Chapter 4.2
Collision Detection and Resolution

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

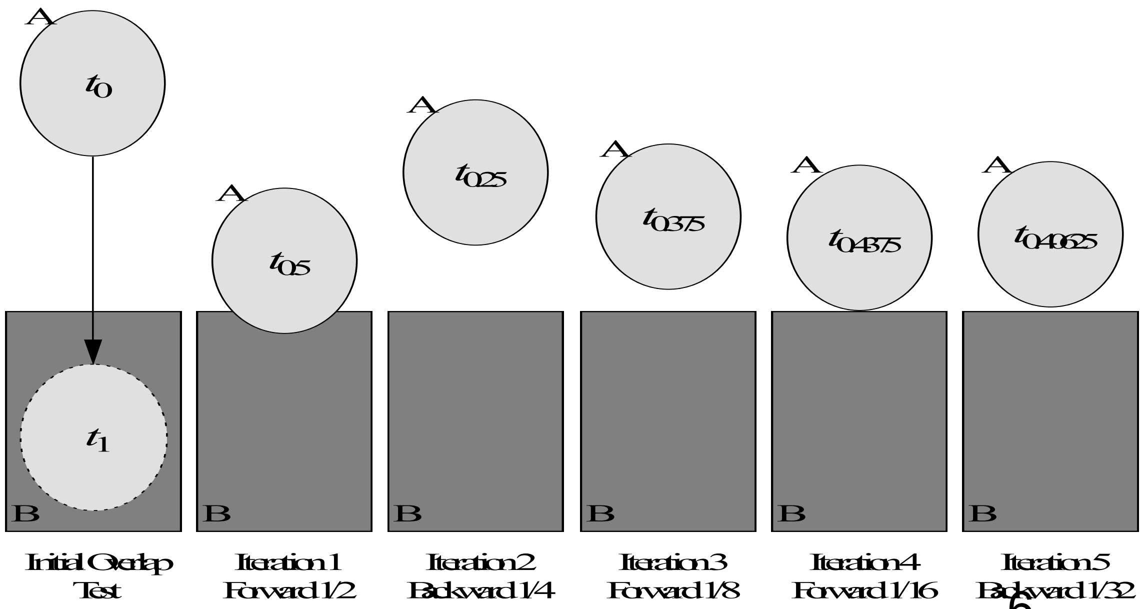
- Facts
  - Most common technique used in games
  - Exhibits more error than intersection testing
- Concept
  - For every simulation step, test every pair of objects to see if they overlap
  - Easy for simple volumes like spheres, harder for polygonal models

