

References

- *Collision Detection between Geometric Models: A Survey*, by M. Lin and S. Gottschalk, Proc. of IMA Conference on Mathematics of Surfaces 1998.
- *I-COLLIDE: Interactive and Exact Collision Detection for Large-Scale Environments*, by Cohen, Lin, Manocha & Ponamgi, Proc. of ACM Symposium on Interactive 3D Graphics, 1995. (More details in Chapter 3 of M. Lin's Thesis)
- *A Fast Procedure for Computing the Distance between Objects in Three-Dimensional Space*, by E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, In IEEE Transaction of Robotics and Automation, Vol. RA-4:193--203, 1988.

Geometric Proximity Queries

- **Given two object, how would you check:**
 - *If they intersect with each other while moving?*
 - *If they do not interpenetrate each other, how far are they apart?*
 - *If they overlap, how much is the amount of penetration*

Collision Detection

- Update configurations w/ TXF matrices
- Check for edge-edge intersection in 2D
(Check for edge-face intersection in 3D)
- Check every point of A inside of B &
every point of B inside of A
- Check for pair-wise edge-edge intersections

Imagine larger input size: $N = 1000+$

Classes of Objects & Problems

- 2D vs. 3D
- Convex vs. Non-Convex
- Polygonal vs. Non-Polygonal
- Open surfaces vs. Closed volumes
- Geometric vs. Volumetric
- Rigid vs. Non-rigid (deformable/flexible)
- Pairwise vs. Multiple (N-Body)
- CSG vs. B-Rep
- Static vs. Dynamic

And so on... This may include other geometric representation schemata, etc.

Some Possible Approaches

- Geometric methods
- Algebraic Techniques
- Hierarchical Bounding Volumes
- Spatial Partitioning
- Others (e.g. optimization)

Voronoi Diagrams

- Given a set S of n points in R^2 , for each point p_i in S , there is the set of points (x, y) in the plane that are closer to p_i than any other point in S , called Voronoi polygons. The collection of n Voronoi polygons given the n points in the set S is the "*Voronoi diagram*", $Vor(S)$, of the point set S .

Intuition: To partition the plane into regions, each of these is the set of points that are closer to a point p_i in S than any other. The partition is based on the set of closest points, e.g. bisectors that have 2 or 3 closest points.

Generalized Voronoi Diagrams

- The extension of the Voronoi diagram to higher dimensional features (such as edges and facets, instead of points); i.e. the set of points closest to a *feature*, e.g. that of a polyhedron.
- FACTS:
 - In general, the generalized Voronoi diagram has quadratic surface boundaries in it.
 - If the polyhedron is convex, then its generalized Voronoi diagram has planar boundaries.

Voronoi Regions

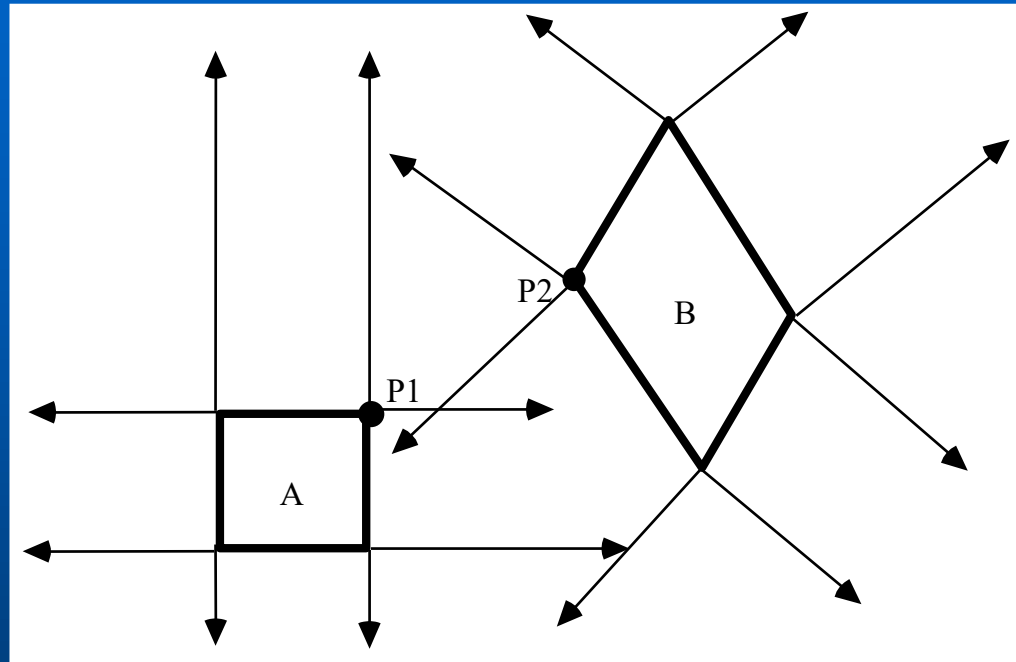
- A Voronoi region associated with a *feature* is a set of points that are closer to that feature than any other.
- FACTS:
 - The Voronoi regions form a partition of space outside of the polyhedron according to the closest feature.
 - The collection of Voronoi regions of each polyhedron is the generalized Voronoi diagram of the polyhedron.
 - The generalized Voronoi diagram of a convex polyhedron has linear size and consists of polyhedral regions. And, all Voronoi regions are convex.

