Supplement to Lecture 16

Global Illumination: View Dependent

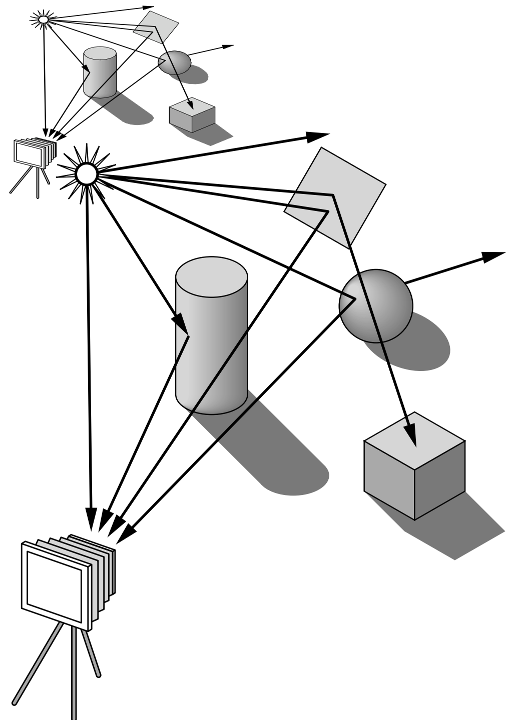
Notes and figures from ***Ed Angel: Interactive Computer Graphics, 6th Ed., 2012 © Addison Wesley***

Local vs Global Illumination

- OpenGL is based on a pipeline model in which primitives are rendered one at time
    - No shadows (except by tricks or multiple renderings)
    - No multiple reflections
- Global approaches
    - Rendering equation
    - Ray tracing
    - Radiosity

Ray Tracing/Casting-1

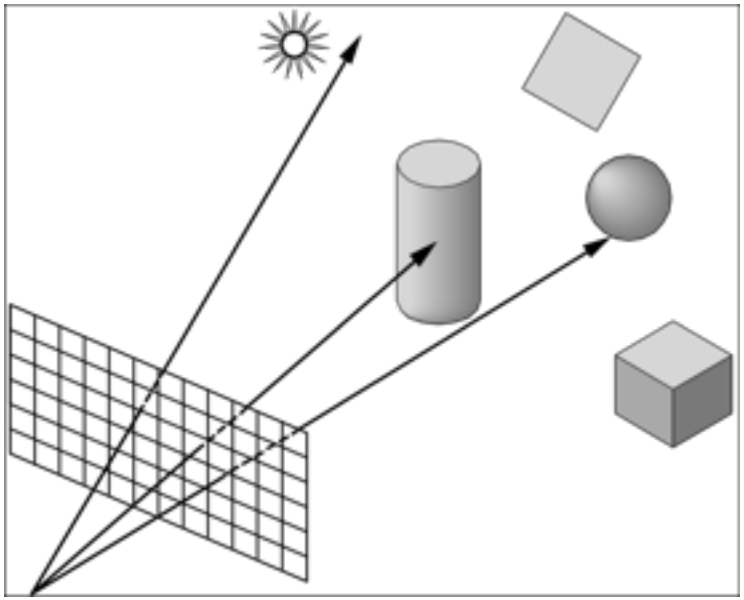
- Follow rays of light from a point source
- Can account for reflection and transmission



