Stanford CS 448Z

Physically Based Animation and Sound

Collision Detection

Prof. Doug James

The need for collision detection

Most sounds are produced by contact interactions

NVIDIA FleX
https://developer.nvidia.com/flex

Distributed Ray Tracing [Cook et al. 1984]
https://graphics.pixar.com/library/DistributedRayTracing/index.html

Collision Detection & Proximity Queries

Terminology

Collisions

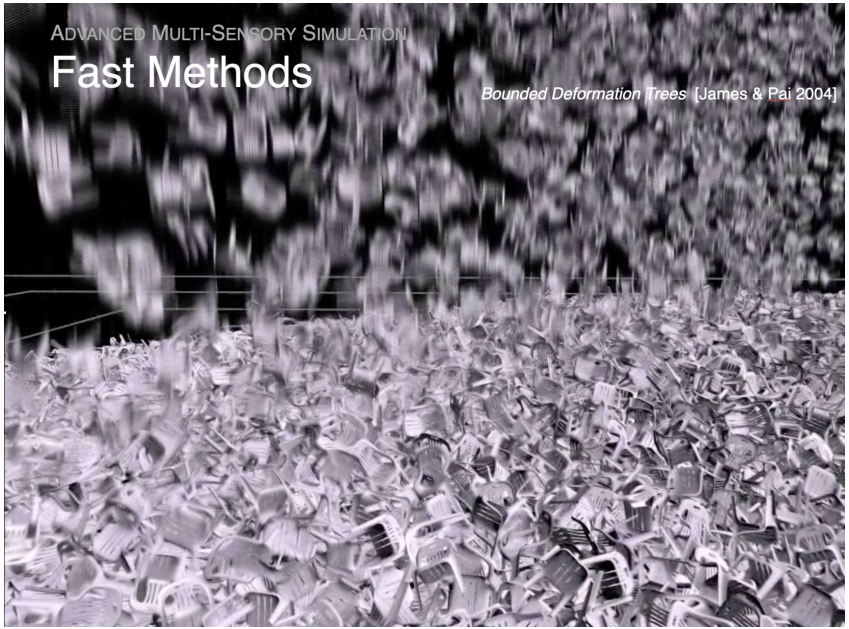
Two parts:

1. Collision detection:
   - Geometric determination of collisions (what, where, when)
2. Collision resolution:
   - Physical application of forces or impulses to modify motion

Collision Detection & Proximity Queries

Fast computer methods for

- **Proximity** and **distance** queries
- Fast **neighbor** finding
- **Overlap** tests, e.g., triangle-triangle test
- Discrete and continuous **collision** tests
- **Ray intersection** tests (rendering!)



Many interesting data structures:

- Geometric bounds
- Bounding volume hierarchies
- Spatial subdivisions (uniform and adaptive grids)
- Sorting and hashing

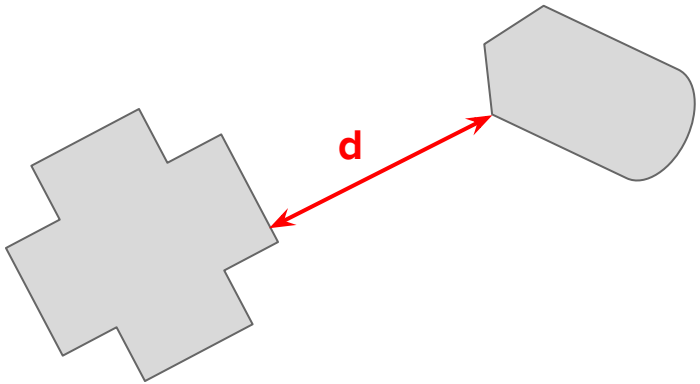
Types of queries

Proximity & Distance Queries

Finding closest objects and features (e.g., vertex, edge, face)

Evaluating minimum distances

- Between primitives
- Between objects



Object representations:

- Mathematical domains
  - Implicit functions, e.g., distance fields
- Polyhedral meshes
- Spline or subdivision surfaces
- Swept motions

Fast methods for most cases.

Point-object distances

Distance Fields

Distance Fields: Encode distance from a **point**, x, to an object, *O*:

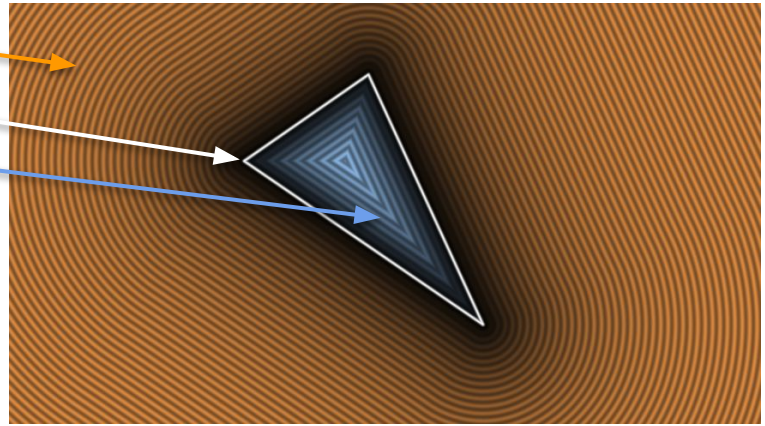
$d(x) = \min \|x-y\|_2$ over all y on *O*.

Signed Distance Fields: Sign indicates if inside/outside object:

$d(x) > 0$: Outside object

$d(x) = 0$: On object

$d(x) < 0$: Inside object



Gradient of d(x) is direction to closest surface.

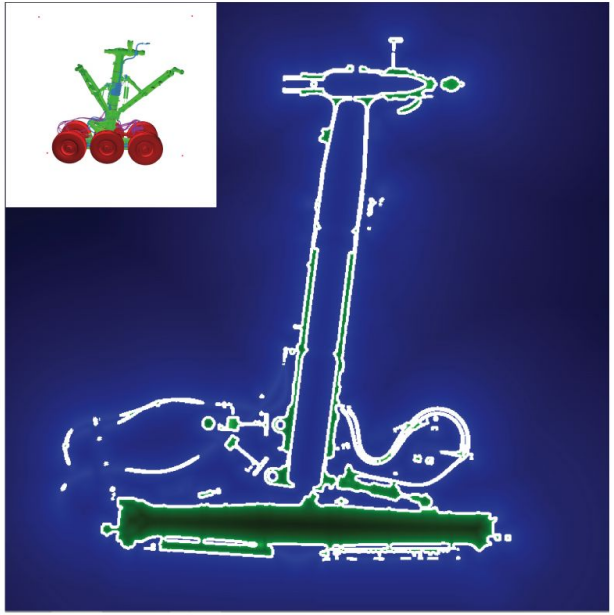
Analytical formulas exist only for simple shapes,

E.g., https://www.shadertoy.com/view/XsXSz4

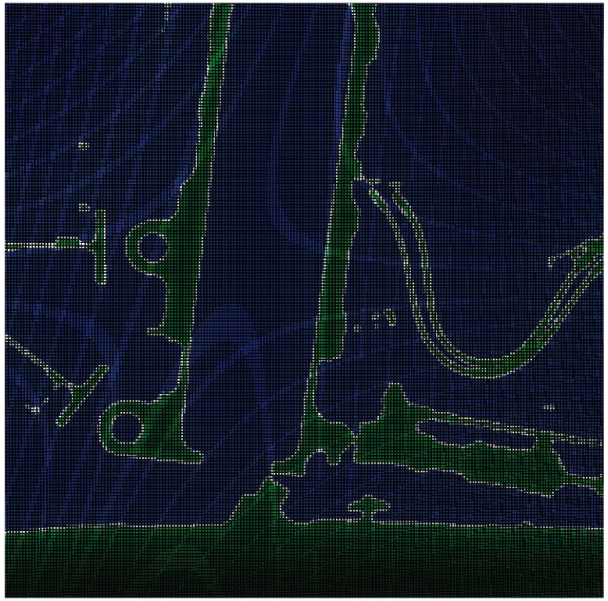
Point-object distances

Distance Fields for Complex Shapes

Interpolate distances previously sampled on a regular (or adaptive) grid



(c) Landing gear 1024x1024x1024 signed distance field



(d) Detailed view of landing gear distance field

Pros:

- Fast d(x) evaluation
- Complex shapes
- Parallel precompute

Cons:

- High memory overhead
  - Adaptive helps
- Precomputation
- Rigid geometry
  - Can't deform
- Only point-object distance

http://barbic.usc.edu/signedDistanceField/

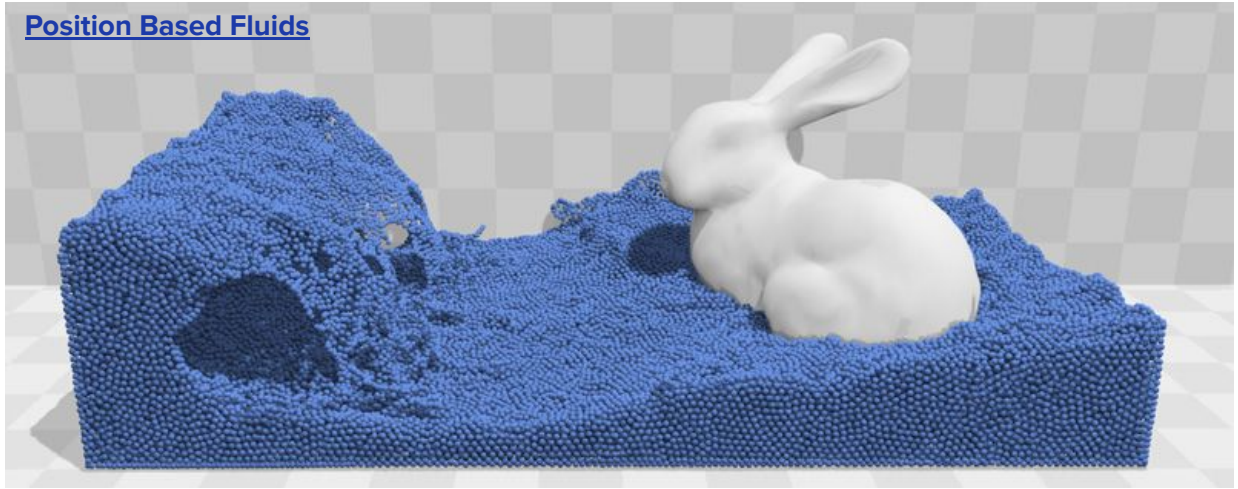
Motivation: Particle-based Fluid Simulation

