Introduction to Rendering

Light vs. Rendering
Scan Line Algorithm

- Assume for each line of screen, we have scan-lines for all polygons intersecting that line
- For each polygon, keep track of extents of scan line
- Whenever the x-extents of two scan lines overlap, determine ordering of two polygons
Scan Line Algorithm

Scan Line Algorithm
Depth ("Z") Buffer

- Maintain a separate buffer storing the closest “z” value for each pixel
- Only draw pixel if depth value is closer than stored “z” value
  - Update buffer with closest depth value

Depth ("Z") Buffer

- Advantages
  - Simple to implement
  - Allows for a streaming approach to polygon drawing
- Disadvantages
  - Requires extra storage
  - Overdraws
**Painter’s Algorithm**

- Sort polygons by distance from viewer
- Draw from back to front

**Painter’s Example**

Sort by depth:
- Green rect
- Red circle
- Blue tri
Painter’s Algorithm

1. Sort all objects’ $z_{\text{min}}$ and $z_{\text{max}}$

2. If an object is uninterrupted (its $z_{\text{min}}$ and $z_{\text{max}}$ are adjacent in the sorted list), it is fine
Painter’s Algorithm

- Sometimes there is NO ordering that produces correct results!!!

The splitting step
- Find a plane to split polygon by so that each new polygon is entirely in front of or entirely behind the other
- If polygons intersect, split each polygon by the other

After splitting, resort the list to remove overlap
Painter’s Algorithm - Summary

• Advantages
  - Simple algorithm for ordering polygons

• Disadvantages
  - Sorting criteria difficult to produce
  - Redraws same pixel many times
  - Sorting can be expensive

Hidden Surfaces
Backface Culling

view direction $v$

Backface Culling
Backface Culling

\[ n \cdot v < 0, \text{ draw polygon} \]

view direction \( v \)
Backface Culling

\[ n \cdot v \geq 0, \text{ cull polygon} \]
Backface Culling

Counter clockwise orientation, draw polygon

Clock-wise orientation, cull polygon
Backface Culling

• Advantages
  - Improves rendering speed by removing roughly half of polygons from scan conversion

• Disadvantages
  - Assumes closed surface with consistently oriented polygons

Ray Tracing
Ray Tracing

• Provides rendering method with
  - Refraction/Transparent surfaces
  - Reflective surfaces
  - Shadows

Essential Information for Ray Tracing

• Eye point
• Screen position/orientation
• Objects
  - Material properties
  - Reflection/Refraction coefficients
  - Index of refraction
• Light sources
Basic Ray Tracing

• For each pixel
  - Intersect ray from eye through pixel with all objects in scene
  - Find closest (positive) intersection to eye
  - Compute lighting at intersection point
**Defining E-ray**

Define ray by parametric line with parameter \( t \),

\[ R(t) = R_o + R_d \cdot t \text{ for all values } t > 0 \]

- \( R_o \) = Origin of the ray (at eye position)
- \( R_d \) = Direction of ray, unit vector pointing from eye to pixel on image plane (different for each pixel)

---

**Intersecting Objects**

1. Find intersection of ray nearest object in scene (Eye ray)

- Objects in scene could be:
  - Polygons
  - Curved Surfaces
  - Spheres - easiest to compute...

- Ray-sphere intersection:
  - Define object, such as a sphere:
  
  \[ (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - S_r^2 = 0 \]

  with \( S_r \) as radius and \( S_c \) as center
Ray- Sphere Intersection

1. Find value of t for ray-sphere intersection

Substitute the ray equation:

\[
\begin{align*}
  x &= x_0 + x_d \cdot t \\
  y &= y_0 + y_d \cdot t \\
  z &= z_0 + z_d \cdot t
\end{align*}
\]

Into sphere equation as:

\[
(x_0 + x_d \cdot t - x_c)^2 + (y_0 + y_d \cdot t - y_c)^2 + (z_0 + z_d \cdot t - z_c)^2 - r^2 = 0
\]

*Note, quadratic in terms of t*

Ray- Sphere Intersection

Actual intersection point:

\[
\begin{align*}
  x_i &= x_0 + x_d \cdot t_i \\
  y_i &= y_0 + y_d \cdot t_i \\
  z_i &= z_0 + z_d \cdot t_i
\end{align*}
\]

Compute normal (unit length) at intersection:

\[
R_n = \left[ \frac{(x_i - x_c)}{S_r}, \frac{(y_i - y_c)}{S_r}, \frac{(z_i - z_c)}{S_r} \right]
\]

*Note, if ray is inside the sphere, negate to point at ray*
Ray/Plane Intersection

• Plane is defined by \([A,B,C,D]\) as \(Ax + By + Cz + D = 0\)
• \(A^2 + B^2 + C^2 = 1\)
• Normal vector \(N = [A,B,C]\)

