

Reflection and Refraction

Review: Illuminate routine

Given first intersection of ray with object

p - point

obj - object it intersects including material properties

Color

reflectivity

n - normal vector of object at that point

Calculate the color of that point

Color = **illuminate**(p,obj,n)

Color = illuminate(p,obj,n)

Need:

- Position in space that is to be illuminated
 - To form vectors to light, camera, etc.
- Normal vector
 - To form reflection vector, compute angle to surface
- Access to object's material properties
 - Color, Reflection coefficients, specular power
- Access to scene data including
 - Lights: position, type, color, etc
 - Camera: position
 - Other objects (shadows, reflections, etc.)

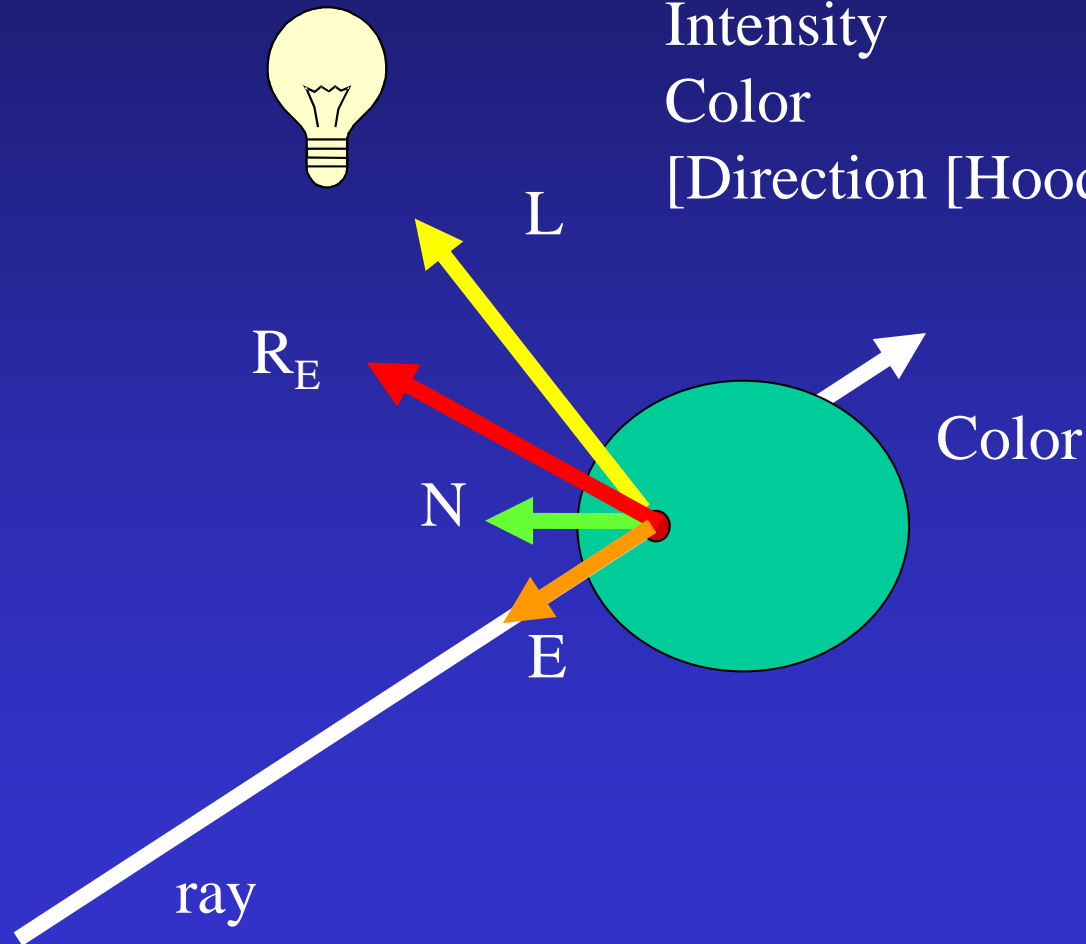
Review: Illumination routine

Directional/Point/Warn

Intensity

Color

[Direction [Hood Angle]]



