Algorithms and Data Structures for Moving Objects

Pankaj K. Agarwal

Center for Geometric Computing

Department of Computer Science
Duke University

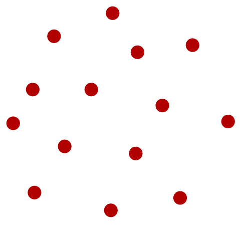
Center for Geometric Computing

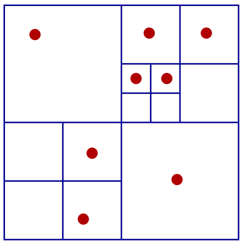
Geometric Data Structures

S: Set of geometric objects
Points, segments, polygons

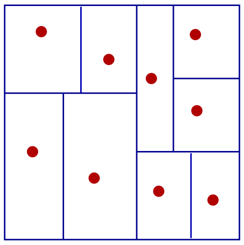
☆ Ask several queries on S

- Range searching
- Nearest-neighbor searching

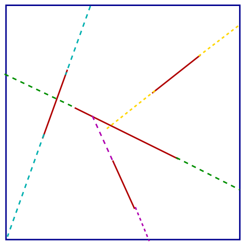




Quad Tree



kd–Tree



BSP

Center for Geometric Computing

Dynamic Data Structures

- ☆ Allow insertion/deletion of objects at discrete time instances
- ☆ Several dynamization techniques
  (Bentley-Saxe; Overmars)
- ☆ Amortized vs worst-case update time (Overmars)
- ☆ Desirable to update the data structure in $O(\log n)$ time

Moving Objects: Applications

- ☆ Traffic management
  - Location based services
  - Emergancy services
  - Air traffic control
- ☆ Digital battlefields
- ☆ Animation
- ☆ Physical simulation
- ☆ Adhoc networks

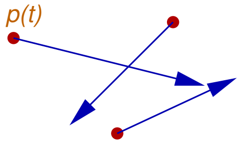
Need data structures for storing, analyzing, querying moving objects.

Dynamic data structures not suitable for handling moving objects!

Modeling Motion

$p(t) = (x(t), y(t))$: Position of p at time t.

- $x(\cdot), y(\cdot)$: Polynomials
- *Degree* of motion: max degree of $x(\cdot), y(\cdot)$.
- *Linear motion:* Degree of motion is 1
  $p(t) = \mathbf{a}t + \mathbf{b}, \quad \mathbf{a}, \mathbf{b} \in \mathbb{R}^2$



☆ Mostly assume motion to be linear

☆ Trajectory of points can change

☆ Trajectory can be piecewise linear

Issues:

☆ Sampled motion

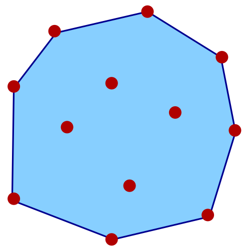
☆ Hierarchical motion

☆ Uncertainty

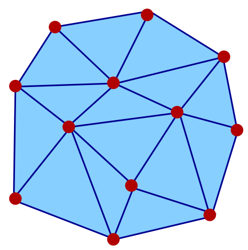
Early Work

☆ Off-line setting: Linear motion, known in advance

☆ Bound the number of *combinatorial* changes in various geometric structures as points move [Atallah, 1985]
E.g. Convex hull, closest pair, Delaunay triangulation



CH(S)



DT(S)

☆ Number of changes in $CH(S)$: $\Theta(n^2)$

☆ Number of changes in $DT(S)$: $O(n^3), \Omega(n^2)$
(A.-Sharir, 2000)

Early Work

Open Problems:

- ☆ How many times does the Delaunay triangulation of a point set change?
- ☆ How many times does the smallest disk containing a point set change?

Early Work

Similar problems arise in combinatrial optimization

- ☆ *Senstivity analysis* of optimization problems:
- ☆ *Kinetic minimum spanning tree*:
  - $G = (V, E)$, $w(e) = a \cdot t + b$
  - How many combinatorially different MST's are there?
  - # different MSTs = $O(mn^{1/3})$ (Gusfield, 1979; Dey, 1998)
- ☆ Similar questions for max flow, shortest paths, ...

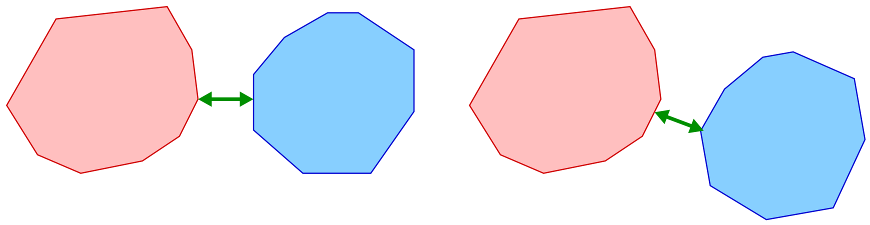
Time Sampling Approach

- ☆ Choose fixed time step
- ☆ Update the positions of moving objects at each time stamp
- ☆ Update the data structure with the new positions of objects
- ☆ Use temporal coherence to expedite the updates