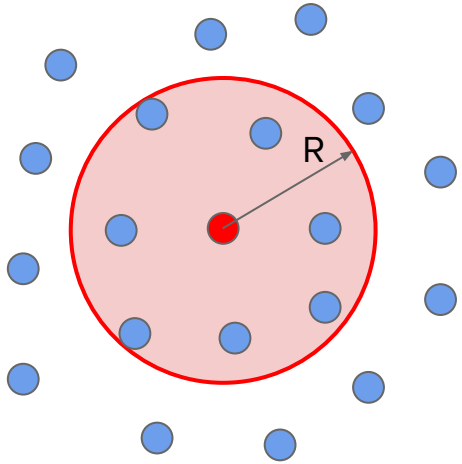
Fast neighbor finding

Often want to know neighbors within some distance cutoff, R.

Example: Particle-based fluid simulation

- Particle-particle interactions only occur within some small distance, R.
- Each particle needs to find neighboring particles within distance, R.



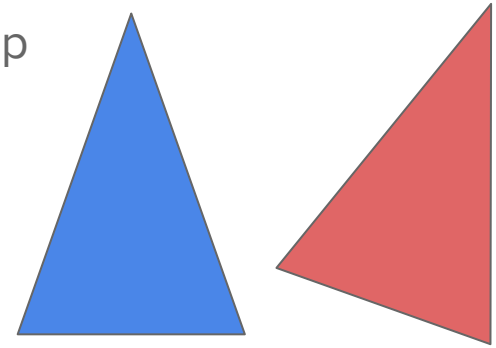


Motivation: Particle-based Fluid Simulation

Fast neighbor finding

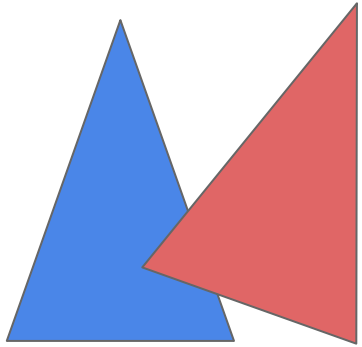
Overlap and Intersection Tests

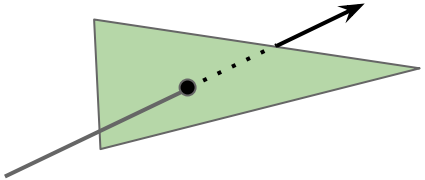
Determine if shapes or primitives (triangles, rays, etc.) overlap



Different queries possible:

- Determine if overlap occurs
  - Return true or false
  - E.g., boolean collision check
- Determine all overlaps
  - Enumerate all overlapping pairs for meshes
- Determine where primitives overlap
  - E.g., to estimate “contact points” for collision processing
  - E.g., to find “intersection point” for ray-triangle intersection



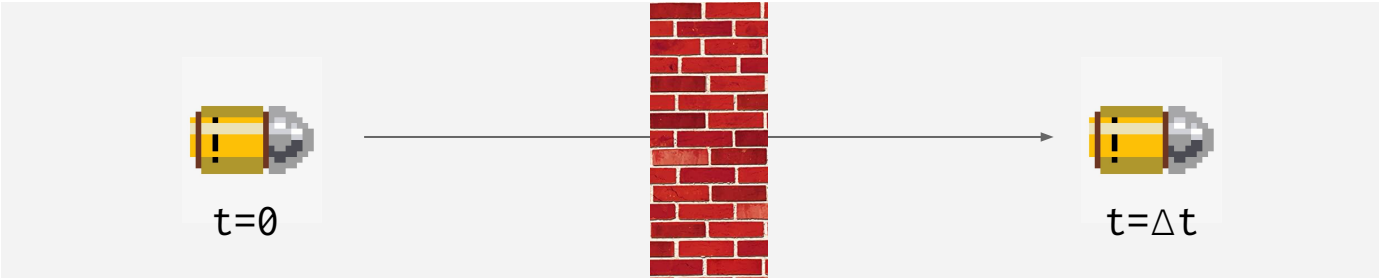


Discrete vs continuous time collision checks

Accounting for Motion

Problem:

Missed collisions

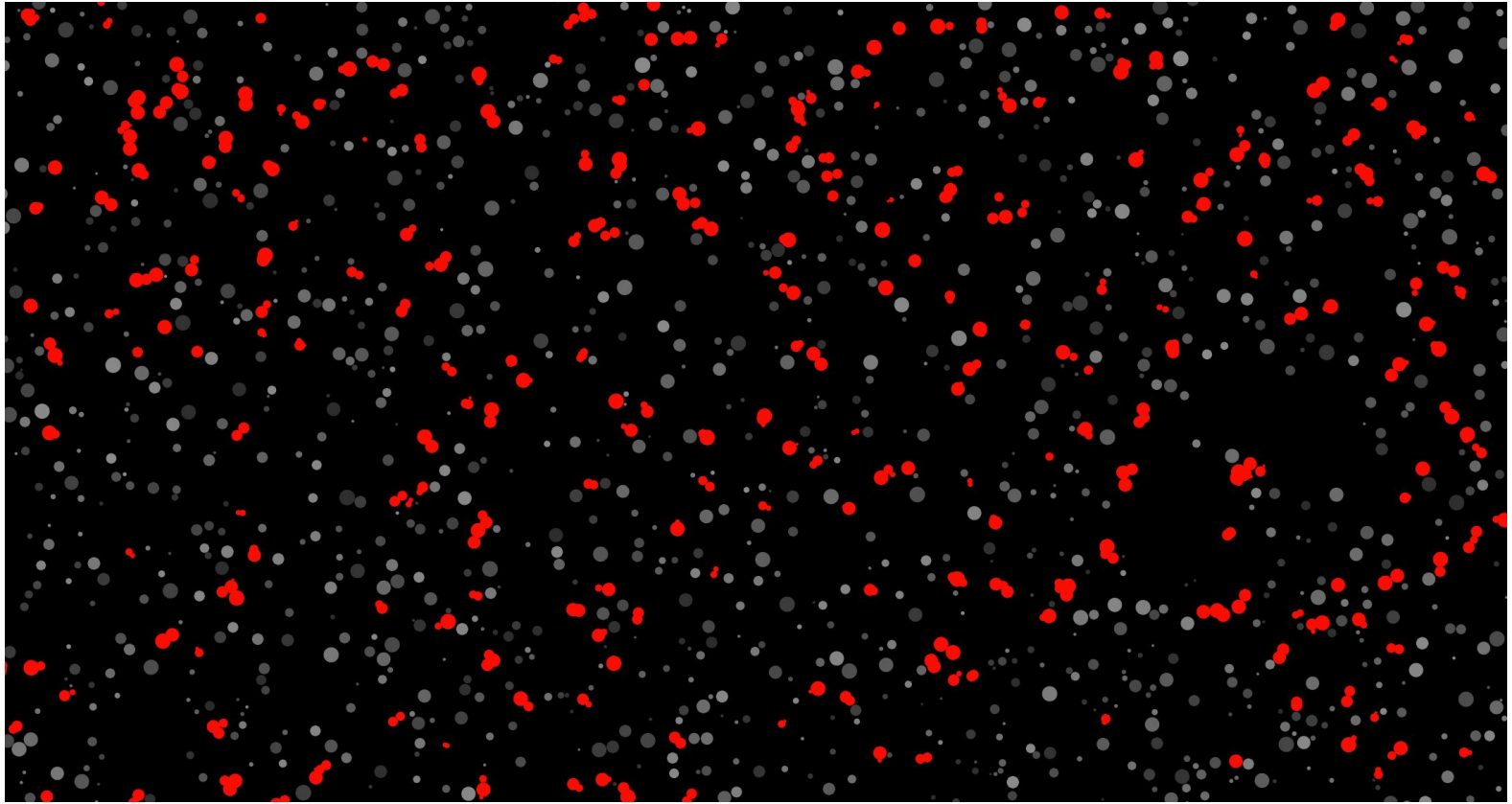


Discrete (sampled) vs continuous time collision checks

- Avoids missed collisions
  - Common in game physics libraries, e.g., Bullet
- Useful for accurate time-of-collision (ToC) estimates
  - Ensures proper ordering of collision events
- Important for simulation of cloth and thin objects
  - Less common for volumetric simulations

Fundamental Problem:

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



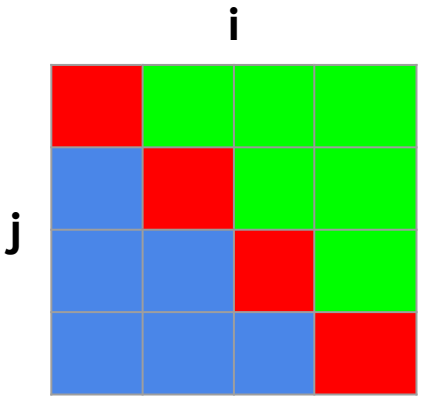
Fundamental Problem:

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

Double-loop calculations don't scale well for large numbers of things, 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
```

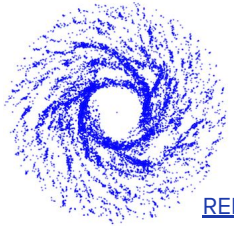
Complexity of Broad Phase Collision Detection:

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

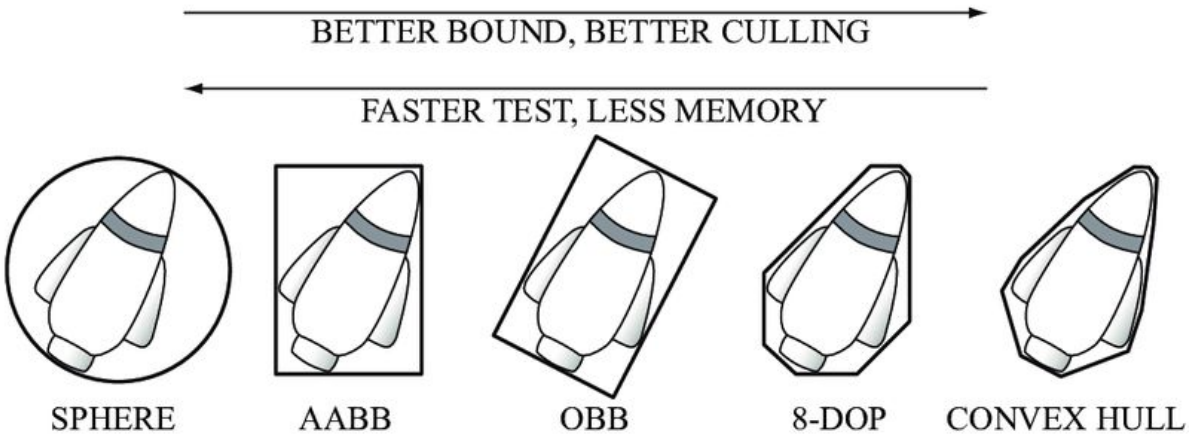
- O(1) contacts per body → O(N) contacts for N bodies

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

How? Many approaches.

Avoiding all-pairs checks

Bounding Volumes



[Ericsson 2004]

Simple convex bounds.

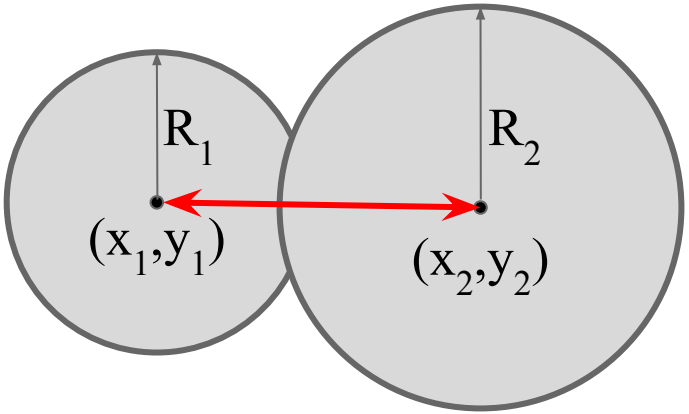
Fast & simple overlap tests:

- False positives possible
- False negatives impossible

Fast methods to fit bounds exist, e.g., O(N) for sphere, AABB, 8-DOP.

Bounding Volume Overlap Tests

Bounding Sphere



Overlaps if $dist_{12} < R_1 + R_2$

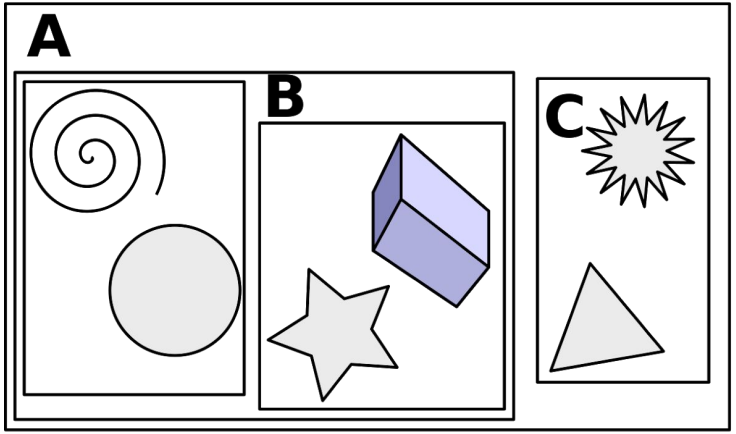
Avoid the square-root using:

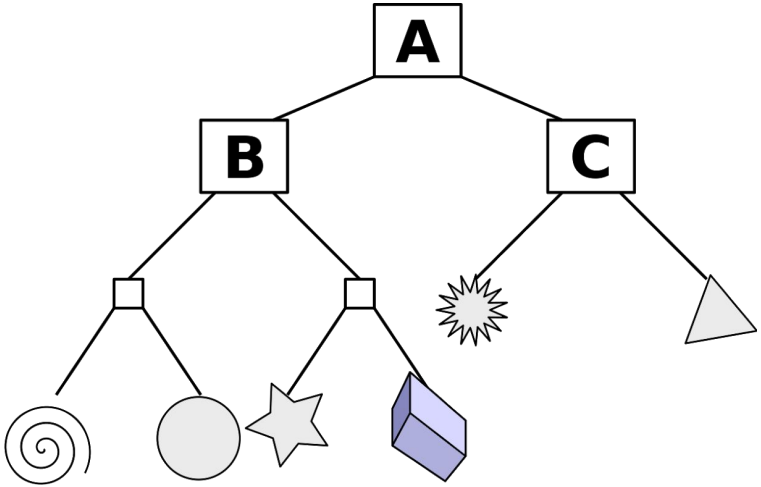
$$(x_1-x_2)^2+(y_1-y_2)^2 < (R_1+R_2)^2$$

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

Avoiding all-pairs checks

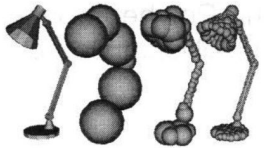
Bounding Volume Hierarchies





Common BVHs in graphics:

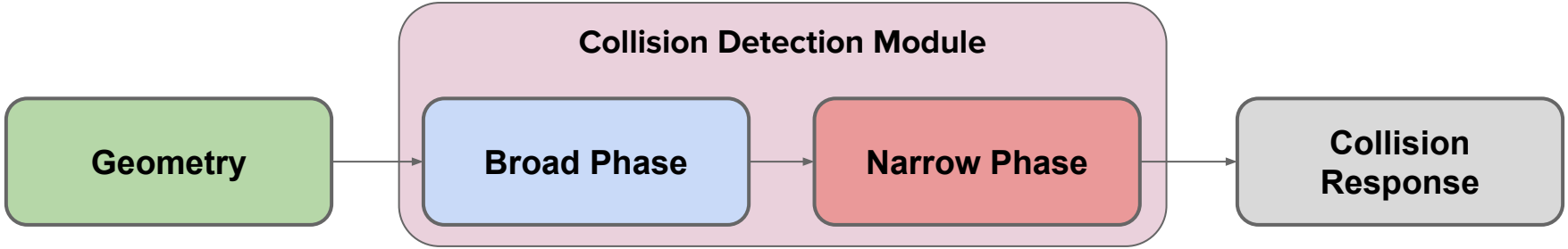