Ambient: a

Diffuse: $N \cdot L$

Specular: $(L \cdot R_E)^f$

Introduce SHADE routine

to prepare for recursive organization

For each pixel

 Compute ray, R , from eye through pixel

$C = \text{shade}(R)$

 Pixel.color = C

```
Color shade(R)
```

```
{
```

```
  intersect objects – get closest intersection
```

```
  c = ambient
```

```
  For each light source
```

```
    Add in diffuse and specular components to c
```

```
  Return c
```

```
}
```

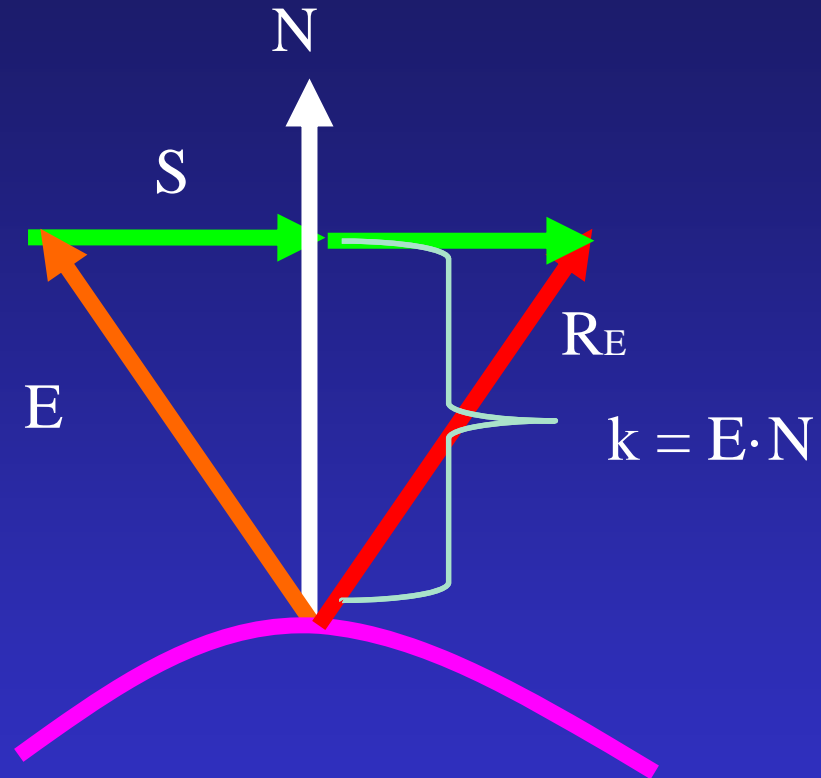
Reflective Ray Tracing

It include reflection effects, in shade routine:

- Compute color of intersection point, just like before
- If object is shiny, spawn a reflective ray and call **shade** on that ray
- The color returned by reflective ray is attenuated by object's shininess and added to point color
- Limit number of recursive calls by including count and don't spawn ray if maximum is exceeded.

