Point Cloud Simplification and CAD Compression

ME 735: Course Project Report

Submitted in partial fulfilment of credits for the course
ME 735 - Computer Graphics and Product Modelling
Autumn Semester, 2017-18.

by,
Karan Jain (140100023)
Abhishek Kumar Chaudhary (140100024)
Akshay Raut (140100034)
Chinmay Maheshwari (140100069)

under the Guidance of,
Prof. Sanjay S. Pande (Instructor)

Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Powai, Mumbai 400 076

November 2017

Contents

1. Introduction

1.1. Point cloud

A point cloud is a set of points which represents the geometric model of an object, essentially its surface. The points are generally obtained via laser scanning of the object from various angles and then combining all of these into one master point cloud which represents the entire object.

1.2. Point Cloud Simplification

When a point cloud is obtained via laser scanning, generally we have too many points for the object and many of those points are redundant. Point Cloud Simplification is a technique for reducing the no. of points in a point cloud, at the same time not (or minimally) disturbing the object geometry or topology, essentially the object's identity.

1.3. CAD Compression

CAD Compression is something on similar lines as Point Cloud Simplification, both are used to reduce the file size and remove any kind of redundancy. But CAD compression actually involves numerical methods to save data, and does not reduce the number of points. The numerical methods will be explained in detail later in the report.

1.4. Motivation behind this project

Every person in the world would prefer higher speed in almost any field. Reducing the file size would have a direct effect on time it takes to transfer the files from one device to another (either through hard drives, or through a cloud). Laser scanned point clouds are really big in size, and the file size can be reduced by a huge factor (of the order of 10-100) by eliminating redundant points, with small (negligible) errors associated with that information loss.

So the advantages are 2-fold:

i) Lesser memory requirement which means additional space for more files

ii) Faster file transfer rate, or lesser time and data per file for transmission.

2. Implementation Methods: 3 Approaches

2.1. Point cloud simplification with preserved edge based on normal vector

Traditionally point cloud were analysed by polygonal meshes which was and complex process, to overcome this people started looking for the simplification process directly from the points without manipulating them further. For this to work one has to do the following

a) Identify the redundant points, which may contain points which do not change the topology significantly.
b) Remove the redundant point and iterate till all redundant points are removed

2.2. Adaptive simplification of point cloud using K mean clustering

The point cloud data generated from 3D scanning devices produces huge amount of data points which requires large storage space and long processing time. So this algorithm tries to select the representative points and remove the redundant points. It uses k-means clustering algorithm to gather similar points together based on the normal vectors. Using a set of 6-6 adjacent points normal vector at each point is found. So the n data points are partitioned into k clusters with each point belonging to the cluster with nearest mean.

The point cloud is divided into small clusters and normal vectors are calculated for each points. Depending upon the deviation between the normal vectors the new clusters are formed. This will give better point cloud having higher density of points at the edges where there is sharp bend/turn and relatively less points where the curvature is less.

2.3. Compression using Frequency Domain filtering

Point cloud is essentially numerical data - it has x, y and z coordinates of various points and each point is a 4-byte floating point number. If nearby points are grouped together then storing them as the coordinates itself takes up more space. A more efficient way would be to store them in the frequency domain, eliminate some of the high frequency components and store the retained components which occupy lesser space.

Theory and explanation for the code is covered in detail in the dedicated section.

3. Point cloud simplification with preserved edge based on normal vector

We followed a similar ideas as presented in [1], the algorithm implemented is as follows

1) Construct kd-tree and search k-neighboring points for each point in the point cloud
2) Compute the normal vector for each point :
   The normal vector is defined as selecting 20 neighbors of every point and fit a least square plane passing through the queried point.
3) Detect the edge points and be retained :
   A point is characterized as edge point if all the neighbor points lie on one side of it. The ratio of difference of points lying on either side of the point of interest(in both x, y and z plane shifted there)to total number of neighbor points. This ratio was taken to be 0.7. This is a parameter which can be tuned for better results.