- AABB Trees [van den Bergen 1998]: good for deformable models
- OBB Trees [Gottschalk et al. 1996]: tighter fitting
- Sphere Trees [Hubbard 1996]

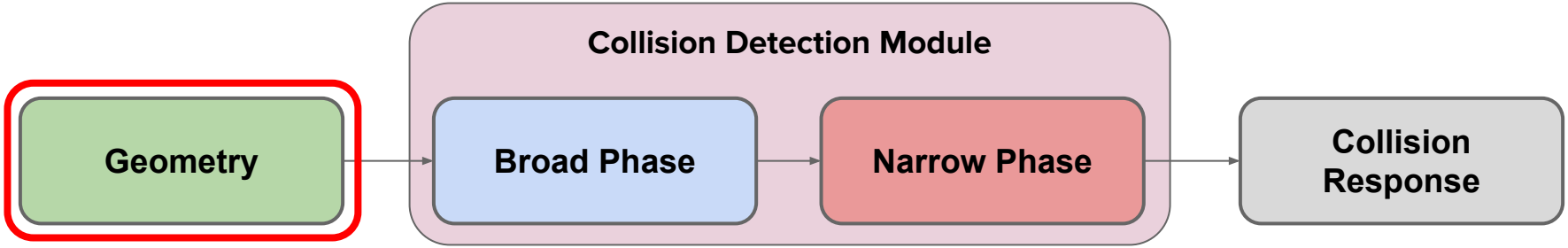


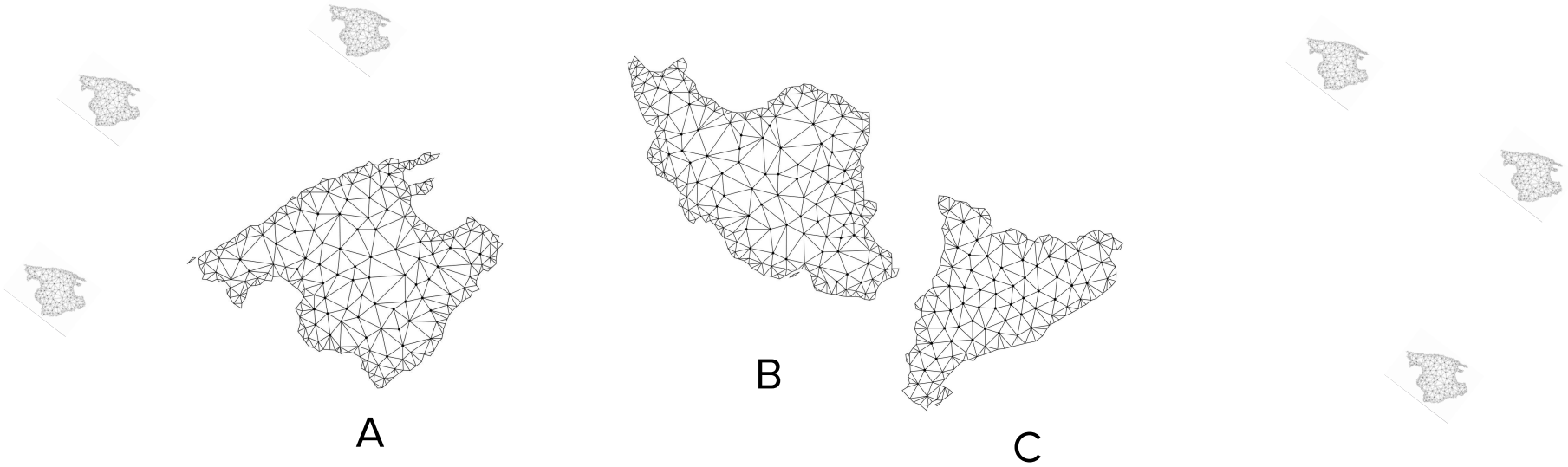
Many strategies for BVH construction

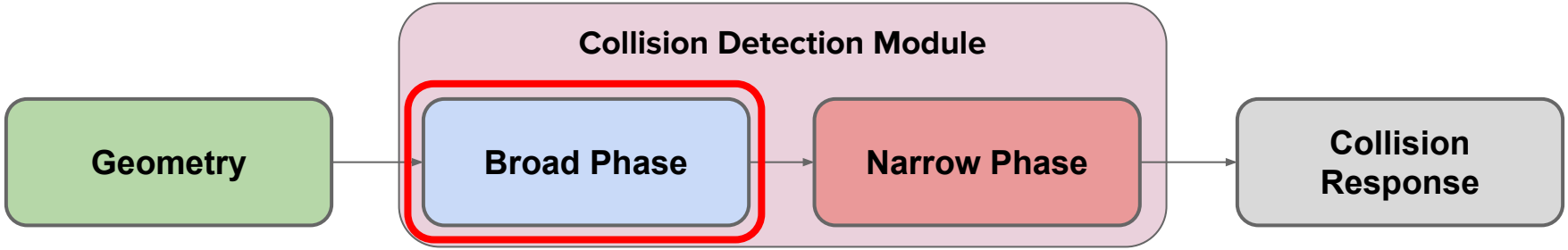
Terminology

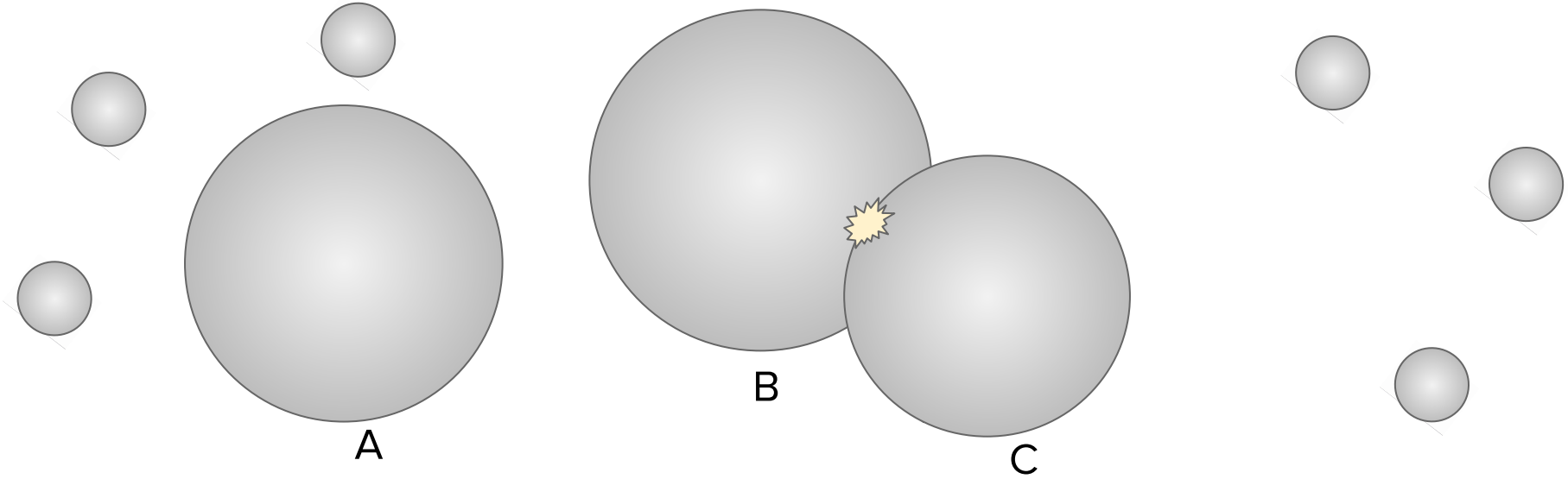
Broad- and Narrow-Phase Checks

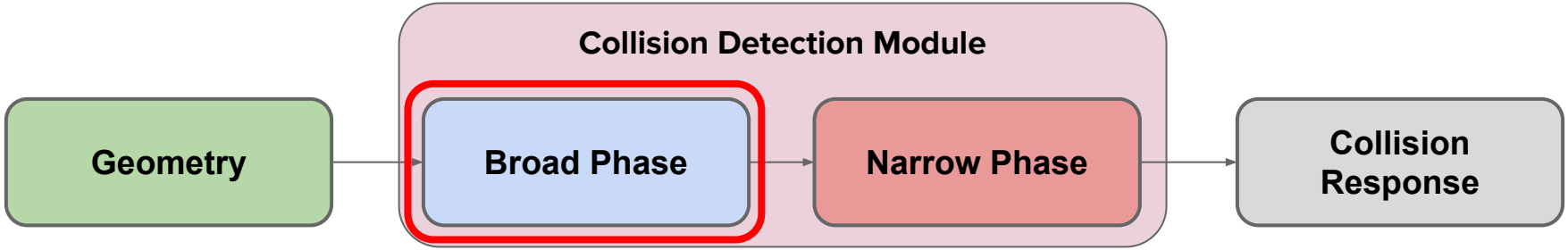


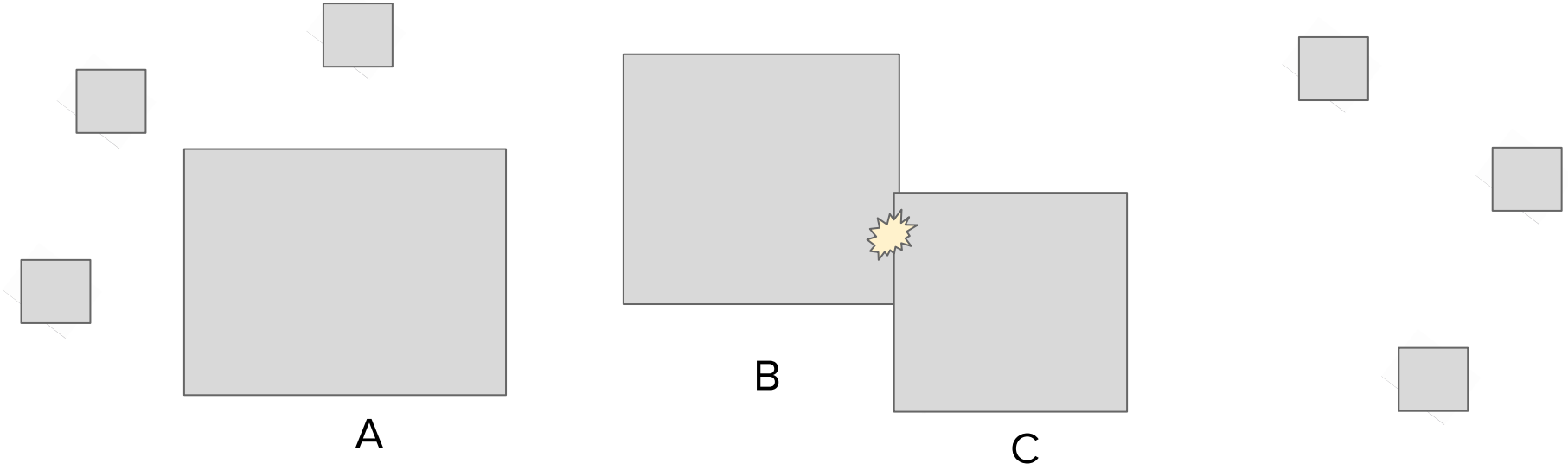


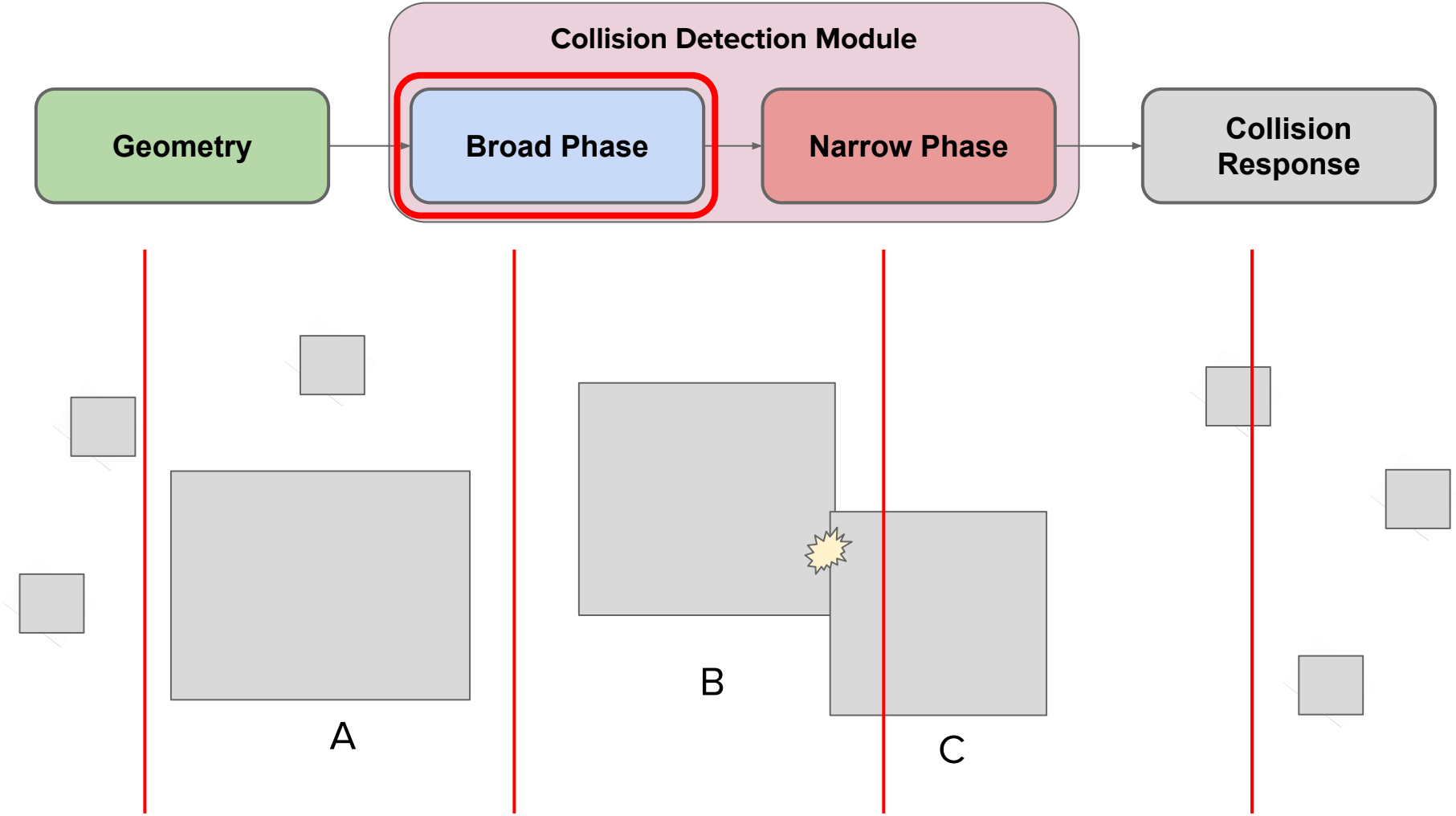


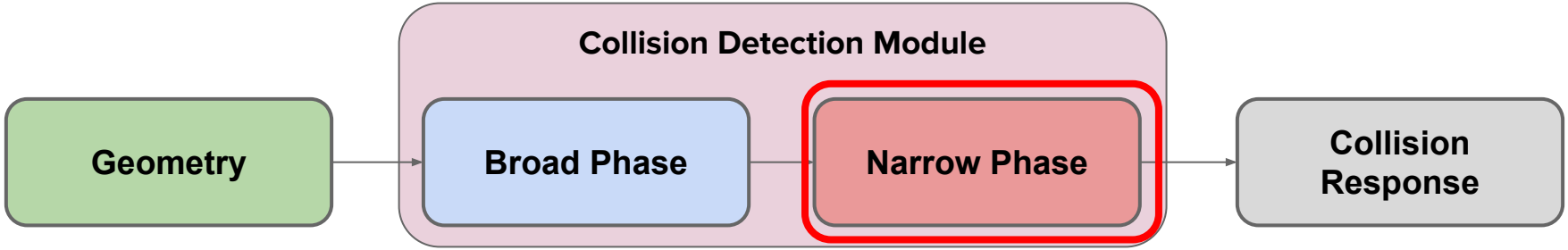


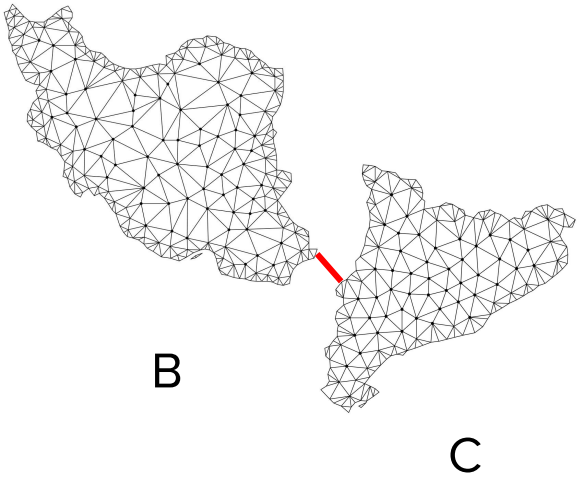












Terminology

Broad- and Narrow-Phase Checks

Broad Phase:

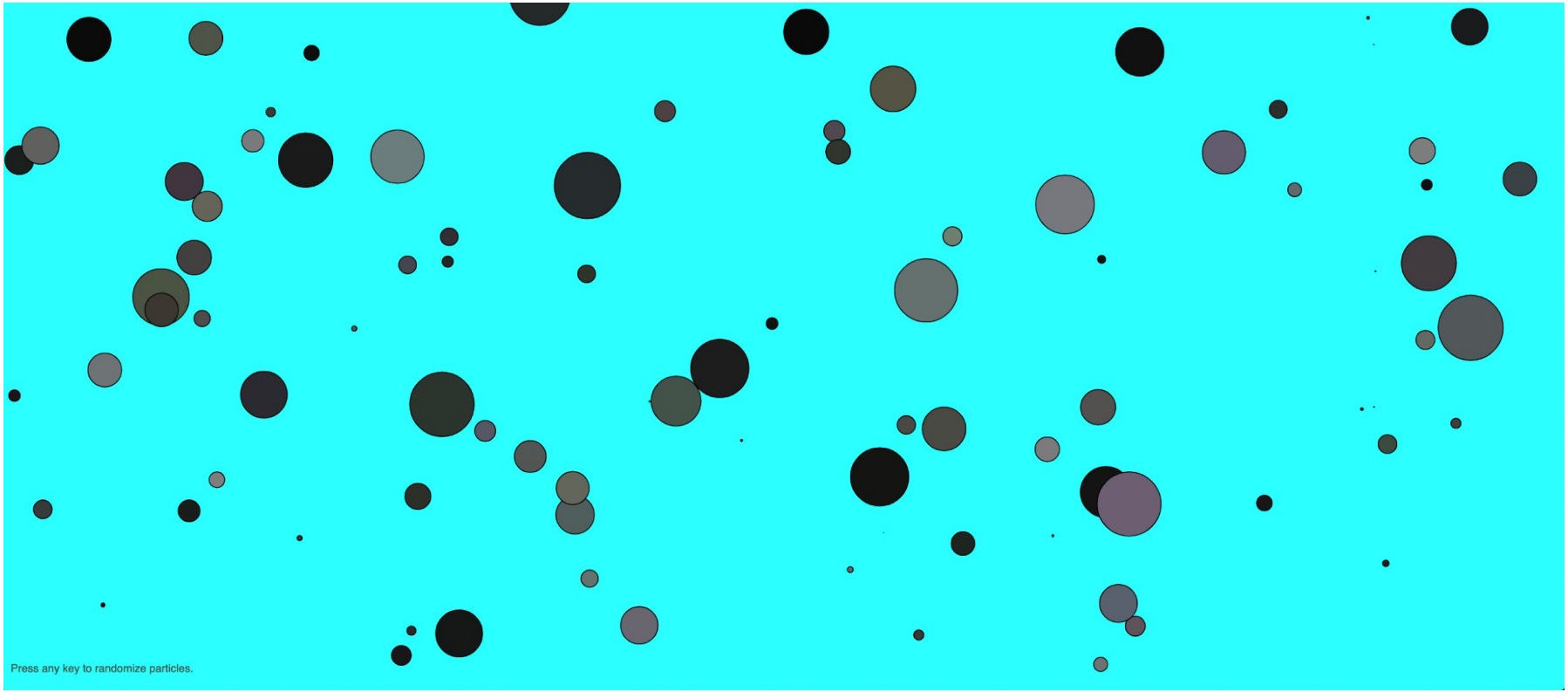
- Coarse object-level view of simulation, e.g., as bounding volumes
- Determines object pairs that ***may*** collide, and culls other pairs
- Usually fast
- Acceleration schemes:
  - Trees:
    - Bounding Volume Hierarchy (BVH)
  - Sorting
    - Sweep & Prune
  - Spatial subdivisions
    - Uniform grids; hashing
    - Adaptive grids: quadtree (2D) and octree (3D), kd-trees

Narrow Phase:

- Processes object-object pairs
- Detailed collision checks
  - Triangle-triangle
  - Ray-triangle
- May require its own broad- & narrow- phase layer