Reflective Ray

$$S = kN - E$$

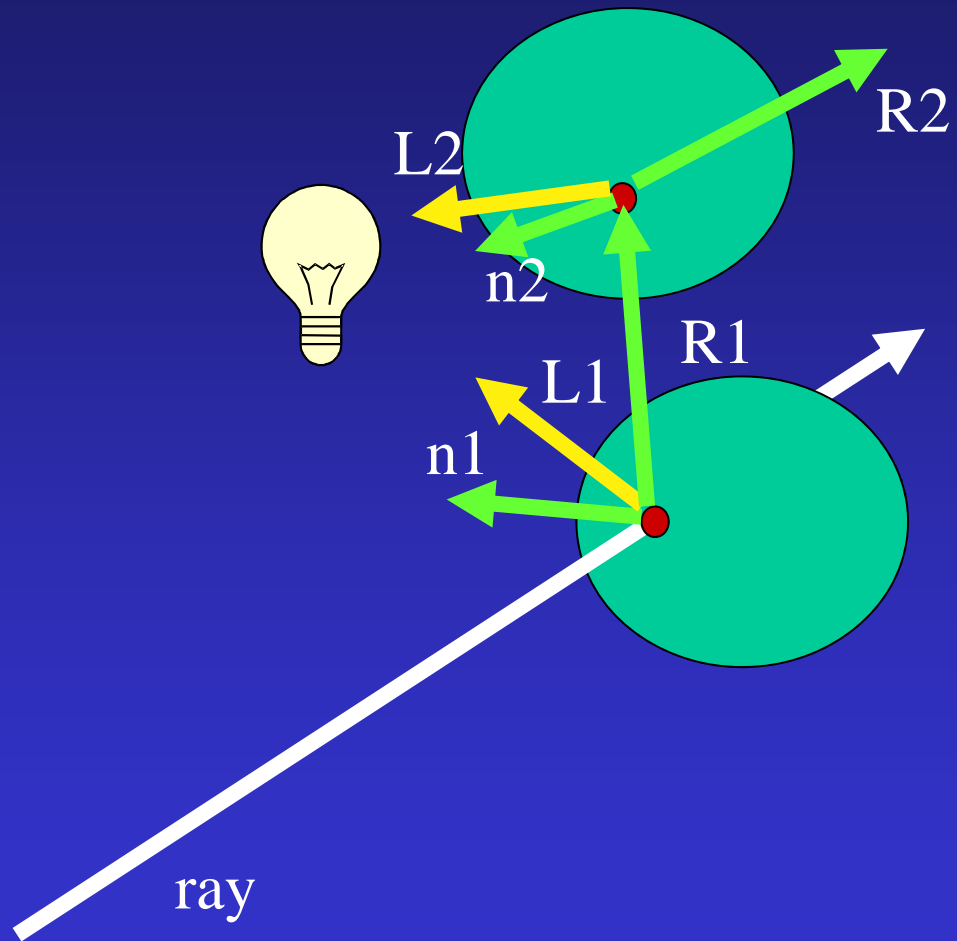


$$R_E = E + S + S = e + 2(kN - E) = 2(E \cdot N)N - E$$

Reflective Ray Tracing

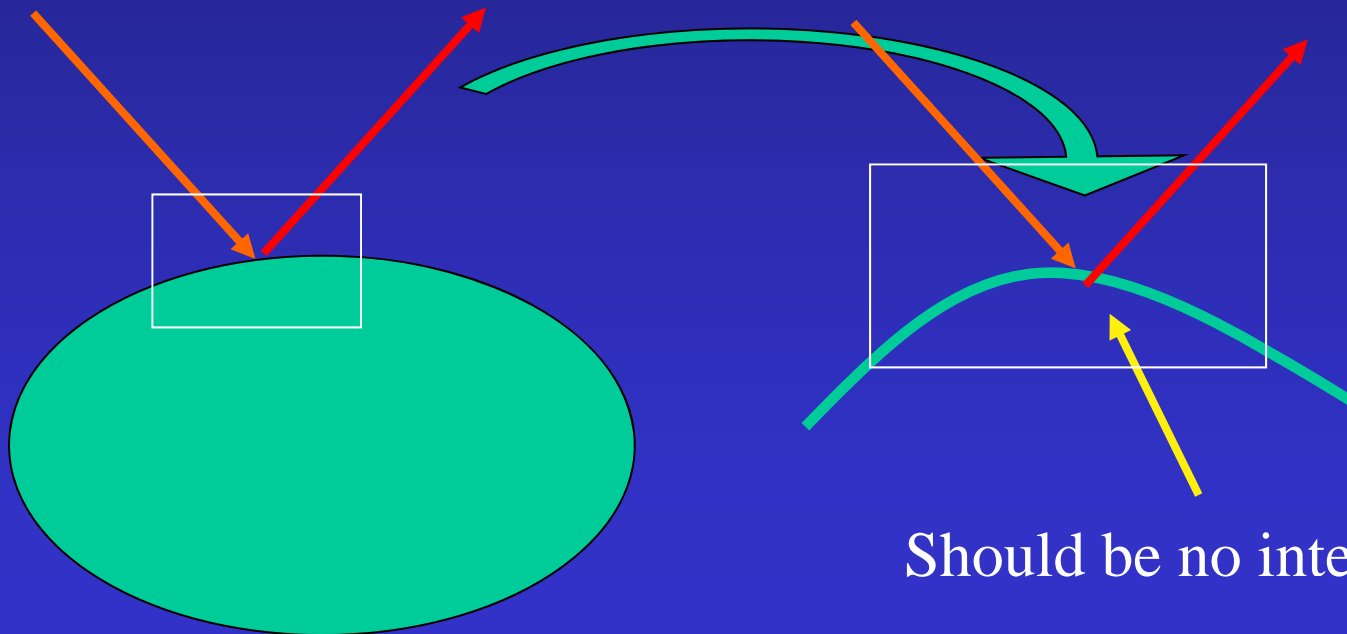
```
Color shade(ray,recursionDepth)
{
  intersect objects to get point, normal, object
  If (intersection) {
    c = ambient color
    Compute reflective ray, R, based on ray and normal
    For each light source
      Compute and add in diffuse and specular components to c
      If ( (recursionDepth < maxRecursion) && (object is shiny) )
        c += object.shininess * shade(R,recursionDepth+1)
  }
  Else c = backgroundColor;
  Return c
}
```


