Raytracing

COSC 4328/5327

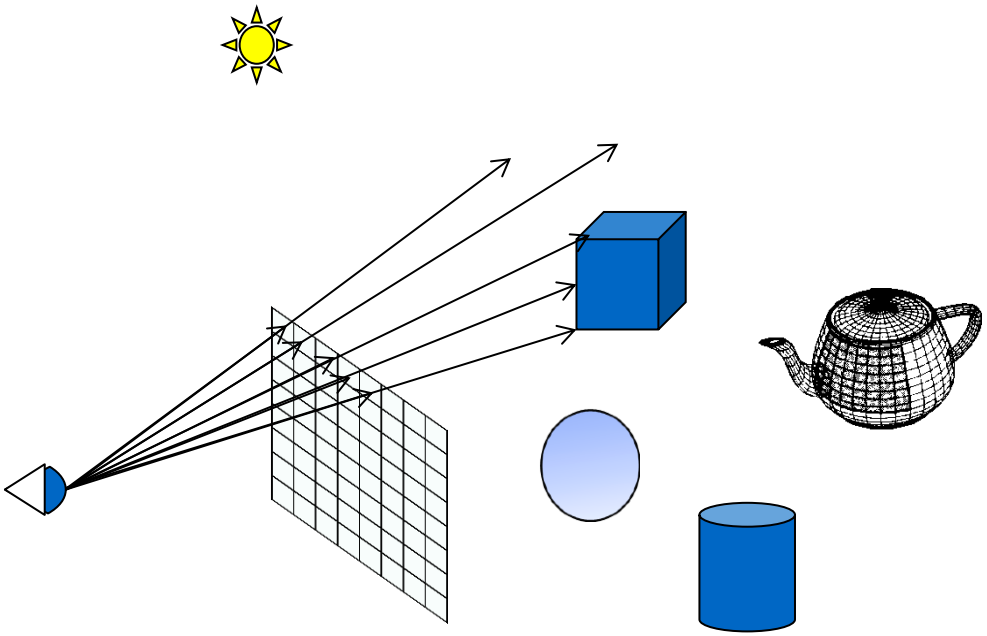
Scott A. King

Basic Ray Casting Method

- ∀ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection

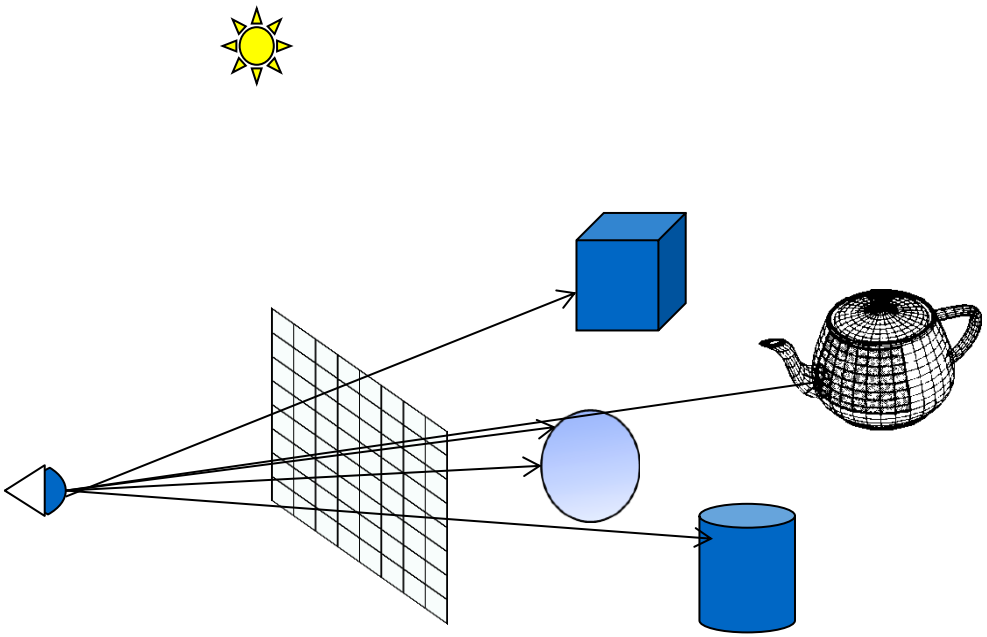
Basic Ray Casting Method

- ∀ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection



Basic Ray Casting Method

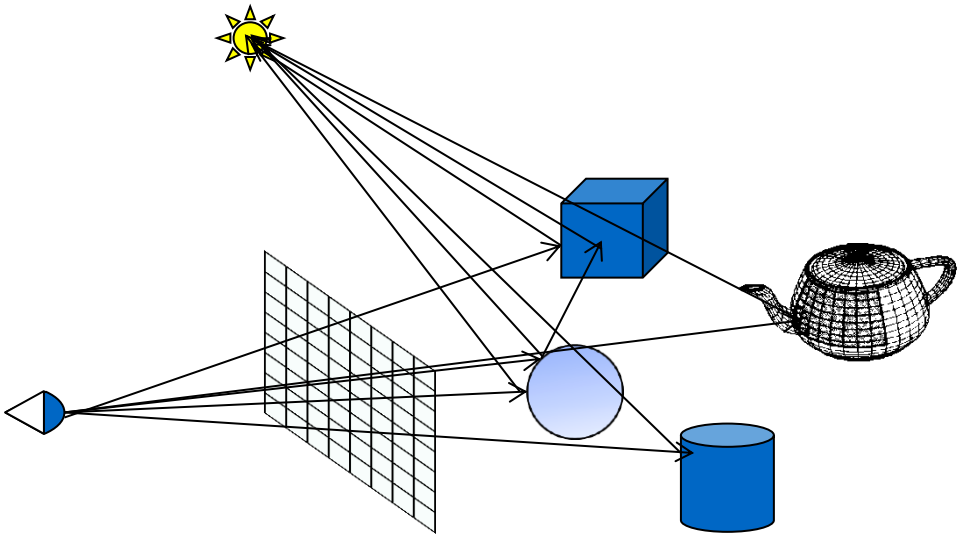
- ∀ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - Get color at intersection



