CS 362: Computer Graphics

Ray Tracing

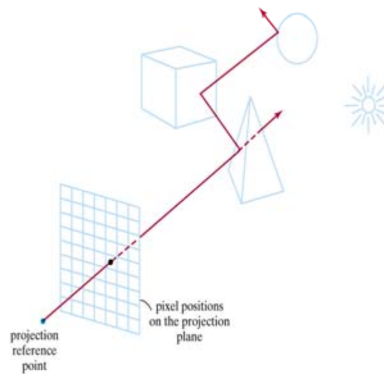
Dr. Samit Bhattacharya
Dept. of Comp. Sc. & Engg.
IIT Guwahati, Assam, India

Ray-Tracing

- Idea based on geometric optics
  - Light rays from the surface in scene emanates in all direction
  - Some of them pass through pixel positions on the projection plane
- Ray-tracing works by tracing the light path backward
  - Since number of light rays emanating from a surface is infinite, it's easier to consider a finite number of backward light rays from each pixel position

RT Setup

- Co-ordinate system for RT
  - Projection reference point on z-axis
  - Pixel positions on the xy plane (view plane)



RT Setup

- Describe geometry of the scene in this co-ordinate system
- Generate pixel rays
  - For perspective view, rays should originate at the projection point, passes through a pixel center and continues into the scene

RT Setup

- In the scene, a ray is branched to form various reflection and transmission paths
- Pixel intensity – accumulation of contributions of intensities along the path

Basic RT Steps

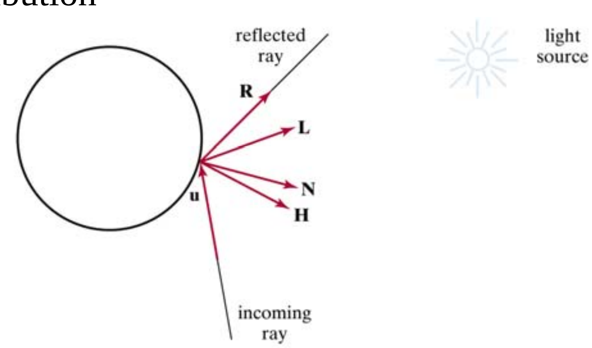
- Generate a pixel ray
- Detection of visible surface
  - Process the list of surfaces in the scene to check for any ray-surface intersection
  - When there is an intersection, calculate distance between pixel-intersection point
  - The smallest distance among all the pixel-surface intersection points indicates visible surface
- The corresponding ray is called *primary ray*

Basic RT Steps

- Once primary ray detects visible surface, perform reflection and refraction
  - Reflect a ray along the specular-reflection direction from the surface intersection point off the visible surface
  - For transparent surfaces, also send a ray through the surface in the refraction direction
- The reflection and refraction rays are called *secondary rays*
  - Repeat the process for secondary rays

RT & Illumination Model

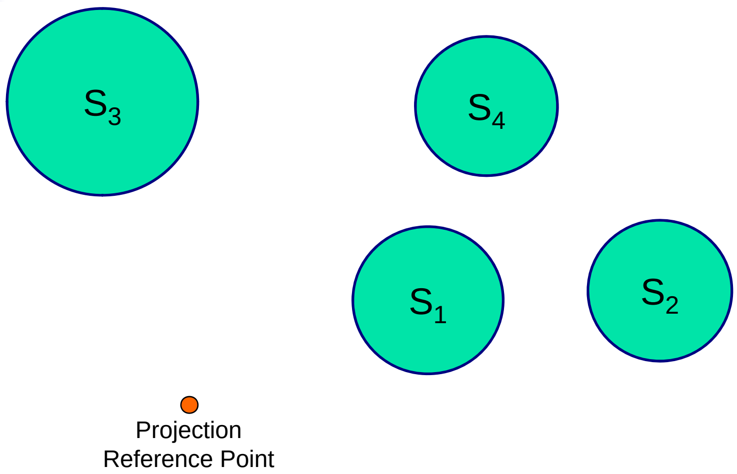
- At each intersection point, illumination model is invoked to determine the surface intensity contribution



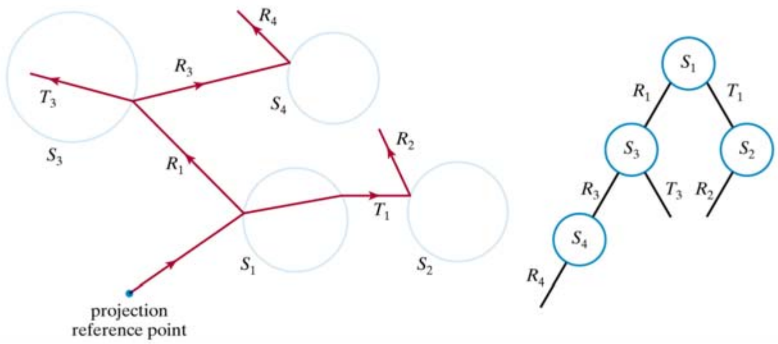
Ray-Tracing Tree

- As the rays ricochet around the scene, each intersected surface is added to a binary ray-tracing tree
  - Left branches represent reflection path
  - Right branches represent refraction path
- Tree nodes store intensity at that surface
- Helps keep track of all contributions to a pixel intensity

