Ray Tracing

Reading

Required:

- Watt, sections 1.3-1.4, 12.1-12.5.1 (handout)

Further reading:

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

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**.

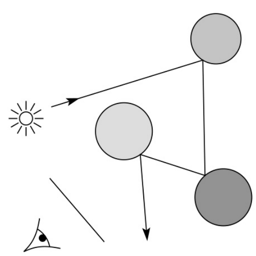
Here are the rules of geometric optics:

- 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 form the light sources to the eye, but the physics is invariant under path reversal (reciprocity).

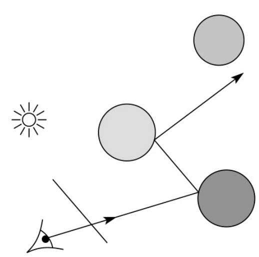
Eye vs. light ray tracing

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)

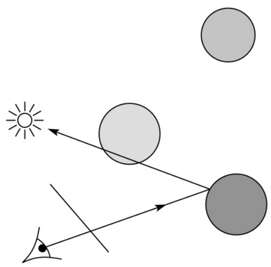


We will generally follow rays from the eye into the scene.

Precursors to ray tracing

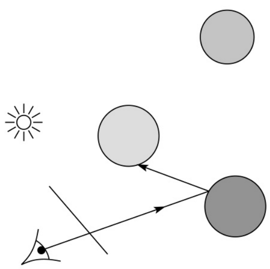
Local illumination

- Cast one eye ray, then shade according to light



Appel (1968)

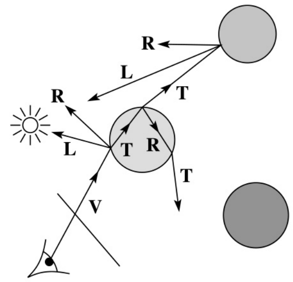
- Cast one eye ray + one ray to light



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:

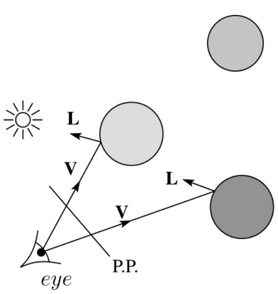
1. For each pixel, trace a **primary ray** in direction **V** to the first visible surface.

2. For each intersection, trace **secondary rays**:

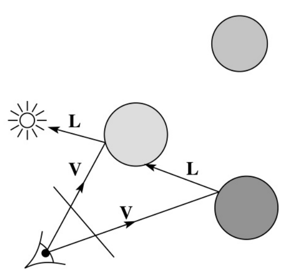
- **Shadow rays** in directions $\mathbf{L}_i$ to light sources
- **Reflected ray** in direction **R**.
- **Refracted ray** or **transmitted ray** in direction **T**.

Whitted algorithm (cont'd)

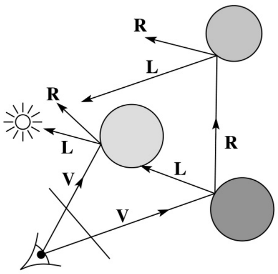
Let's look at this in stages:



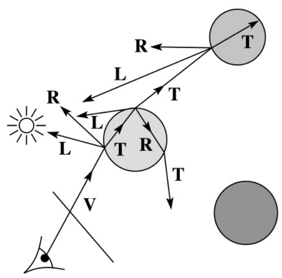
Primary rays



Shadow rays

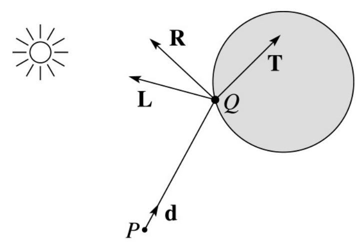


Reflection rays



Refracted rays

Shading



A ray is defined by an origin **P** and a unit direction **d** and is parameterized by t:

$$P + t\mathbf{d}$$

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

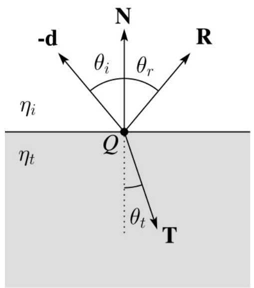
$$I(P, \mathbf{d}) = I_{\text{direct}} + I_{\text{reflected}} + I_{\text{transmitted}}$$

where

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

Typically, we set $k_r = k_s$ and $k_t = 1 - k_s$.

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 η_i , η_t are **indices of refraction**.

In all cases, **R** and **T** are co-planar with **d** and **N**.

