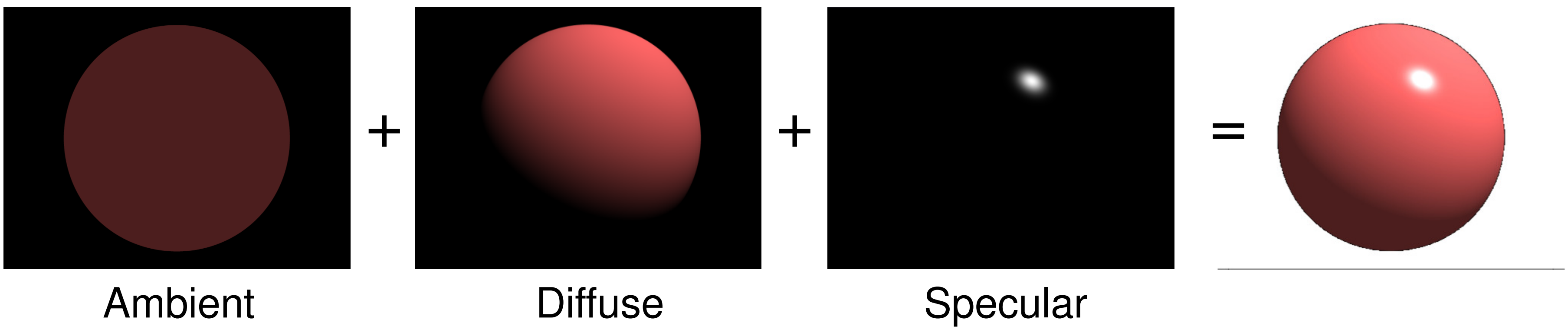
CSC 321 Computer Graphics

Ray Tracing

Review

- Local Illumination Model (1-hop reflection only)
  - Non-physical model: "looks good"
  - Ambient, diffuse and specular components



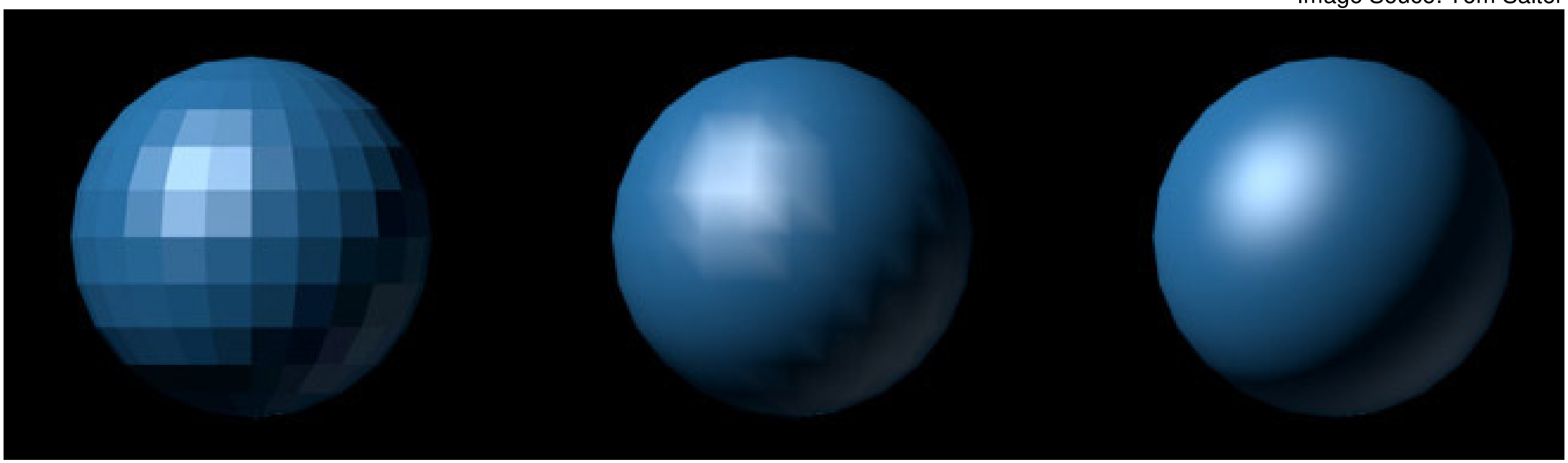
$$I = I_{amb} + I_{diff} + I_{spec}$$

$$= I_A k_a + I_L f_{att} (k_d (N \cdot L) + k_s (R \cdot V)^n)$$

Review

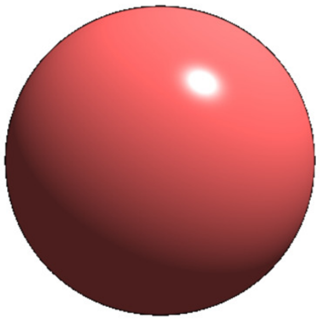
- Drawing polygons using local illumination
  - Visibility culling (z-buffer)
  - Shading (flat, Gouraud, and Phong)
  - Texturing

