Design of Human Interface Game Software

– Collision Detection

Collision Detection

- Collision Detection
  - The computational problem of detecting the intersection of two or more objects

Collision Detection

Complicated for two reasons

1. Geometry is typically very complex, potentially requiring expensive testing
2. Naive 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

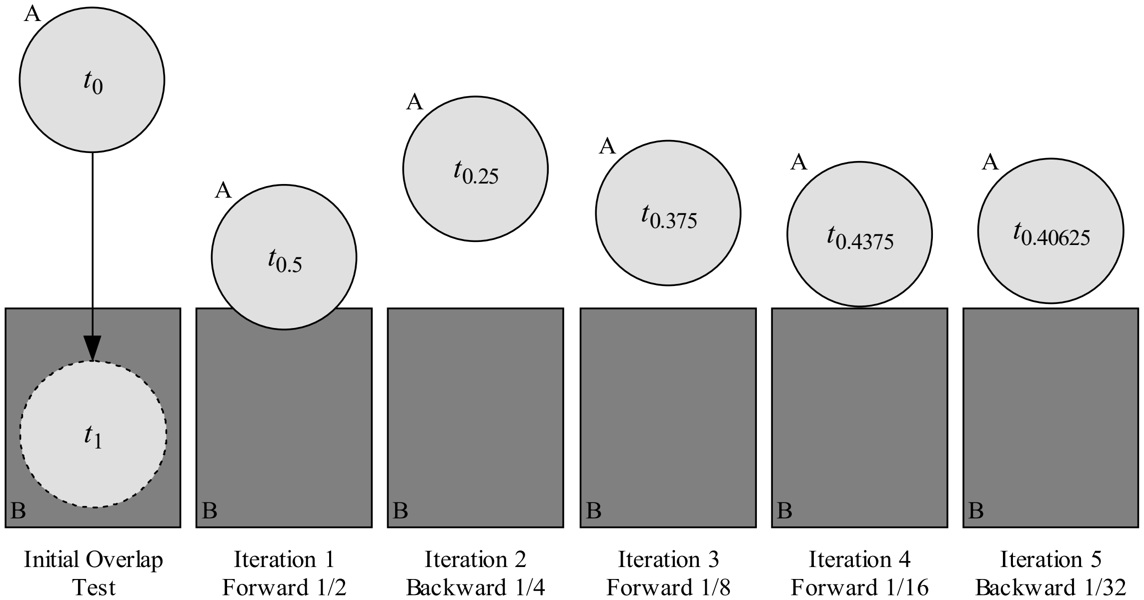
- Facts
  - Most common technique used in games
  - Exhibits more error than intersection testing
- Concept
  - For every simulation step, test every pair of objects to see if they overlap
  - Easy for simple volumes like spheres, harder for polygonal models

Overlap Testing: Useful Results

- Useful results of detected collision
  - Time collision took place
  - Collision normal vector

