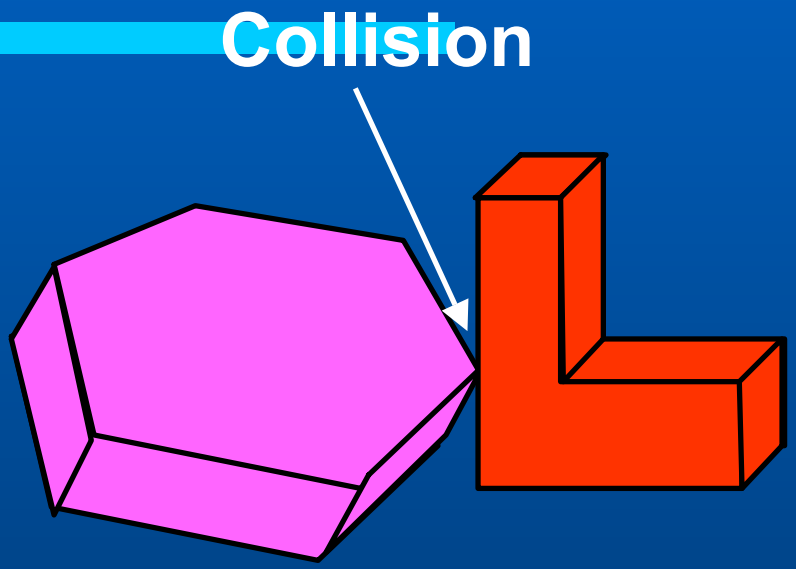
Collision Detection for Real-Time Simulation

Eric Larsen

SCEA R&D

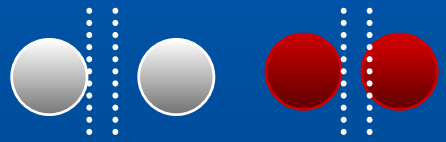
Problem Definition



- Finding the time, place, depth, or existence of contact between pairs of objects in a simulation

Common Subroutines in a Collision System

- Collision detection: do two models overlap?
- Gap detection: does a given gap exist between two models?
- Distance computation: what is the distance between the models?
- Penetration depth computation: what is the amount of penetration?
- Time of impact: what will be the time of impact of two moving bodies?



Variations of Collision Systems

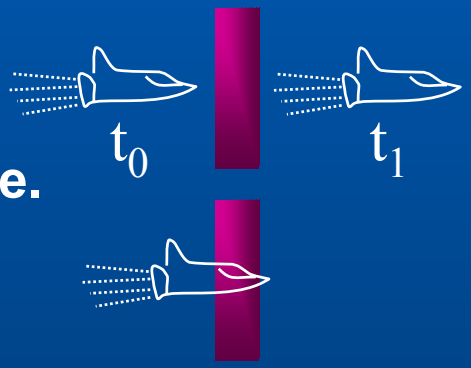
- **Collision Scheduling Methods**
  - methods for choosing when to do collision checks
- **Object Representations**
  - geometric models for objects in a simulator
- **Broad & Narrow Phase Methods [Hubbard93]**
  - Broad phase: methods for culling collision checks
  - Narrow phase: methods for performing each collision check

Variations of Collision Systems

- **Collision Scheduling Methods**
  - **Fixed timestep**
  - **Adaptive timestep**
    - **Bisection**
    - **Prediction**

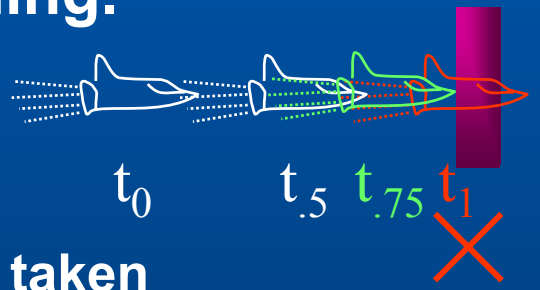
Fixed Timestep

- **Do collision checking at regular time intervals**
  - **Pros:**
    - **Collision scheduling is simple**
    - **Performance is relatively predictable.**
  - **Cons:**
    - **Can miss collisions**
    - **Allows interpenetration of solids**
    - **Needs penetration distance or some heuristic for computing a separating impulse**
    - **Several opposing impulses can keep objects interpenetrating**



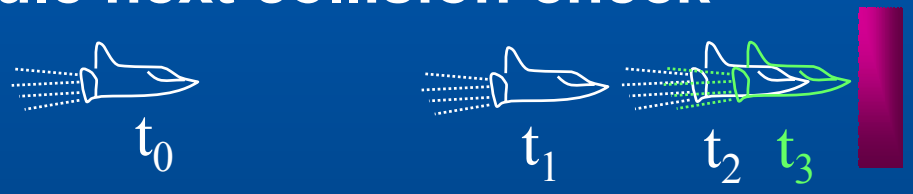
Adaptive Timestep 1:Bisection

- **When collision detected, bisect preceding time interval until models are separated but close enough to be considered colliding.**
  - **Pros:**
    - **Prevents penetrations - time is backed up to earliest collision before another forward step is taken**
  - **Cons:**
    - **Can still miss collisions**
    - **Rapid collisions will stunt advance of time**



Adaptive Timestep 2: Prediction

- **Compute lower bound to time of impact (TOI) for a pair of models & schedule next collision check at this time.**
  - **Pros:**
    - **Delays collision checks for distant models**
    - **Prevents penetration** and **missed collisions**
  - **Cons:**
    - **Close contact increases frequency of checks**
    - **TOI's require fixed or bounded motion paths - user input or collision can invalidate all TOIs connected with a particular object.**



Variations of Collision Systems

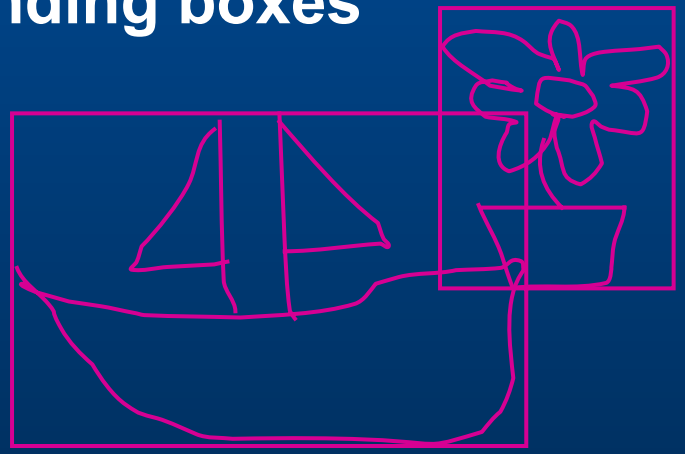
- **Object Representations**
  - **Basic primitives**
    - **e.g., spheres, cones, cylinders, boxes**
  - **Polygons/polyhedra**
    - **Polygon soups**
    - **Convex polyhedra**
    - **Closed meshes**
  - **CSG, implicit rep., parametric rep.**
  - **Unions**

Variations of Collision Systems

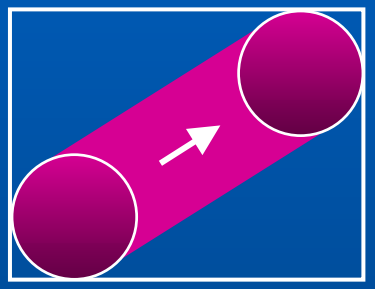
- **Broad and Narrow Phases [Hubbard93]**
  - **Broad phase:**
    - **bounding box methods**
      - **uniform grid**
      - **hierarchical hash tables**
      - **sweep-and-prune**
  - **Narrow phase:**
    - **convex models**
      - **distance / penetration depth**
    - **polygon soups**
      - **collision / distance**

Bounding Boxes

- A very common broad-phase tool
- Stationary object bounding box
  - relevant to fixed timestep scheduling
  - place an axes-aligned box around each object at a given instant
  - only do narrow phase collision checking on pairs of models whose bounding boxes overlap



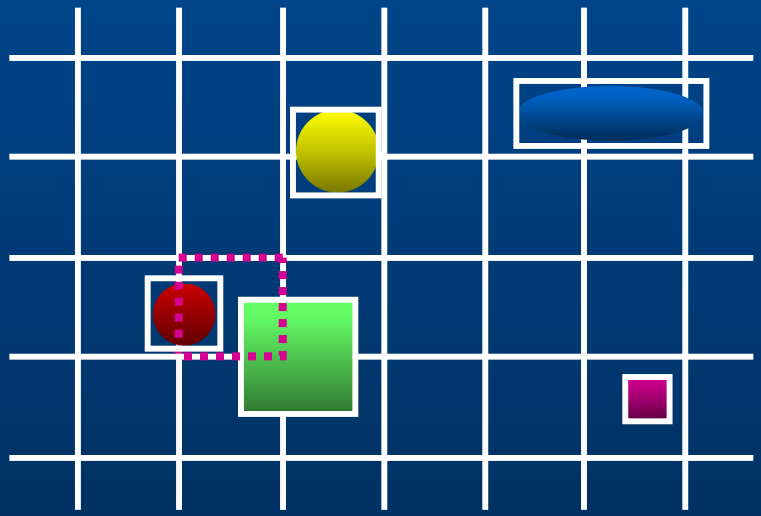
Bounding Boxes



- Moving object bounding box [Mirtich96]
  - bound sweep of object between start and finish time
  - overlapping boxes indicate possible collisions during that time interval
  - applicable to prediction scheduling
    - ordinarily need all TOIs to take a collision-free time step.
    - instead, take a tentative step forward, but compute TOIs for objects whose sweep bounding boxes overlap.
    - truly advance to smallest TOI found

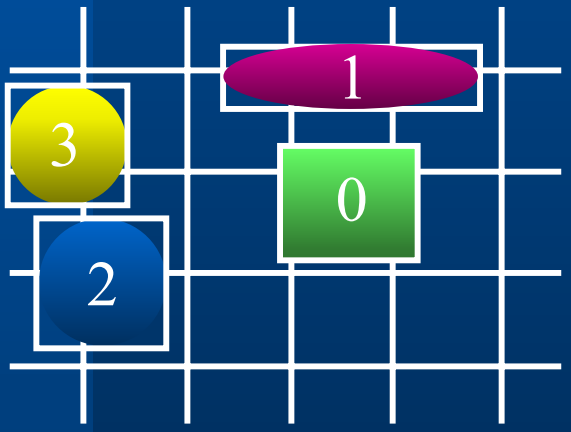
Uniform Grid

- Need to avoid $O(n^2)$ box tests with n boxes
- Uniform Grid
  - divide space into regular grid of cells
  - bucket each object box into cells it overlaps
  - compare boxes in same cell only - called *close* boxes in following slides.