- However, scattering produces many (infinite) additional rays

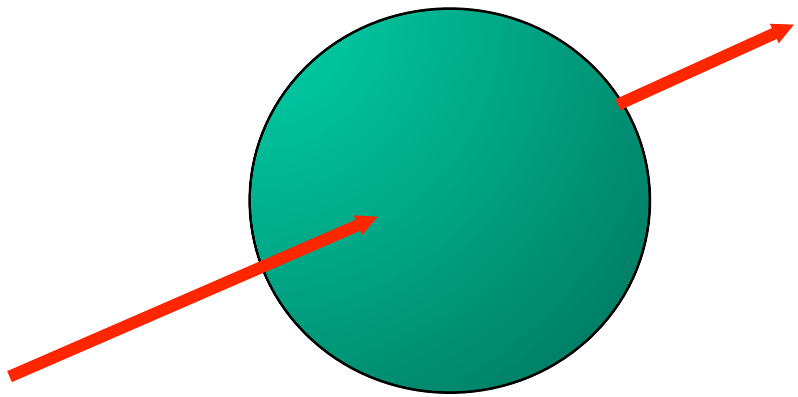
Ray Tracing/Casting-2

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel



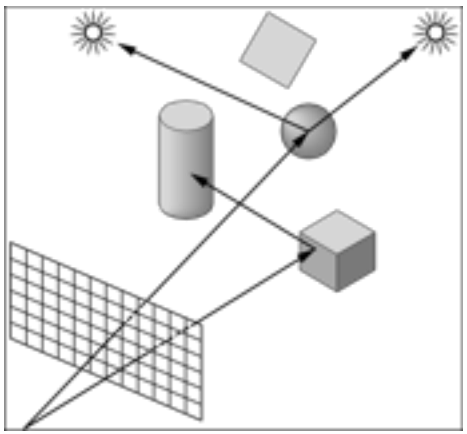
Ray Tracing/Casting a Sphere

- Ray is parametric
- Sphere is quadric
- Resulting equation is a scalar quadratic equation which gives entry and exit points of ray (or no solution if ray misses)



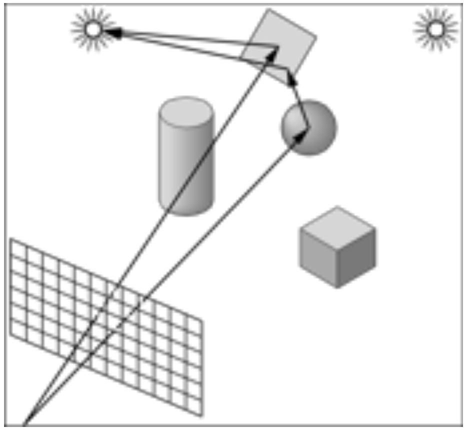
Shadow Rays

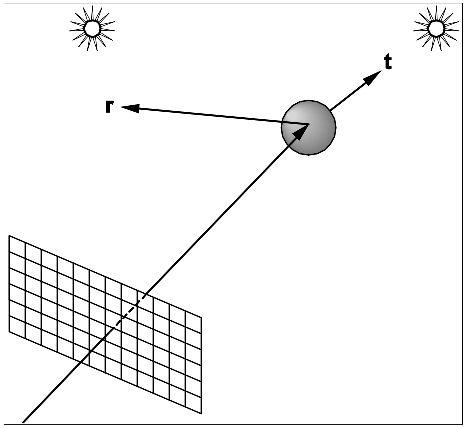
- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow or feeler rays

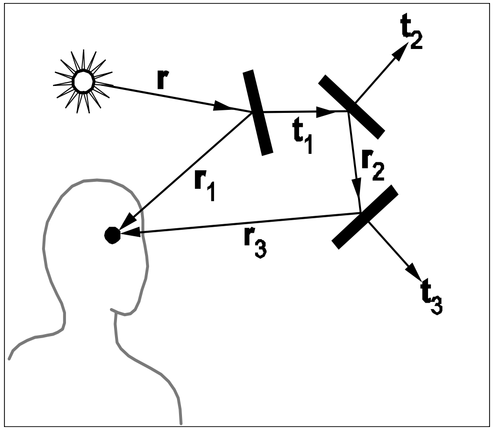


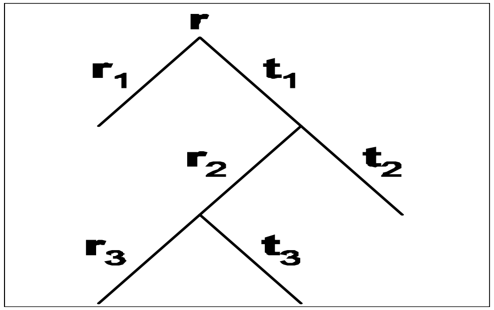
Reflection/Transmission

- Must follow shadow rays off reflecting or transmitting surfaces - Ray Trees
- Ray Tree Traversal - Process is recursive