Tree Example



Tree Example (cont…)



Terminating Ray-Tracing

- A ray-tracing path is terminated when any one of the following conditions is satisfied
  - The ray intersects no surfaces
  - The ray intersects a light source that is not a reflecting surface
  - The tree has been generated to its maximum allowable depth
    - Maximum depth can be set beforehand (as user-option or determined by the available storage)

Pixel Intensity Calculation

- After the ray-tracing tree has been completed for a pixel, the intensity contributions are accumulated
  - Start at the terminal nodes (bottom) of the tree
- The surface intensity at each node is
  - Attenuated by the distance from the parent surface
  - Added to the intensity of the parent surface

Pixel Intensity Calculation

- The sum of the attenuated intensities at the root node is assigned to the pixel
  - If ray intersect no surface, set pixel intensity as the background intensity
  - If ray intersects a non-reflecting light source, set pixel intensity as the intensity of the light source

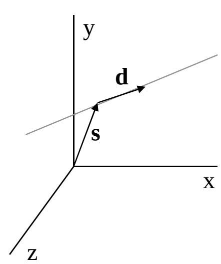
Shadow Ray

- The path from a ray-surface intersection to the light source is known as the **shadow ray**
- If any object intersects the shadow ray between the surface and the light source
  - The surface is in shadow with respect to that source

Intersection Calculation

- Need to represent ray
  - Ray is not a vector
- Represented as sum of two vectors

  $\mathbf{r}(t) = \mathbf{s} + t\mathbf{d}$ $(t>=0)$
- **s** is a vector from co-ordinate system origin to the origin point of the ray
- **d** is a vector along the direction of ray (Usually a unit vector)



Intersection Calculation

- **s** is initially set to the pixel position on the projection plane
  - Can be chosen as the projection reference point for perspective projection
- Unit vector **d** is calculated initially from the pixel position and the projection reference point

$$\vec{d} = \frac{\vec{P}_{pix} - \vec{P}_{prp}}{|\vec{P}_{pix} - \vec{P}_{prp}|}$$

$\vec{P}_{pix}$ = pixel location vetor, $\vec{P}_{prp}$ = projection refernce point vector

  - For parallel projection, it is unit normal vector of the xy plane

Intersection Calculation

- To locate ray-surface intersection point, use the surface equation to solve for **r(t)**
  - Gives the value of t – the distance from **s** to the surface along the ray path
- At each intersection point, update **d** and **s** for the secondary rays
  - Reflection direction is specular reflection direction
  - Transmission direction along the refraction path

Intersection Calculation

- Efficient ray-surface intersection algorithms developed for commonly occurring shapes
  - Plane, sphere, spline etc…

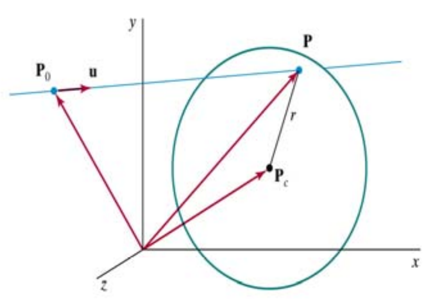
Ray-Sphere Intersection

- The simplest objects to ray trace are spheres
- Sphere equation

$$|\vec{p} - \vec{p}_c| = r$$

$\vec{p}$ = point on sphere, $\vec{p}_c$ = centre vector, r = radius

- Replace ray equation into sphere equation and square both sides



$$|\vec{s} + t\vec{d} - \vec{p}_c|^2 = r^2$$

Ray-Sphere Intersection

- After expanding, rearranging and simplifying

$$t = \frac{-B \pm \sqrt{B^2 - AC}}{A}$$

$$A = |\vec{d}\,|^2,\ B = (\vec{s} - \vec{p}_c).\vec{d},\ C = |\,\vec{s} - \vec{p}_c\,|^2 - r^2$$

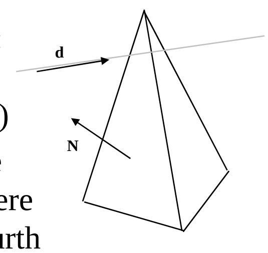
- $B^2 - AC < 0$, no intersection
- $B^2 - AC = 0$, ray (or its negative extension) touching the sphere