Image Souce: Tom Salter

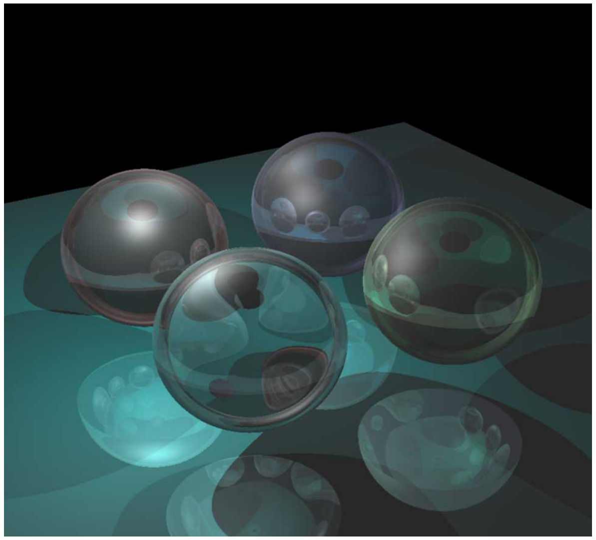


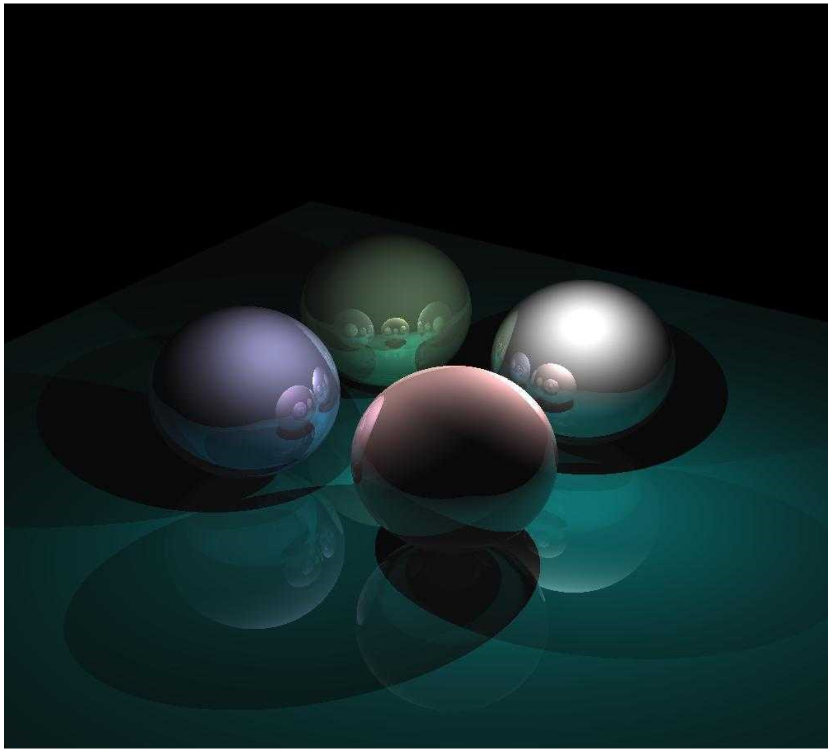
Flat Shading Gouraud Shading Phong Shading

What are we missing?