Uniform Grid

- **Two tricks:**
  - **if grid infinite, store only the finite number of nonempty buckets in a hash table**
  - **several cells may contain the same pair of boxes: to report *close* pairs only once, count number of times pair found *close* in 2D table of close-counters [Mirtich96].**

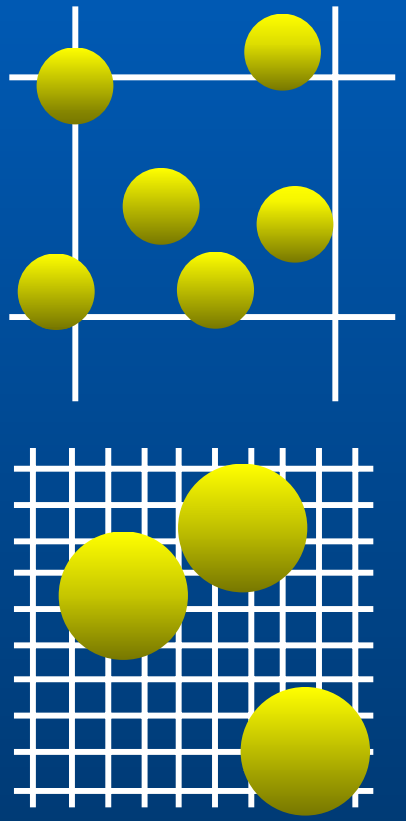


2D table

	0	1	2	3
0		3	0	0
1			0	0
2				2
3				

Uniform Grid

- **Problem - choosing cell size**
  - **If too large, many or all objects can fall in the same cell - still $O(n^2)$ cost**
  - **If too small, each object covers many cells**
    - **Hash table size grows**
    - **Object updates more costly**
    - **A box pair can share many cells**
  - **If objects vary greatly in size, no cell size solves both problems.**

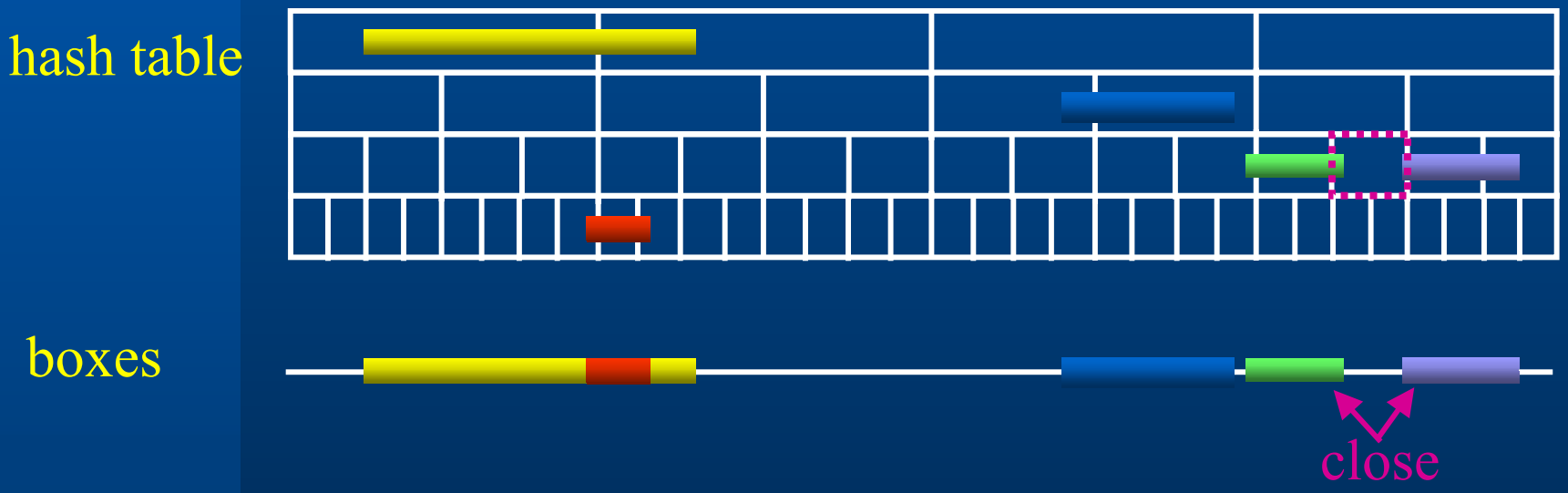


Hierarchical Hash Table [Mirtich96]

- **Uses grids at several resolutions.**
  - **n cell sizes chosen, ρ_1 through ρ_n**
  - **For any box size S, the ratio between S and some cell size ρ_k is bounded between two constants, $0 < \alpha < 1$, $\beta \geq 1$:**
    $\alpha \leq (S / \rho_k) \leq \beta$
  - **i.e., a box takes up at least some fraction α of a cell size, but no more than β cell sizes at its "matching" resolution (denoted res(X), for box X)**

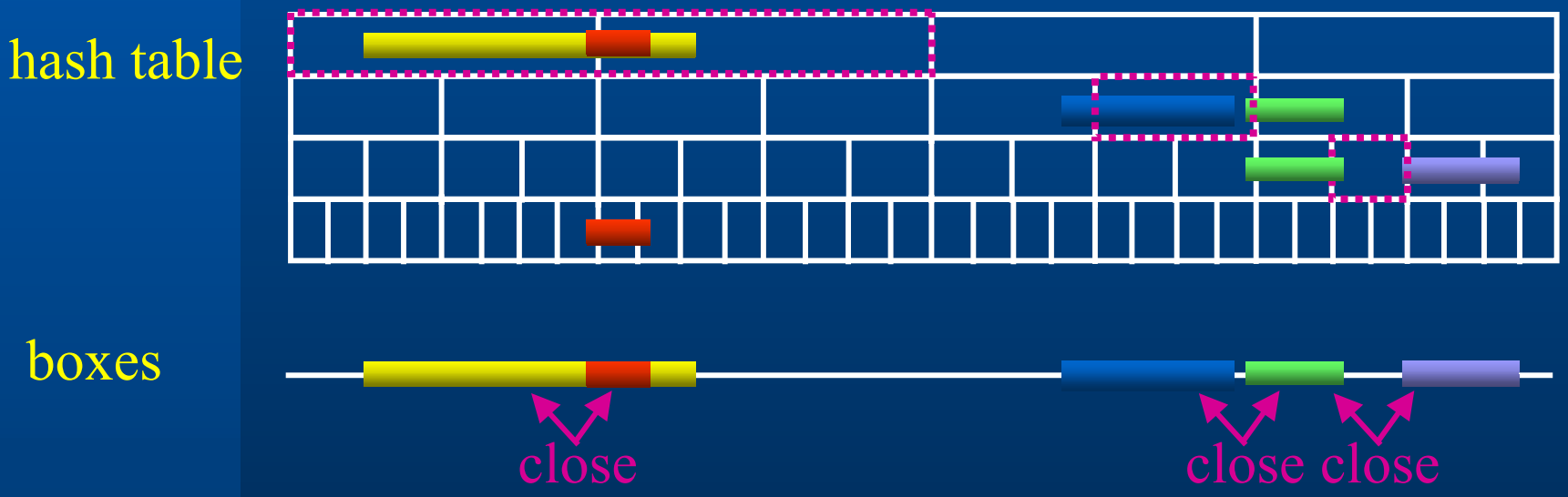
Hierarchical Hash Table [Mirtich96]

- **1D example - boxes are intervals**
  - **box inserted first in buckets at matching resolution - boxes in same buckets are *close***
  - **2D table counts times each pair found *close***

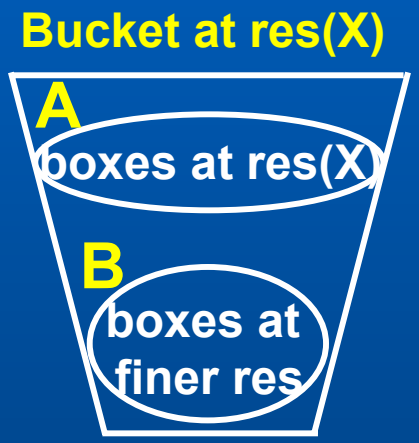


Hierarchical Hash Table [Mirtich96]

- **Boxes at different resolutions may overlap too:**
  - **Extend *close* to mean boxes X,Y that overlap the same bucket at the coarser of the two resolutions, res(X) and res(Y)**



Hierarchical Hash Table [Mirtich96]



- **Finding all *close* boxes**
  - Each bucket has *list A* for boxes at its own resolution and *list B* for boxes at finer resolutions (Mirtich describes one list)
  - For each box X,
    - Insert X in overlapped buckets at res(X):
      - put X in list A
      - any boxes in list A or B of these buckets are *close*
    - Insert X in overlapped buckets at coarser resolutions:
      - put X in list B
      - any boxes in list A of these buckets are *close*
  - Time bound O(n + c) where n is number of boxes and c is number of *close* box pairs

Hierarchical Hash Table [Mirtich96]

- For coherence - preserve hash table, close counters, and list of close boxes between invocations.
  - When box leaves a bucket, decrement close counters for close boxes in that bucket
  - When box enters a bucket, increment close counter for close boxes in that bucket
  - Decrement to zero takes pair off the close pairs list; increment to 1 puts the pair on the list.
  - A box that stays in the same buckets causes no updates to the data structures