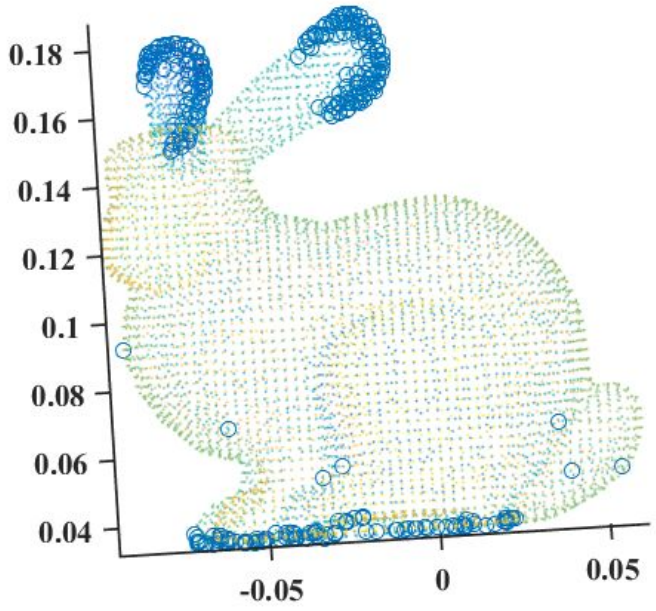
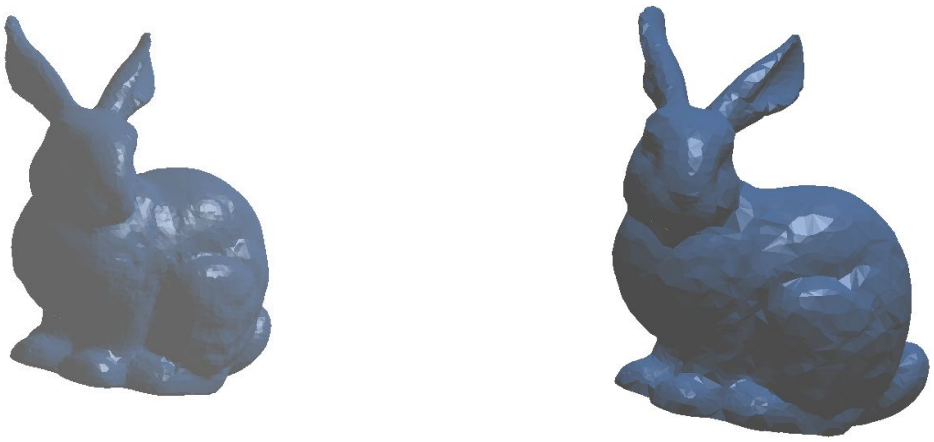


Figure 1: The bigger points denote the identified edge points and the smaller dots denote the original bunny point cloud taken from [4].

4) For non-edge points, on the basis of the normal vector, calculate the importance value for each point us. Repeat Step 5 and 6 progressively, until the simplification rate is reached
5) Delete the least important point.
6) Update the normal vector and importance value for the affected points



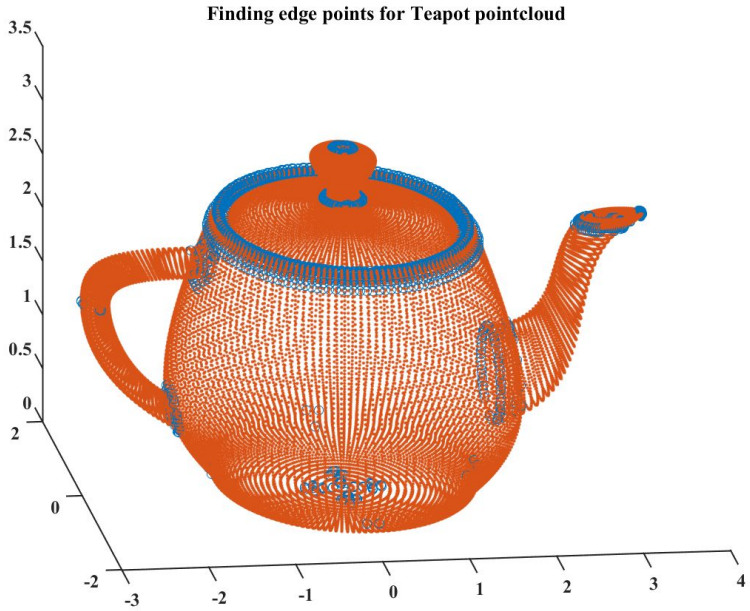
(a) (b)