Local
illumination

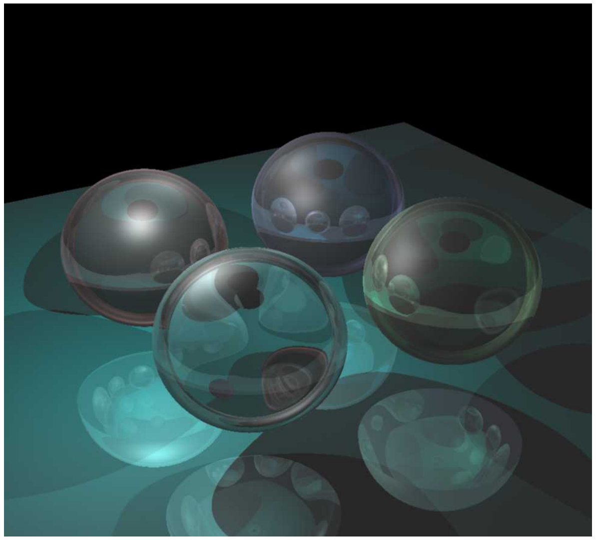


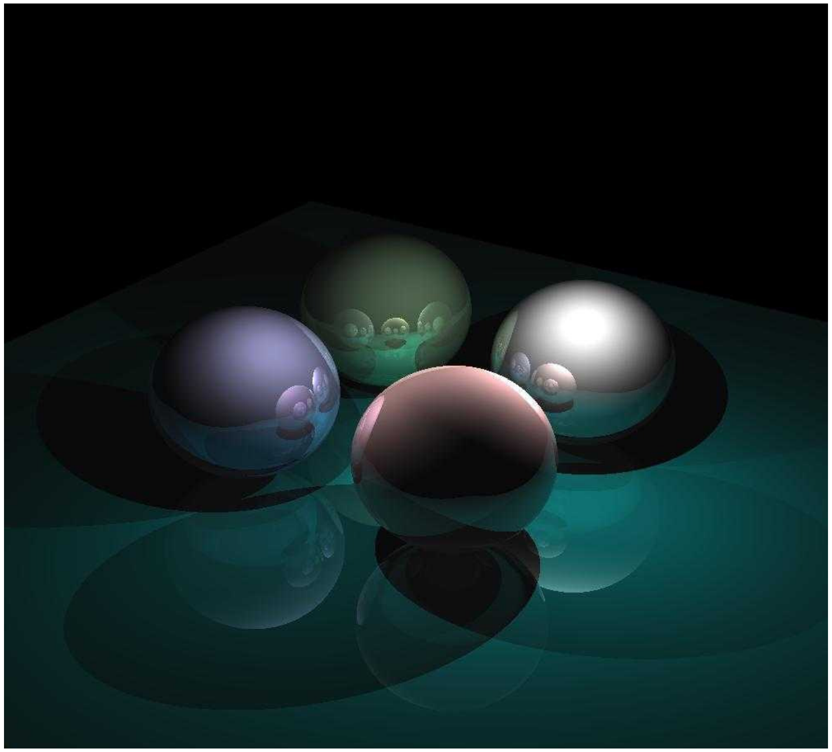


Global
illumination
By Michael Moran, 2000

Ray Tracing

- A global illumination method
  - Shadows
  - Reflection
  - Refraction





Global illumination

By Michael Moran, 2000

What Is Ray Tracing

- Goal: Capture multiple hops of light rays
- Forward ray casting
  - Trace the path of each ray coming out of the light source
    - Expensive, and many rays don't contribute to the rendering
- Backward ray casting
  - Trace backwards in each view direction
    - Initiate one ray per pixel
    - When the ray hits a surface, calculate color using local illumination (if not in shadow), and spawn new rays along reflective and refractive directions
    - Accumulate color for all rays