Voronoi Marching

Basic Ideas:

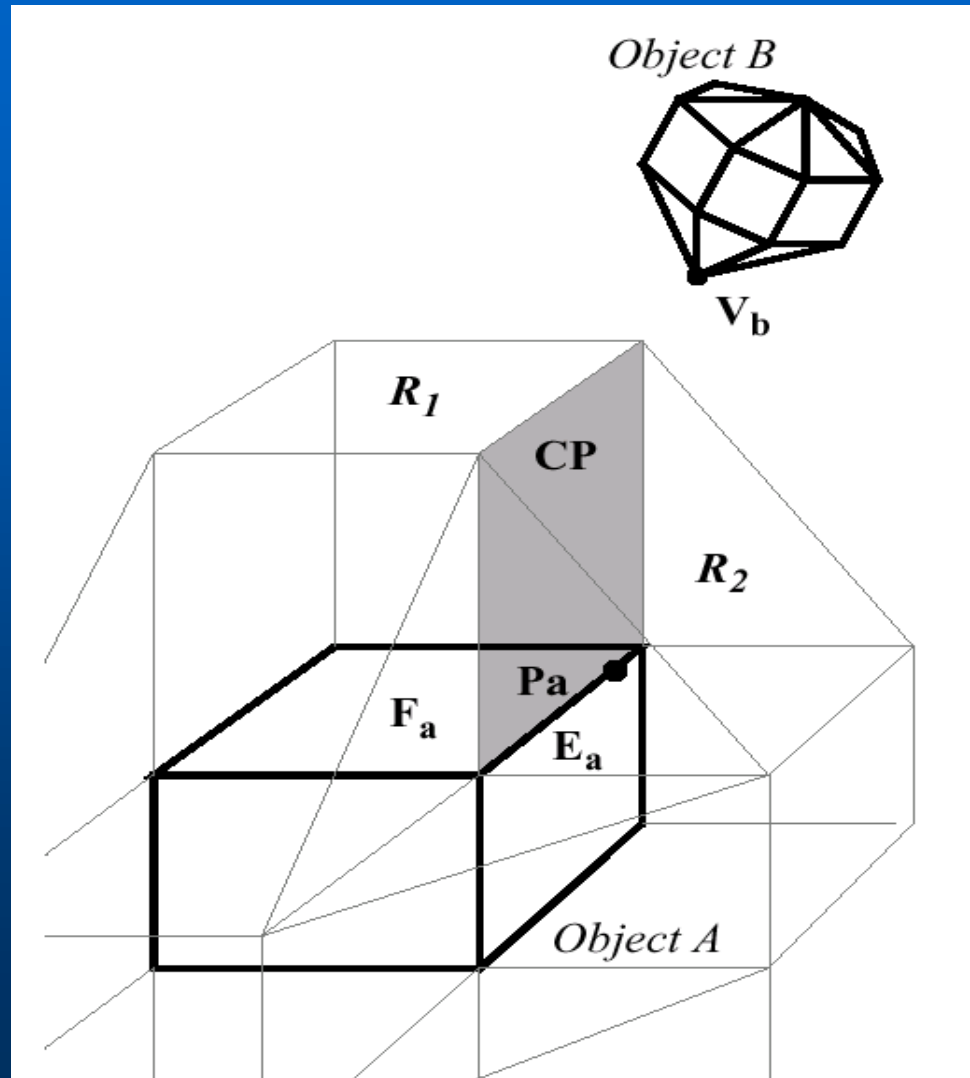
- **Coherence:** local geometry does not change much, when computations repetitively performed over successive small time intervals
- **Locality:** to "*track*" the pair of closest features between 2 moving convex polygons(polyhedra) w/ Voronoi regions
- **Performance:** expected **constant** running time, independent of the geometric complexity