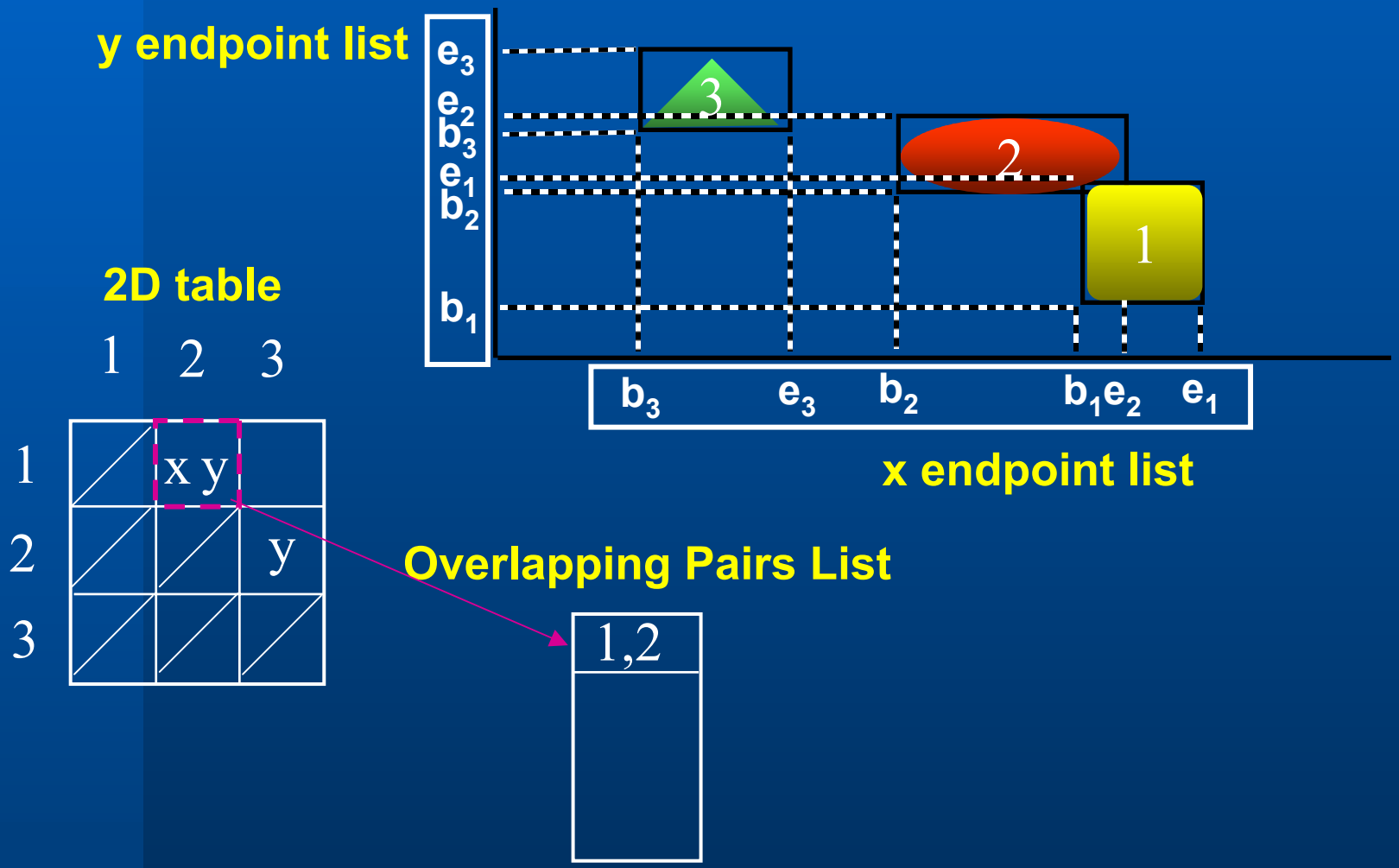
Hierarchical Hash Table [Mirtich96]

- **Extending to dynamically resized boxes:**
  - **Scales of bounding boxes change as simulation progresses.**
  - **Mirtich suggests that basing resolutions on maximum radii of objects is adequate.**
- **Extending to 3D:**
  - **Can create 3D grid at several resolutions.**
  - **Could possibly create three 1D grids, and treat box intervals independently - box pair is close iff it is close in 3 dimensions.**

Sweep and Prune [CLMP95] (I-Collide)

- Different scheme to cull bounding box comparisons - dimension reduction.
- Data structures:
  - On each coordinate axis, a box projects to an interval; keep a sorted list of interval endpoints on each axis.
  - Keep a 2D table that stores for each pair of bounding boxes whether their intervals overlap on each axis.
  - Keep a list of the overlapping boxes

Sweep and Prune [CLMP95] (I-Collide)



Sweep and Prune [CLMP95] (I-Collide)

- **Algorithm:**
  - **Assume data structures are valid for a frame of the simulation.**
  - **As each box is updated for the next frame, modify the endpoints in the sorted lists, using insertion sort to keep lists in sorted order.**
  - **Each swap of a minimum and maximum endpoint toggles overlap status for a pair of box intervals.**
    - **if toggle completes 3 interval overlaps for a box pair, put on overlapping pairs list**
    - **if box pair previously overlapped, take pair off the list**

Sweep and Prune [CLMP95] (I-Collide)

- **Coherence premise:**
  - **With small shifts of box endpoints between frames, few swaps expected in sorted lists.**
  - **Clustering problem - dice on a table [Mirtich96]:**
    - **Each perturbation of a die vertically may require O(n) swaps, resulting in $O(n^2)$ cost.**
  - **Can keep list for 1 axis only**
    - **when a swap makes two intervals overlap, check other two dimensions of the box pair**
    - **Thus, can drop 2 most problematic dimensions - still not a general answer to**

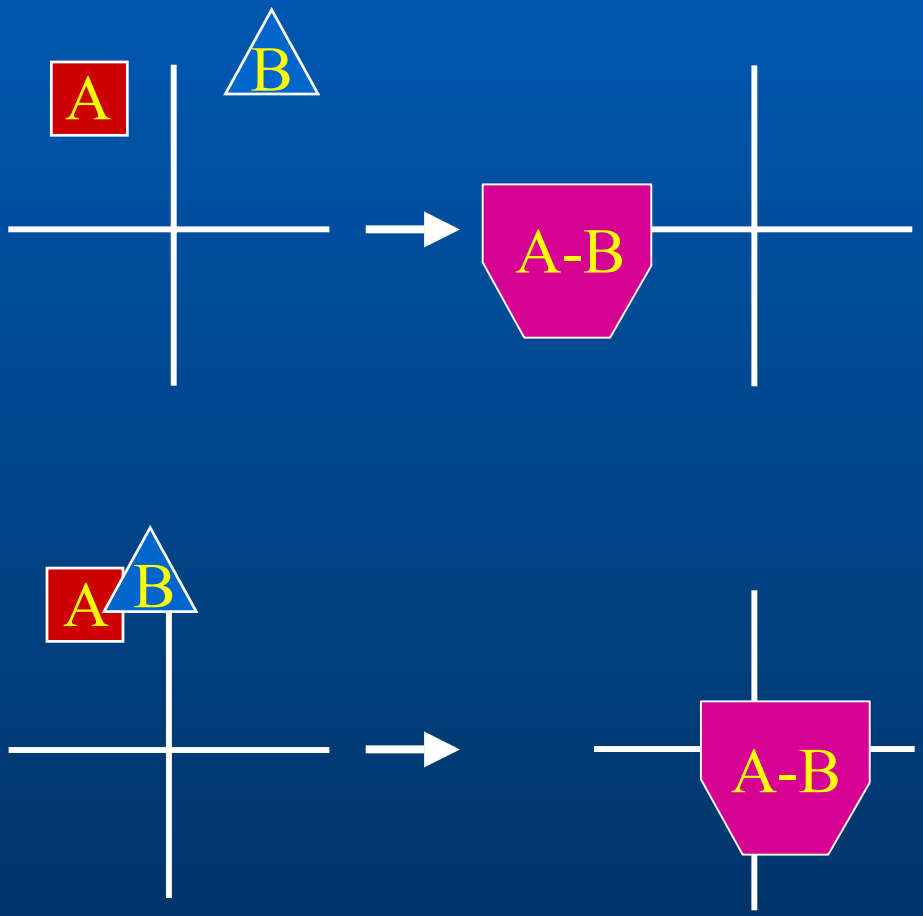
Variations of Collision Systems

- **Narrow phase:**
  - **theory: Minkowski Differences**
  - **convex polyhedra:distance/penetration depth**
    - **GJK [GJK88]**
    - **Lin-Canny [LC91]**
  - **polygon soups: collision/distance**
    - **Sphere Trees [Quinlan94] [Hubbard96]**
    - **OBB Trees [GLM96]**
    - **Swept Sphere Trees [LGLM98]**

Minkowski Sums and Differences

- **Minkowski Sum (A, B) = { a + b | a ∈ A, b ∈ B }**
- **Minkowski Diff (A,B) = { a - b | a ∈ A, b ∈ B } = Minkowski Sum (A, -B)**
- **A and B collide iff Minkowski Diff(A,B) contains origin.**

Some Minkowski Differences



Minkowski Difference and Translation

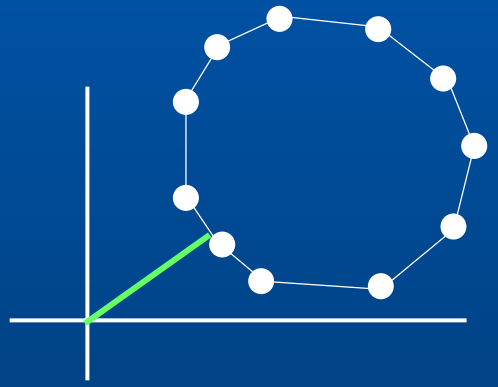
- **Minkowski Diff(Translated(A,**t1**), Translated(B,** t2**)) = Translated (Minkowski Diff(A,B),** t1 **-** t2**)**
- ⇒ **Translated(A,** t1**) and Translated(B,** t2**) intersect iff Minkowski Diff(A,B) contains point** t2 **-** t1**.**

