Ray Tracing

Overview

- Method to produce realistic images
- Determines visible surface at pixel level
  - Operates at per-pixel level
  - Not at a per-surface level like that of z-buffer or BSP tree
- Can be rather CPU intensive

Benefits

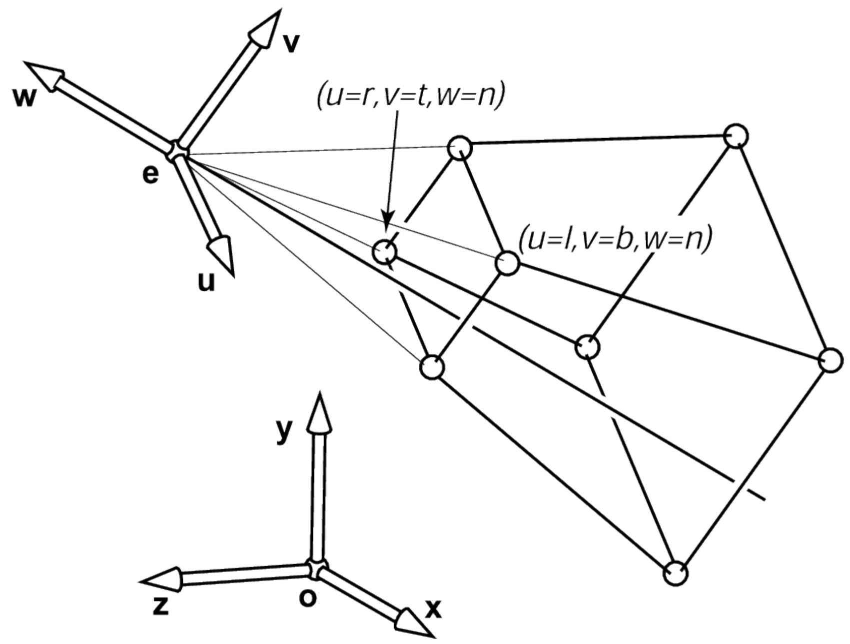
- Relatively straight-forward to compute shadows and reflections
- Ability to “pick” the object seen at a pixel
  - Could also perform this with other rasterization techniques if we stored a surface ID

Basics

- Simplest use is to produce images similar to z-buffer and BSP trees
- Make sure the appropriate surface is “seen” through each pixel
- Resultant colored based on:
  - Material
  - Surface normal
  - Lighting geometry

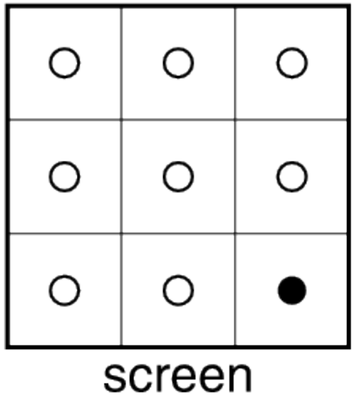
Viewing

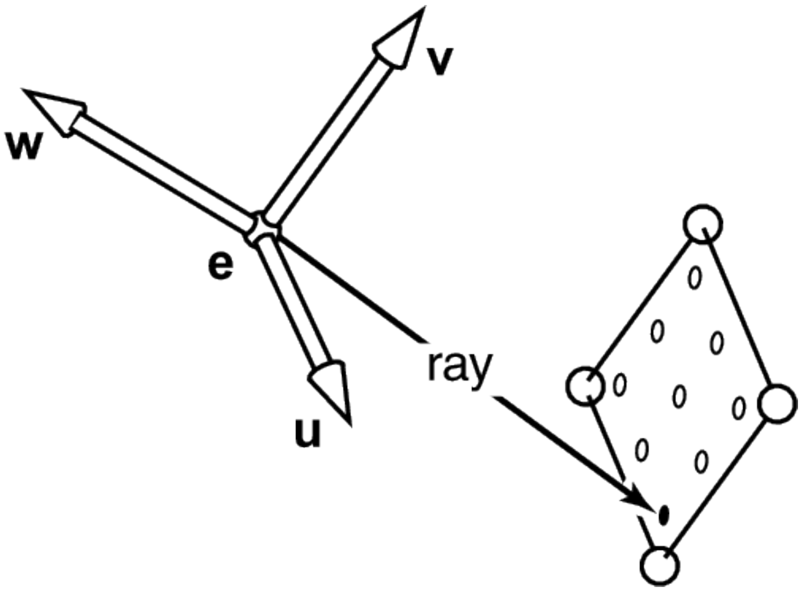
- Geometry is aligned with the origin (eye/ camera) at location e
- The border of the window have simple coordinates in the uvw coordinate system with respect to e



