Lecture 18:

Collisions II

Previous Lecture

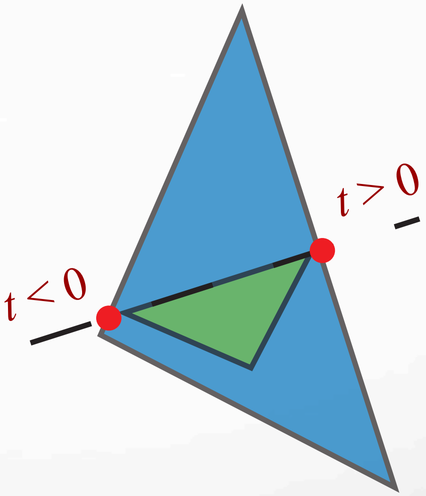
- **Collisions Detection**:
  - Did a collision occur?
  - Where did it occur?
- **Collision Resolution**:
  - Do the objects bounce?
  - Where do they go?

Today's Lecture

- **Optimization**
  - How to make detection faster
  - How detection works in industry
- **Tunneling**
  - How to prevent it (maybe)
  - What to do when it happens

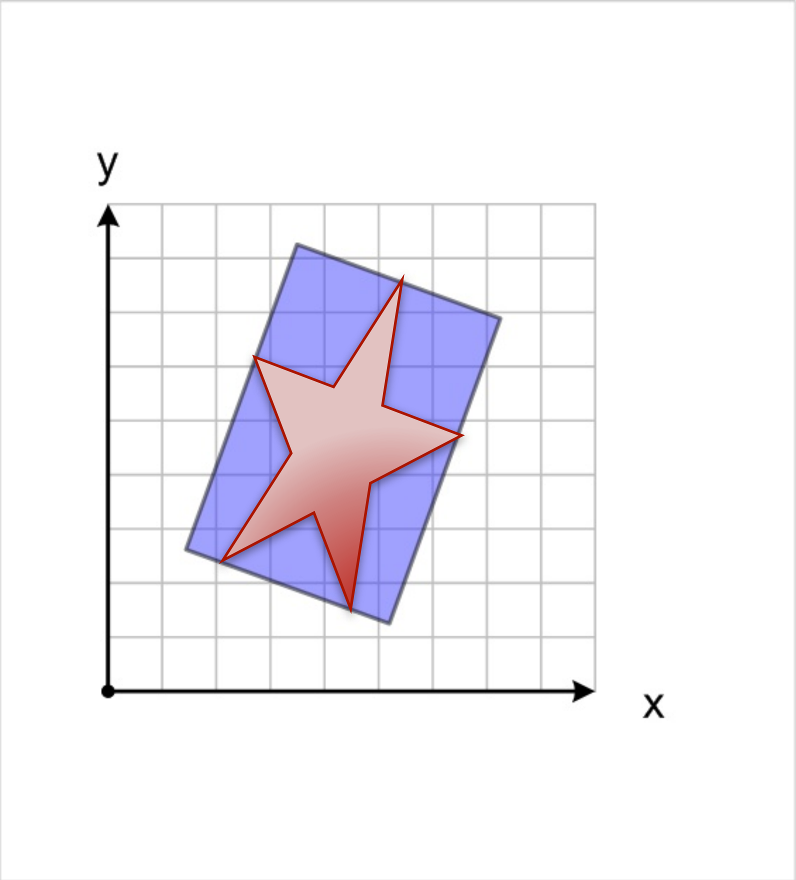
Performance: Two Issues

- $O(n^2)$ comparison of pairs of triangles
  - Need to limit pairs to compare
  - **Standard trick**: grid the space
    - Check only same/neighboring cell
    - Purpose of Programming Lab 4
- Triangle intersection is "fiddly"
  - Lots of corner cases to check
  - What about other convex shapes?



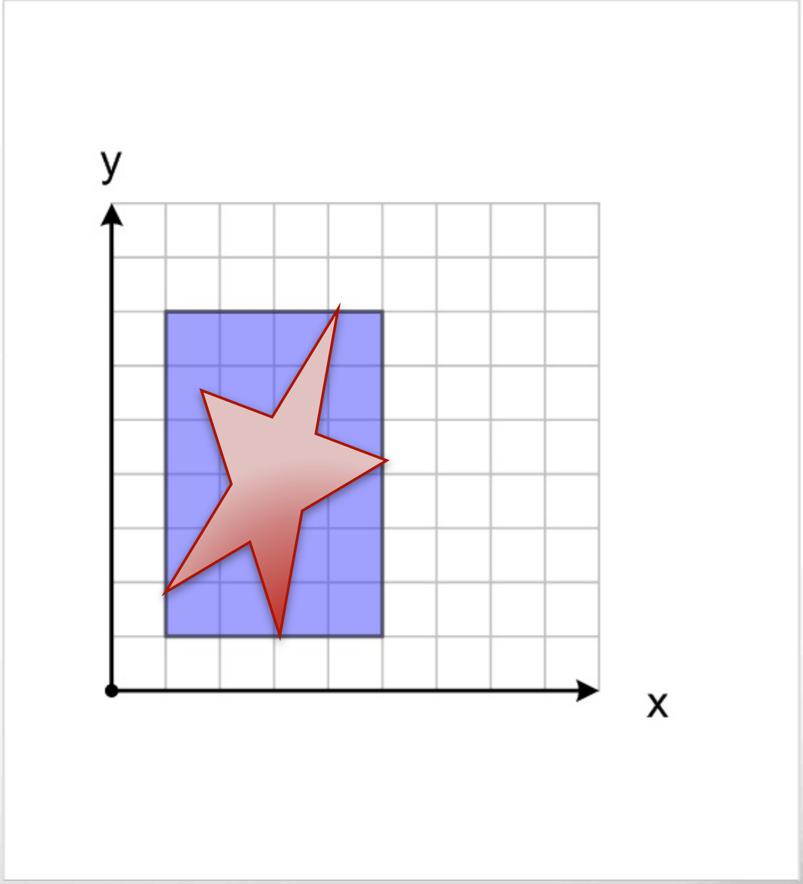
Alternative: Oriented Bounding Box

- Rectangular bounds
  - Minimal rectangle fitting
  - May be angled to fit
- Often less tight of a fit
  - Creates false positives
- Just as slow as triangles
  - Boxes have may have different orientations
  - Leads to same corner cases



