

# Ray Tracing

# Reading

Required:

- ♦ Watt, sections 1.3-1.4, 12.1-12.5.1.
- ♦ Watt errata

Further reading:

- ♦ A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989. [In the lab.]
- ♦ K. Turkowski, "Properties of Surface Normal Transformations," Graphics Gems, 1990, pp. 539-547. [In the reader.]
- ♦ T. Whitted. An improved illumination model for shaded display. Communications of the ACM 23(6), 343-349, 1980. [In the reader.]

# Geometric optics

Modern theories of light treat it as both a wave and a particle.

We will take a combined and somewhat simpler view of light – the view of **geometric optics**.

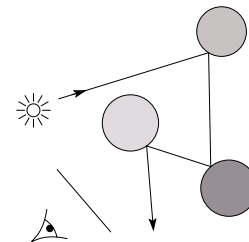
Here are the rules of geometric optics:

- ♦ Light is a flow of photons with wavelengths. We'll call these flows "light rays."
- ♦ Light rays travel in straight lines in free space.
- ♦ Light rays do not interfere with each other as they cross.
- ♦ Light rays obey the laws of reflection and refraction.
- ♦ Light rays travel from the light sources to the eye, but the physics is invariant under path reversal (reciprocity).

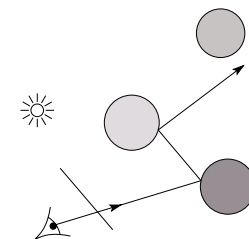
# Eye vs. light ray tracing

Where does light begin?

At the light: light ray tracing (a.k.a., forward ray tracing or photon tracing)



At the eye: eye ray tracing (a.k.a., backward ray tracing)

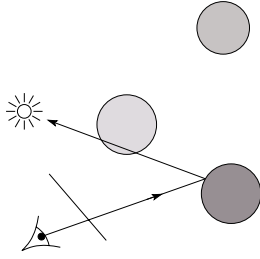


We will generally follow rays from the eye into the scene.

# Precursors to ray tracing

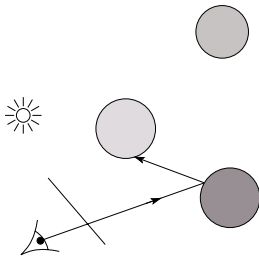
## Local illumination

- Cast one eye ray, then shade according to light



## Appel (1968)

- Cast one eye ray + one ray to light

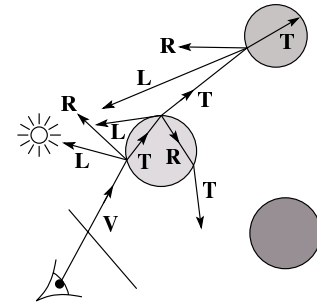


5

# Whitted ray-tracing algorithm

In 1980, Turner Whitted introduced ray tracing to the graphics community.

- Combines eye ray tracing + rays to light
- Recursively traces rays



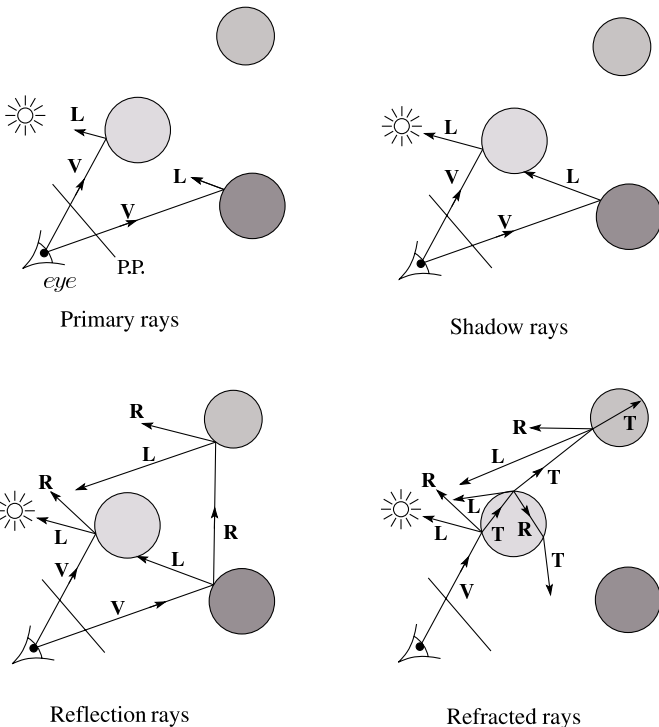
## Algorithm:

- For each pixel, trace a **primary ray** in direction  $V$  to the first visible surface.
- For each intersection, trace **secondary rays**:
  - Shadow rays** in directions  $L_i$  to light sources
  - Reflected ray** in direction  $R$ .
  - Refracted ray or transmitted ray** in direction  $T$ .