Basic Ray Casting Method

- ∀ pixels in screen
  - Shoot ray $\vec{p}$ from the eye through the pixel.
  - Find closest ray-object intersection.
  - **Get color at intersection**

Illumination Model



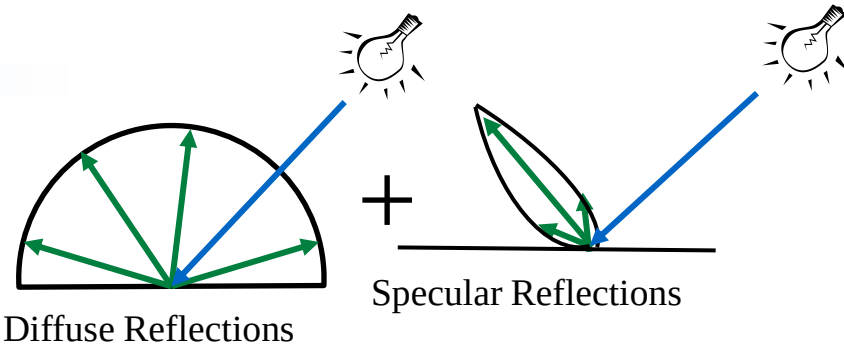
Basic Ray Model

- Let's treat a ray as a vector. Namely we can represent a ray by the vector form:

$$\vec{p} = \vec{u} + \vec{v}t$$

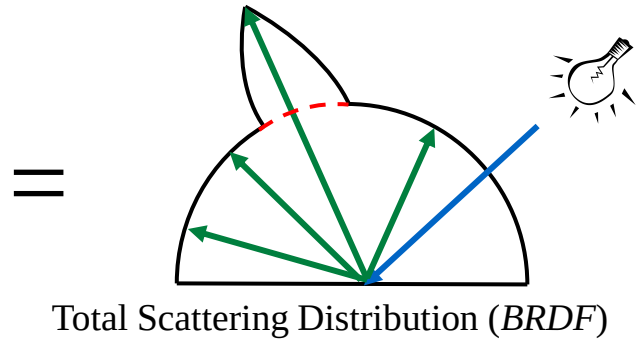
Reflectance: BRDF

- Light arriving at a surface can scatter in many directions
  - Intensity for a given outgoing direction is dependent on incoming direction and material properties
- Model or measurement of reflectance is called the ***bidirectional reflectance distribution function (BRDF)***: for any incoming light ray how much energy is reflected for any outgoing light ray
  - can measure or model



Diffuse Reflections

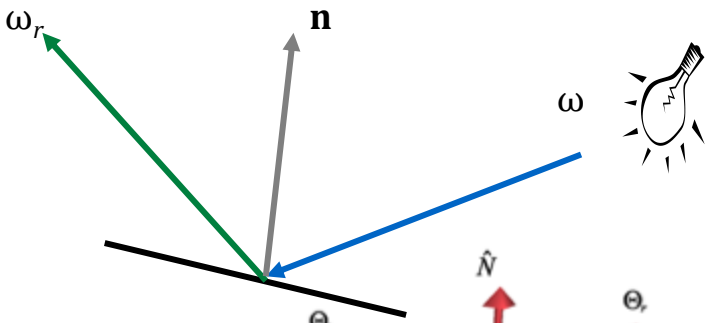
Specular Reflections



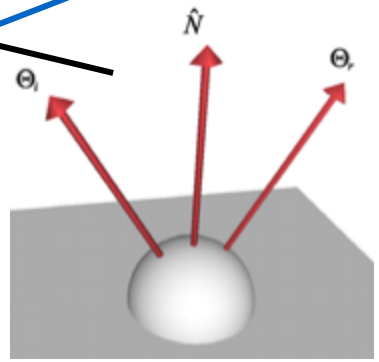
Total Scattering Distribution (*BRDF*)

BRDF for simple model of diffuse + specular reflection, e.g., the Phong Lighting Model

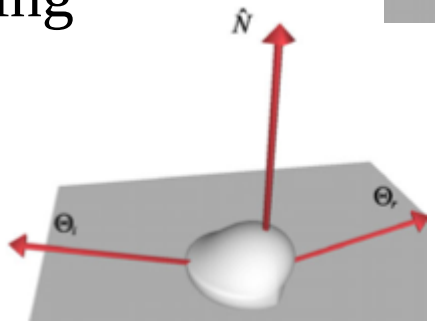
Simple Reflectance Models



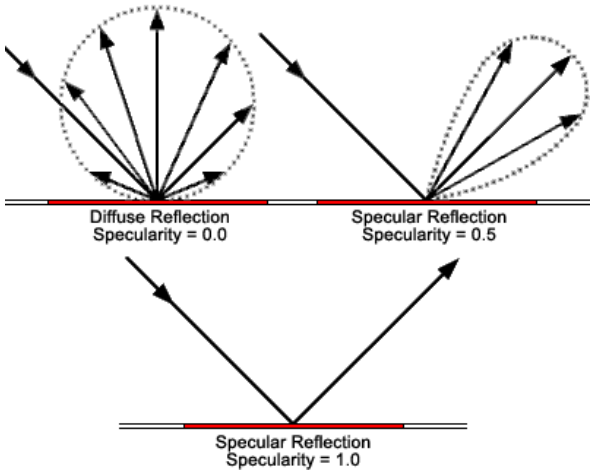
- ***Mirror*** light scatters in a single direction, $\omega_r = \omega - 2(\omega \cdot \mathbf{n})\mathbf{n}$