Rays

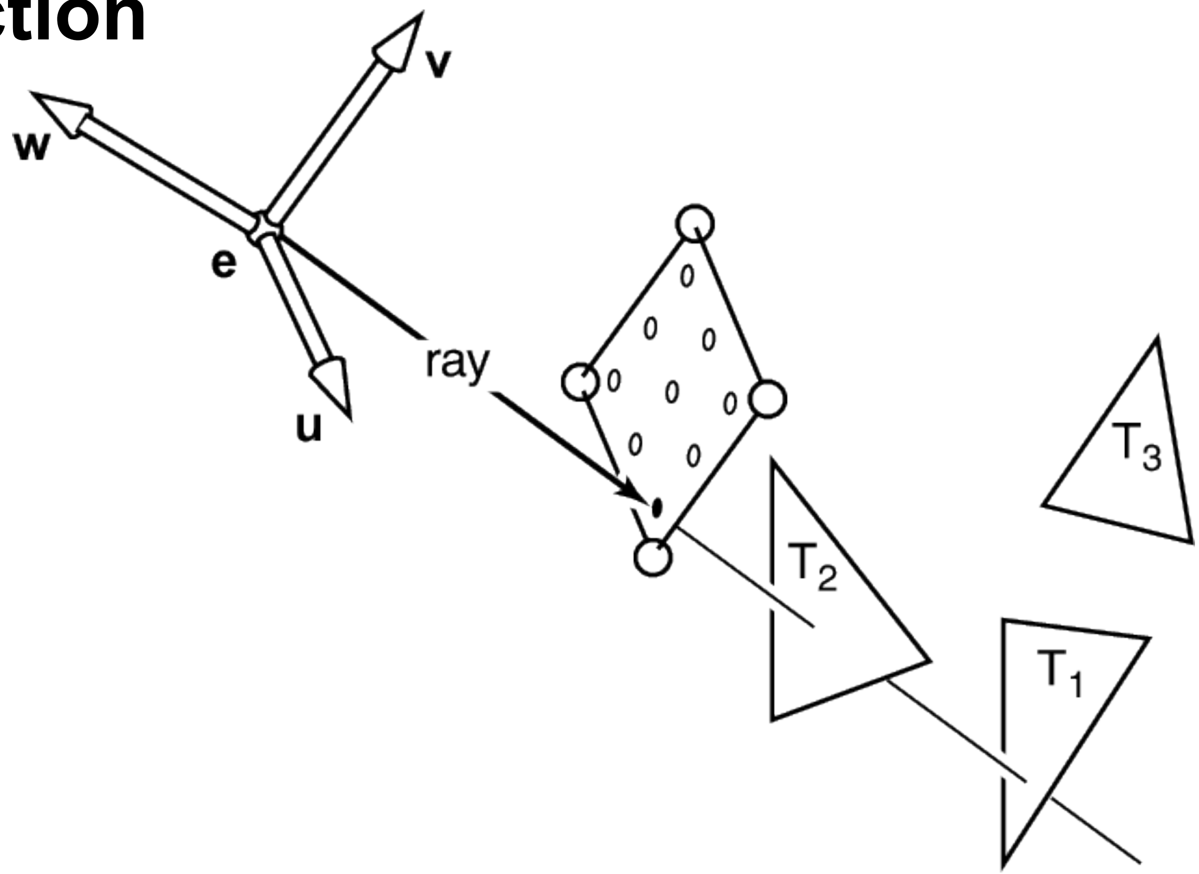
- **Basic idea in ray tracing is to identify locations on the w = n plane that correspond to pixel centers**
- **A ray is just a 3D line from the origin sent out to that point**





