Joseph Min
Computer Integrated Surgery II
Seminar Presentation
21 April, 2016

"Adaptive simplification of point cloud using *k*-means clustering"
Authors: Bao-Quan Shi, Jin Liang, Qing Liu

Introduction

Our project uses an image-range camera to generate a point cloud, which we register to a surface mesh created from CT scans. We then use this registration and an AX=XB calibration for the Robotic Ear nose and throat Microsurgery System (REMS) robot to align the robot in preparation for surgery to allow for a safer, more accurate procedure. However, our generated point clouds are generally pretty large (>200,000 points), which has high space and processing costs so a big struggle has been to simplify our point clouds. The paper that is reviewed here outlines a technique to take a dense point cloud and simplify it using a technique that preserves the integrity of the point cloud itself, which was one solution to our problem that we examined.

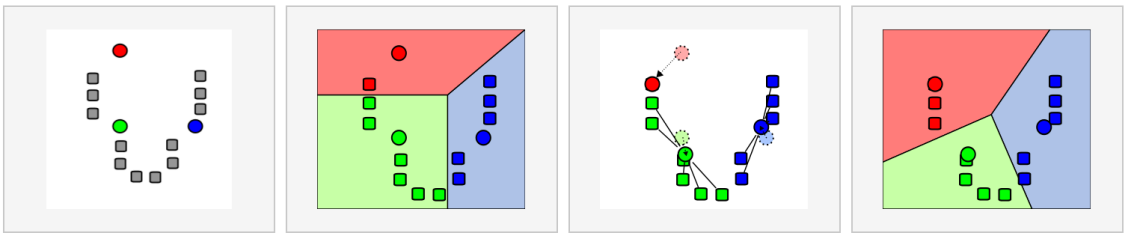
Summary

Though there have been many algorithms put forth previously to reduce the number of points in a point cloud, they each had their limitations. The simplification algorithm created by Shi et al. is meant to address three of those major concerns: to efficiently and effectively identify as many redundant points as possible, to keep the boundary integrity of the point cloud as intact as possible, and to balance the resulting point cloud as much as possible. The paper aims to quell these concerns by applying the *k*-means clustering technique to find average points (also known as means or centroids) to be representative of a set of similar points (reducing redundant points), using boundary detection to place means near the true boundary of the

Technical Approach

K-means clustering

K-means clustering is a popular method for cluster analysis, and it aims to split *n* observations (or data points) into *k* means, where each of the *n* data points belongs to the cluster of the mean that it is closest to. It does so by initializing *k* means (often randomly, but we will see soon how the initialization step happens for algorithm in the paper), and associating every point with the original means. Then, a new mean is calculated by taking the average feature (in this case, Euclidean distance) of the points in each cluster. The cluster partitions are recalculated from these new means, and the process repeats until convergence. Here is the algorithm in action for a 2D set of points:



For a 3D set of points, the general process is the same, but uses 3D clusters instead.

In this paper, the initialization step is not random, but is instead determined by the following procedure:

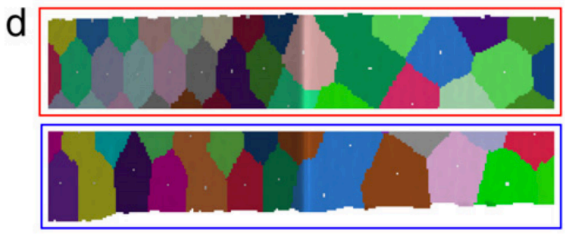
1. Build a k–d tree [40] using the N input points in R^3.
2. For $i = 1$ to N do
3. If P_i is non-marked, search the fixed radius neighbors of P_i, here the fixed radius (space interval—DT) should be manually specified by users.
4. Mark those fixed radius neighbors.
5. End if.
6. End for
7. Select non-marked points as initial cluster centroids.

This algorithm basically takes the first point as the mean (referred to above as a centroid) of the first cluster, takes all the points within some user-defined Euclidian distance (*DT*) in its cluster, and marks them as ineligible to also be means of a cluster. It then iterates until it finds an unmarked point, makes this the mean of the next cluster, and repeats the process until every point is either the mean of an initial cluster or is a member of a cluster. The number of initial means is our *k*. Because of the use of that user-defined Euclidian distance, the centroids are, at the end of this step, uniformly distributed (all roughly *DT* from each other).