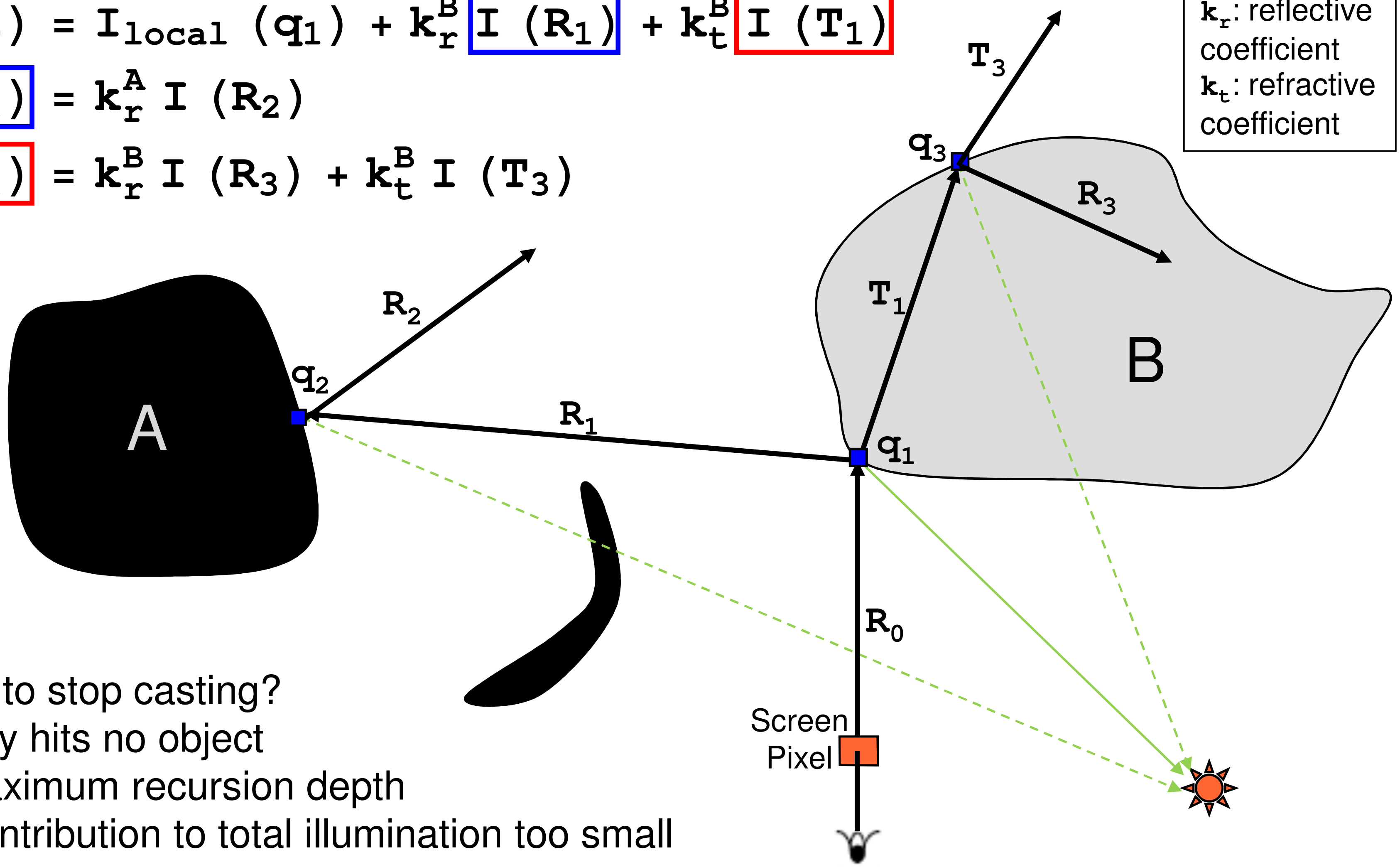
Backward Ray Casting

$$I(R_0) = I_{local}(q_1) + k_r^B\, I(R_1) + k_t^B\, I(T_1)$$

$$I(R_1) = k_r^A\, I(R_2)$$

$$I(T_1) = k_r^B\, I(R_3) + k_t^B\, I(T_3)$$

k_r: reflective coefficient
k_t: refractive coefficient



When to stop casting?

1. Ray hits no object
2. Maximum recursion depth
3. Contribution to total illumination too small

Recursive Algorithm

- Main loop

```
For each pixel on the screen
      Form a ray L from the eye to the pixel
      pixel color = RayTrace(L)
```

- Recursive ray-tracer

```
RayTrace(L)
      Find nearest intersection of L with all surfaces
      If no intersection found
            Return 0
      Else
            Compute local illum. I at intersection
            Cast reflection ray R, refraction ray T
            Return I + k_r RayTrace(R) + k_t RayTrace(T)
```

- That's all!

Forming A Ray

- Locating a pixel (i,j) in world coordinates
  - Viewport: **w** pixels wide, **h** pixels high
  - 3D pixel location (on the far plane) after WTC transform:

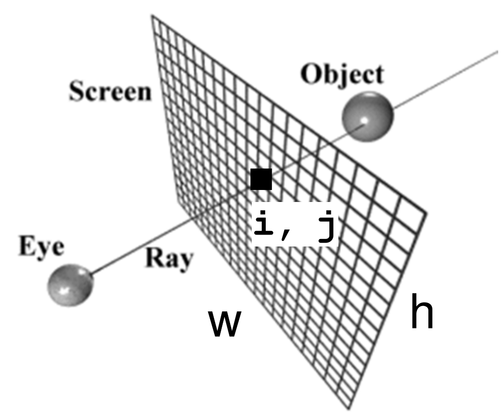
$$q_s = \left\{ (i + 0.5)\,\frac{2}{w} - 1,\; 1 - (j + 0.5)\,\frac{2}{h},\; -1 \right\}$$