Reflective Ray Tracing



WARNING !

Be careful not to have spawned reflection ray immediately intersect the object it is leaving!



Refractive Ray Tracing

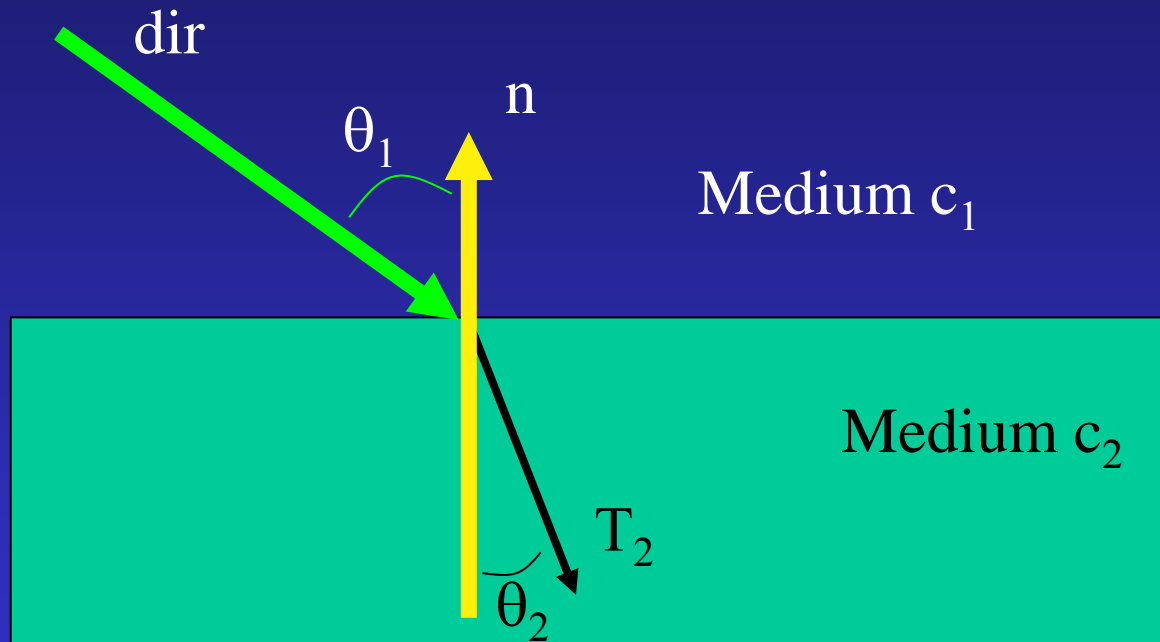
Same idea as Reflective ray tracing – spawn a secondary ray

The direction of the spawned ray is in the direction of the incoming ray altered a little bit

The alteration is based on the material that the ray is leaving (e.g. air) and the material that the ray is entering (e.g., glass)

Once this direction is computed, then the shade routine is called recursively, just as with reflective rays.