6

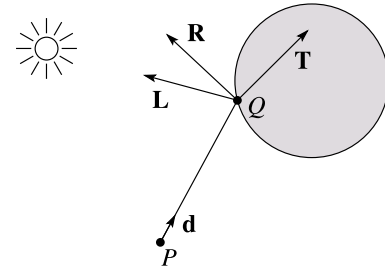
# Whitted algorithm (cont'd)

Let's look at this in stages:



7

# Shading



A ray is defined by an origin  $p$  and a unit direction  $d$  and is parameterized by  $t$ :

$$P + td$$

Let  $I(P, d)$  be the intensity seen along that ray. Then:

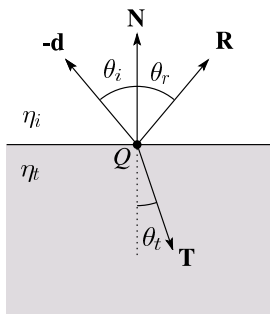
$$I(P, d) = I_{\text{direct}} + I_{\text{reflected}} + I_{\text{transmitted}}$$

where

- $I_{\text{direct}}$  is computed from the Phong model
- $I_{\text{reflected}} = k_r I(Q, R)$
- $I_{\text{transmitted}} = k_t I(Q, T)$

8

## Reflection and transmission



Law of reflection:

$$\theta_i = \theta_r$$

Snell's law of refraction:

$$\eta_i \sin \theta_i = \eta_t \sin \theta_t$$

where  $\eta_i$ ,  $\eta_t$  are **indices of refraction**.

9

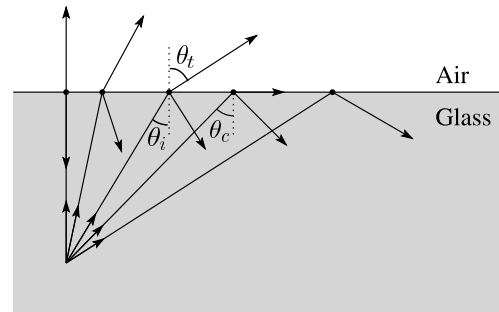
## Total Internal Reflection

The equation for the angle of refraction can be computed from Snell's law:

What happens when  $\eta_i > \eta_t$ ?

When  $\theta_i$  is exactly  $90^\circ$ , we say that  $\theta_i$  has achieved the "critical angle"  $\theta_c$ .

For  $\theta_i > \theta_c$ , *no rays are transmitted*, and only reflection occurs, a phenomenon known as "total internal reflection" or TIR.



10

## Error in Watt!!

In order to compute the refracted direction, it is useful to compute the cosine of the angle of refraction in terms of the incident angle and the ratio of the indices of refraction.

On page 24 of Watt, he develops a formula for computing this cosine. Notationally, he uses  $\mu$  instead of  $\eta$  for the index of refraction in the text, but uses  $\eta$  in Figure 1.16(!?), and the angle of incidence is  $\phi$  and the angle of refraction is  $\theta$ .

Unfortunately, he makes a grave error in computing  $\cos \theta$ .

The last equation on page 24 should read:

$$\cos \theta = \sqrt{1 - \mu^2 (1 - \cos^2 \phi)}$$

In any case, see the errata for corrections that you can write into your books.

11

## Ray-tracing pseudocode

We build a ray traced image by casting rays through each of the pixels.

**function** *tracelImage* (scene):

**for each** pixel (i,j) in image

$S = \text{pixelToWorld}(i,j)$

$P = \text{COP}$

$d = (S - P) / \|S - P\|$

$I(i,j) = \text{traceRay}(\text{scene}, P, d)$

  end for

**end function**

12

## Ray-tracing pseudocode, cont'd

```
function traceRay(scene, P, d):  
  (t, N, mtrl) ← scene.intersect(P, d)  
  Q ← ray(P, d) evaluated at t  
  I = shade( )  
  R = reflectDirection( )  
  I ← I + mtrl.kr * traceRay(scene, Q, R)  
  if ray is entering object then  
    ni = index_of_air  
    nt = mtrl.index  
  else  
    ni = mtrl.index  
    nt = index_of_air  
  if (notTIR( )) then  
    T = refractDirection( )  
    I ← I + mtrl.kt * traceRay(scene, Q, T)  
  end if  
  return I  
end function
```

13

## Terminating recursion

Q: How do you bottom out of recursive ray tracing?

Possibilities:

## Shading pseudocode

Next, we need to calculate the color returned by the *shade* function.

```
function shade(mtrl, scene, Q, N, d):  
  I ← mtrl.ke + mtrl.ka * scene->Ia  
  for each light source λ do:  
    atten = λ -> distanceAttenuation( ) *  
    λ -> shadowAttenuation( )  
    I ← I + atten*(diffuse term + spec term)  
  end for  
  return I  
end function
```

15

## Shadow attenuation

Computing a shadow can be as simple as checking to see if a ray makes it to the light source.

For a point light source:

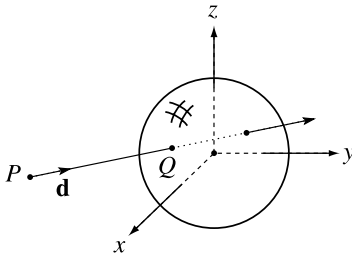
```
function PointLight::shadowAttenuation(scene, P)  
  d = (this.position - P).normalize()  
  (t, N, mtrl) ← scene.intersect(P, d)  
  Q ← ray(t)  
  if Q is before the light source then:  
    atten = 0  
  else  
    atten = 1  
  end if  
  return atten  
end function
```

Q: What if there are transparent objects along a path to the light source?

