Ray tracing

- simple ray casting
- recursive ray tracing
- cheap tricks
- optimizations

Global effects

- shadows
- specular reflection
- transmission

Color: Ray tracing

For each channel, we'll approximate the color at the intersection point as the sum of five terms:
- emission
- ambient reflection
- diffuse reflection (check for shadows)
- specular reflection (check for shadows)
- recursive term (indirect reflection)
indirect reflection of light

specular transmission

cast ray

specular reflection

shadow

do we see reflected in the surface?

does light hit the surface?

transmission

what do we see through the surface?

recursive rays

what do we see reflected in the surface?

do light hit the surface?

transmission

what do we see through the surface?

shadow

do we see reflected in the surface?

do light hit the surface?

transmission

what do we see through the surface?

occlusion (shadows)

normal in direction of viewer

intersection point P

light L is occluded if the ray R'=(P,-ld)
intersects some object in the scene

recursive ray implementation

offset R'

slightly so it doesn't intersect at P

offset EPSILON n:

n is the normal at the point of intersection toward the incoming ray

intersection point P

specular reflection

what do we see reflected in the surface?

specular reflection

do light hit the surface?

transmission

what do we see through the surface?
specular reflections

- cast ray reflected at P into scene
- find closest intersection point P’ (if any)
- compute color C at P’
- scale by msc(P) (c=r,g,b) and add to color at P

recursive rays

what do we see reflected in the surface?

what do we see through the surface?

does light hit the surface?

transmission

- cast ray transmitted at P into scene
- find closest intersection point P’ (if any)
- compute color at P’
- scale by ktrans(P) and add to color at P

refraction - Snell’s law

- incoming ray (P0,v)
- transmitted ray (P,v’)

Snell’s law

\[ \eta_{\text{out}} \sin \theta_{\text{out}} = \eta_{\text{in}} \sin \theta_{\text{in}} \]

\[ \sin \theta_{\text{out}} = \frac{\beta \sin \theta_{\text{in}}}{\eta_{\text{in}}} \]

provided \(0 \leq \beta \sin \theta_{\text{in}} \leq 1\)

What if \(\beta \sin \theta_{\text{in}} < 1\) ?

\(\theta_{\text{out}} > 90^\circ\) : no transmission
**Snell's Law**

Snell's law describes the relationship between the angles of incidence and refraction when a light ray passes from one medium to another.

\[ n_1 \sin \theta_1 = n_2 \sin \theta_2 \]

- \( n_1 \) and \( n_2 \) are the refractive indices of the two media.
- \( \theta_1 \) and \( \theta_2 \) are the angles of incidence and refraction, respectively.

**In vs. Out**

When a ray passes through a medium, its trajectory changes according to Snell's law.

\[ \frac{v_{in}}{v_{out}} = \frac{n_{out}}{n_{in}} \]

- \( v_{in} \) and \( v_{out} \) are the velocities of the ray in the initial and final media, respectively.
- \( n_{in} \) and \( n_{out} \) are the refractive indices of the initial and final media, respectively.

\[ s = v_{in} - n \cdot v_{in} \]

- \( s \) is the change in the direction of the ray.

**Thin Transparent Surfaces**

For very thin surfaces, we can approximate the behavior using Snell's law.

\[ \beta = \frac{1}{n_{in}} \]

- \( \beta \) is the angle of refraction.

**Building Thick Objects**

When building thick objects, we need to consider how the ray interacts with the object.

**Oriented Triangles**

In oriented triangles, the order of vertices is important:

From the front, vertices are ordered counter-clockwise.

\[ \tau_2 \]

- \( \tau_1 \) and \( \tau_2 \) are the vertices of the triangle.

- \( \tau_3 \) is the vertex that is in front when viewed from the front.

You need to know whether you are entering or leaving the object!
building thick objects

fronts of triangles face toward the outside of object

how to keep all of this straight

when you find an intersection point:
1. choose the normal directed toward the start of the ray
2. set the entering variable to true or false

recursive stopping conditions

recurse until:

- cast new ray from P into scene
- find closest intersection point P'(if any)
- compute color at P'
- scale and add to color at P

recursive stopping conditions

recurse until:

- cast new ray R' from P into scene
- find closest intersection point P'(if any)
- compute color at P'
- getColor(R', rDepth-1, cVal*s)
- scale (by s) and add to color at P

implementation issues

- offset new ray slightly to make sure you don't find P again!!!