Casting Rays

- We then gaze in the direction of the ray to see the first object (if any) seen in that direction



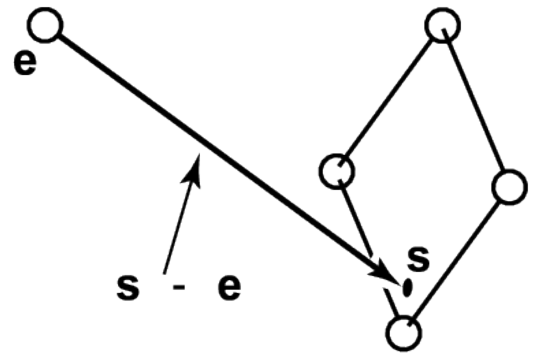
Basic Algorithm

compute *u*, *v*, *w*, basis vectors
    for each *pixel* **do**
        compute viewing *ray*
        find first object hit by *ray* and its surface normal *n*
        set *pixel* color to the value based on material, light, n

Computing Viewing Rays

- First, we need to determine a mathematical representation of a ray
  - Ray is just an origin point and a propagation direction
  - 3D parametric line is ideal for this
- Line from point ***e*** to a point ***s is***

$$\boldsymbol{p(t) = e + t(s - e)}$$



Finding Viewing Pixels

- First, we find the coordinates of s in the uvw coordinate system with origin e
- Using a windowing transform yields:

$$u_s = l + (r - l)\frac{i + .5}{n_x}$$

$$v_s = b + (t - b)\frac{j + .5}{n_y}$$

- Where *i* & *j* are pixel indices

Converting to Canonical Coordinates

- To convert to canonical coordinates:

$$\mathbf{s} = \mathbf{e} + u_s\mathbf{u} + v_s\mathbf{v} + w_s\mathbf{w}$$

- Or, in matrix form

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_e \\ 0 & 1 & 0 & y_e \\ 0 & 0 & 1 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_s \\ v_s \\ w_s \\ 1 \end{bmatrix}$$

Ray-Object Intersection

- Given a ray ***e*** + *t**d*** we want to find the first intersection where $t > 0$
- Smaller values of t indicate closer objects, whereas larger values of *t* indicate farther objects
- If 2 objects are both intersected, the one with the smallest *t* value (the closest) is recorded

Ray-Sphere Intersection

- Given a ray $p(t) = e + td$ **and an implicit surface** $f(p) = 0$**, we'd like to know where they intersect**
- **Intersection occurs when points satisfy the implicit equation**

$$f(p(t)) = 0$$

- **This is just**

$$f(e + td) = 0$$

Ray-Sphere Intersection

- A sphere with center ***c*** = *(x_c, y_c, z_c)* and radius *R* can be represented by the implicit equation

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = 0$$

- In vector form

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$$

- Any point ***p that satisfies this equation is on the sphere.***

Ray-Sphere Intersection

- ***Plugging*** in the ray we can solve for the values of *t on the ray which yields points on the sphere*

$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0$$

- *Rearranging terms yields*

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) - R^2 = 0$$

- *Everything is known except t, so this is a classic quadratic equation in t, meaning it has form*

$$At^2 + Bt + C = 0$$

Ray-Sphere Intersection

- Plugging in the actual terms for *t*:

$$t = \frac{-\mathbf{d}\cdot(\mathbf{e}-\mathbf{c}) \pm \sqrt{(\mathbf{d}\cdot(\mathbf{e}-\mathbf{c}))^2 - (\mathbf{d}\cdot\mathbf{d})((\mathbf{e}-\mathbf{c})\cdot(\mathbf{e}-\mathbf{c}) - R^2)}}{(\mathbf{d}\cdot\mathbf{d})}$$

- The discriminant (portion under the square root) determines how many real solutions there are
  - If discriminant is negative, there exists no real solutions, thus no intersection with sphere
  - If discriminant is positive, there exist 2 real solutions, ray entry and ray exit
  - If discriminant is zero, there exists a single solution, ray grazes surface at a single point