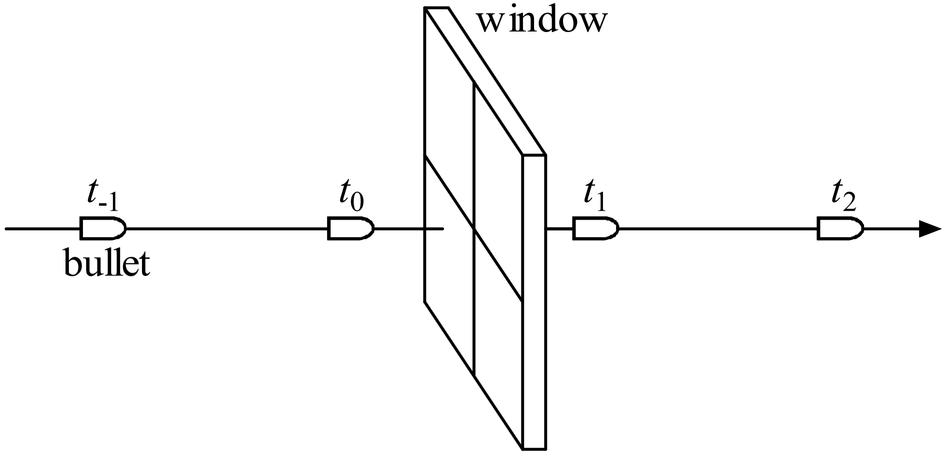
Overlap Testing: Collision Time

- 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

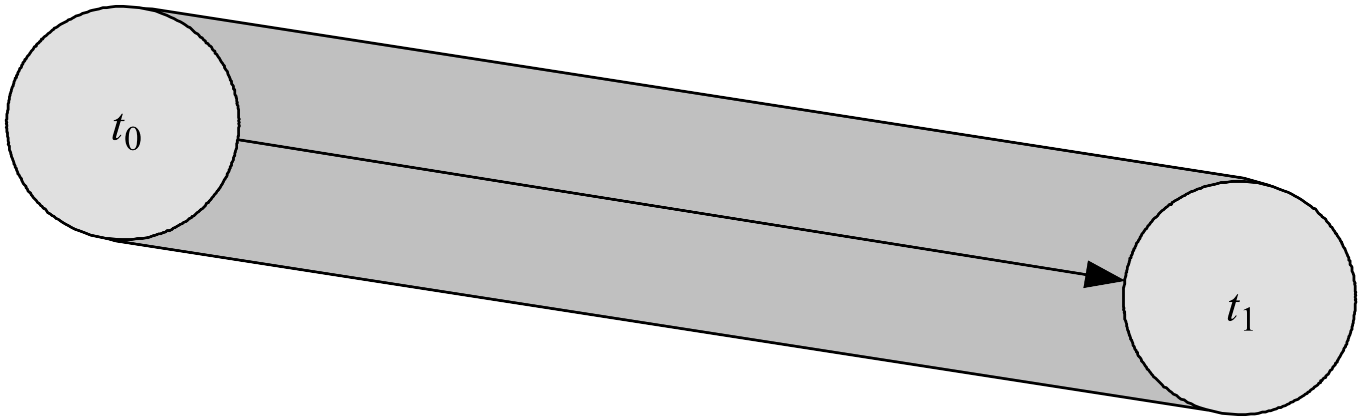


Intersection Testing

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

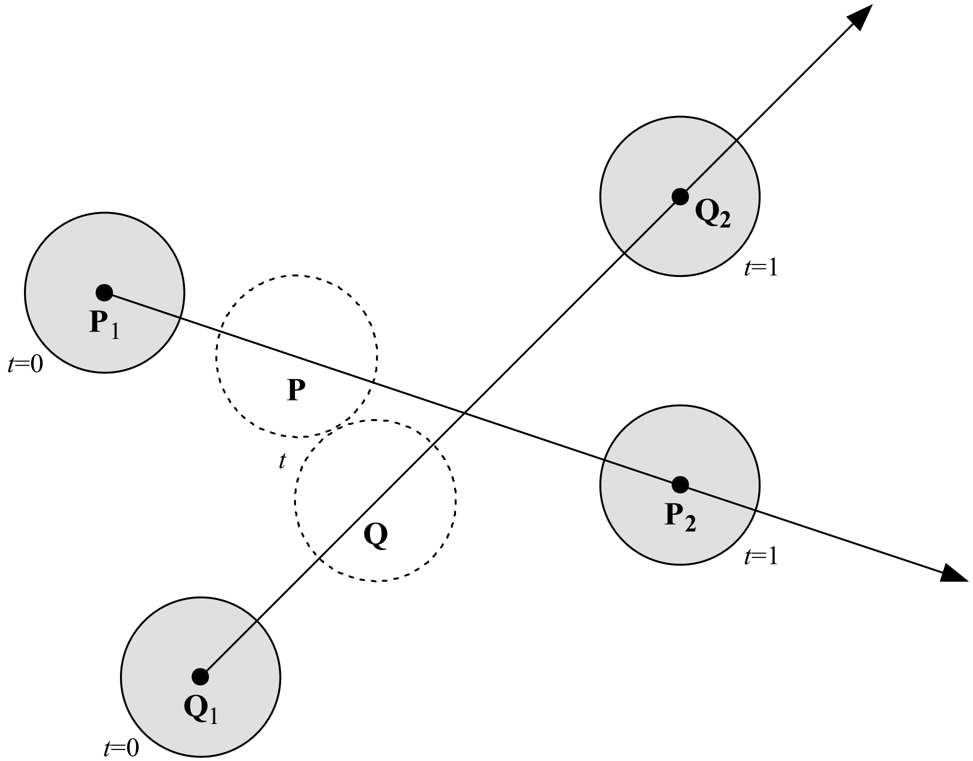
Intersection Testing: Swept Geometry

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



Intersection Testing: Sphere-Sphere Collision

$$t = \frac{-(\mathbf{A}\cdot\mathbf{B}) - \sqrt{(\mathbf{A}\cdot\mathbf{B})^2 - B^2\left(A^2 - (r_P + r_Q)^2\right)}}{B^2}, \qquad \begin{aligned}\mathbf{A} &= \mathbf{P}_1 - \mathbf{Q}_1\\ \mathbf{B} &= (\mathbf{P}_2 - \mathbf{P}_1) - (\mathbf{Q}_2 - \mathbf{Q}_1).\end{aligned}$$



Intersection Testing: Sphere-Sphere Collision

- Smallest distance ever separating two spheres:

$$d^2 = A^2 - \frac{(\mathbf{A} \cdot \mathbf{B})^2}{B^2}$$

- If $d^2 > (r_P + r_Q)^2$
there is no collision

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

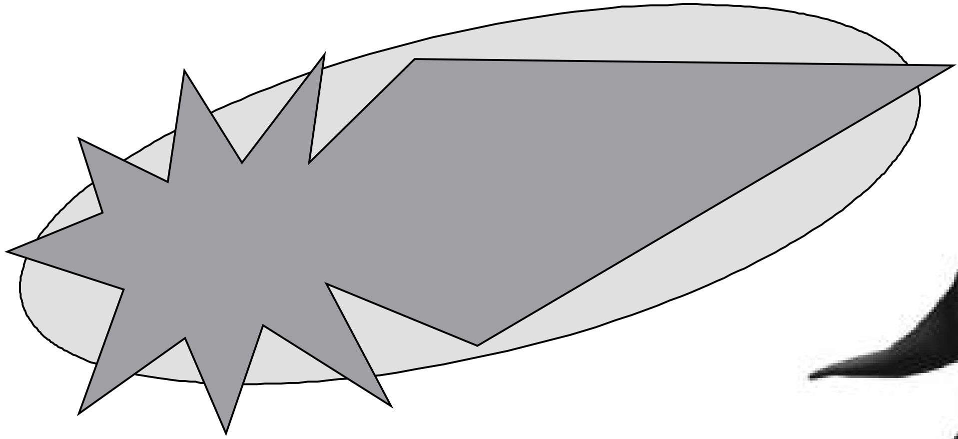
Dealing with Complexity

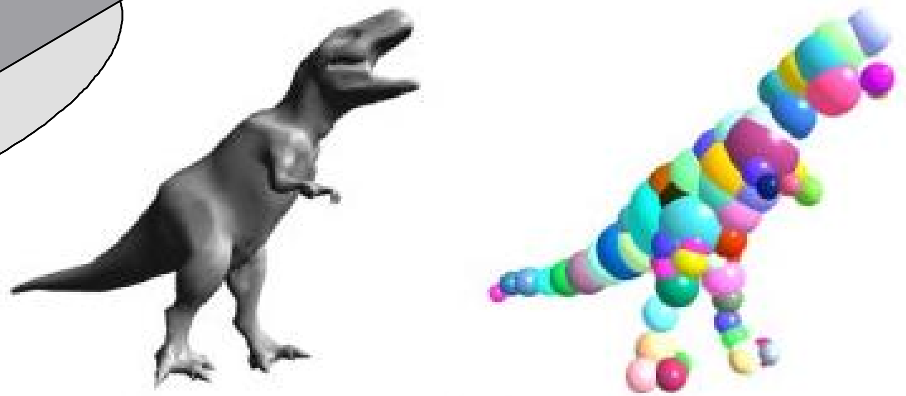
Two issues

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

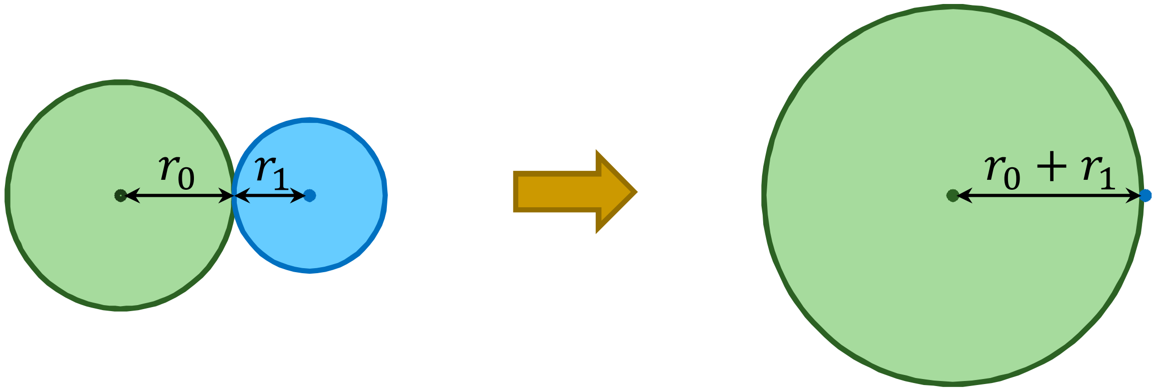