  - E.g., BVH on triangle meshes to determine triangle-triangle tests.
- Usually slower
  - Unless objects separated
- Worst case is still $O(N^2)$ work!
  - Why?

Speeding up the broad phase:

Culling pairwise checks for distant pairs



Back to basics

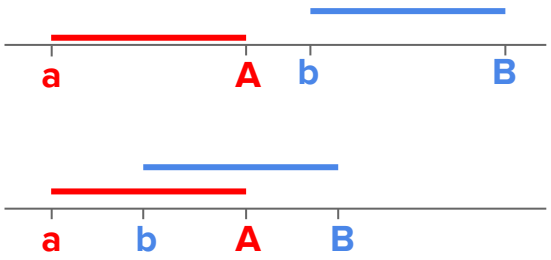
Collision Detection in 1D

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**).



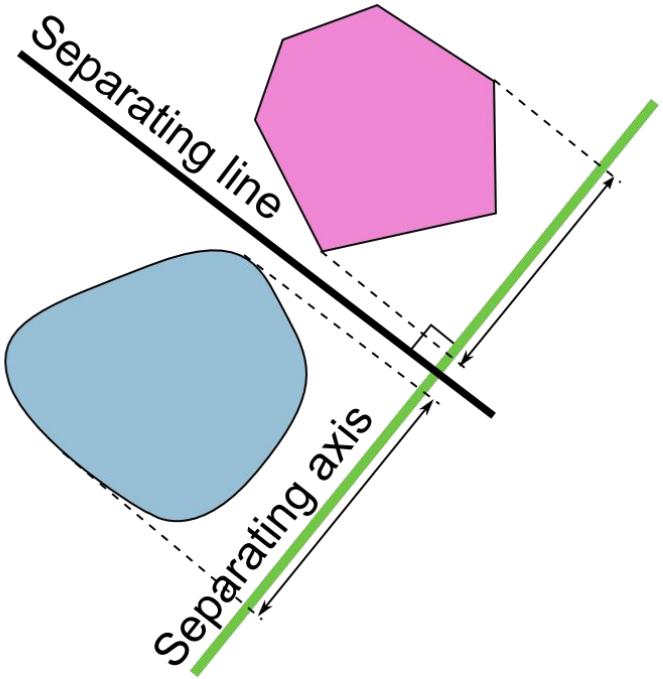
OpenProcessing Tip: Use `Math.min` & `Math.max` whenever possible.

Generalization to higher dimensions?

Convex objects

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

https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169

Convex objects

Separating Axis Theorem (SAT)

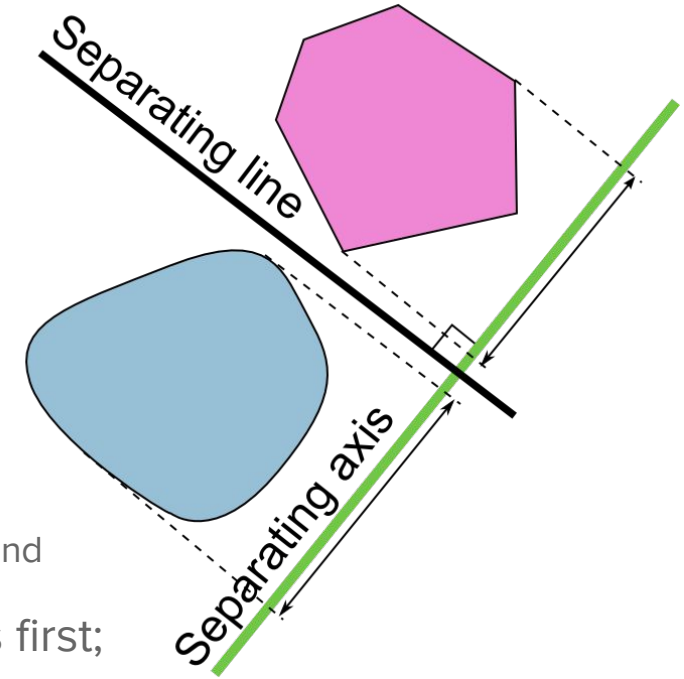
Useful for overlap tests between convex shapes:

- Lines, triangles, boxes (AABB & OBB), balls, convex hulls, etc.

Approach for polygons:

- Determine list of axes to test, $\{n_i\}$
  - E.g., all face normals
- For each direction:
  - Project vertices onto axis
  - Find max/min values, [a,A] & [b,B]
  - Perform overlap tests
    - Return false ASAP if a non-overlapping interval is found

Optimizations: Use bounds and/or center-of-mass axis first; test incrementally and early exit



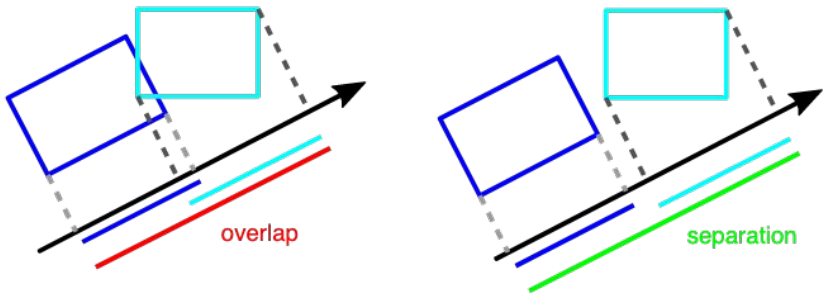
