Global Illumination: Ray Tracing

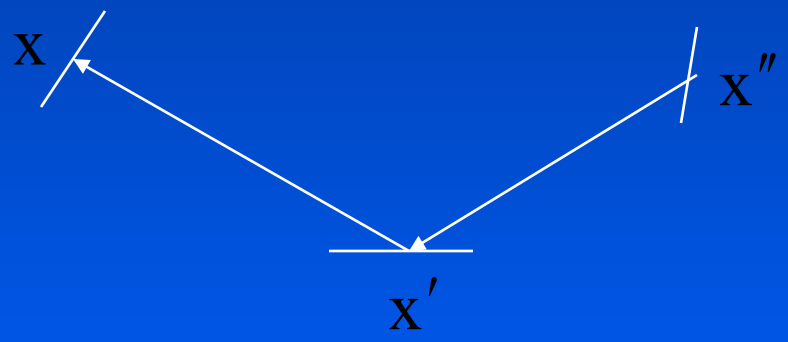
Global Illumination

- View-dependent : (e.g. Ray-tracing)
  - Discretize view plane to evaluate illumination equation
  - Good for specular phenomena that depend on view position
  - Bad for diffuse phenomena that very little over image plane
  - Can’t take advantage of previous calculation

- View-independent : (e.g. Radiosity)
  - Discretize object space surfaces
  - Bad for specular
  - Good for diffuse
  - Followed by view-dependent step

Rendering Equation [Kajiya 86]

- Light going from x’ to x is sum of:
  - light emitted by x’ in the direction of x
  - light that comes from everywhere x’’ to x’ that is reflected to x



Rendering Equation

$$I(x,x') = g(x,x')[\varepsilon(x,x') + \int_s \rho(x,x',x'')I(x',x'')dx'']$$

$g(x,x') = 0$ when x,x' occluded

$$= \frac{1}{r^2} \text{ when visible}$$

$\varepsilon(x,x')$ - light emitted by x' to x

$\rho(x,x',x'')$ - fraction of light scattered from x'' to x by x'

$I(x',x'')$ - intensity of light passing from x'' to x'

s - union of all surfaces

Classical Illumination

- x: eye, x': surface, x'': light
- Consider only first scatter
- Classical visible surface algorithm

Radiosity [Cohen 85]

- Discretize space (x, x', x'' now surface patches)
- Solve each equation simultaneously
- Assume diffuse reflection (ρ uniform)
- Problem with specular♪

Recursive ray tracing [Whitted 79]

- X' and X" only in perfectly reflective direction
- Start with center of projection and trace ray backward through the pixels
- Trace additional rays at intersections to handle
  - Shadows: trace ray to light source
    If blocked by opaque object, point is in shadow
  - Reflection: trace ray in mirror direction by reflecting about N
  - Refraction: trace ray in refraction direction by Snell's law
- Recursively spawn new shadow, reflection, refraction rays at each intersection

- Note we use both diffuse and specular components in the illumination equation
- Hybrid of recursive ray-tracing and local illumination model

$$I_\lambda = k_{a\lambda} I_a + \sum_{1 \le i \le m} S_i f_{att_i} I_{p\lambda_i} [k_d (\overline{N} \cdot \overline{L}_i) + k_s (\overline{R}_i \cdot \overline{V})^n] + k_s I_{r\lambda} + k_t I_{t\lambda})$$