Simple 2D Example



Objects A & B and their Voronoi regions: P1 and P2 are the pair of closest points between A and B. Note P1 and P2 lie within the Voronoi regions of each other.

Basic Idea for Voronoi Marching



Linear Programming

In general, a d -dimensional linear programming (or linear optimization) problem may be posed as follows:

- Given a finite set A in R^d
- For each a in A , a constant K_a in R , c in R^d
- Find x in R^d which minimize $\langle x, c \rangle$
- Subject to $\langle a, x \rangle \geq K_a$, for all a in A .

where $\langle *, * \rangle$ is standard inner product in R^d .

LP for Collision Detection

Given two finite sets A, B in R^d

For each a in A and b in B ,

Find x in R^d which minimize *whatever*

Subject to $\langle a, x \rangle > 0$, for all a in A

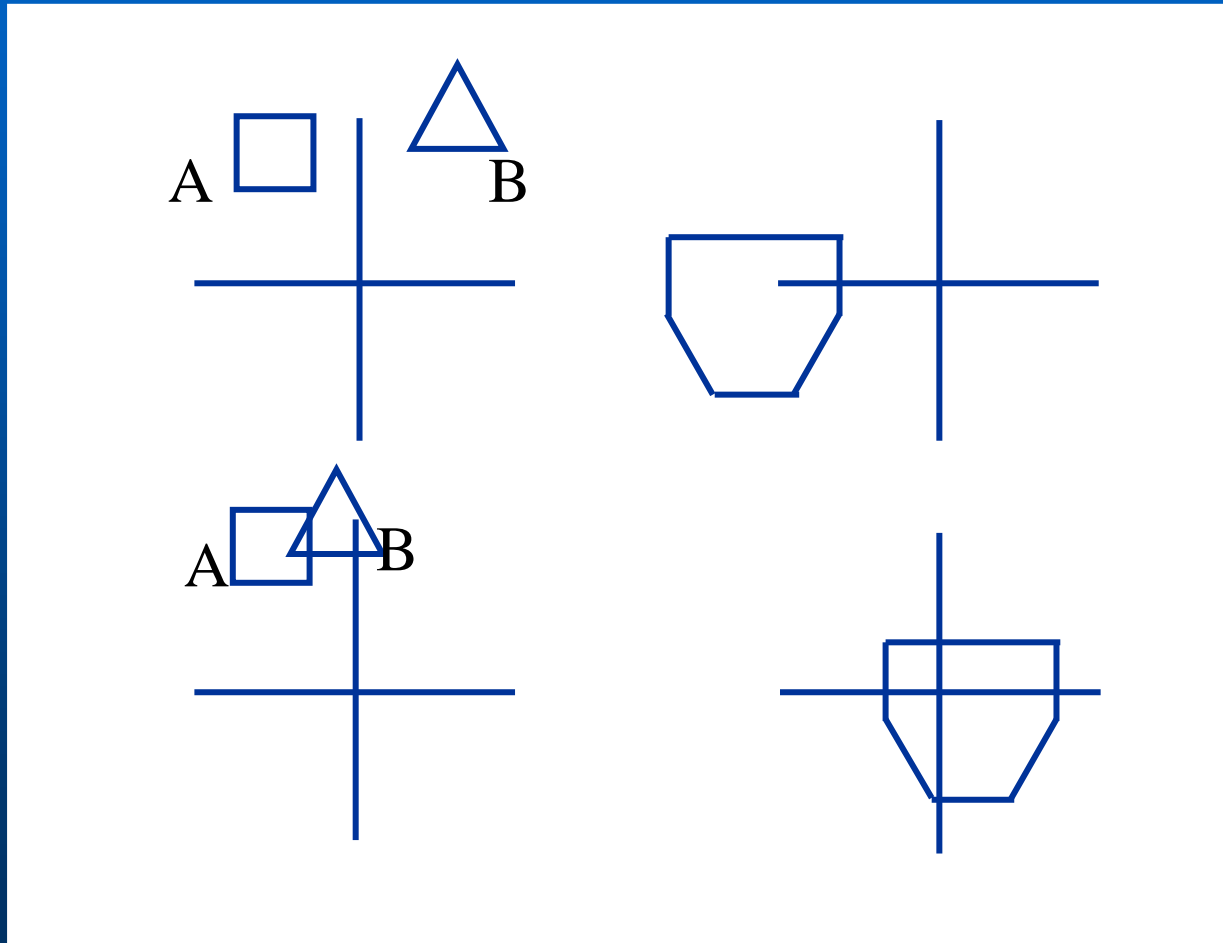
And $\langle b, x \rangle < 0$, for all b in B