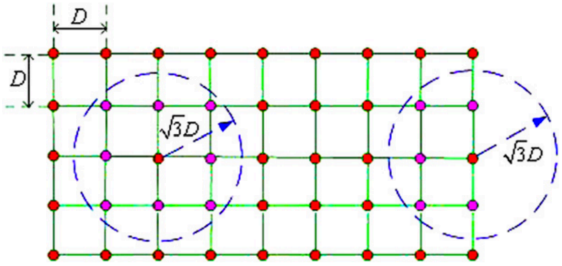
Note that the use of *k*-means clustering in this paper is limited to the initialization step – the subsequent convergence steps are skipped.

Boundary Detection and Preservation

Because of how the initial clusters are chosen, it is easy to see how it would be possible that some centroids would be far from the true boundary of the original object. An example of this is shown to the right, where the centroids (the white dots in the middle of the colored clusters) of a few clusters of a rectangular object are distant from the edge of the original object.

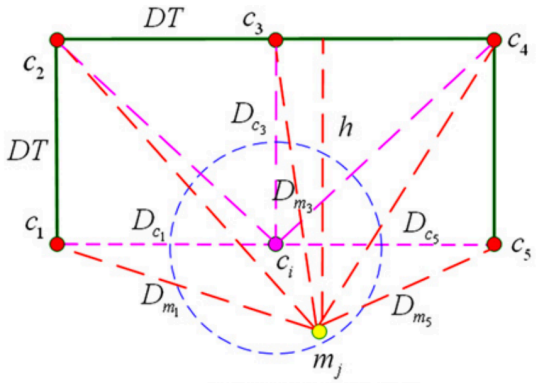


To consistently determine if a centroid is too far from a true boundary, a boundary detection algorithm is defined. First, we need to find the clusters on the boundary of the object.



This is done by counting the number of centroids within $\sqrt{}$(3)*D away from a given centroid, and if that number is fewer than 6, then the given centroid must be that of a boundary cluster. This is made possible due to the fact that the initialization step generates roughly evenly spaced centroids as noted before. A 2D example is shown to the left.

Now that we have our boundary clusters, we have to determine if their centroids are close enough to the boundary to preserve boundary integrity. They do so by utilizing an algorithm that essentially finds the point in a boundary cluster with the furthest average distance from the centroids of neighboring clusters (the ones within that $\sqrt{}$3*D radius), and determines if that average distance is above some user-defined threshold. If so, that particular boundary cluster has a centroid that does not preserve boundary integrity.



To the left is a 2D example of such a situation in which c_i represents the centroid of the current cluster, c_1-c_5 represent the centroids of neighboring clusters, and m_j represents any point in the current cluster. Every D_c is the Euclidean distance from c_i to the centroids of neighboring clusters, and every D_m is the Euclidean distance from m_j to the centroids of neighboring clusters. We examine every m_j in the current cluster and find the point with the highest average distance from the centroids of the neighboring clusters. That is, we define D_M as $D_M = \frac{1}{N}\sum_{j=1}^{N} D_m$, and proceed to find max(D_M). We then compare this to D_C which is defined as $D_C = \frac{1}{N}\sum_{j=1}^{N} D_c$. If max($D_M$) > αD_c for a user-defined α, then the centroid of the current boundary cluster is deemed to be too far from the true boundary.

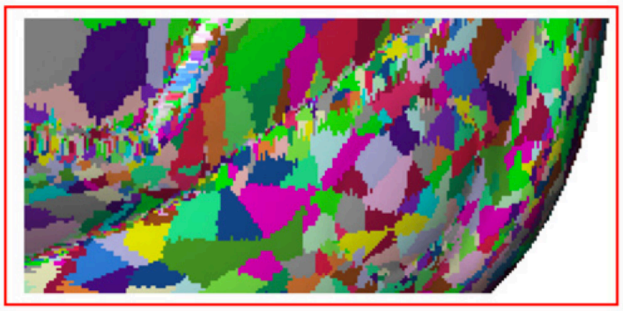
To correct this issue, the boundary cluster is split into two subclusters where one has a centroid that is the same as the original cluster and the other has a centroid that is the point that corresponded with the max(D_M). The members of the original cluster are sorted into the two subclusters using the standard *k*-means process as described previously.