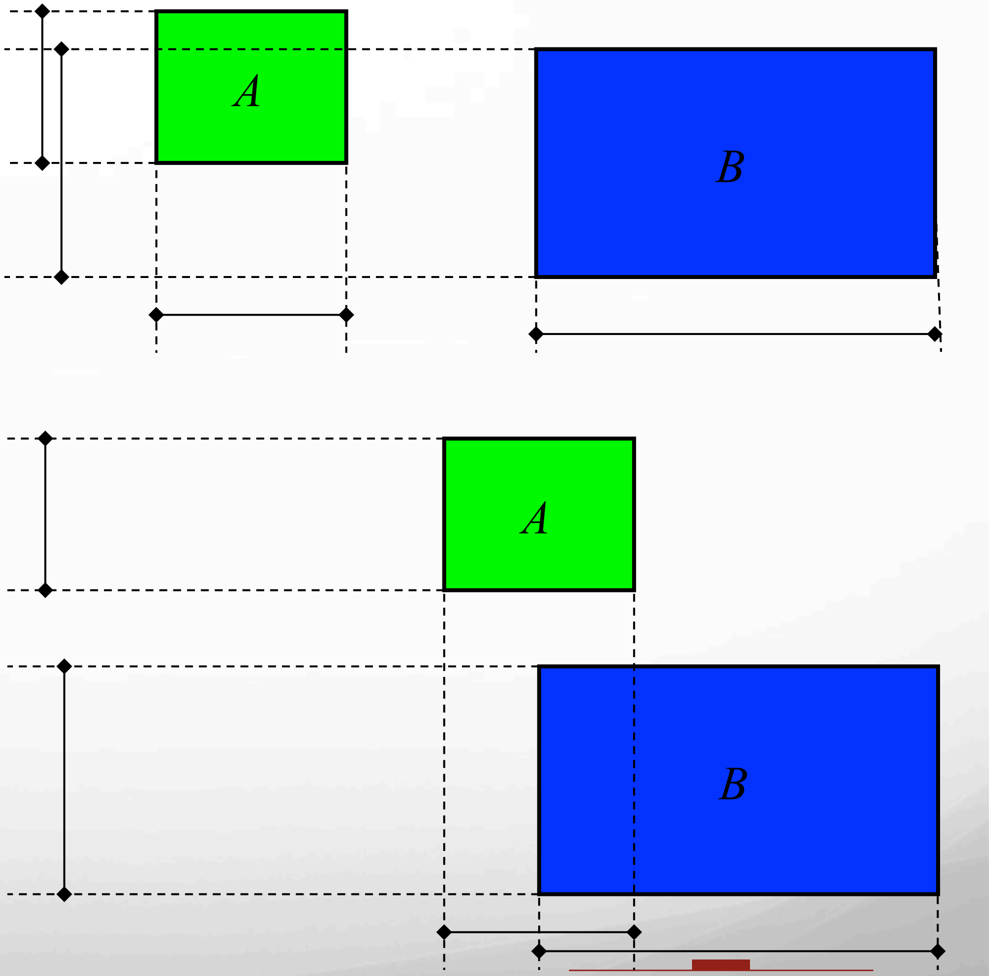
Alternative: Axis-Aligned Bound Box

- Similar to OBBs
  - Also rectangular fit
  - Must align with *x*-*y* axes
- Often a very poor fit
  - False positives are likely
- But check is very cheap
  - Project box onto axes
  - Check whether both intervals overlap



Alternative: Axis-Aligned Bound Box

- Similar to OBBs
  - Also rectangular fit
  - Must align with *x*-*y* axes
- Often a very poor fit
  - False positives are likely
- But check is very cheap
  - Project box onto axes
  - Check whether both intervals overlap



Collision Detection in Practice

- **Broad phase**:
  - Find pairs of objects that *potentially* collide
  - Use AABBs; allow false positives
  - Many optimizations from database technology
- **Narrow phase**:
  - Determine exact contact between two shapes
  - 2D: Triangle intersection from last lecture
  - 3D: Gilbert-Johnson-Keerthi (GJK) algorithm

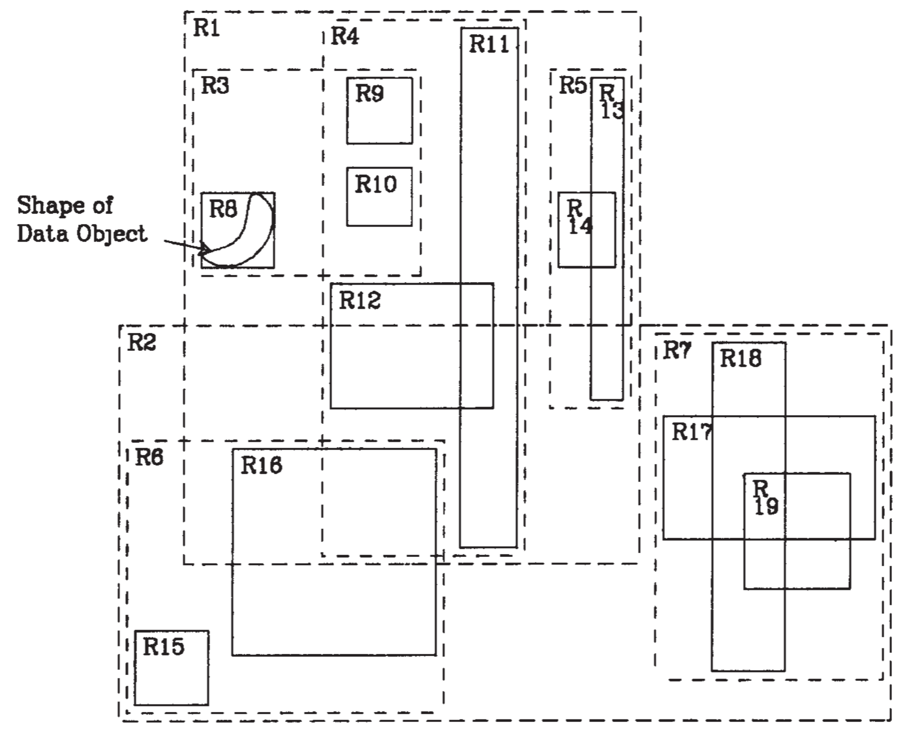
Collision Detection in Practice

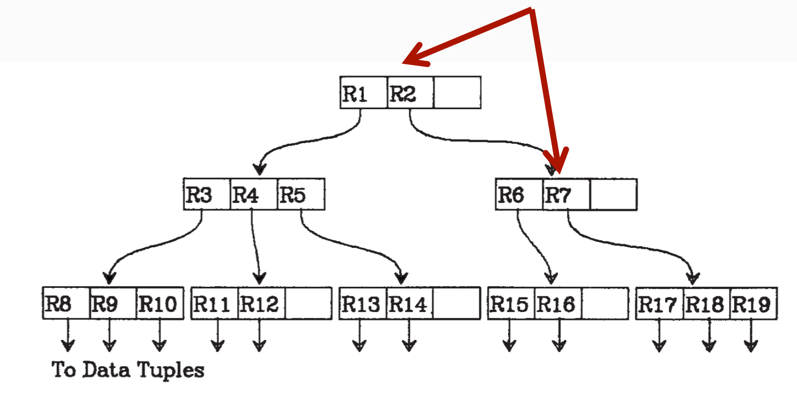
- **Broad phase**:
  - Find pairs of objects that *potentially* collide
  - Use AABBs; allow false positives
  - M logy

Some developers introduce a **Mid Phase**

- **Narrow phase**:
  - Determine exact contact between two shapes
  - 2D: Triangle intersection from last lecture
  - 3D: Gilbert-Johnson-Keerthi (GJK) algorithm

Broad Phase: R-Trees



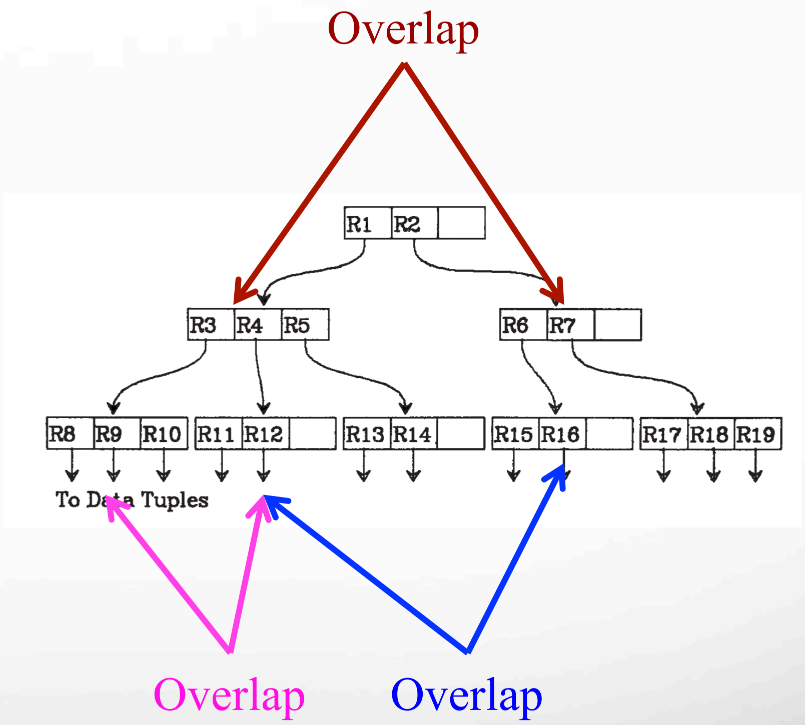


Internal AABBs

Object AABBs

Broad Phase: R-Trees

- Each internal node is an AABB
  - But these AABB may *overlap*
  - Not like a traditional search tree
  - **Trade-off**: Insertion vs. Search
- Ignore any pairs in non-overlapping AABBs
  - Descend from the root
  - Track overlaps for each node
  - Overlaps at leaves give result