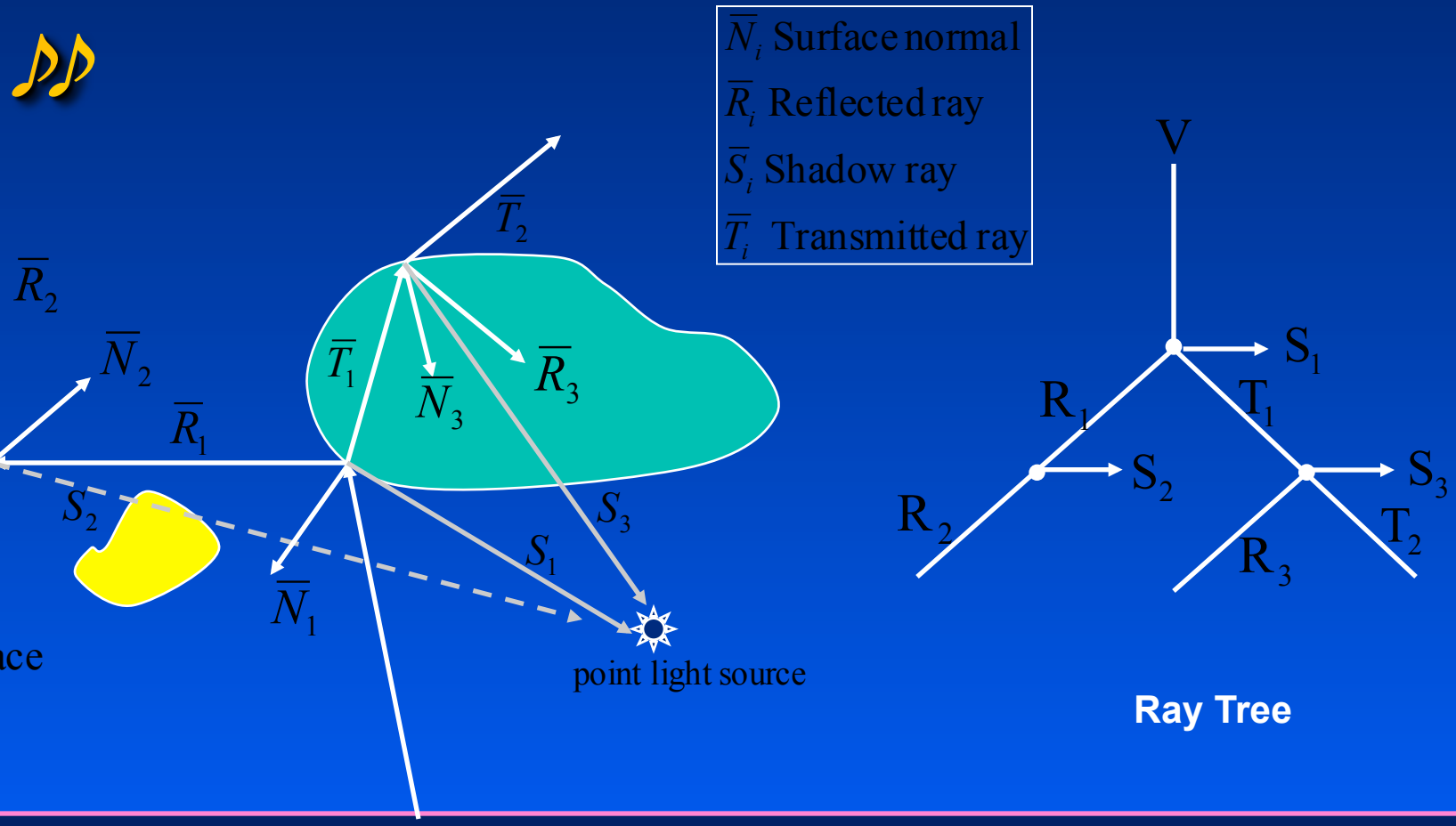
$I_{t\lambda}$ - intensity of refracted(transmitted) ray

k_t - transmission coefficient ($o < k_t < 1$)

