Start Up
- Any questions from last time?
- Go over Phong shading?
- Or intersection algorithms?

Ray/Plane Intersection
Ray is defined by \( R(t) = R_o + R_d \cdot t \) where \( t \geq 0 \)
\( R_o = \) Origin of ray at \( (x_o, y_o, z_o) \)
\( R_d = \) Direction of ray \([x_d, y_d, z_d]\) (unit vector)

Plane is defined by \([A, B, C, D]\)
\( A x + B y + C z + D = 0 \) for a point in the plane
Normal Vector, \( N = [A, B, C] \) (unit vector)
\( A^2 + B^2 + C^2 = 1 \)
\( D = -N \cdot P_0 \) \( (P_0 \) - point in plane)
**What Can Happen?**

- \( N \cdot R_d = 0 \)
- \( N \cdot R_d > 0 \)

**Ray/Plane Summary**

Intersection point:

\[(x_i, y_i, z_i) = (x_0 + x_d t_i, y_0 + y_d t_i, z_0 + z_d t_i)\]

1. Calculate \( N \cdot R_d \) and compare it to zero.
2. Calculate \( t_i \) and compare it to zero.
3. Compute intersection point.
4. Flip normal if \( N \cdot R_d \) is positive

**Ray-Parallelepiped Intersection**

- Axis-aligned
- From \((X_1, Y_1, Z_1)\) to \((X_2, Y_2, Z_2)\)
- Ray \( P(t) = R_o + R_d t \)

**Naïve ray-box Intersection**

- Use 6 plane equations
- Compute all 6 intersection
- Check that points are inside box
- \( Ax + By + Cz + D \leq 0 \)

**Factoring out computation**

- Pairs of planes have the same normal
- Normals have only one non-zero component
- Do computations one dimension at a time
- Maintain \( t_{\text{near}} \) and \( t_{\text{far}} \) (closest and farthest so far)

**Test if parallel**

- If \( R_d x = 0 \), then ray is parallel
- If \( R_d x < X_1 \) or \( R_d x > x_2 \) return false
If not parallel

- Calculate intersection distance t1 and t2
  - t1 = (X1 - Rx)/Rdx
  - t2 = (X2 - Rx)/Rdx

Test 1

- Maintain tnear and tfar
  - If t1 > t2, swap
  - if t1 > tnear, tnear = t1
  - if t2 < tfar, tfar = t2
  - If tnear > tfar, box is missed

Test 2

- If tfar < 0, box is behind

Algorithm recap

- Do for all 3 axes
  - Calculate intersection distance t1 and t2
  - Maintain tnear and tfar
  - If tnear > tfar, box is missed; Done
  - If tfar < 0, box is behind; Done
  - If box survived tests, return intersection at tnear
  - If tnear is negative, return tfar

Homogeneous Coordinates

- Add an extra dimension
  - In 2D, we use 3 x 3 matrices
  - In 3D, we use 4 x 4 matrices
- Each point has an extra value, w

\[
\begin{pmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{pmatrix} =
\begin{pmatrix}
a & b & c & d \\
e & f & g & h \\
i & j & k & l \\
m & n & o & p
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix}
\]

\[
p' = M p
\]
Homogeneous Coordinates

Most of the time \( w = 1 \), and we can ignore it

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Translate \((t_x, t_y, t_z)\)

Why bother with the extra dimension? Because now translations can be encoded in the matrix!

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  1 & 0 & 0 & t_x \\
  0 & 1 & 0 & t_y \\
  0 & 0 & 1 & t_z \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Scale \((s_x, s_y, s_z)\)

Isotropic (uniform) scaling: \( s_x = s_y = s_z \)

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  s_x & 0 & 0 & 0 \\
  0 & s_y & 0 & 0 \\
  0 & 0 & s_z & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

You only have to implement uniform scaling

Rotation

About \( z \) axis

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  \cos \phi & -\sin \phi & 0 \\
  \sin \phi & \cos \phi & 0 \\
  0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]

Rotation

About \((k_x, k_y, k_z)\), an arbitrary unit vector (Rodrigues Formula)

\[
\begin{pmatrix}
  x' \\
  y' \\
  z'
\end{pmatrix} =
\begin{pmatrix}
  k_k(1-c)+c & k_k(1-c)k_s & k_k(1-c)+k_s \\
  k_k(1-c)+k_s & k_k(1-c)+c & k_k(1-c)-k_s \\
  k_k(1-c)-k_s & k_k(1-c)-k_s & k_k(1-c)+c
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]

where \( c = \cos \theta \) & \( s = \sin \theta \)
How are transforms combined?