Figure 2: The figure contains the original bunny (a) and point cloud reduced bunny(b)

Some more examples

1) **Teapot**

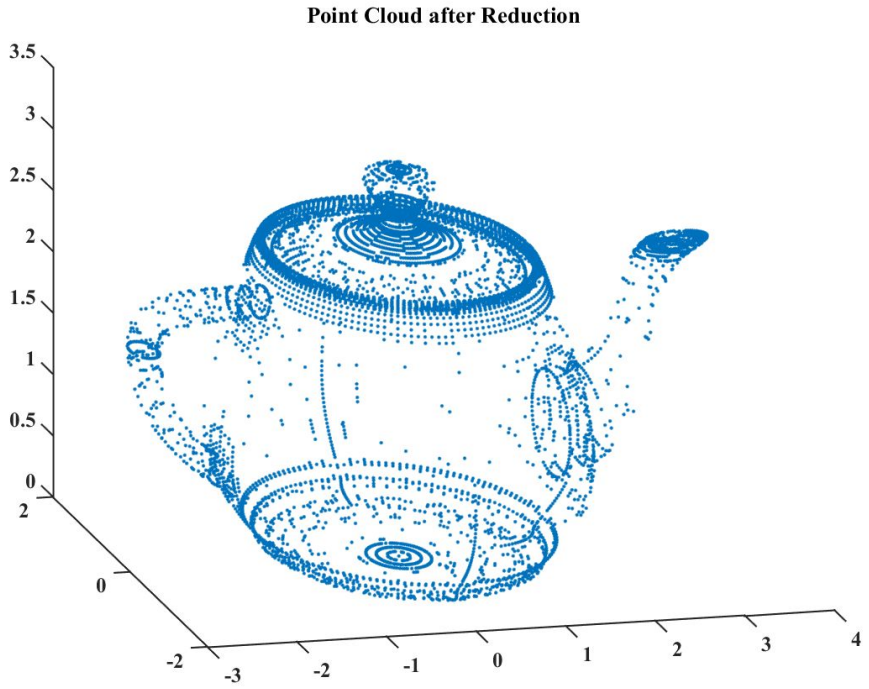
   a) **Edge detection**



Edge points are depicted by blue dots while original point cloud in red. The above algorithm detected the edge points in about 4 minutes.

b) Point cloud reduction

The algorithm reduced 41000 points to around 10000 points in about 50 minutes. The reduced point cloud is shown below