- ***Lambertian*** light scatters equally (output radiance is equal) in all outgoing directions (i.e. viewer independent)
- ***Diffuse*** light scatters (possibly unevenly) in all outgoing directions, e.g., rug, paper, unvarnished wood

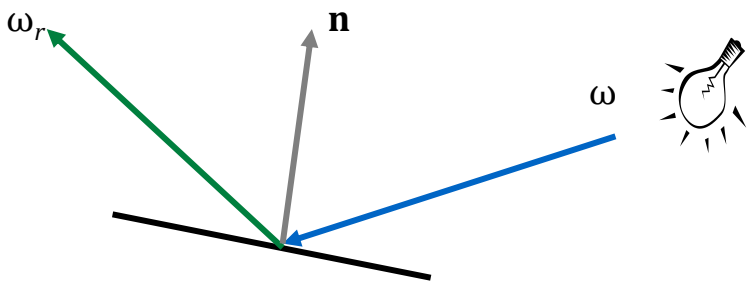


- ***Specular*** light scatters tightly around a particular direction (shiny objects with sharp highlights)
- ***Glossy*** light scatters weakly around a particular direction

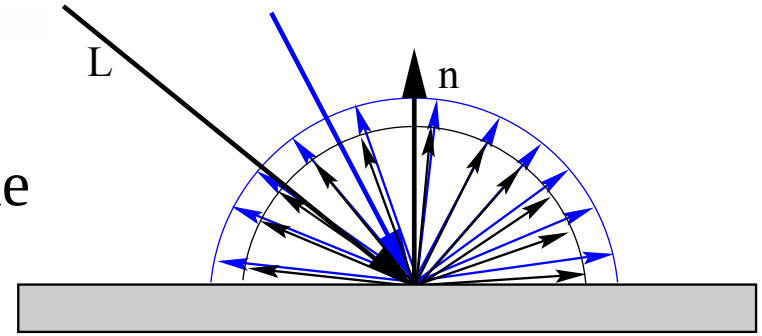


A Perfect Reflector

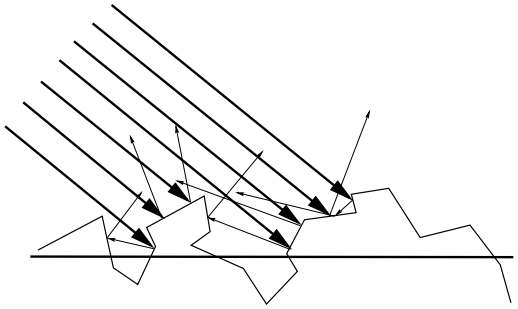
- The angle between the normal and incoming ray is maintained for the outgoing ray

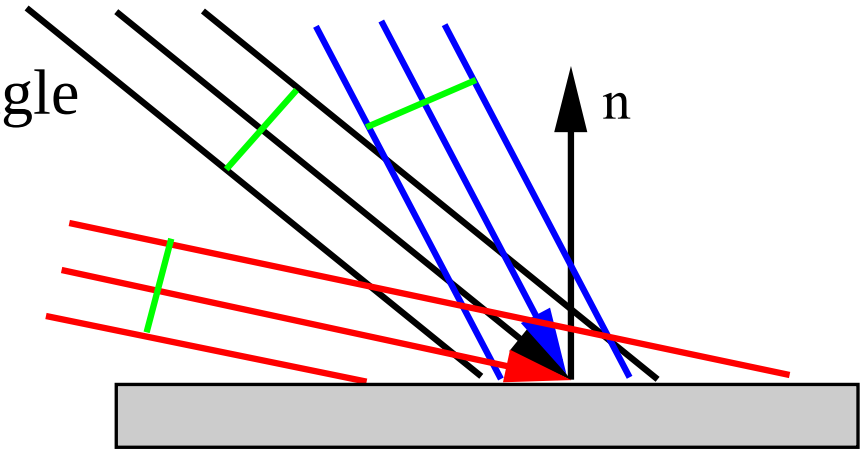


Diffuse Reflection

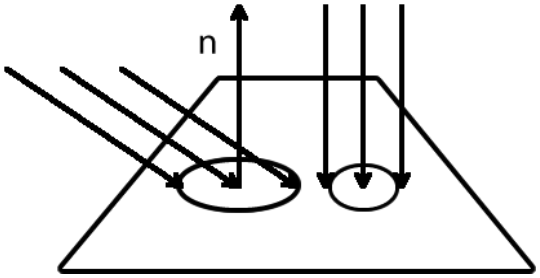


- An ideal diffuse reflector (a Lambertian Reflector, e.g. chalk) is the simplest to model.
- Incoming light is scattered equally in all directions, so brightness does not depend on the viewing direction.
- Reflected brightness depends on the direction and brightness of illumination (cos of light/normal angle





Lambertian Reflection



As angle between light and normal increases, light's energy spreads across a larger area