$I_{r\lambda}$ - intensity of reflected ray

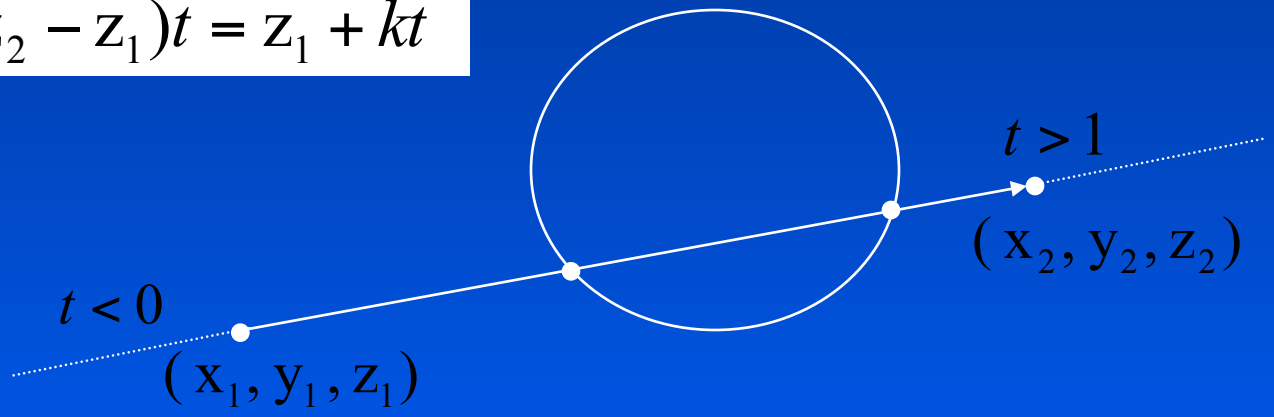
S_i - shadow ray

Recursive rays example



Ray/Sphere intersection

- Ray given by (x_1, y_1, z_1) and (x_2, y_2, z_2)
- $$x = x_1 + (x_2 - x_1)t = x_1 + it$$
  $$y = y_1 + (y_2 - y_1)t = y_1 + jt$$
  $$z = z_1 + (z_2 - z_1)t = z_1 + kt$$



- Sphere center at (l,m,n) radius r

$$(x-l)^2+(y-m)^2+(z-n)^2=r^2$$

- Substitute x, y, z into above
  - quadratic equation of form $at^2+bt+c=0$
- Find root, substitute into line equation to get intersection point
- Normal at intersection point: $N=(\frac{x_i-l}{r},\frac{y_i-m}{r},\frac{z_i-n}{r})$

Ray/Polygon intersection

- Equation for plane containing polygon

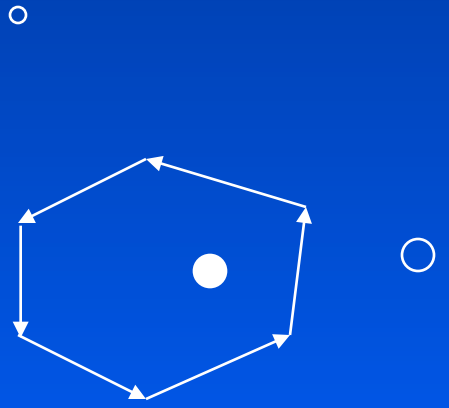
$$ax + by + cz + d = 0$$

  - N for plane: (a, b, c)
  - d- substitute any point into (x,y,z)
- Check for intersection between plane and parametric ray
  - plug parametric equation into above solve for t
  - substitute back into parametric equation to get values for x, y, z

$$t = -\frac{ax_1 + by_1 + cz_1 + d}{ai + bj + ck}$$

$$\Rightarrow x_i,\ y_i,\ z_i$$

- Check to see if (x_i, y_i, z_i) is inside polygon
  - Point is “left” of each edge
  - Sum of angles between point and each vertex=360



Efficiency

Adaptive depth control

- More nodes further down tree, less influence
- Attenuation due to reflection/transmission coefficient and passing through medium
- Accumulate product of reflection/transmission coefficient
- Stop recursion when attenuation is below threshold

Culling using Bounding volumes

- E.g. spheres, cubes, …
- Tradeoff between fit and cost of intersection test
- Hierarchy of bounding volumes

First-hit speedup

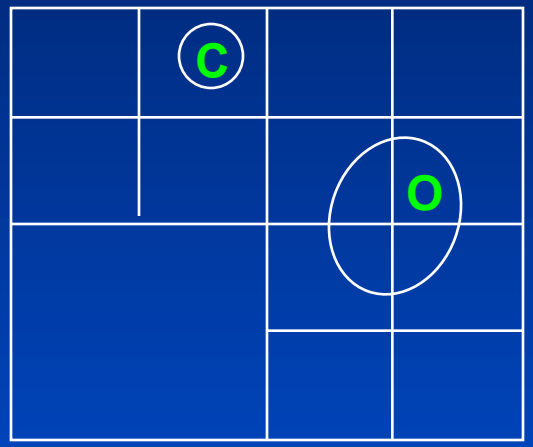
- Average depth of ray 1-2
- Use z-buffer to calculate first hit, use regular ray-tracing from then on
- Can be used selectively to ray trace certain objects

