Lecture 5

Collision Detection and Proximity Queries (Part 1)

Fundamentals of Computer Graphics

Animation & Simulation

Stanford CS248B, Fall 2023

Profs. Karen Liu & Doug James

Announcements

- HW1: Due today (before midnight)
- P1: Pinball!
  - Due on Tuesday Oct 24
    - Start early
  - Leaderboard demo
    - https://openprocessing.org/sketch/2036124
  - Ray-marching ball collisions
    - Continuous collision checks (Thursday)
- Document:

  https://docs.google.com/document/d/1hSuawgrm4JGrTjaGGljJRwNsl6i9aXWZ5Ry7t89SXoc/edit?usp=sharing
- 248B Pinball Startercode

  https://openprocessing.org/sketch/1651988
  -

Stanford CS 248B Fall 2022 – Programming Assignment #1

Pinball!

Colliding Particles for Fun

Summary

In your first programming assignment you will design, model, simulate and play a virtual game of pinball. The main challenge is to simulate the ball motion subject to collisions with an environment composed of rigid obstacles (mostly static but some moving). The environment will be modeled using simple 2D shape primitives, for which we can use signed-distance fields (SDFs) to help process collisions robustly, even for fast-moving balls. Your implementation will be done in Open Processing (OP), and you and your classmates will be able to share and play each others' games (without sharing the code). Yes, this will be a groovy start to CS248B.

https://en.wikipedia.org/wiki/Pinball

2D Particle Simulation

For this first assignment, we will model the ball using a simple 2D point-like circular primitive (with unit mass, m=1, radius r, position **p**, and velocity **v**) falling under a small gravitational acceleration, **g**. This assumption ignores the more complex dynamics of a 3D spherical balling with rolling and frictional contact. The main challenge is not the time-stepping of the free-flight motion, but the detection and robust processing of ball-object collisions. As discussed in class, you can use a Symplectic Euler integrator for the ball dynamics, and use impulses to modify the velocity to resolve collisions. In difficult cases, such as for fast-moving balls, you will need to perform continuous collision processing to avoid missing collisions or having objects interpenetrate (more on that later).

The starter code includes a `Ball class` that can `draw()` a circle, and has a `timestep(dt)` method that performs discrete collision detection using the scene's signed-distance field, and applies a suitable collision impulse. You will need to modify the time-stepping scheme to handle collisions more robustly, as discussed below.

Today's Class: Detecting Collisions

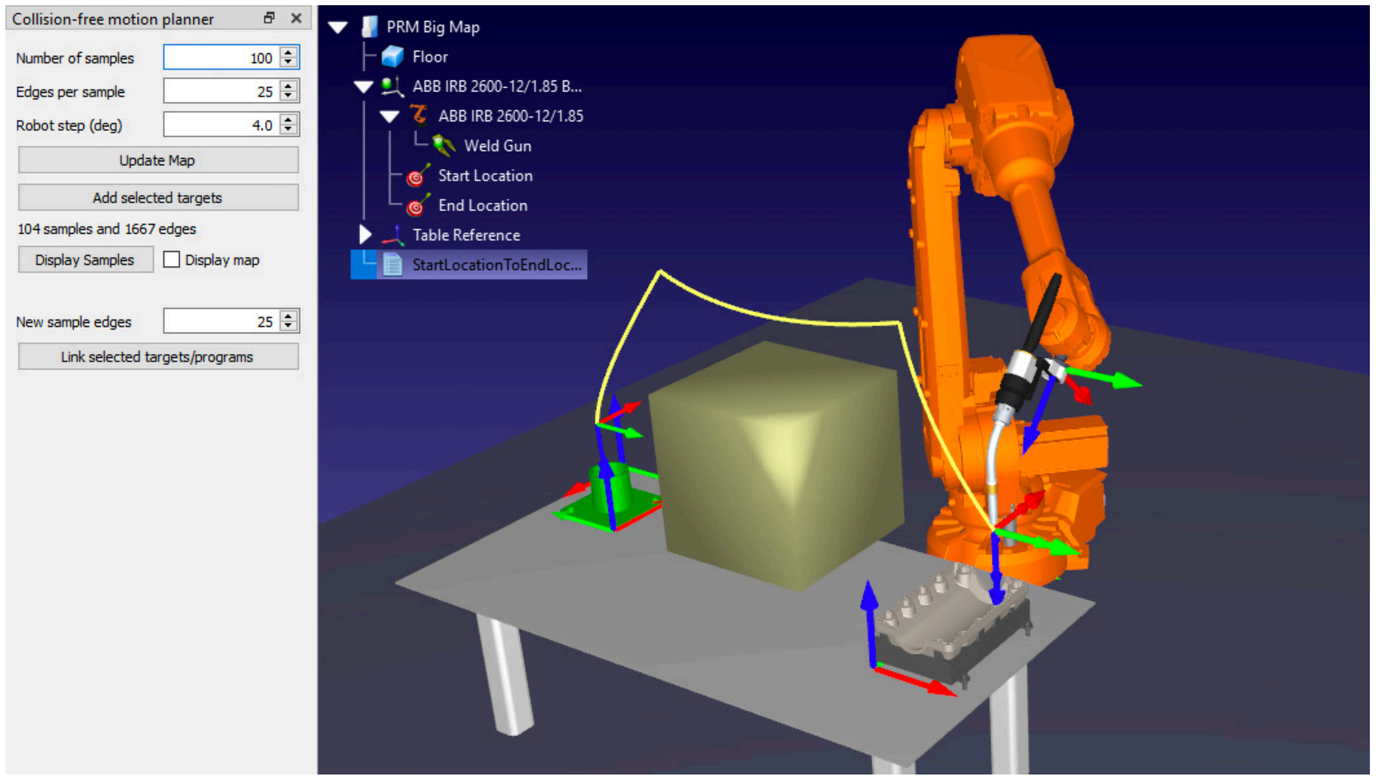
https://twitter.com/TansuYegen/status/1578277990320197632?t=Lr8yFGPgqCba7y5R5InNiA&s=19

Motivation: Collision detection

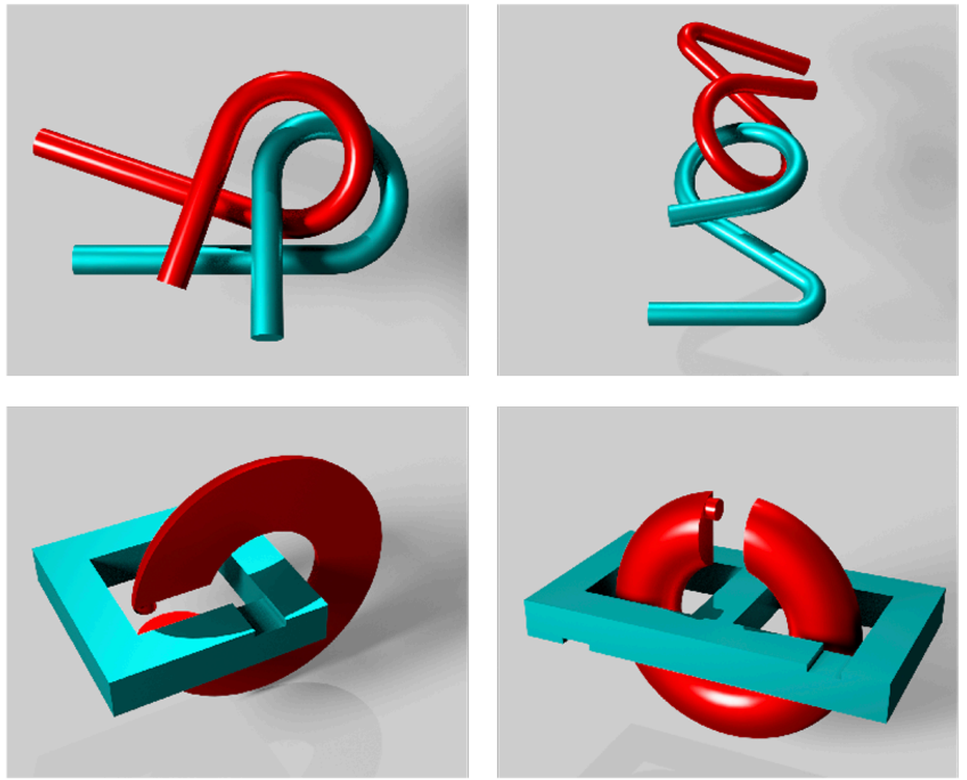
Determine if two triangle meshes collide

Example: Motion planning

- Verify that a motion plan is free of collisions



https://robodk.com/blog/motion-planning-trend/



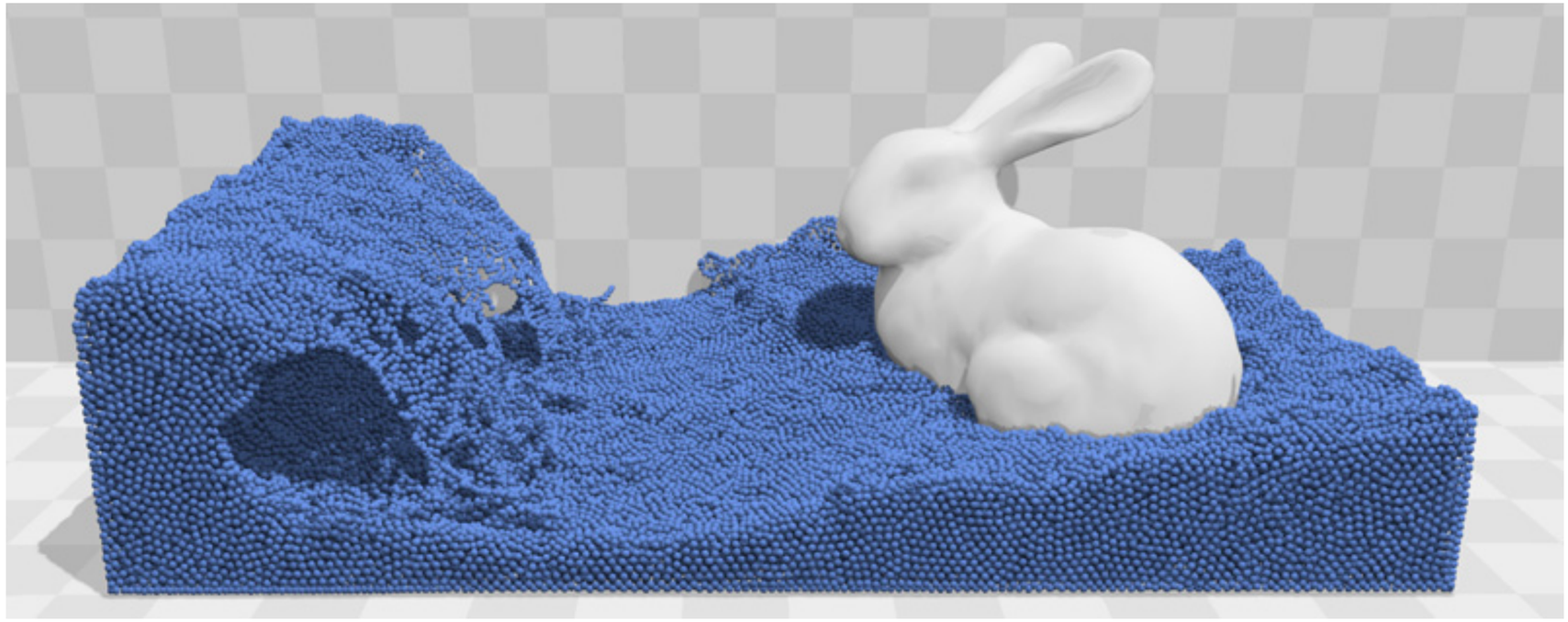
https://xinyazhang.gitlab.io/puzzletunneldiscovery/

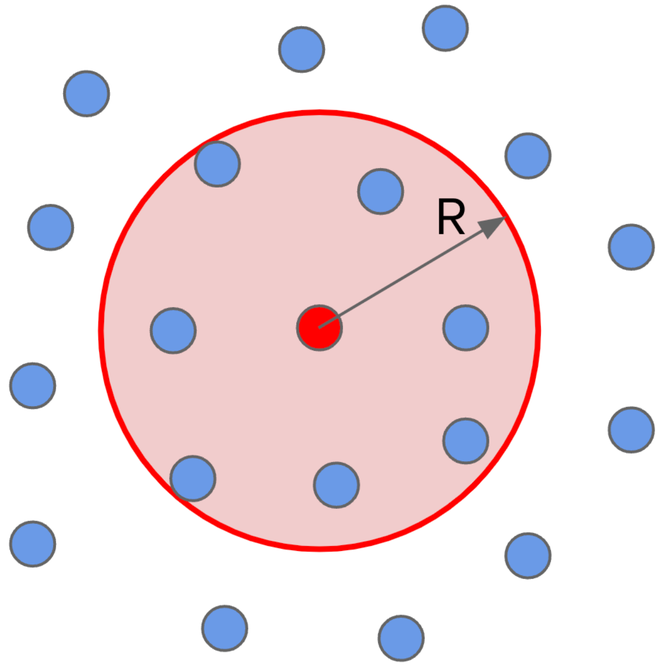
Motivation: Proximity queries

Find neighboring particles within a distance cutoff, R

Example: Particle-based fluid simulation

- Particle-particle interactions only occur within some small distance, R.





Position Based Fluids [Macklin and Mueller 2013]

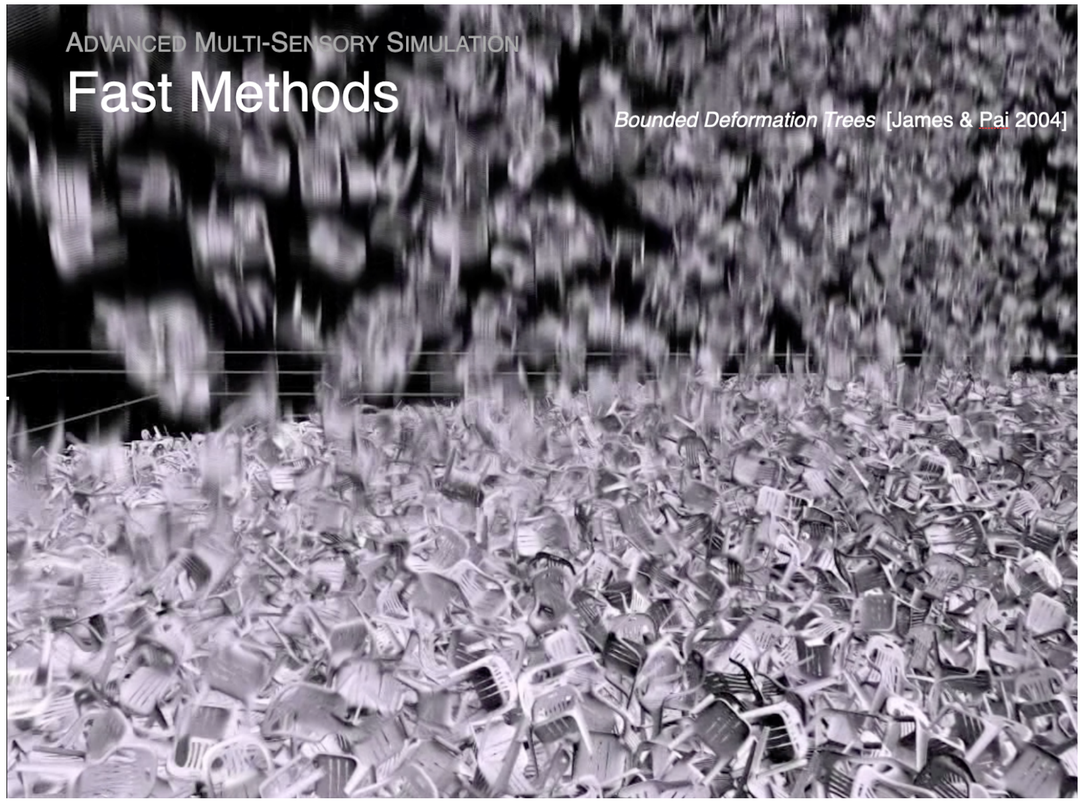
Collision Detection and Proximity Queries

Fast computer methods for

- Proximity and distance queries
- Penetration depth calculation
- Fast neighbor finding
- Overlap tests, e.g., triangle-triangle test
- Discrete and continuous collision tests
- Ray intersection tests (rendering → 248A)