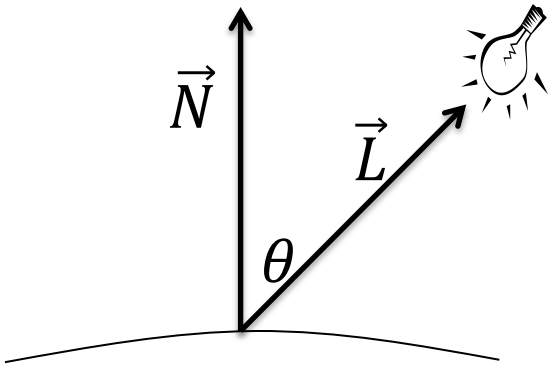
- Use Lambert's law, which says the Intensity of the reflected energy (light) depends upon the angle between the incoming light and the surface normal.
- The intensity is view independent!

$$I = I_{dir} k_d \cos\theta = I_{dir} k_d \vec{N} \cdot \vec{L}$$

- where

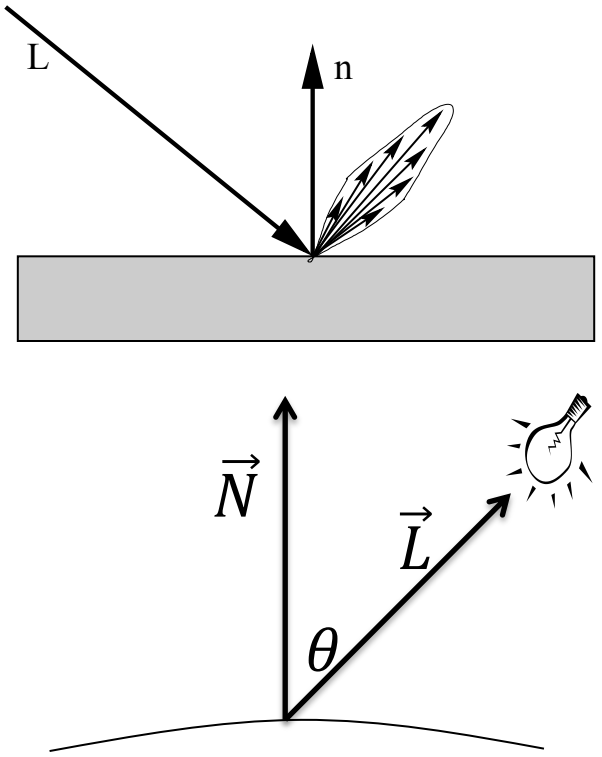
I_{dir} is the intensity (color) of the incident light (parallel to L),

k_d is the diffuse constant of the surface (0-1) (wavelength dependant)

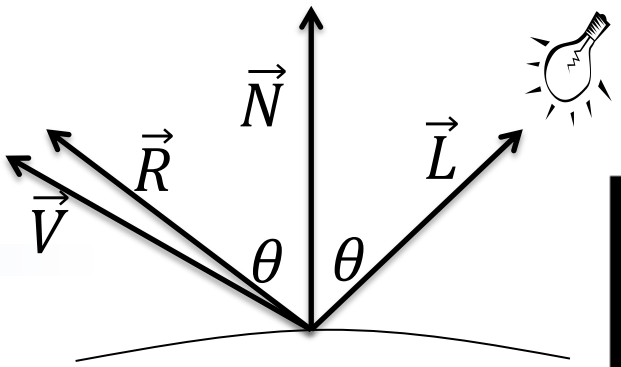


Specular Reflection

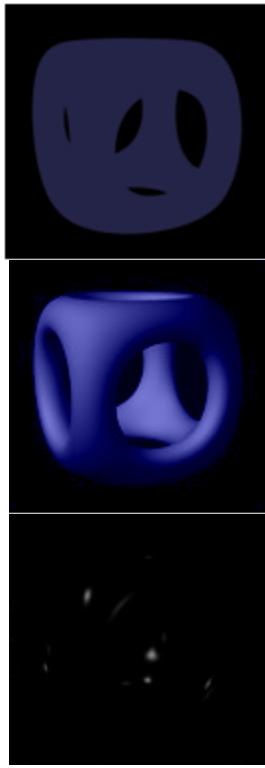
- Shiny surfaces reflect light coherently in a narrow beam (expanding cone) in the *specular* direction.
  - The specular direction is the angle of incident light reflected about the surface normal.
- If your eye is in that cone, the surface looks brighter (a highlight) that reduces away from the center
- The cone's width is dependent on smoothness of surface



Phong Illumination



- Ambient – Non-specific constant global lighting (hack) $I_a k_a$
- Diffuse –use Lambert's model to handle viewer independent diffuse reflections $Ik_d \vec{N} \cdot \vec{L}$
- Specular – Highlights on shiny objects (hack) $Ik_s \left(\vec{R} \cdot \vec{V}\right)^p$
  - Proportional to $(\boldsymbol{R} \cdot \mathbf{V})^{\alpha}$ so a larger α gives in a more concentrated highlight