Cluster Subdivision

We now have a cloud that has roughly evenly distributed points (where each point is a centroid of each cluster) but we have lost a lot of detail because of this. It is clear to see that places with higher detail or change in features would need more points to represent said detail. To create more points to represent these features, the paper introduces cluster subdivision, which takes clusters that need higher resolution and recursively splits the clusters into smaller clusters.

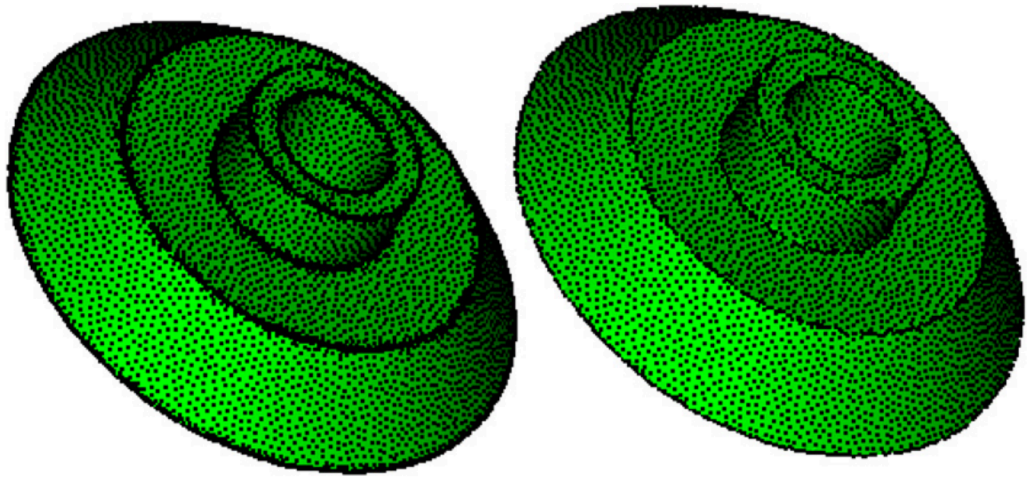
In order to identify the clusters that need higher resolution, the paper takes the original cloud data set and estimates the surface normal of each point by “the fitting plane obtained by applying a least squares method to the *k* nearest neighbors of the point in the point cloud”. Then, for each cluster, it compares the normal vector deviations of every point to every other point, and finds the maximum normal deviation. If this maximum normal deviation is above a user-defined threshold, the current cluster is divided into two subclusters whose centroids are the two points with the two highest vector normal deviations, and the points of the original cluster are placed into subclusters appropriately. Below is an image depicting how more curvature leads to a higher number of smaller clusters.

d



Cluster Refinement

Now we have a relatively representative cloud, but it is very unbalanced – areas of high detail have the necessary definition now, but might be too densely outlined as seen in the green pulley below on the left. So, the paper proposes a method to refine and balance the distribution of clusters.



It is essentially as follows: because the clusters are split into two subclusters with every subdivision, the procedure establishes a binary tree structure of clusters. Thus, importance of a particular subcluster could be defined as a function of its depth in such a tree (distance below an initial cluster) and the number of points in that particular cluster. The importance weight is defined in the paper as the quotient of a particular subcluster's depth and its number of member points: $W_i = Dep_i/nCount_i$ These weights are then analyzed by the refinement algorithm below, and subclusters deemed as unnecessary are eliminated.

1. If $Dep_{Ci} > 2$, where Dep_{Ci} is the depth value of the initial cluster c_i.
2. Sort the sub-clusters in descending order according to their weights.
3. If $d(subc_j, c_j) < \delta DT$ /// in our implementation $\delta = 1/3$.
4. Remove the sub-cluster $subc_j$.
5. End if.
6. If $W_{subc_j} == W_{subc_1}$, $j \in [2, N]$, where N is the number of the sub-clusters.
7. Push $subc_j$ into the queue **Q**.
8. End if.
9. Iteratively compute $d(subc_j, subc_k), j \in [1, \mathbf{Q}.size()], k \in [1, \mathbf{Q}.size()], j \neq k$.
10. If $Maximum(d(subc_j, subc_k)) > \delta DT$
11. Preserve $subc_j$ and $subc_k$. ///Keep two sub-clusters.
12. If $d(subc_j, subc_l) > \delta DT$ && $d(subc_k, subc_l) > \delta DT$
13. Preserve $subc_l$. /// $l \in [1, \mathbf{Q}.size()]$
14. End if. // Keep another important sub-cluster.
15. If $d(subc_j, subc_m) > \delta DT$ && $d(subc_k, subc_m) > \delta DT$
16. Preserve $subc_m$. /// $m \in [\mathbf{Q}.size(), N]$
17. End if. // Keep a sub-cluster in the flat area.
18. Else
19. Preserve $subc_1$
20. End if.
21. End if.