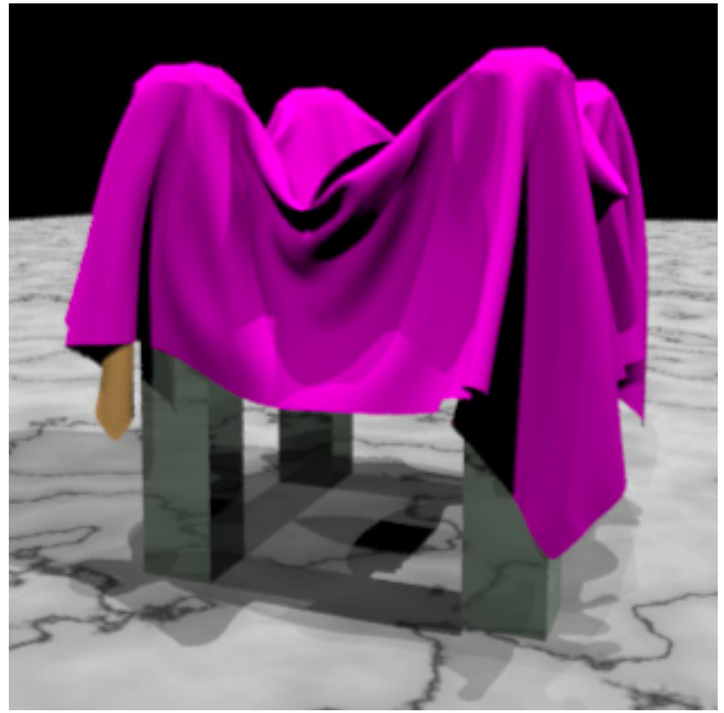
Many interesting data structures:

- Geometric tests
- Bounding volumes and related hierarchies
- Spatial subdivisions (uniform and adaptive grids)
  - Sorting and hashing
  - Distance fields



Overview

- Concepts:
  - Overlap & proximity queries
  - Broad & narrow phases
- Narrow phase
  - Primitive computations
  - Separating Axis Theorem
  - Convex bounds (Sphere, AABB, OBB, k-DOP, convex hull)
    - Algorithms for fitting and querying the bounds
- Broad phase
  - Brute-force all-pairs CD and the need for culling
  - Spatial subdivisions. Spatial hashing.
    - Spatial trees (quadtree/octree, kd-tree, etc.)
  - Bounding volume hierarchies (BVHs)
    - Sphere, AABB, OBB
    - Fitting and traversal
- More advanced topics (later)
  - Hybrid approaches
  - Updating data structures for moving and deforming geometry
  - Self-collision detection
  - Advanced culling strategies
  - Continuous collision detection



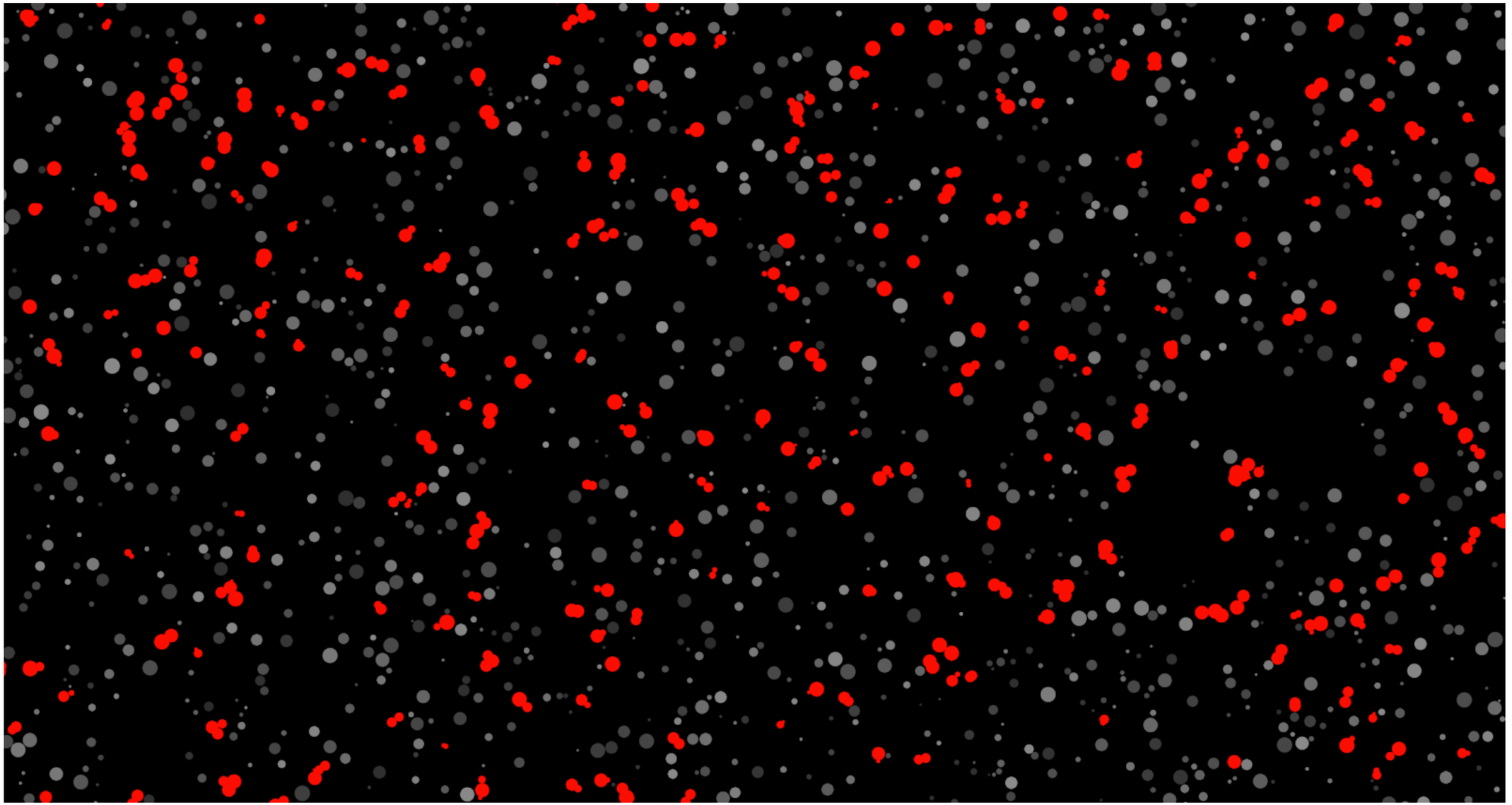
[Bridson et al. 2002]

Wide range of queries and methods

- Interesting area due to wide range of inputs & outputs, types of algorithms, hardware, and performance constraints, e.g., off-line vs time-critical queries.
- Example queries
  - Overlap test with boolean output
  - Overlap test with output list of colliding primitives
  - Distance between two objects
  - Penetration depth between two objects
  - K-nearest primitives
  - Objects within distance, d
  - Closest feature(s)
  - Farthest point

Fundamental Problem:

Pairwise calculations are $O(N^2)$ work



Best case?

Worst case??

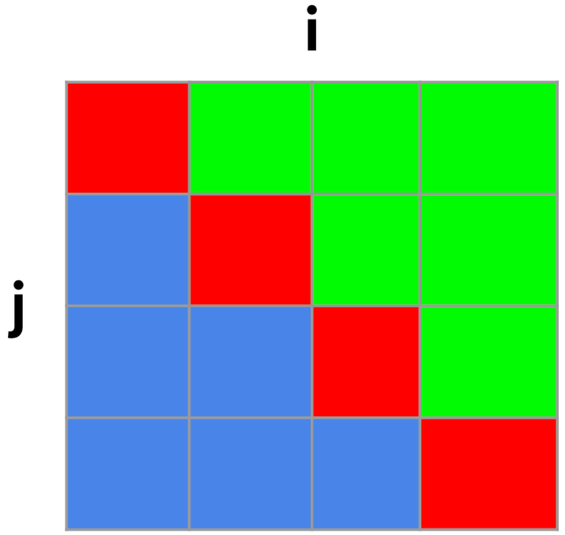
Fundamental Problem:

Pairwise calculations are O(N^2) work

Double-loop calculations don't scale well for large numbers of particles, N

Naive all-pairs loop over N^2 entries:

```
for(i = 1 .. N)
   for (j = 1 .. N)
      processPair(i,j);
```



- N diagonal (i=j)
- N^2-N off-diagonals (i≠j)
- (N^2-N)/2 unique pairs

Better: Only the (N^2-N)/2 unique pairs:

```
for(i = 1 .. N)
   for (j = i+1 .. N)
      processPair(i,j); // unique pair, i<j
```

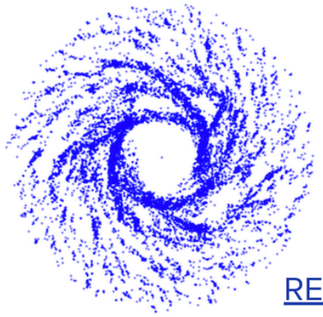
Fundamental Problem:

Pairwise calculations are $O(N^2)$ work

Double-loop calculations don't scale to large number of particles, N

Sometimes there are $O(N^2)$ non-zero interactions:

- E.g., N-body gravitational forces
- Deal with it using different strategies:
  - Parallel computing
  - Fast-summation algorithms (Barnes-Hut, fast multipole)



REBOUND

But in many problems, most interactions are inactive and can be "culled"

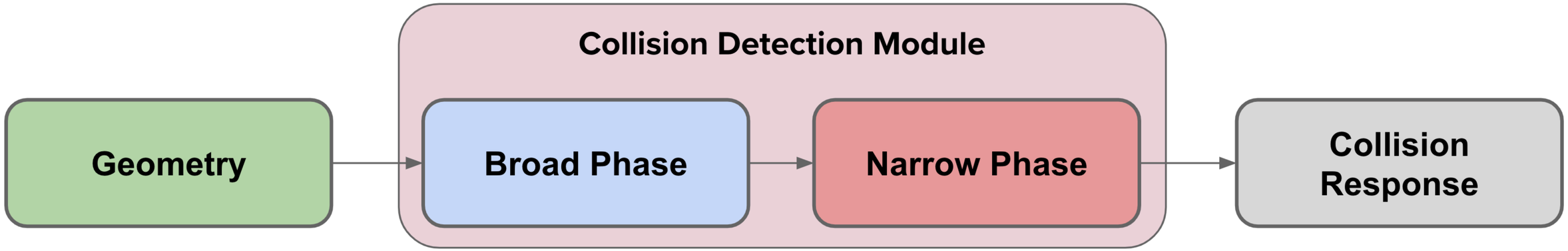
E.g., expect #contacts proportional to #bodies in typical cases

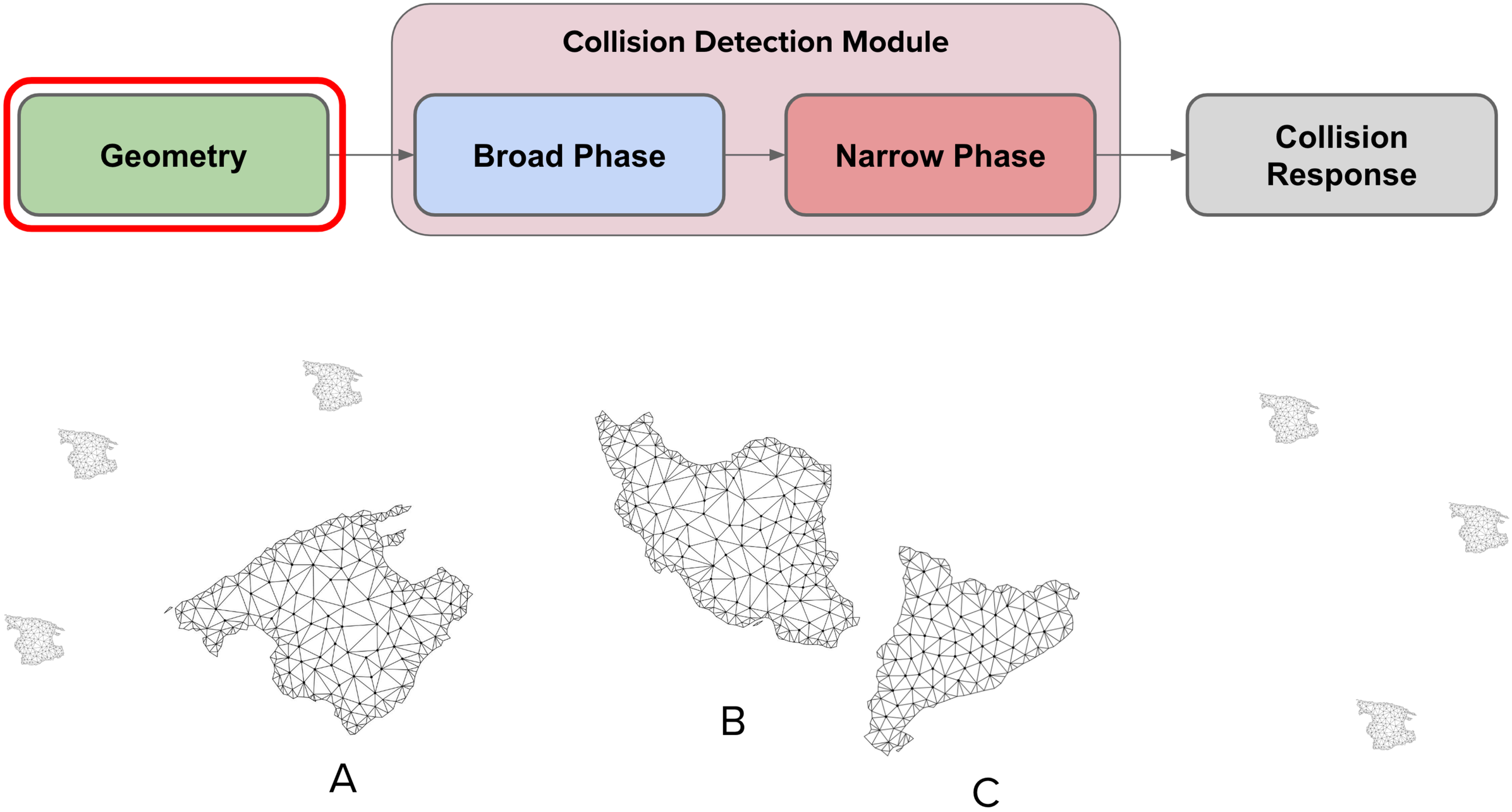
- O(1) contacts per body → O(N) contacts for N bodies

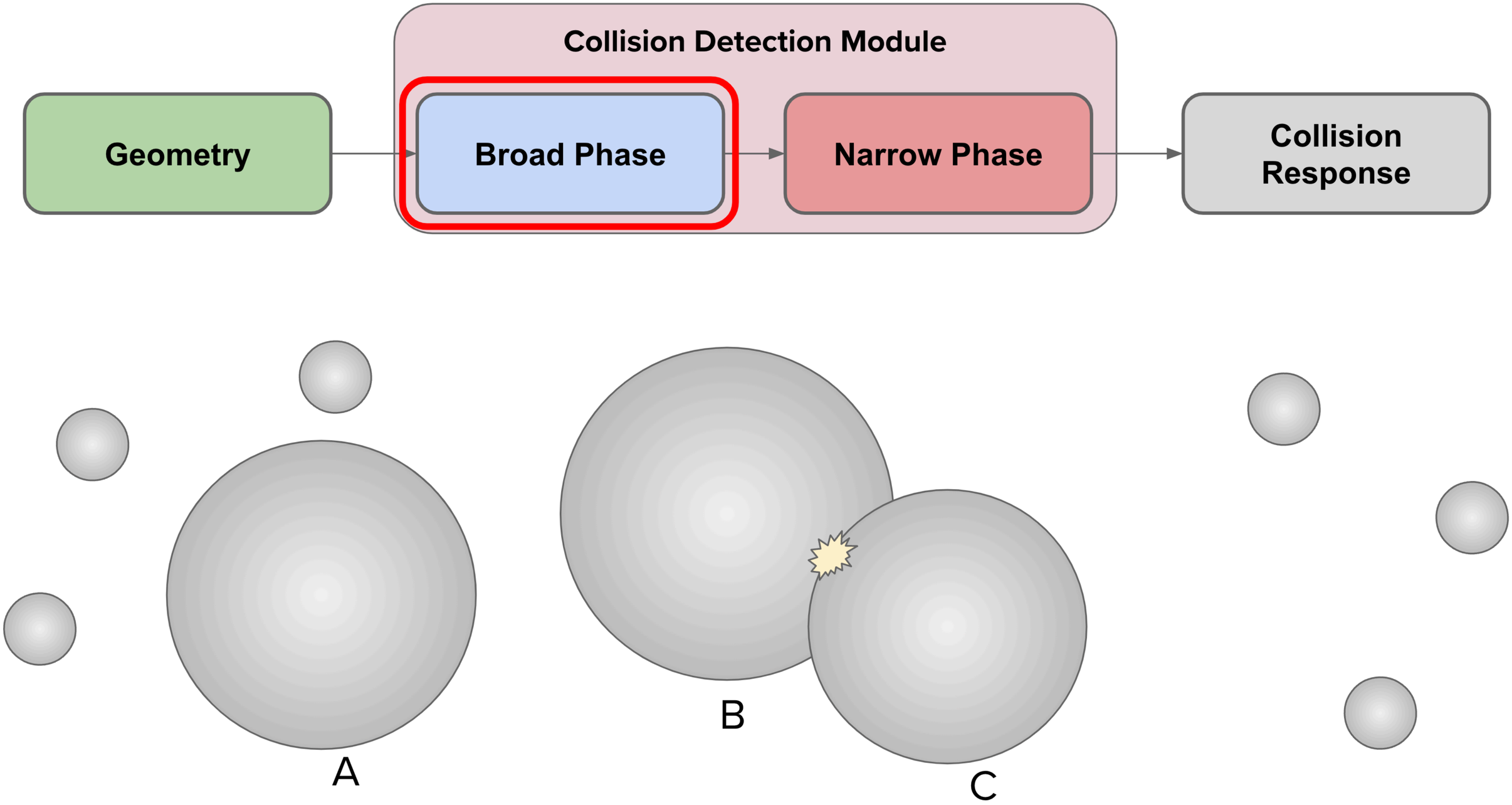
Strategy: Cull inactive contributions, e.g., distant bodies in collision detection.