Diffuse Surfaces

- Theoretically the scattering at each point of intersection generates an infinite number of new rays that should be traced
- In practice, we only trace the transmitted and reflected rays but use the Phong model to compute shade at point of intersection
- Radiosity works best for perfectly diffuse (Lambertian) surfaces

Building a Ray Tracer

- Best expressed recursively
- Can remove recursion later
- Image based approach
  - For each ray …….
- Find intersection with closest surface
  - Need whole object database available
  - Complexity of calculation limits object types
- Compute lighting at surface
- Trace reflected and transmitted rays

When to stop

- Some light will be absorbed at each intersection
  - Track amount left
- Ignore rays that go off to infinity
  - Put large sphere around problem
- Count steps

Recursive Ray Tracer I

```
Program Sketch for Ray Tracing

program raytrace
var lsou; (* intensity of light source *)
    back; (* background intensity *)
    ambi; (* ambient light intensity *)
    depth; (* depth of ray tree consisting of multiple
        reflection/refraction paths *)

    ray = record                    (* ray      x = a + ti
            point: (a, b, c)                y = b + tj
            unit direction: (i, j, k)       z = c + tk  *)
          end;
        r: ray;
```

Recursive Ray Tracer II

```
function intensity (r);
                (* intensity = spec + refr + dull
                   spec = specular reflection component
                   refr = refraction component
                   dull = non-reflecting, non refracting
                   component *)
L: unit vector pointing to light source
V: unit vector pointing from current position to eye
N: unit surface normal
Objects [1...n] (* list of n objects in scene *)
Ka [1...n](* ambient reflectivity factor for each object
Ks [1...n](* specular reflectivity factor for each object
Kr [1...n](* refractivity index for each object *)
Kd [1...n](* diffuse reflectivity factor for each object
S[1...n](* shininess factor for each object *)
(* Additional Comments: For a transparent object, Kd[j]=0
 and Ks[j]+Kr[j]=1    i.e. partly reflecting + partly
 refracting. For an opaque object Kr[j]=0,  Ks[j] and
 Kd[j] can be anything as no simple relation between
 them *)
```

Recursive Ray Tracer III

```
function intensity(r: ray): rgb
    var flec, frac: ray;    spec, refr, dull: rgb;
    begin
      depth := depth +1
      if depth >5 then intensity :=back
      else
       begin (* label 1 *)
        check ray r for intersection with all objects
                                             in scene
        if no intersection
        then if r parallel to L
             then intensity :=lsou
             else intensity :=back
        else
         begin (* label2 *)
        Take closest intersection which is object[j]
        compute normal N at the intersection point
        if Ks[j] >0 (* non-zero specular reflectivity *)
        then begin
        compute reflection ray flec;
```

```
            spec := Ks[j]*(intensity(flec) + (normalize(r).V)^S[j
                end
              else spec:=0;
              if(Kr[j]>0)         (* non-zero refractivity *)
              then begin
                 compute refraction ray frac;
                 refr := Kr[j]*intensity(frac);
                end
              else refr:=0;
              check for shadow;
              if shadow
              then dull:= Ka[j]*ambi
              else dull:= Kd[j]*lsou* N.L +Ka[j]*ambi);
              intensity :=spec +refr +dull;
            end (* label2 *)
          end( *label 1*)
         depth := depth -1
end(* function *)

begin (* raytrace*)
    for each pixel P of projection viewport in raster order
    begin
      r = ray emanating from viewer through P; V = r;
      set intensity(r) to the frame buffer pixel correspondin
        to P
    end
end (*raytrace *)
```

Computing Intersections

- Implicit Objects
  - Quadrics
- Planes
- Polyhedra
- Parametric Surfaces

Implicit Surfaces

Ray from $\mathbf{p}_0$ in direction $\mathbf{d}$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