Overlap Testing: Useful Results

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

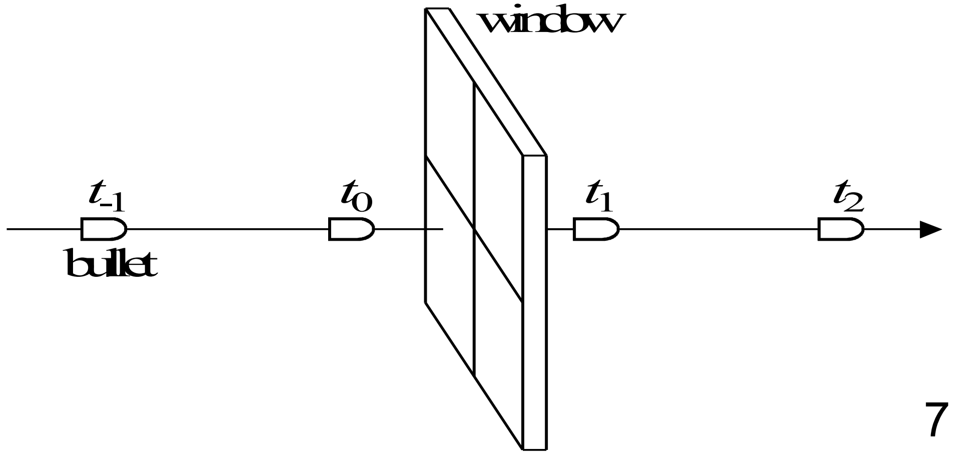
Overlap Testing: Collision Time

- 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

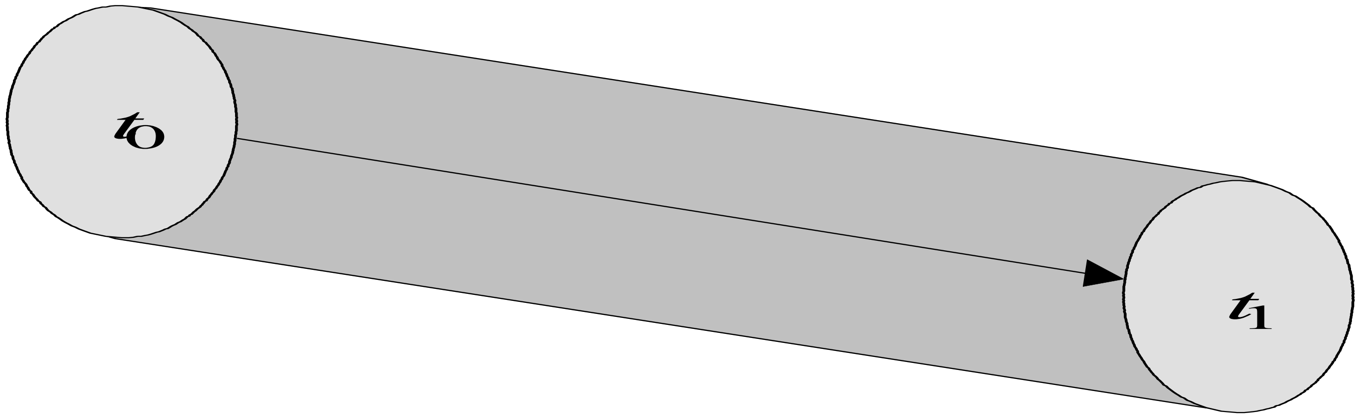


Intersection Testing

- 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



Reminder about nomenclature

A (bolded variables are vectors)

A (italicized variables are scalars)

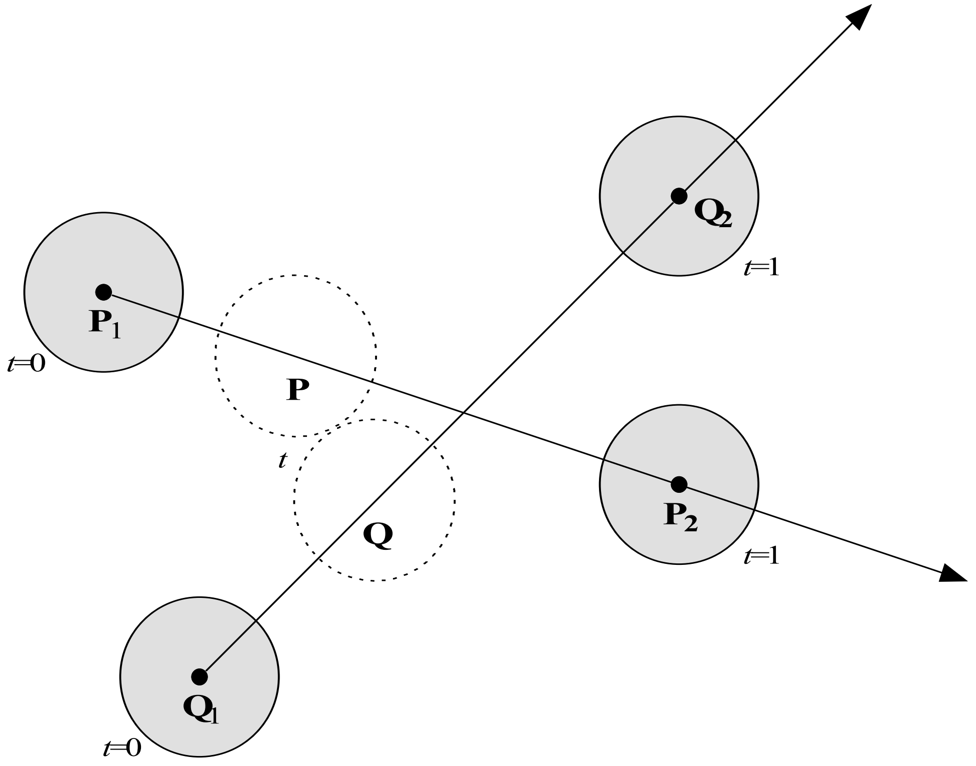
In cases where the name is the same, the scalar is the magnitude of the Vector (Pythagoras).

Intersection Testing: Special Case, Sphere-Sphere

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

$$\mathbf{A} = \mathbf{P}_1 - \mathbf{Q}_1$$

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



$$ax^2 + bx + c = 0,$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

Intersection Testing: Sphere-Sphere Collision

- Smallest distance ever separating two spheres:

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

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

  there is a collision

Intersection Testing: Limitations

- More costly then object overlap
- 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

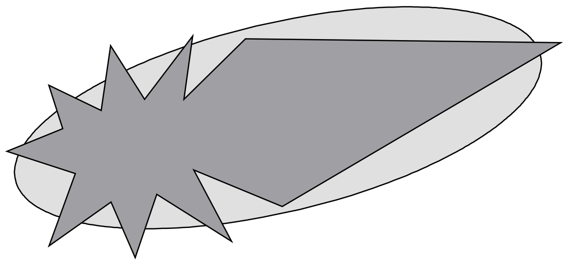
Dealing with Complexity

Two issues

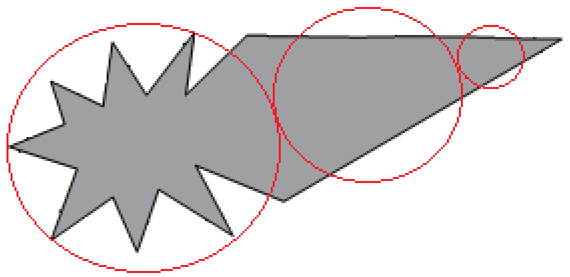
1. Complex geometry must be simplified
2. Reduce number of object pair tests