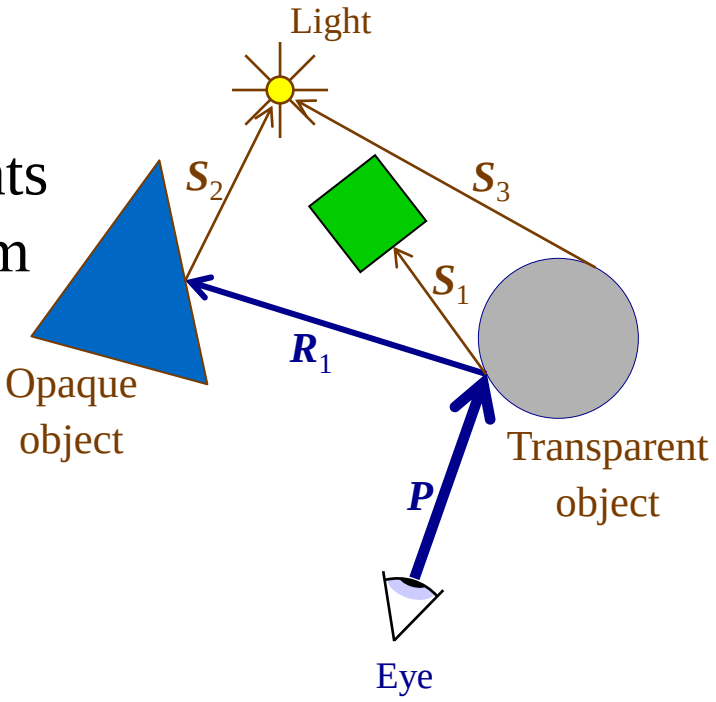


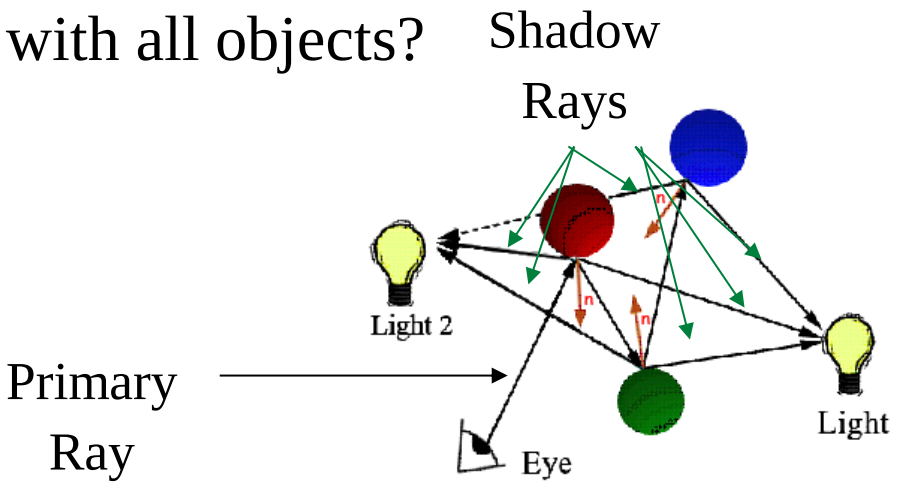
Only lights not in shadow

$$I = I_a k_a + \sum_{j}^{NumLights} I_j \left(k_d \vec{N} \cdot \vec{L} + k_s \left(\vec{R} \cdot \vec{V}\right)^p\right)$$

Shadow Rays

- Only want to add contribution from lights that aren't in shadow, that is visible from the surface.
- How do we figure this out?
- Where does the ray start?
- What is the direction of the ray?
- Should direction be unit?
- Should we intersect ray with all objects?



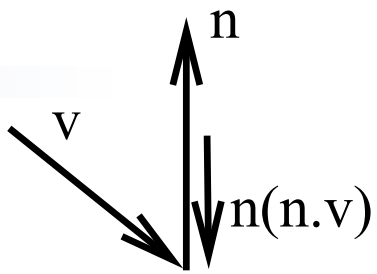


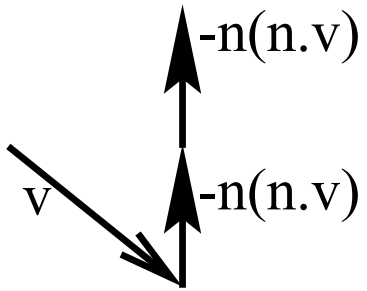
Reflection Rays

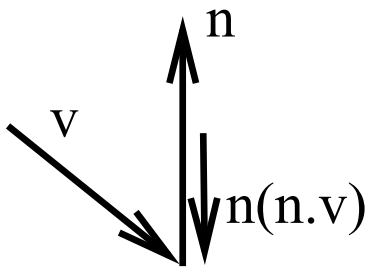
- If the object is reflective, shoot a reflection ray.
- The ray starts at the intersection point and has a direction

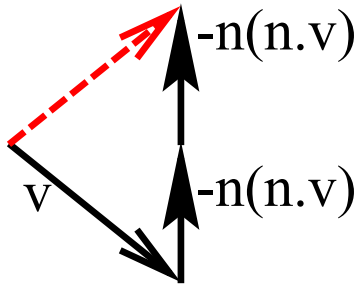
$$\vec{V} - 2\vec{N}\cdot\left(\vec{N}\cdot\vec{V}\right)$$

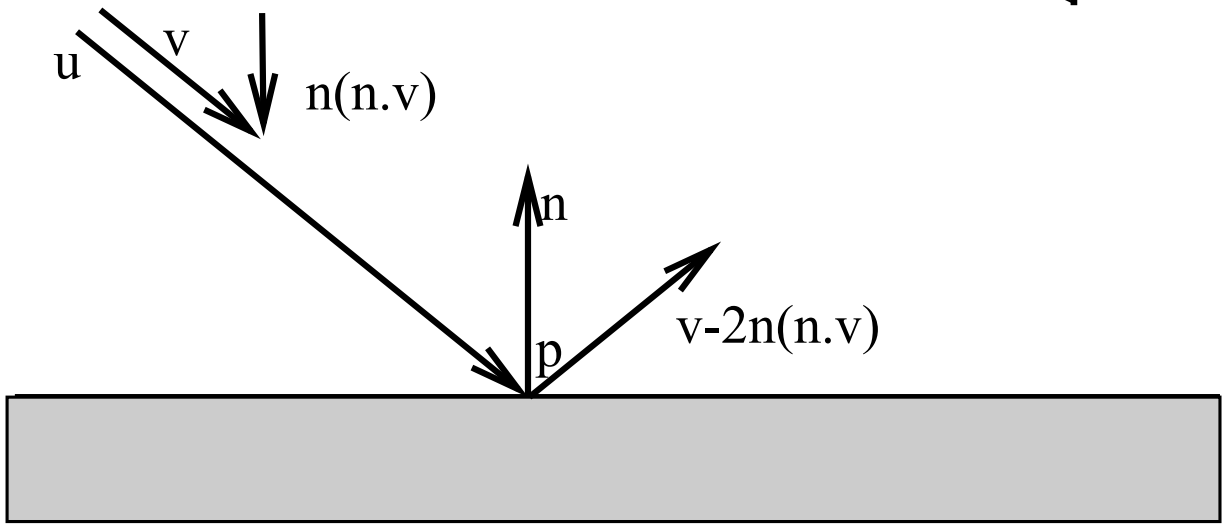
- Notice: unit vectors!
- What is V?