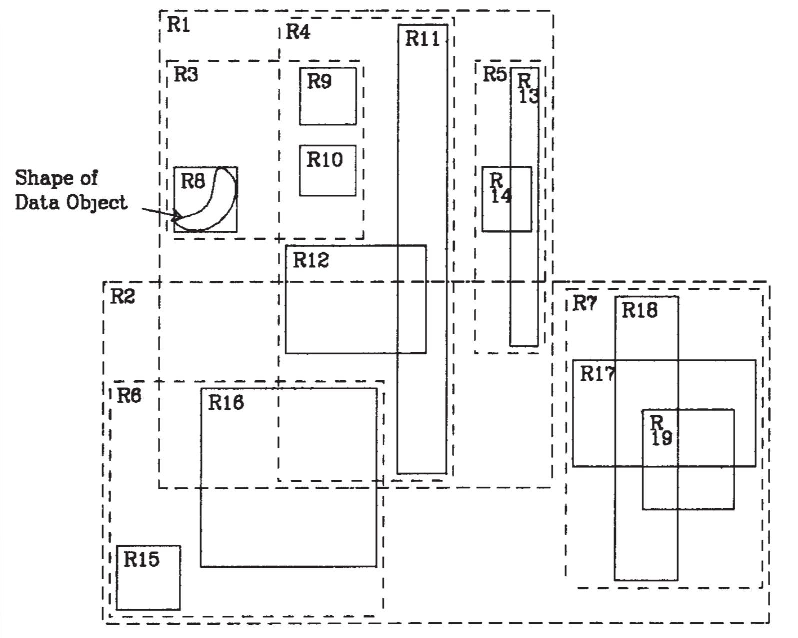
Overlap

Overlap

Overlap

R-Tree Disadvantages

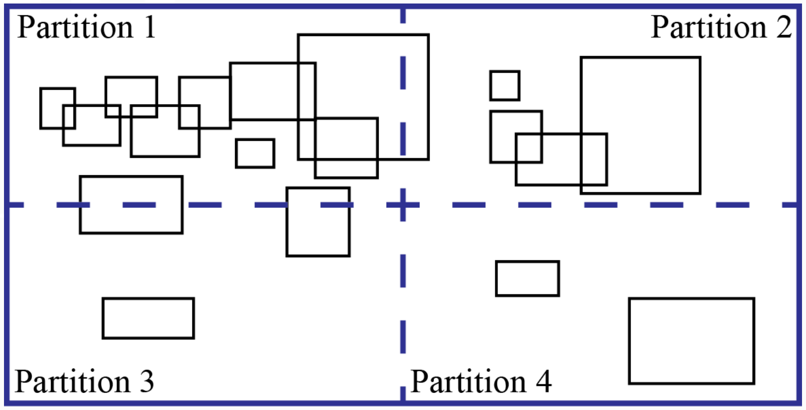
- Insertion-deletion is complicated
  - Where to add is not unique
  - Specialized algorithms to split
  - Significant overhead
- Games have too high a churn
  - Objects are always moving
  - Must remove/re-add to tree
  - Cheaper to build new tree from scratch each frame



- R-Trees often best if no one moves

Broad Phase: Partition & Sweep

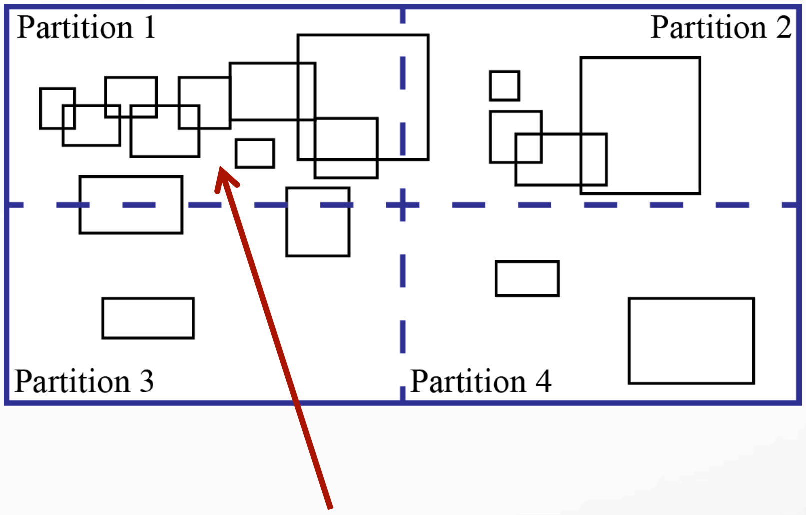
- Break up into partitions
  - Remember Lab 4 solution
  - But sizes not always equal
  - Size determined by cache
- Sort on one dimension
  - E.g. sort by box left side
- Scan sorted list in order
  - Compare all pairwise
  - Drop when opposite side of sort is out of bounds



Complex algorithms to rebalance partition size when necessary

Broad Phase: Partition & Sweep

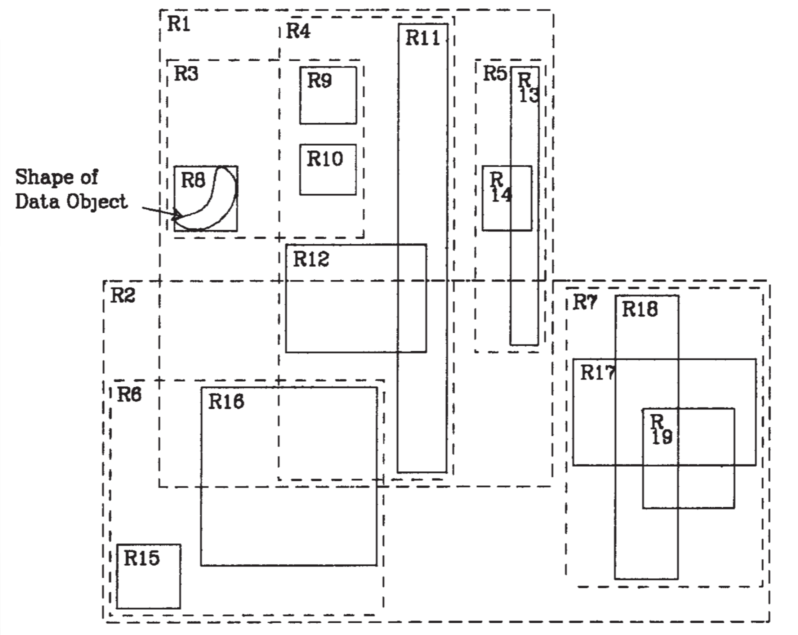
- Fastest solution possible (!?)
- Secret is the cache size
  - Memory accessed in cache lines
  - Get more memory than ask for
  - Whole partition in cache at once
- Much faster than any tree
  - Trees are not cache friendly
  - Internal nodes are often in different cache lines
  - Loading into cache expensive