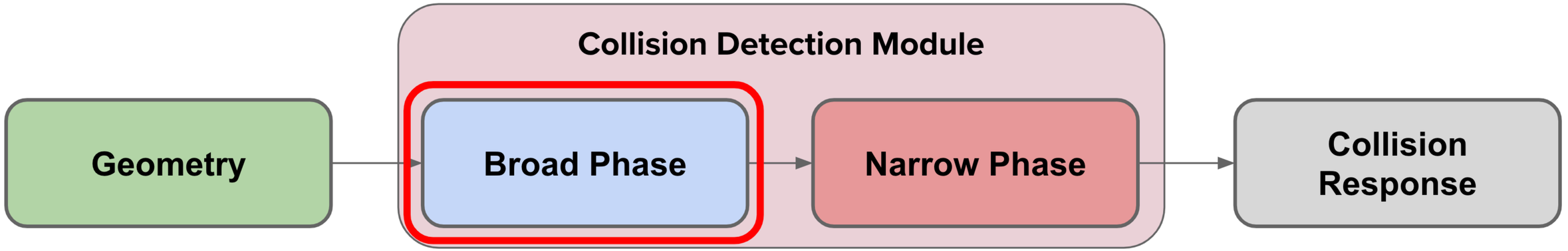
How? Many approaches.

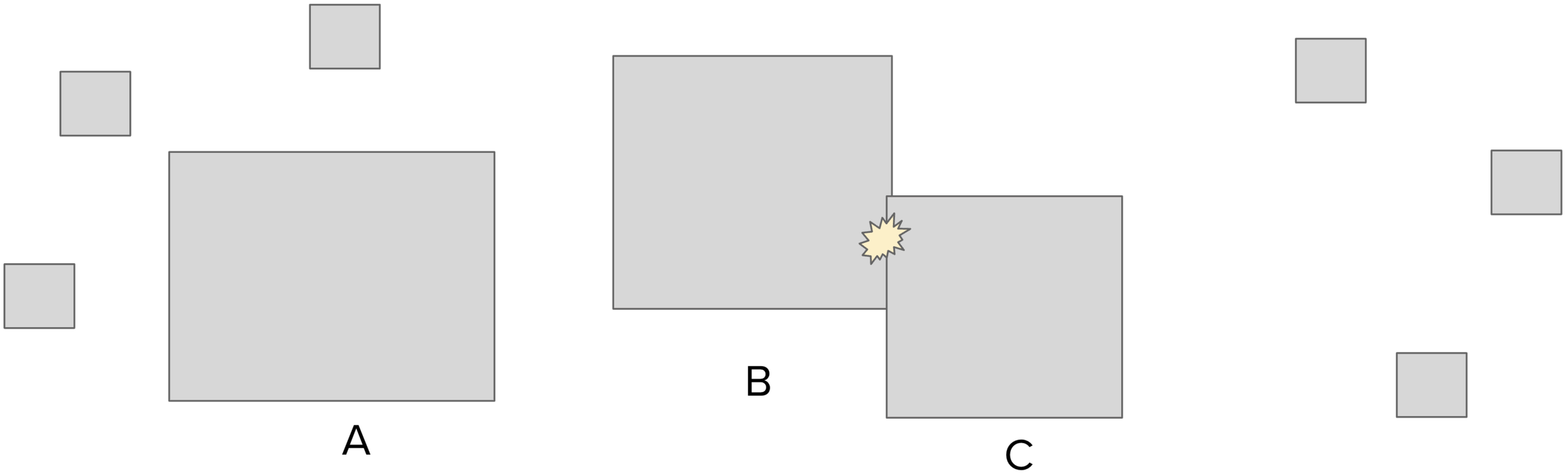
Broad and Narrow Phase CD

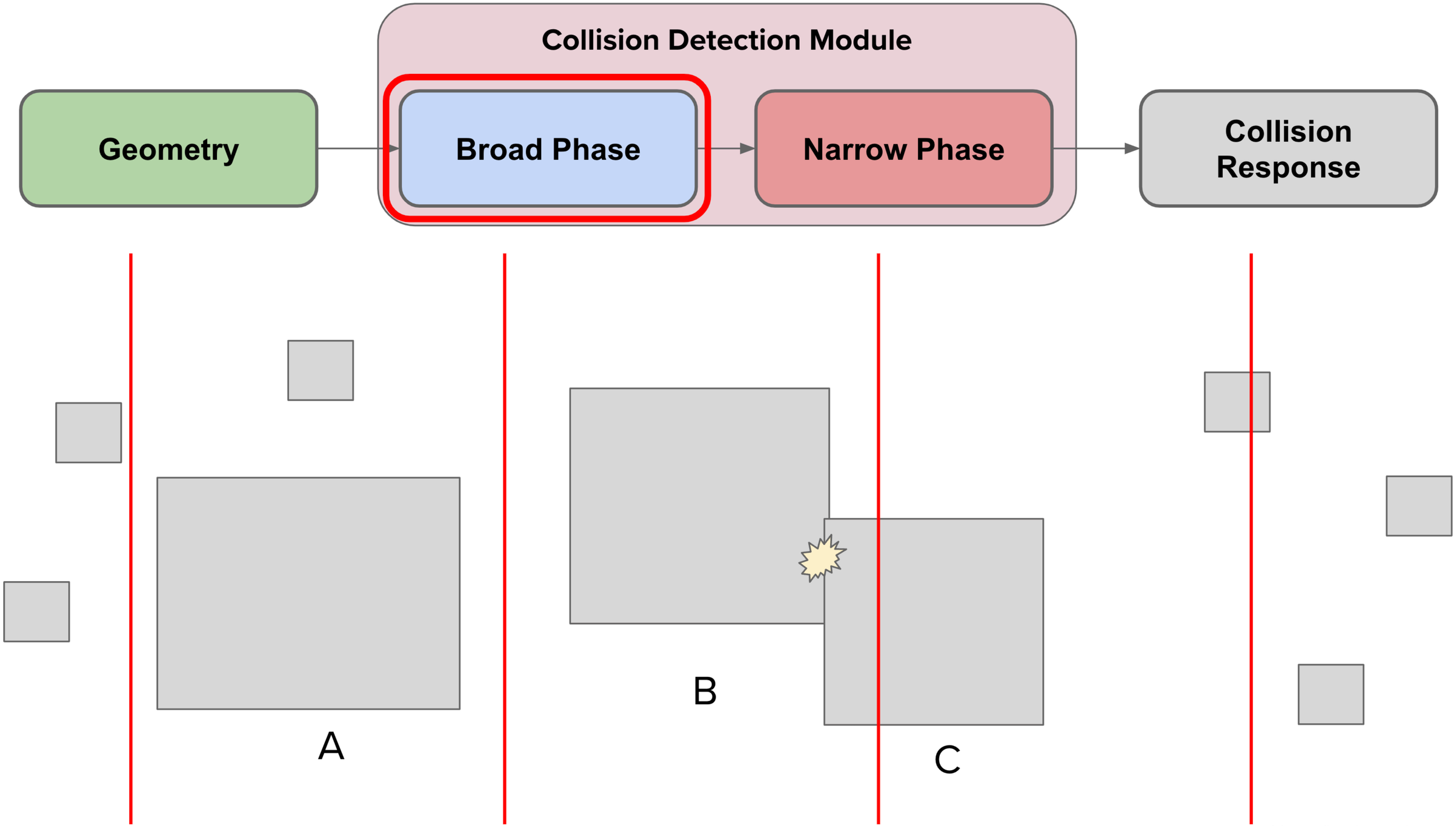


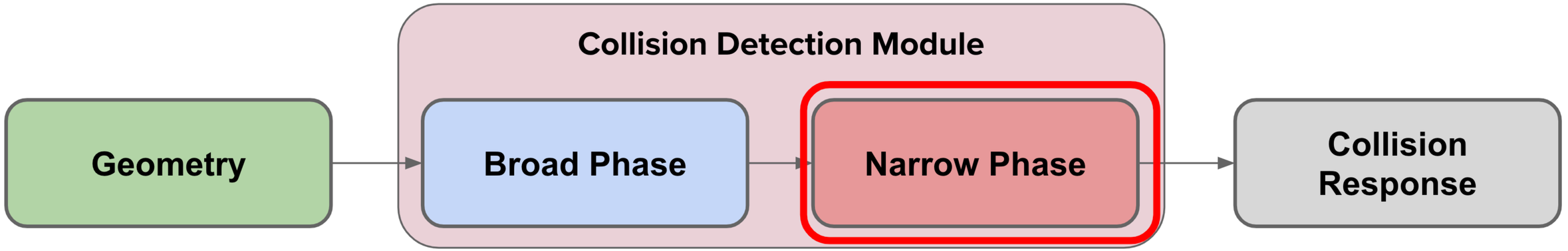


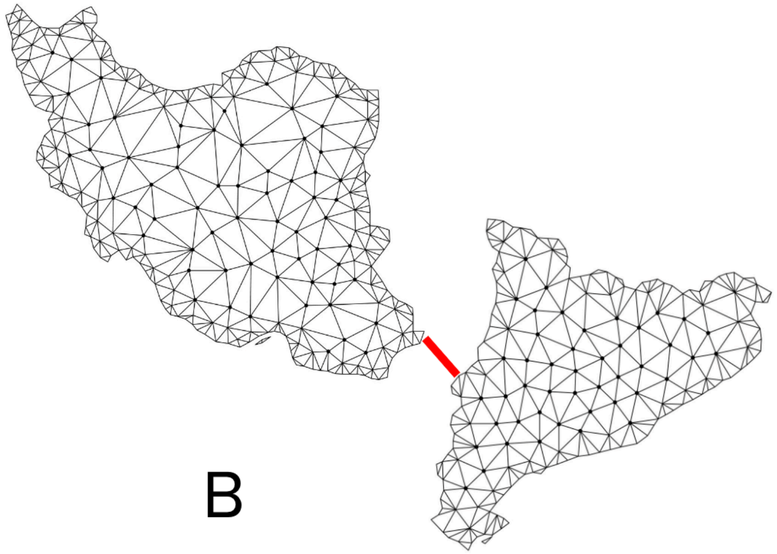












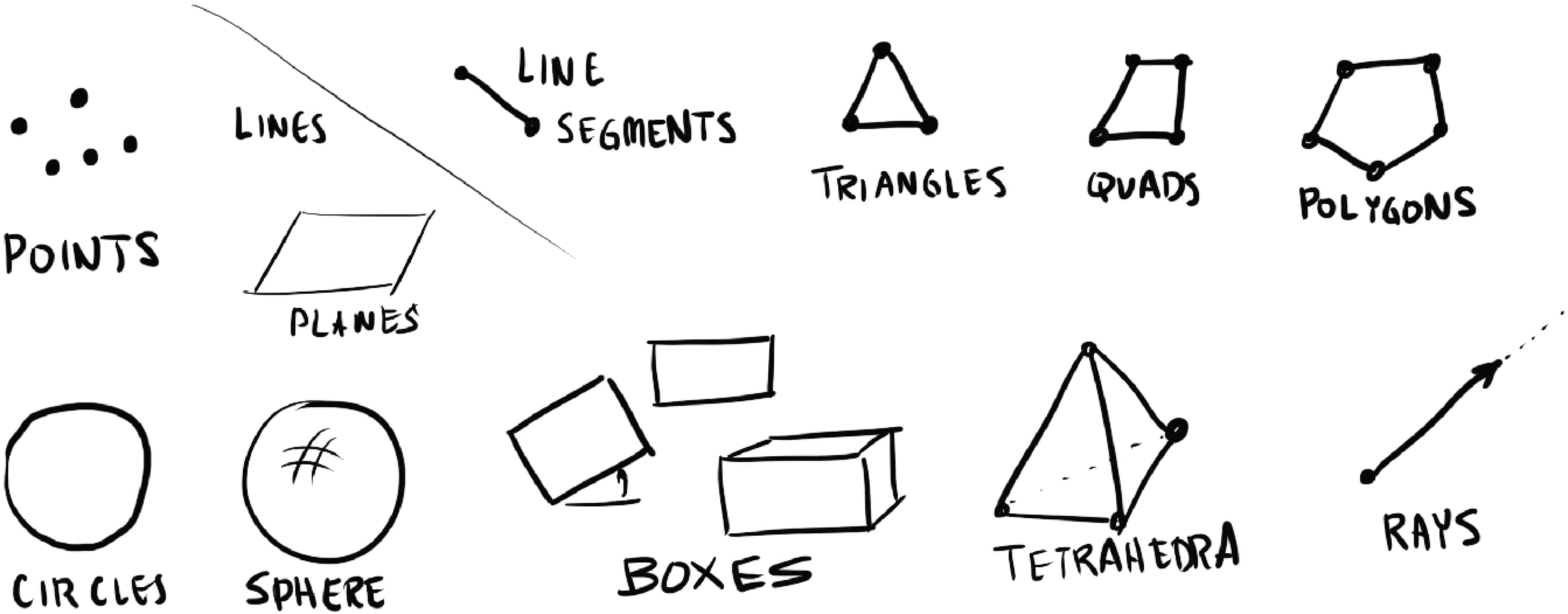
Broad and Narrow Phase CD

- Narrow phase
  - Given two objects, perform a CD query on them
  - EX: Determine if two triangles overlap.
- Broad phase
  - Given N objects, determine which pairs must be processed
  - Goal is to avoid brute-force all-pairs $O(N^2)$ checks with narrow phase code
  - EX: Given a mesh with N triangles, determine which triangles are within distance, d_0
- Often many phases in complex collision queries
  - Consider collision processing for 12001 plastic chairs modeled as triangle meshes
    - First find which chairs are close (macro-scale broad phase)
    - Second, process nearby chairs to find which triangles are close (meso-scale broad phase)
    - Finally, determine which triangle pairs are overlapping and where (narrow phase)

Narrow Phase Checks

Geometric Primitives

- Many primitive types in nD.

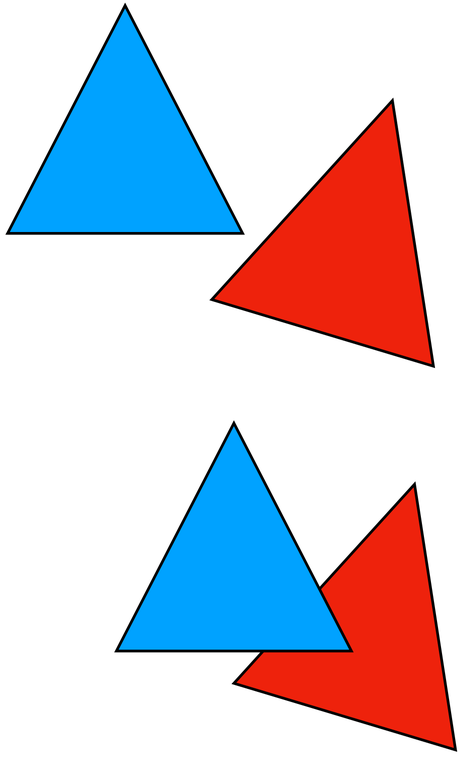


Pairwise primitive CD/proximity computations

- Overlap
- Distance
- Penetration depth
- Nearest feature(s)

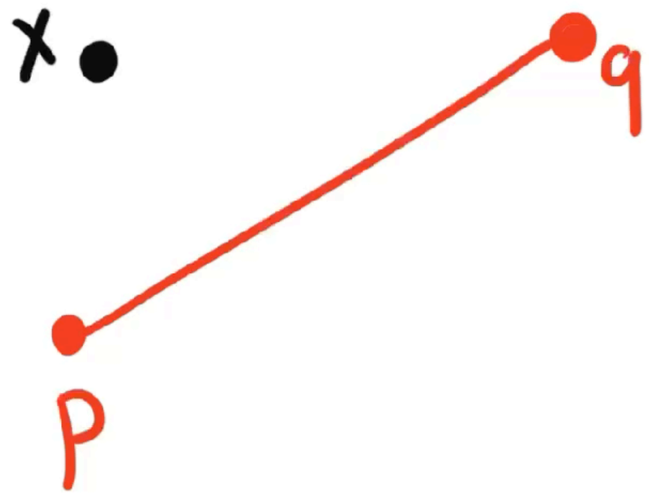
Overlap and intersection tests

- Determine if shapes or primitives (triangles, rays, etc.) overlap
- Different queries possible:
  - Determine if any overlap occurs
    - Return true or false, e.g., boolean collision check
  - Determine if and where overlap occurs
    - E.g., "contact points" for collision processing
    - E.g., "intersection point" for ray-triangle intersection
- Overlap tests:
  - Usually multiple approaches.
  - Different trade-offs in speed and robustness of floating-point implementations.