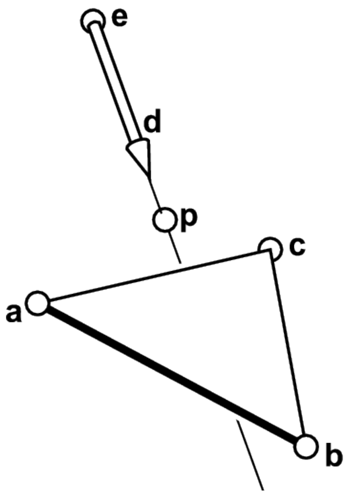
Ray-Sphere Intersection Efficiency

- The discriminant alone is sufficient for determining if there is an intersection or not
- Check discriminant first, if negative there is no intersection
  - Abort further computation of the rest of the formula

Ray-Triangle Intersection

- There are a number of methods for determining ray-triangle intersection, we'll utilize a barycentric approach
- If ***a***, ***b*** & ***c*** are the vertices of triangle we know that the ray intersects the plane if and only if

$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$



Ray-Triangle Intersection

- The hit point ***p*** will be at ***e*** + *t**d***
- ***We know that the hit point is inside the triangle if and only if***

$$\beta > 0, \gamma > 0, \beta + \gamma < 1$$

- ***Otherwise, it hits the plane outside of the triangle***

Ray-Triangle Intersection

- To solve for t, beta and gamma, expand from vector form into 3 equations (one for each coordinate)

$$x_e + tx_d = x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)$$
$$y_e + ty_d = y_a + \beta(y_b - y_a) + \gamma(y_c - y_a)$$
$$z_e + tz_d = z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)$$

- Can be rewritten as a standard linear equation

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}$$

Ray-Triangle Intersection

- Solve using Cramer's rule

$$\beta = \frac{\begin{vmatrix} x_a - x_e & x_a - x_c & x_d \\ y_a - y_e & y_a - y_c & y_d \\ z_a - z_e & z_a - z_c & z_d \end{vmatrix}}{|A|} \qquad \gamma = \frac{\begin{vmatrix} x_a - x_b & x_a - x_e & x_d \\ y_a - y_b & y_a - y_e & y_d \\ z_a - z_b & z_a - z_e & z_d \end{vmatrix}}{|A|}$$

$$t = \frac{\begin{vmatrix} x_a - x_b & x_a - x_c & x_a - x_e \\ y_a - y_b & y_a - y_c & y_a - y_e \\ z_a - z_b & z_a - z_c & z_a - z_e \end{vmatrix}}{|A|}$$

- Where ***A*** is

$$A = \begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix}$$

Ray-Triangle Intersection

- Substituting dummy values

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}$$

- Re-expressed using Crammer's rule

$$\beta = \frac{j(ei-hf)+k(gf-di)+l(dh-eg)}{M}$$

$$\gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M}$$

$$t = -\frac{f(ak-jb)+e(jc-al)+d(bl-kc)}{M}$$

$$M = a(ei-hf)+b(gf-di)+c(dh-eg)$$

Ray-Triangle Intersection

- Substituting dummy values

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}$$

- Re-expressed using Crammer's rule

$$M = g(bf - ec) + h(dc - af) + i(ae - bd)$$

$$t = \frac{j(bf - ec) + k(dc - af) + l(ae - bd)}{M}$$

$$\beta = \frac{d(hl - ki) + e(ji - gl) + f(jh - gk)}{M}$$

$$\gamma = -\frac{a(hl - ki) + b(ji - gl) + c(jh - gk)}{M}$$

Ray-Triangle Intersection Efficiency

- There are a number of efficiencies that can be introduced
  - Solve expressions (such as *ei – hf) once and store them, as they are used again*
  - *Solve for t, if outside of viewing parameters, abort rest of calculations*