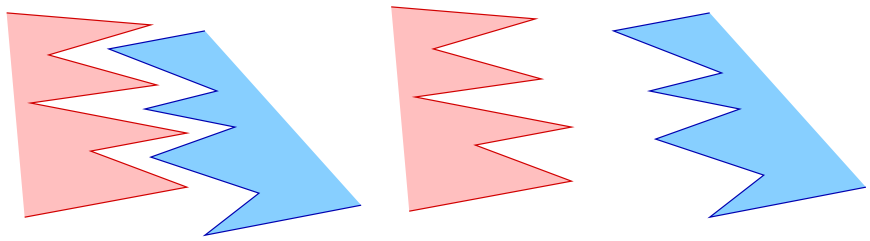
Works well when

- ☆ Few objects are moving
- ☆ All of them are moving with roughly the same speed

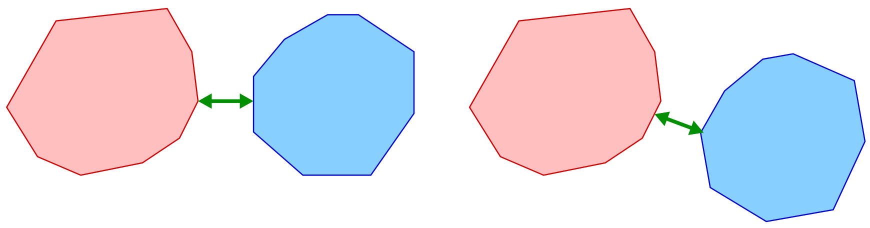
Collision Detection



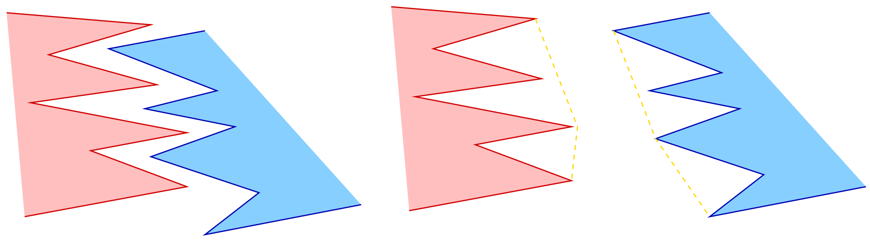
Closest pair might not be unique!



Collision Detection

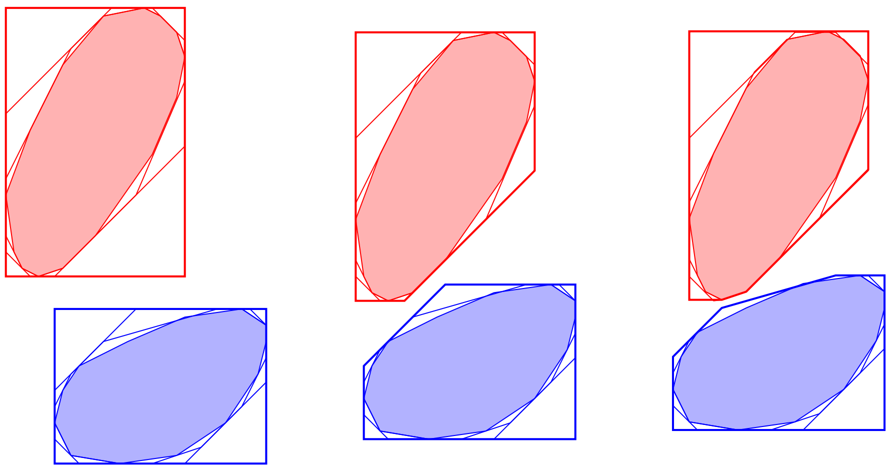


Closest pair might not be unique!



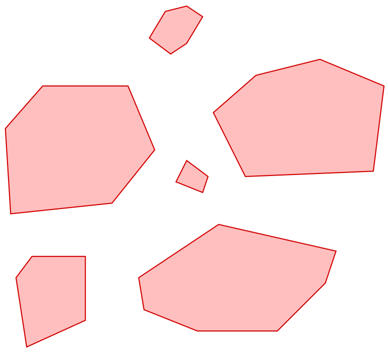
Collision Detection

- ☆ Maintain a hierarchical representation of each object
- ☆ Level of approximation changes adaptively



Collision Detection

Expensive if many objects are moving with different speeds.