tyrannosaurus, 64 spheres

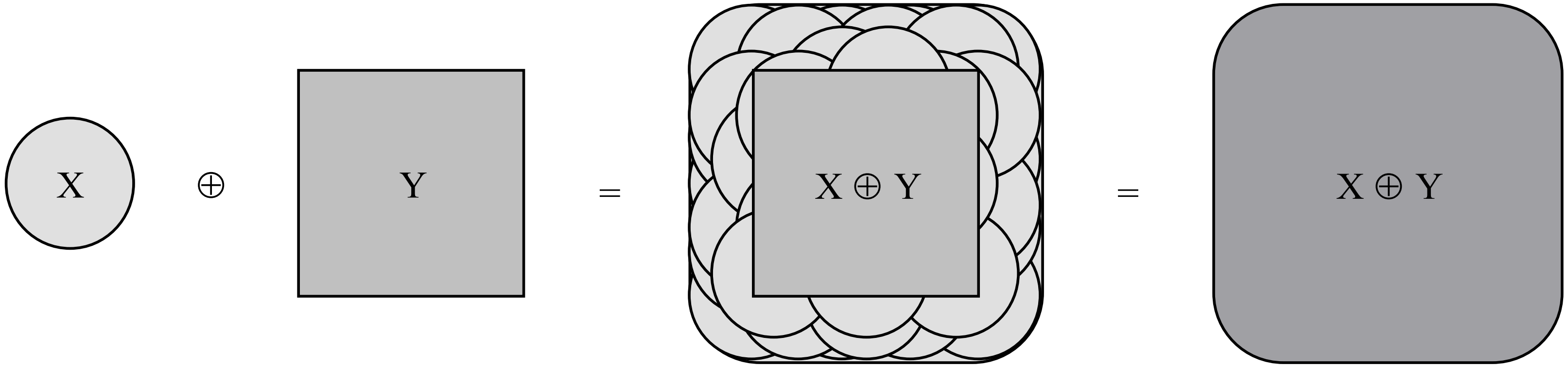
Dealing with Complexity: Minkowski Sum

- 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

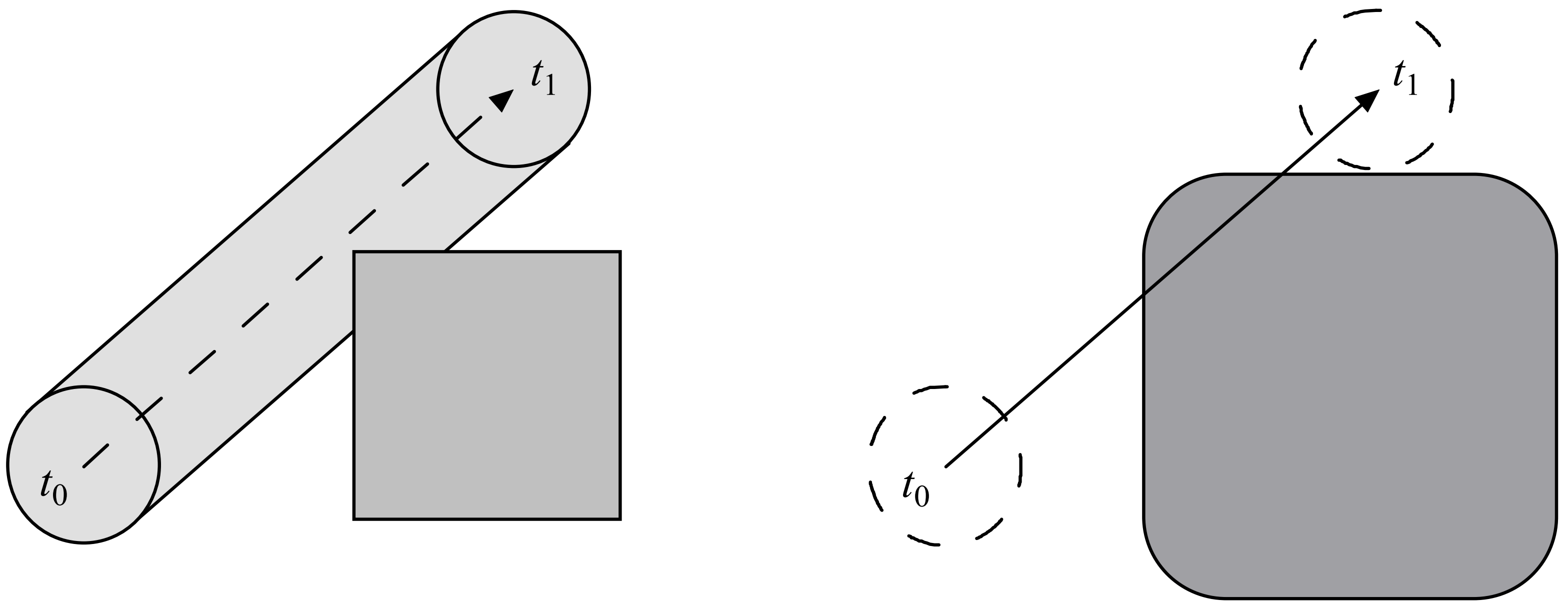


Dealing with Complexity: Minkowski Sum

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



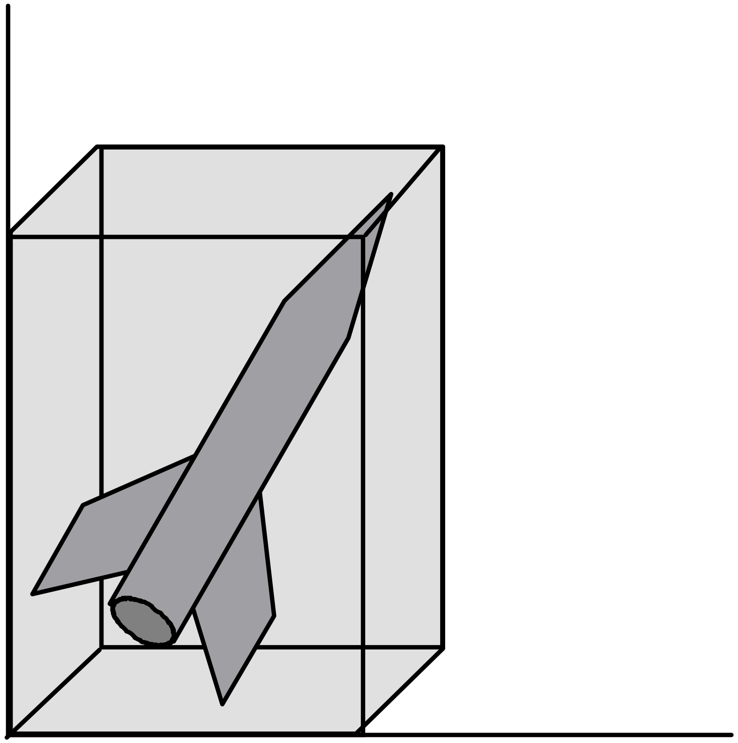