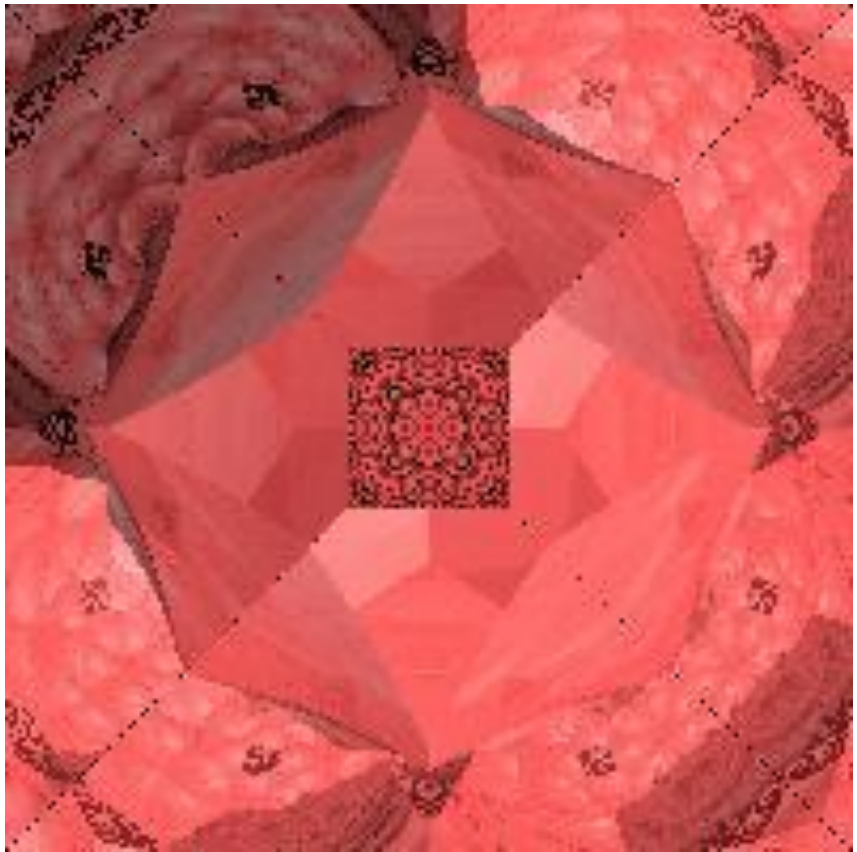


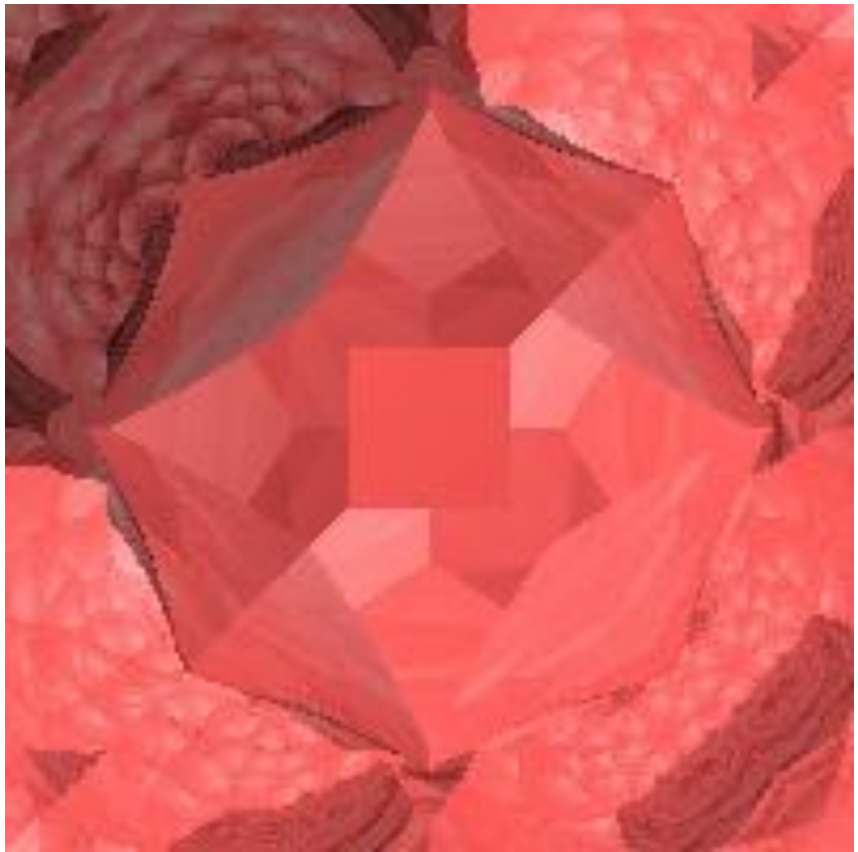


Cancer

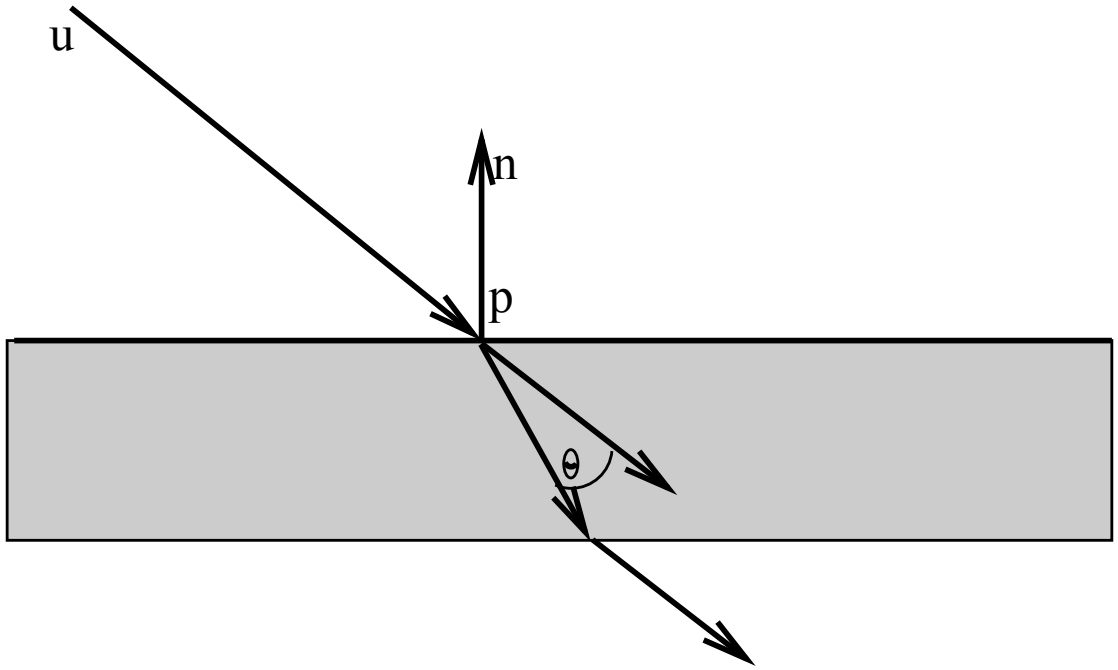
Numerical issues cause you to think reflection rays intersect with object when it doesn’t.

Move position off of surface by small amount

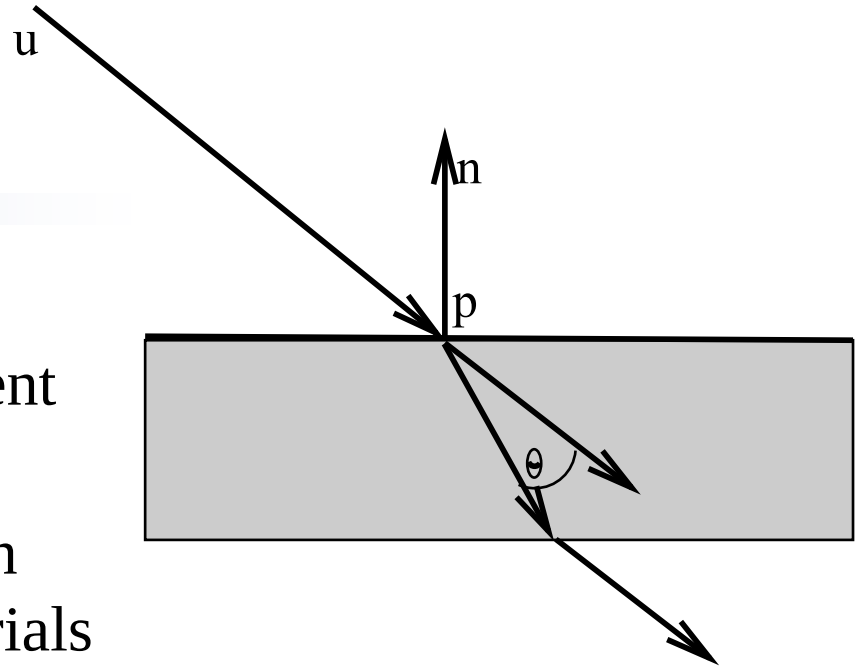




Transmission Rays