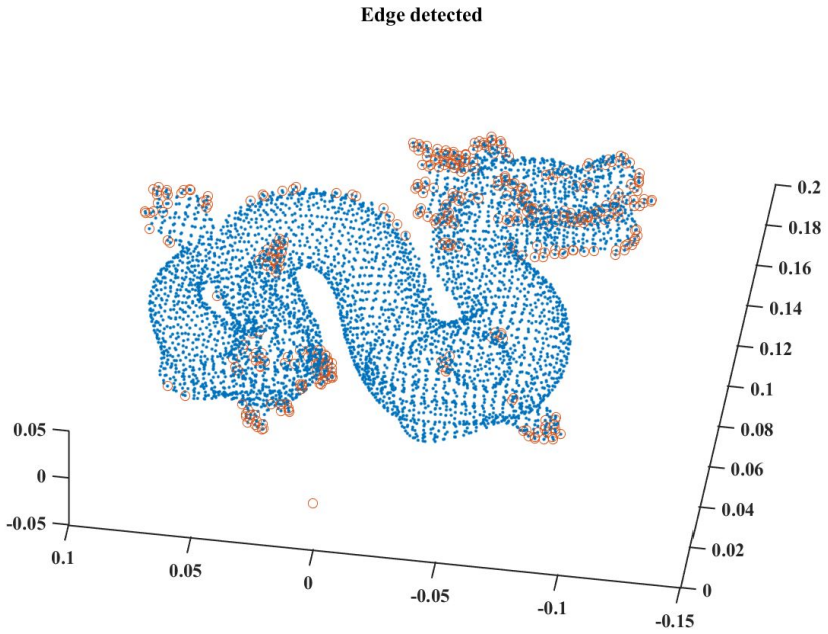


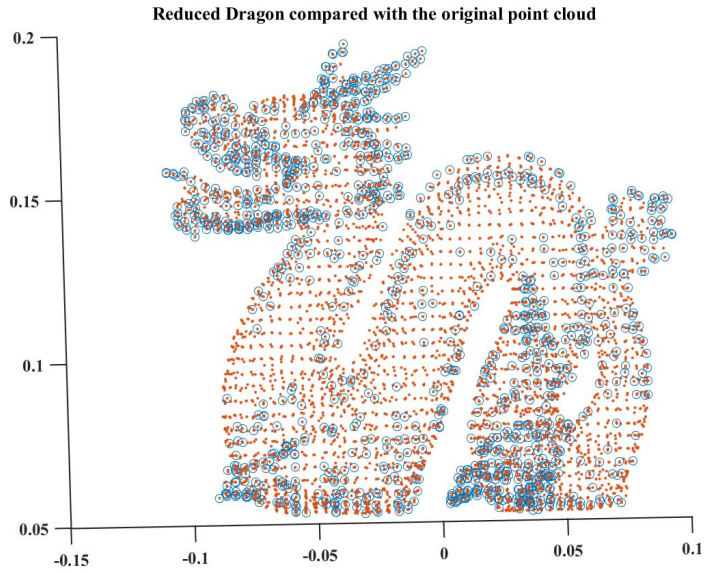
2) Cylindrical box

Image on right is original rendered point cloud with 4260 points which was reduced to 440 points (90 % reduction). Time taken by algorithm is 350 s.

3) Dragon

A reconstructed Dragon point cloud imported from stanford graphics project. Originally it had 5200 points. Reduced to 1200 points(more than 75 % reduction) in around 330 s.





Future tasks:

1) Automate the tuning the parameters quantifying the edge, non-edge points and the number of neighboring points considered, which as per present thinking depends on point cloud

4. Adaptive simplification of point cloud using K mean clustering

The following steps were performed for implementing the adaptive simplification of the point cloud :

a) Ensure that the point cloud is completely stitched in 3-D. If point cloud from multiple views are present, apply the appropriate geometric transformation and generate the completely stitched point cloud. (Let's call this point cloud as PC01)

b) Split the point cloud into number of clusters (the number of point clusters can be chosen by the user). This clusters are generated by using the k-mean cluster formation algorithm. (Let's call this point cloud as PC02)

c) Then finding the normal vectors at each and every points in the point cluster

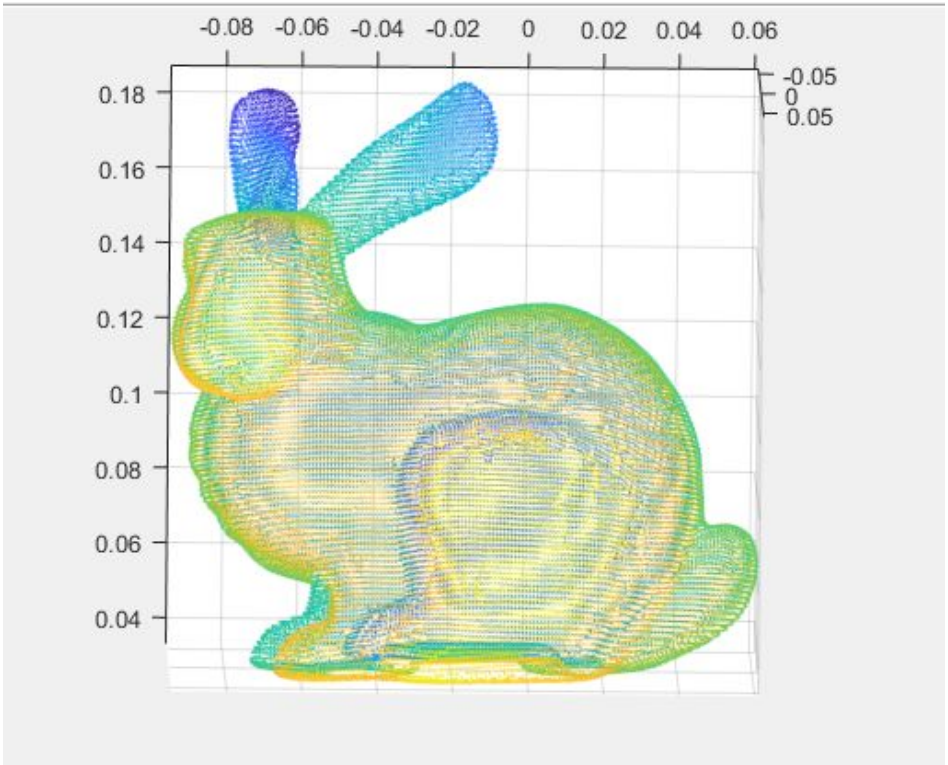
d) Further, we move on to the detection of boundary points i.e. the surface patch where the curvature is high (this critical value of curvature can be set by the user). We use the normal vector dot product to detect the curvature and predict the high curvature regions.

