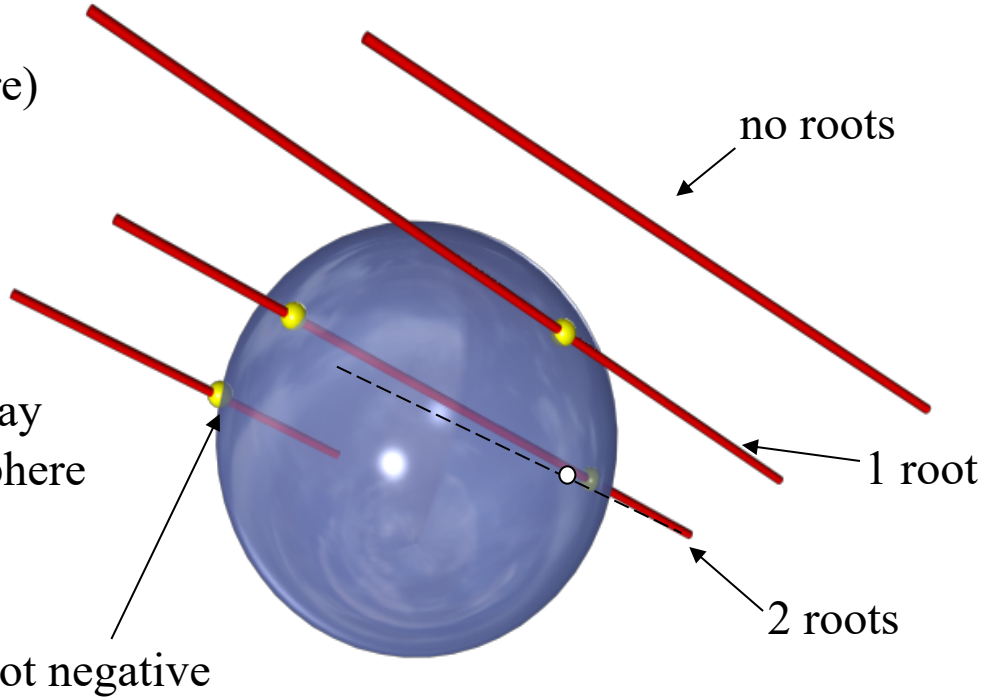
- Depending on the number of *real roots* we have a number of outcomes which have nice geometric interpretations
- we use the *discriminant*: $d = B^2 - 4C$

$d = 0 \Rightarrow$ 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 normal to the surface of the sphere 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:

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

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