Refractive Ray Tracing



Snell's Law

$$n_1 \sin(\theta_1) = n_2 \sin(\theta_2)$$

Refractive Ray Direction

$$\cos(\theta_1) = -\text{dir} \cdot \mathbf{n}$$

$$\sin(\theta_1) = \sqrt{1 - \cos^2(\theta_1)}$$

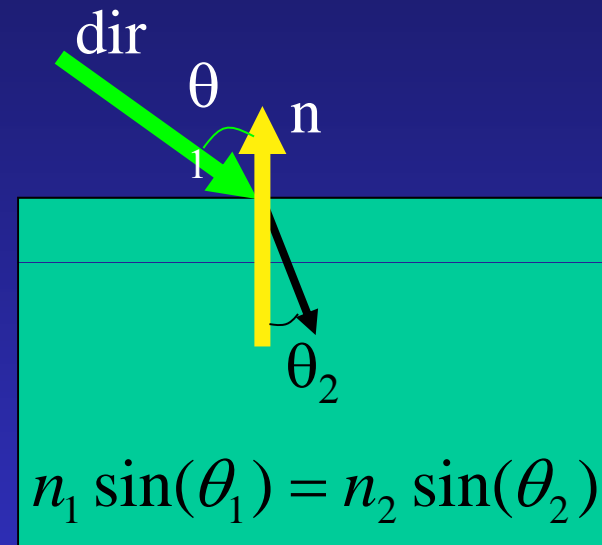
$$\sin(\theta_2) = n_1 \cdot \sin(\theta_1) / n_2$$

$$\cos(\theta_2) = \sqrt{1 - \sin^2(\theta_2)}$$

$$\cos(\theta_2) = \sqrt{1 - (n_1 \cdot \sin(\theta_1) / n_2)^2}$$

$$\cos(\theta_2) = \sqrt{1 - (n_1 / n_2)^2 \cdot (1 - \cos^2(\theta_1))}$$

$$\cos(\theta_2) = \sqrt{1 - (n_1 / n_2)^2 \cdot (1 - (\text{dir} \cdot \mathbf{n})^2)}$$



NOTE: if radical is negative, no refraction!

Refractive Ray Direction

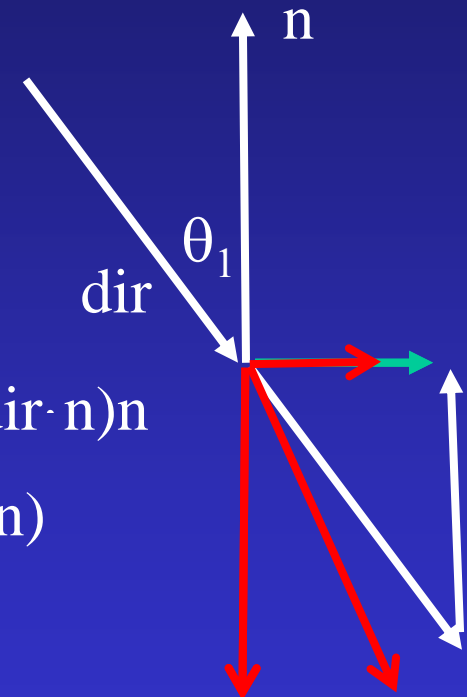
Go $\cos(\theta_2)$ in the $-n$ direction

Go $\sin(\theta_2)$ in the orthogonal direction
(while still in the plane of dir and n)