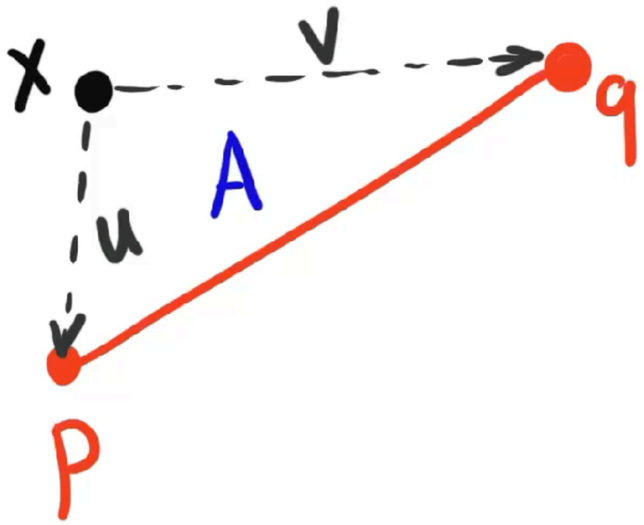
2D Warm-up Example: Are 3 points collinear?

- Numerical implementation is important
- Simple test: Check to see if area of triangle (p,q,x) is zero



2D Warm-up Example: Are 3 points collinear?

- Numerical implementation is important
- Simple test: Check to see if area of triangle is zero

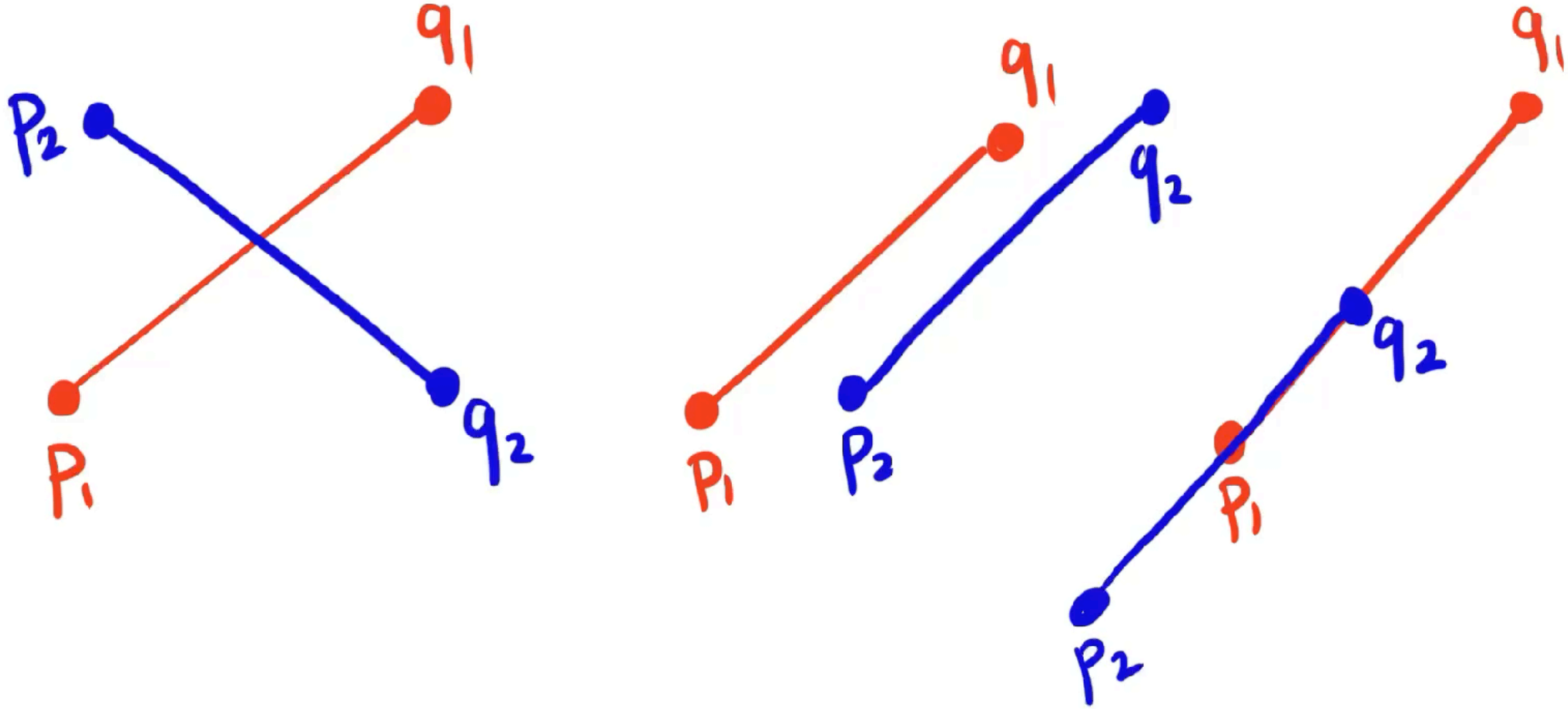


$$A = \frac{1}{2}\|u \times v\|$$

$$= \frac{1}{2}\begin{vmatrix} p_1 & q_1 & x_1 \\ p_2 & q_2 & x_2 \\ 1 & 1 & 1 \end{vmatrix}$$

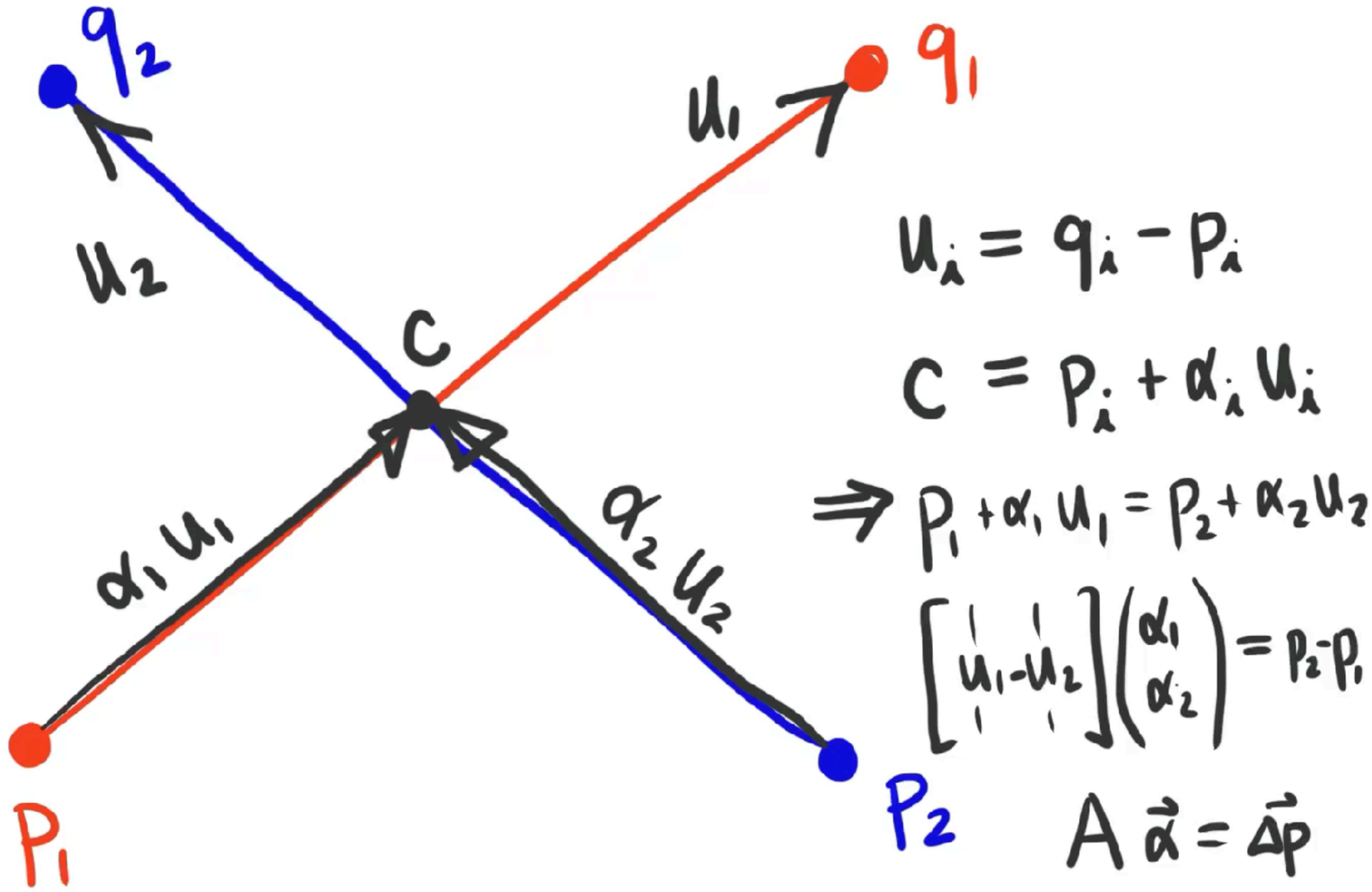
- Numerical issues when testing for A equal to 0?

2D Example: Overlapping line segments



2D Example: Overlapping line segments

- Approach #1: Compute line intersection point, c, and see if it is on the line segments.



$$u_i = q_i - P_i$$

$$c = P_i + \alpha_i u_i$$

$$\Rightarrow P_1 + \alpha_1 u_1 = P_2 + \alpha_2 u_2$$

$$\begin{bmatrix} | & | \\ u_1 & -u_2 \\ | & | \end{bmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = P_2 - P_1$$

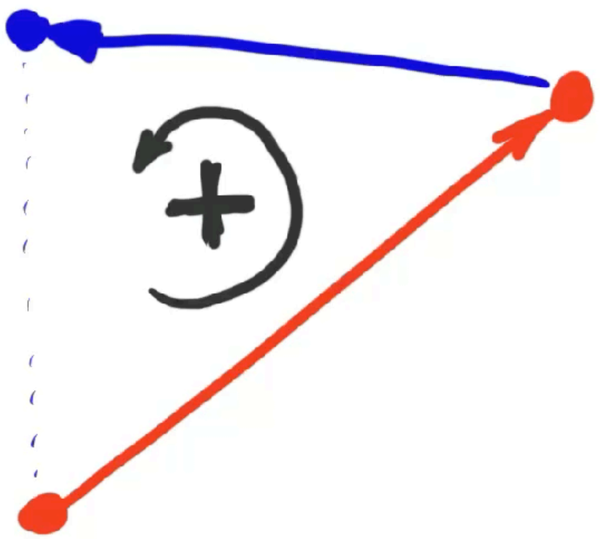
$$A\vec{\alpha} = \vec{\Delta p}$$

- Check to see if α_1 and α_2 are on [0,1]
- What can go wrong?

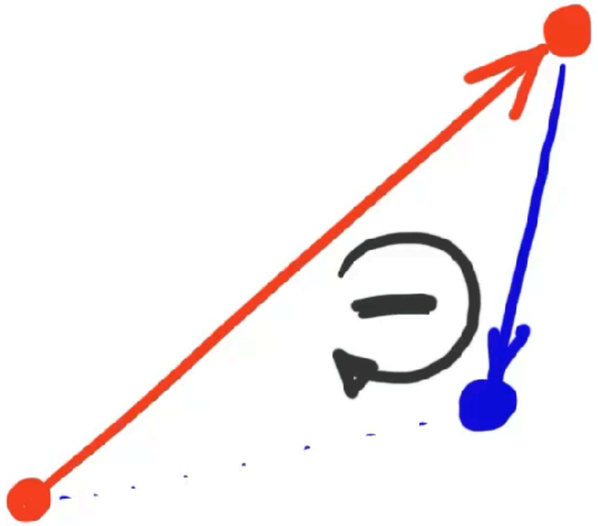
2D Example: Overlapping line segments

- Approach #2: Check orientation of triangles formed by 4 triangles (just sign tests)

A sign test for 2D triangle orientation (+ or -)

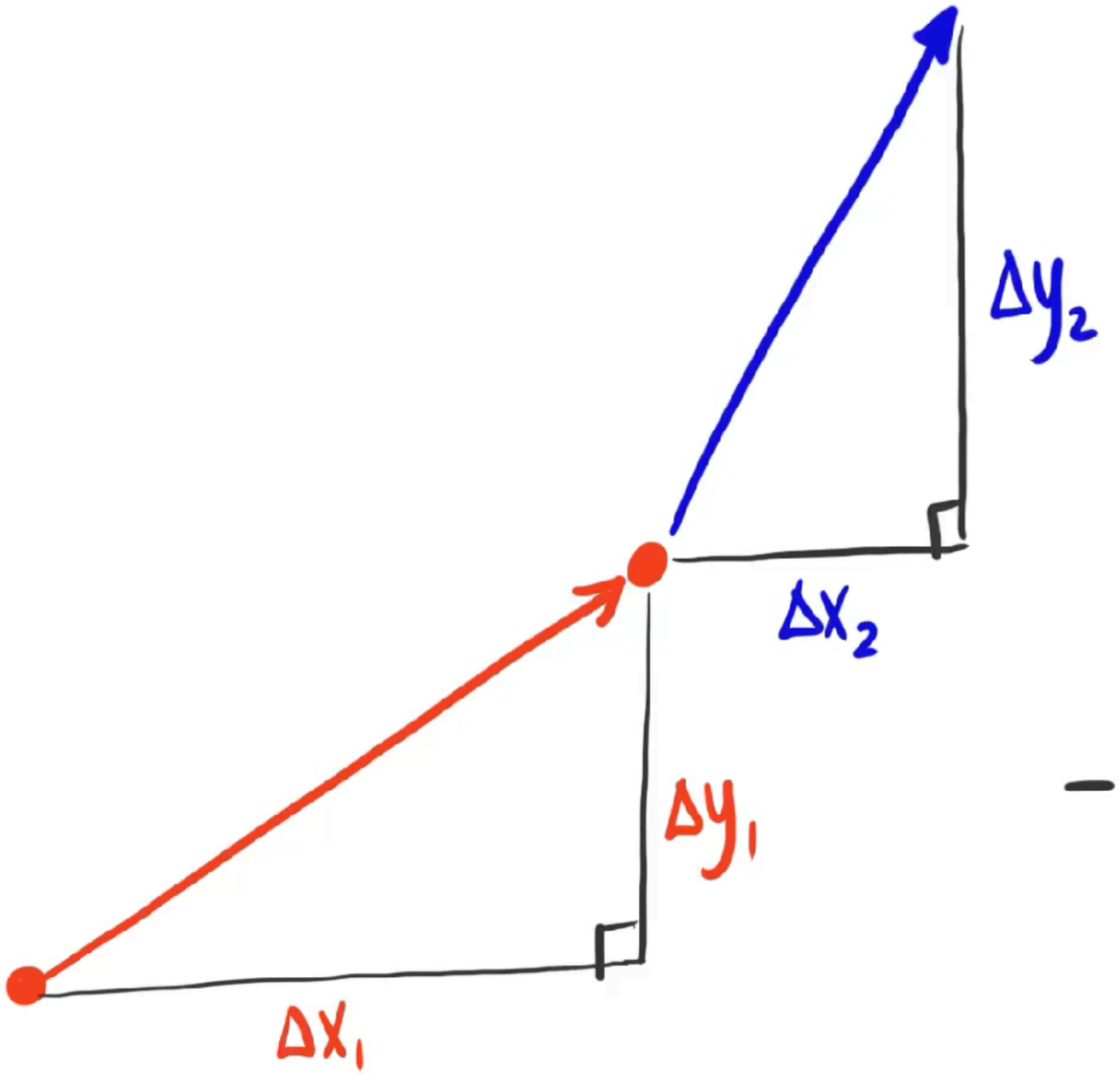


POSITIVE ORIENTATION



NEGATIVE ORIENTATION

A sign test for 2D triangle orientation (+ or -)



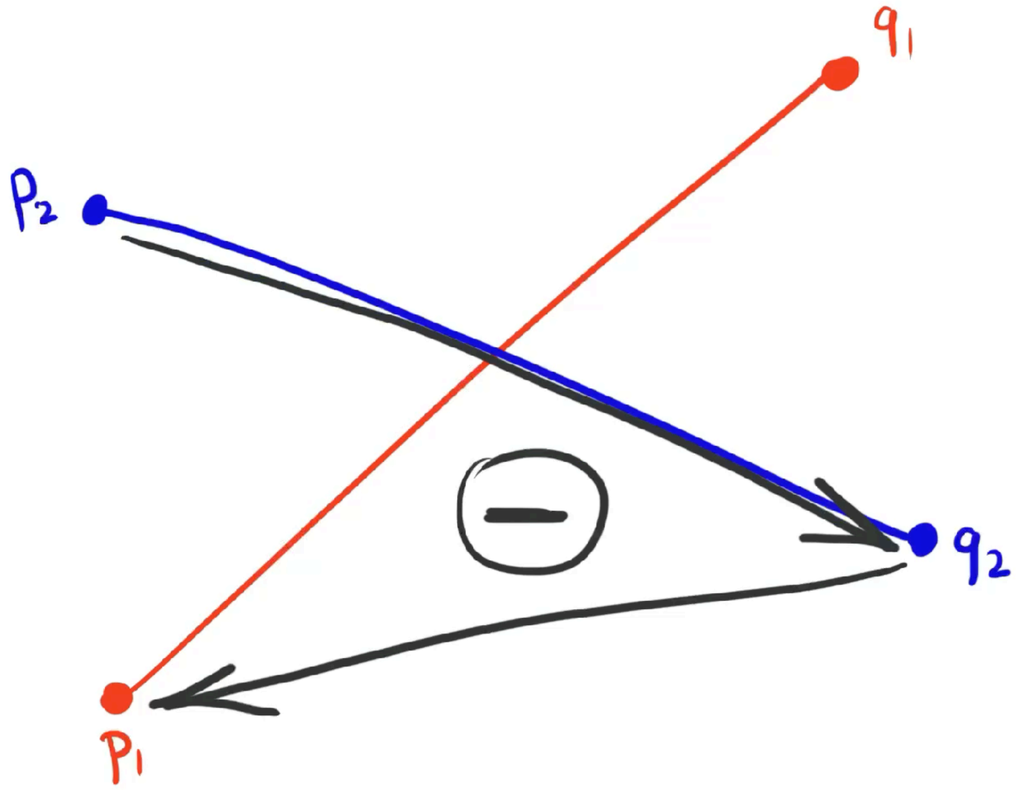
Check sign of

$$-\Delta y_1 \Delta x_2 + \Delta y_2 \Delta x_1$$

- What can go wrong?