Ray-Triangle Intersection Efficiency

- Optimized version of the ray-triangle intersection that has early conditions for termination

boolean raytri(ray *r*, vector *a*, vector *b*, vector *c*, interval $[t_0, t_1]$)
 compute t
 if $(t < t_0)$ or $(t > t_1)$ **then**
  return false
 compute γ
 if $(\gamma < 0)$ or $(\gamma > 1)$ **then**
  return false
 compute β
 if $(\beta < 0)$ or $(\beta > 1 - \gamma)$ **then**
  return false
 return true

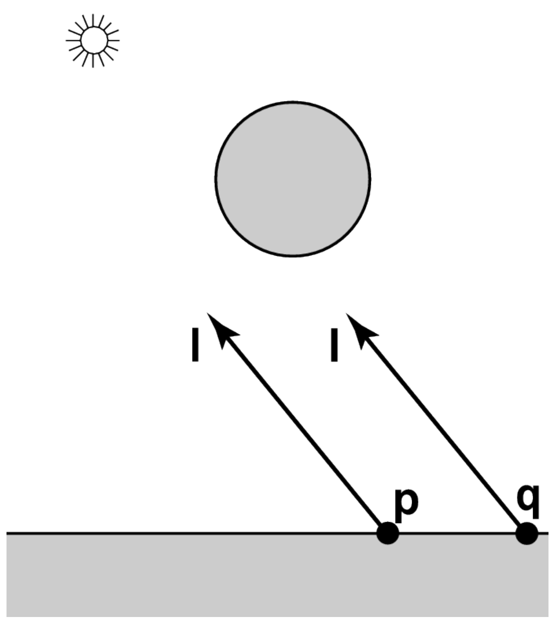
Shadows

- Shadows can easily be added to ray tracers

Call of the Wild - Gilles Tran, Bonsais - Jaime Vives Piqueres

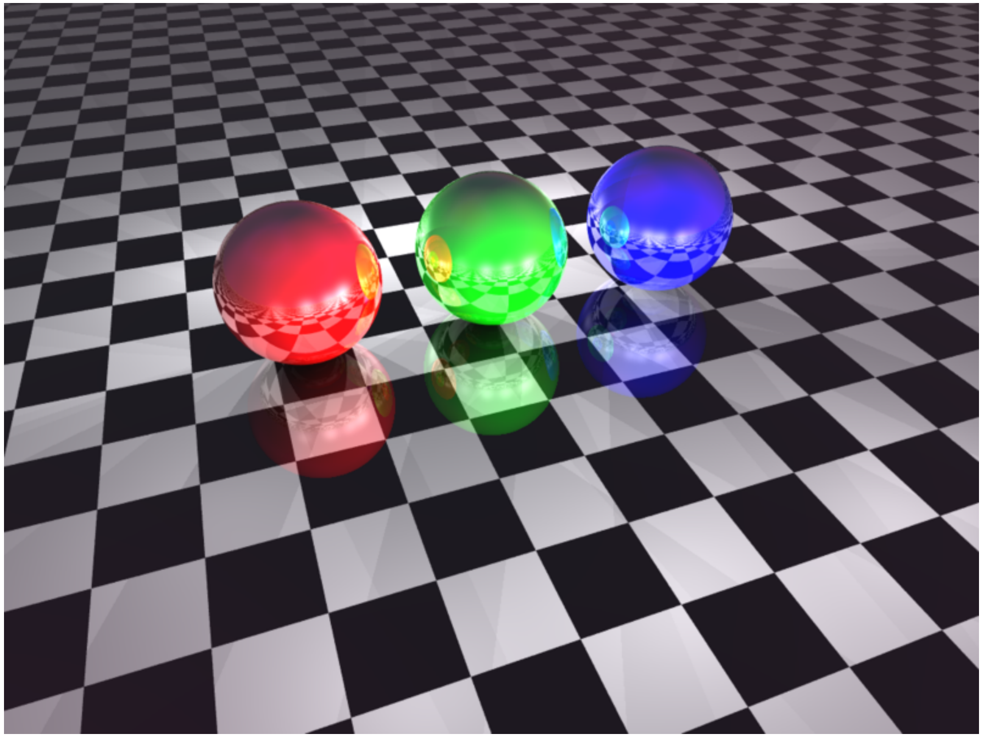
Shadows

- If we imagine our point on the surface being shaded, it is in shadow if we look into the direction of the light and cannot see it
  - Rays from *p*/*q* to *l* known as shadow rays
  - May be multiple light sources to check against