e) For detecting the surface patch with higher curvature, the algorithm works in the following way-
    1. Choose a particular cluster.
    2. Find the normal vector on each point belonging to that cluster using the neighbouring 6 (or more) points.
    3. Define a parameter (sum) whose value is the the summation of dot products of the consecutive normal vectors
    4. If the value (of sum) is very small (can be twerked by the user), then the chosen cluster is a boundary cluster. Else it's a surface with low curvature.

f) Hence, after detecting the surface patch with high curvature, we keep the point density in that very high using some user input parameters, so as to capture the surface patch with minimal loss.

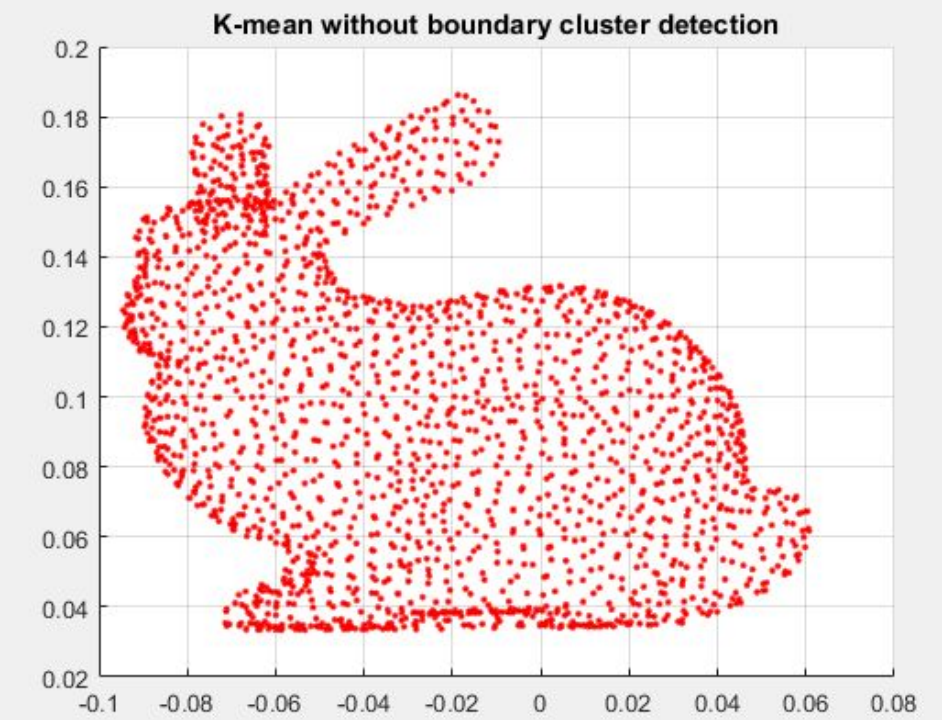
g) Below are 2 illustrative examples::

    Example no. 1
    Rabbit with approx. 40000 points was created first.
    Aim is to simplify the point cloud.