2D Example: Overlapping line segments

- Approach #2: Check orientation of triangles formed by 4 triangles (just sign tests)
- General case (non-degenerate):
  - (p_1,q_1,p_2) and (p_1,q_1,q_2) have different orientations, *and*
  - (p_2,q_2,p_1) and (p_2,q_2,q_1) have different orientations.



- What can go wrong?

Many object intersection algorithms and codes available

- EX: https://www.realtimerendering.com/intersections.html

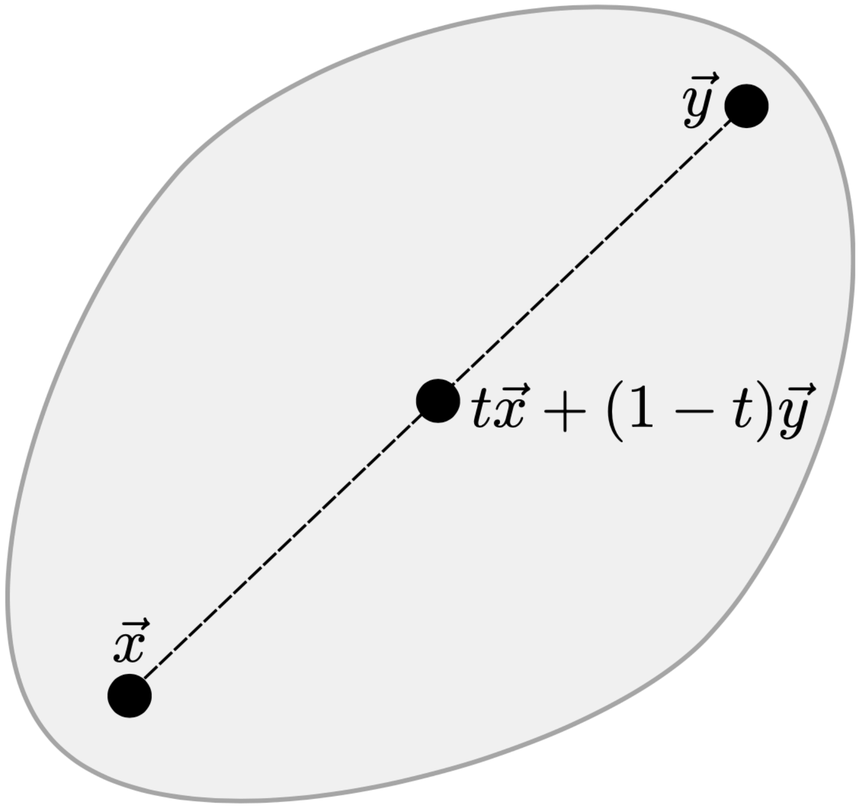
Static Object Intersections

Entries are listed from oldest to newest, so often the *last* entry is the best. This table covers objects not moving; see the next section for dynamic objects.

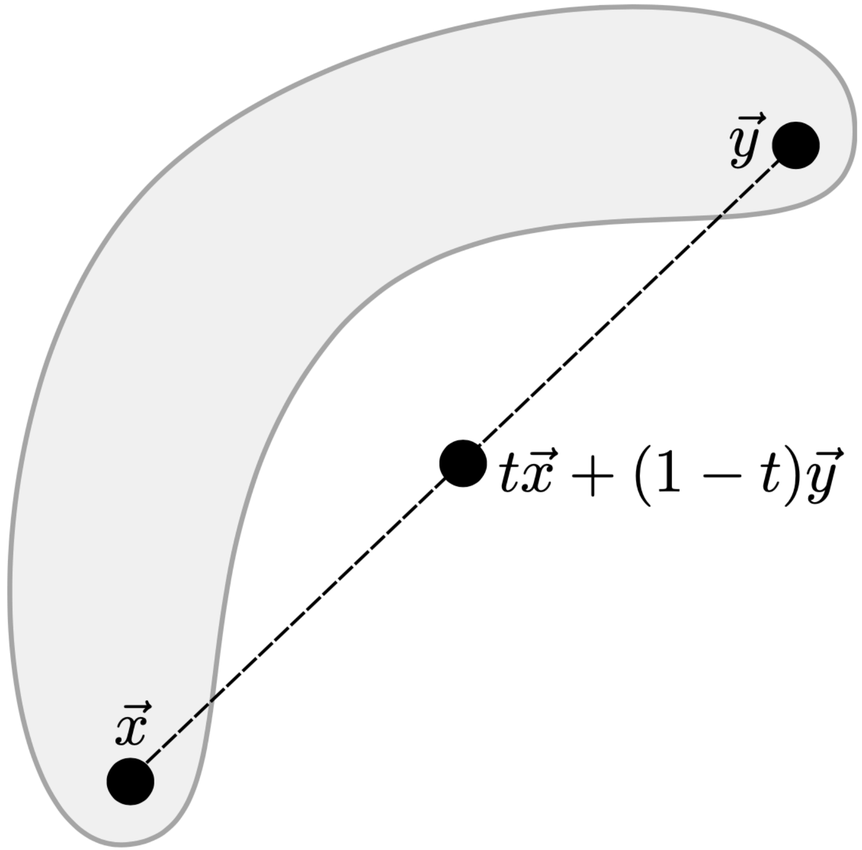
	ray	plane	sphere	cylinder	cone	triangle	AABB	OBB	frustum	polyhedron
ray	Gems p.304; SG; TgS; RTCD p.198; SoftSurfer: code; RTR4 p.989	IRT p.50,88; SG; GTCG p.482; TgS; RTCD p.175; SoftSurfer (more): code; GPC; Graphics Codex	IRT p.39,91; Gems p.388; Held jgt 2(4); GTweb; 3DG p.16; GTCG p.501; TgS; RTCD p.127,177; Graphics Codex; RTR4 p.955; GPC; Shadertoy (demo)	IRT p.91; Gems IV p.356; Held jgt 2(4); GTweb; GTCG p.507; TgS; RTCD p.194; Shadertoy (demo)	IRT p.91; Gems V p.227; Held jgt 2(4); GTweb doc; GTCG p.512	Möller-Trumbore jgt 2(1): code (mirror), paper draft; IRT p.53,102; Gems IV p.24; Held jgt 2(4); GTweb; 3DG p.17; Möller (mirror); GTCG p.485; TgS; RTCD p.153,184; Löfstedt jgt 10(2): code, paper draft Chirkov jgt 10(3): code; Lagae jgt 10(4): code, paper draft; SoftSurfer: code; Havel TVCG June 2009; Woop JCGT 2(1); Baldwin JCGT 5(3); GPC; Graphics Codex; RTR4 p.962; Shadertoy (demo)	IRT p.65,104; Gems p.395; Smits; 3DG p.20; Terdiman (optimized Woo); Schroeder; GTCG p.626; TgS; RTCD p.179; Mahovsky jgt 9(1); Williams jgt 10(1) (code); Eisemann jgt 12(4) (code); Shirley 2016; GPC; Chatzianastasiou; Majercik JCGT 7(3); RTR4 p.959; Wiche; Shadertoy (demo)	(IRT p.104; Gems II p.247); GTweb; Gomez; GTCG p.630; TgS; RTCD p.179; Shadertoy GPC; RTR4 p.960; (demo)	(IRT p.104; Gems II p.247)	IRT p.104; Gems II p.247; GTCG p.493; Platis jgt 8(4); RTCD p.198; SoftSurfer: code
plane	IRT p.50,88; SG; GTCG p.482; TgS; RTCD p.175; SoftSurfer (more): code; GPC; Graphics Codex	GTweb; SG; GTCG p.529; RTCD p.207; GPC	distance of center to plane <= radius; GTweb; Gomez; GTCG p.548; TgS; RTCD p.160,219; GPC	GTweb; GTCG p.551; TgS; GTweb doc	GTCG p.563; RTCD p.164	Check if all vertices are on one side;(Möller jgt 2(2)); GTCG p.534; SoftSurfer: code; GPC	Gems IV p.74; GTCG p.634; TgS; RTCD p.161,222; GPC; RTR4 p.970	GTweb; Gomez; GTCG p.635; TgS; RTCD p.161; GPC; RTR4 p.972		
sphere	IRT p.39,91; Gems p.388; Held jgt 2(4); GTweb; 3DG p.16; GTCG p.501; TgS; RTCD p.127,177; Graphics Codex; RTR4 p.955; GPC; Shadertoy (demo)	distance of center to plane <= radius; GTweb; Gomez; GTCG p.548; TgS; RTCD p.160,219; GPC	If radii A+B >= center/center distance; Held jgt 2(4); GTweb; Gomez; GPG p.390; GTCG p.602; TgS; RTCD p.88,215,223; Ellipsoids: GTweb doc; GPC; RTR4 p.976	If radii A+B >= center/axis distance; Held jgt 2(4); (GTCG p.602); TgS; RTCD p.114	GTweb; GTweb doc; (GTCG p.602); Hale	Held jgt 2(4); GTweb; Karabassi jgt 4(1); TgS; RTCD p.135,167,226; GPC	Gems p.335; Gomez; GTCG p.644; TgS; RTCD p.130,165,228; Ellipsoid: GTweb doc; GPC; RTR4 p.977	TgS; RTCD p.132,166 Larsson; GPC	GTweb; Assarsson; GPG p.433; 3DG p.302; GPC; RTR4 p.984	3DG p.462; RTCD p.142
(capped) cylinder	IRT p.91; Gems IV p.356; Held jgt 2(4); GTweb; GTCG p.507; TgS; RTCD p.194; Shadertoy (demo)	GTweb; GTCG p.551; TgS; GTweb doc	If radii A+B >= center/axis distance; Held jgt 2(4); (GTCG p.602); TgS; RTCD p.114	If radii A+B >= axis/axis distance; GTweb doc; GTweb doc; GTCG p.602,646; TgS; RTCD p.114	(GTCG p.602)	Held jgt 2(4); TgS; GTweb doc	TgS	TgS	GPG p.380	
(capped) cone	IRT p.91; Gems V p.227; Held jgt 2(4); GTweb doc; GTCG p.512	GTCG p.563; RTCD p.164	GTweb; GTweb doc; (GTCG p.602); Hale	(GTCG p.602)	(GTCG p.602)	Held jgt 2(4); GTweb doc; GTCG p.583	GTWeb doc	GTweb doc		
triangle (polygon)	Möller-Trumbore jgt 2(1): code (mirror), paper draft; IRT p.53,102; Gems IV p.24; Held jgt 2(4); GTweb; 3DG p.17; Möller (mirror); GTCG p.485; TgS; RTCD p.153,184; Löfstedt jgt 10(2): code, paper draft Chirkov jgt 10(3): code; Lagae jgt 10(4): code, paper draft; SoftSurfer: code; Havel TVCG June 2009; Woop JCGT 2(1); Baldwin JCGT 5(3); GPC; Graphics Codex; RTR4 p.962;	Check if all vertices are on one side;(Möller jgt 2(2)); GTCG p.534; SoftSurfer: code; GPC	Held jgt 2(4); GTweb; Karabassi jgt 4(1); TgS; RTCD p.135,167,226; GPC	Held jgt 2(4); TgS; GTweb doc	Held jgt 2(4); GTweb doc; GTCG p.583	Möller jgt 2(2); Held jgt 2(4); GTweb; Möller (mirror); GPG p.393; GTCG p.539; TgS; RTCD p.155,172; Shen jgt 8(1); Guigue jgt 8(1): code; SoftSurfer: code; GTweb doc; GPC; RTR4 p.972	Gems III p.236; Gems V p.375; TgS; RTCD p.169; Möller; GPC; RTR4 p.974	GTweb; GTweb doc TgS; GPC	homogeneous clipping: Gems p.84	generalized clipping

Bounding Volumes

Convex vs nonconvex shapes



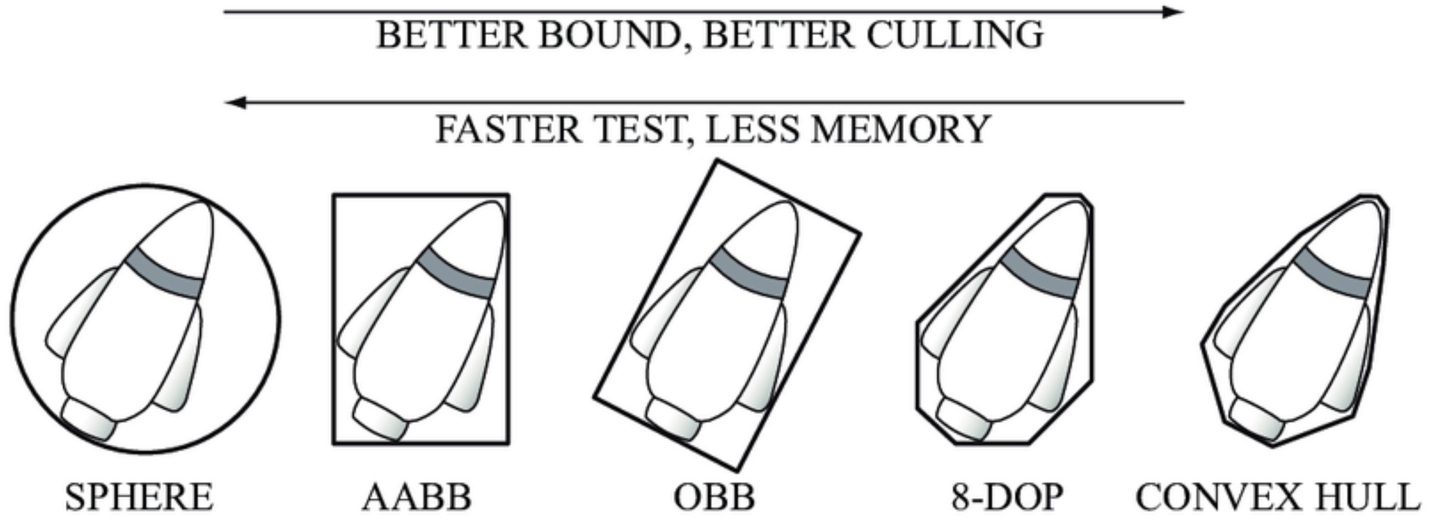
Convex



Nonconvex

Solomon, *Numerical Algorithms*

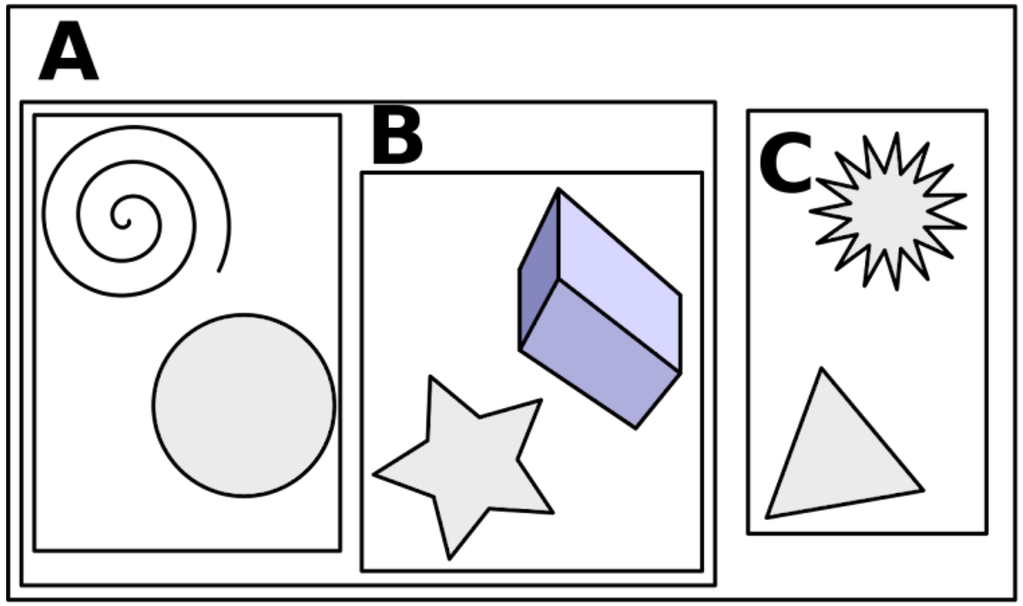
Bounding Volumes (BVs)