Ray-Sphere Intersection

- $B^2 - AC > 0$, two possible intersection points, t1 and t2
  - If both values are negative, no intersection
  - If one of them zero and the other positive, ray originates from a point on the sphere and intersects it
  - If the two values differ in sign, the ray originates from inside the sphere and intersects it
  - If both are positive, the ray intersects the sphere twice (enter and exit)-smaller value corresponds to the intersection point that is closer to the starting point of the ray

Ray-Polyhedron Intersection

- More computation intensive
  - Identify front faces of the polyhedron (faces that satisfy: **d.N** < 0, where **N** is the corresponding surface normal)
  - For each front faces, solve the plane equation: **N.P** = -*D*, where **N** = (A, B, C) and *D* is the fourth plane parameter
  - **P** is both on the plane and the ray path if: **N.(s**+t**d**) **= -***D*



Ray-Polyhedron Intersection

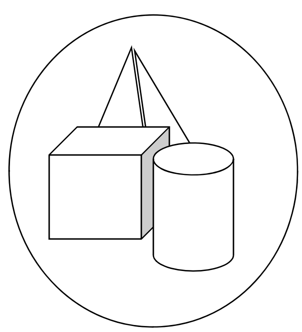
- Solving for t: $t = -\frac{D + \vec{N}.\vec{s}}{\vec{N}.\vec{d}}$
- It gives a point on the plane containing the surface
- Need to see if the point is inside the polygon surface
  - Perform inside-outside test
- The smallest t for an intersected polygonal face identifies the intersection position for the polyhedron
  - If no inside point is found, the ray does not intersect the polyhedron

Reducing Intersection Calculation

- Most work is spent testing ray-object intersections (about 95% computation time) – need speed up techniques
  - Bounding volumes
    - Hierarchical bounding volume
  - Spatial subdivision-another speed-up technique

Bounding Volumes

- Enclose group of adjacent objects within a bounding volume
  - Sphere, box



Bounding Volumes

- Test for ray-bounding volume intersection
  - If no intersection, remove entire group of objects from further intersection tests
  - Else, check for intersection points for each surface in the bounding volume
- Hierarchical bounding volume
  - Create hierarchy of bounding volumes
  - Start intersection test from the top of the hierarchy (the outermost volume)

Space Subdivision Method

- Enclose the entire scene within a cube
- Successively subdivide the cube into regions (cell), till each cell contains no-more than a pre-defined number of surfaces
  - Space subdivision of the cube can be stored in an octree or a binary-partition tree

Space Subdivision Method

- Subdivision can be
  - Uniform: divide cube into eight equal sized octants in each step
  - Adaptive: divide only those regions that contain surfaces

Space Subdivision Method

- Each cell of the cube has a list of surfaces that it contains
- First find out intersection point of ray-outermost cube
  - Determines the first cell to be checked for subsequent tests

Space Subdivision Method

- Trace rays through the individual cells
  - Check only those cells that contain surfaces
- The first surface intersected is the visible surface for that ray

Space Subdivision Method

- Given unit ray direction **d** and ray entry point to a cell P_{in}, the potential exit faces are those that satisfy: $\mathbf{d}.\mathbf{N}_k > 0$
  - $\mathbf{N}_k$ unit surface normal for face k
- Exit position can be obtained from the ray equation: $\mathbf{P}_{out,k} = \mathbf{P}_{in} + t_k\mathbf{d}$
  - t_k is the distance along the ray from P_{in} to $P_{out,k}$
- Substitute ray equation into plane equation for each cell face: $\mathbf{N}_k.\,\mathbf{P}_{out,k} = -D_k$

Space Subdivision Method

- The ray distance to each candidate exit face is given by

$$t_k = -\frac{D_k + \vec{N}_k . \vec{P}_{in}}{\vec{N}_k . \vec{d}}$$

- The smallest t_k identifies the exit face
- Assume the cell faces aligned parallel to the Cartesian-coordinate planes
  - Simplifies calculation

Anti-aliasing in RT

- RT essentially depicts continuous scene by taking discrete samples
  - Suffers from aliasing
- Two basic anti-aliasing techniques employed in RT
  - Super-sampling
  - Adaptive sampling

Super-sampling

- Each pixel divided into sub-pixels
- Separate ray sent through the centers of each sub-pixel
- Pixel color = average of the color values returned by the sub-pixel rays

Adaptive Super-sampling

- Send one ray through pixel center and four additional rays through its corners
  - If the five rays return similar color, pixel will not be subdivided
  - Otherwise subdivide pixel in 2×2 sub-pixel grid
- Repeat for each sub-pixel
- Terminate when a pre-set level of subdivision is reached

RT Examples