Kinetic Data Structures

Geometric Algorithms

Kinetic Data Structures
Motivation

• Motion is everywhere.

• Many computer science disciplines (graphics, robotics, vision, ...) deal with the modeling of motion.

• Motion data change due to interaction between objects.

• Modeling the physical world with the computer needs to combine discrete and continuous aspects.
Objectives

• Simulate system of continuously moving objects.

• Efficiently maintain discrete attributes of objects:
  – closest pair of objects
  – convex hull
  – minimum spanning tree
  – binary space partition
  – ...

Example: Quadtree

http://www.cs.umd.edu/~mount/Indep/Ransom/test0.htm
Example: Kinetic Collision Detection

http://www.win.tue.nl/~speckman/demos/kcdsp/index.html
Example: convex hull
Example: convex hull
Dynamic data structures???

- Allow insertions and deletions of objects at discrete times.
- Not suitable for handling moving objects!
Time sampling approach

• Choose fixed time step.

• Update the positions of moving objects at each time step.

• Update the data structure with the new positions of objects.
Time sampling approach

• How to choose the proper time step?

• Oversampling

• Undersampling
• Combinatorial changes occur in irregular patterns.
Kinetic Data Structures

Data:

value/coordinate = (known) function over time

"flight plan"

for instance

• affine \( a + bt \)
• bounded-degree algebraic \( a = bt + ct^2 \)
• pseudo-algebraic: any certificate of interest flips true/false \( O(1) \) times

Today for simplicity: only affine
Kinetic Data Structures

Operations:

• modify($x, f(t)$): replace $x$’s function by $f(t)$
  “motion estimate accurate for a while”

• advance($t$): go forward in virtual time

• other updates/queries usually about present (virtual) time
Kinetic Data Structures

Approach:

• store data structure accurate now
• augment with *certificates*: conditions under which the data structure is accurate which are true now
• Compute *failure time* for each certificate
• store failure times in a priority queue
• as certificates invalidate, fix data structure & replace certificate
Example: convex hull

Certificates = proof of correctness:
- \(a\) is to the left of \(bc\)
- \(d\) is to the left of \(bc\)
- \(b\) is to the right of \(ad\)
- \(c\) is to the left of \(ad\)
Example: convex hull

Certificate

<table>
<thead>
<tr>
<th>Certificate</th>
<th>Failure time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a is to the left of bc</td>
<td>never</td>
</tr>
<tr>
<td>d is to the left of bc</td>
<td>never</td>
</tr>
<tr>
<td>b is to the right of ad</td>
<td>$t_1$</td>
</tr>
<tr>
<td>c is to the left of ad</td>
<td>$t_2$</td>
</tr>
</tbody>
</table>

Never
Performance measures

• A KDS is called
  – **responsive** when certificate expires (event), can fix data structure quickly
  – **compact** no. of certificates is small -> low space
  – **local** no object participates in many certificates -> modify is fast
  – **efficient** if the worst-case number of events handled by the data structure is small compared to some worst case number of „necessary changes“
    (usually study worst-case behavior for affine/pseudo-alg. data with no updates)
Example: Find-Max

- Set of n points moving continuously along y-axis, each with constant (but possibly different) velocities.
Find-Max in BST

• First try: maintain sorted order in BST
• Certificates: \( \{x_i \leq x_{i+1}\} \) where \( x_1, \ldots, x_n \) is an in-order traversal
• Failure = \( \inf \{t \geq \text{now} \mid x_i(t) \geq x_{i+1}(t)\} \)
• Advance\((t)\):
  – While \( t \geq Q.\text{min} \):
   • now = \( Q.\text{min} \)
   • event(\( Q.\text{delete-min} \))
  – now = t
Find-Max in BST

- Event($x_i \leq x_{i+1}$):
  - Swap $x_i$ & $x_{i+1}$ in BST
  - Add certificate ($x_i \leq x_{i+1}$)
  - Replace certificate $x_{i-1} \leq x_i$ with $x_{i-1} \leq x_i'$
    and certificate $x_{i+1} \leq x_{i+2}$ with $x_{i+1}' \leq x_{i+2}$
  - Update failure times in priority queue
Find-Max in BST

- responsive: $O(\log n)$
- local: $O(1)$
- compact: $O(n)$
- efficient: $O(n)$

-> efficient if you want order, inefficient for max
Tournament tree

2nd try for find-max

Geometric Algorithms  Kinetic Data Structures
Tournament tree
Tournament tree

For each internal node maintain in an event queue the next time where the children flip order.
Processing of an event: Replace the winner and replace $O(\log(n))$ events in the event queue
Takes $O(\log^2(n))$ time $\Rightarrow$ responsive

Linear space $\Rightarrow$ compact

Each point participates in $O(\log n)$ events $\Rightarrow$ local
What is the total # of events?
Events at r correspond to changes at the upper envelope, let's say there are $O(n)$

Events at 1 correspond to change at the upper envelope of \( \{b \ d\} \Rightarrow O(n/2) \ldots \)

In total we get $O(n \log(n))$ events $\Rightarrow$ efficient
Handling insertions/deletions?
Use some kind of a balanced binary search tree

Each node charges its events to the upper envelope of its subtree

Without rotations we get $O(n \log(n))$ events
Kinetic Heap

- 3rd try for find-max: a heap
- want find-min (& delete-min ) in O(log n) time
- store in a min-heap
- certificates: \( x \leq y \) and \( x \leq z \)
- event \( (x \leq y) \):
  - swap \( x \) & \( y \) in the tree
  - update certificates
Kinetic Heap

• responsive: \( O(\log n) \)
• local: \( O(1) \)
• compact: \( O(n) \)
• efficient: \( O(\log n) \)
Amortized analysis

- Consider $\Phi(t) = \sum_x$ number of descendants of $x$ at time $t$ that will overtake $x$ in future $>t$
- $\Phi(0)$ is in $O(n \log n)$
- $\Phi(t)$ decrements with every event (blackboard)
  $\rightarrow O(n \log n)$ events
Efficient KDS exist for

- 2d convex hull
- Closest pair
- Delaunay triangulation
- Diameter, width of point sets
- Collision detection
- ...