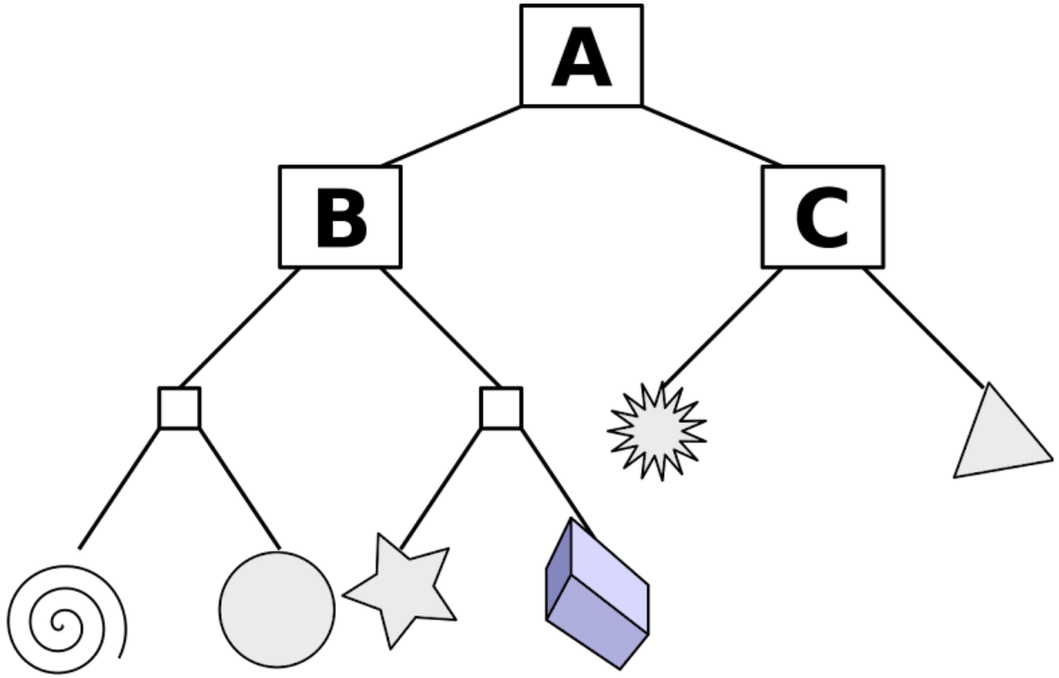


[Ericsson 2004]

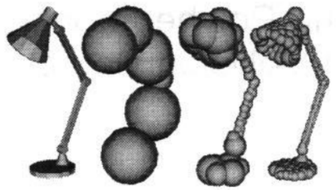
- Simple *convex* bounds.
- Fast & simple overlap tests:
  - False positives possible
  - False negatives impossible
- Fast methods to fit bounds exist, e.g., O(N) for sphere, AABB, 8-DOP.

Bounding Volume Hierarchies (BVHs)





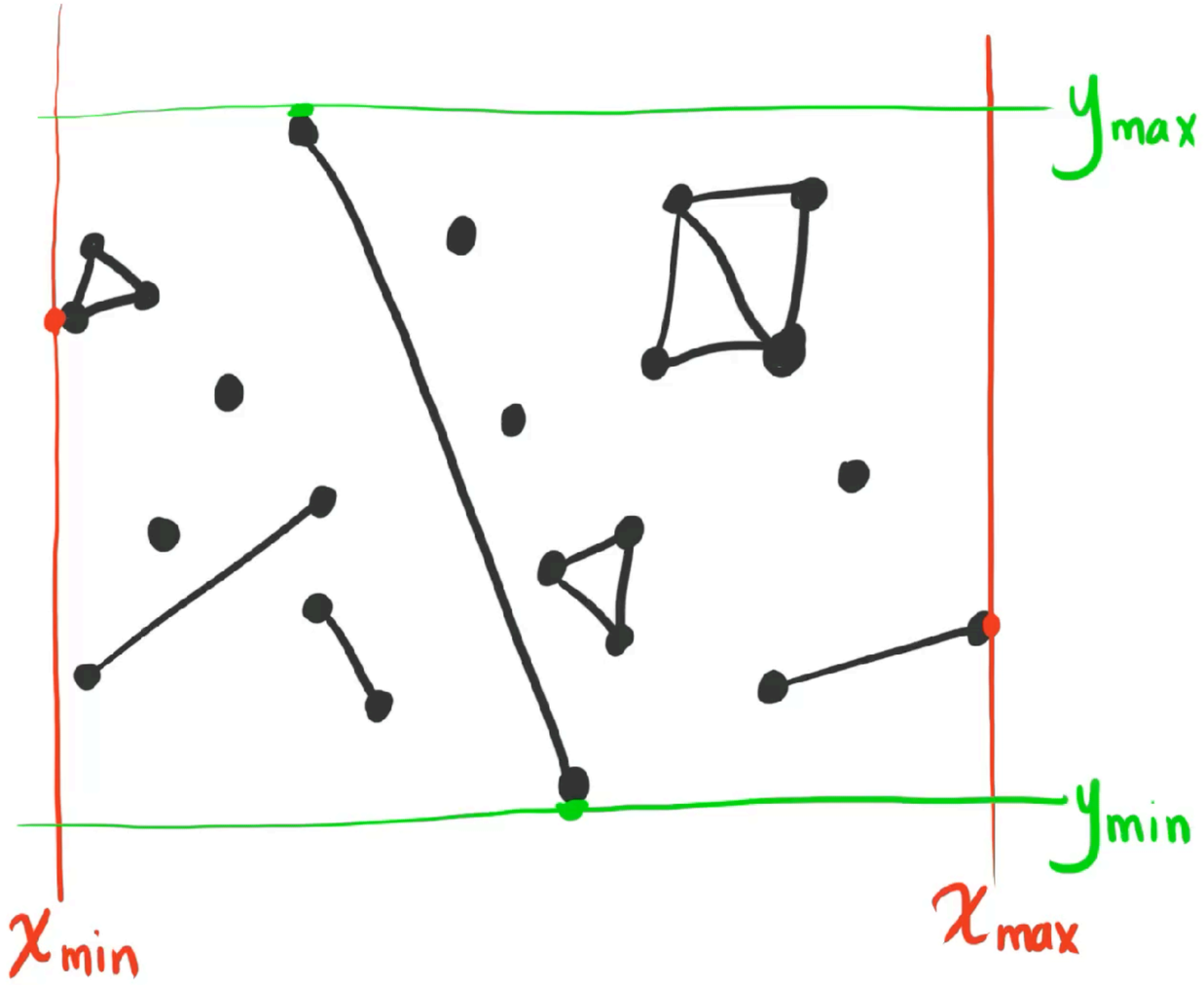
- **Common BVHs in graphics:**
  - **AABB Trees [van den Bergen 1998]: good for deformable models**
  - **OBB Trees [Gottschalk et al. 1996]: tighter fitting**
  - **Sphere Trees [Hubbard 1996]**



Fitting Bounding Volumes

Fitting BVs: Axis-aligned bounding box (AABB)

- 2D AABB is given by $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$
- Scan N primitives to find max/min along each axis.

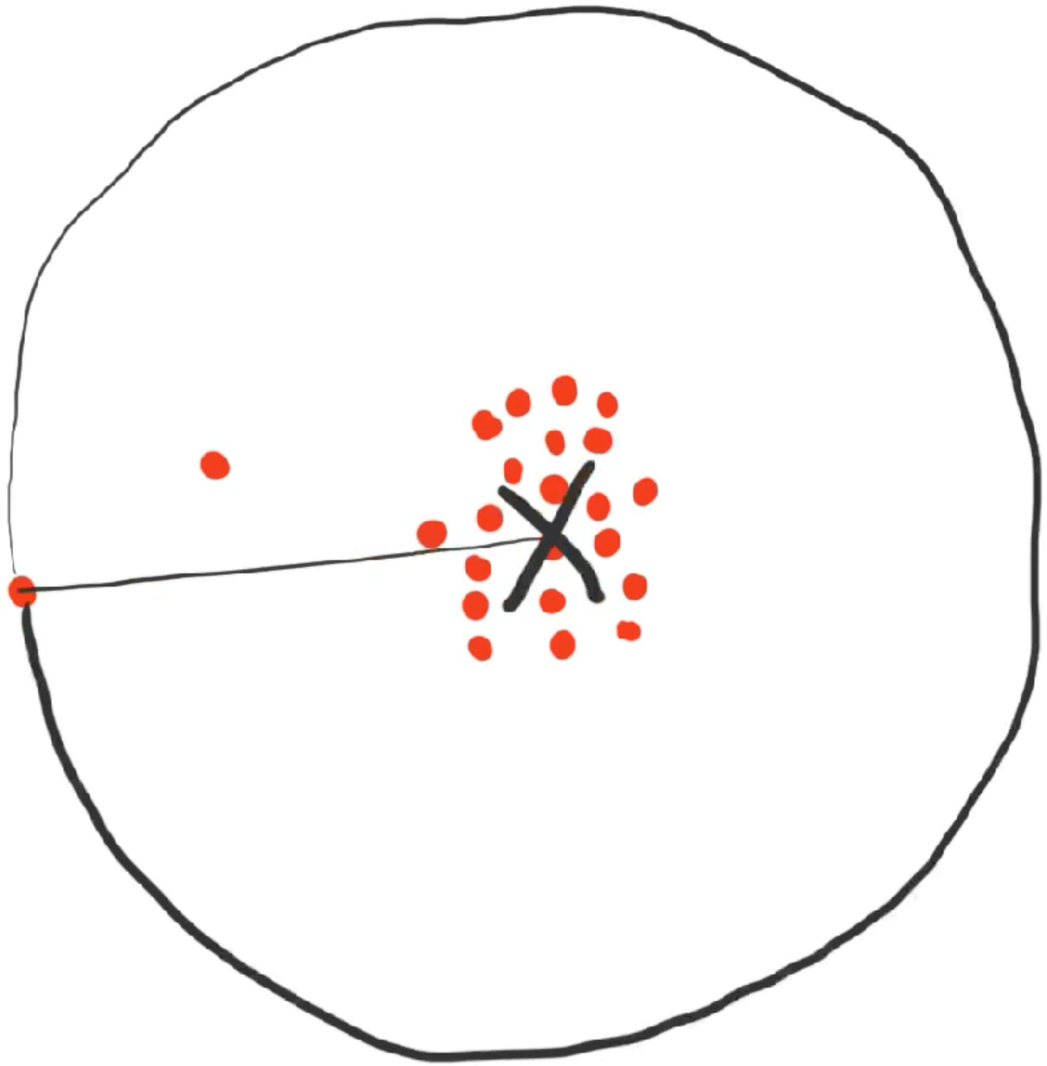


Fitting BVs: Bounding disks and spheres

- Approach #1 (simple): Find centroid of points, then compute minimum bounding radius
  - Problem?

Fitting BVs: Bounding disks and spheres

- Approach #1 (simple): Find centroid of points, then compute minimum bounding radius
  - Problem?

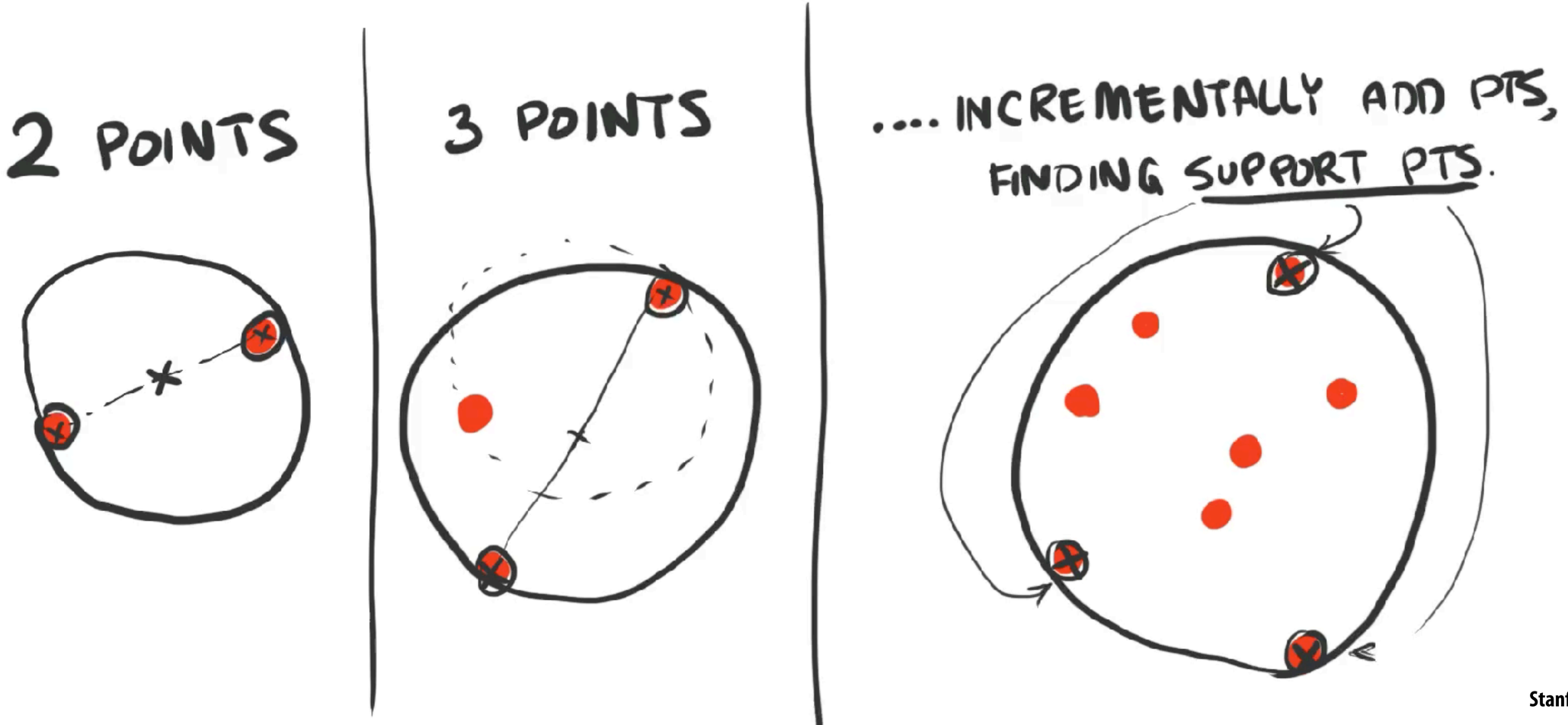


Fitting BVs: Bounding disks and spheres

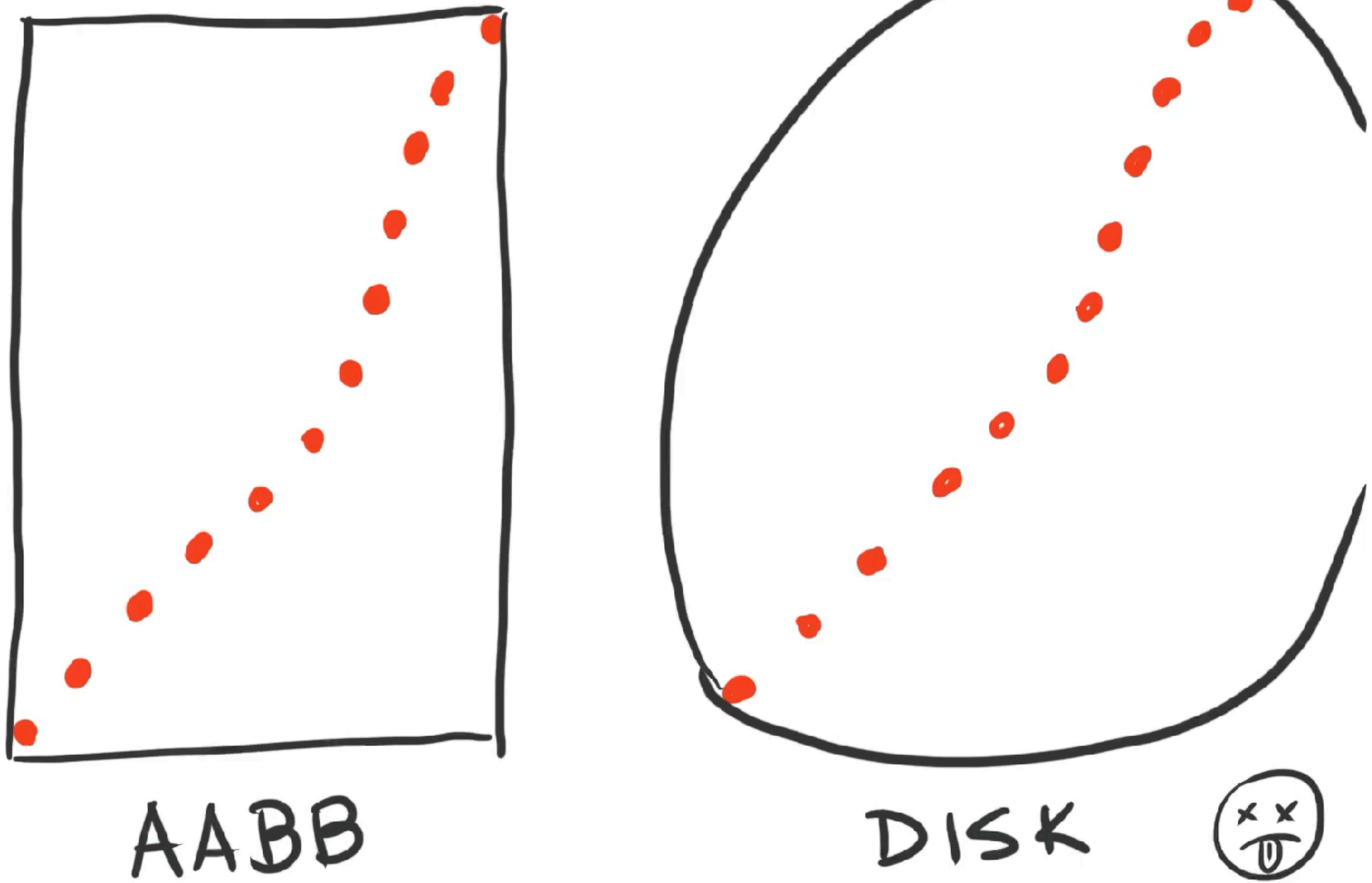
- Approach #2 (Welzl's MiniBall): O(N) randomized algorithm for incremental construction.
  - At each step know support points that define smallest bound.
  - If a new point is not inside, make it a support point (remove others, as needed)