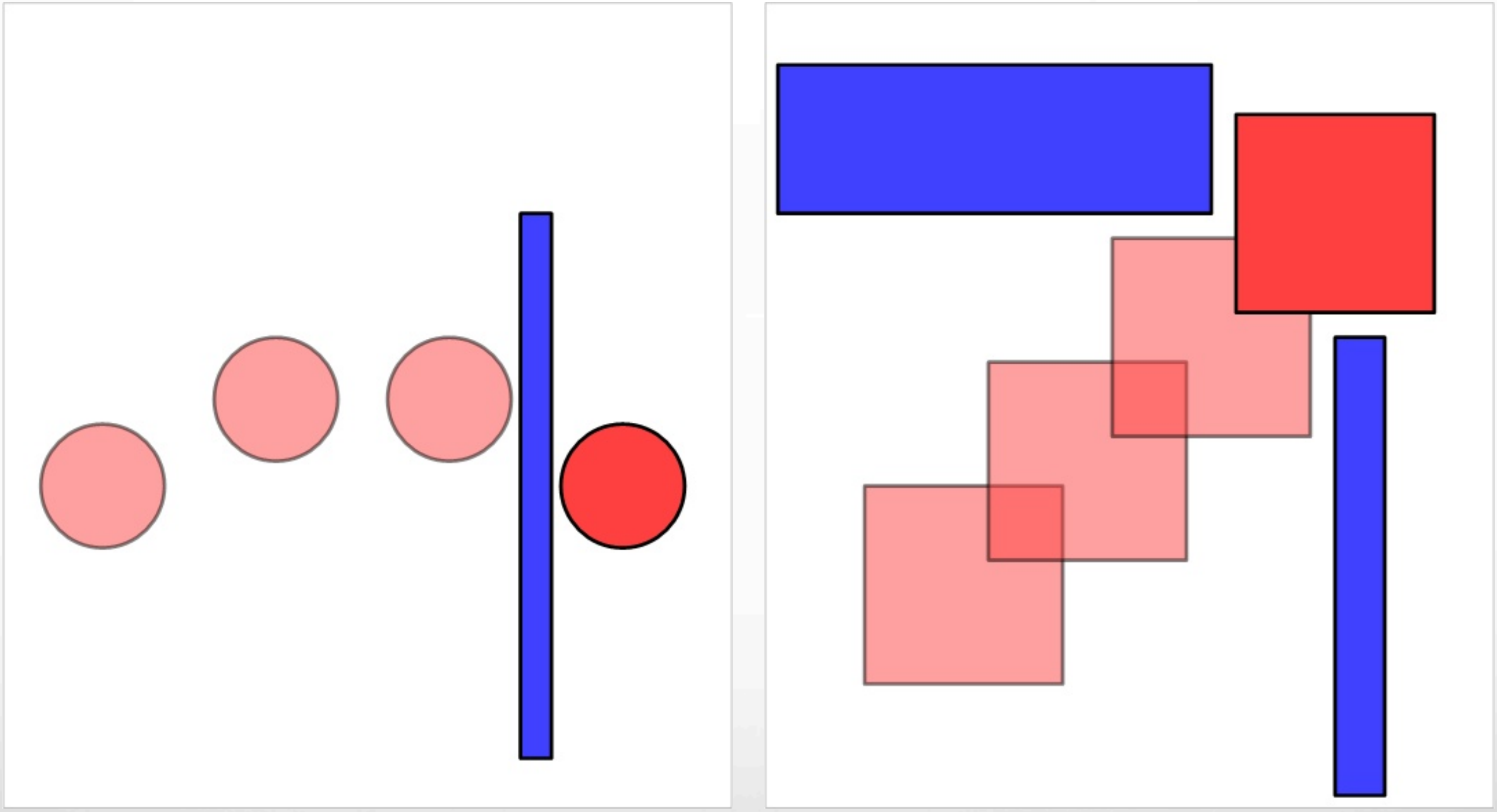
Pairwise comparison faster than loading in a new cache line

Broad Phase: Learning More

- Robust area of study
  - 30 years of database research
  - Referred to as spatial joins
  - Heavily optimized for hardware
    - Cache size dependent
    - Parallel algorithms
- Area of research in our group
  - Lots of studies into what works
  - Ask me if interested in more



Tunneling

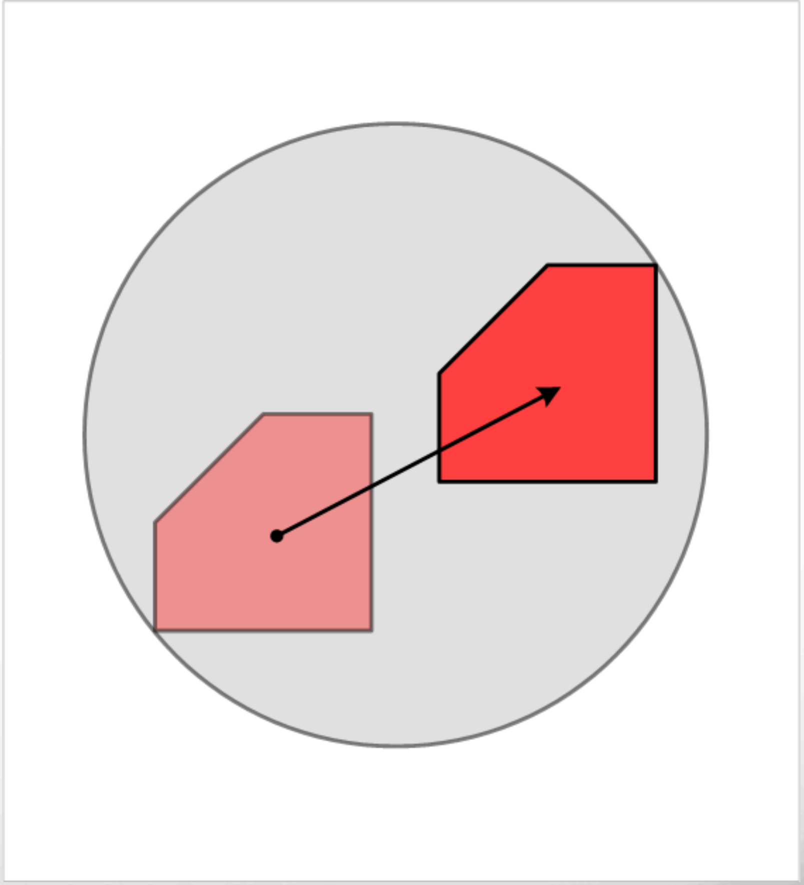


First Attempt at a Solution

- Possible solutions
  - Minimum size requirement?
    - Fast objects still tunnel
  - Maximum speed limit?
    - Speed limit is a function of object size
    - So small & fast objects (bullets) not allowed
  - Smaller time step?
    - Essentially the same as a speed limit
- All of these solutions are inadequate

