Nearest Neighbor Search

William Jagels and Rushil Kumar

December 19, 2016

Abstract

In this paper, we set out to research and implement algorithms for Nearest Neighbor Search (NNS). Our implementation is able to make point clouds with arbitrary dimensional points. The nearest neighbor is also suited to work with these point clouds at any dimension. We discuss our testing results and compare our implementation to other common implementations

1. Problem

Given a target point, and a list of other points, find the closest point to the target. This problem appears in the everyday world, such as finding the nearest stores on a retailer's website or finding the closest airport for an aircraft in distress. Although many appearances of this problem are trivially solved by a linear search, sometimes a more efficient algorithm is needed.

2. k-Nearest Neighbors

We can observe that with the regular NNS problem, it is impossible to get any better than $\Theta(N)$ time unless we can make assumptions about the ordering of our point cloud. In the k-NNS problem, where we find the closest k neighbors, our naïve linear search will degenerate into a $\Theta(kN)$ time complexity algorithm. At larger values of k, it becomes clear that a linear search is no longer reasonable.

3. k-Dimensional Tree

One data structure commonly used to solve the NNS problem is a k-dimensional(k-d) tree. A k-d tree is a binary search tree that is sorted in multiple dimensions, allowing for a $O(n \log n)$ search time. This data structure uses space partitioning to achieve its performance characteristics. At each level of a k-d tree, there is a specified axis that the left and right subtrees are divided by. We implemented and used a k-d tree specifically for nearest neighbor search, but it can also be used for problems such as range searching.

4. k-d Tree NNS

We implemented a nearest neighbor search that works for our k-d tree implementation, and found that it could reliably find the nearest neighbor to a point very quickly.

5. Results

Using $3 * 10^6$ points in 3 dimensions, linear search takes on the order of 10^{-2}s. On the same test, k-d tree construction takes on the order of 10^0s, and NNS takes on the order of 10^{-6}s. It is important to note that the k-d tree construction only has to happen once per point cloud, and any additional nearest neighbor searches will take the same amount of time. Since the k-d tree construction is slower by a factor of 10^2, if we perform any more than 10^2 searches, we should see a performance gain over the linear search. 10^2 searches may seem large, however, that is 0.0033% of the size of the point cloud.

6. Analysis

Our tests revealed that our NNS algorithm is very fast on an already created k-d tree, but the k-d tree construction is rather inefficient. Rather than attempt to optimize our own k-d tree (reinventing the wheel), we adapted our code to run on more popular k-d tree implementations.

7. Further Testing