This direction (scaled by $\sin(\theta_1)$) is: $dir - (dir \cdot n)n$

And scaled by $\sin(\theta_2)$ is: $(n_1/n_2)(dir - (dir \cdot n)n)$

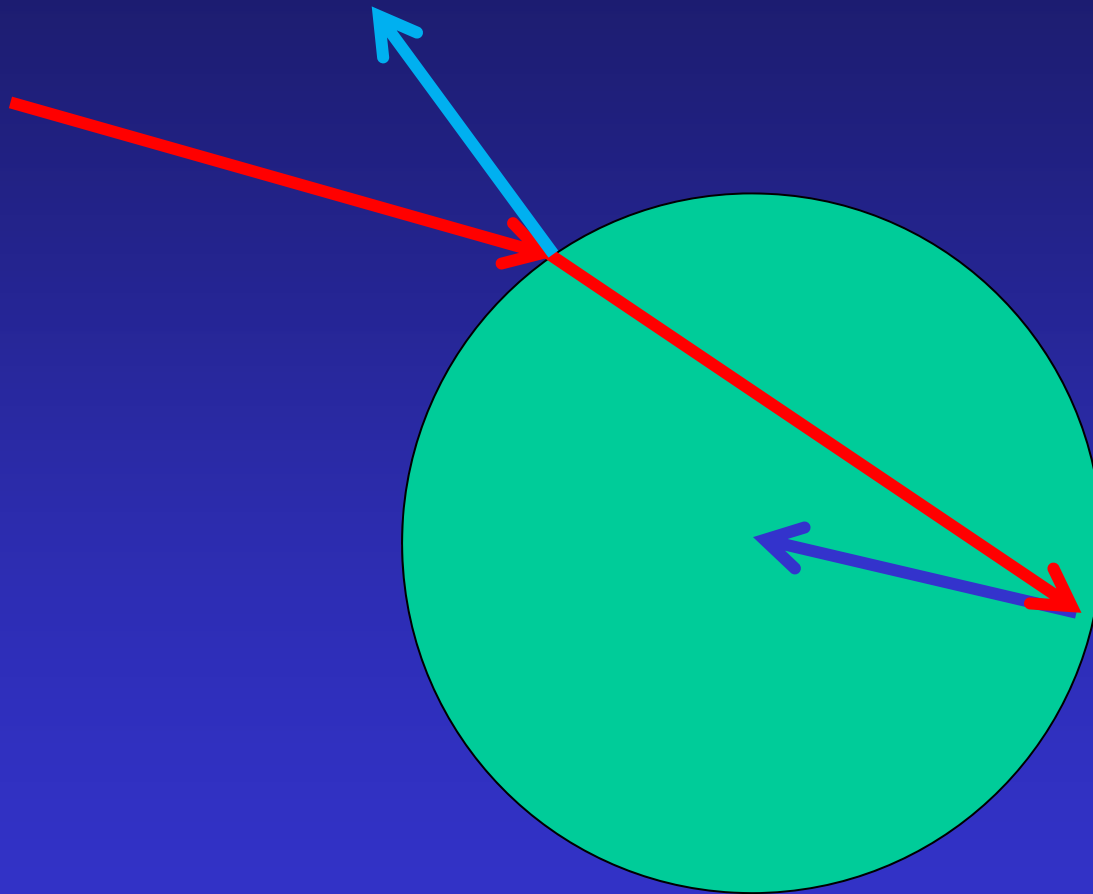
$$T = (n_1/n_2)dir - (n_1/n_2)(dir \cdot n)n - \cos(\theta_2)n$$