Dealing with Complexity: Minkowski Sum



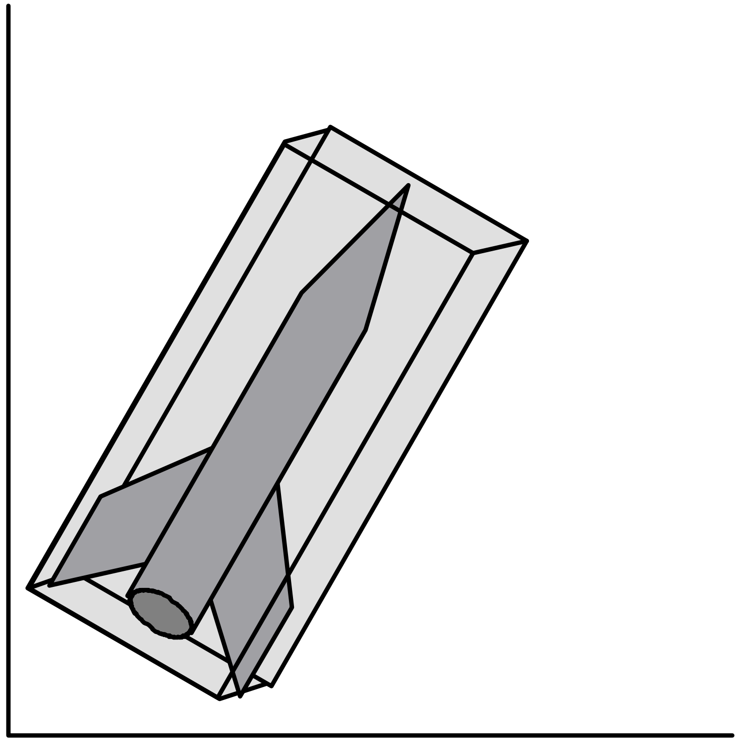
Dealing with Complexity: Bounding Volumes

- 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



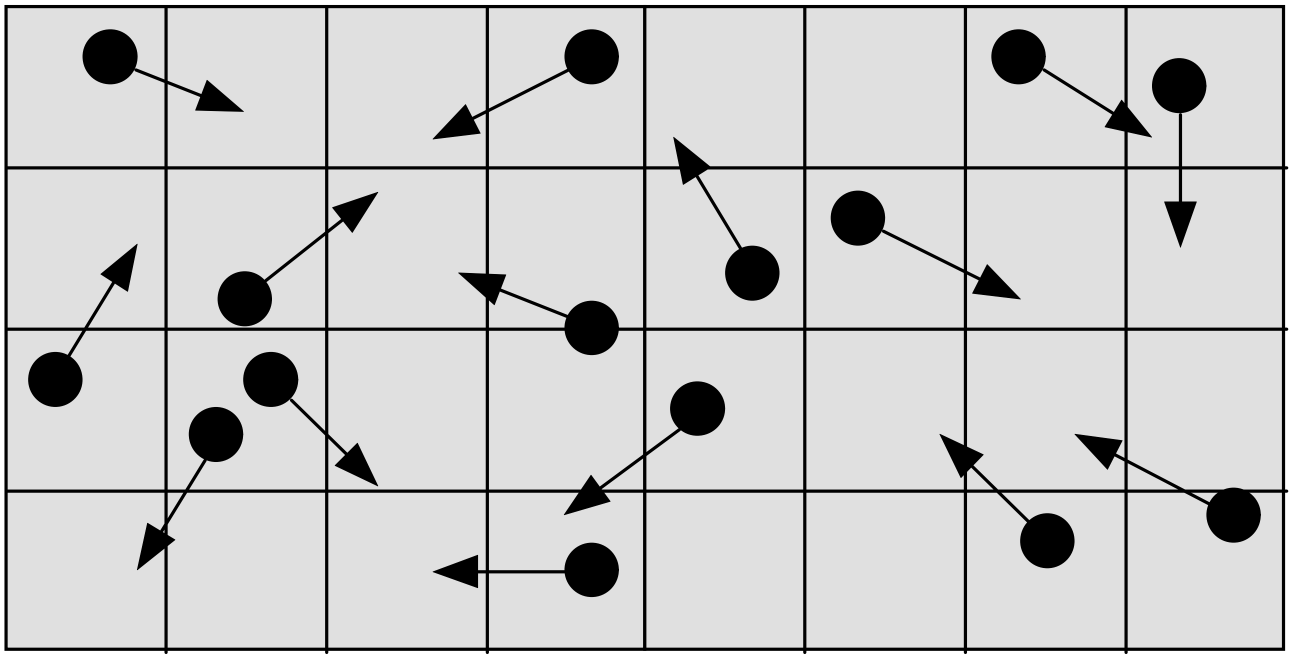
Axis-Aligned Bounding Box



Oriented Bounding Box

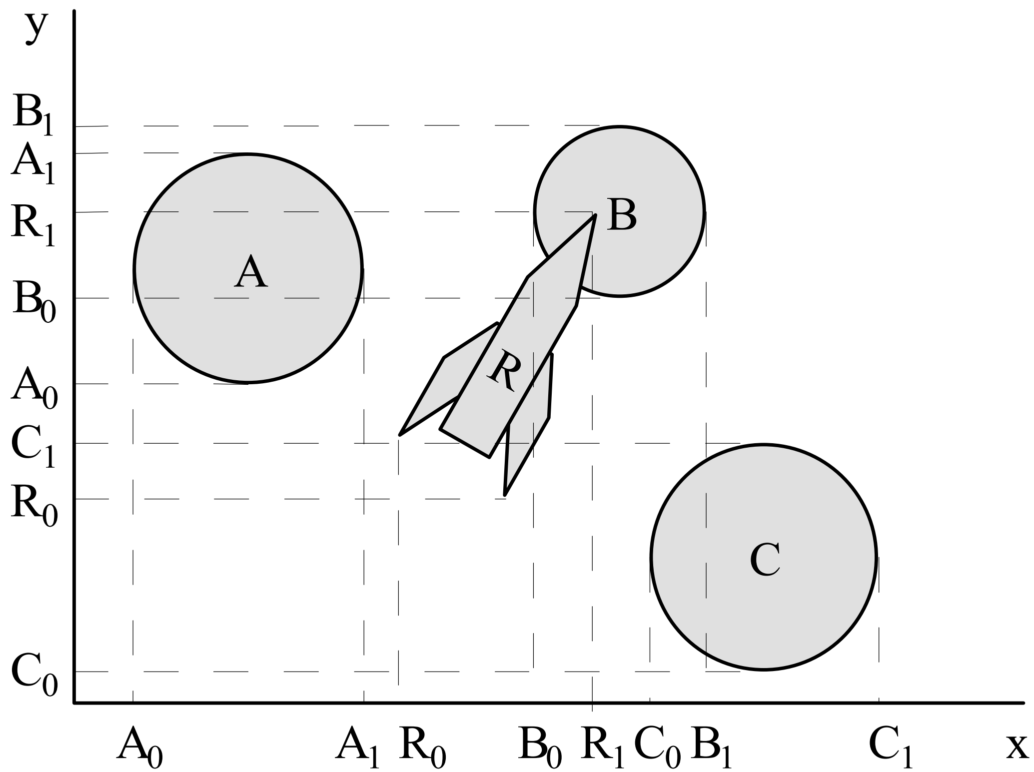
Dealing with Complexity: Achieving O(n) Time Complexity