Dealing with Complexity: Simplified Geometry

- Approximate complex objects with simpler geometry, like this ellipsoid



- Or multiple spheres

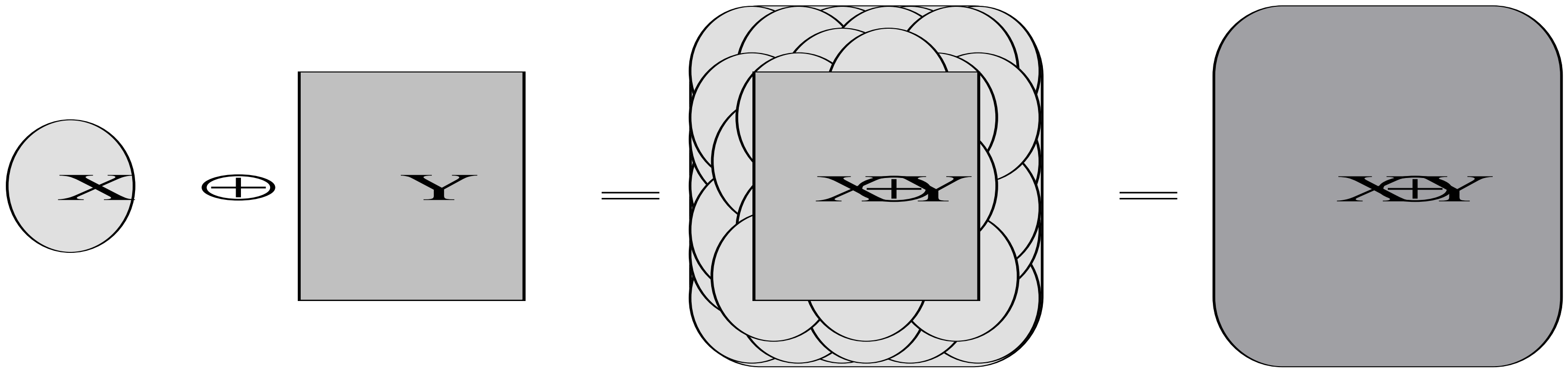


Dealing with Complexity: Minkowski Sum

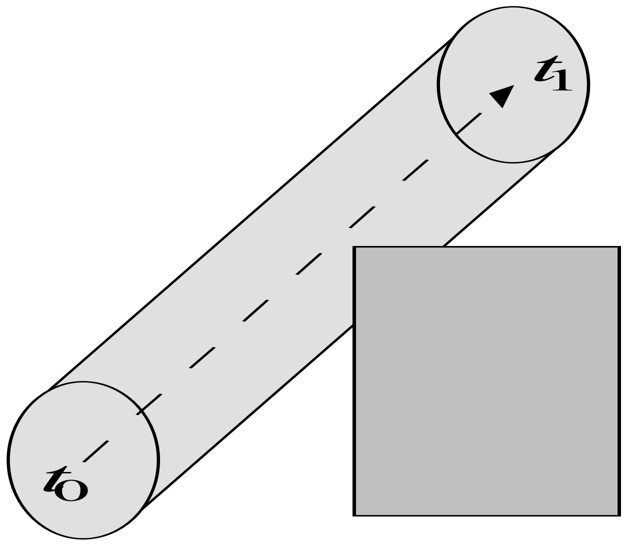
- Two complex shapes might take dozens of test to determine if they overlap.
- By taking the Minkowski Sum of two complex volumes and creating a new volume, overlap can be found by testing if <u>a single point</u> is within the new volume

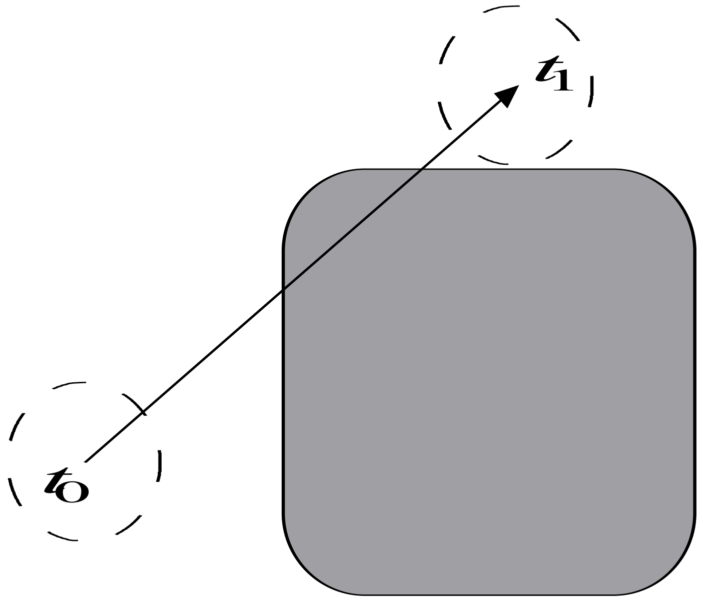
Dealing with Complexity: Minkowski Sum

[illegible]