where $d = 2$ (or 3).

Minkowski Sums/Differences

- Minkowski Sum $(A, B) = \{ a + b \mid a \in A, b \in B \}$
- Minkowski Diff $(A, B) = \{ a - b \mid a \in A, b \in B \}$
- A and B collide iff Minkowski Difference(A,B) contains the point 0.

Some Minkowski Differences



Minkowski Difference & Translation

- $\text{Minkowski-Diff}(\text{Trans}(A, t_1), \text{Trans}(B, t_2)) = \text{Trans}(\text{Minkowski-Diff}(A, B), t_1 - t_2)$
- ⇒ $\text{Trans}(A, t_1)$ and $\text{Trans}(B, t_2)$ intersect iff $\text{Minkowski-Diff}(A, B)$ contains point $(t_2 - t_1)$.

Properties

- Distance

- $\text{distance}(A,B) = \min_{a \in A, b \in B} \|a - b\|_2$
- $\text{distance}(A,B) = \min_{c \in \text{Minkowski-Diff}(A,B)} \|c\|_2$
- if A and B disjoint, c is a point on boundary of Minkowski difference

- Penetration Depth

- $\text{pd}(A,B) = \min\{ \|t\|_2 \mid A \cap \text{Translated}(B,t) = \emptyset \}$
- $\text{pd}(A,B) = \min_{t \notin \text{Minkowski-Diff}(A,B)} \|t\|_2$
- if A and B intersect, t is a point on boundary of Minkowski difference