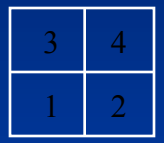
Culling using Spatial Subdivision

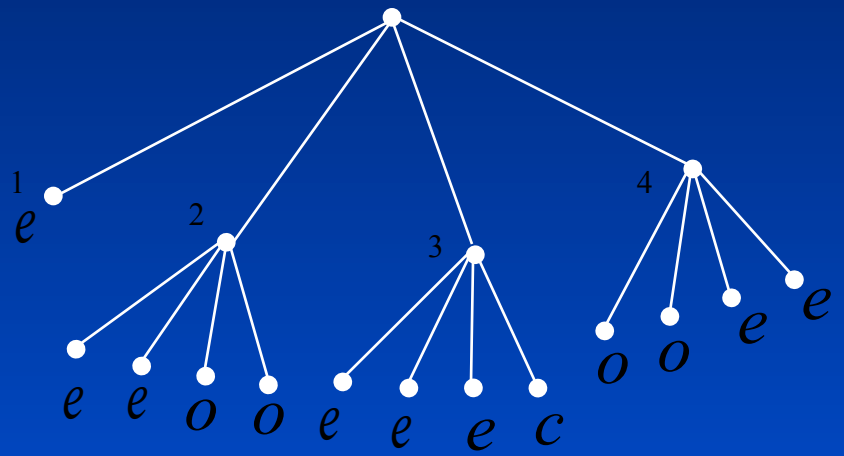
- Exploit spatial coherence
- Subdivide space into non-overlapping regions
- Each sub-region contains a number of objects
- If a ray is in a particular region, can quickly determine which objects are potential candidates

Octrees

- Hierarchical subdivision of space
- Voxels: 3-D volume elements
- Any region containing an object or part of object subdivided – until small enough



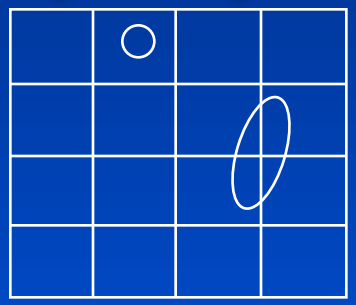




- As ray traverse space, go from one terminal sub-region to next
- As we decrease size of terminal voxel, “tight fit” but more terminal voxels
- Adoptable to scenes with varying density

Uniform space subdivision

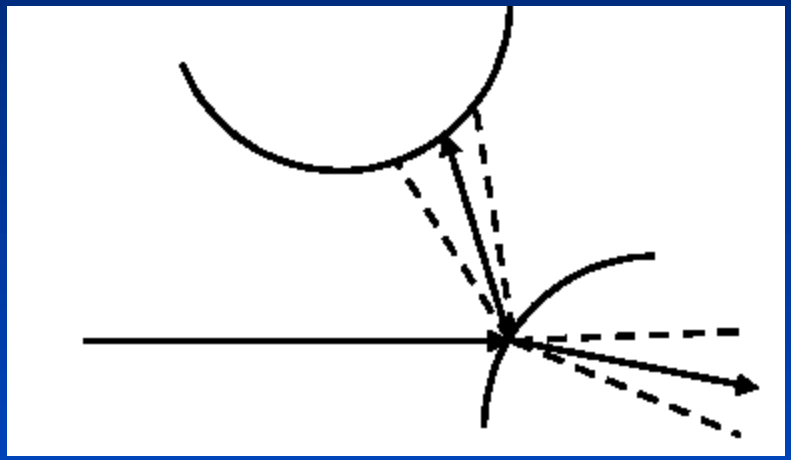
- SEADS (Spatially Enumerated Auxiliary Data Structure)
- More voxels to traverse than octrees but easier to determine neighboring voxels: DDA (Digital Differential Analyzer)



- Tradeoff resolution of grid and number of intersection tests w/ objects
- Can combine w/ octree: within a level use DDA

Area Sampling

- Prevent/reduce aliasing
- Efficient (coherence)
- Diffuse illumination
- Area light sources