Fitting BVs: Bounding disks and spheres

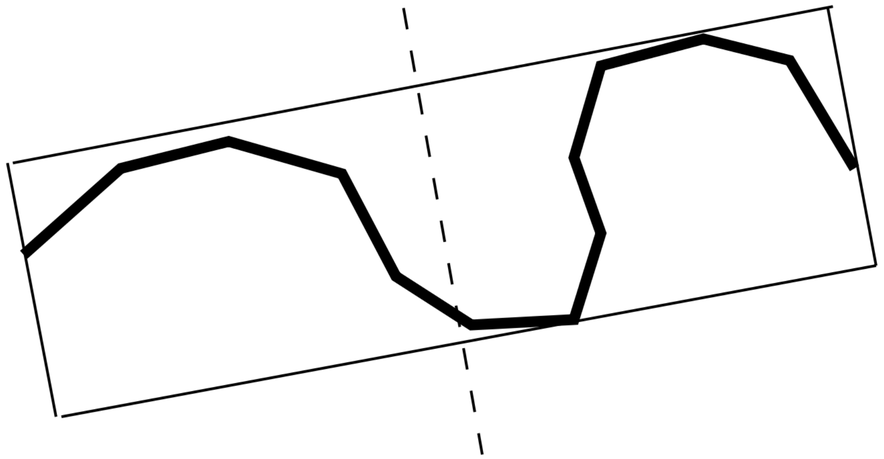
- Approach #2 (Welzl's MiniBall): O(N) randomized algorithm for incremental construction.
  - At each step know support points that define smallest bound.
  - If a new point is not inside, make it a support point (remove others, as needed).



Problem: Loose fit



Fitting BVs: Oriented Bounding Box (OBB)



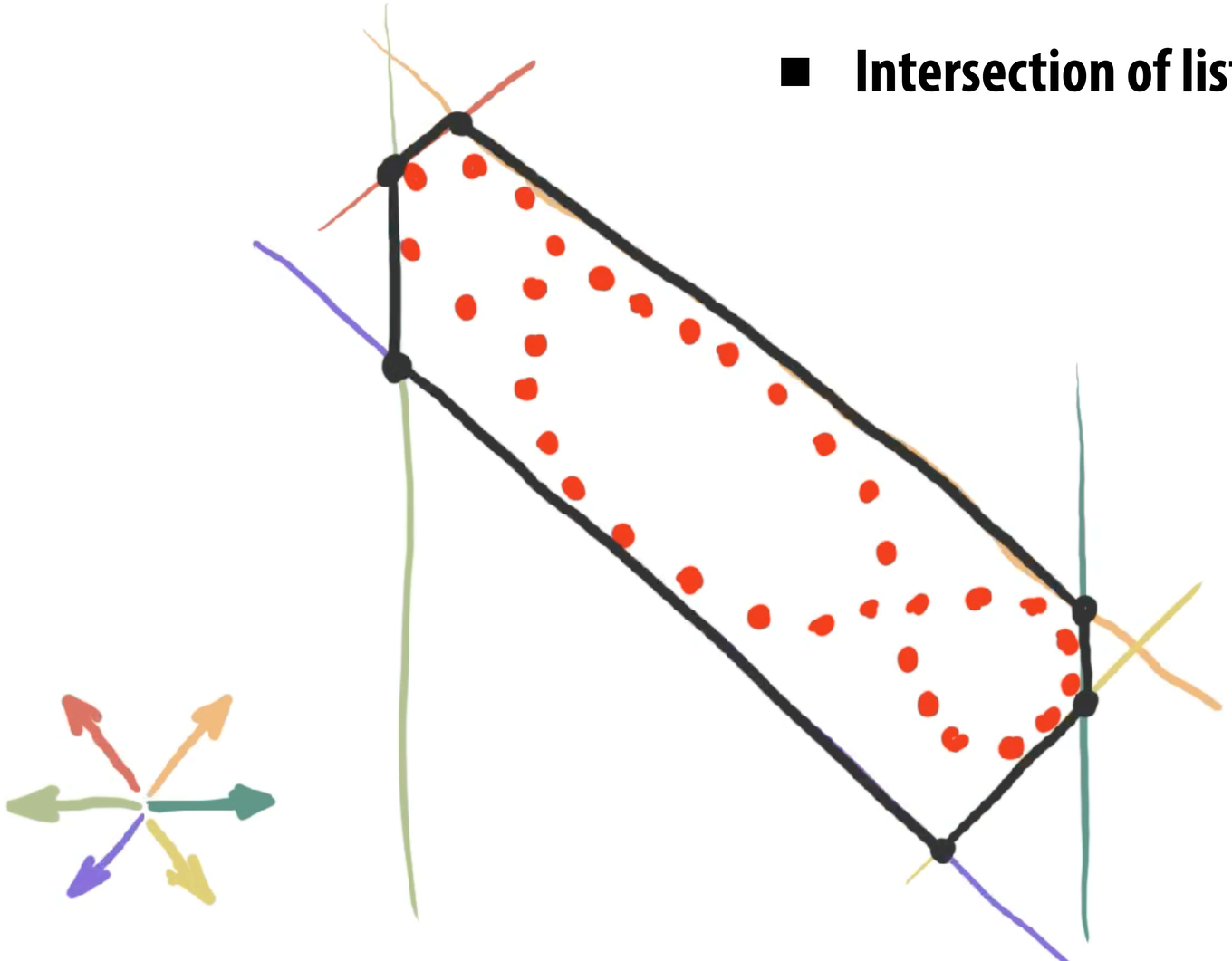
- Fitting:
  - Find box rotation, R, that reduces bounding box volume
  - Scan through all N points to find max/min along each axis of R
- Finding R:
  - Compute principal component directions of point cloud using PCA
  - Problem: Interior points can distort orientation
  - Solution: Uses area-weighted points on object's convex hull
  - Reference: "OBB-Tree" paper [Gottschalk et al. 1996]

Fitting BVs: k-DOP bound

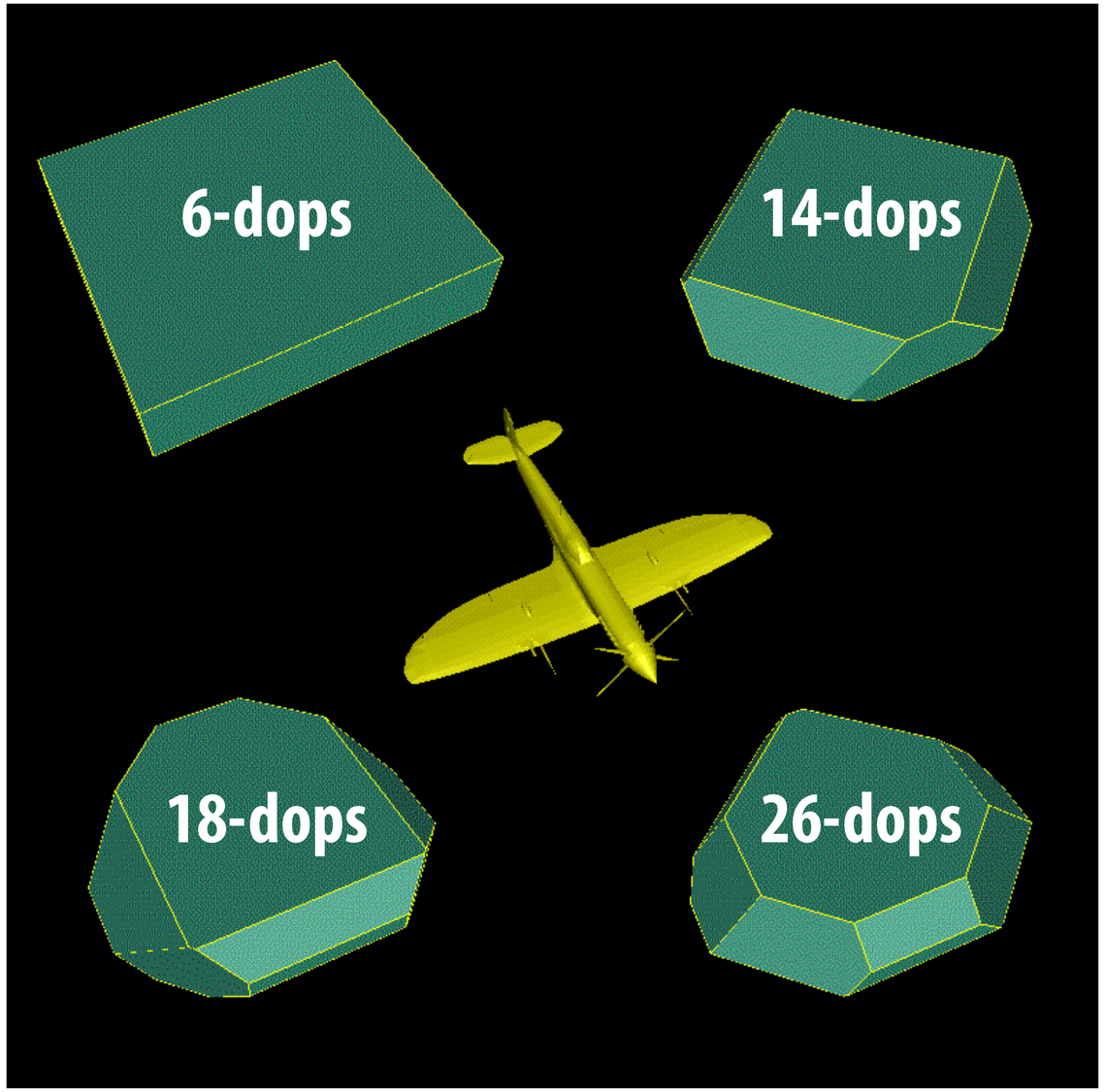
- Avoids finding "best" orientation for OBB by discretizing to k directions
- Generalizes AABB bounds for ±axis directions to k directions, e.g., a 2D AABB is a 4-DOP

Fitting BVs: k-DOP bound

- Avoids finding "best" orientation for OBB
- Generalizes AABB bounds for ±axis directions to k directions, e.g., a 2D AABB is a 4-DOP.
  - Intersection of list of half spaces, $\mathbf{n}_i \cdot \mathbf{x} \leq d_i$



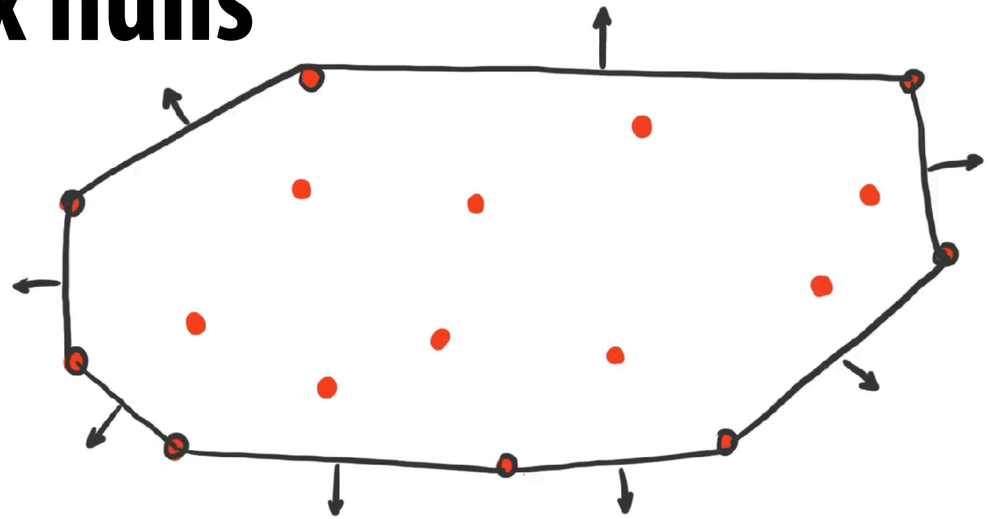
Fitting BVs: k-DOP bound

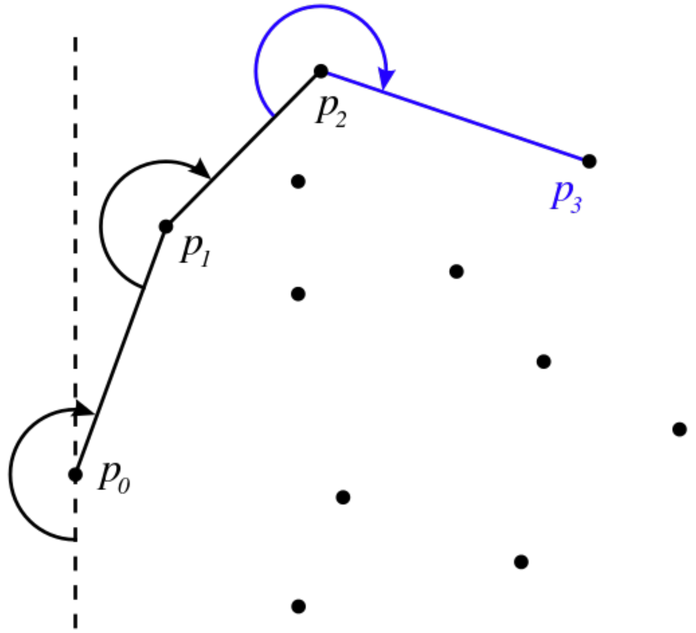


https://www.cosy.sbg.ac.at/~held/projects/collision/level_0.gif

Fitting bounding volumes: convex hulls

- Convex hulls give the tightest convex bound, e.g., "shrink wrap" or "elastic band"
  - Also the most complex bounding volume
- Intersection of list of half spaces, $\mathbf{n}_i \cdot \mathbf{x} \leq d_i$
- Many convex hull algorithms
  - Related to sorting algorithms
  - https://en.wikipedia.org/wiki/Convex_hull_algorithms

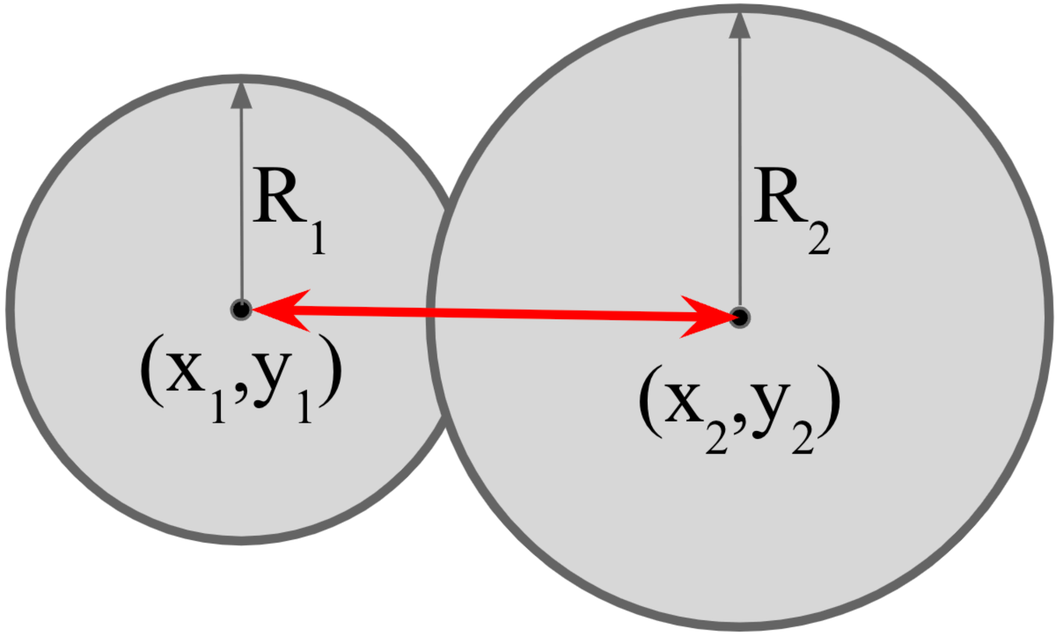




"Gift wrapping" algorithm

Testing Bounding Volumes

Testing Bounding Disks and Spheres



Overlaps if $\text{dist}_{12} < R_1 + R_2$

Avoid the square-root using:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 < (R_1 + R_2)^2$$

Back to Basics

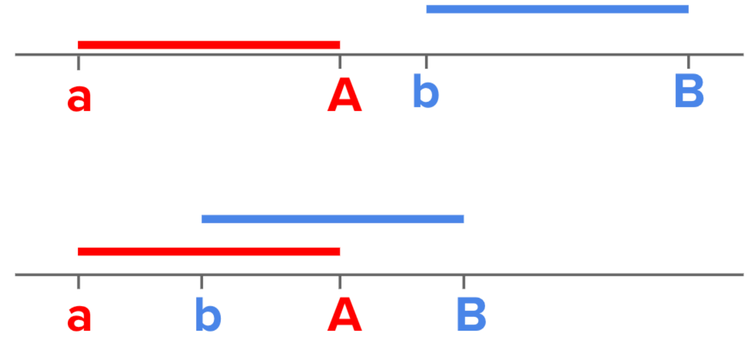
Collision Detection in 1D

Objects are intervals, $[x_{min}, x_{max}]$ for $x_{min} <= x_{max}$.