Practicality

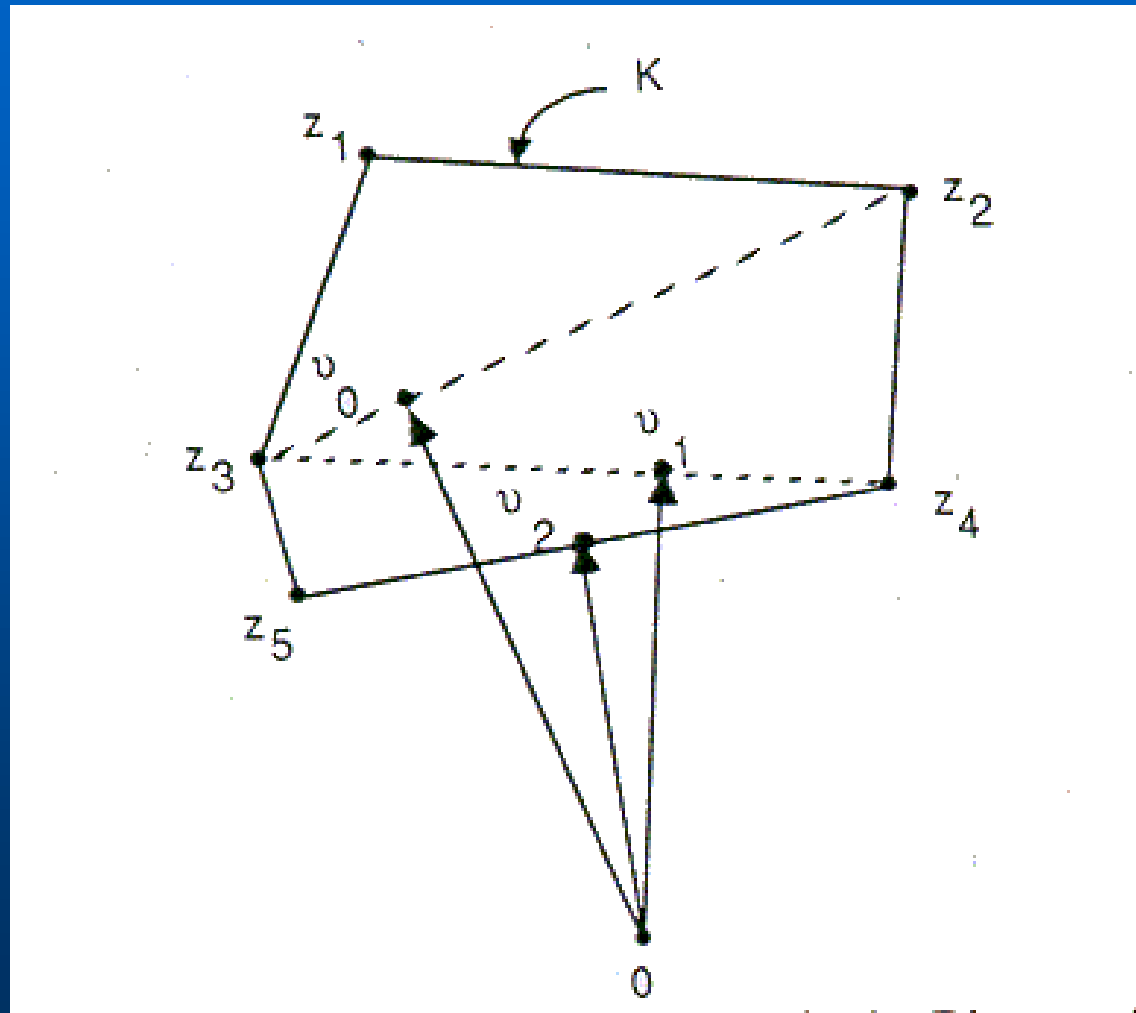
- Expensive to compute boundary of Minkowski difference:
 - For convex polyhedra, Minkowski difference may take $O(n^2)$
 - For general polyhedra, no known algorithm of complexity less than $O(n^6)$ is known

GJK for Computing Distance between Convex Polyhedra

GJK-DistanceToOrigin (P) // dimension is m

1. Initialize P_0 with $m+1$ or fewer points.
2. $k = 0$
3. while (TRUE) {
4. if origin is within $\text{CH}(P_k)$, return 0
5. else {
6. find $x \in \text{CH}(P_k)$ closest to origin, and $S_k \subset P_k$ s.t. $x \in \text{CH}(S_k)$
7. see if any point p_{-x} in P more extremal in direction $-x$
8. if no such point is found, return $|x|$
9. else {
10. $P_{k+1} = S_k \cup \{p_{-x}\}$
11. $k = k + 1$
12. }
13. }
14. }

An Example of GJK



Running Time of GJK

- Each iteration of the while loop requires $O(n)$ time.
- $O(n)$ iterations possible. The authors claimed between 3 to 6 iterations on average for any problem size, making this “expected” linear.
- Trivial $O(n)$ algorithms exist if we are given the boundary representation of a convex object, but GJK will work on point sets - computes CH lazily.

More on GJK

Given $A = \text{CH}(A')$ $A' = \{a_1, a_2, \dots, a_n\}$ and
 $B = \text{CH}(B')$ $B' = \{b_1, b_2, \dots, b_m\}$

- Minkowski-Diff(A, B) = $\text{CH}(P)$, $P = \{a - b \mid a \in A', b \in B'\}$
- Can compute points of P on demand:
 - $p_{-x} = a_{-x} - b_x$ where a_{-x} is the point of A' extremal in direction $-x$, and b_x is the point of B' extremal in direction x .
- The loop body would take $O(n + m)$ time, producing the “expected” linear performance overall.

Large, Dynamic Environments

- For dynamic simulation where the velocity and acceleration of all objects are known at each step, use the scheduling scheme (implemented as heap) to prioritize “critical events” to be processed.
- Each object pair is tagged with the estimated time to next collision. Then, each pair of objects is processed accordingly. The heap is updated when a collision occurs.