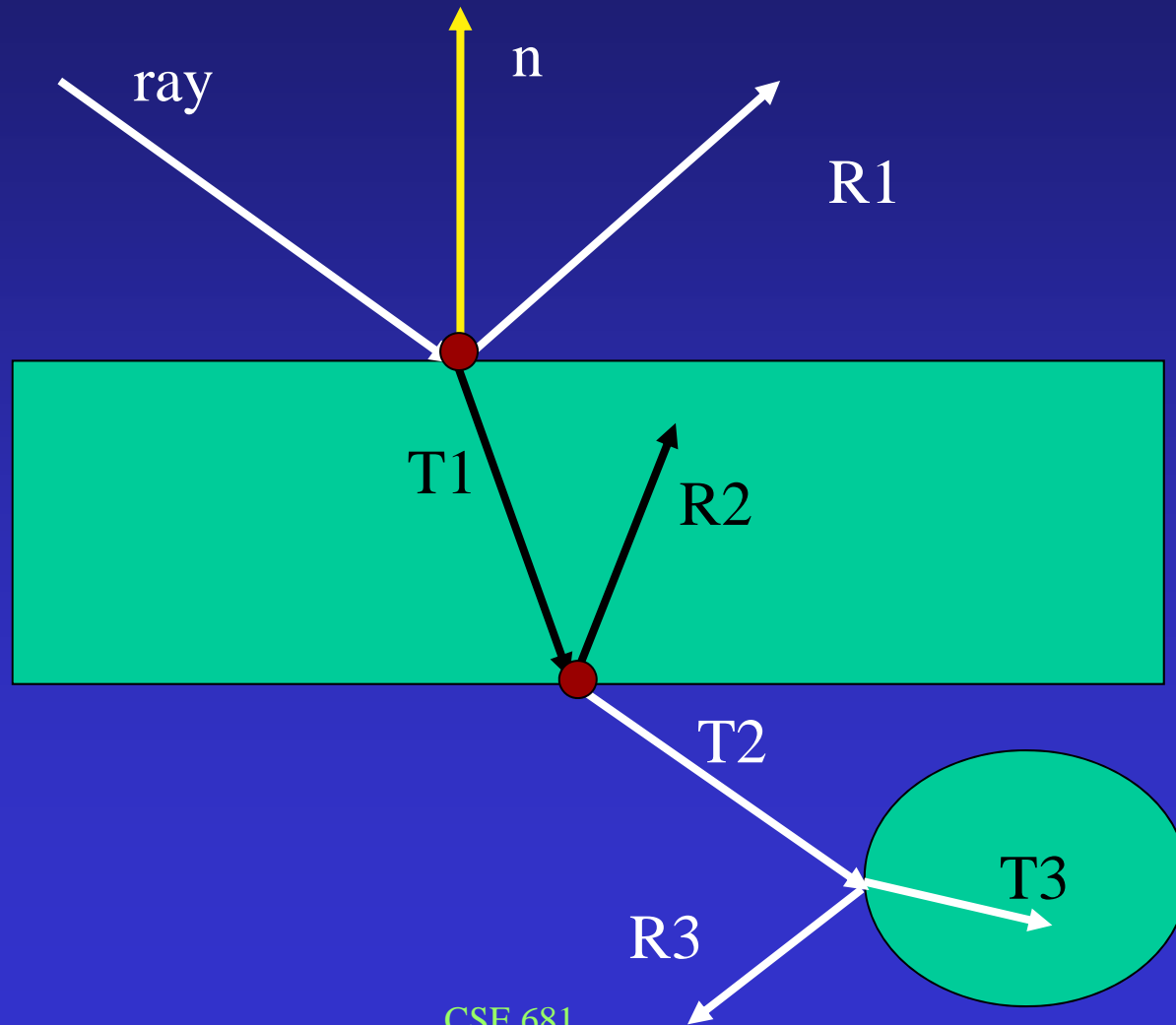
Recursive Ray Tracing

```
Color shade(ray,recursionDepth)
{
  intersect objects...
  compute R ...
  Process each light source ...
  If (recursionDepth < maxRecursion) {
    If (object is shiny) c += shininess * shade(R, recursionDepth+1)
    If (object is transmittive) {
      Compute refractive ray, T, based on ray, normal, and Snell constants
      c = (1-transmittive)*c + transmittive * shade(T, recursionDepth+1)
    }
  }
  Return c
}
```

Refractive Intersection Normal



Recursive Ray Tracing



Reflectivity varies with incident angle

NOTE: refraction is really **wavelength dependent** (where rainbows come from).

Fresnel equations

Schlick approximation:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

R_0 is reflectance at normal incidence:

$$R_0 = 0.8$$

From book:

Assume one of the n_i
is always 1 (air);
Call the other one n_2

Also uses Beer's Law
to attenuate light
passing through
material (p. 214):
 $I(s) = I(0)e^{-a}$

```
IF (p is on a dielectric) THEN
r = reflect(d,n)
IF (d.n<0) THEN {
    refract(d,n,n2,t)
    c=-d.n
    kr=kg=kb=1
}
Else {
    kr=exp(-art)
    kg=exp(-agt)
    kb=exp(-abt)
    If refract(d,-n,1/n,t) then c=t.n
    Else return k*color(p+tr)
}
R0=(n2-1)2/(n2+1)2
R= R0+(1- R0)(1-c)5
Return k(Rcolor(p+tr)+(1-R)color(p+tt))
```