One solution is to partition space

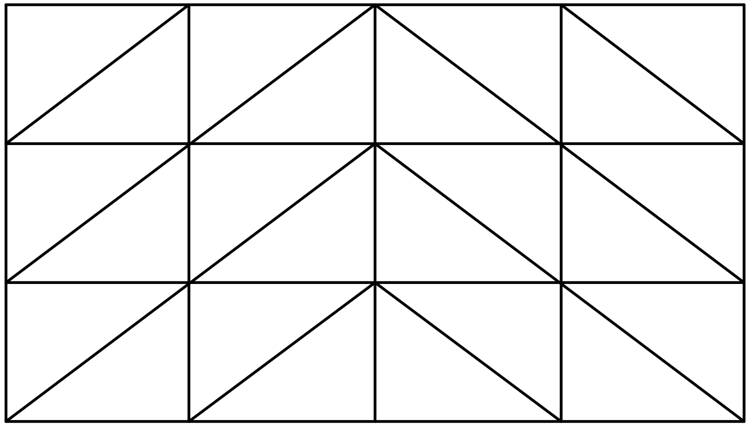


Dealing with Complexity: Achieving O(n) Time Complexity

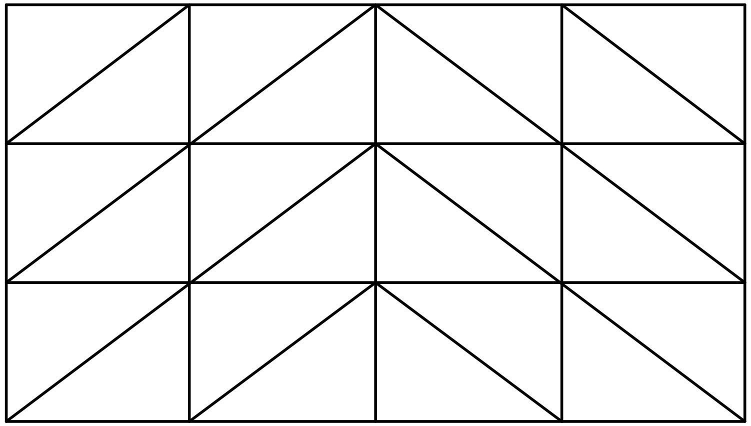
Another solution is the plane sweep algorithm



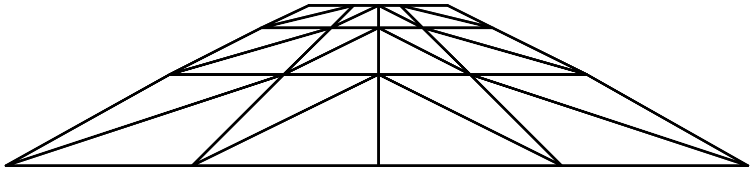
Terrain Collision Detection: Height Field Landscape



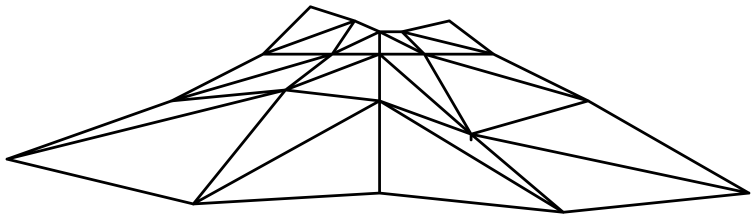
Top-Down View



Top-Down View (heights added)