14

16

## Intersecting rays with spheres



### Given:

- The coordinates of a point along a ray passing through  $P$  in the direction  $\mathbf{d}$  are:

$$x = P_x + td_x$$

$$y = P_y + td_y$$

$$z = P_z + td_z$$

- A unit sphere  $S$  centered at the origin defined by the equation:

**Find:** The  $t$  at which the ray intersects  $S$ .

17

## Intersecting rays with spheres

### Solution by substitution:

$$x^2 + y^2 + z^2 - 1 = 0$$

$$(P_x + td_x)^2 + (P_y + td_y)^2 + (P_z + td_z)^2 - 1 = 0$$

$$at^2 + bt + c = 0$$

where

$$a = d_x^2 + d_y^2 + d_z^2$$

$$b = 2(P_x d_x + P_y d_y + P_z d_z)$$

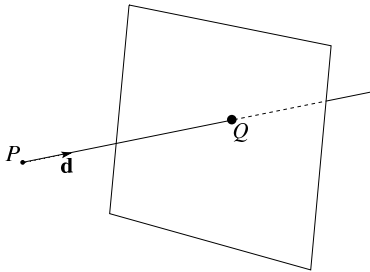
$$c = P_x^2 + P_y^2 + P_z^2 - 1$$

**Q:** What are the solutions of the quadratic equation in  $t$  and what do they mean?

**Q:** What is the normal to the sphere at a point  $(x, y, z)$  on the sphere?

18

## Ray-plane intersection



We can write the equation of a plane as:

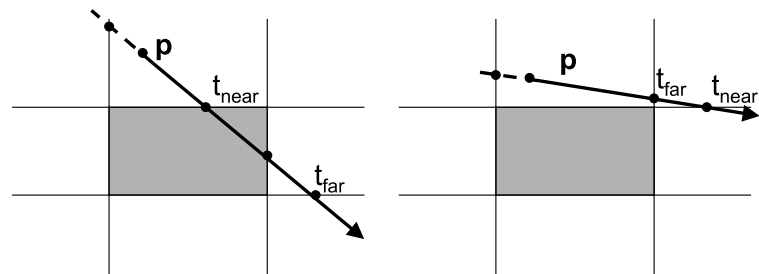
$$ax + by + cz + d = 0$$

The coefficients  $a$ ,  $b$ , and  $c$  form a vector that is normal to the plane,  $\mathbf{n} = [a \ b \ c]^T$ . Thus, we can rewrite the plane equation as:

We can solve for the intersection parameter (and thus the point):

19

## Ray with cube intersection

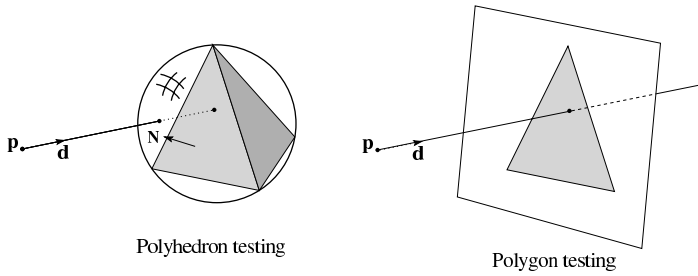


To intersect a ray with an axis-aligned cube

- for each pair of parallel planes, compute  $t$  intersection values for both. Let  $t_{\text{near}}$  be the smaller,  $t_{\text{far}}$  be the larger
- let  $t_1 = \text{largest } t_{\text{near}}$ ,  $t_2 = \text{smallest } t_{\text{far}}$
- ray intersects cube if:
  - intersection point given by:

20

## Intersecting rays with polyhedra



To intersect a ray with a polyhedron:

- ◆ Test intersection of ray with bounding sphere.
- ◆ Locate the “front-facing” faces of the polyhedron with

$$\mathbf{d} \cdot \mathbf{N}$$

- ◆ Intersect the ray with each front face's supporting plane.
- ◆ Use a point-in-polygon test to see if the ray is inside the face.
- ◆ Sort intersections according to smallest  $t$ .

21

## Intersecting with xformed geometry

In many cases, objects such as spheres, cylinders, and boxes will be placed using transformations. What if the object being intersected were transformed by a matrix  $M$ ?

Apply  $M^{-1}$  to the ray first and intersect in object (local) coordinates!

23

## Epsilons

Due to finite precision arithmetic, we do not always get the exact intersection at a surface.

Q: What kinds of problems might this cause?

Q: How might we resolve this?

22

## Intersecting with xformed geometry

The intersected normal is in object (local) coordinates. How do we transform it to world coordinates?

24

# Summary

What to take home from this lecture:

1. The meanings of all the boldfaced terms.
2. Enough to implement basic recursive ray tracing.
3. How reflection and transmission directions are computed.
4. How ray--object intersection tests are performed on spheres, planes, and cubes
5. How ray epsilons are used.
6. How intersections with transformed geometry are done.