General implicit surface

$$f(\mathbf{p}) = 0$$

Solve scalar equation

$$f(\mathbf{p}(t)) = 0$$

General case requires numerical methods

Quadrics

General quadric can be written as

$$\mathbf{p}^T \mathbf{A} \mathbf{p} + b^T \mathbf{p} + c = 0$$

Substitute equation of ray

$$\mathbf{p}(t) = \mathbf{p}_0 + t\, \mathbf{d}$$

to get quadratic equation

Sphere

$$(\mathbf{p} - \mathbf{p}_c) \bullet (\mathbf{p} - \mathbf{p}_c) - r^2 = 0$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

$$\mathbf{p}_0 \bullet \mathbf{p}_0\, t^2 + 2\, \mathbf{p}_0 \bullet (\mathbf{d} - \mathbf{p}_0)\, t + (\mathbf{d} - \mathbf{p}_0) \bullet (\mathbf{d} - \mathbf{p}_0)$$
$$- r^2 = 0$$

Planes

$$\mathbf{p} \bullet \mathbf{n} + c = 0$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

$$t = -(\mathbf{p}_0 \bullet \mathbf{n} + c) / \mathbf{d} \bullet \mathbf{n}$$

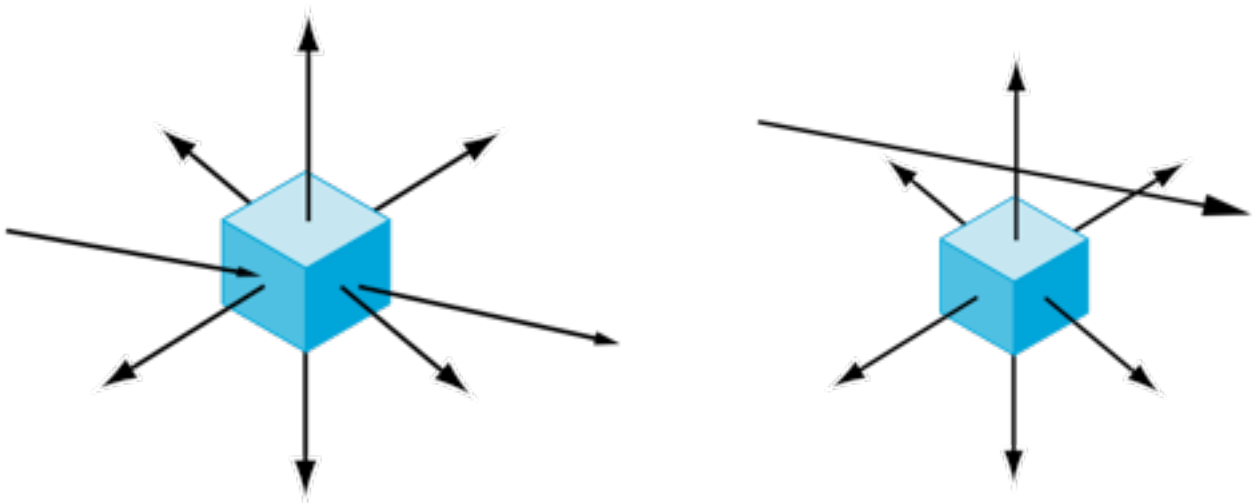
Polyhedra

- Generally we want to intersect with closed objects such as polygons and polyhedra rather than planes
- Hence we have to worry about inside/ outside testing
- For convex objects such as polyhedra there are some fast tests

