cs155 – z sweedyk

ray tracing

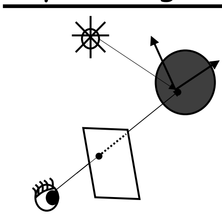
ray tracing

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

global effects

- shadows
- specular reflection
- transmission

ray tracing



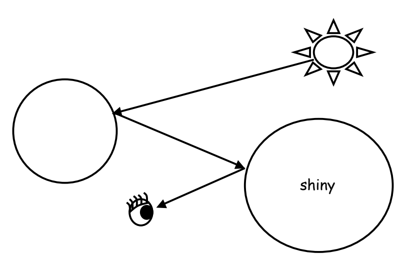
- cast ray through pixel into scene
- find closest intersection (if any)
- compute luminance at intersection
  - direct illumination (**no occlusions**)
  - **indirect reflection**

color: ray tracing

for each channel we'll approximate the color at the intersection point as the sum of five terms

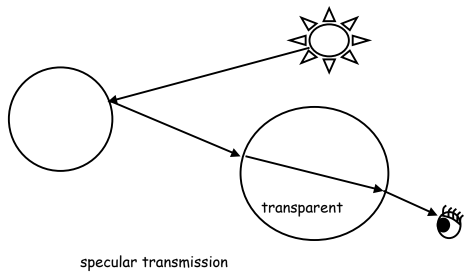
- emission
- ambient reflection
- diffuse reflection **(check for shadows)**
- specular reflection **(check for shadows)**
- **recursive term (indirect reflection)**

indirect reflection of light

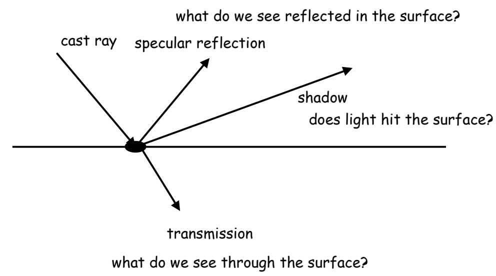


specular (mirror-like) reflection

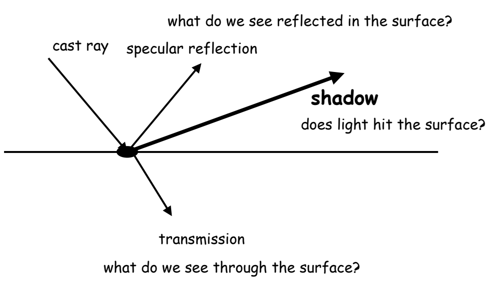
indirect reflection of light



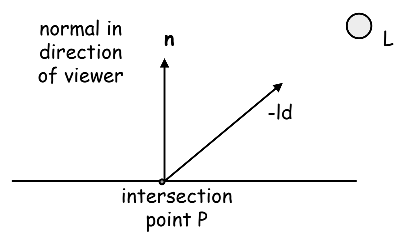
recursive rays



shadow



occlusion (shadows)

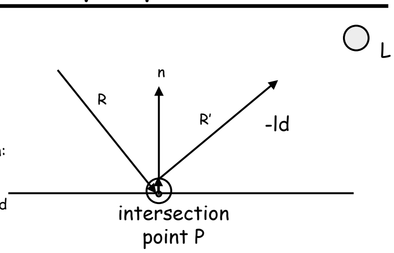


light L is occluded if the ray R'=(P,-ld) intersects some object in the scene

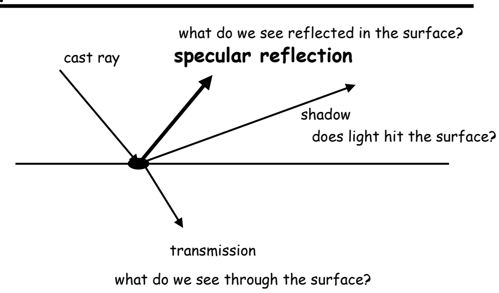
recursive ray implementation

offset R' slightly so it doesn't intersect at P

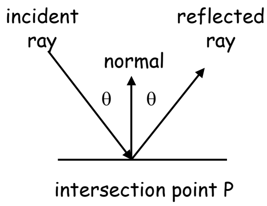
offset EPSILON n: n is the normal at the point of intersection toward the incoming ray



specular reflection

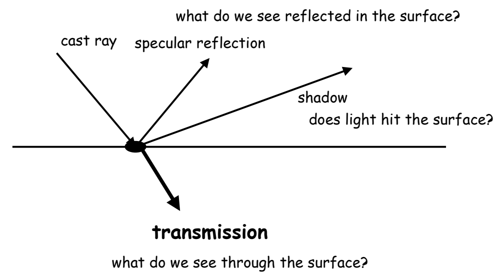


specular reflections

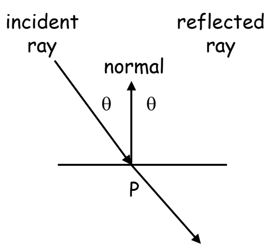


- cast ray reflected at P into scene
- find closest intersection point P' (if any)
- compute color *C* at P'
- scale by msc(P) (c=r,g, & b) and add to color at P