https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169

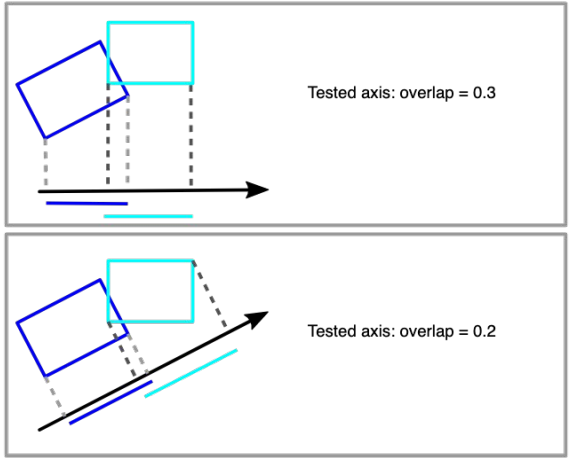
Testing Oriented Bounding Boxes (OBBs)

Separating Axis Theorem (SAT)

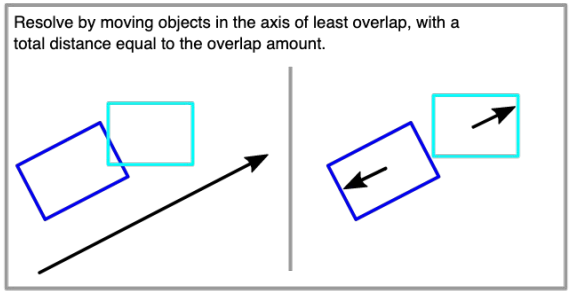
Projection of box shapes onto a potential separating axis (the arrow)



Tracking axis and overlap distance for each test



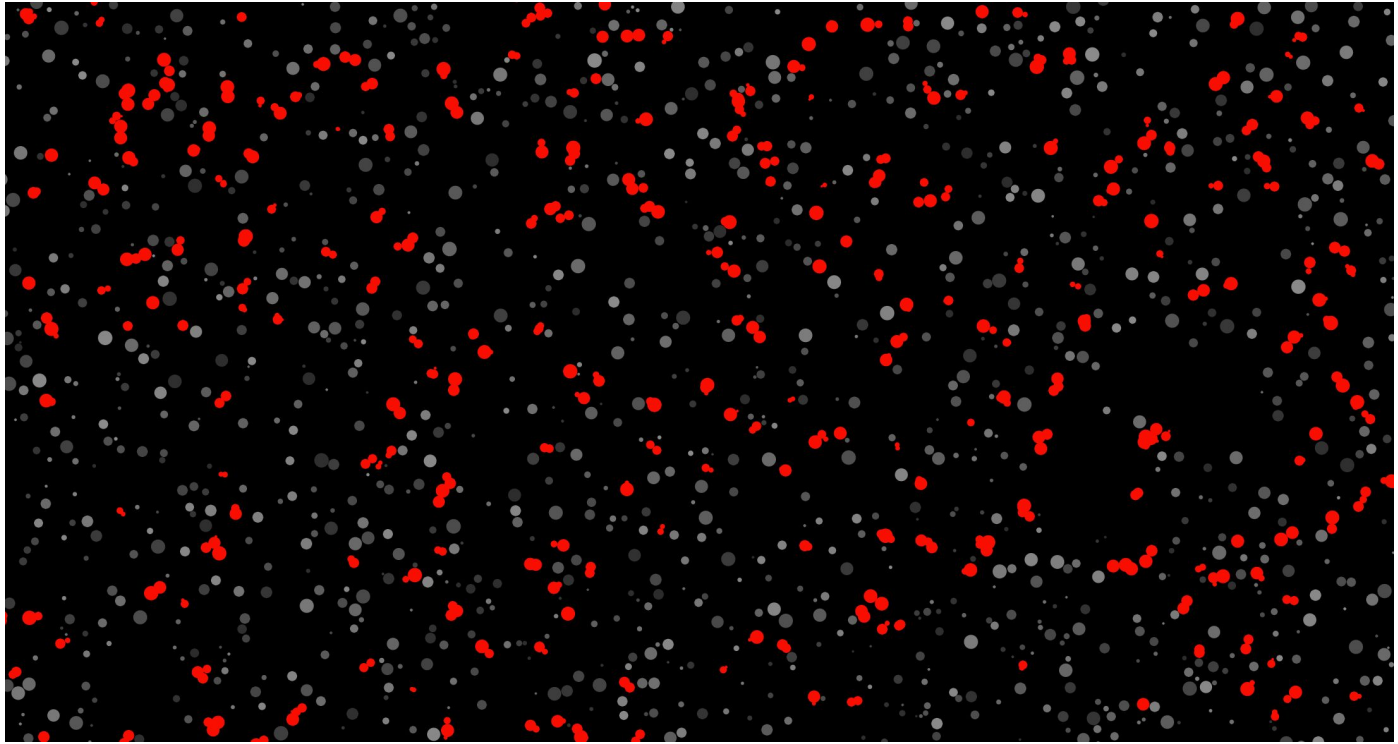
Repeat for all potential separating axes...



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

Demo

N-Particles Overlap Tests With 1D & 2D Uniform Subdivision



https://www.openprocessing.org/sketch/973557

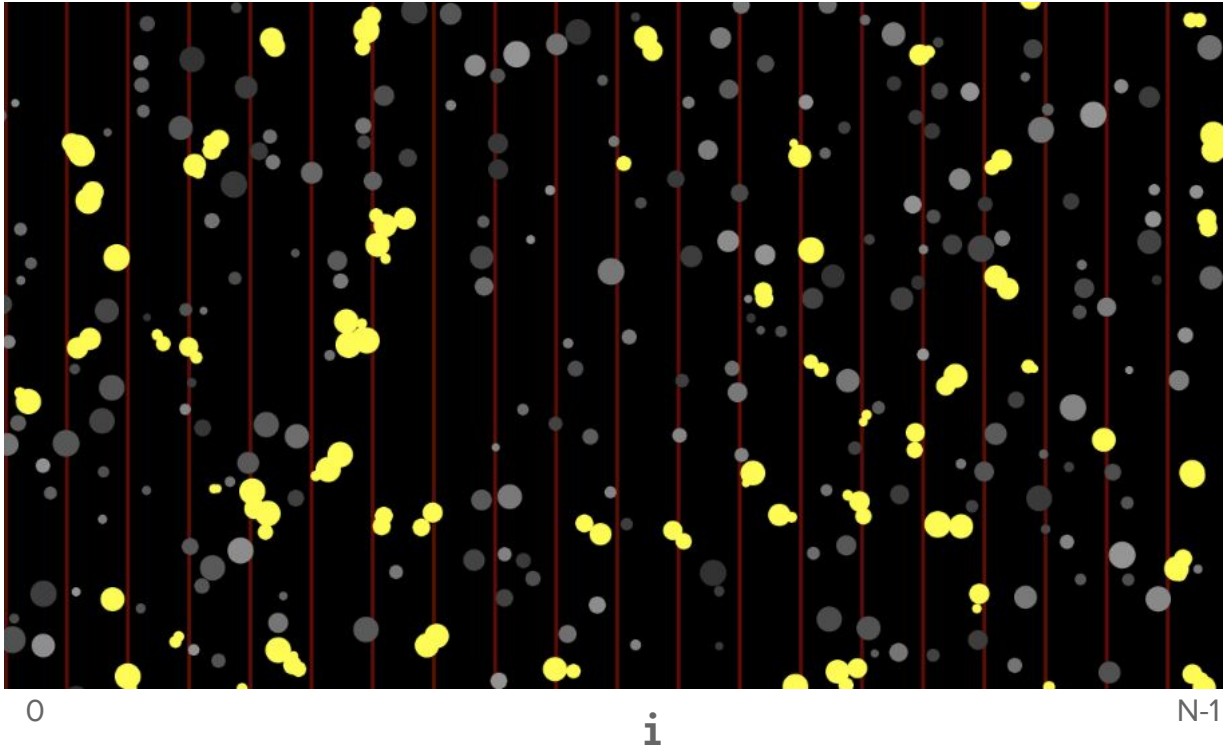
Broad-Phase Tests

1D Uniform Spatial Subdivision

Construction:

- Divide space into N bins of equal width, h
- Add each object to each bin that its bounding volume overlaps:
  - Use 1D overlap test

Cell Index, i: Given coordinate x, find containing cell `index(x)` using `Math.floor(x/h)` clamped to [0,N-1].



Broad-Phase Tests

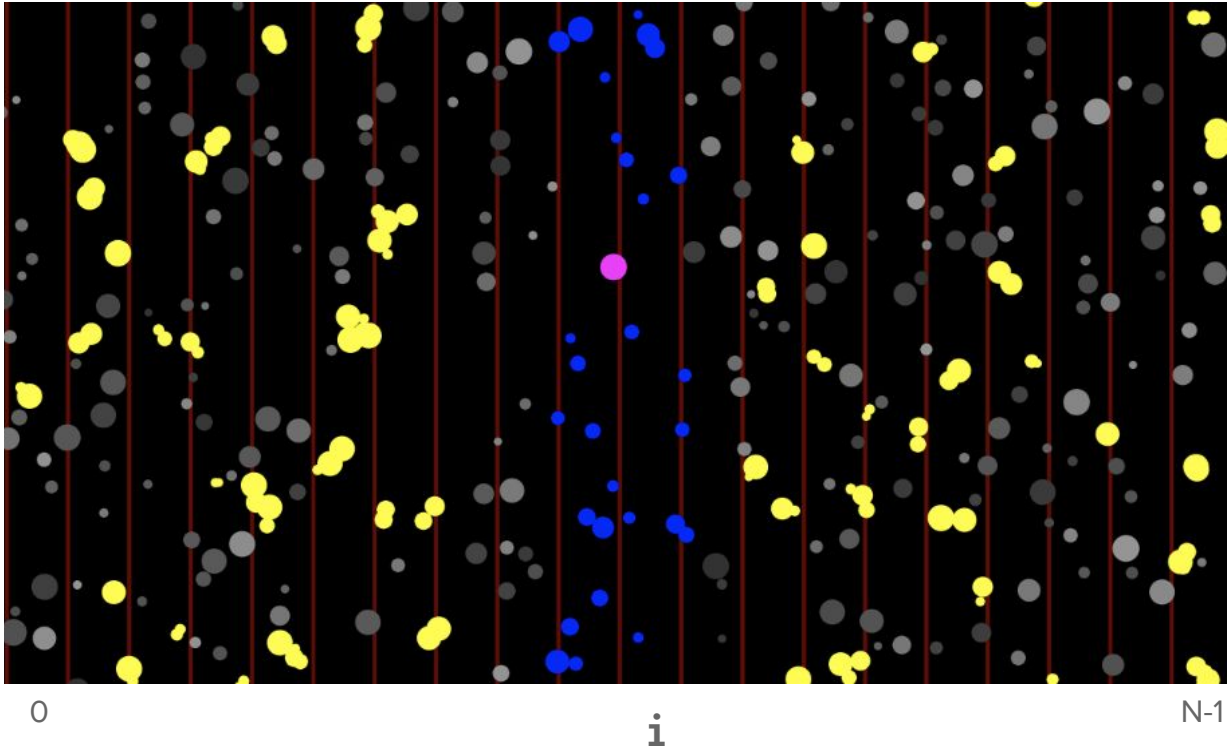
1D Uniform Spatial Subdivision

Overlap Testing:

- Given test bound
- Find overlapping cells, and for each bound
  - Do overlap test
- Return overlapping results as a set.

Q: Can duplicate overlaps occur?

Weakness of 1D subdivision?



Broad-Phase Tests

