Intersection Testing
Chapter 16

Department of Computer Engineering
Chalmers University of Technology
What for?

- A tool needed for the graphics people all the time...

- Very important components:
  - Need to make them fast!

- Finding if (and where) a ray hits an object
  - Picking
    - Ray tracing and global illumination

- For speed-up techniques

- Collision detection (treated in a later lecture)
Example

Midtown Madness 3, DICE
Some basic geometrical primitives

- Ray:
- Sphere:
- Box
  - Axis-aligned (AABB)
  - Oriented (OBB)
- \( k \)-DOP
Four different techniques

- Analytical
- Geometrical
- Separating axis theorem (SAT)
- Dynamic tests

Given these, one can derive many tests quite easily
  - However, often tricks are needed to make them fast
Analytical: Ray/sphere test

- Sphere center: $c$, and radius $r$
- Ray: $r(t) = o + td$
- Sphere formula: $\|p-c\| = r$
- Replace $p$ by $r(t)$, and square it:

\[
(r(t) - c) \cdot (r(t) - c) - r^2 = 0
\]

\[
(o + td - c) \cdot (o + td - c) - r^2 = 0
\]

\[
(td + (o - c)) \cdot (td + (o - c)) - r^2 = 0
\]

\[
(d \cdot d)t^2 + 2((o - c) \cdot d)t + (o - c) \cdot (o - c) - r^2 = 0
\]

\[
t^2 + 2((o - c) \cdot d)t + (o - c) \cdot (o - c) - r^2 = 0 \quad \| d \| = 1
\]
Analytical, continued

\[ t^2 + 2((\mathbf{o} - \mathbf{c}) \cdot \mathbf{d})t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 = 0 \]

- Be a little smart…
  
  \begin{align*}
  (\mathbf{o} - \mathbf{c}) \cdot \mathbf{d} &> 0 \ ? \\
  (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 &< 0 \ ?
  \end{align*}

- Such tests are called "rejection tests"

- Other shapes:
  
  \begin{align*}
  p_x^2 + p_y^2 &= r^2 \\
  (p_x / a)^2 + (p_y / b)^2 + (p_z / c)^2 &= 1 \\
  (p_x / a)^2 + (p_y / b)^2 - p_z &= 0
  \end{align*}
**Geometrical: Ray/Box Intersection**

- Boxes and spheres often used as bounding volumes
- A slab is the volume between two parallel planes:
- A box is the logical intersection of three slabs (2 in 2D):
Geometrical: Ray/Box Intersection (2)

- Intersect the 2 planes of each slab with the ray
  - Keep max of $t^{\text{min}}$ and min of $t^{\text{max}}$
  - If $t^{\text{min}} < t^{\text{max}}$ then we got an intersection
  - Special case when ray parallel to slab
Two convex polyhedrons, A and B, are disjoint if any of the following axes separate the objects:

- An axis orthogonal to a face of A
- An axis orthogonal to a face of B
- An axis formed from the cross product of one edge from each of A and B
SAT example: Triangle/Box

- E.g an axis-aligned box and a triangle
- 1) test the axes that are orthogonal to the faces of the box
- That is, x, y, and z
Triangle/Box with SAT (2)

- Assume that they overlapped on x,y,z
- Must continue testing
- 2) Axis orthogonal to face of triangle
Triangle/Box with SAT (3)

- If still no separating axis has been found…
- **3) Test axis:** \( t = e_{\text{box}} \times e_{\text{triangle}} \)
- **Example:**
  - **x-axis from box:** \( e_{\text{box}} = (1,0,0) \)
  - \( e_{\text{triangle}} = v_1 - v_0 \)
- Test all such combinations
- If there is at least one separating axis, then the objects do not collide
- Else they do overlap
Rules of Thumb for Intersection Testing

- **Acceptance and rejection test**
  - Try them early on to make a fast exit

- **Postpone expensive calculations if possible**

- **Use dimension reduction**
  - E.g. 3 one-dimensional tests instead of one complex 3D test, or 2D instead of 3D