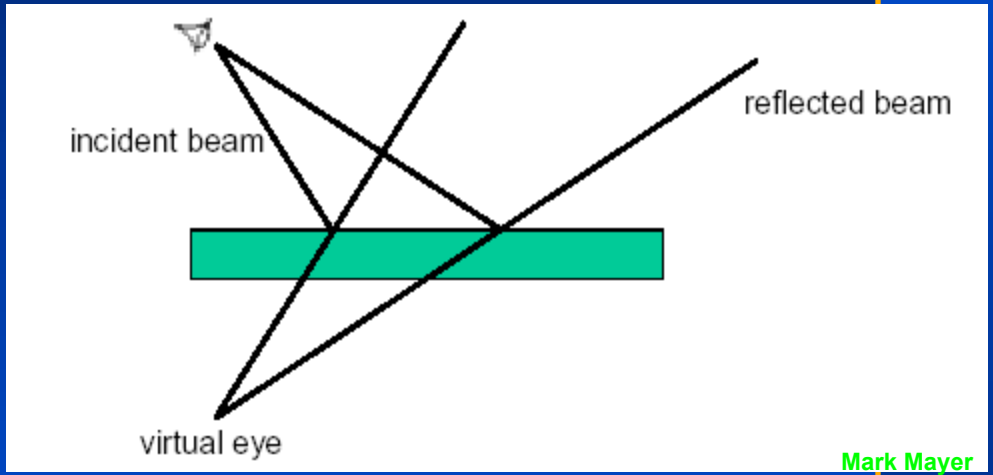
Cone Tracing

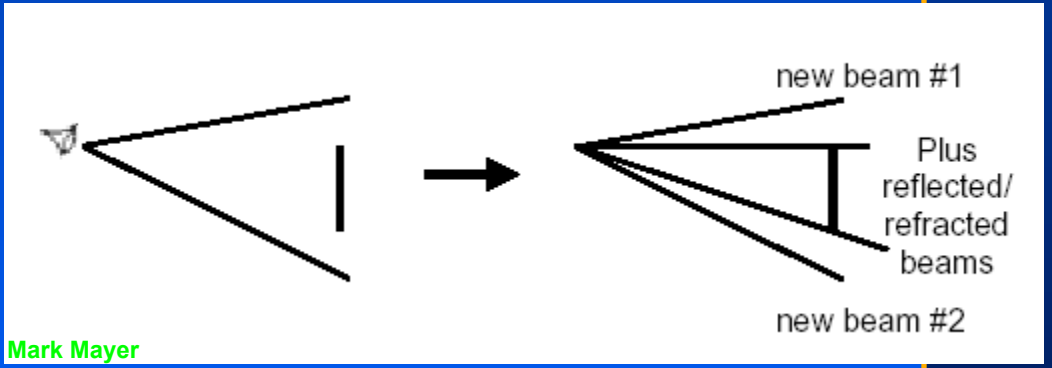
- Shoot cones from eye through pixel
- Approximate intersection area
- Spawn reflection and refraction cones (based on optics and curvature of object)

Beam Tracing

- Start w/ one large beam pyramid- the view volume
- Intersect beam w/ polygons
- Remove that part from beam
- Spawn reflection and refraction beam
- Object-precision beam tree of polygons
- Recursively rendered using polygon scan-conversion algorithm

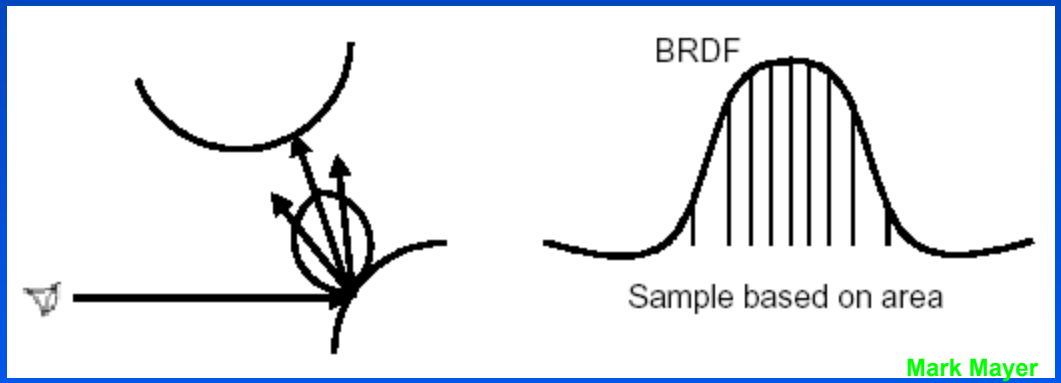
- Each polygon rendered using local illumination model
- Reflected and refracted child polygons rendered on top and blended





Distributed Ray Tracing (Monte Carlo Method)

- Integrate the BRDF using Monte Carlo integration
- Importance sampling improve efficiency



Reflection Maps (Environment Maps)

- Texture map (pre-processing step)
- Ray-trace the environment from one point (only first hit)
- Store in a texture map (e.g. spherical)
- Indexed by reflection direction
- Accurate if object is at center of projection