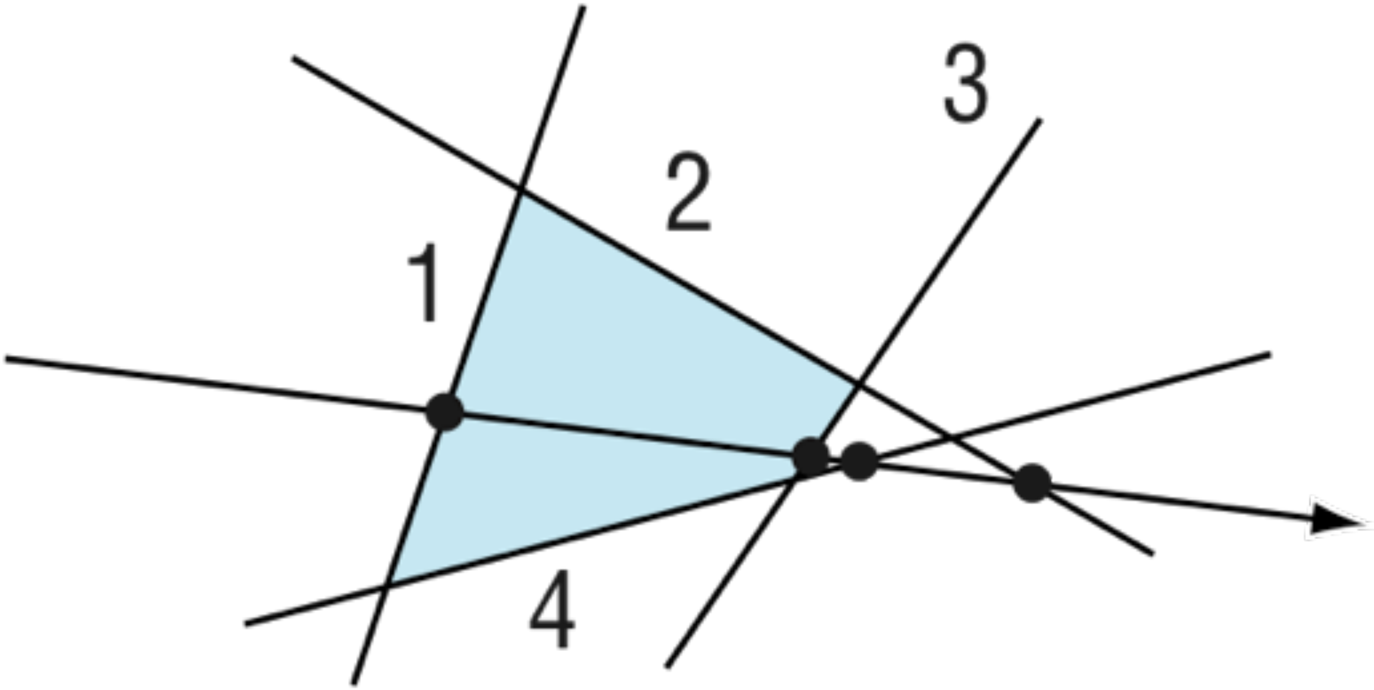
Ray Intersection with Polyhedron

- If ray enters an object, it must enter a front facing polygon and leave a back facing polygon
- Polyhedron is formed by intersection of planes
- Ray enters at furthest intersection with front facing planes
- Ray leaves at closest intersection with back facing planes
- If entry is further away than exit, ray must miss the polyhedron