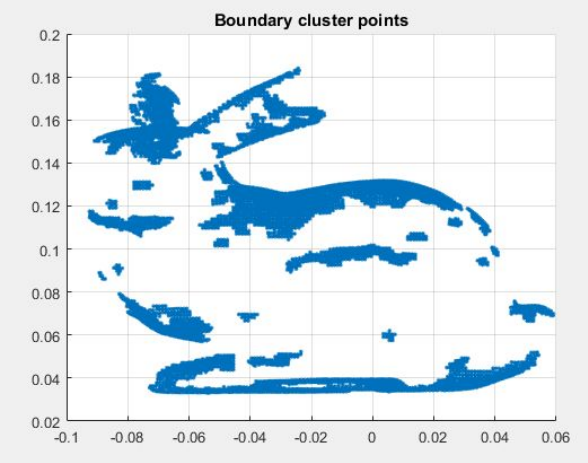
(PC01)

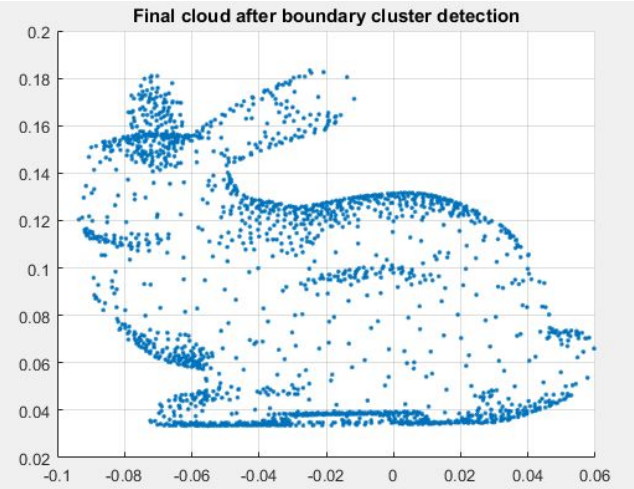
Applying K-mean, reduces the number of points from 40000 to 2000 **(5%)** in 30 seconds



(PC02)