Other Properties

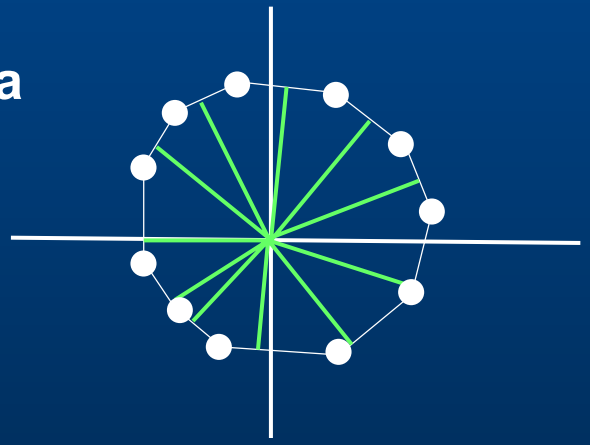
- If A & B convex, Minkowski Diff(A,B) is convex
- Distance:
  - distance(A,B) = $\min_{a \in A,\ b \in B} \| a - b \|_2$
  - distance(A,B) = $\min_{c \in \text{Minkowski-Difference(A,B)}} \| c \|_2$
  - if A and B disjoint, c is closest point to origin on boundary of Minkowski difference
- Penetration-Depth:
  - penetration (A,B) = $\min\{ \| t \|_2 \mid A \cap \text{Translated}(B,t) = \varnothing \}$
  - penetration (A,B) = $\min_{t \notin \text{Minkowski-Difference(A,B)}} \| t \|_2$
  - if A and B intersecting, t is closest point to origin on boundary of Minkowski difference

Other Properties

- However, distance & penetration-depth not equally matched problems, even for convex models:
  - Distance - one local minimum



  - Penetration Depth - many local minima



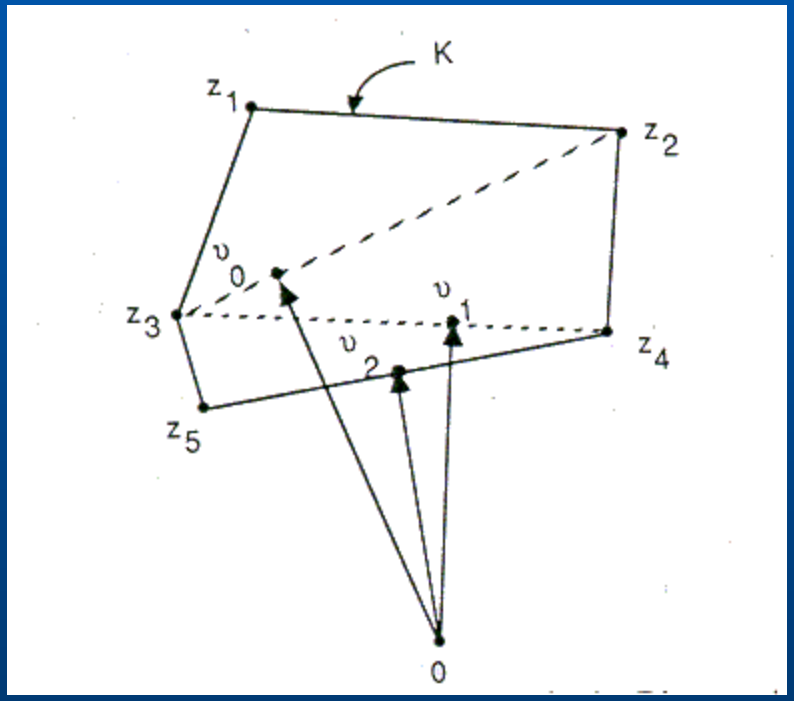
Practicality

- **Expensive to compute boundary of Minkowski Difference:**
  - **structure changes if objects rotate independently**
  - **For two convex polyhedra with m and n vertices, Minkowski Difference may take O(m $\times$ n)**
  - **For polygon soups, no practical algorithm known.**
- **However, GJK algorithm uses Minkowski Difference quite efficiently to find distance, computing it on demand.**

GJK Method for Convex Polyhedra

- GJK-DistanceToOrigin (P) // dimension is m
- 1. Initialize point set P_0 with m+1 or fewer points of P
- 2. k = 0
- 3. while (TRUE) {
- 4. if origin is within CH(P_k), return 0
- 5. else {
- 6. find $x \in CH(P_k)$ closest to origin, and simplex $S_k \subset P_k$ s.t. $x \in CH(S_k)$
- 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. }

GJK - 2D Example



GJK - Running Time

- Each iteration of the while loop requires O(n) time.
- O(n) iterations possible, but authors claim between 3 and 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.
- Also, extends readily to two convex point sets

GJK - Two Convex Objects

- **A = CH(A′) A′ = { a_1, a_2, ... , a_n }**
- **B = CH(B′) B′ = { b_1, b_2, ... , b_m }**

- **Minkowski-Diff(A,B) = CH(P), P = {a - b | a∈ A′, b∈ B′}**
- **Thus, GJK-DistanceToOrigin(P) will find distance(A, B), but P has m × n points.**
- **Solution - 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 now take O(n + m) time, producing the same expected linear performance overall.**

GJK - Extensions

- **Penetration Depth [Cameron97]**
  - **Estimate penetration depth by finding some point on Minkowski Difference boundary.**
  - **Highest quality estimates with coherence and shallow intersections.**
- **Coherence**
  - **Models may transform very little between distance checks.**
  - **Cache previous closest points, search neighborhood of closest points find new closest points**

Lin-Canny Method for Convex Polyhedra

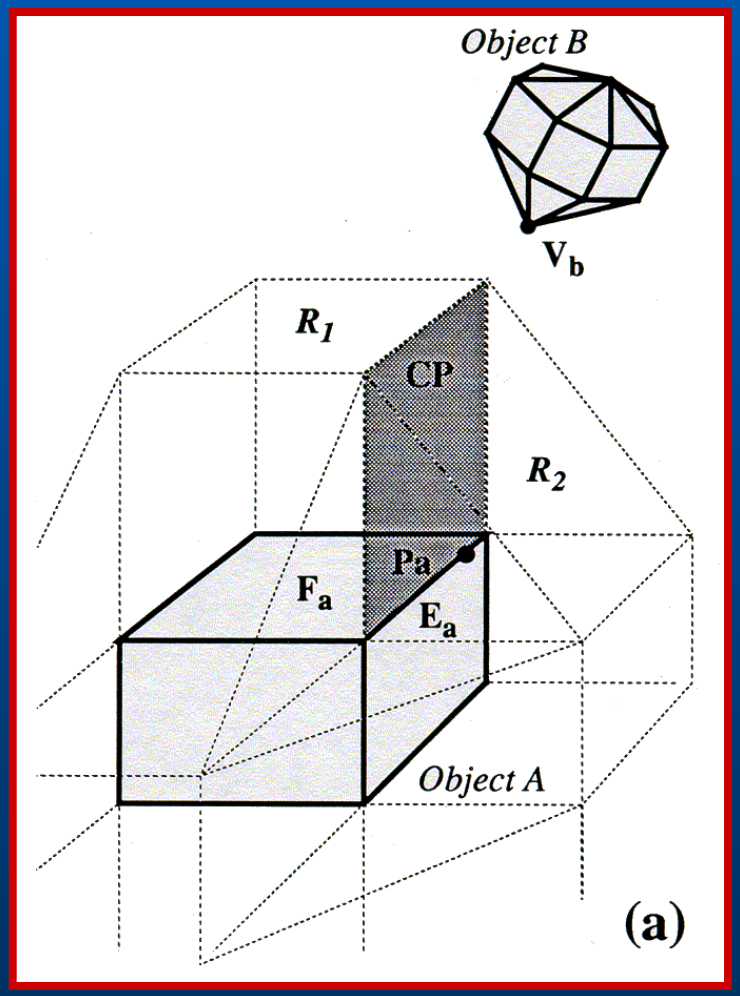
- **Foundations:**
  - **Coherence in object transforms between distance checks.**
  - **One can confirm that two features (edges, vertices, faces) are the closest points of two convex objects in constant time, given some preprocessing of the polyhedra.**
- **Method:**
  - **track closest points**
  - **after each transformation, make expected constant time adjustment of closest points**

Voronoi Regions

- **Localized verification of closest features made possible by precomputing an external Voronoi diagram for the polyhedron**

- **External Voronoi diagram - divide space outside polyhedron into regions, such that points inside each region are closest to a corresponding “feature” - a face, edge, or vertex - of the polyhedron.**

Voronoi Regions

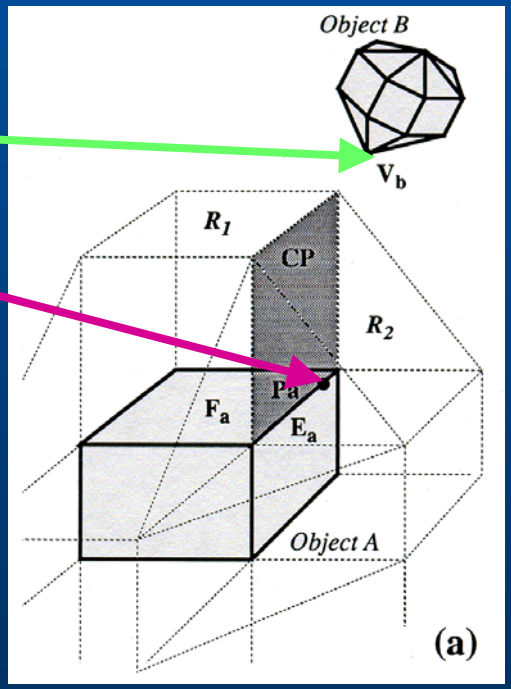


Lin-Canny Algorithm

- Given one feature from each polyhedron, find the closest points of the two features. If each point is in the Voronoi region of the other feature, closest features have been found.

V_b in Voronoi(E_a)

P_a in Voronoi(V_b)



Lin-Canny Algorithm

- **Otherwise, one of the points (call its feature F) is in the Voronoi region of another feature F′, and therefore closer to it. Can select F and F′ as next candidate feature pair.**

Lin-Canny Running Time

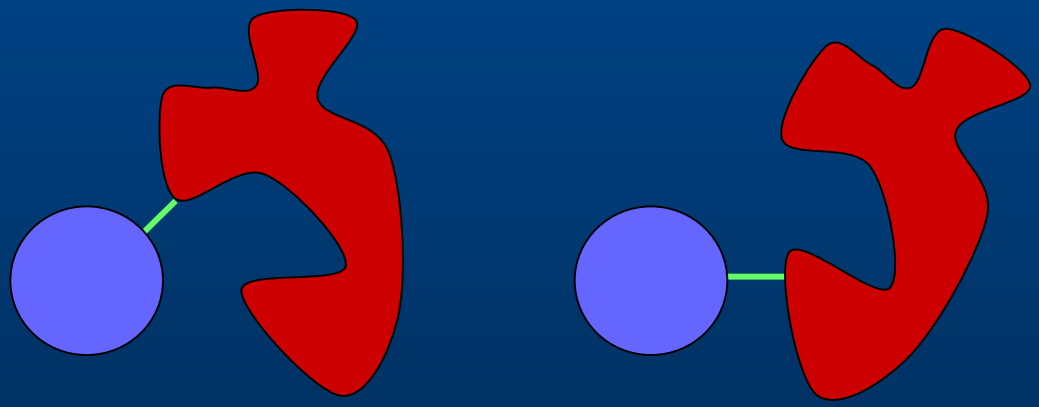
- Distance strictly decreases with each change of feature pair, and no pair of features can be selected twice.
- Worst case $O(m \times n)$ pairs checked.
- Convergence to closest pair typically much better:
  - "near" constant time in simulations with coherence.
  - Closer to $O(m + n)$ even in worst case.