Dealing with Complexity: Minkowski Sum

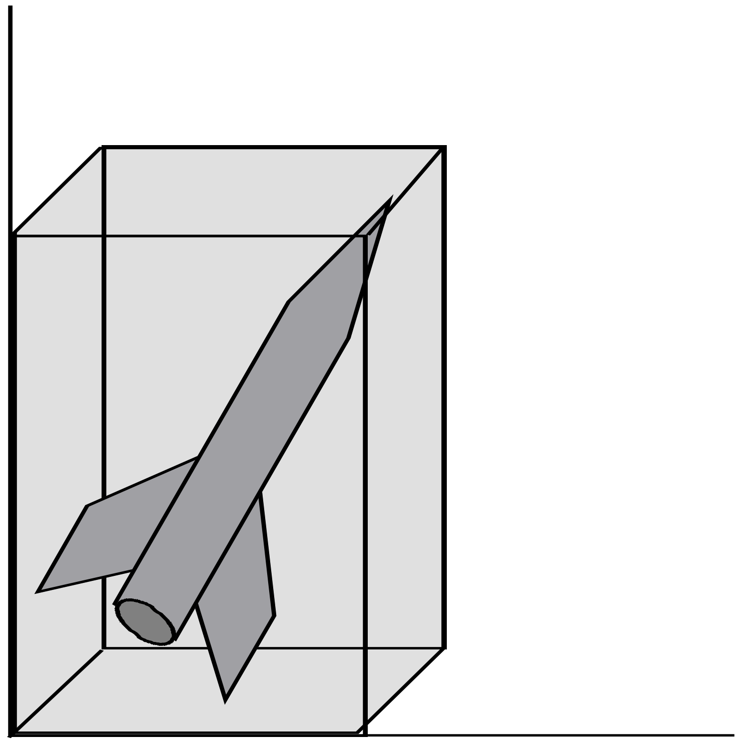


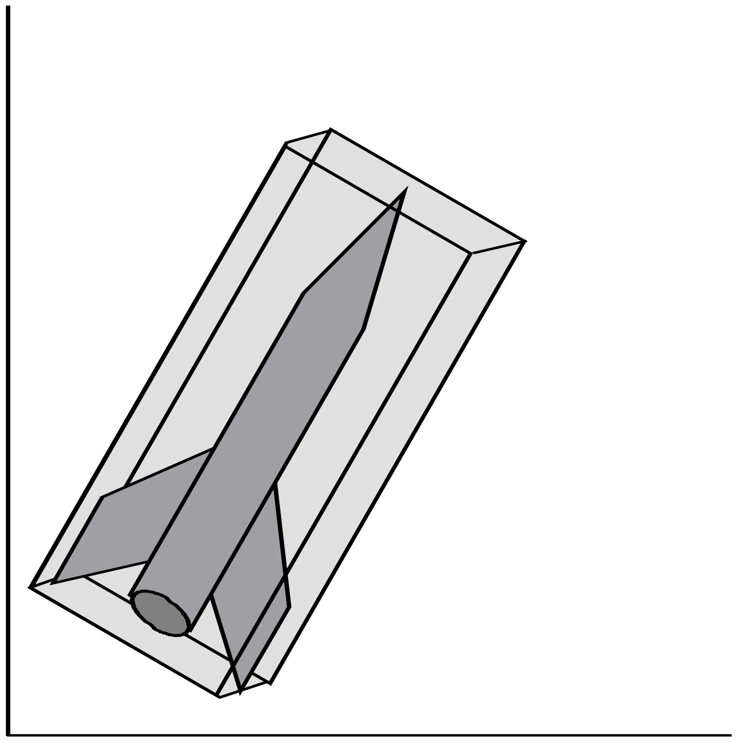


Dealing with Complexity: Bounding Volumes

- 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

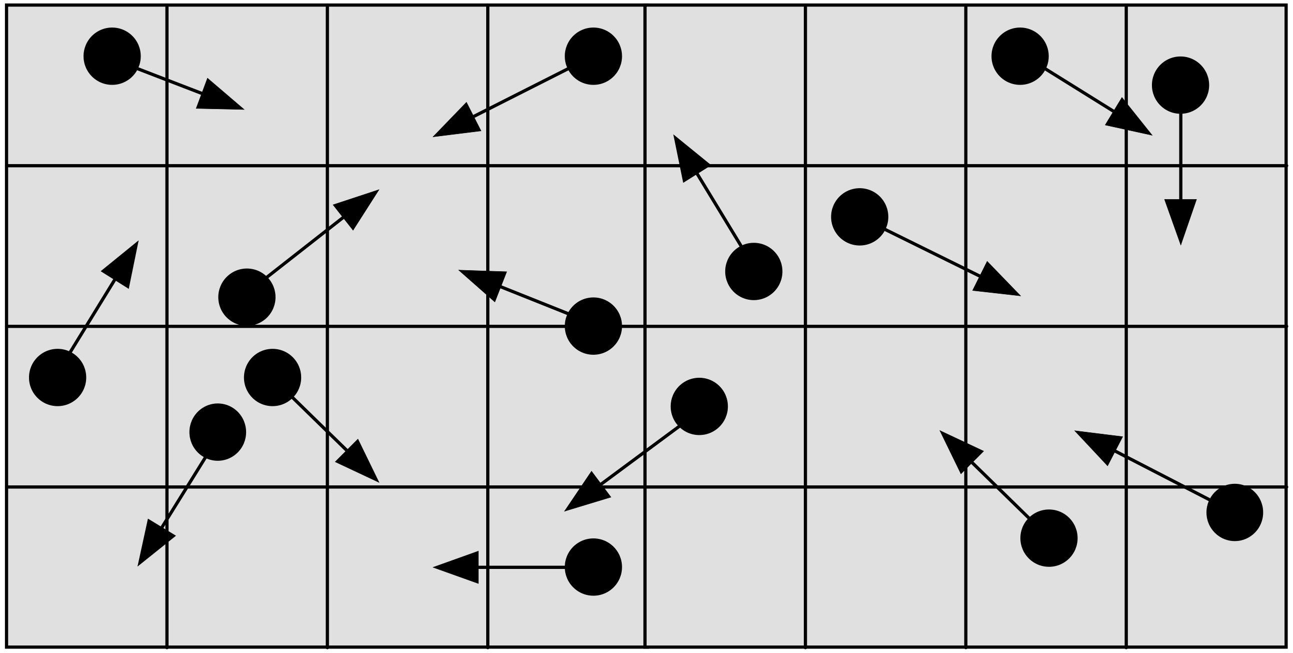
Dealing with Complexity: Box Bounding Volumes