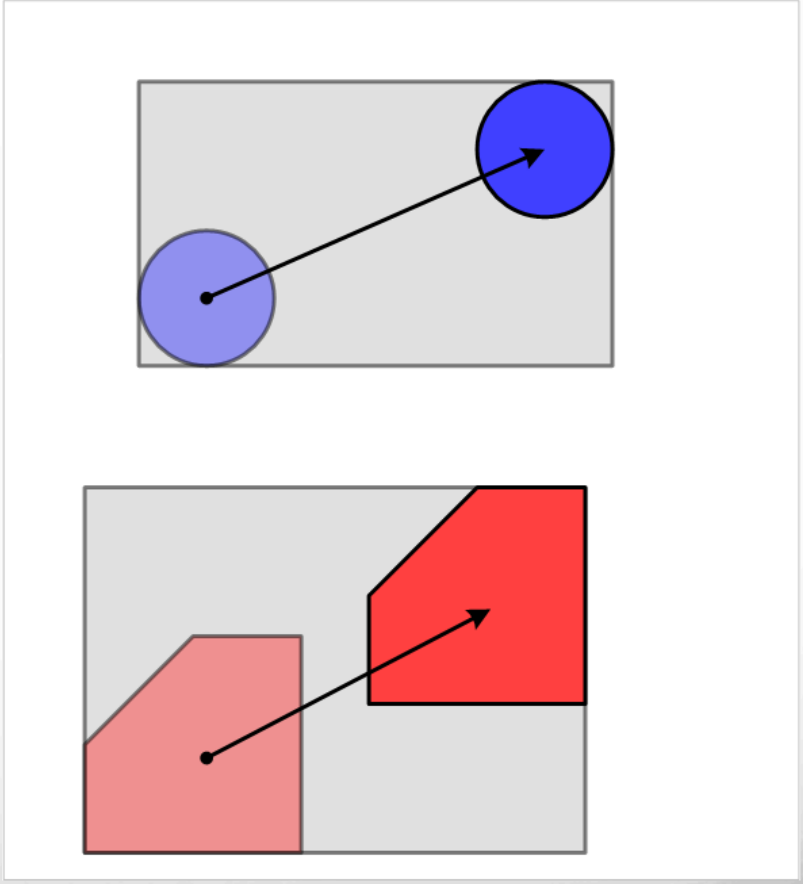
Movement Bounds

- Bounds that contain motion
  - At all times, object in bounds
  - Again, want convex bounds
- Examples
  - Disk/circle
  - AABB (axis aligned box)
  - OOBB (oriented box)



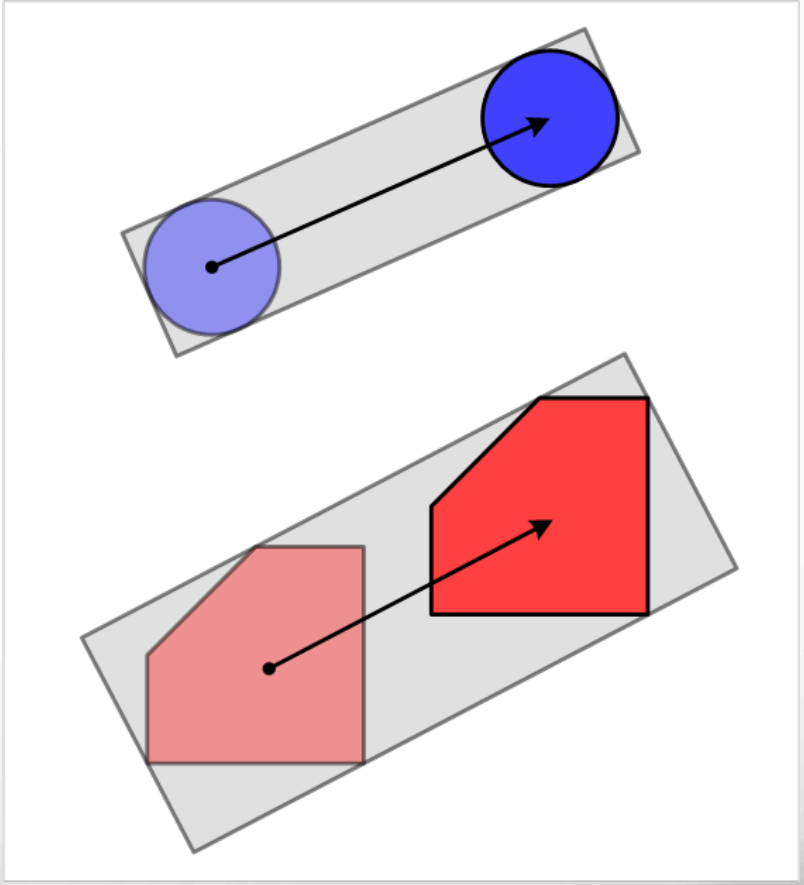
Movement Bounds

- Bounds that contain motion
  - At all times, object in bounds
  - Again, want convex bounds
- Examples
  - Disk/circle
  - AABB (axis aligned box)
  - OOBB (oriented box)



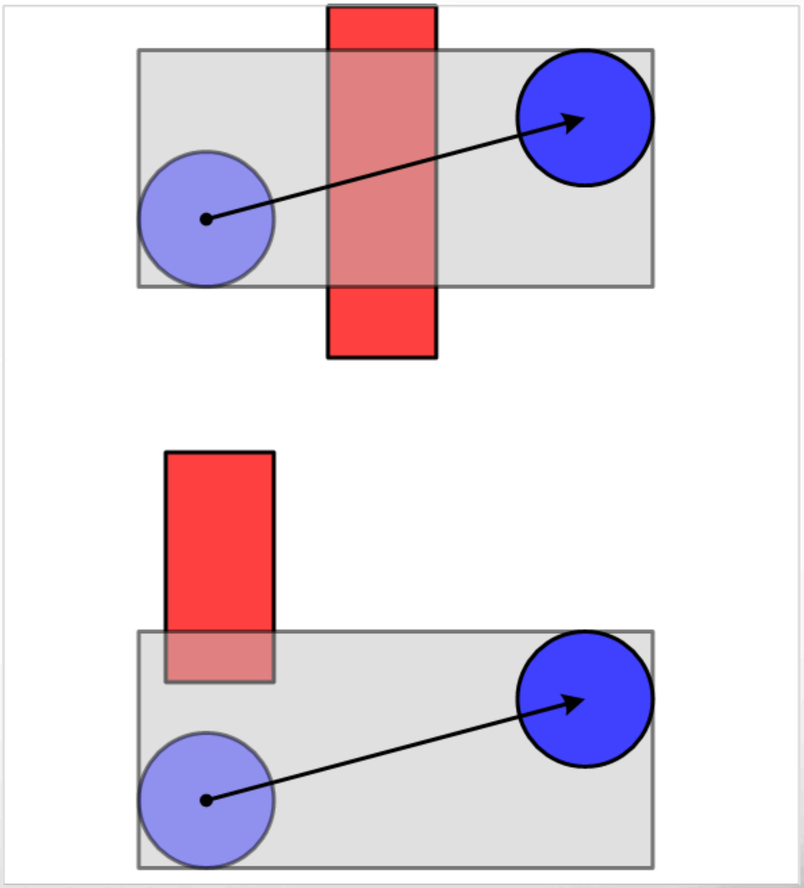
Movement Bounds

- Bounds that contain motion
  - At all times, object in bounds
  - Again, want convex bounds
- Examples
  - Disk/circle
  - AABB (axis aligned box)
  - OOBB (oriented box)