Lin-Canny Extensions

- **Penetration Depth**
  - **Original algorithm would not terminate if input polyhedra overlapped**
  - **Extension divides interior of polyhedra into pseudo-Voronoi regions.**
  - **Overlap can be detected and penetration depth approximated.**

Non-Convex Models

- Extensions of convex methods to non-convex models not obvious.
- Convex assumptions no longer valid:
  - Discontinuous change of closest features with transformations
  - When a local minimum is found, no longer able to disregard rest of the model.

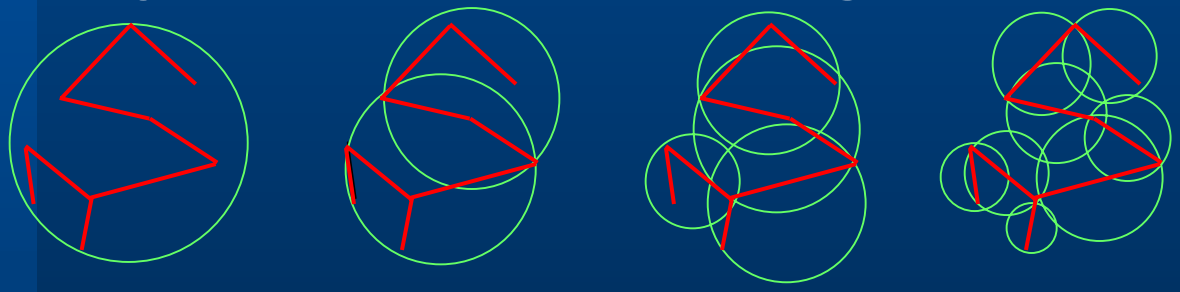


Non-Convex Models

- **Could decompose into convex pieces:**
  - **Distance: take minimum distance over all pairs of pieces**
  - **Penetration Depth: *not* maximum penetration depth, although can still estimate it.**
- **Problems:**
  - **If m & n are number of convex pieces in each model, O(m $\times$ n) pairs**
  - **Minimal decomposition is NP-hard, although approximations exist for closed solids.**
  - **No solution for polygon soups besides list of polygons**

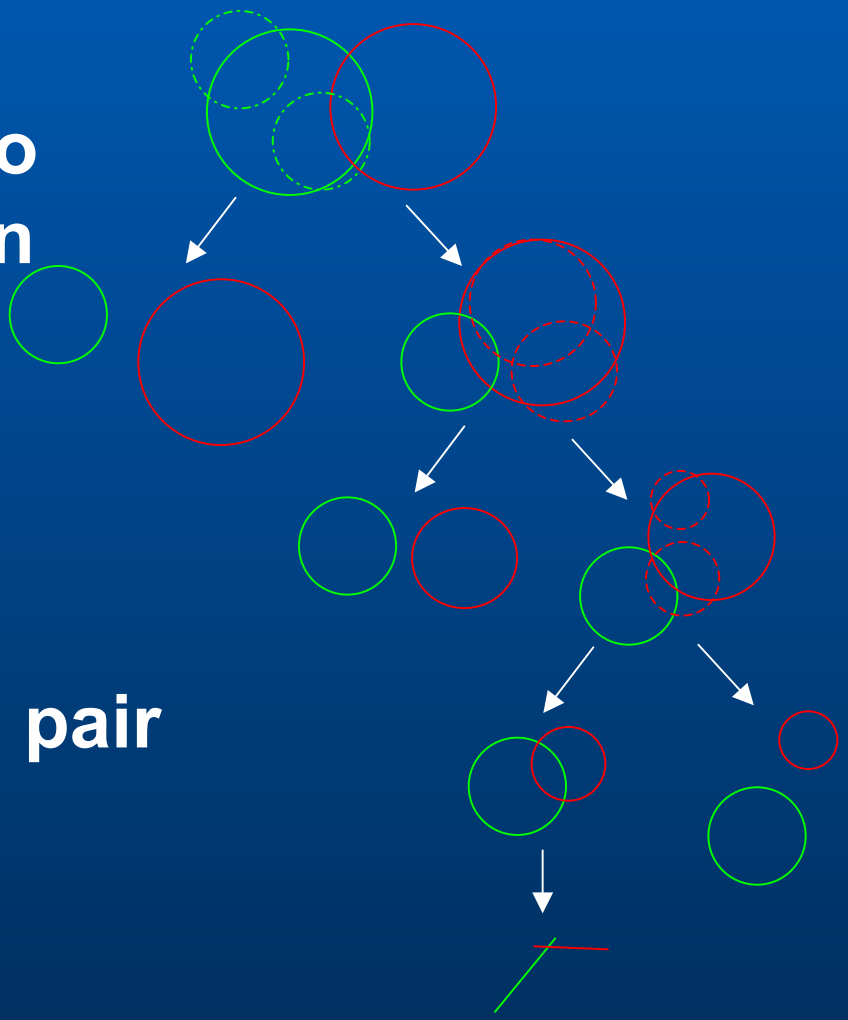
Bounding Volume Hierarchies

- BVHs can improve O(m×n) performance of distance & collision detection on m, n pieces
- Most commonly associated with polygon soups:
  - Each node has a shape that bounds a set of polygons.
  - Children contain volumes that each bound a different portion of the parent's polygons.
  - The leaves of the hierarchy usually contain individual polygons.
- A binary BVH for some line segments:



BVH Collision Detection

- Check root BVs first
- If BVs do not overlap, no contained polygons can overlap
- Else, subdivide one BV into its children, giving two new BV pairs
- Recursively check each pair



BVH Collision Detection

```
1.  Recursive-Collide(BV a, BV b)  {
2.      if (! bv-overlap(a,b)) return;
3.      if (leaf(a) and leaf(b)) {
4.          tri-overlap(tri(a), tri(b))
5.      }
6.      else if (!leaf(a)) {
7.          Recursive-Collide(lchild(a), b)
8.          Recursive-Collide(rchild(a), b)
9.      }
10.     else {
11.         Recursive-Collide(a, lchild(b))
12.         Recursive-Collide(a, rchild(b))
13.     }
14. }
```

BVH Distance Computation

- **Test at least one pair of triangles to get a *candidate* minimum distance.**
- **Terminate recursion when distance between bounding volumes greater than candidate minimum distance.**
- **BV distance generally more expensive than overlap - why?**

BVH Distance Computation

```
1.  Recursive-Distance(Real distance, BV a, BV b) {
2.      if (bv-distance(a,b) > distance) return;
3.      if (leaf(a) and leaf(b)) {
4.          distance = min(distance, tri-distance(tri(a), tri(b)))
5.      }
6.      else if (leaf(b) or (!leaf(a) and size(a) > size(b))) {
7.          Recursive-Distance(distance, lchild(a), b)
8.          Recursive-Distance(distance, rchild(a), b)
9.      }
10.     else {
11.         Recursive-Distance(distance, a, lchild(b))
12.         Recursive-Distance(distance, a, rchild(b))
13.     }
14. }
```

BVH Distance Computation

- Assume first candidate minimum distance found is the actual minimum distance.
- For each bv test, could use either:
  - bv-distance(a,b) > distance
  - !bv-overlap(a, b grown by distance)
- i.e., distance with BVH at its best is like collision detection with one BVH grown by minimum distance
- Explains why performance is worse.

Distance Demo

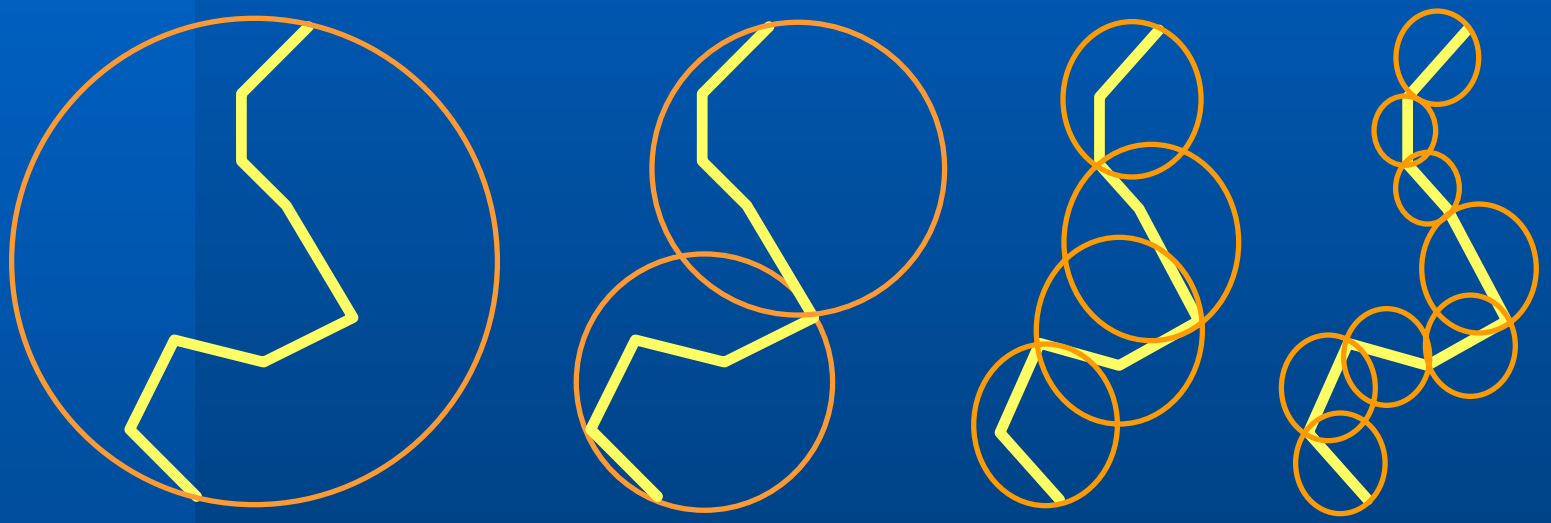
BV Types

- Spheres [Hubbard93, Quinlan94]
- AABBs [PML95, SOLID97]: axis-aligned bounding boxes
- OBBs [GLM96, BCGMT96] - oriented bounding boxes
- K-DOPs [KHM+96] - polytopes with k discrete face orientations
- Convex Hulls [LC92, Lin93]
- Spherical Shells [KPLM98] - portions of space between concentric spheres
- Swept-spheres - sphere extended primitives