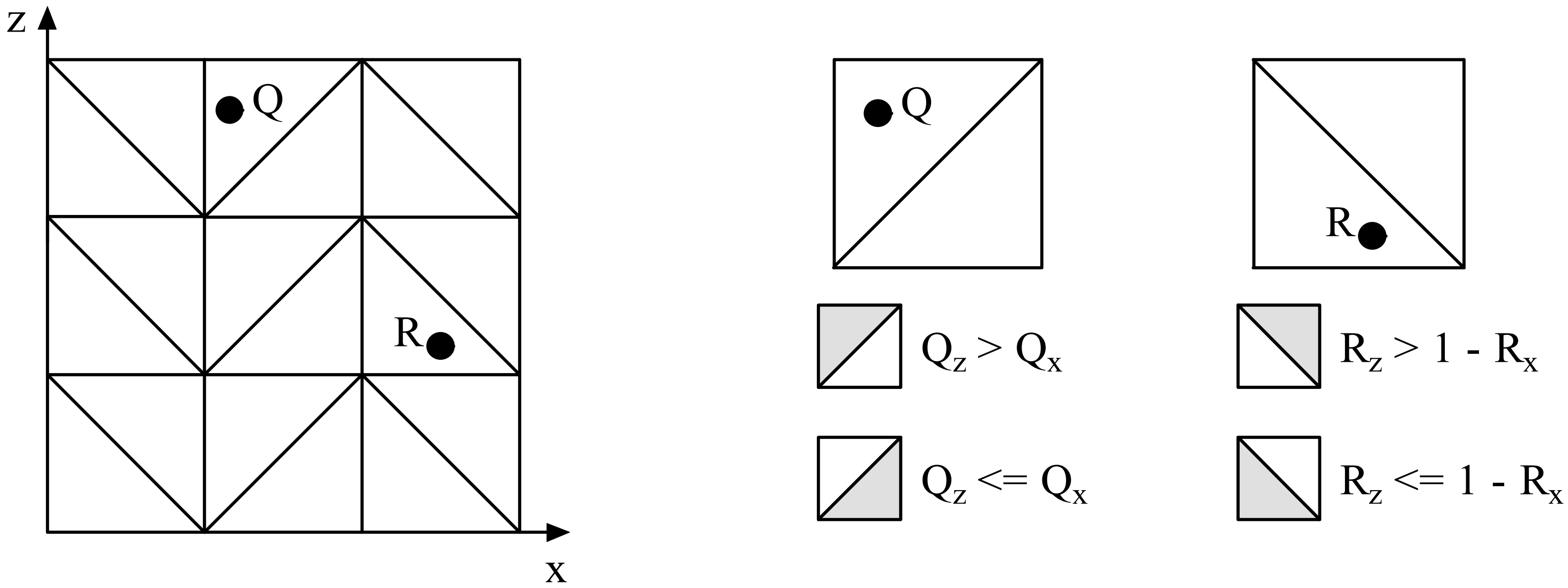


Perspective View



Perspective View (heights added)

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
- Where $D = -\mathbf{N} \cdot \mathbf{P}_0$
  with one of the triangles
  vertices being $\mathbf{P}_0$
- Giving: $\mathbf{N}_x(x) + \mathbf{N}_y(y) + \mathbf{N}_z(z) + (-\mathbf{N} \cdot \mathbf{P}_0) = 0$

Terrain Collision Detection: Locate Point on Triangle

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

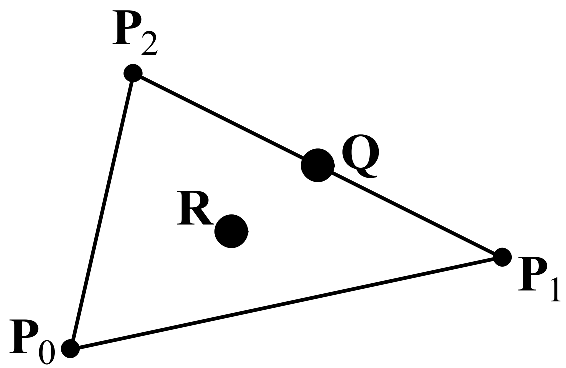
$$\mathbf{N} = (\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P}_2 - \mathbf{P}_0)$$

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

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

Terrain Collision Detection: Locate Point on Triangle

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



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

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

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

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

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

Collision Resolution: Parts

- Resolution has three parts
  1. Prologue
  2. Collision
  3. Epilogue

Collision Resolution: Prologue

- Collision known to have occurred
- Check if collision should be ignored
- Other events might be triggered
  - Sound effects
  - Send collision notification messages

Collision Resolution: Collision

- Place objects at point of impact
- Assign new velocities
  - Using physics or
  - Using some other decision logic