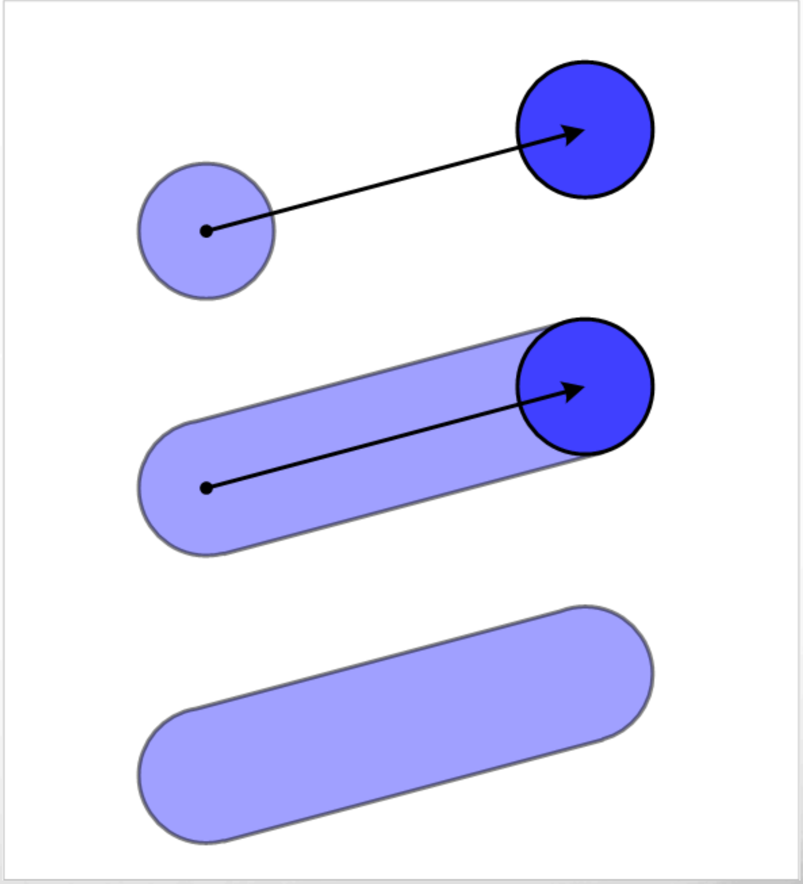
Movement Bounds

- Bounds that contain motion
  - At all times, object in bounds
  - Again, want convex bounds
- Examples
  - Disk/circle
  - AABB (axis aligned box)
  - OOBB (oriented box)
- Question: Bounds intersect?
  - **False positives** still likely



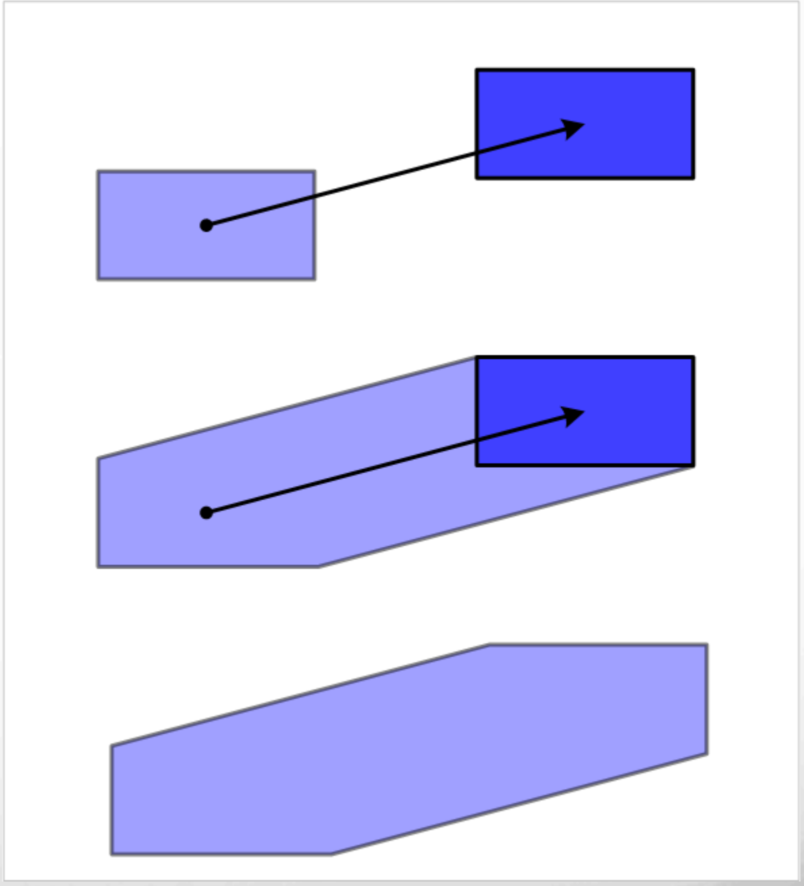
Swept Shapes

- Similar to movement bounds
  - "Cylinder" with shape at ends
  - Guaranteed perfect fit!
- Examples
  - Swept disk: capsule
  - Swept AABB: convex poly
  - Swept triangle: convex poly
  - Swept convex: convex poly



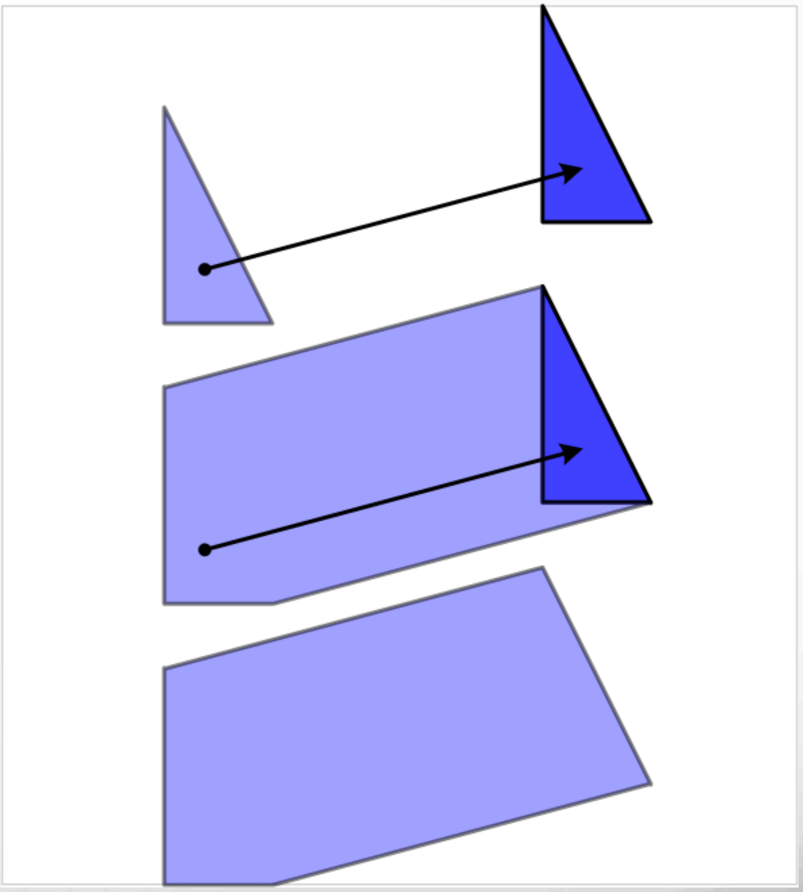
Swept Shapes

- Similar to movement bounds
  - "Cylinder" with shape at ends
  - Guaranteed perfect fit!
- Examples
  - Swept disk: capsule
  - Swept AABB: convex poly
  - Swept triangle: convex poly
  - Swept convex: convex poly