Evaluating BV Types

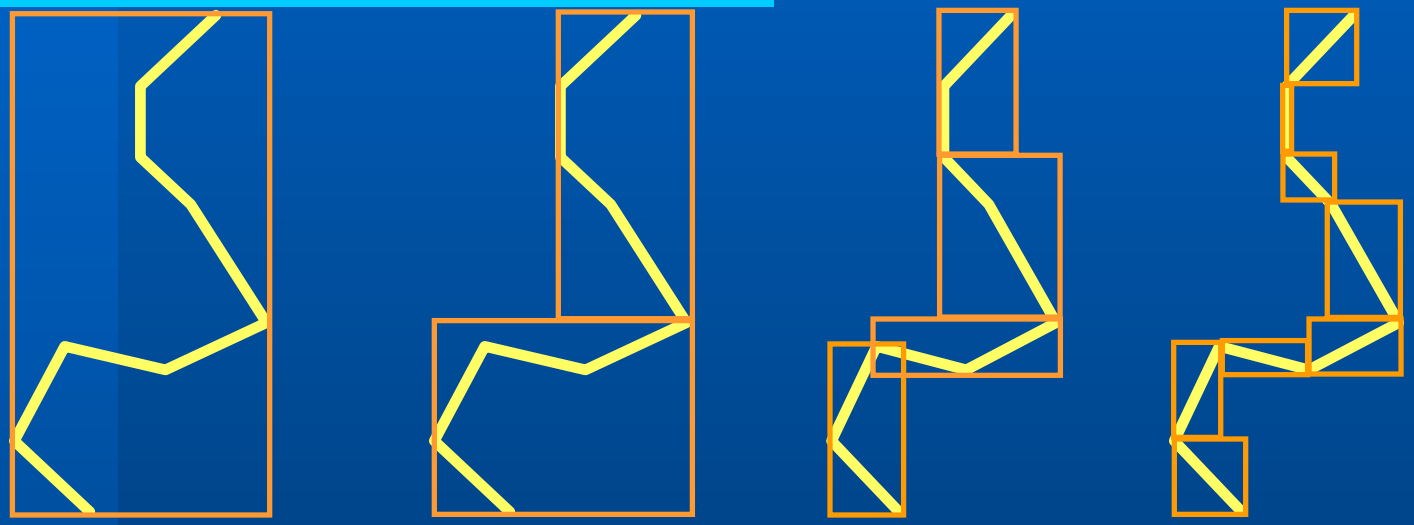
- **Common BV choice trade-off:**
  - **tightness of fit; power to “prune” search**
  - **speed of BV overlap/distance tests - includes speed of transforming BV as BVH rotates**

Spheres



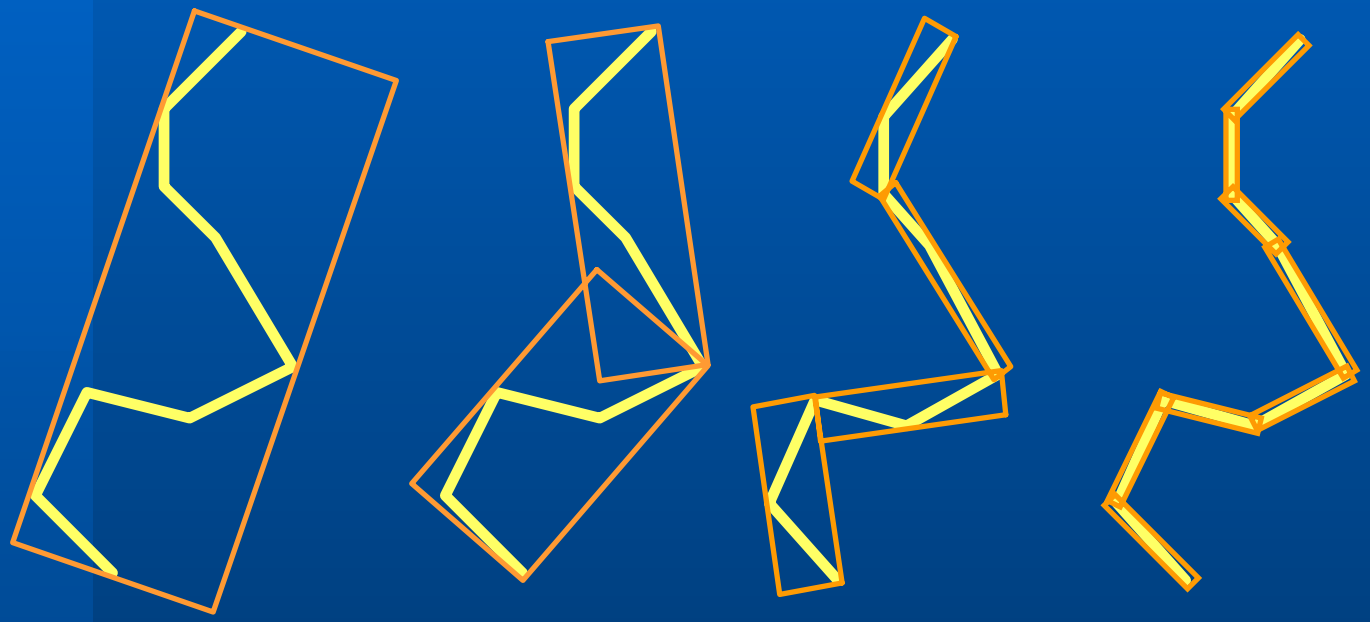
- Very fast overlap/distance tests
- Poor fit for elongated or flat geometry

AABBs



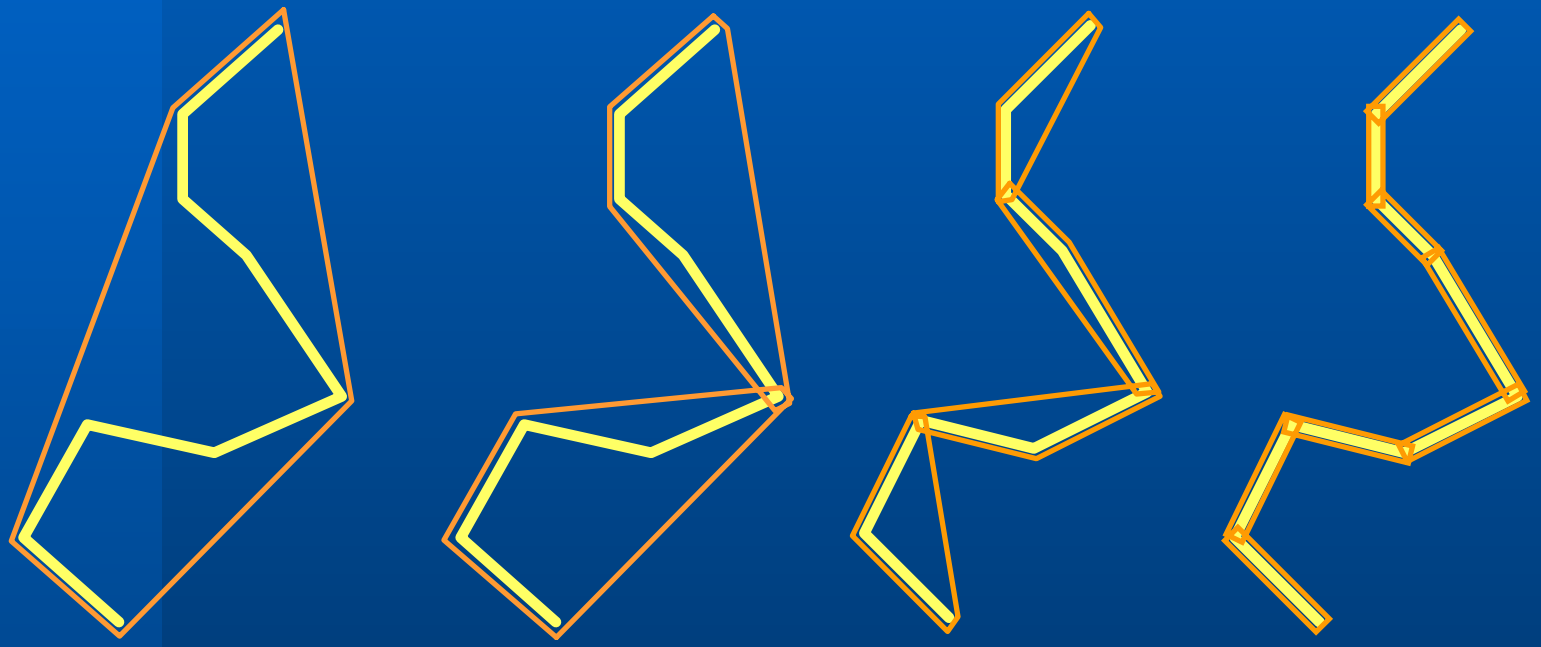
- Very fast overlap/distance test without rotation.
- Tightness depends on orientation of geometry
- With rotation, need OBB tests, or to realign boxes with axes (increases test cost)

OBBs



- Overlap test more expensive, but well optimized / no optimized distance test yet.
- Tighter fit, even for flat, long geometry.