Scheduling Scheme

- a_{max} : an upper bound on relative acceleration between any two *points* on any pair of objects.
- a_{lin} : relative absolute linear
- α : relative rotational accelerations
- ω : relative rotational velocities
- r : vector difference btw CoM of two bodies
- d : initial separation for two given objects

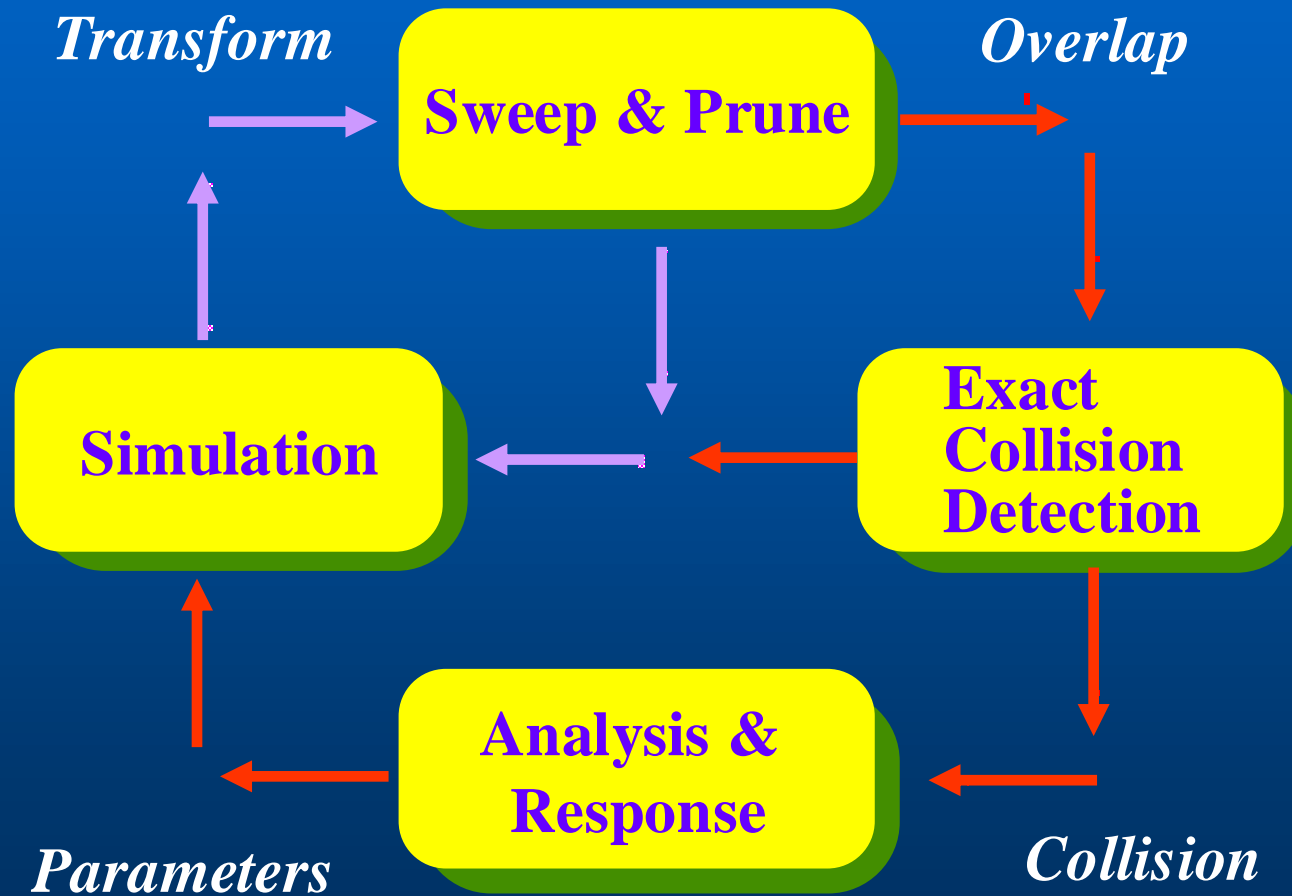
$$a_{max} = | a_{lin} + \alpha \times r + \omega \times \omega \times r |$$