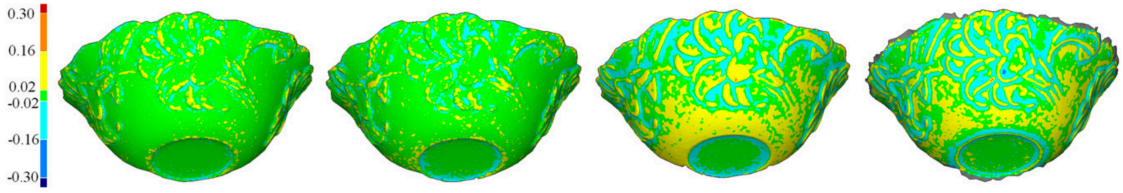
Results

Overall, the results of this algorithm are pretty impressive. As seen in the table below, it can drastically reduce the number of points in a data set (in some cases to a number that was less than 7% of the original number of points), and incredibly quickly, even for the more complicated shapes.

Time statistic of the novel method for different point cloud sets. The time data was collected on a PC with an AMD Athlon 64 X2 dual core processor 4400++, 2.0 GB Memory.

Models	Original number	Timings for different steps			Preserved number
		Cluster initialization (s)	Boundary cluster detection (s)	Recursive subdivision and refinement (s)	
Dragon	435 545	2.765	0.141	1.969	32 925
Bunny	34 834	0.25	0.016	0.063	6500
Moeller	123 397	0.781	0.047	0.156	8494
Car door	111 843	0.703	0.125	0.094	28 250
Phone shell	379 578	2.062	0.204	0.39	38 168
Lamp bracket	123 581	0.75	0.062	0.266	15 809
Twist drill	51k	0.344	0.015	0.266	2k
Huanhuan	449k	3.875	0.063	4.000	10k

The error comparison to other popular algorithms was also pretty impressive. It seems as though the feature retention was much higher for this algorithm, at least in the example the paper outlines as seen below. From left to right, the results are for the paper's algorithm, a 3D Grid method, a Curvature-Aware method, and a moving least squares method. As you can see below, the far left has the most green and the least amount of error.



Analysis

The paper outlines a relatively impressive algorithm and does so in an easily understandable way. None of the math was really that complicated, and the pseudocode was also well written. There was also error presentation for a couple distinct examples for varying thresholds of refinement, which was nice. However, some of the paper's shortcomings come from its overall results presentation and comparisons, and the lack of comprehensive error analysis and comparisons. It is obviously nice to see visual results and even a table of simplifications and their associated times, but there was never a succint, quantifiable way to judge the error of the results of the paper other than the last comparison with the bowl. This comparison also only compared error – had there been a table of number of points they were reduced to and how long it took, that would have been extremely useful and interesting as well.

The algorithm outlined in this paper is also not very useful for our project because it is very sensitive to noise due to the fact that it uses the maximum normal vector deviation as a parameter for subdivision (and noisy outliers are clearly likely to have high normal vector deviations). The data we receive from our image-range camera is not as clean as would be a 3D scan (which I think is what this paper is more targeted towards), and would likely cause high error after simplification.

Works Cited

Shi, Bao-Quan, Jin Liang, and Qing Liu. "Adaptive Simplification of Point Cloud Using Kk-means Clustering." Computer-Aided Design 43.8 (2011): 910-22. Science Direct. Computer-Aided Design, 9 Apr. 2011. Web. 21 Apr. 2016.

Pace, Winston. Digital image. *Wikipedia*. Wikimedia Foundation, 26 July 2007. Web. 21 Apr. 2016.