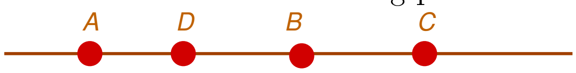
- ☆ How do we choose the time step?
- ☆ Time step is governed by the fastest moving objects
- ☆ Unnecessary work done on slow moving objects

Kinetic Data Structures

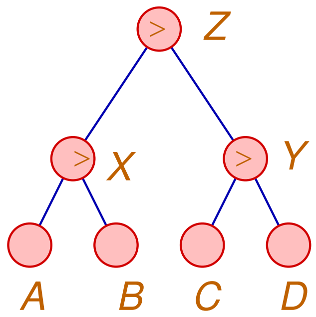
(Basch, Guibas, Hershberger, 1998)

- ☆ Event based structure
- ☆ Exploit combinatorial structure of the algorithm
  - Combinatorial structure changes at discrete times
- ☆ Store some additional auxiliary information to expedite the simulation

Maintain the maximum of a set of moving points on a line



Kinetic Data Structures



Computing max

- ☆ $A > B, C > D, X > Y$ hold: no recomputation is necessary (*certificates*)
- ☆ $A = B, C = D, X = Y$: Update the structure (*events*)
- ☆ Spend $O(\log n)$ time at each event
- ☆ Process events in the sorted order

Kinetic Data Structures

- ☆ Maintain a set of *certificates*
- ☆ Certificates provide a proof of the correctness of the structure
- ☆ Determine when a certificate fails; an *event*
  - Event times are roots of certain polynomials
- ☆ Update the structure and compute new certificates
- ☆ A global priority queue stores the events

Kinetic Data Structures

Performance of a KDS depends on

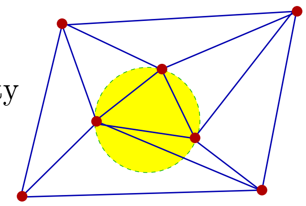
- ☆ Number of events (*efficiency*)
- ☆ Number of certificates (*Compactness*)
- ☆ Time spent at each event (*Locality*)

Efficient KDS developed for

- ☆ 2D convex hull
- ☆ Closest pair
- ☆ Delaunay triangulation
- ☆ Diameter, width
- ☆ Collision detection between two polygons

KDS for Delaunay Triangulation

- ☆ ab is an edge in DT $\Leftrightarrow \exists$ an empty disk containing only a and b
- ☆ abc is a triangle in DT $\Leftrightarrow$ circle $\bigcirc abc$ is empty
- ☆ *Events:* a, b, c, d become cocircular.
- ☆ *Certificates:* $d \notin \bigcirc abc$, $\triangle abd \in DT$.



events = $O(n^3)$, update time = $O(\log n)$

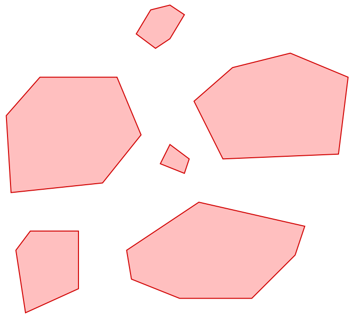
Challenging Issues

- ☆ Canonical vs noncanonical structures
- ☆ Exact vs approximate structures
- ☆ Motion and randomization
- ☆ Querying motion and prediction

Challenging Issue: I

Noncanonical Structures

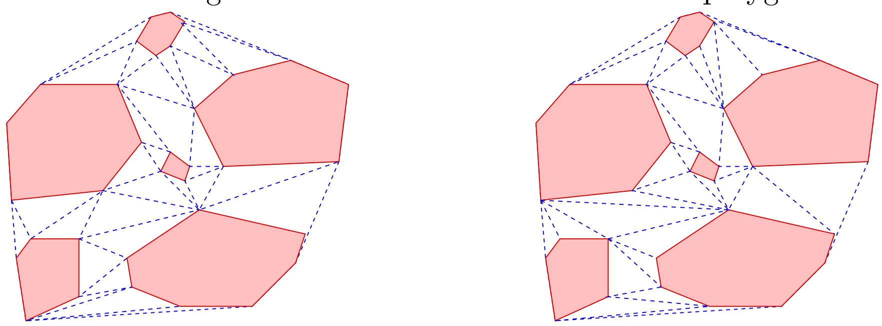
Collision Detection



- ☆ Each polygon moving with a different velocity
- ☆ Previous approach would require pairwise interaction
- ☆ *Need to maintain a global structure!*

Collision Detection

Maintain a triangulation of the common exterior of polygons



Collision Detection

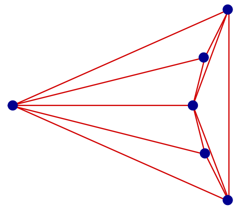
- ☆ Deform triangulation as the polygons move
- ☆ Collision → Self-intersection
- ☆ Triangulation is not a canonical structure:
  Which triangulation should be maintained!

Triangulations

☆ Maintain a triangulation of a point set

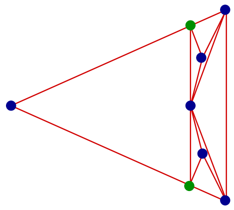
☆ Existing algorithms canonicalize the triangulation by introducing additional properties

- Delaunay triangulation
- Sweep triangulation



☆ Best known upper bound on # changes: $n^{7/3}$
[A., Basch, Guibas, Hershberger, Zhang 2000]

☆ Allowing Steiner points



Lower Bound: $\exists$ moving points for which any linear-size triangulation changes $\Omega(n^2)$ times
[A., Basch, de Berg, Guibas, Hershberger 2000]

Triangulations: Open Problems

☆ Design a data structure that processes $O(n^2)$ events (edge flips).