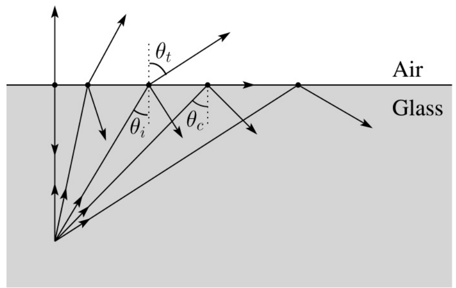
Total Internal Reflection

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

What happens when $\eta_i > \eta_t$?

When θ_t is exactly 90°, we say that θ_I has achieved the "critical angle" θ_c .

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



Watt handout

Watt uses different symbols. Here is the translation between them:

$$\mathbf{I} = -\mathbf{d}$$
$$\phi = \theta_i$$
$$\theta = \theta_r$$
$$\mu_1 = \eta_i$$
$$\mu_2 = \eta_r$$

Also, Watt had some important errors that I have already corrected in the handout.

But, if you're consulting the original text, be sure to refer to the errata posted on the syllabus for corrections.

Ray-tracing pseudocode

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

function *traceImage* (scene):
    for each pixel (i,j) in image
        S **=** *pixelToWorld*(i,j)
        P = **COP**
        d = (*S* - *P*)/|| *S* – *P*||
        I(i,j) = *traceRay*(scene, *P*, **d**)
    end for
end function

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.k_r ∗ *traceRay*(scene, *Q*, **R**)

if ray is entering object **then**

n_i = index_of_air

n_t = mtrl.index

else

n_i = mtrl.index

n_t = index_of_air

if (*notTIR* ( )) **then**

T = *refractDirection* ( )

I ← I + mtrl.k_t ∗ *traceRay*(scene, *Q*, **T**)

end if

return I

end function

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.k_e + mtrl. k_a * scene->I_a

for each light source ℓ **do**:

atten = ℓ -> *distanceAttenuation*( ) *

ℓ -> *shadowAttenuation*( )

I ← I + atten*(diffuse term + spec term)

end for

return I

end function

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**)

Compute t_{light}

if (t < t_{light}) **then**:

atten = 0

else

atten = 1

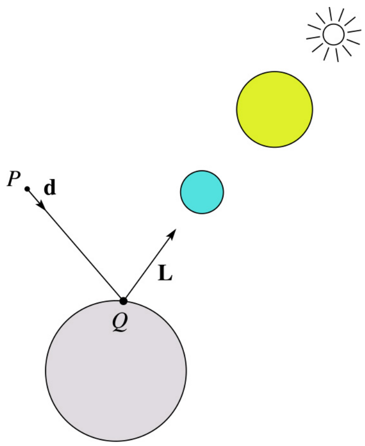
end if

return atten

end function

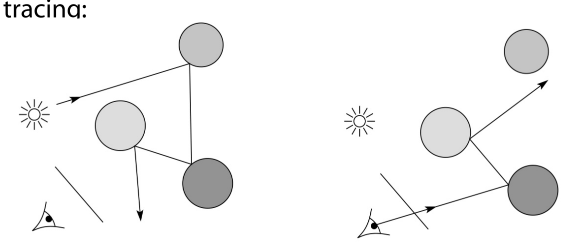
Shadow attenuation (cont'd)

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

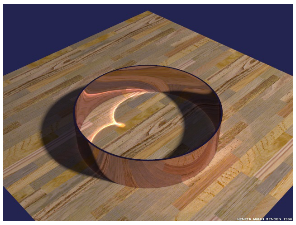


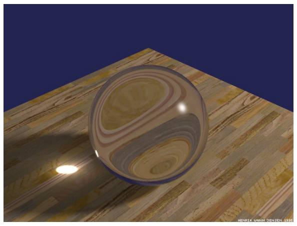
Photon mapping

Combine light ray tracing (photon tracing) and eye ray tracing:



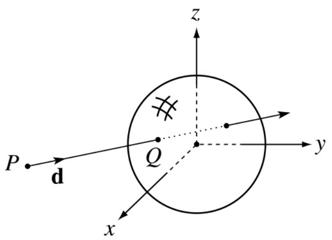
…to get **photon mapping**.





Renderings by Henrik Wann Jensen: http://graphics.ucsd.edu/~henrik/images/caustics.html

Intersecting rays with spheres



Given:

- The coordinates of a point along a ray passing through *P* in the direction **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*.

