3D Computer Game Programming

Collision Detection

Collision Detection

Complicated for two reasons:

1. Geometry is typically very complex, potentially requiring expensive testing
2. Naïve solution is $O(n^2)$ time complexity, since every object can potentially collide with every other object

Collision Detection

Two basic techniques

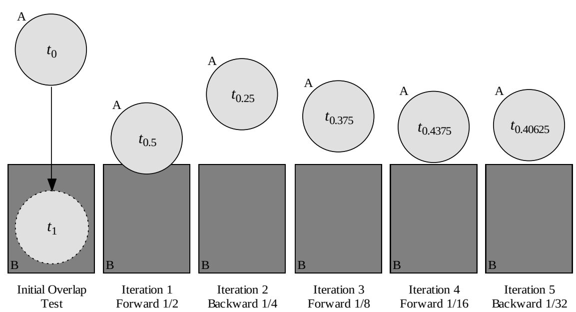
1. Overlap testing
   - Detects whether a collision has already occurred
2. Intersection testing
   - Predicts whether a collision will occur in the future

Overlap Testing

- Facts
  - Most common technique used in games
  - Exhibits more error than intersection testing
- Concept
  - For every simulation step, test every pair of objects to see if they overlap
  - Easy for simple volumes like spheres, harder for polygonal models

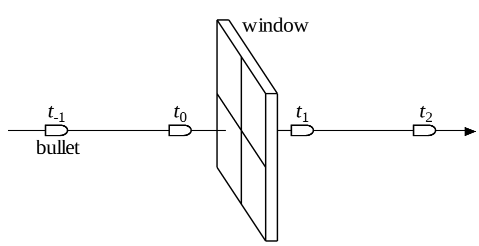
Overlap Testing: Collision Time

- Collision time calculated by moving object back in time until right before collision
  - Bisection is an effective technique



Overlap Testing: Limitations

- Fails with objects that move too fast
  - Unlikely to catch time slice during overlap
- Possible solutions
  - Design constraint on speed of objects
  - Reduce simulation step size

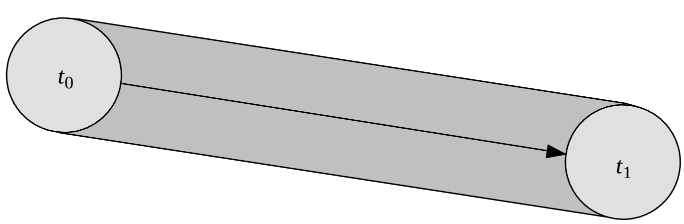


Intersection Testing

- Predict future collisions
- When predicted:
  - Move simulation to time of collision
  - Resolve collision
  - Simulate remaining time step

Intersection Testing: Swept Geometry

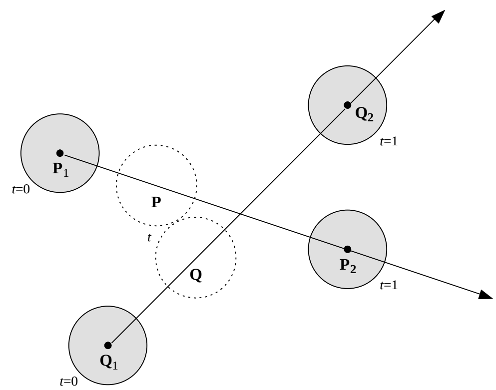
- Extrude geometry in direction of movement
- Swept sphere turns into a “capsule” shape



Intersection Testing: Sphere-Sphere Collision

$$t = \frac{-(\mathbf{A}\cdot\mathbf{B}) - \sqrt{(\mathbf{A}\cdot\mathbf{B})^2 - B^2\left(A^2 - (r_P + r_Q)^2\right)}}{B^2},$$

$$\mathbf{A} = \mathbf{P}_1 - \mathbf{Q}_1$$
$$\mathbf{B} = (\mathbf{P}_2 - \mathbf{P}_1) - (\mathbf{Q}_2 - \mathbf{Q}_1).$$



Intersection Testing: Sphere-Sphere Collision

- To see if there is a collision, check if

$$d^2 > (r_P + r_Q)^2$$

where $d^2 = A^2 - \dfrac{(\mathbf{A}\cdot\mathbf{B})^2}{B^2}$,

the smallest distance separating the centers of the two spheres

Intersection Testing: Limitations

- Issue with networked games
  - Future predictions rely on exact state of world at present time
  - Due to packet latency, current state not always coherent
- Assumes constant velocity and zero acceleration over simulation step
  - Has implications for physics model and choice of integrator