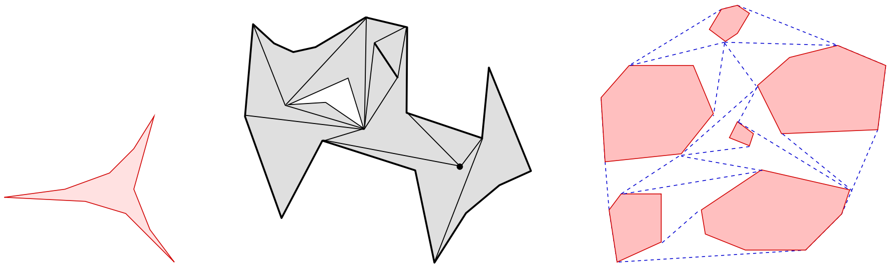
☆ Maintain a Triangulation in 3D

☆ Prove $\Omega(n^2)$ lower bound without any assumption on the number of Steiner points.

Pseudo-Triangulations

Collision Detection: Too many events in a triangulation
Maintain a coarser structure!

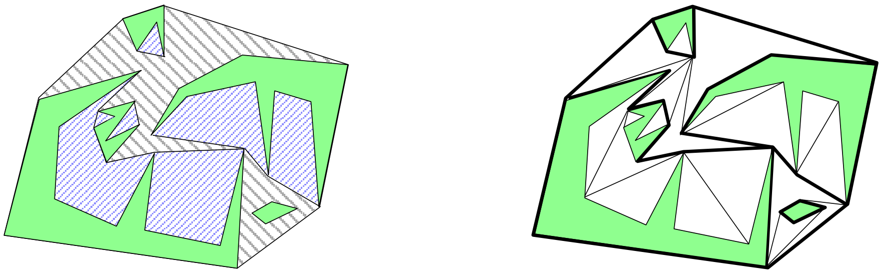
Pseudo-triangle: Region bounded by three concave chains



- ☆ Maintain a pseudo-triangulation of the common exterior
  [A., Basch, Guibas, Hershberger, and Zhang],
  [Kirkaptrick, Snoeyink, Speckman]
- ☆ Number of pseudo triangles: ≈ # polygons
- ☆ Updating pseudo-triangulation is also simple

Pseudo-Triangulations

- ☆ If polygons are nonconvex, maintaining a pseudo-triangulation is more difficult
- ☆ Different approaches proposed
  [A., Basch, Guibas, Hershberger, and Zhang],
  [Kirkaptrick, Snoeyink, Speckman]



Challenging Issue: II

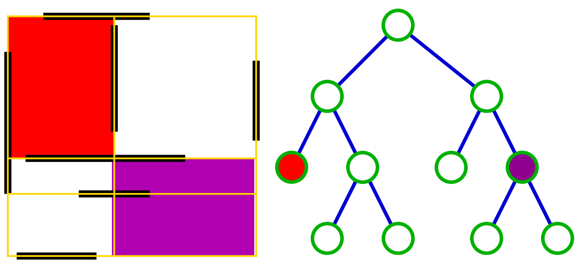
Kinetic Algorithms and Randomization

Binary Space Partitions (in $\mathbb{R}^3$)

☆ Each node v is associated with a convex polytope τ_v.

☆ If v is a leaf, τ_v is empty; otherwise, τ_v is split by a plane h_v

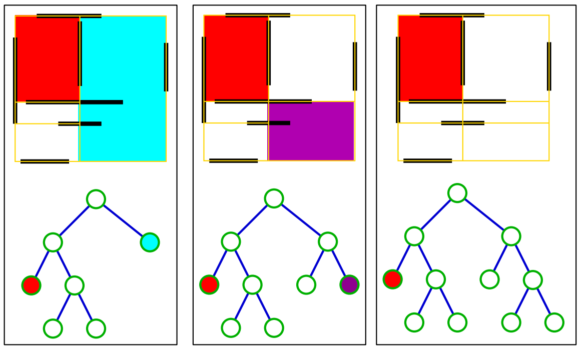
☆ h_v and triangles lying in h_v stored at v.



☆ Leaves induce a partition of the space

☆ BSP induces a hierarchical partition of $\mathbb{R}^3$.

Binary Space Partitions (in $\mathbb{R}^3$)



Incremental construction of a BSP

- ☆ BSP is a noncanonical structure
- ☆ Size depends on the insertion order
- ☆ Best-known size in 2D: $O(n \log n)$, $\Omega((n \log n)/ \log \log n)$
- ☆ Best-known size in 3D: $\Theta(n^2)$

Kinetic BSP

$S(t)$: Objects at time t
(segments in 2D, triangles in 3D).