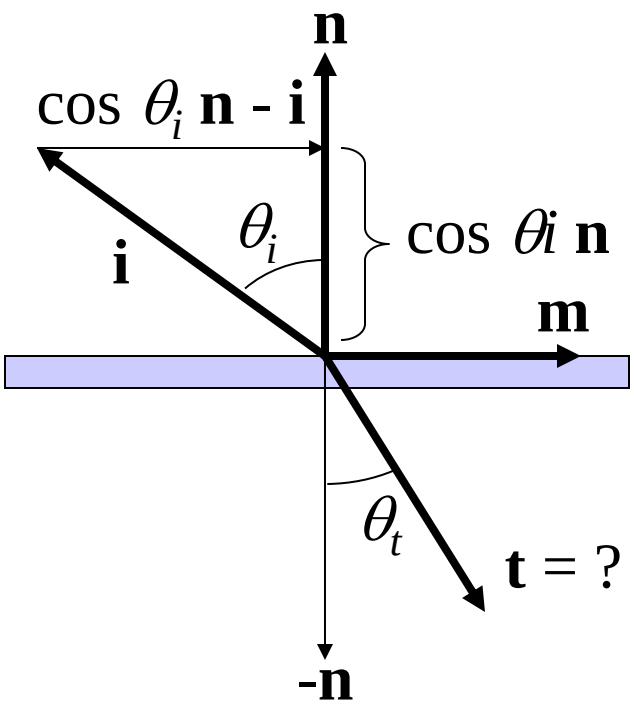


Refraction



- Why does refraction occur?
- Different velocities through different materials.
- Rays bend toward the normal when going from sparser to denser materials (air to water), and away for the opposite case.
- We treat all light as bending the same amount. Is this accurate?