Axis-Aligned Bounding Box


Oriented Bounding Box

Dealing with Complexity: Achieving O(n) Time Complexity

One solution is to partition space



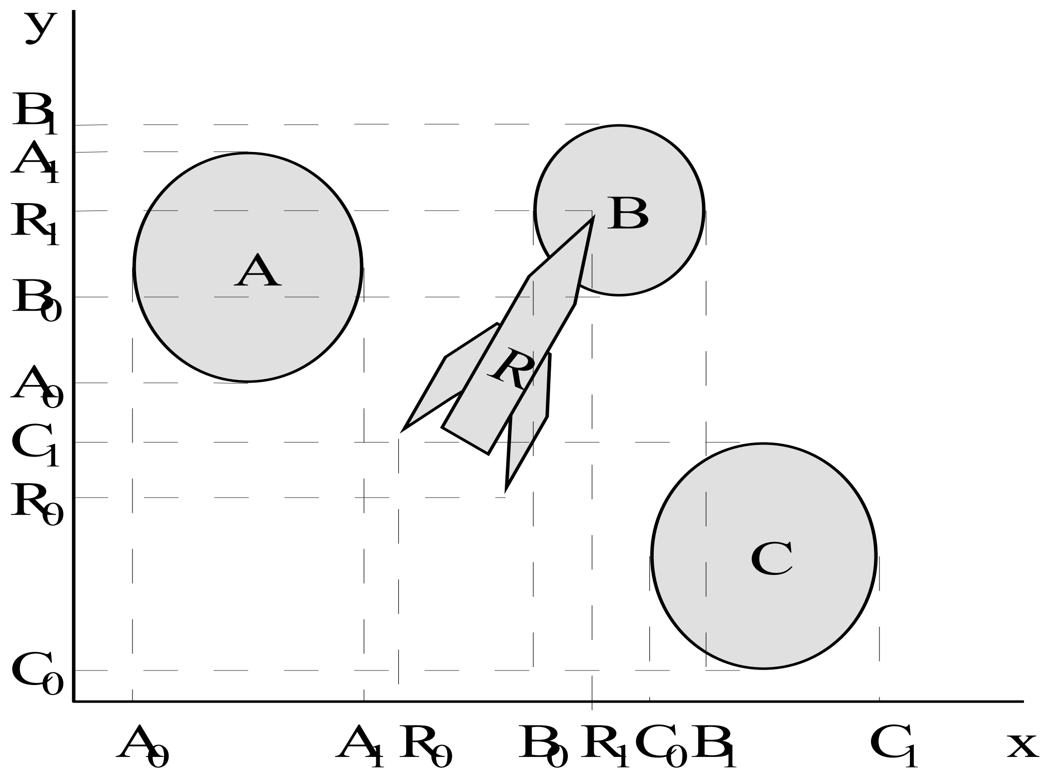
Game Entities – Identification (Hash Maps)

UID's allow multiple different lists or data structure over same object set.