Intersecting rays with spheres

Solution by substitution:

$$x^2 + y^2 + z^2 - 1 = 0$$
$$(P_x + td_x)^2 + (P_y + td_y)^2 + (P_z + td_z)^2 - 1 = 0$$
$$at^2 + bt + c = 0$$

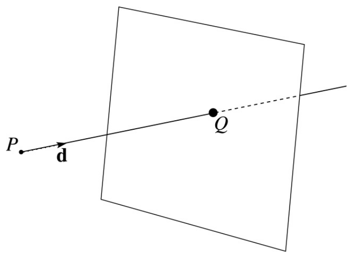
where

$$a = d_x^2 + d_y^2 + d_z^2$$
$$b = 2(P_x d_x + P_y d_y + P_z d_z)$$
$$c = P_x^2 + P_y^2 + P_z^2 - 1$$

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?

Ray-plane intersection



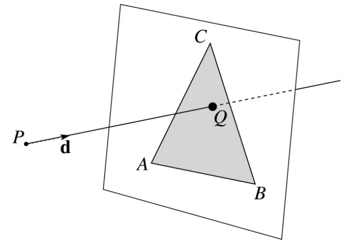
We can write the equation of a plane as:

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

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

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

Ray-triangle intersection



To intersect with a triangle, we first solve for the equation of its supporting plane.

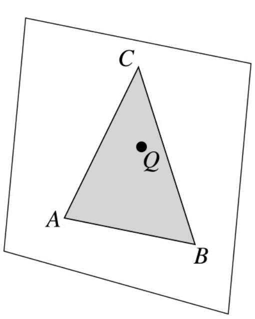
How might we compute the (un-normalized) normal?

Given this normal, how would we compute *d*?

Using these coefficients, we can solve for *Q*. Now, we need to decide if *Q* is inside or outside of the triangle.

3D inside-outside test

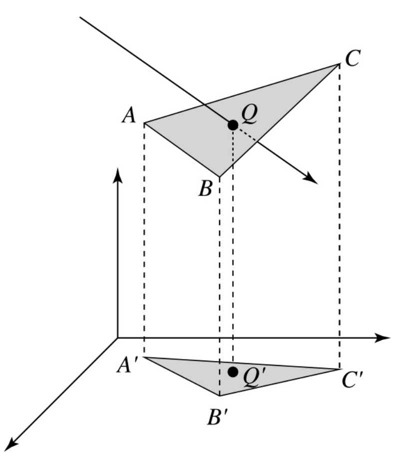
One way to do this "inside-outside test," is to see if *Q* lies on the left side of each edge as we move counterclockwise around the triangle.



How might we use cross products to do this?

2D inside-outside test

Without loss of generality, we can perform this same test after projecting down a dimension:



If *Q'* is inside of *A'B'C'*, then *Q* is inside of *ABC*.

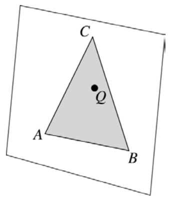
Why is this projection desirable?

Which axis should you "project away"?

Barycentric coordinates

As we'll see in a moment, it is often useful to represent Q as an **affine combination** of A, B, and C:

$$Q = \alpha A + \beta B + \gamma C$$



where:

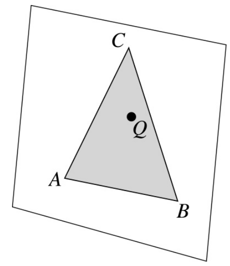
$$\alpha + \beta + \gamma = 1$$

We call α, β, and γ, the **barycentric coordinates** of Q with respect to A, B, and C.

Barycentric coordinates

Given a point Q that is inside of triangle ABC, we can solve for Q's barycentric coordinates in a simple way:

$$\alpha = \frac{\text{Area}(QBC)}{\text{Area}(ABC)} \quad \beta = \frac{\text{Area}(AQC)}{\text{Area}(ABC)} \quad \gamma = \frac{\text{Area}(ABQ)}{\text{Area}(ABC)}$$



How can cross products help here?

In the end, these calculations can be performed in the 2D projection as well!

Interpolating vertex properties

The barycentric coordinates can also be used to interpolate vertex properties such as:

- material properties
- texture coordinates
- normals

For example:

$$k_d(Q) = \alpha k_d(A) + \beta k_d(B) + \gamma k_d(C)$$

Interpolating normals, known as Phong interpolation, gives triangle meshes a smooth shading appearance. (Note: don't forget to normalize interpolated normals.)

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?

Summary

What to take home from this lecture:

- The meanings of all the boldfaced terms.
- Enough to implement basic recursive ray tracing.
- How reflection and transmission directions are computed.
- How ray--object intersection tests are performed on spheres, planes, and triangles
- How barycentric coordinates within triangles are computed
- How ray epsilons are used.