2D Uniform Spatial Subdivision

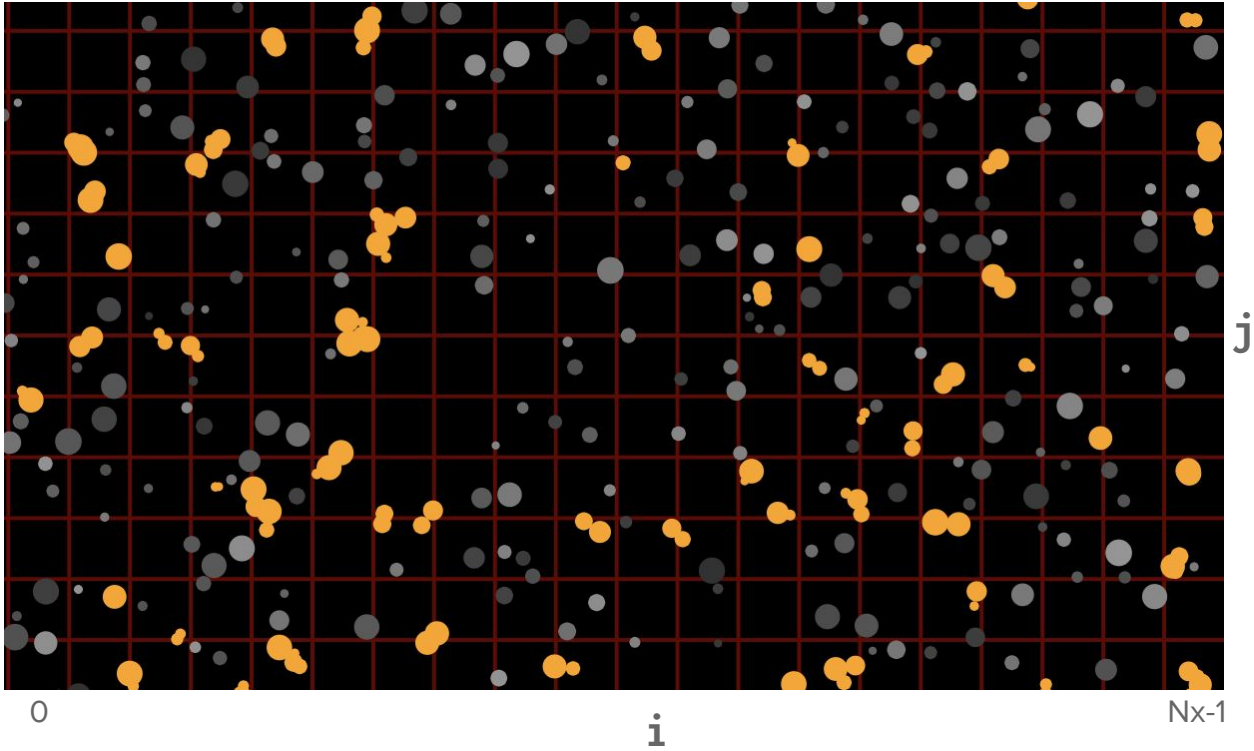
Construction:

- Divide space into Nx-by-Ny bins of constant width, h (or hx & hy)
- Add each object to each bin that its bounding volume overlaps:
  - Use 1D overlap tests

Cell Index (i,j): Given coords x & y,

`i = floor(x/h`$_x$`)` clamped to [0,Nx-1],

`J = floor(y/h`$_y$`)` clamped to [0,Ny-1].



Broad-Phase Tests

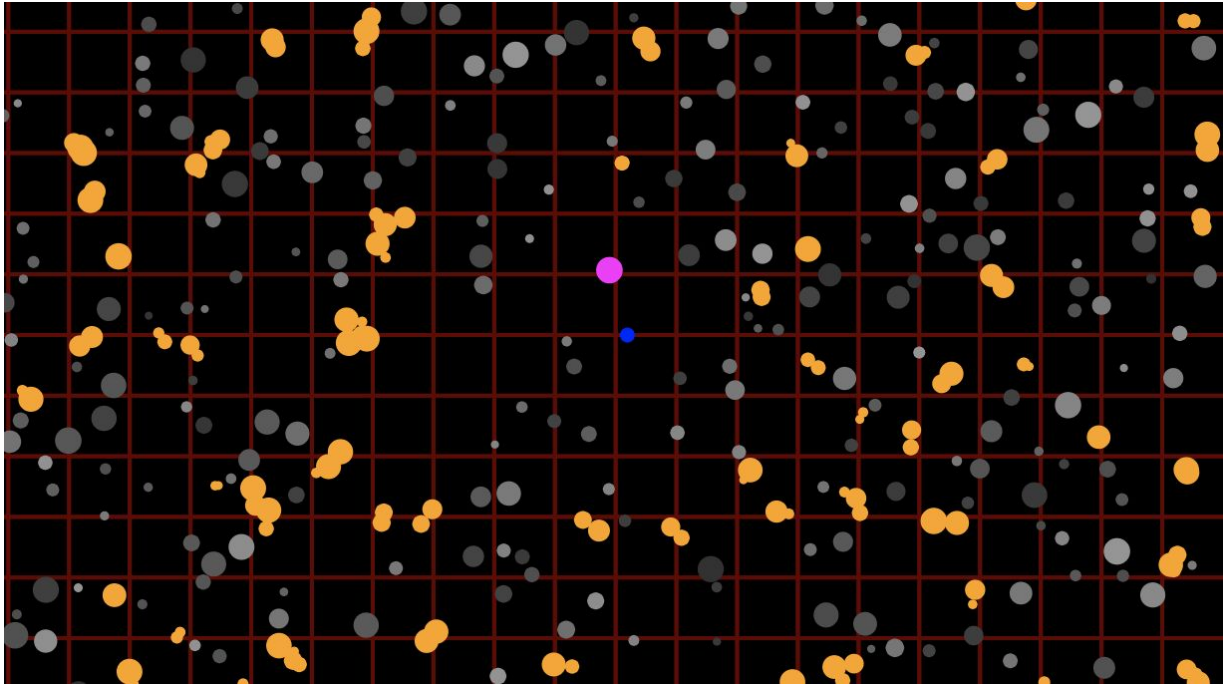
2D Uniform Spatial Subdivision

Overlap Testing:

- Given test bound
- Find overlapping cells, and for each bound
  - Do overlap test
- Return overlapping results as a set.

Q: Can duplicate overlaps occur?

Weakness of 2D subdivision?



Broad-Phase Tests

2D Uniform Spatial Subdivision

GridCheck2D:

- Simplified implementation. Just drop in your sketch.
- See Sketch: https://www.openprocessing.org/sketch/973557

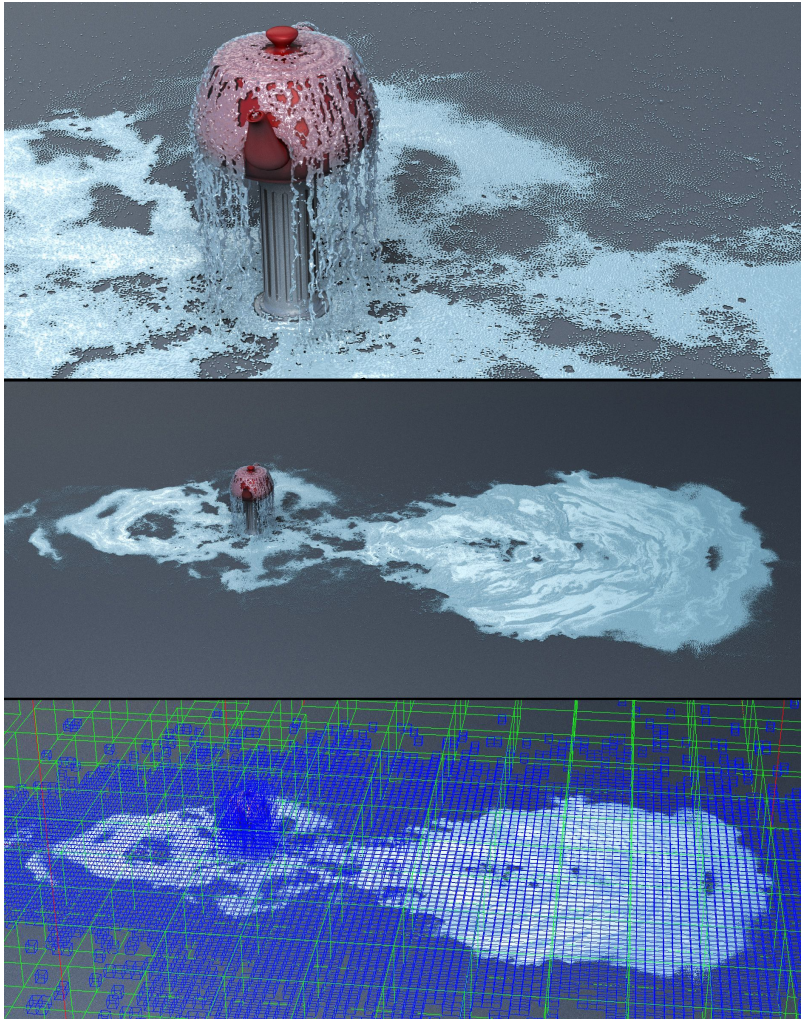
```
// A simple uniform 2D subdivision for interval-overlap tests on [0,width]x[0,height].
// Given an object and rectangular interval, [xmin,xmax]x[ymin,ymax], the object is stored in cells it overlaps.
// A test rectangle can be used to find objects with overlapping intervals.
// Doug L. James, Sept 2020, http://graphics.stanford.edu/~djames
class GridCheck2D {