Observer model (objects could register their current quadrant with CD object)

Dealing with Complexity: Achieving O(n) Time Complexity

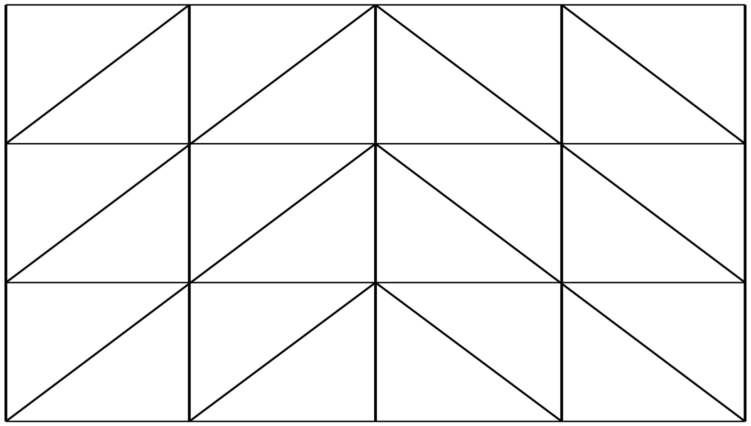
Another solution is the plane sweep algorithm

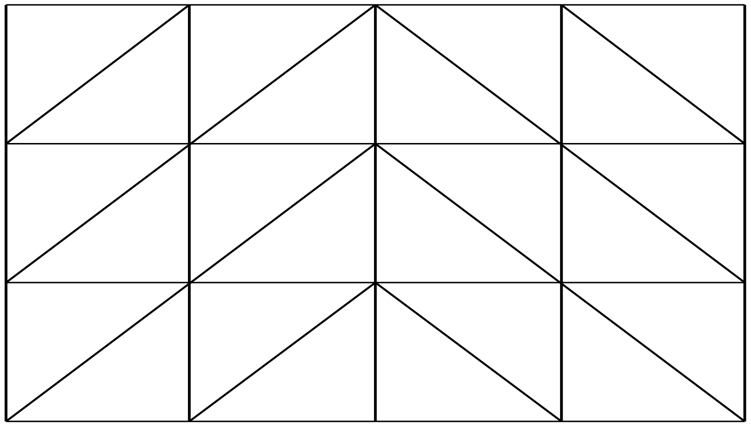


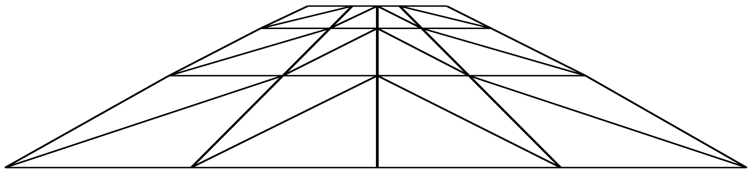
1. Find bounds in the X, Y and Z planes.
2. Add values to appropriate lists.
3. Lists are sorted initially with *quicksort* Θ(n(log(n))
4. Object coherence says that objects from frame to frame won't move much.
5. Use bubblesort to do fast update Θ(n).

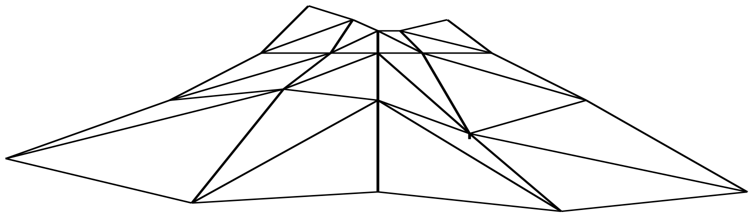
Terrain Collision Detection: Height Field Landscape