Specular Reflection

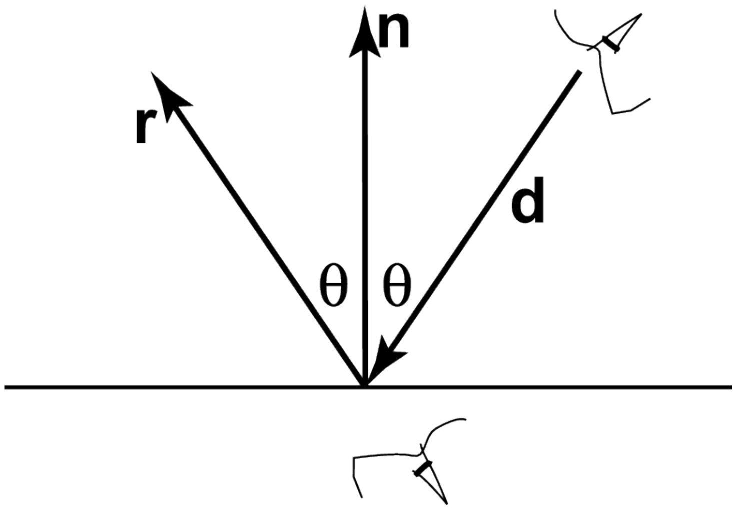
- Mirror-like reflection of light



The Dark Side of the Trees - Gilles Tran, Spheres - Martin K. B.

Specular Reflection

- Key to specular reflection is to viewer looking in direction ***d sees whatever the viewer "below" the surface sees looking in direction r***
- ***In the real world***
  - ***Energy loss on the bounce***
  - ***Loss different for different colors***



Refraction

- Change in direction of light wave

glass - Bcrowell, Sphere - László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, Mátyás Premecz

Refraction

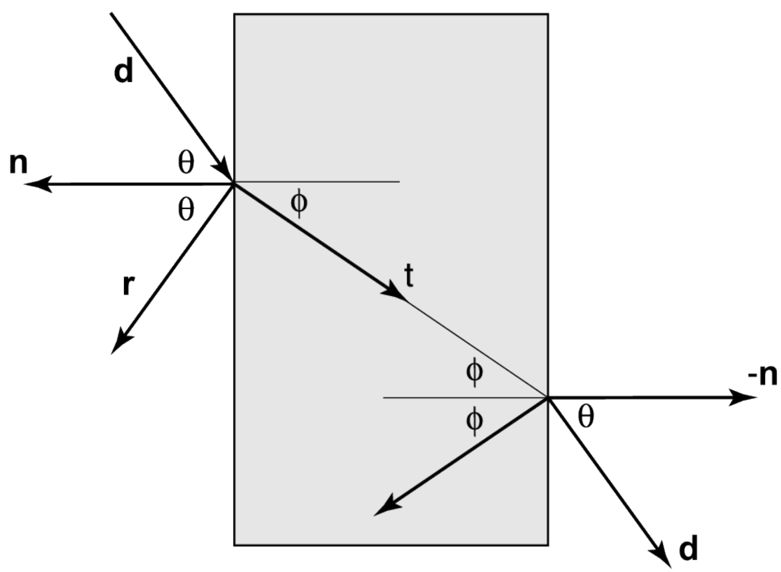
- Change in direction of light due to a change in speed
  - Typically a result of light passing from one medium to another
- When ray travels from a medium with refractive index n into one with refractive index n_t, some light is transmitted and bends

Snell's Law

- Snell's law tells us that

$$n \sin \theta = n_t \sin \Phi$$

- Note that if n_t *and* n_t *are reversed then the angles are as well (right side of picture)*



Ray Tracing – Optimization

- Bounding boxes
- Hierarchical bounding boxes
- Uniform spatial subdivision
- Binary space partitioning

Additional Features

- Constructive Solid Geometry (CSG)
- Antialiasing
- Soft shadows
- Depth of field
- Glossy
- Motion Blur