$$v_i = | v_{lin} + \omega \times r |$$

- Estimated Time to collision:

$$t_c = \{ (v_i^2 + 2 a_{max} d)^{1/2} - v_i \} / a_{max}$$

Collide System Architecture

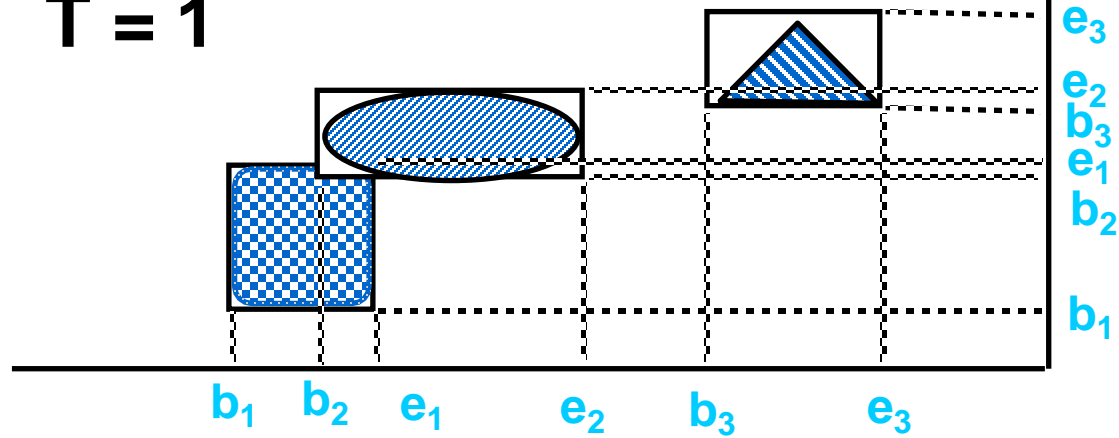


Sweep and Prune

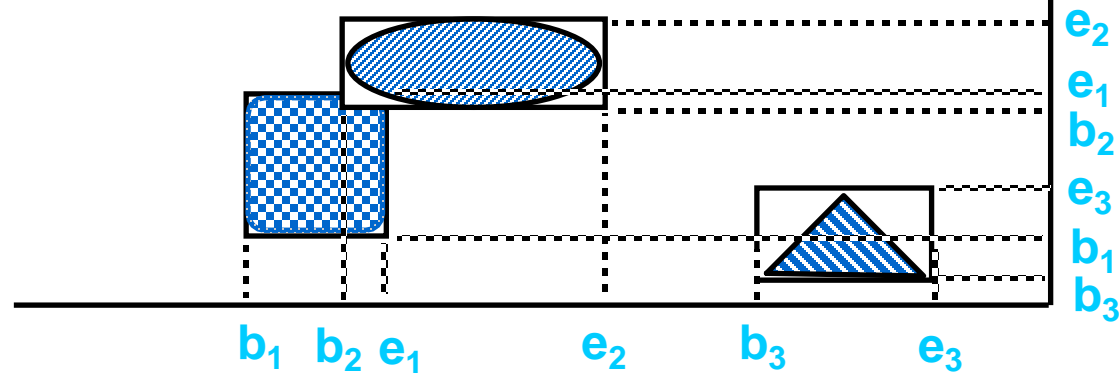
- Compute the axis-aligned bounding box (fixed vs. dynamic) for each object
- Dimension Reduction by projecting boxes onto each x , y , z - axis
- Sort the endpoints and find overlapping intervals
- Possible collision -- only if projected intervals overlap in all 3 dimensions

Sweep & Prune

T = 1



T = 2



Updating Bounding Boxes

- **Coherence** (greedy algorithm)
- **Convexity properties** (geometric properties of convex polytopes)
- **Nearly constant time**, if the motion is relatively “small”

Use of Sorting Methods

- **Initial sort** -- quick sort runs in $O(m \log m)$ just as in any ordinary situation
- **Updating** -- insertion sort runs in $O(m)$ due to coherence. We sort an almost sorted list from last stimulation step. In fact, we look for “swap” of positions in all 3 dimension.

Implementation Issues

- Collision matrix -- basically *adjacency matrix*
- Enlarge bounding volumes with some tolerance threshold
- Quick start polyhedral collision test -- using bucket sort & look-up table