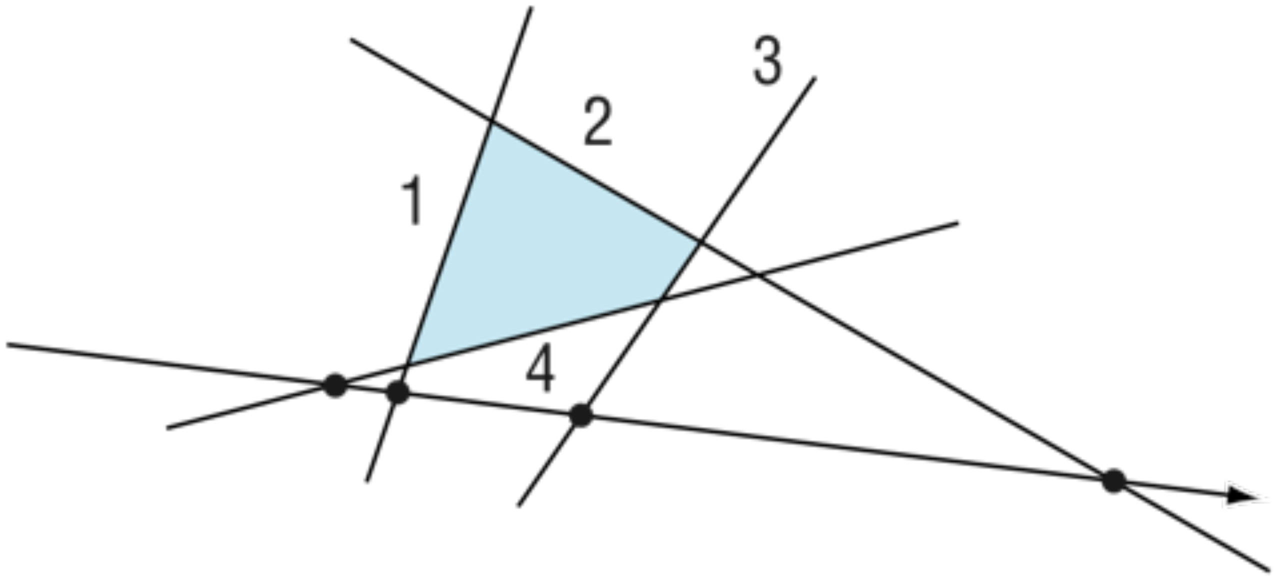
Ray Intersection with a Polyhedron



Ray Intersection with a Polygon

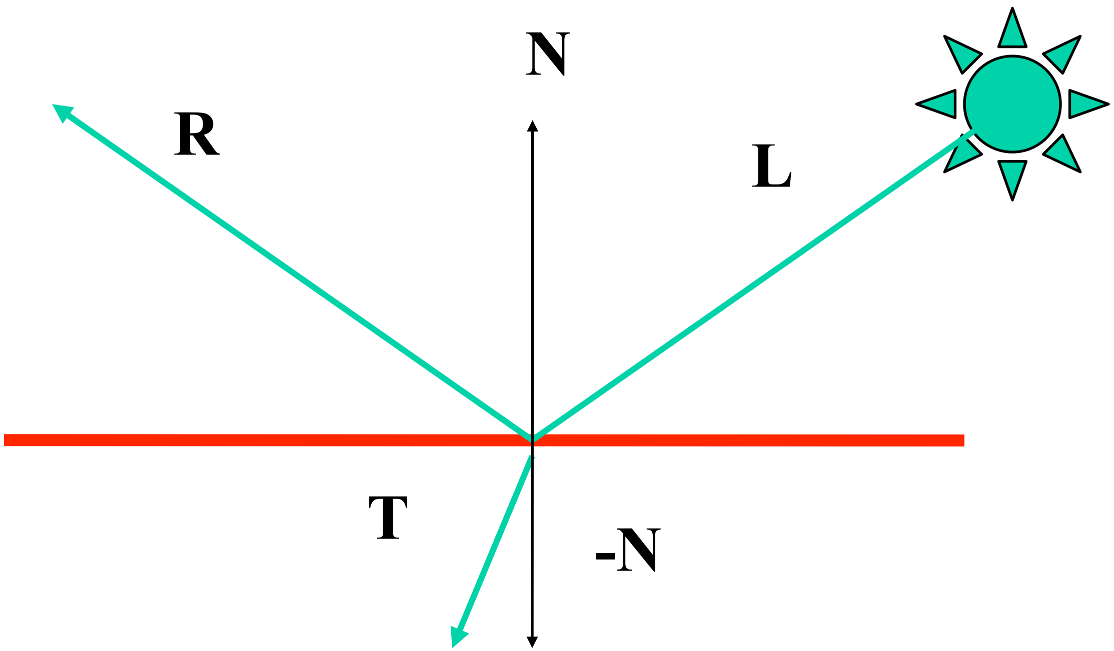


Ray Intersection with a Polygon



Ray Transmission Vectors

- Ray tracers can make use of all these effects in a global calculation by tracing rays



Refraction

With pure refraction, all the light is transmitted but the angle of refraction is determined by Snell's law

$$ή_I \sin θ_I = ή_t \sin θ_t$$

where $ή_I$ and $ή_t$ are the speed of light relative to the speed of light in a vacuum

Let $ή = ή_I / ή_t$

Computing T

$$\acute{\eta}^2 \sin^2\theta_l = \acute{\eta}^2 (1- \cos^2\theta_l) = \sin^2\theta_t = 1-\cos^2\theta_t$$

