Nearest Neighbor Design Notes

Bill Rubin

July 27, 2010

Problem Statement Modifications

I modified the problem statement as follows:

1. List of points and target point are not read in; they're just embedded in the program. I saw no advantage in performing this input operation, especially since the input file format is unspecified.
2. Assumed purpose: A smart phone end user is attempting to touch an object displayed on the touch screen. Therefore, the target is assumed to be quite close to at least one point.
3. If the end user touches far from any points, the algorithm will simply report failure, rather than still trying to find the nearest point no matter what.
4. Effort is taken to organize the set of points before dealing with the target point, as per Marty's email. Thus, the effort to process a single target point is minimized. (Multiple target points are assumed to be run against a single set of display points.)

Two different approaches: Region-based and Hash-based

I implemented two different approaches, region-based and hash-based, both in Visual C++.

Region Based Approach

Design: Partition the 601x601 screen into square regions (portions of regions on the screen edges). The algorithm starts by searching for the closest point in the region containing the target. If other regions are near the target, they may be searched too, if doing so could find a closer point. Up to three neighboring regions (one horizontal, one vertical, and one diagonal neighbor) are searched before reporting the closest point or giving up. Manhattan distance is used for efficiency.

Performance: In order for this to work, the array of input points is pre-sorted so that *the points are ordered by region*. An array of *region iterators* pointing into the point array is then pre-computed. The pre-sorting is an order **n log n** process; computing the iterator array is an order **n** process, where n is the number of input points. The target search only requires constant time, plus an order **k** effort, where k is the number of points in the target region. Presumably k << n.

Region Size: The size m of the regions determines how far from the target the algorithm will search before giving up. Larger values of m require more effort to find the closest point, but will allow the closest point to be further away. If m=601 (screen size), the algorithm reduces to the brute force approach of computing the distance to every point. With 1,000 points, this is 1,000 point comparisons.

If m=301 (half the screen size), an average of only about 270 random point comparisons are needed. That's because regions are not searched if they cannot improve the result. In the worst case, all four regions are searched, but that worst case is very unlikely to occur.

For m=151, an average of 80 point comparisons is needed.

For m=50, an average of only about 16 point comparisons are needed, with range between 8 and 32 point comparisons. This is only 1.6 percent of the brute force effort.

Point Acceptance Criterion: If the algorithm finds no points in any of the (up to four) regions explored, it gives up. On the other hand, if the algorithm finds a point in an explored region, it only accepts the point as a candidate if the point is within the *radial distance* from the target. The radial distance is target-dependent. If the target is at a corner of its region, the radial distance is m. If the target is at the center of its region, the radial distance is m/2. So the radial distance varies between m/2 and m, depending upon where the target lies in its region.

Without this point acceptance criterion, that is, if all points within each of the four regions is allowed, the algorithm can select a "closest" point which is actually not closest. In the worst case (target is at the center of its region), the algorithm could incorrectly select a point at the far end of the diagonal region as "closest", when a point in an anti-neighboring region is closer: up to 6 times closer using manhattan distance, and up to around 4.2 times closer using Euclidean distance.

Hash Based Approach

Design: Store all points in a hash table. Beginning at the target, look up all points in concentric "circles" around the target in the hash table. When one is found, it is the closest. The algorithm can "give up" if it finds no points within some distance d of the target (though I didn't bother to implement this).

Performance: The preparation step (storing all n points in the hash table) is an order **n** process. Looking up one point in the hash table requires only constant time. The target search requires looking up K points, where K is proportional to d^2, and d is the distance to the nearest point. So lookup is an order $\mathbf{d^2}$ process. Thus, target search time increases rapidly with distance to the nearest point.

Comparing Region and Hash Based Approaches

The two approaches are fundamentally different. In my view, it would be difficult to meaningfully compare their performance without re-implementing both of them for a smartphone platform. My Visual C++ implementation makes heavy use of standard library sorting and hashing.

Performance of Preparation Step: The preparation step for region based is order n log n, while that for hash based is only order n. But these asymptotic performance properties could be overridden by other factors when n is sufficiently small.

Performance of the target search step for the region based approach: This depends in a complicated way on the region size and the point distribution. The region based approach also has a natural failure mode (consistent with the modified problem statement). The computation time tends to be roughly equal for all targets, but the closeness criterion varies by a factor of 2 with the target point.

Performance of the target search step for the hash based approach: This increases with the square of the distance to the nearest point. Although there is no natural stopping criterion (unlike for the region approach), it would make sense to terminate the algorithm after a certain small distance is reached, so that execution time does not get out of hand (figuratively speaking).

Complexity of Approaches: In the C++ implementation, the region-based approach is noticeably more complicated to design and implement, with many domain-specific issues. The hash-based approach is relatively simple to implement; the only tricky part is incrementing the iterator in concentric "circles" around the target. However, both approaches employ nontrivial library algorithms for sorting (region) and hashing (hash).