Refract



Snell's Law: $\eta_i \sin\theta_i = \eta_t \sin\theta_t$

Let $\eta = \eta_i / \eta_t = \sin\theta_t / \sin\theta_i$

Let $\mathbf{m} = (\cos\theta_i\,\mathbf{n} - \mathbf{i}) / \sin\theta_i$

Then…

$$\mathbf{t} = \sin\theta_t\,\mathbf{m} - \cos\theta_t\,\mathbf{n}$$
$$= (\sin\theta_t / \sin\theta_i)(\cos\theta_i\,\mathbf{n} - \mathbf{i}) - \cos\theta_t\,\mathbf{n}$$
$$= (\eta\cos\theta_i - \cos\theta_t)\mathbf{n} - \eta\,\mathbf{i}$$

$$\mathbf{t} = \left(\eta(\mathbf{n}\cdot\mathbf{i}) - \sqrt{1-\eta^2(1-(\mathbf{n}\cdot\mathbf{i})^2)}\right)\mathbf{n} - \eta\mathbf{i}$$

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t}$$
$$= \sqrt{1-\eta^2\sin^2\theta_i}$$

Can be negative for grazing angles when $\eta > 1$, say when going from glass to air, resulting in *total internal reflection* (no refraction).

```
Ray refract(Ray r) {
    double ni = dot(n,-r.d);
    double eta = current_index/new_index;
    return Ray((x,eta*ni - sqrt(1 - eta*eta*(1-ni*ni)))*n + eta*r.d);
}
```

Refraction

- Refractive index:
- Light travels at speed c/n in a material with a refractive index n.
- Use Snell's law, n1 sin 1 = n2 sin 2, to calculate the refracted ray.
- Common values for index of refraction:
- air – 1.000293
- water - 1.33
- glass 1.5
- diamond 2.419
- http://en.wikipedia.org/wiki/List_of_refractive_indices

Refraction Demos

- http://lectureonline.cl.msu.edu/mmp/kap25/Snell/app.htm
- http://micro.magnet.fsu.edu/primer/java/refraction/index.html
- stwww.weizmann.ac.il/Lasers/laserweb/Java/Twoangles2.htm

Recursive Raytracer

Ray.start = camera;
for each pixel:
    Ray.direction = normalize(pixelcenter – Ray.start);
    for each object:
        Intersect Ray with object
    Select object with smallest positive t (visible object)
    if no object with a positive t, then use backgroud color and go to next pixel
    pixelColor = getIllumination(Ray, ObjectHit)

getIllumination(Ray, Hit)
    transform ObjectSpace Normal to world space
    get world space Hitpoint, and V
    for each *visible* light:
        calculate, R, L and local lighting using Phong Illumination model
    if object transparent, shoot refraction ray starting at hit, using Snell's law for dir
    if object shiny, shoot reflection ray starting at hit and reflect –V about N
    add all light contributions and return value

Recursive Ray Tracing

```
RayCast(screen)
for all pixels (x,y) in screen:
   trace(rayFromEyeThrough(x,y))
```

```
trace(ray)
  if (intersection= closestIntersection(ray))
    return Shade(intersection, ray)
  else return backgroundColor
```

```
closestIntersection(ray)
   for all objects find intersection
   for closest intersection return the intersection point, surface normal,
     surface, surface attributes, etc.
```

```
Shade(point, ray)
   Color = background;
   for each light
      if !Shadow(point, ray, light)
         Color += PhongIllumination(point, ray, light)
   if specularMaterial Color+=ks*trace(reflect(point, ray))
   if refractive Color+=kt*trace(refraction(point,ray))
   return Color
```

File Format

Command	description
#	A line that begins with a # is a comment line, ignore the entire line.
view ***n d***	The scene is ***n***X***n*** pixels. ***d*** is the distance to the corner of the image plane. The **perspective** camera at **(0,0,1)** and points toward the origin with **(0,1,0)** being the up vector. The image plane therefore runs from ***(-d,-d,0)*** to ***(d,d,0)***.
sphere	draw a sphere using the current material and transformed by the current transformation.
scale ***sx sy sz***	Adds a scale by ***sx sy sz*** to the current transformation.
move ***x y z***	Adds a translate to ***x y z*** to the current transformation.
rotate ***angle x y z***	Adds a rotate ***angle*** degrees about the axis defined by the vector ***(x y z)***
light ***r g b x y z***	Define a light located at ***(x y z)*** with color ***(r g b)***
background ***r g b***	Set the scene background color to ***(r g b)*** , the default is black (0,0,0)
ambient ***r g b***	Set the scene ambient light to ***(r g b)***.
group	group the following objects, transforms and materials together.
groupend	Signifies the end of a group.
material ***dr dg db sr sg sb p***	Sets the current material value to have a diffuse color of ***(dr,dg,db)*** a specular color of ***(sr,sg,sb)*** and a phong highlight coefficient of ***p***. The material becomes part of the state and all subsequent objects are of this material.
refraction ***r g b i***	Sets the refraction coefficients for the current material. Default is **(0 0 0 1.0002926)**. The default index for the scene will be 1.0002926 (air). This becomes part of the state, just like material.