Polygonal mesh with/without height field

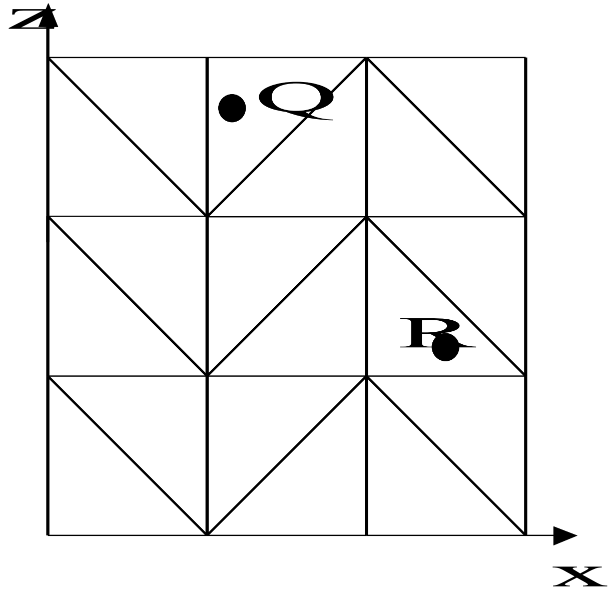

Top-Down View

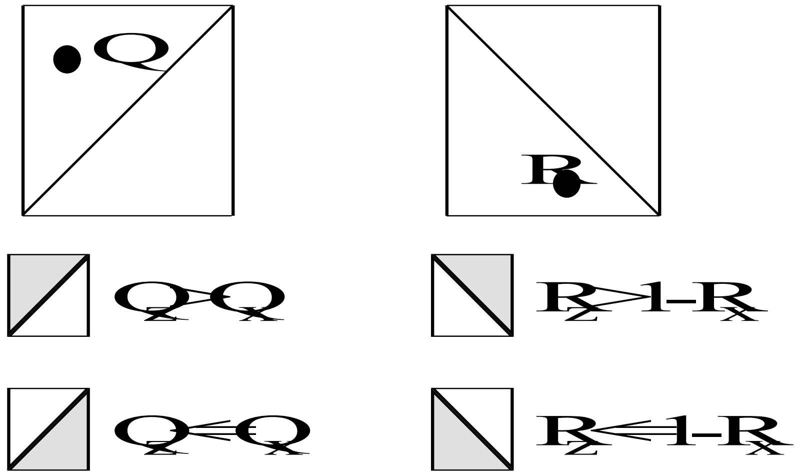

Top-Down View (heights added)


Perspective View


Perspective View (heights added)

Terrain Collision Detection: Locate Triangle on Height Field

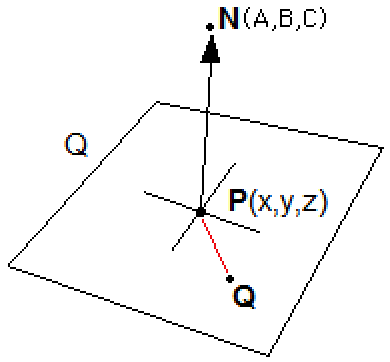




Q is the heel of the foot of the character.

With triangle located determine height.

Flashback



Remember:

Dot product of two perpendicular vectors is 0.

$$\mathbf{V} \cdot \mathbf{W} = \|\mathbf{V}\| \|\mathbf{W}\| \cos \alpha$$

Cross product of two vectors is a vector perpendicular to the other two vectors.

Planes in 3D

Given a 3D point **P**<x,y,x> and a point **N**<A,B,C> we can define a plane Q as the set of all points **Q** such that the line from **P** to **Q** is perpendicular to the line from **P** to **N.**

Definition of a plane restated

Definition of a plane:

The set of points Q such that:

$(\mathbf{N} - \mathbf{P}) \cdot (\mathbf{Q} - \mathbf{P}) = 0$

Note: We commonly reduce **N** to a distance vector and when w do the equation becomes:

$\mathbf{N} \cdot (\mathbf{Q} - \mathbf{P}) = 0$

Your book persists in calling N a normal vector, which would only make sense if the plane is already defined.

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:

$$Q_y = \frac{-N_x Q_x - N_z Q_z + (\mathbf{N} \cdot \mathbf{P}_0)}{N_y}$$

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

Prologue

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

Collision

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

Epilogue

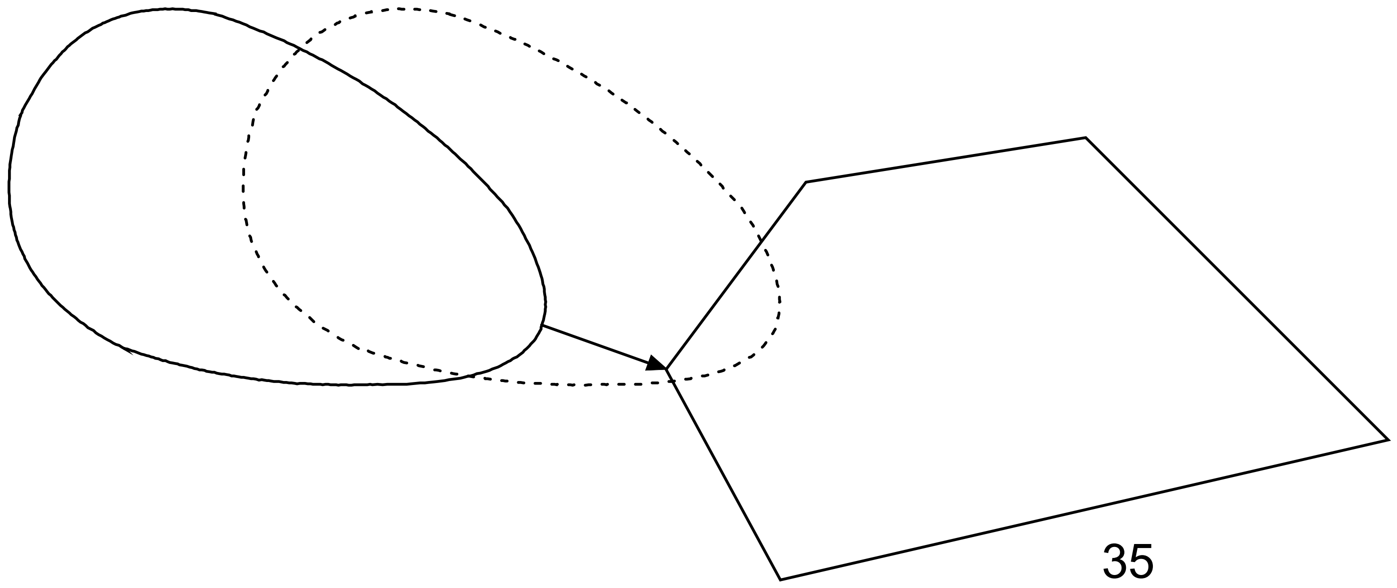
- Propagate post-collision effects
- Possible effects
  - Destroy one or both objects
  - Play sound effect
  - Inflict damage
- Many effects can be done either in the prologue or epilogue