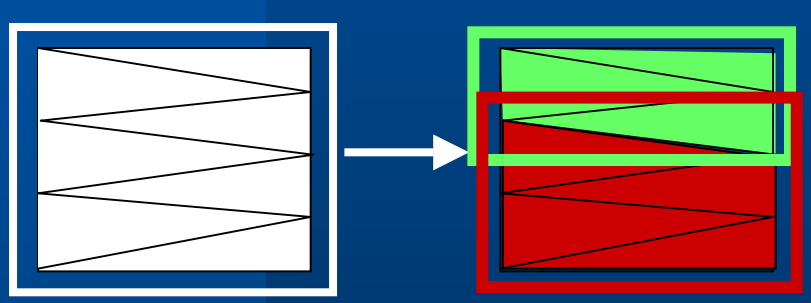
Convex Hull Tree



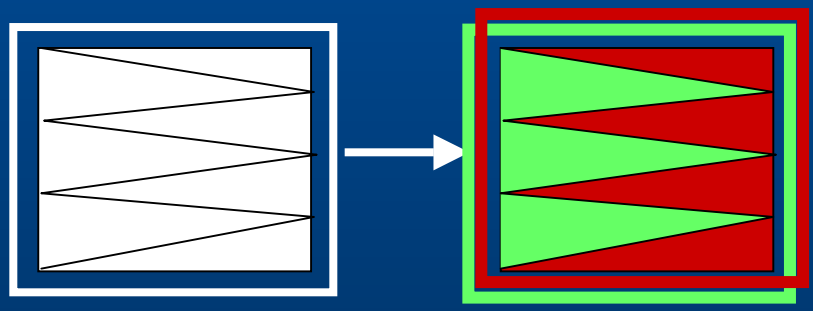
- Overlap test very expensive - even with coherence.
- Tightest fit among convex BVs

Evaluating BVHs

- How BVs are placed around geometry is as important as which type used:
  - BVHs should partition space
  - When children mostly overlap each other and parent, split is wasted.



Good Split

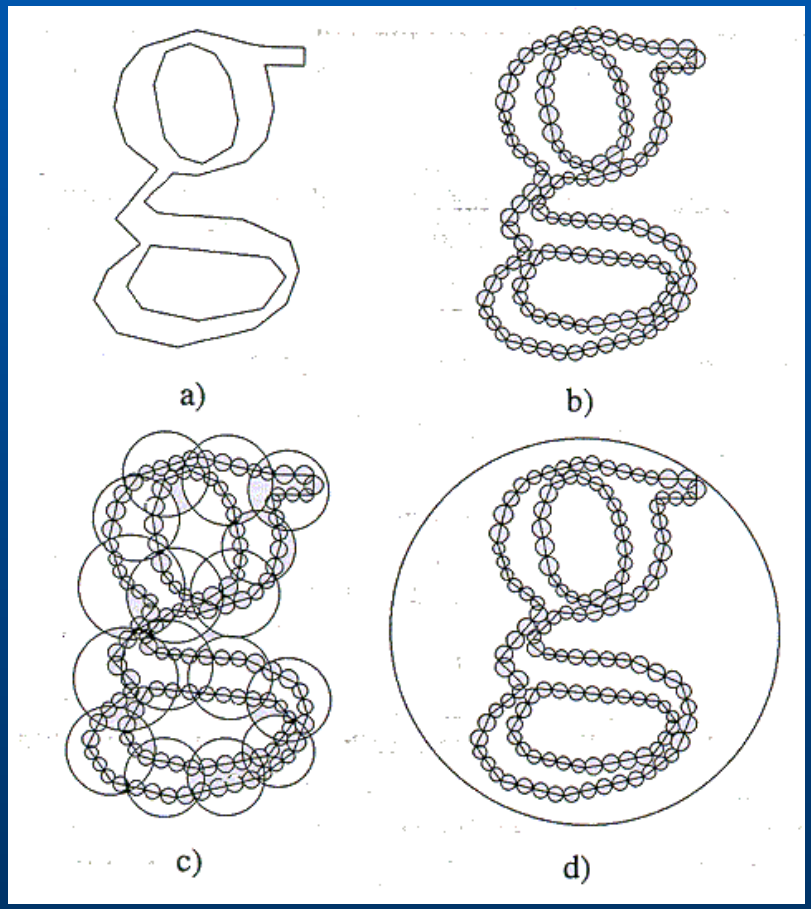


Bad Split

Sphere Trees [Quinlan94]

- **Hierarchy building:**
  - **First tiles surface of triangles with many small spheres, so that many leaf nodes may have a pointer to the same triangle.**
  - **Builds a hierarchy top down that bounds these spheres, instead of triangles.**

Sphere Trees [Quinlan94]



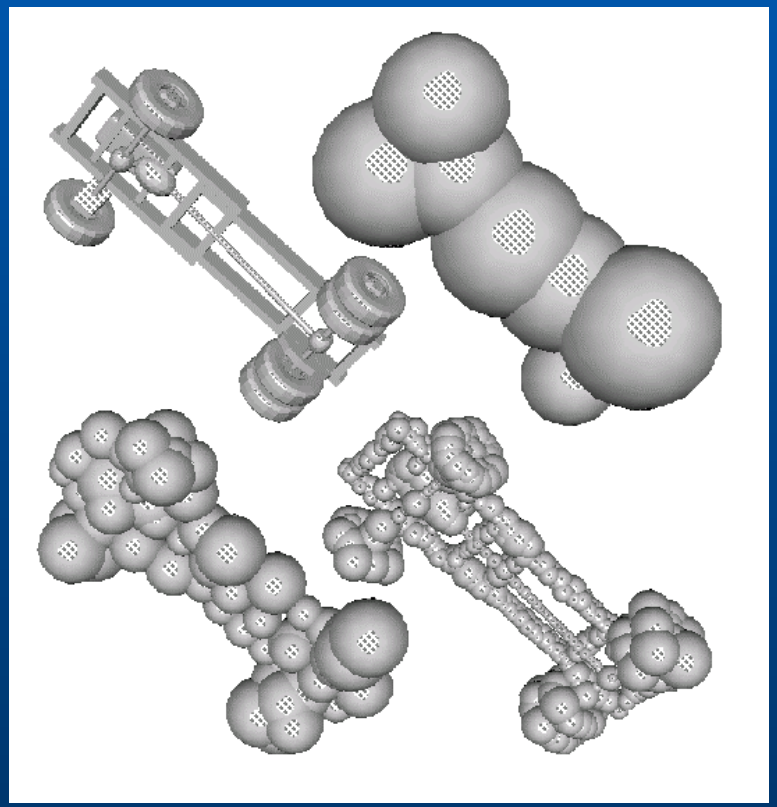
Sphere Trees [Quinlan94]

- **Distance Computation:**
  - **Same method as previously outlined, except that many leaf node pairs may correspond to same triangle.**
  - **Triangle pair distances are hashed to avoid redundant computations.**

Sphere Trees[Hubbard96]

- At lowest level, approximates model with spheres:
  - Distributes points evenly over surface of model
  - Builds 3D Voronoi diagram, capturing skeletal shape.
  - Each leaf sphere bounds four Voronoi vertices
- Hierarchy built by merging pairs of spheres
  - Merges prioritized by tightness of resulting sphere
  - Does merges to make 8 children for root, recurs on each of the 8 children

Sphere Trees [Hubbard96]



Sphere Trees [Hubbard96]

- **Time-critical collision detection**
  - **Do subdivision until available time runs out.**
  - **Base collision-response on overlapping spheres at whatever level reached.**
- **Exact collision detection**
  - **Keep pointer to covered polygon in each leaf sphere; compare polygons when two leaf spheres overlap - similar to Quinlan's approach**

OBB Trees [GLM96] (RAPID)

- **Hierarchy built downward - “split and fit”**
  - **Root box fits all triangles**
  - **Triangles are split into two subsets**
  - **Boxes recursively fit to subsets**

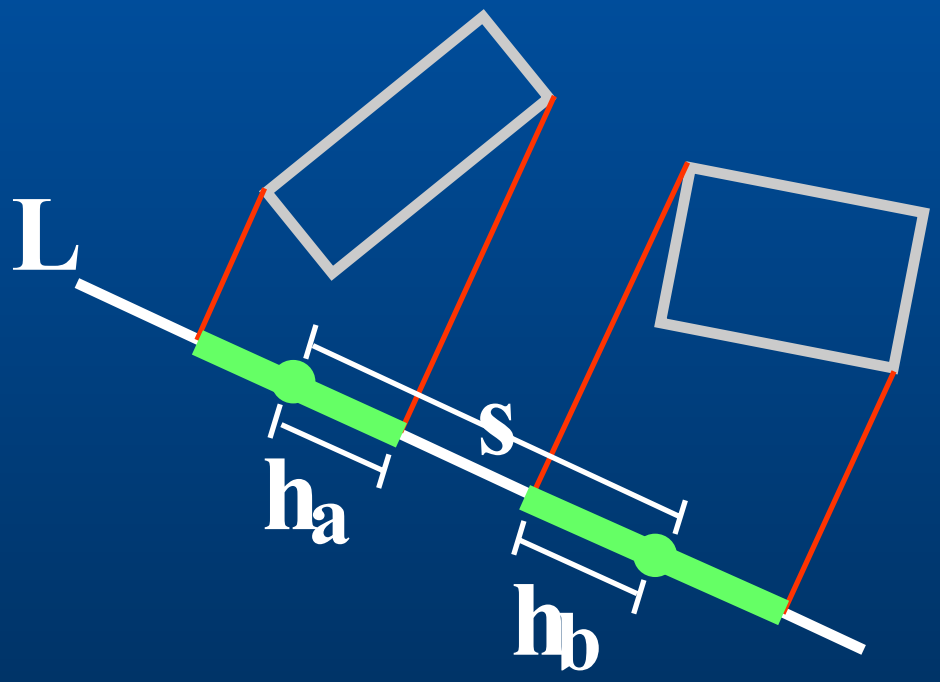
- **Fitting method:**
  - **Box orientation obtained from eigenvectors of the covariance matrix of the vertices.**
  - **Axes can align with a row of vertices, reducing tightness of fit.**
  - **Sampled convex hull of vertices works better**

OBB Trees [GLM96] (RAPID)

- **OBB overlap test based on separating axis theorem:**
  - **Two convex polyhedra are disjoint iff their projections on one of the following axes are disjoint:**
    - **the face normals of the polyhedra**
    - **all cross products of two edge directions, one from each polyhedron.**
  - **With OBB's there are only 15 such axes.**
  - **Box orthogonality yields optimizations: most dot products needed are encoded in relative orientation between the boxes.**