Collision detection amounts to detecting overlapping intervals.

1D Overlap Test: Two intervals **[a,A]** & **[b,B]** overlap iff

max(**a**,**b**) <= min(**A**,**B**).

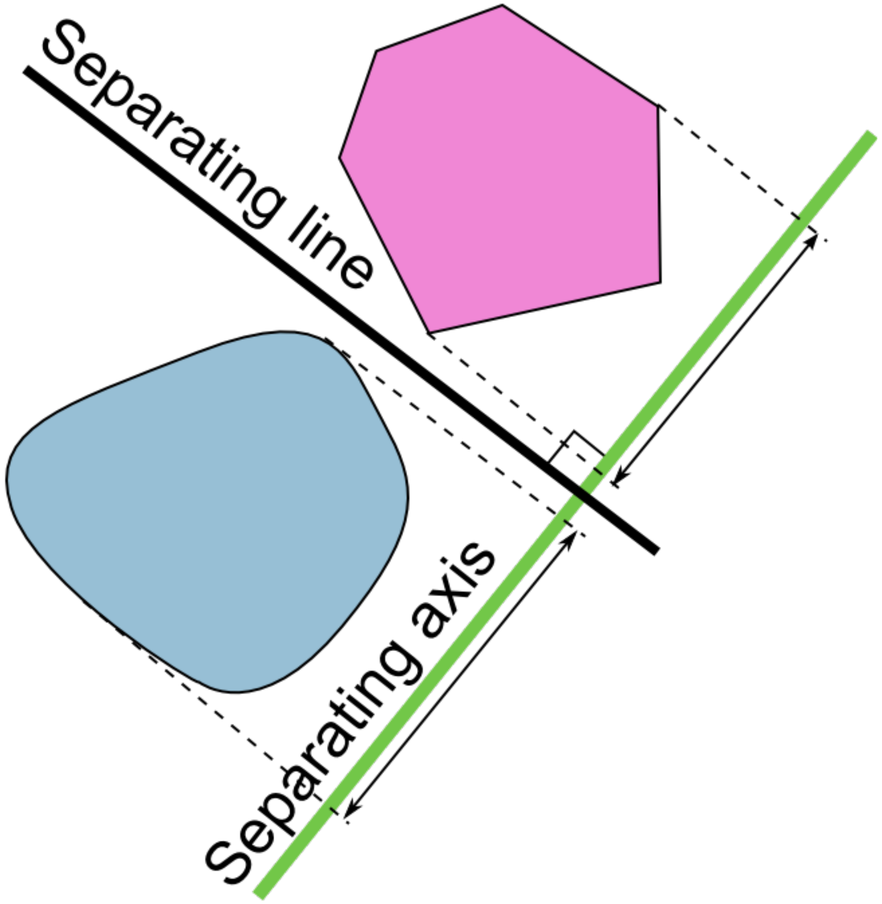


OpenProcessing Tip: Use `Math.min` & `Math.max` whenever possible.

Generalization to higher dimensions?

Separating Axis Theorem (SAT)

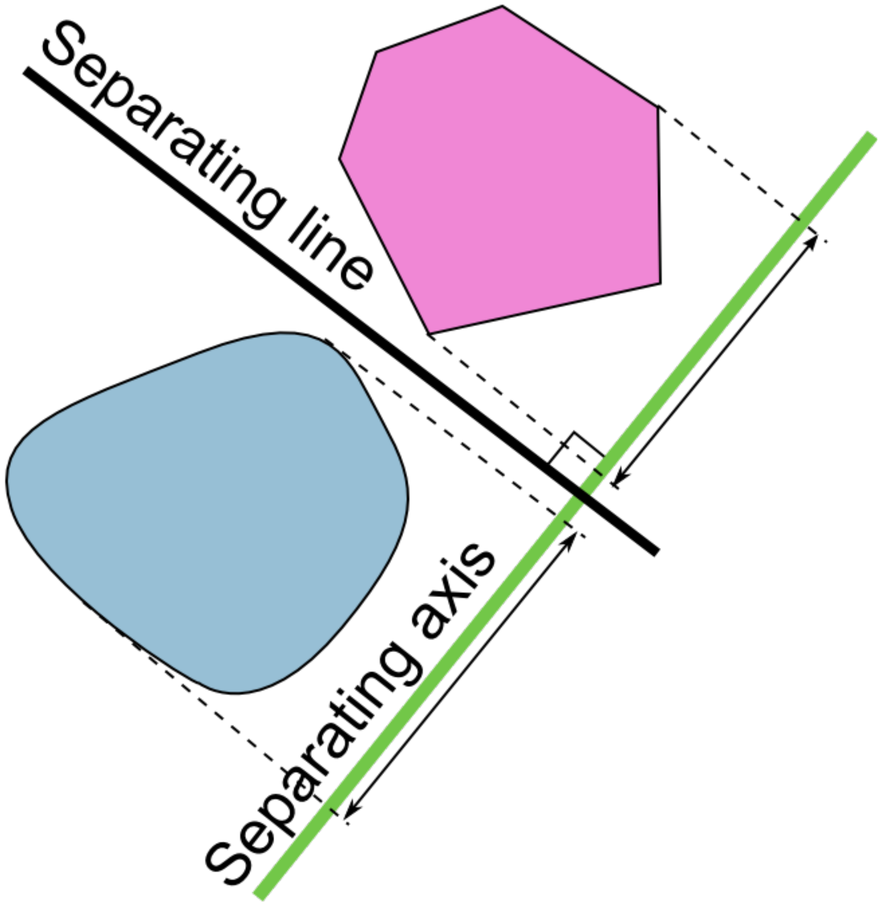
"Two convex objects do not overlap if there exists a line (called axis) onto which the two objects' projections do not overlap."



https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

Separating Axis Tests

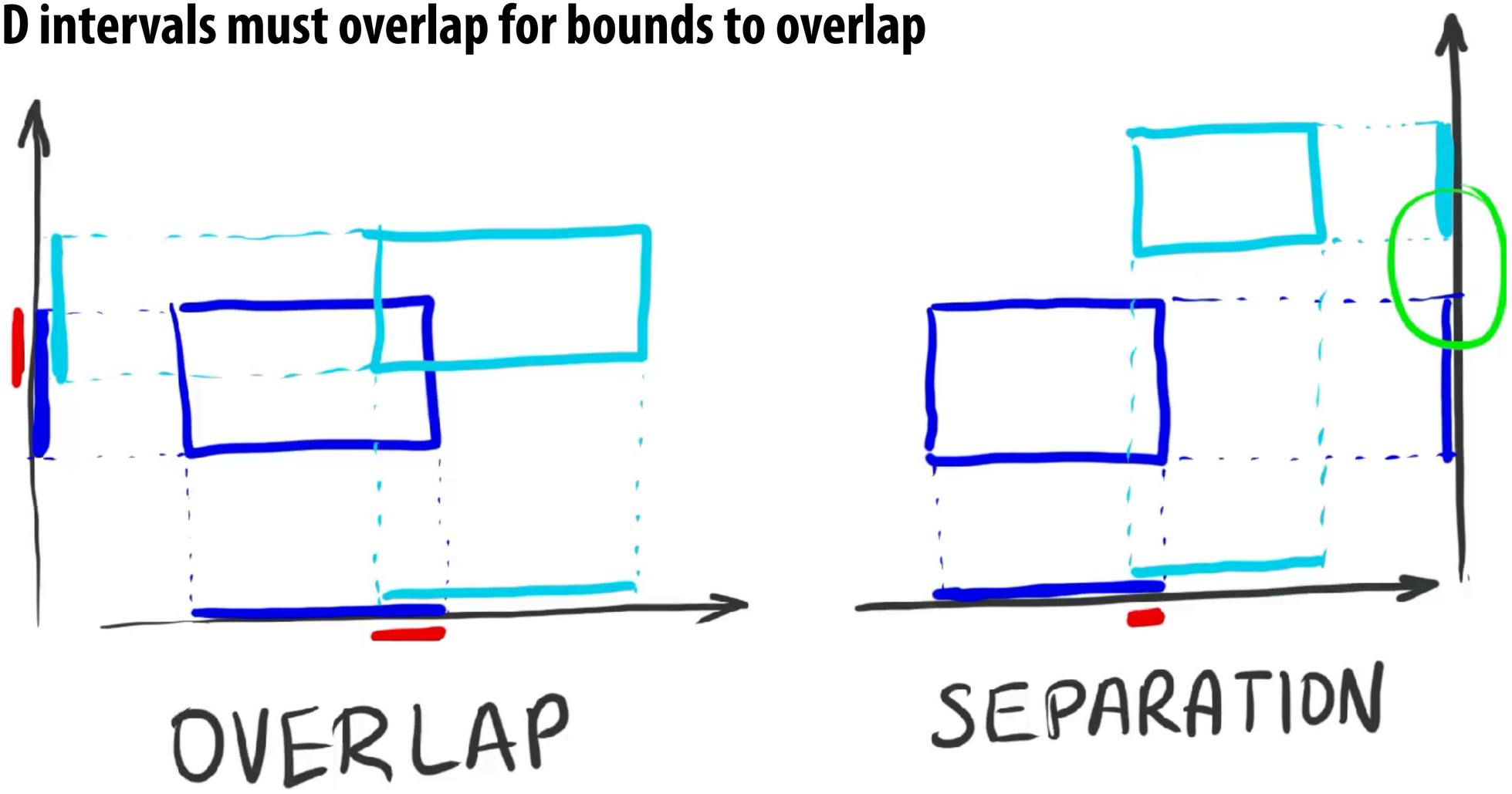
- Useful for overlap tests between convex shapes:
  - Lines, triangles, boxes (AABB & OBB), etc.
- Approach for polygons:
  - Compute list of axes to test, $\{\mathbf{n}_i\}$
  - For each direction, $\mathbf{n}_i$:
    - Project vertices onto axis, $\mathbf{n}_i$
    - Find min/max values, [a,A] & [b,B]
    - Perform 1D overlap tests
      - Return false ASAP if a non-overlapping interval is found
      - Return true if ALL intervals overlap
- Optimizations: Use bounds and/or center-of-mass axis first; test incrementally and "early exit"



https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

EX: SAT: Testing AABBs and k-DOPs

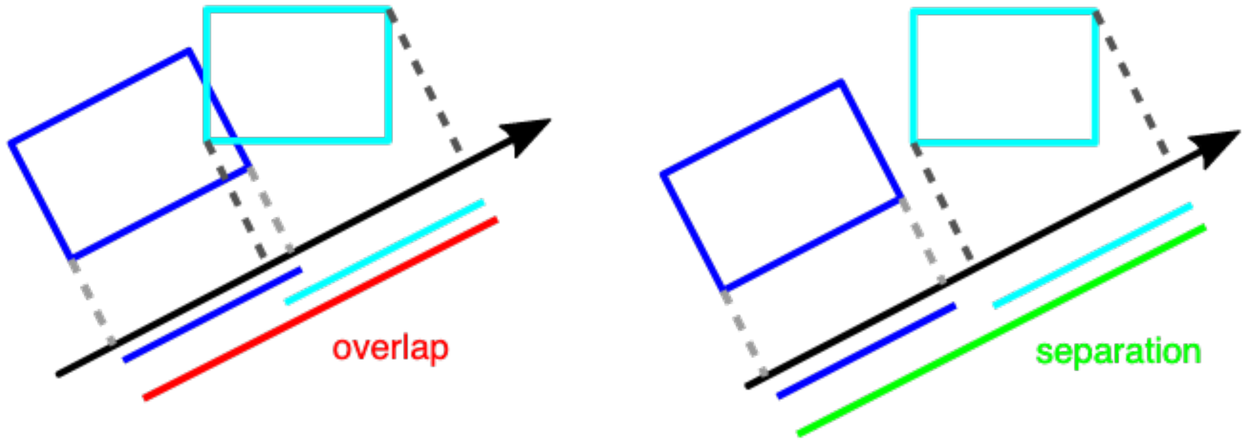
- Direct application of 1D overlap tests to each axis
  - All 1D intervals must overlap for bounds to overlap



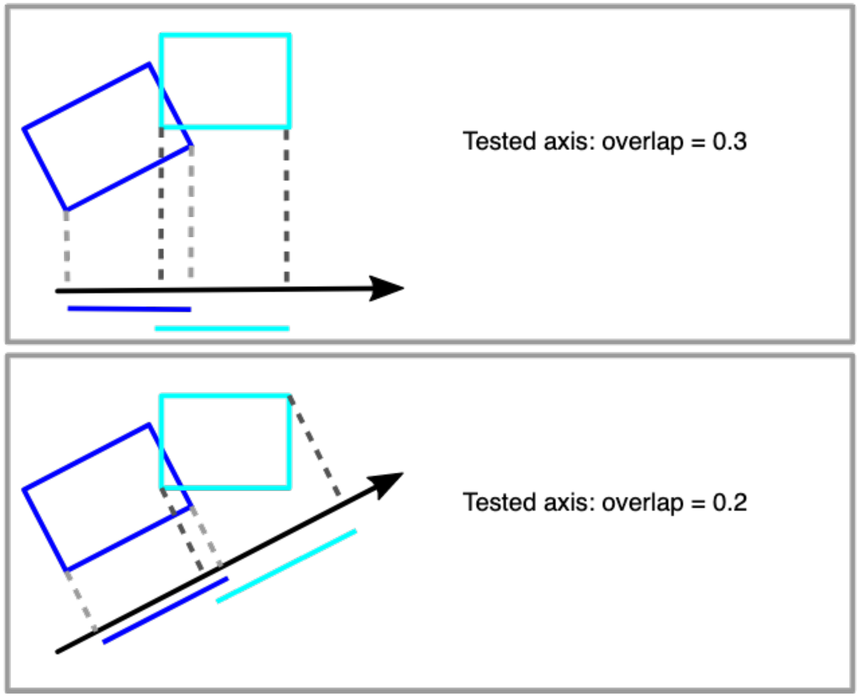
EX: SAT: Testing OBBs

- More complicated
  - 4 axes in 2D
  - 15 axes in 3D [Gottschalk et al. 1996]

Projection of box shapes onto a potential separating axis (the arrow)

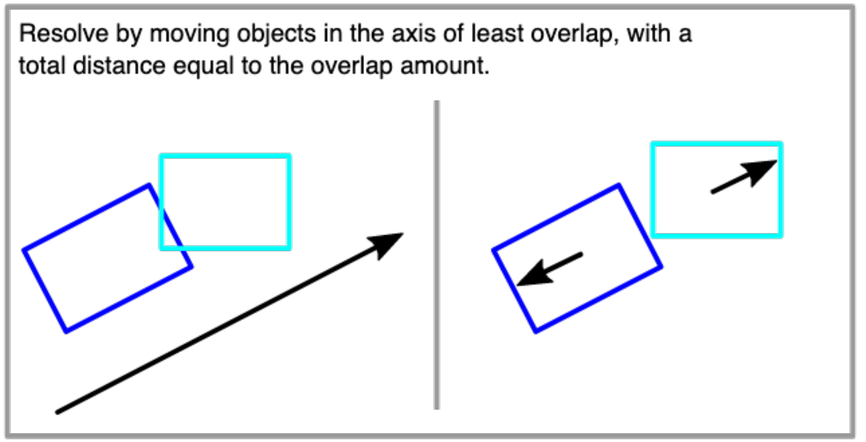


Tracking axis and overlap distance for each test



Repeat for all potential separating axes...

Resolve by moving objects in the axis of least overlap, with a total distance equal to the overlap amount.



https://www.atoft.dev/posts/2020/04/12/implementing-3d-collision-resolution/