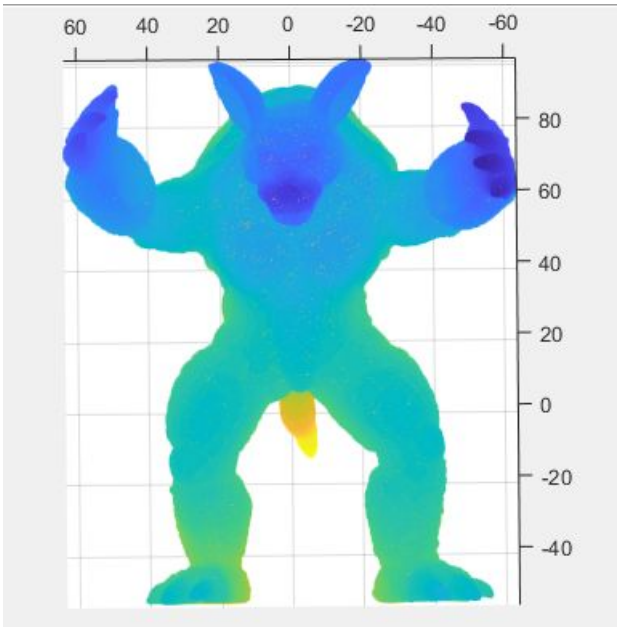
Applying Boundary cluster detection



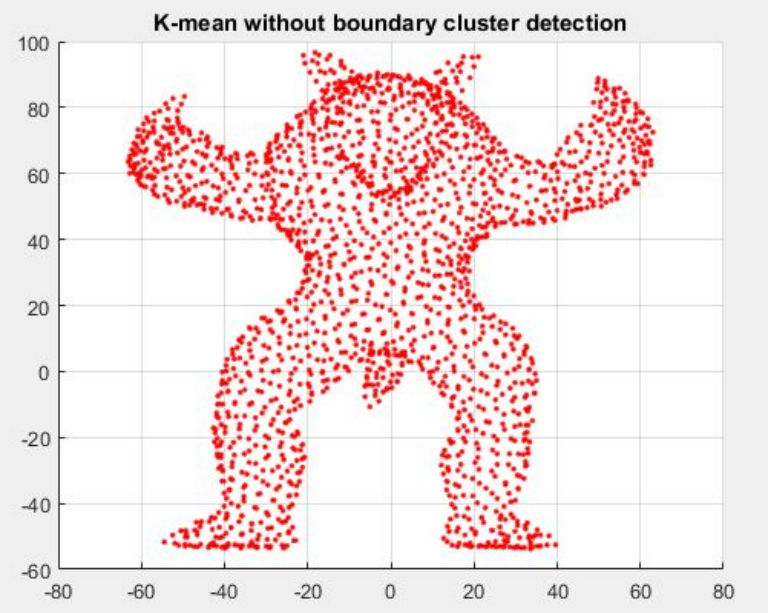


(Only Boundary Points) (Point after boundary detection and k-mean)

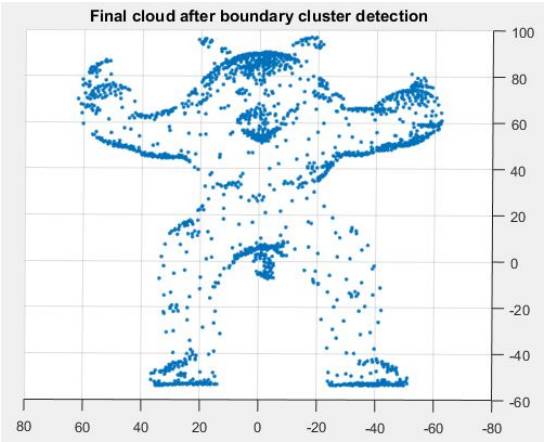
Example no. 2
Armadillo with approx. 1,72,0000 points was created first.
Aim is to simplify the point cloud.



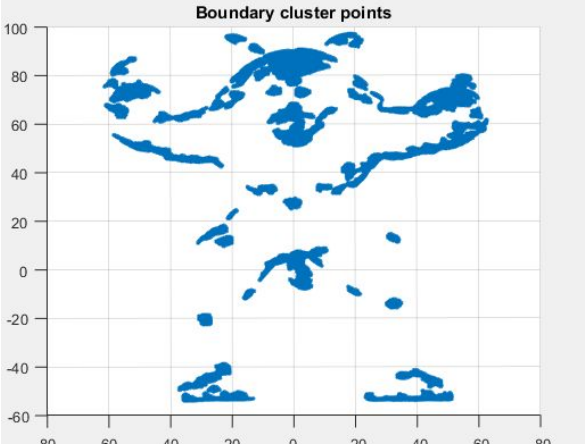
Applying K-mean, reduces the number of points from 1,72,000 to 2000 in 256 seconds



Applying Boundary cluster detection algorithm



(Only Boundary Points)



(Point after k-mean)

5. Compression using Frequency Domain filtering

5.1. An Illustrative Example

A simple illustrative example follows:
Imagine we want to store 2 numbers - x and y.
Instead of directly storing them as x and y, we can instead store them as (x+y)/2 and (x-y)/2. This data can then give us x and y back through simple manipulation.
Note that (x+y)/2 is actually their mean, and (x-y)/2 is their distance from the mean.
Now if these numbers were very close to each other in value and we can tolerate some error, then we need not store the (x-y)/2 component. (x+y)/2 will be enough information since we are within the specified error margin. Hence we save memory by eliminating the AC component - (x-y)/2 and retaining the DC component - (x+y)/2.
But, if their difference is large compared to the number themselves, then the method we use is not suitable and can give large errors.

Analogous to above example, ‘n’ numbers can be converted to their frequency domain values with one DC component and ‘n-1’ AC components. This is achieved by the Discrete Cosine Transform function or the DCT function. Now the closer the points are, the more insignificant the higher frequency components will be and lesser the error on eliminating them.
Typically, n is taken as a power of 2, because DCT computation is very fast for that.