OBB Trees [GLM96] (RAPID)

- **Check whether axis L separates boxes:**



OBB Trees [GLM96] (RAPID)

- **Which factor dominates - fit or cost of tests?**
  - **Gottschalk formalized this somewhat: showed OBBs converge to finely tesselated geometry asymptotically faster than spheres or AABBs.**
  - **For two close parallel surfaces, OBBs can be a big win.**
  - **In many experiments, OBBs have justified higher test cost**

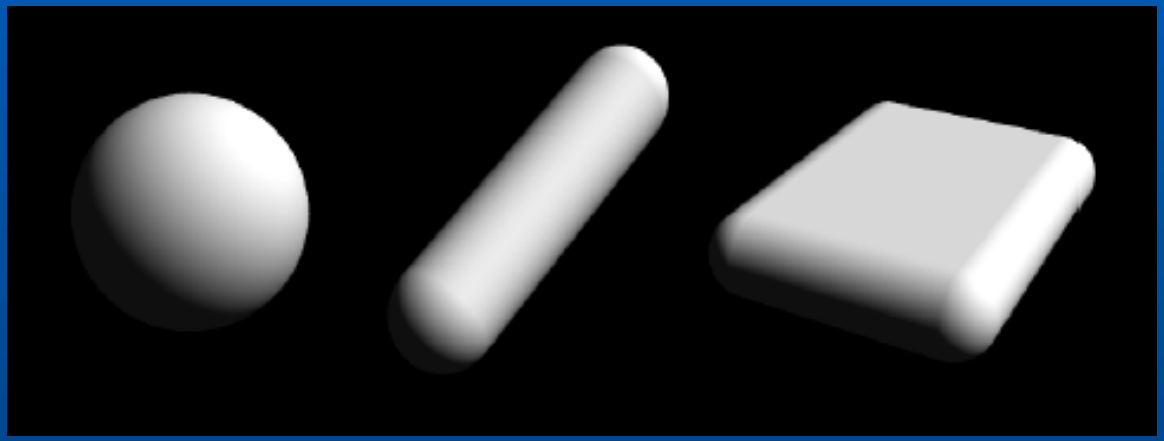
AABB Trees [SOLID97]

- **Builds AABB tree to model, but reorients AABBs with model.**
  - **OBB tests needed, but only one relative orientation between pair of OBBs when comparing two hierarchies.**
  - **Reduces OBB test cost, but loses convergence.**
  - **Efficient extension for deformable models.**

Swept Sphere Trees [LGLM98] (PQP)

- OBBs have good convergence properties, but what about distance computation?
- Premise:
  - Spheres are cheap - you get a lot of mileage out of a point and a uniform offset
  - Could replace point with something more complicated, like a line or rectangle. Equivalent to sweeping sphere along that shape.
  - Distance between swept spheres is distance between the core shapes minus the sphere radii.

Swept Sphere Trees [LGLM98] (PQP)



- **Why points, lines, and rectangles?**
  - **Line-swept spheres fit elongated shapes**
  - **Rectangle-swept spheres fit flat shapes / orthogonal**
  - **Give a variation of fit quality, storage needs, and distance test cost.**
  - **Once rectangle-pair distance test available, all other pairwise distance tests are easy - good candidate for hybrid hierarchies**

Swept Sphere Trees [LGLM98] (PQP)

- **Rectangle Distance Test:**
  - **If closest points are on edges**
    - **Lin-Canny style approach to find edge-pair with closest points**
    - **Simple Voronoi regions (half-spaces) make this efficient**

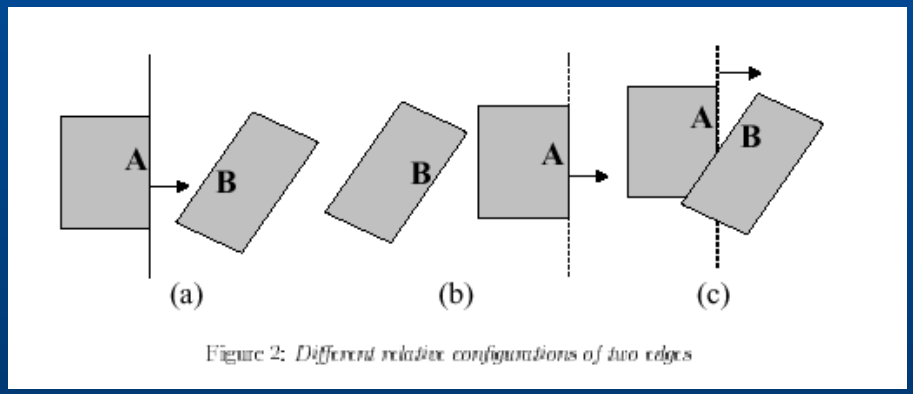


Figure 2: *Different relative configurations of two edges*

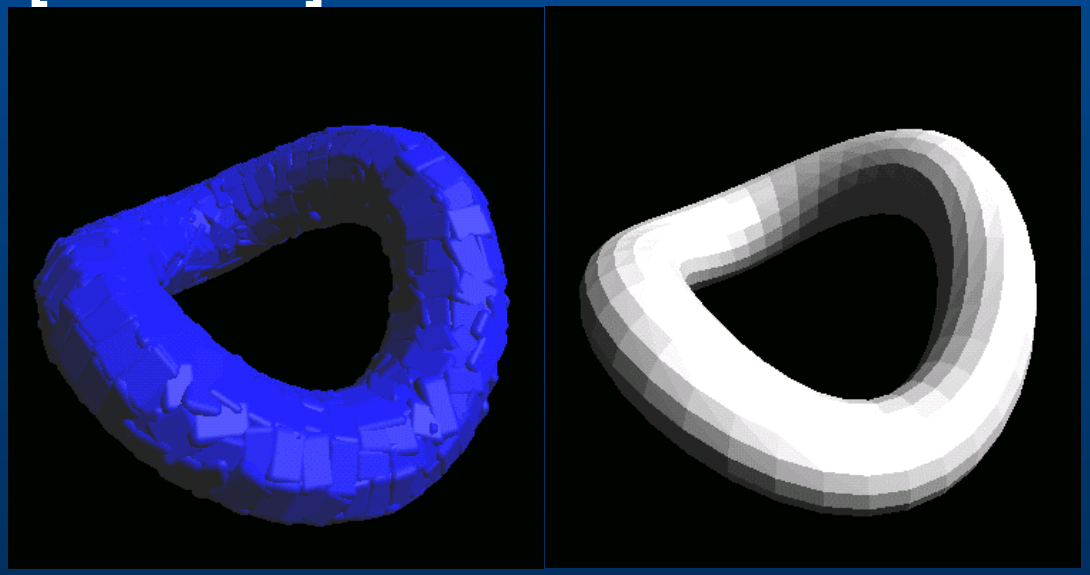
  - **Otherwise:**
    - **Takes projections of rectangles on face normals**
    - **Max separation of these projections is distance**

Swept Sphere Trees [LGLM98] (PQP)

- **Performance:**
  - **To date, using all types *not* faster than using rectangles only.**
  - **In several experiments, faster than Quinlan's distance library, after standardizing triangle-pair test; using small spheres w/ Quinlan's library makes performance similar, but requires large amounts of memory.**
  - **Besides BVs, coherence trick aided our system:**
    - **Cache the closest triangle pair, and use to initialize the distance estimate in next query: yields small factor of speedup**
  - **Slower than RAPID for checking overlap**

Swept Sphere Trees [LGLM98] (PQP)

- **Future possibilities with PQP:**
  - **Incorporate points and lines for higher performance and/or lower memory usage.**
  - **Use BVs to approximate object shape [Hubbard96]**
  - **Providing multiple contact points for constraint forces [Baraff89]**



Conclusions

- **Many different methods in existence for engineering a collision system.**
- **Unfortunately, numerous sacrifices and trade-offs:**
  - **object complexity v. algorithm speed / available algorithms**
  - **realistic simulation v. real-time performance**

Conclusions

- **Big strides could still be made with nonconvex models:**
  - **BVHs not quite satisfying: Convex model algorithms nicely exploit coherence, estimate penetration depth**
  - **On the other hand, BVHs work with very general input.**
  - **Playstation2-optimized version of PQP in the works...**

The End.

References

- [BCG+96] Barequet, Chazelle, Guibas, Mitchell, Tal - Boxtree: A hierarchical representation of surfaces in 3d - 1996
- [Baraff89] David Baraff - Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies - 1989
- [Cameron97] Stephen Cameron - Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra - 1997
- [CLMP95] Cohen, Lin, Manocha, Ponamgi - I-COLLIDE: An interactive and exact collision detection system for large-scale environments - 1995
- [GJK88] Gilbert, Johnson, Keerthi - A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space - 1988
- [GLM96] S. Gottschalk, M. Lin, D. Manocha - Obb-tree: A hierarchical structure for rapid interference detection - http://www.cs.unc.edu/~geom/OBB/OBBT.html - 1996

References

- [Hubbard93] Philip Hubbard - Approximating Polyhedra with Spheres for Time-Critical Collision Detection - 1996
- [Hubbard96] Philip Hubbard - Interactive Collision Detection - 1993
- [KHM+98] Klosowski, Held, Mitchell, Sowizral, Zikan - Efficient collision detection using bounding volume hierarchies of k-dops - 1998
- [KPLM98] Krishnan, Pattekar, Lin, Manocha - Spherical shell: A higher order bounding volume for fast proximity queries - 1998
- [LGLM98] Larsen, Gottschalk, Lin, Manocha - Fast Proximity Queries with Swept Sphere Volumes (tech. report) - http://www.cs.unc.edu/~geom/SSV/ - 1998
- [LC91] Lin, Canny - A fast algorithm for incremental distance calculation - 1991
- [Lin93] Ming Lin - Efficient Collision Detection for Animation and Robotics - 1993

References

- [Mirtich96] - Impulse-based Dynamic Simulation of Rigid Body Systems - 1996
- Moore, M. and Wilhelms, J. - Collision detection and response for computer animation - 1988
- [PLM95] Ponamgi, Manocha, Lin - Incremental algorithms for collision detection between solid models.
- [Quinlan94] Sean Quinlan - Efficient Distance Computation between Non-Convex Objects - 1994
- [SOLID97] SOLID Interference Detection System http://www.win.tue.nl/cs/tt/gino/solid - 1997