$\mathcal{A}$: An algorithm for constructing a BSP

$B(t)$: BSP for $S(t)$ constructed by $\mathcal{A}$.

Maintain $B(t)$ efficiently.

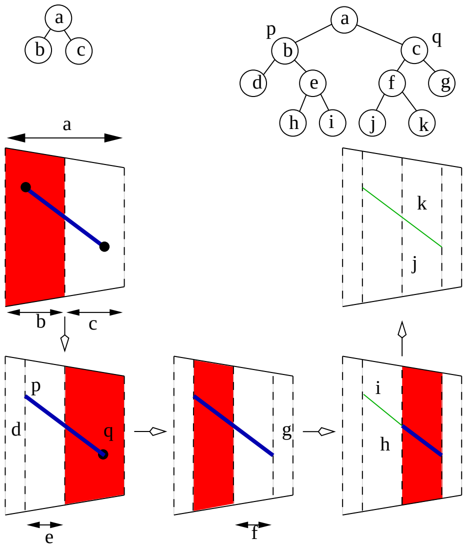
Commonly used approach:

Discretize time into short intervals and modify the BSP at the end of each interval — *insert/delete objects that have moved*

How does one choose the time interval?

BSP for Segments

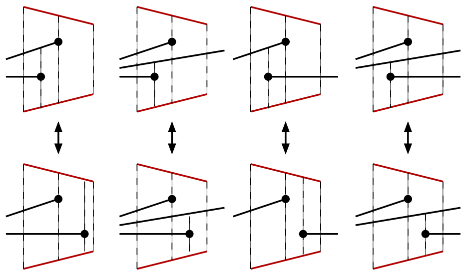
Randomized incremental algorithm: Add a random segment



BSP: KDS Framework

- Maintain the BSP of $S(t)$ constructed by the randomized incremental algorithm
  - Permutation remains fixed
- Maintain a set of *certificates* to keep track of when the structure of BSP changes.
- *Critical events:* One of the certificates becomes invalid.
- Update the BSP at critical events.
- *Assume coin-flips are private.*
- At each event:
  Expected # certificate changes = $O(1)$
  Expected $O(\log n)$ time to update the BSP at a critical event.

Critical Events for Segments



Kinetic BSP

k: # moving objects, each moving along an algebraic trajectory

- ☆ Disjoint segments
  Expected update time: $O(\log n)$
  Expected # events: $O(nk)$
- ☆ Set of intersecting segments
  Expected update time: $O(\log n)$
  Expected # events: $O(nk\lambda_s(n))$
- ☆ Set of disjoint triangles in 3D
  Expected update time: $O(\log^2 n)$
  Expected # events: $O(nk\lambda_s(n) \log n)$

Update algorithm does not require trajectories to be algebraic.

Open Problems

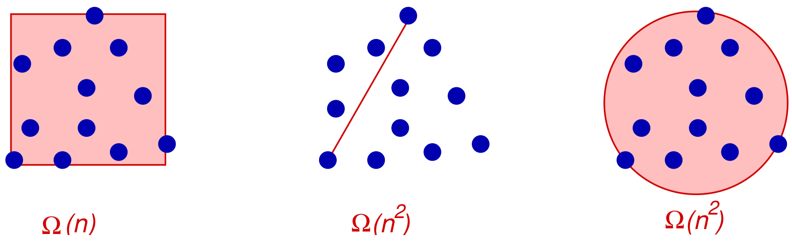
- ☆ Cannot assume coin flips to remain private
- ☆ Can we change the random permutation preiodically to fool an adversary
- ☆ Prove lower bounds on the number of events for maintaining a BSP

Challenging Issue: III

Exact vs Approximate Structures

Maintaining Approximate Structures

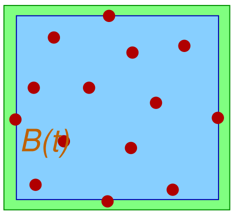
- ☆ Maintaining exact structure is expensive:
  - Minimum bounding rectangle (MBR) changes $\Omega(n)$ times
  - Diameter, smallest enclosing disk change $\Omega(n^2)$ times



- ☆ Maintain the structure approximately
  - Maintain an enclosing rectangle that is not much bigger than the MBR
  - Maintain a pair of points that approximates diameter

Smallest Enclosing Box

- ☆ $B(t)$: Smallest box enclosing $S(t)$
- ☆ $B^\varepsilon(t)$: Outer ε-approx of $B(t)$
  $$B(t) \subseteq B^\varepsilon(t) \subseteq (1+\varepsilon)B(t)$$

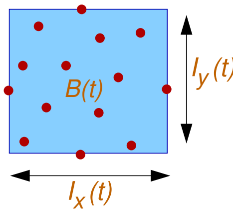


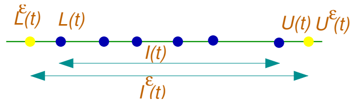
Can one maintain $B^\varepsilon(t)$ so that its combinatorial structure changes only a few times?