Collision Resolution: Epilogue

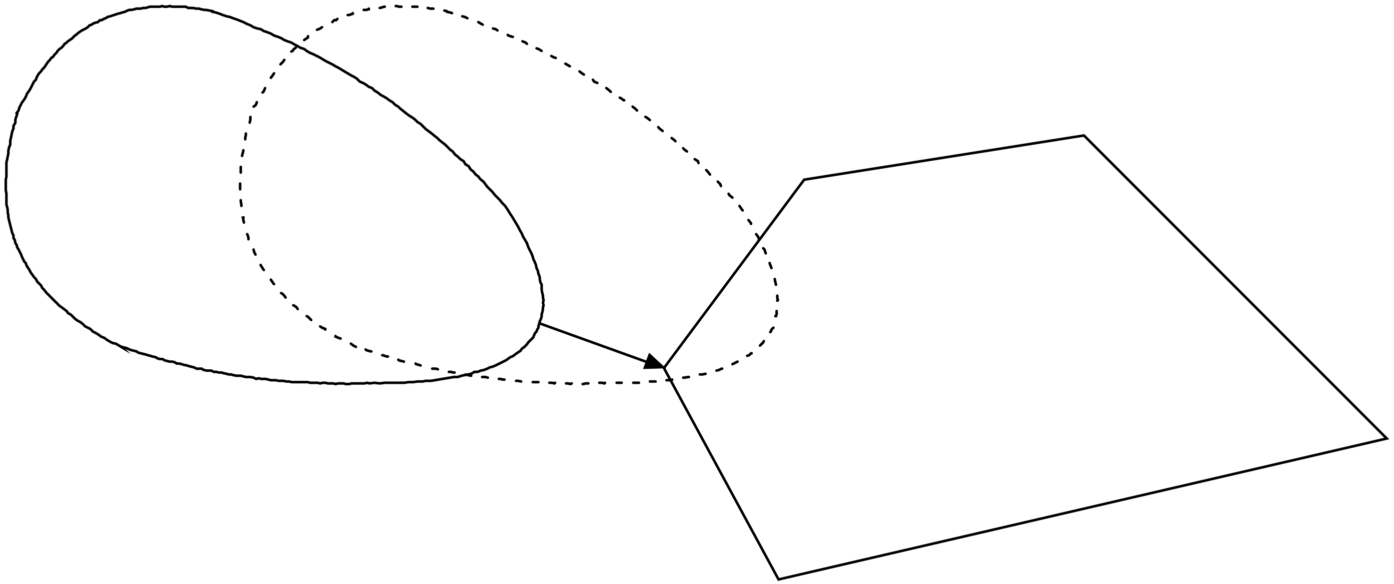
- Propagate post-collision effects
- Possible effects
  - Destroy one or both objects
  - Play sound effect
  - Inflict damage
- Many effects can be done either in the prologue or epilogue

Collision Resolution: Resolving Overlap Testing

1. Extract collision normal
2. Extract penetration depth
3. Move the two objects apart
4. Compute new velocities

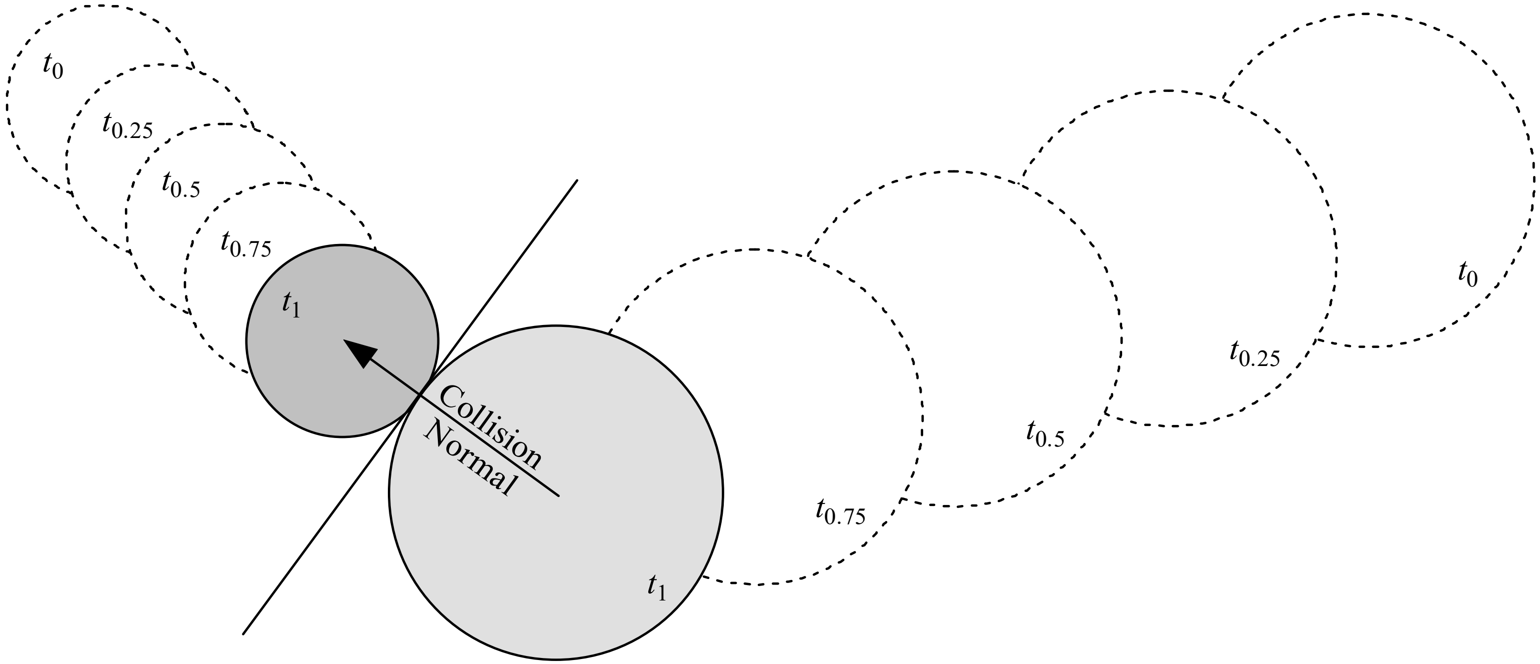
Collision Resolution: Extract Collision Normal

- Find position of objects before impact
- Use two closest points to construct the collision normal vector



Collision Resolution: Extract Collision Normal

- Sphere collision normal vector
  - Difference between centers at point of collision



Collision Resolution: Resolving Intersection Testing

- Simpler than resolving overlap testing
  - No need to find penetration depth or move objects apart
- Simply
  1. Extract collision normal
  2. Compute new velocities