  constructor(nXCells, width, nYCells, height) {
    this.nXCells       = nXCells;
    this.nYCells       = nYCells;
    this.nXCellsMinus1 = nXCells-1; // used often
    this.nYCellsMinus1 = nYCells-1; // used often
    this.dx     = width /this.nXCells;
    this.dy     = height/this.nYCells;
    this.invDX  = 1/this.dx; // used often
    this.invDY  = 1/this.dy; // used often
    this.width  = width;
    this.height = height;
```

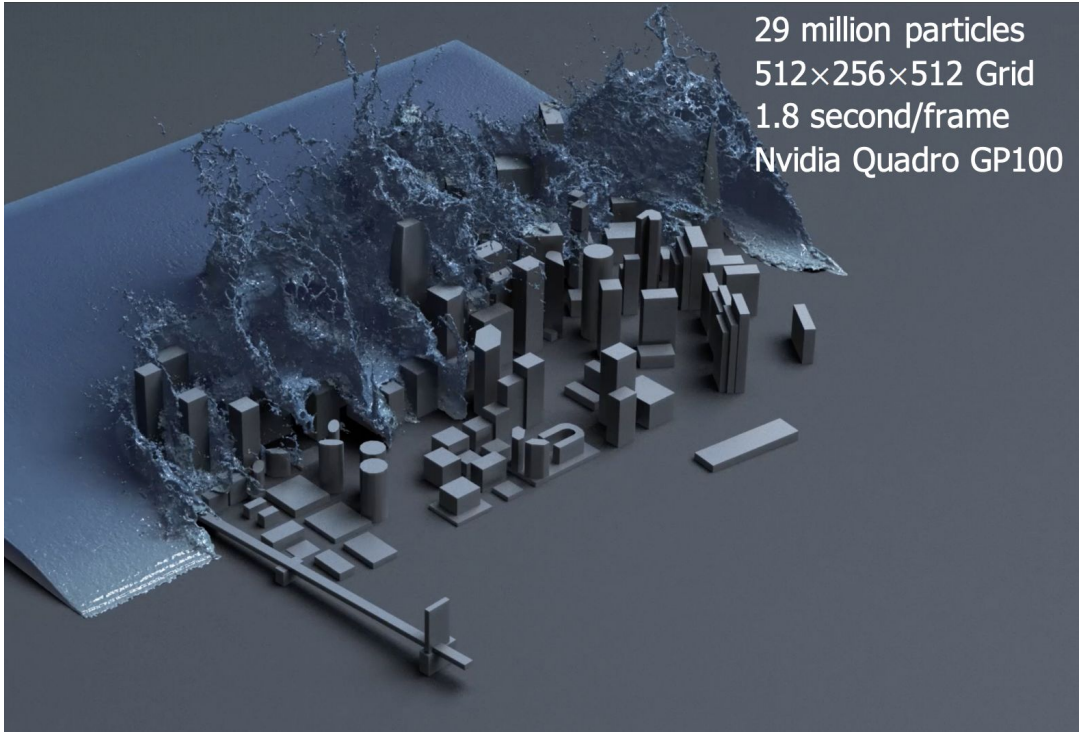
Broad-Phase Tests

Other spatial subdivisions

- **Spatial Hashing:** Virtual grid of *unbounded* size. Use a hashtable to map cells to a finite size storage container.
  - Basic Idea:
    - Map cell (i,j) to a hashcode
      - e.g., key = N*(i%N) + j%N (or something more random)
    - Insert/retrieve contents using hashtable key.
      - Watch for “hashcode collisions,” e.g., due to wraparound. OK, since testing bounds.
  - GPU acceleration very effective. Many schemes.

- **Adaptive grids:** Adapts cell sizes to object distribution. Octrees/Quadtrees. Kd-trees. Hybrid schemes.
  - OpenVDB: Sparse volume data structure





From “Fast Fluid Simulation with Sparse Volumes on the GPU” [Wu et al. 2018]