(A., Har-Peled, 2001)

- ☆ # changes $O(1/\sqrt{\varepsilon})$, total time $O(n\log(1/\varepsilon))$
- ☆ Supports insertions, deletions, change in trajectory in $O(\log^2 n)$ time

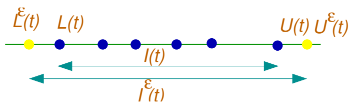
Smallest Enclosing Box

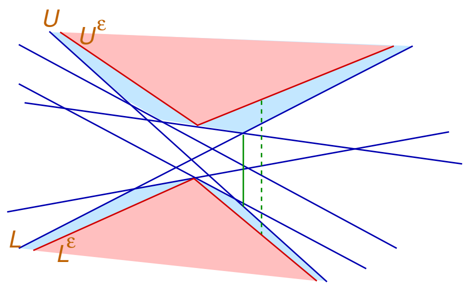




- ☆ Suffices to compute the *extent* in each dimension separately
- ☆ $p_i(t) = a_i t + b_i$
- ☆ $L(t) = \min_i a_i t + b_i$, $U(t) = \max_i a_i t + b_i$
- ☆ $I(t) = U(t) - L(t)$
- ☆ $L(t) - (\varepsilon/2)I(t) \le L^\varepsilon(t) \le L(t)$,
  $U(t) \le U^\varepsilon(t) \le U(t) + (\varepsilon/2)I(t)$,
- ☆ $I^\varepsilon(t) = U^\varepsilon(t) - L^\varepsilon(t)$

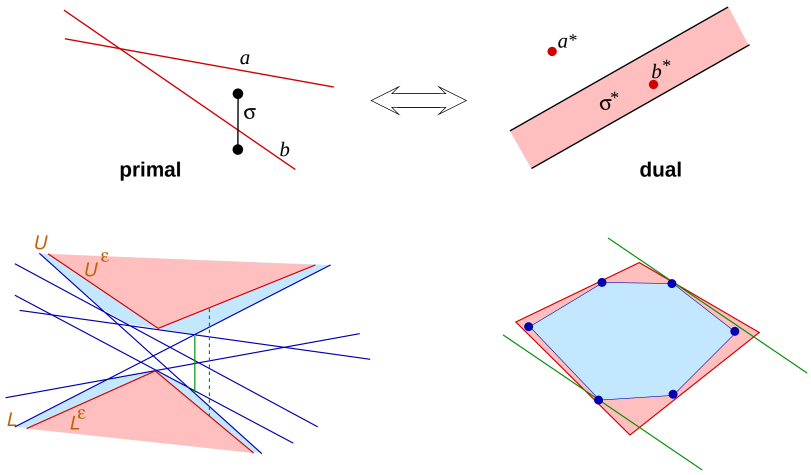
Geometric Interpretation





- ☆ $\ell_i = \bigcup_t (t, p_i(t))$: Trajectory of p_i
- ☆ L: Lower envelope of trajectories; U: Upper envelope
- ☆ L^ε: ε-approximation of the lower envelope
- ☆ U^ε: ε-approximation of the upper envelope

Geometric Interpretation: Duality



Approximate envelope $\Longrightarrow$ Approximate convex hull of points

Summary of Results

An ε-approximation of convex hull with $O(1/\sqrt{\varepsilon})$ vertices can be computed in $O(n\log(1/\varepsilon))$ time
One can even compute it in an on-line manner

☆ ε-approximation of smallest enclosing box

- $O(1/\sqrt{\varepsilon})$ changes
- Update time: $O(\log n)$

☆ Diameter:

- # changes: $O(1/\varepsilon)$
- Update time: $O(1/\varepsilon)$

☆ Smallest enclosing disk:

- # changes: $O(1/\varepsilon^{3/2})$
- Update time: $O((1/\varepsilon)\log n)$

Open Problems

☆ Maintain a "good" clustering of a set of moving points

☆ Maintain approximate k-center

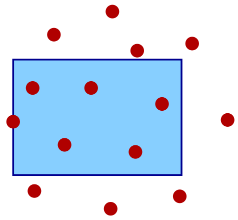
Challenging Issue: IV

Querying Moving Objects

Range Searching

S: Set of points

Preprocess S into a data structure
Report all points of S lying inside a query rectangle



Data Structure	Space	Query
Range tree	$n\dfrac{\log n}{\log\log n}$	$O(\log n + k)$
External Range tree	$\dfrac{n}{B}\dfrac{\log_B n}{\log_B \log n}$	$O(\log_B n + k/B)$
kdB-tree	$O(n/B)$	$O(\sqrt{n/B} + k/B)$

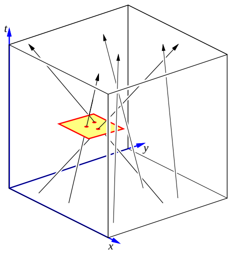
Kinetic Range Searching

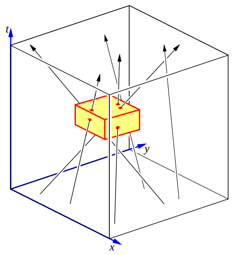
S: Set of points, each moving with fixed velocity in the plane

Preprocess S into a data structure:

Q1 Given a rectangle R and a time value t,
report all points of $S(t) \cap R$

Q2 Given a rectangle R and time values t_1, t_2,
report all points that pass through R during the time interval $[t_1, t_2]$.