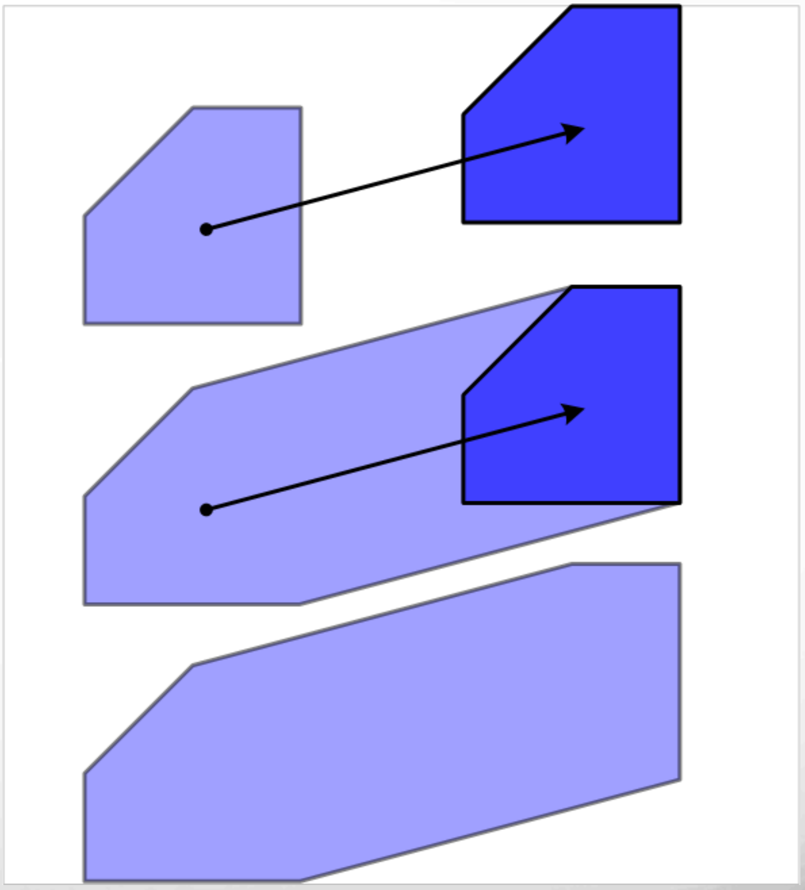
Swept Shapes

- Similar to movement bounds
  - "Cylinder" with shape at ends
  - Guaranteed perfect fit!
- Examples
  - Swept disk: capsule
  - Swept AABB: convex poly
  - Swept triangle: convex poly
  - Swept convex: convex poly



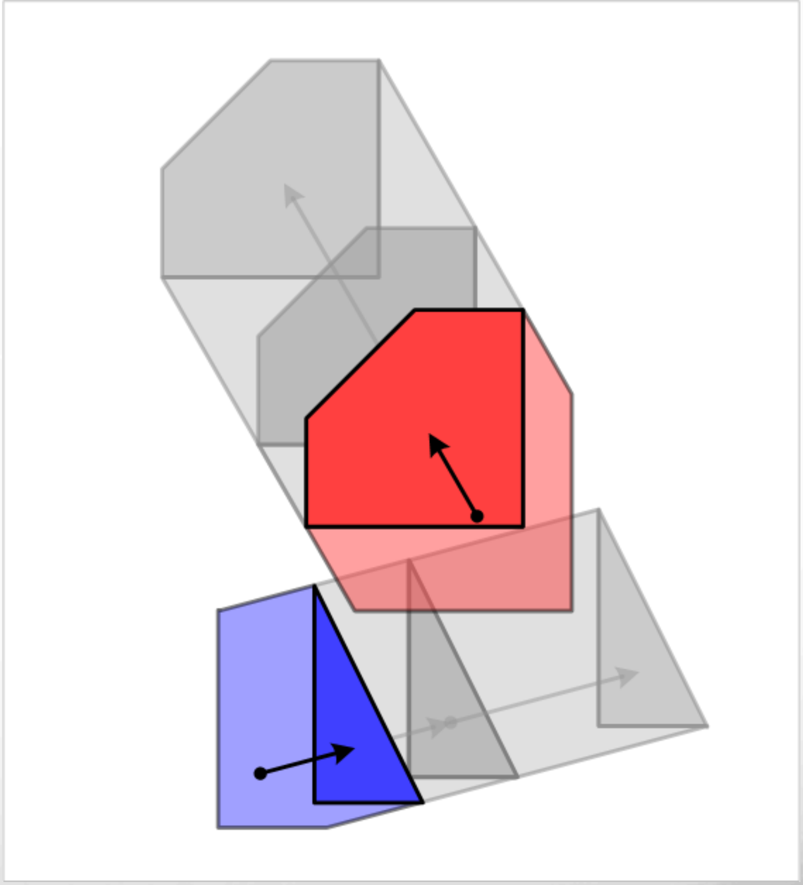
Swept Shapes

- Similar to movement bounds
  - “Cylinder” with shape at ends
  - Guaranteed perfect fit!
- Examples
  - Swept disk: capsule
  - Swept AABB: convex poly
  - Swept triangle: convex poly
  - Swept convex: convex poly



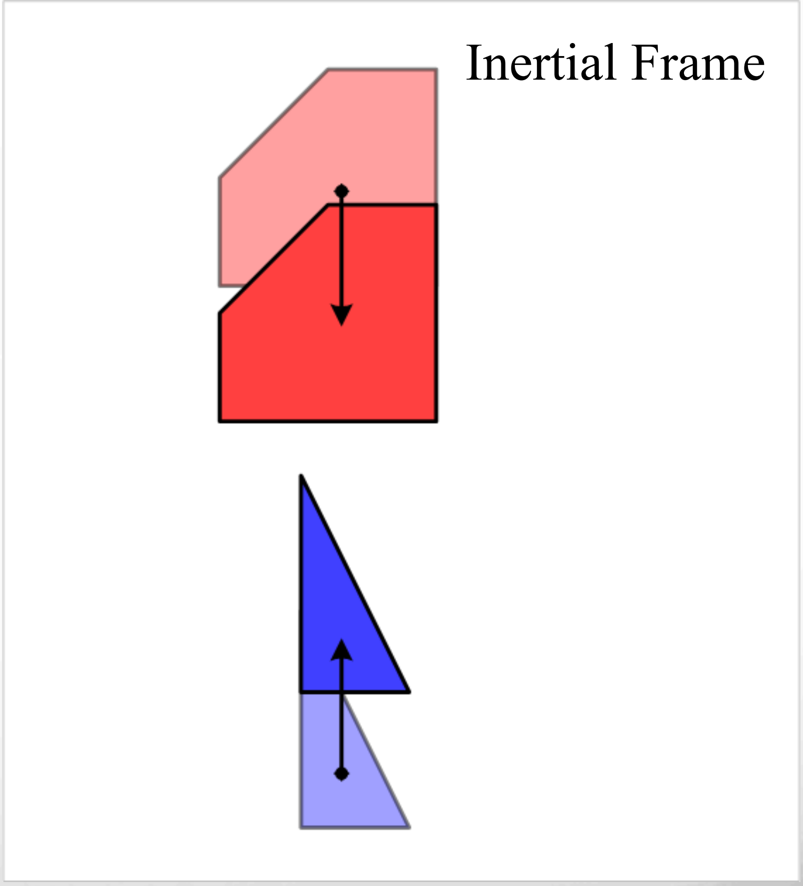
Swept Shapes

- Similar to movement bounds
  - "Cylinder" with shape at ends
  - Guaranteed perfect fit!
- Examples
  - Swept disk: capsule
  - Swept AABB: convex poly
  - Swept triangle: convex poly
  - Swept convex: convex poly
- **False positives still possible**
  - But gets rid of most problems