Collision Resolution: Resolving Overlap Testing

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

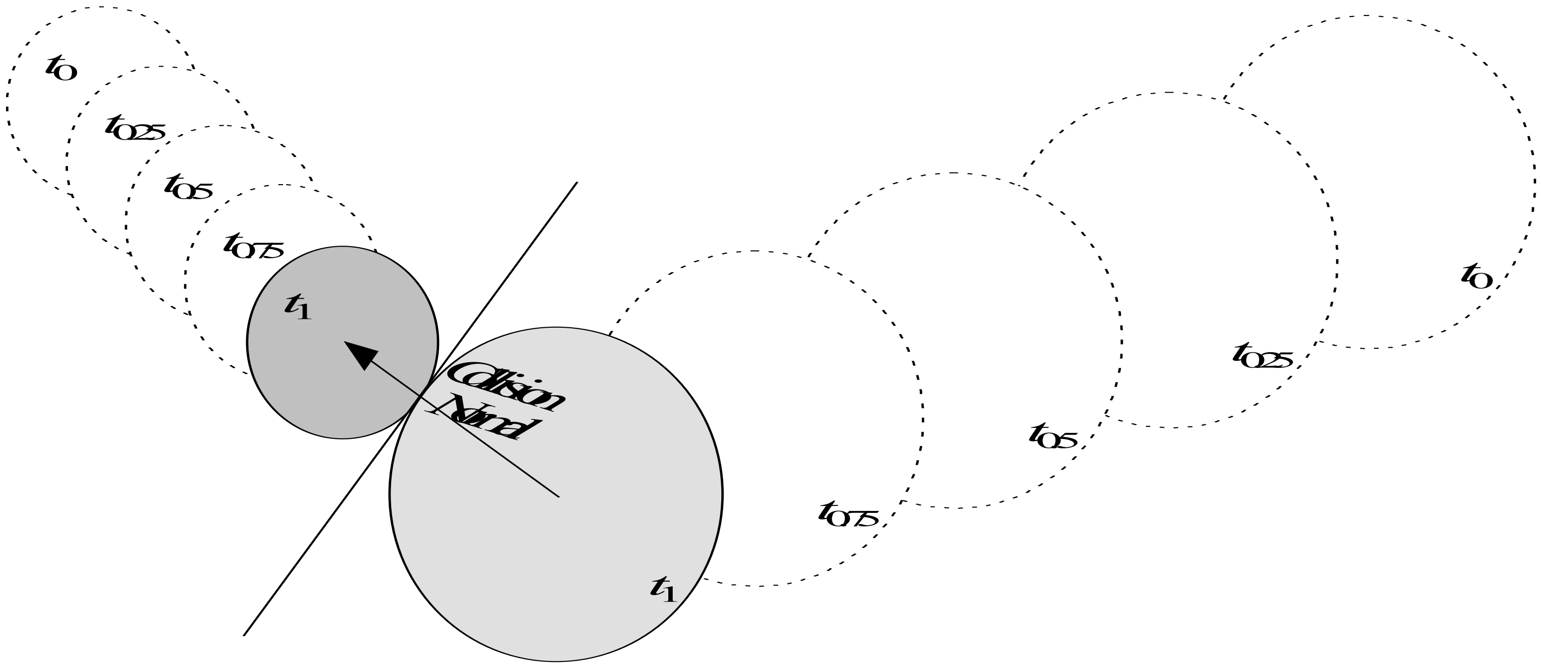
Collision Resolution: Extract Collision Normal

- 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

The End