Kinetic Range Searching

(A., Arge, Erickson, 2001)

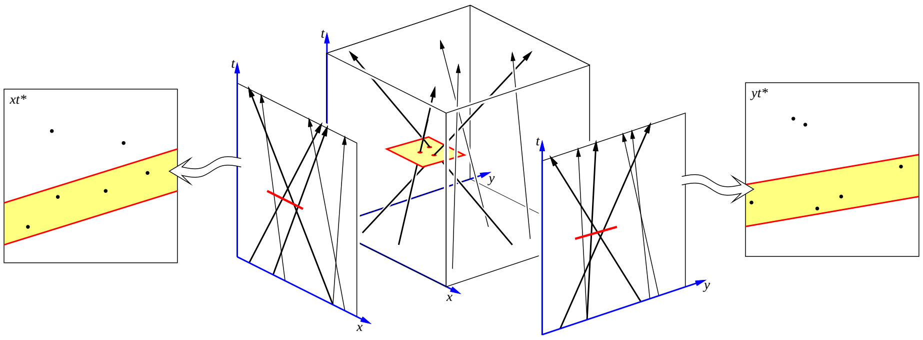
☆ Partition-tree based approach

- $O(n)$ space, $\approx \sqrt{n} + k$ query time
- $O(\log^2 n)$ insertion/deletion/trajectory-change
- Time oblivious scheme

☆ Kinetic range trees

- $O(n \log n / \log \log n)$ space, $O(\log n + k)$ query
- Events: x- or y-coordinates of two points become equal
- $\Theta(n^2)$ events, each requiring $O(\log^2 n)$ time
- Tradeoff between # events and query time
- Queries have to arrive in a chronological order

Partition Tree Based Approach



☆ Trajectory of a point p_i is a line ℓ_i in $\mathbb{R}^3$

☆ $p_i(t) \in R \Longleftrightarrow \ell_i$ intersects (R, t)

☆ ℓ_x, ℓ_y: Projection of ℓ onto the xt- & yt-planes

☆ ℓ intersects $(R, t) \Leftrightarrow \ell_x$ intersects (R_x, t) & ℓ_y intersects (R_y, t)

☆ Use duality and partition trees

Kinetic Range Searching

Time Responsive Approach:

- ☆ Near-future queries are more critical than far-future queries
- ☆ Fast query time for near-future queries
- ☆ Measure *future* by the number of events occurred
  - Measures how much the point configuration has changed

(A., Arge, Erickson, 2000)

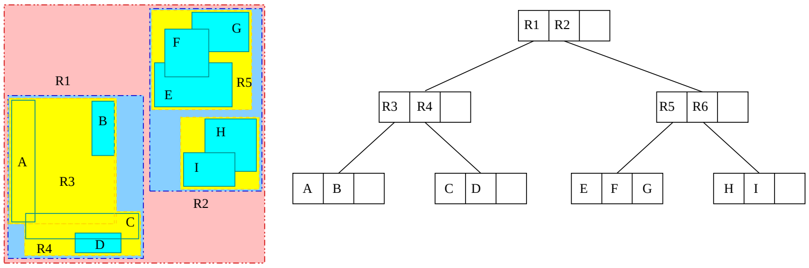
- ☆ Combine partition trees and kinetic range trees
- ☆ Space $O(n \log n / \log \log n)$, # events $O(n^2)$
- ☆ Prove bounds on the query time if trajectories are uniformly distributed

(A., Arge, Varenhold, 2001)

- ☆ Similar bounds but without any assumption on trajectories

R-trees

- ☆ Bounding box hierarchy, B-tree
- ☆ Each node v is associated with a subset S_v of points and the smallest rectangle R_v containing S_v
- ☆ Partition S_v into B clusters, each associated with a child of v
- ☆ Several heuristics are proposed for partitioning S_v into B clusters



Kinetic R-trees

☆ Maintain the smallest box enclosing the set of moving points

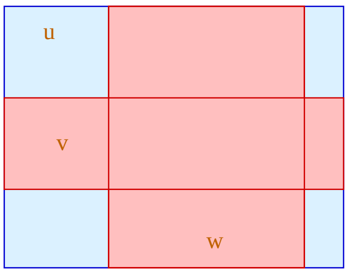
- Use a parametric representation to maintain the box
- The combinatorial structure can change $\Omega(n)$ times
- Maintain an approximation of the smallest enclosing box