Solving for $\cos \theta_t$

Assuming normalized vectors

$\cos \theta_t = \mathbf{T}\cdot\mathbf{N} = (1- \acute{\eta}^2 (1-\cos^2\theta_l))^{1/2}$

where $\cos \theta_l = \mathbf{L}\cdot\mathbf{N}$

$\mathbf{T}$, $\mathbf{N}$, and $\mathbf{L}$ must be coplanar

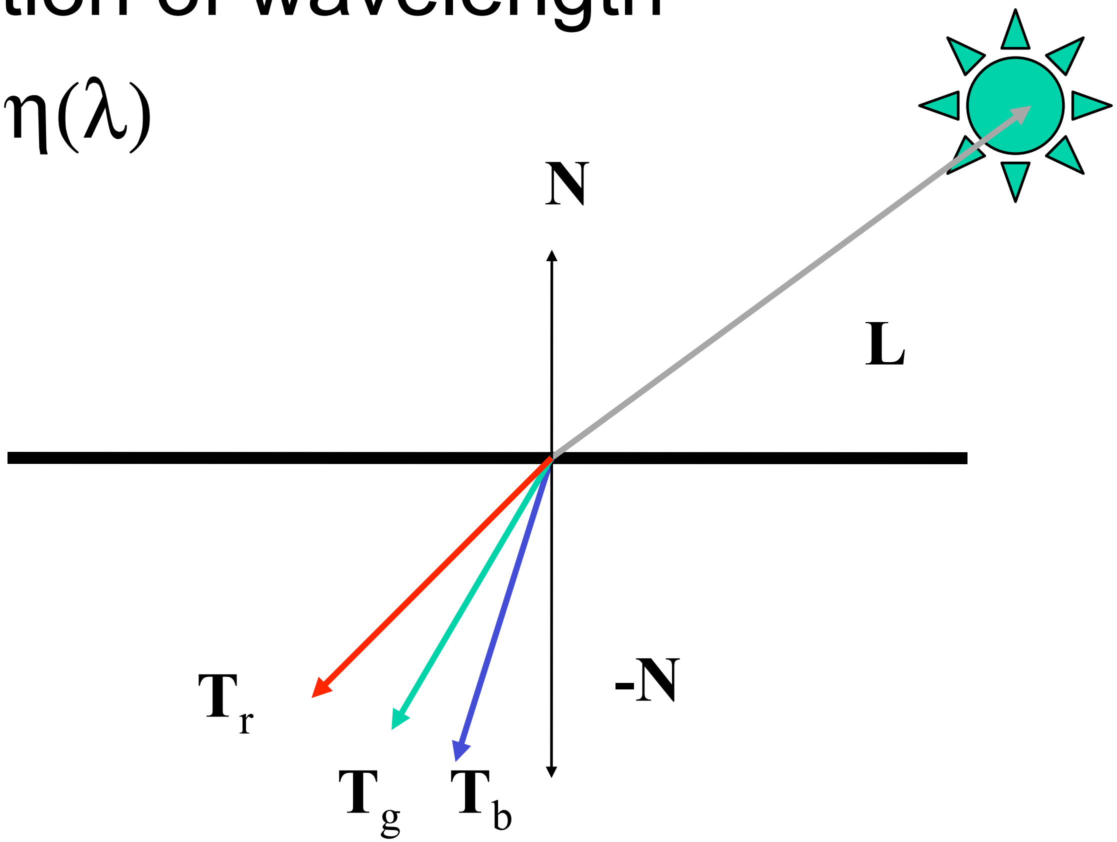
$\mathbf{T} = \alpha \mathbf{L} + \beta \mathbf{N}$ and $\mathbf{T}\cdot\mathbf{T} = 1$

Solving

$\mathbf{T} = -1/\acute{\eta}\, \mathbf{L} - (\cos \theta_t - 1/\acute{\eta} \cos \theta_l)\, \mathbf{N}$

Chromatic Dispersion

- The refraction coefficient is actually a function of wavelength
- $\eta = \eta(\lambda)$



Chromatic Dispersion with Shaders

- Easy to do with reflection maps
- Use three values of η
- Make use of vector operations