  - 3D pixel location in world coordinates:

$$q_w = (S_{xyz}\, S_{xy}\, R\, T)^{-1}\, q_s = T^{-1}\, R^{-1}\, S_{xy}^{-1}\, S_{xyz}^{-1}\, q_s$$

- Representing the ray (parametric equation)
  - Eye point: P

$$P + t\,(q_w - P)$$



Ray-Object Intersection

- General approach
  - Represent ray in *parametric* form

    `q = P + t d`
  - Represent surface in *implicit* form

    `f[q] = 0`
  - Substitute ray into surface, and solve for *t* (P, d are known)

    `f[P + t d] = 0`
  - Substitute *t* back into ray equation, find intersection point q
    - Use the *smallest positive t* (to find nearest intersection point)

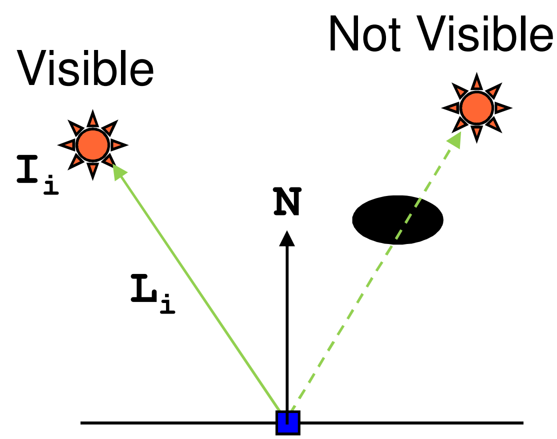
When Ray Hits A Surface…

- Compute local illumination at the intersection
  - If not occluded, compute diffuse and specular light
  - Add ambient light
- Cast more rays and keep tracing
  - Reflected ray (if the reflection coefficient is not zero)
  - Refracted ray (if the refraction coefficient is not zero)
- Sum up all illumination along traced rays

Computing Illumination

- Local illumination at intersection
  - Ambient reflection: $\mathbf{I_{amb} = I_A\, k_a}$
  - Cast a *shadow ray* to each light source
    - A light source is *visible* if the ray is unblocked
  - For each visible light source i:
    - Diffuse reflection: $\mathbf{I_{i,diff} = I_i\, f_{att}\, k_d\, (N \cdot L_i)}$
    - Specular reflection: $\mathbf{I_{i,spec} = I_i\, f_{att}\, k_s\, (R_i \cdot V)^n}$
  - Together:

$$\mathbf{I_{local} = I_{amb} + \Sigma_{visible\ source\ i}\, (I_{i,diff} + I_{i,spec})}$$



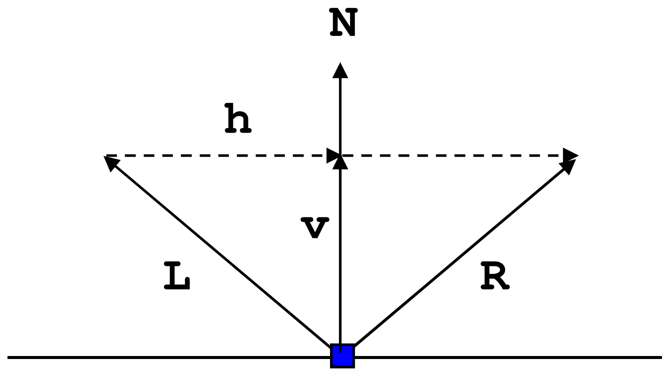
Reflection Ray

- Mirrored by the surface normal

$$\mathbf{v} = (\mathbf{L} \cdot \mathbf{n})\ \mathbf{n}$$

$$\mathbf{h} = \mathbf{v} - \mathbf{L}$$

$$\mathbf{R} = \mathbf{L} + 2\,\mathbf{h} = 2\,(\mathbf{L} \cdot \mathbf{n})\ \mathbf{n} - \mathbf{L}$$