Ray/Plane Intersection

• Substitute the ray equation into the plane:
  \[A(x_0 + x_d t) + B(y_0 + y_d t) + C(z_0 + z_d t) + D = 0\]

• Solving for \(t\):
  \[t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(AxD + ByD + CzD)}\]
  Or \(t = -(N\cdot R0 + D)/(N\cdot Rd)\)

• Note, the normal, \(N\), should face the ray, if not reverse it.
  If \(N \cdot Rd > 0\) then \(N = -[A,B,C]\)
Ray/Plane Intersection

- Using $t$, find exact point of intersection:
  $$(x_i, y_i, z_i) = (x_0 + x_d*t, y_0 + y_d*t, z_0 + z_d*t)$$

- Use $(x_i, y_i, z_i)$ and $N$ to compute the illumination

Ray/Polygon Intersection

- Polygon is defined by set of point vertices

- Planar polygon lies in a plane and we can compute:
  $$A x + B y + C z + D = 0$$
  for any vertex

- Do ray/plane intersection

- If intersection, see if ray passes through polygon
Ray/Polygon Intersection

• If intersection, must still see if ray passes through polygon

\[ (x_i, y_i, z_i) \]

\[ \hat{z} \]

\[ \hat{x} \]

\[ \hat{y} \]

This example:

Z (or C) is principal component of \( \mathbf{N} \),
normal so project on to xy-plane

Use scanline test on x-y plane
Ray/Triangle Intersection

In the case when we have a triangle... we can be a bit more specialized in our intersection calculation.

\[ p = a + \beta (b-a) + \gamma (c-a) \]

Also, we are in the triangle iff:
\[ \beta > 0 \]
\[ \gamma > 0 \]
\[ and \quad \beta + \gamma < 1 \]

Barycentric coordinates for triangles
Ray-tracing surfaces

Given what we already know...

Break the surface into polygons (or triangles!)

Raytrace the polygons

To determine the normal:

Find the tangent in u and v

\[ N = Tu \times Tv \]
Combining with shading

• Normal is determined by Ray-sphere
• Angle of incidence = angle of reflection

Combining with shading

• Angle $\phi$ is between eye-ray and reflected
Combining with shading

For each light source and each color component, the Phong model adds to the illumination as:

\[ I' += k_d L_d \cos\theta + k_s L_s \cos\phi + k_a L_a \]

Shadow ray lets us determine if the point on the sphere is in shadowed by the particular light

If shadowed, do not compute diffuse and specular (only ambient)

---

Shadow Rays

Send a shadow ray to each light source to see if it is in shadow or not

We treat shadow rays the same as eye rays except:
- They begin at the point of intersection
  \[ R_0 = [x_i, y_i, z_i]^T \]
- They are directed toward the light source (normalize vector to find \( R_d \))
Shadows
Shadows

For each pixel $j$
  Determine closest intersection with objects in the scene
  If no hits, color pixel $j$ background color
  Else
    Illumination $j$ for $= 0$;
    For each light $k$
      Compute shadow ray intersection
      If hit, illumination $j +=$ ambient $k$
      Else find illum $k$ and add to illum $j$
    end light
  end hit
end pixel

Algorithm framework
Recursive Ray Tracing

• For each pixel
  - Intersect ray from eye through pixel with all objects in scene
  - Find closest (positive) intersection to eye
  - Compute lighting at intersection point
  - Recur for reflected and refracted rays (if necessary)
Recursive Ray Tracing

\[ I = I_{\text{direct}} + \gamma_e I_{\text{reflected}} + \gamma_a I_{\text{refracted}} \]

Truncate at finite depth!
Reflection

- Mirror-like/Shiny objects

\[ R = 2(V \cdot N)N - V \]
Recursive Ray Tracing

• Recur for reflective/transparent objects
Refraction

• Bending of light caused by different speeds of light in different medium
• Each (semi-)transparent object has an index of refraction $c_i$
• Use Snell’s law to find refracted vector

Snell’s Law

\[
\frac{c_1}{c_2} = \frac{\sin(\theta_1)}{\sin(\theta_2)}
\]
Recursive Ray Tracing

Optimizations

• Lots of rays to cast!
• Ray-Surface intersections are expensive

• Associate with each object
  - Bounding box in 3-space
• If ray doesn’t intersect box, then ray doesn’t intersect object
Parallel Processing

• Ray tracing is a trivially parallel algorithm!
  - Cast rays in parallel
  - Cast reflection, refraction, shadow rays in parallel
  - Calculate ray/surface intersections independently in parallel

Distributed Ray Tracing

Distribute rays - rays add noise or jitter to account for:
Soft shadows, motion blur, more…
Ray Tracing: Results

Limitations of Ray Tracing

• Only considers totally specular interactions
  - rays either reflect perfectly or refract perfectly
• Ray traced scenes don't show “color bleed”