- **Share computations between objects if possible**

- **Timing!**
Another analytical example: Ray/Triangle in detail

- Ray: \( \mathbf{r}(t) = \mathbf{o} + td \)
- Triangle vertices: \( \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2 \)
- A point in the triangle:
  - \( \mathbf{t}(u,v) = \mathbf{v}_0 + u(\mathbf{v}_1 - \mathbf{v}_0) + v(\mathbf{v}_2 - \mathbf{v}_0) = 
  = (1-u-v)\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2 \quad [u,v] \geq 0, \quad u + v \leq 1 \)
- Set \( \mathbf{t}(u,v) = \mathbf{r}(t) \), and solve!

\[
\begin{pmatrix}
-d & \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 \\
\end{pmatrix}
\begin{pmatrix}
t \\ u \\ v
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{o} - \mathbf{v}_0 \\
\end{pmatrix}
\]
Ray/Triangle (2)

\[ \mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0 \quad \mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0 \quad \mathbf{s} = \mathbf{o} - \mathbf{v}_0 \]

- Solve with Cramer's rule:

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

\[
A \mathbf{x} = \mathbf{b}
\]

\[
\begin{pmatrix}
  -\mathbf{d} & \mathbf{e}_1 & \mathbf{e}_2 \\
  \mathbf{t} \\
  \mathbf{u} \\
  \mathbf{v}
\end{pmatrix} =
\begin{pmatrix}
  \mathbf{o} - \mathbf{v}_0
\end{pmatrix}
\]

\[
\begin{pmatrix}
  \mathbf{t} \\
  \mathbf{u} \\
  \mathbf{v}
\end{pmatrix} =
\frac{1}{\det(-\mathbf{d}, \mathbf{e}_1, \mathbf{e}_2)}
\begin{pmatrix}
  \det(\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2) \\
  \det(-\mathbf{d}, \mathbf{s}, \mathbf{e}_2) \\
  \det(-\mathbf{d}, \mathbf{e}_1, \mathbf{s})
\end{pmatrix}
\]
**Ray/Triangle (2)**

- Solve with Cramer’s rule:

\[
\begin{pmatrix}
  t \\
  u \\
  v
\end{pmatrix}
= \frac{1}{\det(-d, e_1, e_2)}
\begin{pmatrix}
  \det(s, e_1, e_2) \\
  \det(-d, s, e_2) \\
  \det(-d, e_1, s)
\end{pmatrix}
\]

- **Use this fact:** \( \det(a, b, c) = (a \times b) \cdot c = -(a \times c) \cdot b \)

- **Share factors to speed up computations**
Be smart!
  - Compute as little as possible. Then test

Examples:
  \[
  p = d \times e_2  \\
  a = p \cdot e_1  \\
  f = 1 / a  \\
  u = f(p \cdot s)
  \]

Compute

Then test valid bounds

if (u<0 or u>1) exit;
Point/Plane

Insert a point \( \mathbf{x} \) into plane equation:

\[
f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d
\]

- \( f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d = 0 \) for \( \mathbf{x} \)'s on the plane
- \( f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d < 0 \) for \( \mathbf{x} \)'s on one side of the plane
- \( f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d > 0 \) for \( \mathbf{x} \)'s on the other side

Plane: \( \pi : \mathbf{n} \cdot \mathbf{p} + d = 0 \)

\[
\mathbf{n} \cdot \mathbf{x}_2 = \| \mathbf{x}_2 \| \cos \gamma < 0
\]

\[
\mathbf{n} \cdot \mathbf{x}_1 = \| \mathbf{x}_1 \| \cos \phi > 0
\]
Sphere/Plane
Box/Plane

- **Sphere**: compute
  \[ f(c) = n \cdot c + d \]
- \( f(c) \) is the signed distance (\( n \) normalized)
- \( |f(c)| > r \) \quad no collision
- \( |f(c)| = r \) \quad sphere touches the plane
- \( |f(c)| < r \) \quad sphere intersects plane

- **Box**: insert all 8 corners
- If all \( f \)'s have the same sign, then all points are on the same side, and no collision

**Plane**: \( \pi : n \cdot p + d = 0 \)
**Sphere**: \( \mathbf{c} \quad r \)
**AABB**: \( \mathbf{b}^{\text{min}} \quad \mathbf{b}^{\text{max}} \)
The smart way (shown in 2D)

Find the two vertices that have the most positive and most negative value when tested against the plane