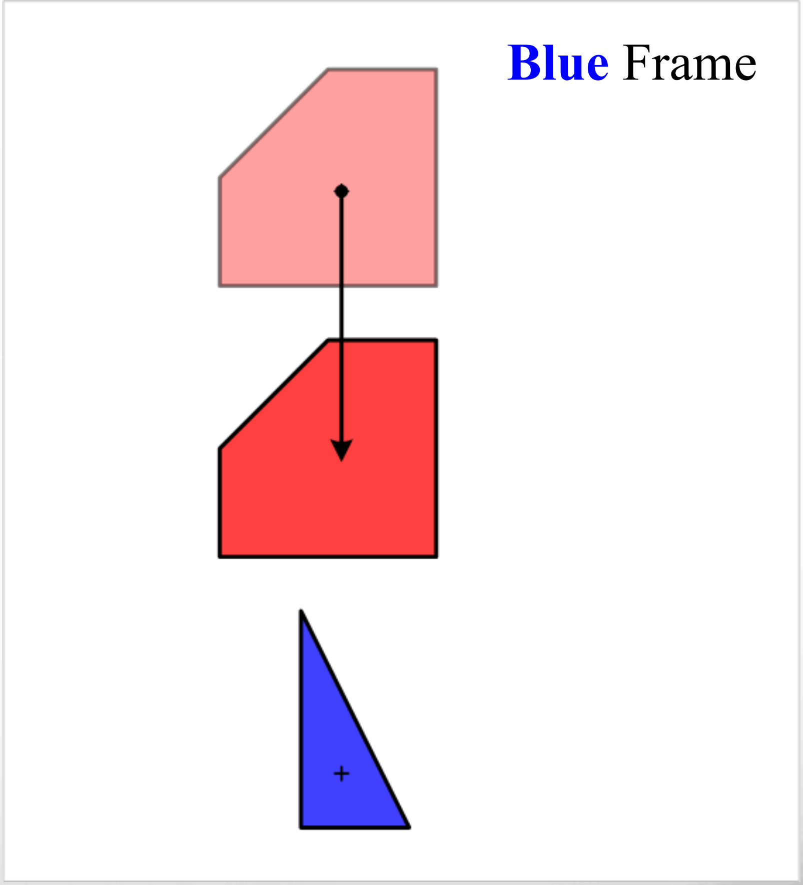
Relative Coordinates

- False positives happen if:
  - Two objects are moving
  - Swept shapes intersect at different times
- What if only one moving?
  - Swept intersects stationary
  - No false positives
- Change reference frames
  - Keep one shape still
  - Move other in new coords



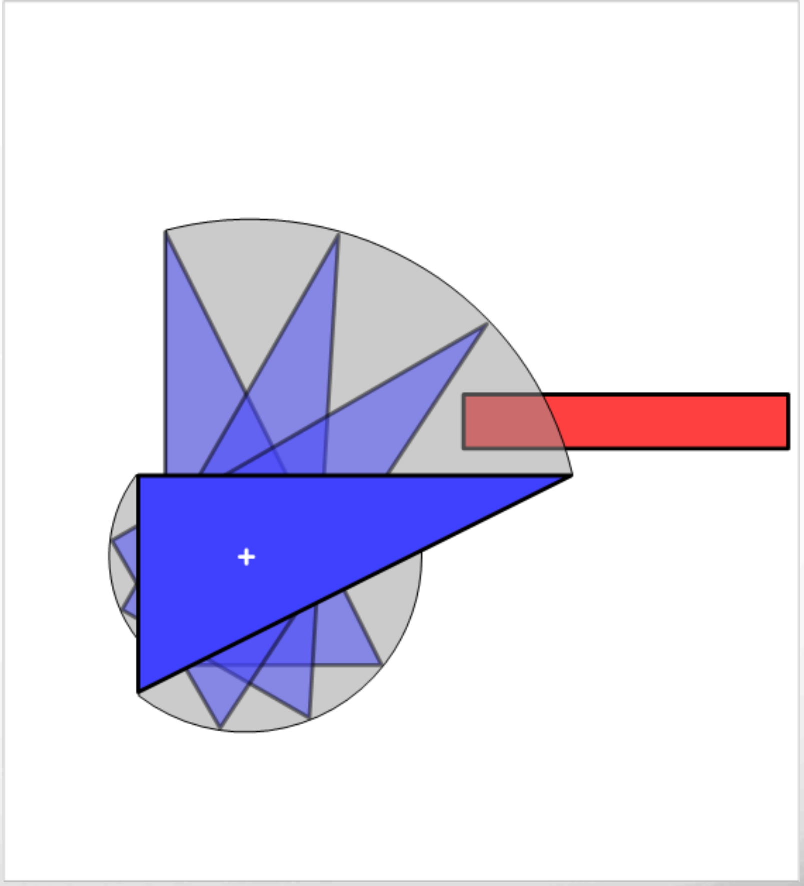
Relative Coordinates

- False positives happen if:
  - Two objects are moving
  - Swept shapes intersect at different times
- What if only one moving?
  - Swept intersects stationary
  - No false positives
- Change reference frames
  - Keep one shape still
  - Move other in new coords



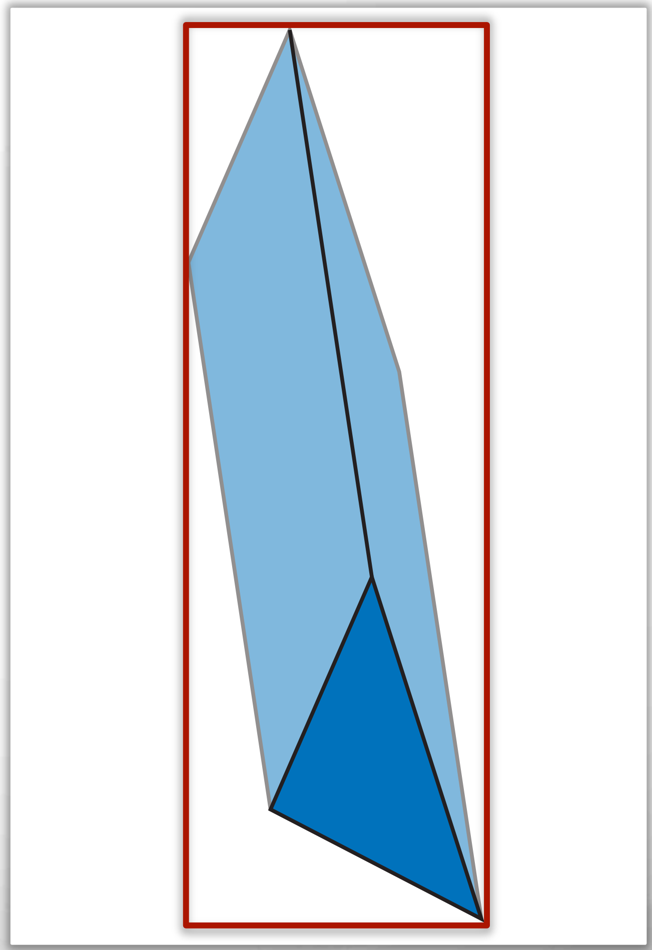
Rotations Suck

- Relative coordinates no help
  - Cannot use swept shapes
  - Actual solution is hard!
- But not so bad…
  - Rotational tunneling not jarring
  - Speed limits are feasible
  - Can do linear approximations
- Many physics systems **never** do rotational collisions well



2D Collisions: From the Top

- Define shape in data file
  - Stored in object coordinates
  - Transform to screen coords
  - XNA support: `Vector2.Transform(…);`
- Compute swept shape of hull
  - Lines between like vertices
  - Can drop internal lines
- Get AABB of swept shape
  - Width: (min *x*, max *x*)
  - Height: (min *y*, max *y*)



2D Collisions: From the Top

- Define shape in data file
  - Stored in object coordinates
  - Transform to screen coords
  - XNA support:
    `Vector2.Transform(…);`
- Compute swept shape of hull
  - Lines between like vertices
  - Can drop internal lines
- Get AABB of swept shape
  - Width: (min *x*, max *x*)
  - Height: (min *y*, max *y*)

- **Broad Phase**
  - Partition the space (Lab 4)
  - Compare AABB pairs
  - Advanced: cache usage
- **Narrow Phase**
  - Fix one of pair in place
  - For the other shape
    - Get relative translation
    - Ignore/limit rotations
    - Construct swept shape
  - Check triangle intersection

Collision Detection in Practice

- **Broad phase**:
  - Find pairs of objects that potentially collide
  - Use AABBs; allow false positives
  - Many optimizations from database technology
- **Narrow phase**:
  - Determine exact contact between two shapes
  - Gilbert-Johnson-Keerthi (GJK) algorithm
  - Mathematics beyond scope of the course…

Summary

- Collisions require geometry
  - Restrict games to convex shapes (e.g. triangles)
  - Complex characters may need several shapes
- Collision detection must tell when collision happened
  - Otherwise, we can have tunneling (very bad)
  - Swept shapes are best nonprofessional solution
- Collision resolution depends on energy transfer
  - Disks/balls are easiest to handle
  - Otherwise, rely on your physics engine