Scale then Translate

Use matrix multiplication: \( p' = T(Sp) = (TS)p \)

\[
TS = \begin{bmatrix}
1 & 0 & 3 & 2 & 0 & 0 \\
0 & 1 & 1 & 0 & 2 & 0 \\
0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

\[
2 & 0 & 3 \\
0 & 2 & 1 \\
0 & 0 & 1
\]

Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate: \( p' = T(Sp) = TS \ p \)

Translate then Scale: \( p' = S(Tp) = ST \ p \)

Transformations in Ray Tracing

Transformations in Modeling

- Position objects in a scene
- Change the shape of objects
- Create multiple copies of objects
- Projection for virtual cameras
- Animations

Scene Description

- Camera
- Lights
- Background
- Materials
- Objects
Simple Scene Description File

Camera {
    center 0 0 10
    direction 0 0 -1
    up 0 1 0
}

Lights {
    numLights 1
    DirectionalLight {
        direction -0.5 -0.5 -1
        color 1 1 1
    }
    Background { color 0.2 0.0 0.6 }
}

Materials {
    numMaterials <n>
    <MATERIALS>
}

Group {
    numObjects <n>
    <OBJECTS>
}

Hierarchical Models

- Logical organization of scene

Simple Example with Groups

Group {
    numObjects 3
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        numObjects 2
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
        }
        Group {
            Box { <BOX PARAMS> }
            Sphere { <SPHERE PARAMS> }
            Sphere { <SPHERE PARAMS> }
        }
    }
    Plane { <PLANE PARAMS> }
}

Adding Materials

Group {
    numObjects 3
    Group {
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        Sphere { <SPHERE PARAMS> }
        Sphere { <SPHERE PARAMS> }
    }
}

Using Transformations

- Position the logical groupings of objects within the scene
- Transformation in group
Directed Acyclic Graph is more efficient and useful

- Leaf Node (superquadric)
- Non-Leaf Node (boolean operations)
- Transformation

Processing Model Transformations
- Goal
  - Get everything into world coordinates
  - Traverse graph/tree in depth-first order
  - Concatenate transformations
  - Can store intermediate transformations
  - Apply/associate final transformation to primitive at leaf node
- What about cylinders, superquadrics, etc.?
  - Transform ray!

Transform the Ray
- Map the ray from **World Space** to **Object Space**

\[
p_{WS} = M \cdot p_{OS}
\]
\[
p_{OS} = M^{-1} \cdot p_{WS}
\]

Transform Ray
- New origin:
  \[
  (p_{WS} + t_{WS} \cdot d_{WS}) \cdot M^{-1}
  \]
- New direction:
  \[
  (p_{WS} + t_{WS} \cdot d_{WS}) \cdot M^{-1}
  \]

Transforming Points & Directions
- Transform point
- Transform direction
- Map intersection point and normal back to world coordinates

Transform Normals
- Why? They're used for shading

object color only
Diffuse Shading
Transforming Normals

- A surface normal is a property, not a geometric entity.
- Correct normal transformation matrix: \( A = (M^{-1})^T \)

See https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/geometry/transforming-normals

How can we implement CSG?