Need only test the red points

OBB almost as easy. Just first project \( n \) on OBB’s axes – see p: 757
Ray/Polygon: very briefly

- Intersect ray with polygon plane
- Project from 3D to 2D
- How?
  - Find $\max(|n_x|, |n_y|, |n_z|)$
  - Skip that coordinate!
- Then, count crossing in 2D
Volume/Volume tests

- Used in collision detection
- Sphere/sphere
  - Compute squared distance between sphere centers, and compare to \((r_1 + r_2)^2\)
- Axis-Aligned Bounding Box (AABB)
  - Test in 1D for x, y, and z
- Oriented Bounding boxes
  - Use SAT [details in book]
View frustum testing

- View frustum is 6 planes:
  - Near, far, right, left, top,
- Create planes from projection matrix
  - Let all positive half spaces be outside frustum
- Sphere/frustum common approach:
  - Test sphere against each of the 6 frustum planes:
    - If outside the plane => no intersection
    - If intersecting the plane or inside, continue
  - If not outside after all six planes, then conservatively consider sphere as inside or intersecting
- Example follows…
View frustum testing example

- Not exact test, but not incorrect
  - A sphere that is reported to be inside, can be outside
  - Not vice versa
- Similarly for boxes
Dynamic Intersection Testing

- Testing is often done every rendered frame, i.e., at discrete time intervals.
- Therefore, you can get "quantum effects".
- Dynamic testing deals with this.
- Is more expensive.
- Deals with a time interval: time between two frames.
Dynamic intersection testing
Sphere/Plane

● No collision occur:
  – If they are on the same side of the plane \((s_c s_e > 0)\)
    ● and: \(|s_c| > r\) and \(|s_e| > r\)

● Otherwise, sphere can move \(|s_c| - r\)

● Time of collision:
  \[ t_{cd} = n + \frac{s_c - r}{s_c - s_e} \]
  \(s_e\) is signed distance

● Response: reflect \(v\) around \(n\), and move \((1 - t_{cd})r\) (\(r=refl\ vector\))
Dynamic Separating Axis Theorem

- SAT: tests one axis at a time for overlap

- Same with DSAT, but:
  - Use a relative system where B is fixed
  - i.e., compute A’s relative motion to B.
  - Need to adjust A’s projection on the axis so that the interval moves on the axis as well

- Need to test same axes as with SAT

- Same criteria for overlap/disjoint:
  - If no overlap on axis => disjoint
  - If overlap on all axes => objects overlap
Dynamic Sweep-and-Prune

Exercises

Create a function (by writing code on paper) that tests for intersection between:

- two spheres
- a ray and a sphere
- view frustum and a sphere
Scan Line Fill

Set active edges to AB and AC
For y = A.y, A.y-1,...,C.y
  If y=B.y → exchange AB with BC
  Compute xstart and xend.
  Interpolate color, depth, texcoords etc for points (xstart,y) and (xend,y)
For x = xstart, xstart+1, ...,xend
  Compute color, depth etc for (x,y) using interpolation.
Using Interpolation

$C_1 C_2 C_3$ specified by `glColor` or by vertex shading
$C_4$ determined by interpolating between $C_1$ and $C_3$
$C_5$ determined by interpolating between $C_2$ and $C_3$
Interpolate between $C_4$ and $C_5$ along span
Rasterizing a Triangle

- Convex Polygons only
- Nonconvex polygons assumed to have been tessellated
- Shader results (e.g. colors) have been computed for the vertices. Depth occlusion resolved with z-buffer.
  - March across scan lines interpolating vertex shader output parameters, as input to the fragment shader.
  - Incremental work small
Flood Fill

• Fill can be done recursively if we know a seed point located inside (WHITE)
• Scan convert edges into buffer in edge/inside color (BLACK)

```c
flood_fill(int x, int y) {
    if(read_pixel(x,y) == WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
    }
}
```
What you need to know

- **Analytic test:**
  - Be able to compute ray vs sphere or other formula
  - ray vs triangle
- **Geometrical tests**
  - Ray/box with slab-test
  - Ray/polygon (3D->2D)
  - AABB/AABB
- **Other:**
  - Point/plane
  - Sphere/plane
  - Box/plane, AABB/plane
- **SAT**
- **Know what a dynamic test is**
- **Understand floodfill**