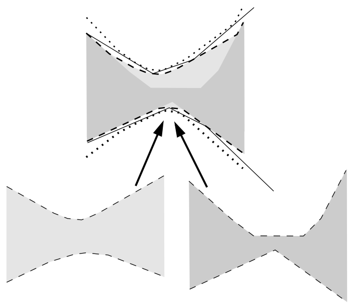
☆ Maintaining the clustering kinetically

- Extend the R*-tree heuristics

Minimum Bounding Rectangle

Merge the rectangles computed at the children of a node





Merging the x-extent

☆ Compute the upper envelope U of the upper chains; approximate it by a simpler chain lying above U

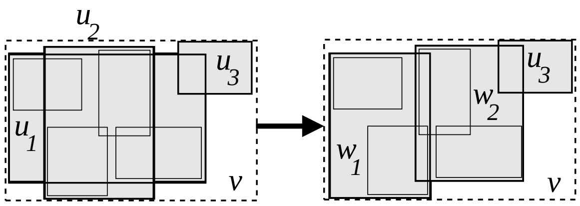
☆ Do the same for the lower chains

☆ b: A parameter
Store at most b edges of each chain to reduce the storage

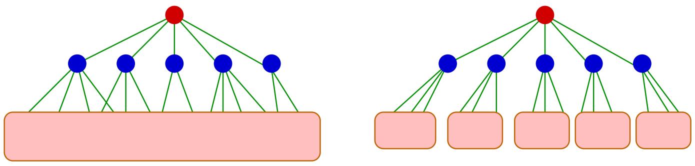
☆ *Box event:* No longer have a valid rectangle at a node

Internal Events

- ☆ Reorganize the children of a node if the rectangles of their children overlap a lot.



- ☆ Collect all the grandchildren of the node

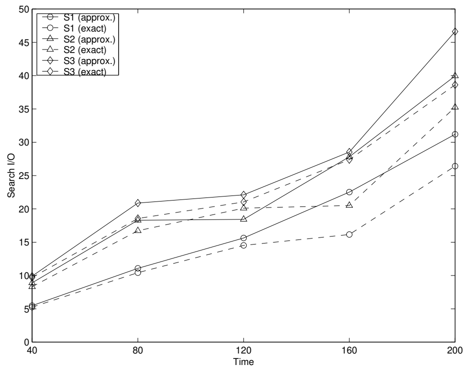


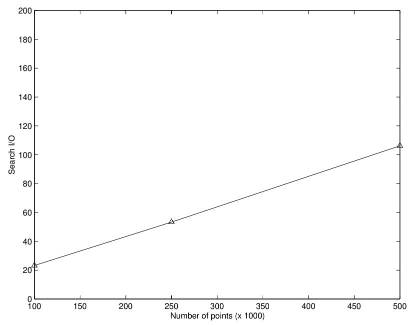
- ☆ Reconstruct a 2-level R-tree on them

Experimental Results

Synthetic Data

- ☆ 100,000–500,000 points inside $1000 \times 1000\ km^2$ area with different distributions
- ☆ Points are inserted/deleted dynamically, at any time at least 80% points present
- ☆ Three range of speed: 45 km/h, 75km/h, 180 km/h



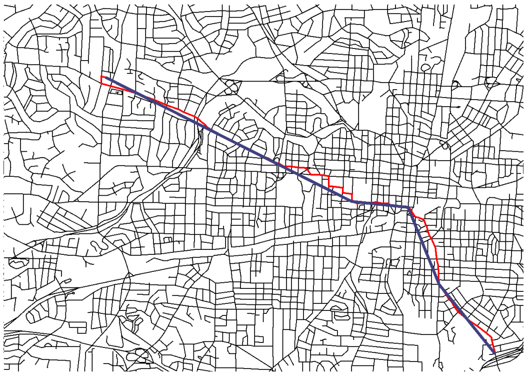


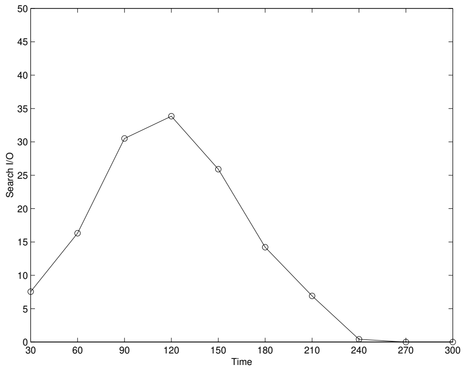
Experimental Results

Realistic Data

- ☆ Extracted the roads map around Durham, NC, within 120 miles centered at Durham ($\approx 250,000$ polygonal chains)
- ☆ Computed a planar map of the road network
- ☆ Chose source and destinations randomly with some distribution
- ☆ Computed a good path using Dijkstra's algorithm — minimize the length + number of turns
- ☆ Used Douglas Peucker algorithm to simplify the paths

Experimental Results





Concluding Remarks

- ☆ Incorporating more realistic motions
- ☆ Hierarchical representation of motion
- ☆ Tradeoff between updates and queries
- ☆ Practical kinetic data structure for nearest-neighbor queries