Computational Geometry

4-18-2006

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment?

Convex Hulls

- The idea is that we are given a set of points and we want to find the smallest convex polygon such that all the points are either interior to or on the edge of that polygon.
- We will talk about two methods that use rotational sweep to solve this. There are also incremental methods, divide-and-conquer methods, and prune-and-search methods.
- A convex hull can be used to find the most distant pair of points in a set.

Graham's Scan

- Pick the lowest point and sort the other points by angle between that point and them.
- Put the first three points on a stack.
- Loop through all the other points in sorted order.
  - While the angle including the top two points turns right pop a point off the stack.
  - Push the point with the angle going left onto the stack.
- After this is done the stack contains the points for the convex hull in counterclockwise order.
- This process is O(n log n) for the sort.

Jarvis's March

- Again pick the lowest point as the starting point. Then find the point with the smallest polar angle relative to that point and include it.
- Repeat this process using the newest point added each time.
- This process is O(nh) where h is the number of segments in the convex hull.

Closest Pair of Points

- Brute forcing this is an $O(n^2)$ algorithm. We will do it in O(n log n) with a divide an conquer method.
- Each invocation will get a subset of points. We have two arrays with the points sorted by x and y respectively. The recursion terminates when n<=3 at which point we use brute force.
- We divide the set with a vertical line such that half the points are on each side. We make recursive calls on those two subsets and get the closest points on each half.
- The shortest pair for the whole set is either one of those two or contains points separated across the dividing line.

Combining Solutions

- The trick is in finding if there are points across the boundary that are closer. Given the smaller of the two separations for the two sides we make an array Y' that is the elements of our y sorted array that are within the shortest known separation distance of the separation line.
- For each point in Y' we consider the 7 points after it in the array and see if any are less distant than what we have seen previously.
- The key to this is that there only have to be 7 points because no more than 8 total points can be that close without one of our recursive checks finding a pair.

KD-Trees

- This is a spatial tree that in binary in nature. Each node splits the children across a specific value of one axis.
- These are simple to write and are good for high dimensional spaces.
- There are different ways for picking how and where to split. There are also questions for how many data elements can be in each node.
- Let's consider the simple case of only allowing one data point per leaf node and nothing internal.

BSP-Trees

- Imagine a KD-tree with the split made in a random orientation, not axis aligned. This is great for representing complex geometries and doing efficient collision/overlap detection.

Reminders

- I have nothing to remind you of.