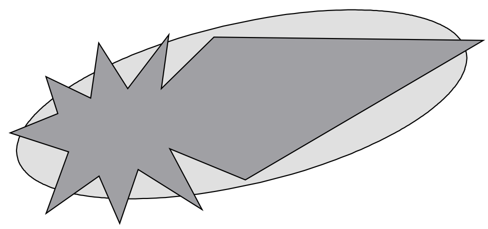
Dealing with Complexity

Two issues

1. Complex geometry must be simplified
2. Reduce number of object pair tests

Dealing with Complexity: Simplified Geometry

- Approximate complex objects with simpler geometry, like this ellipsoid

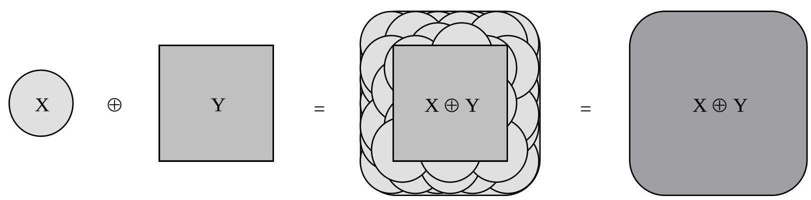


Dealing with Complexity: Minkowski Sum

- By taking the Minkowski Sum of two complex volumes and creating a new volume, overlap can be found by testing if a single point is within the new volume

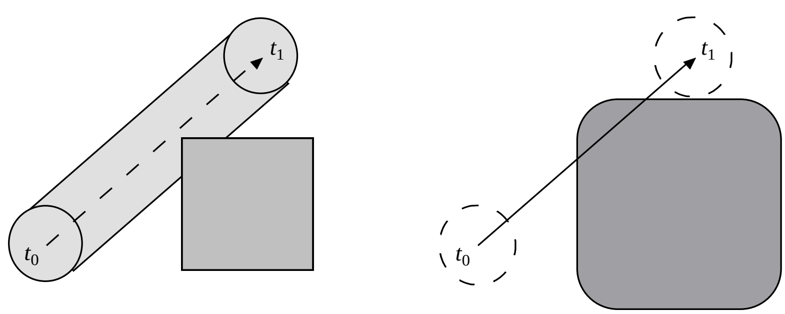
Dealing with Complexity: Minkowski Sum

$$X \oplus Y = \{A + B : A \in X \text{ and } B \in Y\}$$



Minkowski Sum of X and Y – created by sweeping the origin of X over all points Within Y

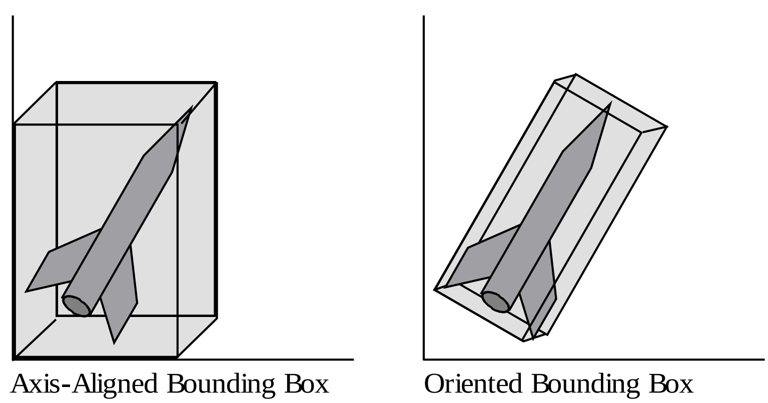
Dealing with Complexity: Minkowski Sum



Dealing with Complexity: Bounding Volumes

- Bounding volume is a simple geometric shape
  - Completely encapsulates object
  - If no collision with bounding volume, no more testing is required
- Common bounding volumes
  - Sphere
  - Box

Dealing with Complexity: Box Bounding Volumes

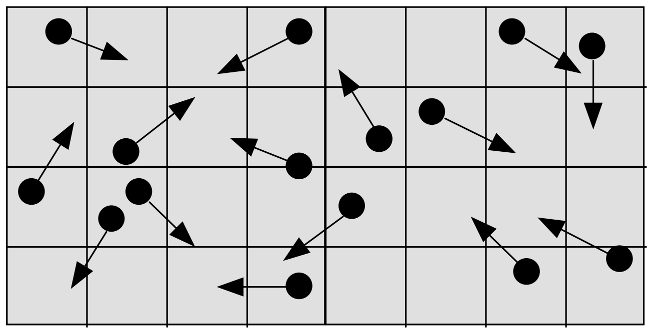


AABB – loose fit around an object in the world coordinate system but collision test against the box is simple