<table>
<thead>
<tr>
<th>Points on A, Outside of B</th>
<th>Points on B, Inside of A</th>
<th>Points on B, Outside of A</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Intersection</th>
<th>Subtraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>A B</td>
<td>A B</td>
</tr>
</tbody>
</table>

Collect all the intersections

<table>
<thead>
<tr>
<th>Points on B, Inside of A</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Union</th>
<th>Intersection</th>
<th>Subtraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>A B</td>
<td>A B</td>
</tr>
</tbody>
</table>

Implementing CSG

1. Test "inside" intersections:
   - Find intersections with A, test if they are inside/outside B
   - Find intersections with B, test if they are inside/outside A

2. Overlapping intervals:
   - Find the intervals of "inside" along the ray for A and B
   - Compute union/intersection/subtraction of the intervals

Constructive Solid Geometry (CSG)

Given overlapping shapes A and B:

- Union
- Intersection
- Subtraction

Ray Tracing CSG Models

Rick Parent
Ohio State U.
Seminal paper
Ray tracing CSG models


CSG
- Form object as booleans of primitive objects
  - Primitives: sphere, cube, cylinder, cone
  - Boolean operators: union, intersection, difference
- Tree structure used to manage operations
  - Leaf nodes are primitive objects
  - Intermediate nodes specify combination operator

Ray tracing CSG models
- Intersect ray with primitives
- Produces “spans” along ray
- Perform Boolean operations on spans
- Determines intersection of evaluated model
- Calculate normal at intersection

Possible ways for 2 spans to overlap

Union
Ray intersects union: at first intersection

\[ \text{Min}(t_{\text{min}}^a, t_{\text{min}}^b) \]

Intersection
First time in B and in C

If \((t_{\text{min}}^a < t_{\text{max}}^b)\) and \((t_{\text{max}}^a < t_{\text{min}}^b)\): \(t_{\text{min}}^a\)
Else if \((t_{\text{max}}^a < t_{\text{min}}^b)\) and \((t_{\min}^a < t_{\text{max}}^b)\): \(t_{\text{min}}^b\)
Else: none
Difference

If \( t_{B_{\text{min}}} < t_{C_{\text{min}}} \): \( t_{B_{\text{min}}} \)
Else if \( t_{C_{\text{max}}} < t_{B_{\text{max}}} \): \( t_{C_{\text{max}}} \)
Else: none

First time in B not in C

Definition: 66

Primitives

Anything that can be intersected (easily) with a ray

- Conics: solve analytically using \( R(t) \)
- Convex polyhedra
- A plane (a cutting plane is useful)

- Can be used as a modeling tool (boolean operations)
- Surface model (e.g., polyhedron) computed from CGS
- Can be used as a model representation
  - Keep tree structure and ray trace directly

Controlling the Combinations

Tree Structure

Tree Structure #1

First time in C not in B

If \( (t_{C_{\text{min}}} < t_{B_{\text{min}}}) \): \( t_{C_{\text{min}}} \)
Else if \( (t_{B_{\text{max}}} < t_{C_{\text{max}}}) \): \( t_{B_{\text{max}}} \)
Else: none

First time in B not in C

Definition: 67

Primitives

Anything that can be intersected (easily) with a ray

- Conics: solve analytically using \( R(t) \)
- Convex polyhedra
- A plane (a cutting plane is useful)

- Can be used as a modeling tool (boolean operations)
- Surface model (e.g., polyhedron) computed from CGS
- Can be used as a model representation
  - Keep tree structure and ray trace directly

Controlling the Combinations

Tree Structure

Tree Structure #1
Tree Structure

- Intersect ray with leaf nodes (primitive objects)
- Combine intersection spans according to intermediate nodes
  - union
  - intersection
  - difference
- Might create multiple spans

Union of Spans

Intersection of Spans

Difference of Spans
Normals of CSG intersections

Normal of some surface (or its negation)

Union or intersection: positive normal of intersected surface

Difference normals

- Intersection is one of:
  - $t_{\min}$ of positive object – normal of surface
  - $t_{\max}$ of negative object – negated normal

Add transformations to tree

http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/csg.html

Bounding Volumes

Construction
- Use bounding volumes at leaf nodes
- Union bounding volumes at interior nodes

Traversal
- Top-down
- Test bounding volume at interior

Example

http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/csg.html

Example
Wrap Up

- Discuss status/problems/issues with next programming assignment