Collision Detection

In the General Case, there are Two Types of Collision Detection

1. Discrete
2. Continuous

Detecting Collisions Between Two Objects

1. Do as many fast rejections as you can
2. Do hierarchical fast rejections
3. **Discrete**: compare all edges of Object A against all faces of Object B
4. **Continuous**: create “pseudo-edges” by connecting respective points in Object A across the time step, then compare all these pseudo-edges of Object A against all faces of Object B

Try to Simplify the Intersection Test– Break the Scene into a Grid

**Discrete**: You only have to do intersection tests against objects that live in the same grid square.
Try to Simplify the Intersection Test - A Bounding Sphere

These spheres overlap if:
\[ \text{Distance}(C_1, C_2) < R_1 + R_2 \]

To avoid the square root:
\[ \text{Distance}^2(C_1, C_2) < (R_1 + R_2)^2 \]

Try to Simplify the Intersection Test - Bounding Spheres

Try to Simplify the Intersection Test - Bounding Boxes

Quickly compare two objects by fitting each with a bounding box and then comparing the two bounding boxes.

These boxes do not overlap if:
\[ X_{\text{max}1} < X_{\text{min}2} \quad \text{||} \quad Y_{\text{max}1} < Y_{\text{min}2} \]
\[ X_{\text{max}2} < X_{\text{min}1} \quad \text{||} \quad Y_{\text{max}2} < Y_{\text{min}1} \]

These boxes do overlap if:
\[ X_{\text{max}1} > X_{\text{min}2} \quad \&\& \quad Y_{\text{max}1} > Y_{\text{min}2} \]
\[ X_{\text{max}2} > X_{\text{min}1} \quad \&\& \quad Y_{\text{max}2} > Y_{\text{min}1} \]
Try to Simplify the Intersection Test—Two Types of Bounding Boxes

**Axis-Aligned Bounding Box (AABB)**
Check for overlap by looking for overlap in just X, then just Y, then just Z.

**Arbitrary-Oriented Bounding Box (AOBB)**
This is a tighter fit around the object, but the overlap comparison is more involved.

A Hierarchy of Bounding Boxes

Discrete—You Can Easily Tell if a Point is Inside a Convex Polyhedron

Assume that all surface normals point *outwards* (usual convention).

Use the Distance-from-a-Point-to-a-Plane formula for each face of the polyhedron.

If all distances are *negative*, the point is inside the convex polyhedron.

What if the polyhedron is not convex?
Put a Convex Hull around the polyhedron and test against that.

If the point is not inside the convex hull, then it is not inside the polyhedron either. If it is inside the convex hull, then a more detailed analysis is needed.

Distance from a Point to a Plane

The equation of the plane is:

\[ (x, y, z) - (Q_x, Q_y, Q_z) \cdot (n_x, n_y, n_z) = 0 \]

which expands out to become the more familiar \( Ax + By + Cz + D = 0 \).

The distance from the point \( P \) to the plane is based on the plane equation:

\[ d = (P - Q) \cdot \hat{n} \]

The dot product is answering the question “How much of \( (P-Q) \) is in the normal direction?”. Note that this gives a signed distance. If \( d > 0 \), then \( P \) is on the same side of the plane as the normal.
Discrete and Continuous – Comparing an Edge on Object A against a Face on Object B

If point \( P \) wants to be a point in the plane, then:

\[
(P, P, P) = (Q, Q, Q) \cdot (n, n, n) = 0
\]

If we substitute the parametric expression for \( P \) into the plane equation, then the only thing we don’t know in that equation is \( t \). Knowing \( t^* \) will let us compute the \((x,y,z)\) of the actual intersection using the line equation. If \( t^* \) has a zero in the denominator, then that tells us that \( t^* = \infty \), and the line must be parallel to the plane.

This gives us the point of intersection with the infinite plane. We would now use the method covered a few slides ago to see if \( P \) lies inside the triangle in question.

Is a Point inside a Triangle?

Let:

\[
\begin{align*}
n &= (R - Q) \times (S - Q) \\
q &= (R - Q) \times (P - Q) \\
r &= (S - R) \times (P - R) \\
s &= (Q - S) \times (P - S)
\end{align*}
\]

If \((n \cdot n_q), (n \cdot n_r), (n \cdot n_s)\) are all positive, then \( P \) is inside the triangle QRS.

If Find a Discrete Interference

Do a binary search across the time step until you find the time of collision

If Find a Continuous Interference

Connect all points across time and look for the minimum \( t \) in an intersection with the boundary
Voxelization – another way to do Collision Detection?

Randy Rauwendaal