recursive rays

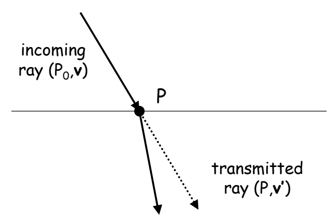


transmission

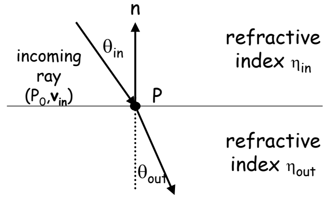


- cast ray transmitted at P into scene
- find closest intersection point P' (if any)
- compute color at P'
- scale by k_{trans}(P) and add to color at P

refraction – snell's law



Snell's law



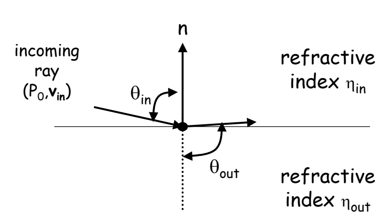
θ_{out} satisfies: $\eta_{out} \sin \theta_{out} = \eta_{in} \sin \theta_{in}$

⬍

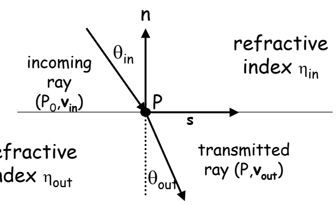
$\sin \theta_{out} = \beta \sin \theta_{in}$ where $\beta = \eta_{in}/\eta_{out}$
provided $0 \le \beta \sin \theta_{in} \le 1$

What if $\beta \sin\theta_{in} < 1$?

$\theta_{out} > 90°$: no transmission



snell's law



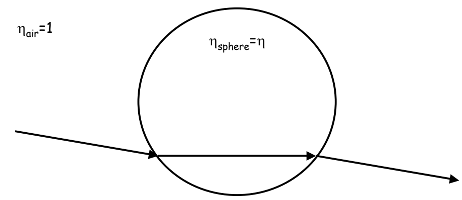
$\boxed{\mathbf{v}_{out} = \cos\theta_{out}\,(-\mathbf{n}) + \sin\theta_{out}\,\mathbf{s}}$

$\sin\theta_{out} = \beta \sin\theta_{in}$

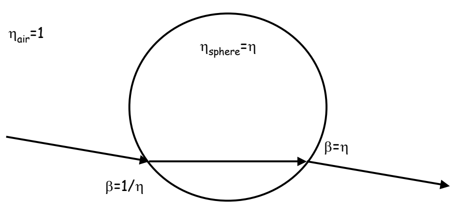
$\cos\theta_{out} = (1-\sin^2\theta_{out})^{1/2} = (1-\beta^2\sin^2\theta_{in})^{1/2}$

$\mathbf{s} = \mathbf{v}_{in} - (-\mathbf{n}\bullet\mathbf{v}_{in})\,\mathbf{n} = \mathbf{v}_{in} + \cos(\theta_{in})\,\mathbf{n}$

in vs. out



in vs. out

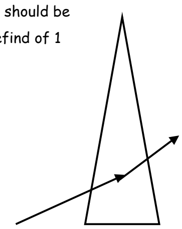


you need to know whether you are entering or leaving object!

thin transparent surfaces

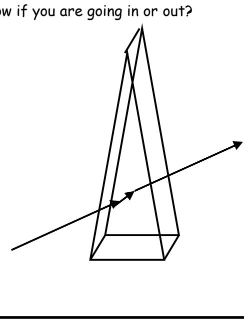
we won't have any!

i.e. any thin surface should be specified with a refind of 1



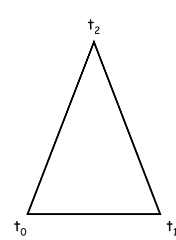
building thick objects

so how do you know if you are going in or out?



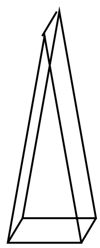
oriented triangles

from the front the vertices are order counter-clockwise



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:

a) maximum recursive depth specified by user is reached

b) contribution to color is less than user specified bound

- 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:

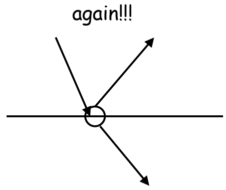
a) maximum recursive depth specified by user is reached

b) contribution to color is less than user specified bound

- 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!!!



- 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