OBB – tighter fit but more computation for the collision test

Dealing with Complexity: Achieving O(n) Time Complexity

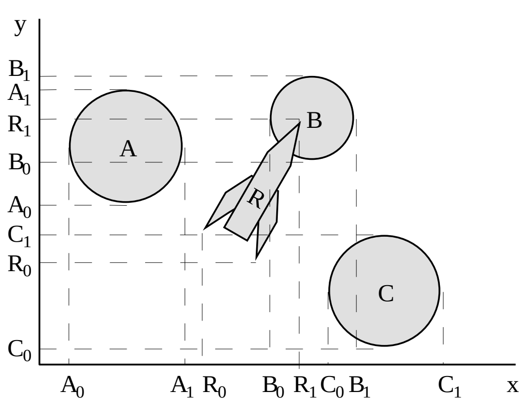
One solution is to partition space



3D Computer Game Programming

Dealing with Complexity: Achieving O(n) Time Complexity

Another solution is the plane sweep algorithm

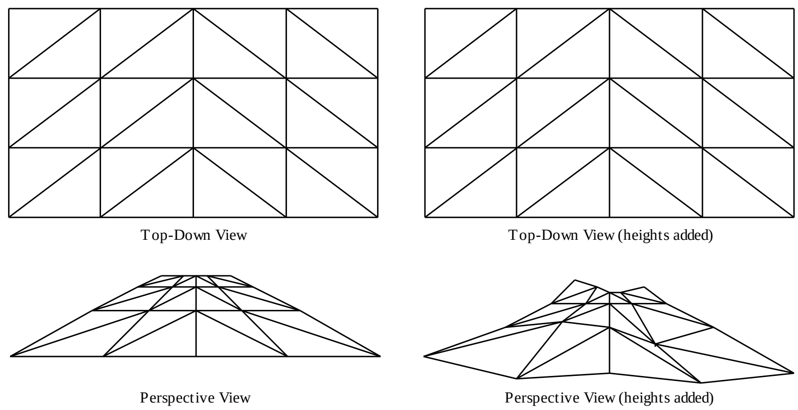


1. Record the bounds of every object on each of the three axis.
2. Perform collision detection for objects that have overlapping bounds in all axes.

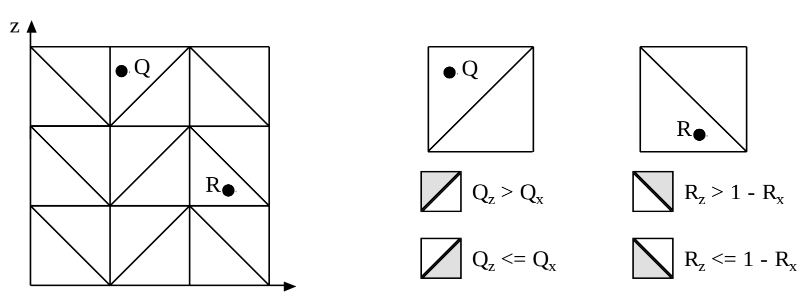
* Sorting involved.

3D Computer Game Programming

Terrain Collision Detection: Height Field Landscape



Terrain Collision Detection: Locate Triangle on Height Field



Terrain Collision Detection: Locate Point on Triangle

- Plane equation: $Ax + By + Cz + D = 0$
- *A*, *B*, *C* are the *x*, *y*, *z* components of the plane's normal vector N with one of the triangles vertices being $\mathbf{P}_0$
- Giving:

$$D = -\mathbf{N} \cdot \mathbf{P}_0$$

$$\mathbf{N}_x Q_x + \mathbf{N}_y Q_y + \mathbf{N}_z Q_z + \left(-\mathbf{N} \cdot \mathbf{P}_0\right) = 0$$

Terrain Collision Detection: Locate Point on Triangle

- The normal can be constructed by taking the cross product of two sides:

$$\mathbf{N} = \left(\mathbf{P}_1 - \mathbf{P}_0\right) \times \left(\mathbf{P}_2 - \mathbf{P}_0\right)$$

- Solve for *y* and insert the *x* and *z* components of Q, giving the final equation for point within triangle:

$$\mathbf{Q}_y = \frac{-\mathbf{N}_x \mathbf{Q}_x - \mathbf{N}_z \mathbf{Q}_z + \left(\mathbf{N} \cdot \mathbf{P}_0\right)}{\mathbf{N}_y}$$