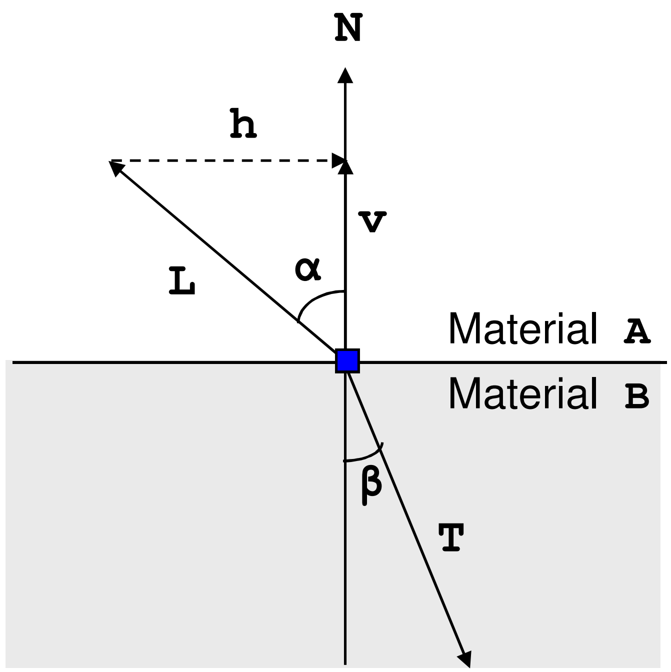
Refraction Ray

- Snell's Law

$$\frac{\text{Sin}[\alpha]}{\text{Sin}[\beta]} = \frac{\eta_B}{\eta_A}$$

  – η_A, η_B: refraction index (speed of light in vacuum / speed of light in that material)
  – Compute refracted ray T:

$$T = \frac{\text{Tan}[\beta]}{\text{Tan}[\alpha]} h - v$$



Recursive Algorithm

- Main loop

```
For each pixel on the screen
        Form a ray L from the eye to the pixel
        pixel color = RayTrace(L)
```

- Recursive ray-tracer

```
RayTrace(L)
        Find nearest intersection of L with all surfaces
        If no intersection found
                Return 0
        Else
                Compute local illum. I at intersection
                Cast reflection ray R, refraction ray T
                Return I + k_r RayTrace(R) + k_t RayTrace(T)
```

Examples

- Internet Ray Tracing Competition (irtc.org)

First Place, January-February 2006

Examples

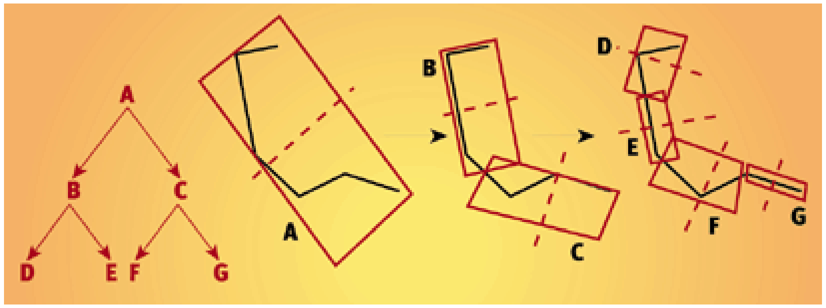
- Internet Ray Tracing Competition (irtc.org)

Third Place, January-February 2006

Speed Up Ray Intersection

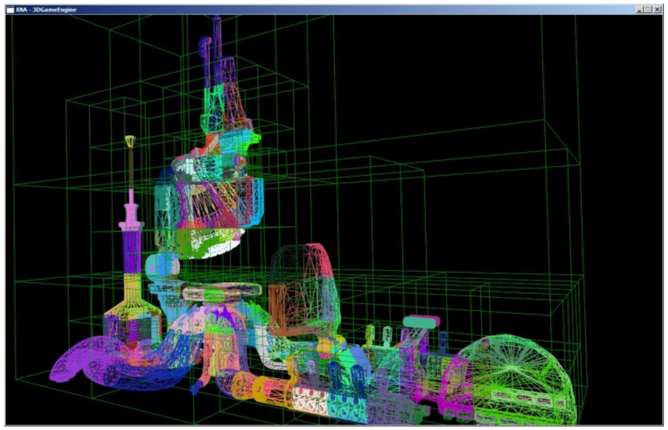
- Bounding boxes
  - Using coarse bounding objects for intersection first
    - If no intersection, than ignore the entire object
    - If yes, than intersect with the actual object
  - Types
    - Sphere (ellipsoid)
    - Axes-aligned bounding boxes (AABB)
    - Oriented bounding boxes (OBB)
  - Often hierarchical

An OBB tree



Speed Up Ray Intersection

- Spatial partitioning
  - Divide space up into small cells
    - Record objects in each cell
    - Trace cells along the ray, intersect only with objects in the cells
  - Types
    - Uniform 3D lattice
    - Adaptive lattice (octree, k-d tree)
    - Binary space partitioning (BSP)



An octree