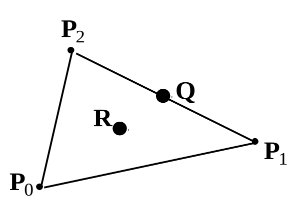
$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}.$$

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k}$$

$$\mathbf{a} \times \mathbf{b} = \mathbf{i}a_2b_3 + \mathbf{j}a_3b_1 + \mathbf{k}a_1b_2 - \mathbf{i}a_3b_2 - \mathbf{j}a_1b_3 - \mathbf{k}a_2b_1.$$

Terrain Collision Detection: Locate Point on Triangle

- Triangulated Irregular Networks (TINs)
  - Non-uniform polygonal mesh
- Barycentric Coordinates



Point = $w_0\mathbf{P}_0 + w_1\mathbf{P}_1 + w_2\mathbf{P}_2$

$\mathbf{Q} = (0)\mathbf{P}_0 + (0.5)\mathbf{P}_1 + (0.5)\mathbf{P}_2$
$\mathbf{R} = (0.33)\mathbf{P}_0 + (0.33)\mathbf{P}_1 + (0.33)\mathbf{P}_2$

Barycentric coordinates are defined by the vertices of a simplex (e.g. triangle).
Barycentric coordinates of a point p are coefficients of p in terms of the vertices.

Terrain Collision Detection: Locate Point on Triangle

- Calculate barycentric coordinates for point Q in a triangle's plane

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{1}{V_1^2 V_2^2 - (\mathbf{V}_1 \cdot \mathbf{V}_2)^2} \begin{bmatrix} V_2^2 & -\mathbf{V}_1 \cdot \mathbf{V}_2 \\ -\mathbf{V}_1 \cdot \mathbf{V}_2 & V_1^2 \end{bmatrix} \begin{bmatrix} \mathbf{S} \cdot \mathbf{V}_1 \\ \mathbf{S} \cdot \mathbf{V}_2 \end{bmatrix}$$

$$\mathbf{S} = \mathbf{Q} - \mathbf{P}_0$$
$$\mathbf{V}_1 = \mathbf{P}_1 - \mathbf{P}_0$$
$$\mathbf{V}_2 = \mathbf{P}_2 - \mathbf{P}_0$$

$$w_0 = 1 - w_1 - w_2$$

- If any of the weights (w_0, w_1, w_2) are negative, then the point Q does not lie in the triangle

Converting to barycentric coordinates

[

Given a point $\mathbf{r}$ inside a triangle it is also desirable to obtain the area coordinates λ_1, λ_2 and λ_3 at this point. We can write the barycentric expansion of vector $\mathbf{r}$ having Cartesian coordinates (x, y) in terms of the components of the triangle vertices ($\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$) as

$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$
$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

substituting $\lambda_3 = 1 - \lambda_1 - \lambda_2$ into the above gives

$$x = \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 - \lambda_2) x_3$$
$$y = \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 - \lambda_2) y_3$$

Rearranging, this is

$$\lambda_1 (x_1 - x_3) + \lambda_2 (x_2 - x_3) + x_3 - x = 0$$
$$\lambda_1 (y_1 - y_3) + \lambda_2 (y_2 - y_3) + y_3 - y = 0$$

This linear transformation may be written more succinctly as

$$\mathbf{T} \cdot \lambda = \mathbf{r} - \mathbf{r}_3$$

Where λ is the vector of area coordinates, $\mathbf{r}$ is the vector of Cartesian coordinates, and $\mathbf{T}$ is a matrix given by

$$\mathbf{T} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}$$

Now the matrix $\mathbf{T}$ is invertible, since $\mathbf{r}_1 - \mathbf{r}_3$ and $\mathbf{r}_2 - \mathbf{r}_3$ are linearly independent (if this were not the case, then $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$ would be colinear and would not form a triangle). Thus, we can rearrange the above equation to get

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \mathbf{T}^{-1} (\mathbf{r} - \mathbf{r}_3)$$

Finding the barycentric coordinates has thus been reduced to finding the inverse matrix of $\mathbf{T}$, a trivial problem in the case of 2×2 matrices.

From Wikipedia

Collision Resolution: Examples

- Two billiard balls strike
  - Calculate ball positions at time of impact
  - Impart new velocities on balls
  - Play "clinking" sound effect
- Rocket slams into wall
  - Rocket disappears
  - Explosion spawned and explosion sound effect
  - Wall charred and area damage inflicted on nearby characters
- Character walks through wall
  - Magical sound effect triggered
  - No trajectories or velocities affected