5.2. Explanation of Algorithm

The final point cloud obtained has some number of points, each point being represented as 3 floating point numbers - one each for x, y and z coordinate. This is a total of 12 bytes per point, which is a lot, and further compression is possible.

a) Since the order of points does not matter in a point cloud, we sort the points such that the list has x-coordinates in increasing order. If multiple points have ‘close-by’ x-coordinate then they are sorted with respect to y-coordinate. Similarly, for ‘close-by’ y-coordinates, sorting is done on z-coordinate basis.
b) Finally in this ‘adaptively sorted point cloud’ to cluster nearby points together, we break the entire cloud into sets of 64 points each. Now to each of this set, we treat x, y, z independently. So we have a set of 64, 4-byte floats. We apply a discrete cosine transform (DCT) to this set, to map these numbers to the frequency domain. We will again get 64 floats which would actually be the frequency domain values for this set. They consist of a DC component (mean value) and 63 AC components of increasing frequency. Note that because of the sorting, higher frequency values are very close to zero. These can be eliminated.
c) We bring in a lossy step here to compress data - we round of the high frequency values close to zero, as equal to zero as their contribution is insignificant. Now we don’t need to

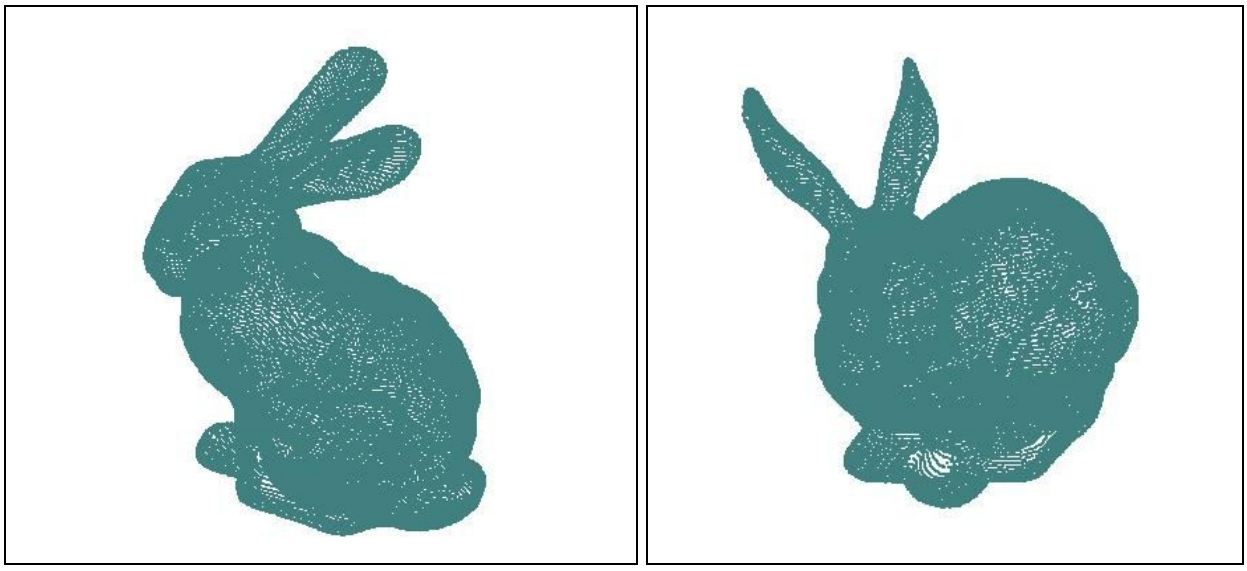
store these numbers, and we save 4-bytes for each rounding off - which can be decided based on the error tolerance.

d) The compression ratio decided by the user will tell the code how much data is to be eliminated and how much is to be retained.
e) Now these frequency components are stored as is (in a .mat file), and when we want to access it as a point cloud, we convert it back to the real domain, using an inverse discrete cosine transform (IDCT), and get those 64 coordinates back, with a small, tolerable error (depending on the number of components we rounded off) in each coordinate.
f) This is done for x,y,z and for each set of 64 points. The reasoning behind the number 64 is that DCT works the fastest when the number of elements it is operating on is a power of two. 64 is not random, but is based on observations after using different values on the entire set, where 64 gave optimal results.

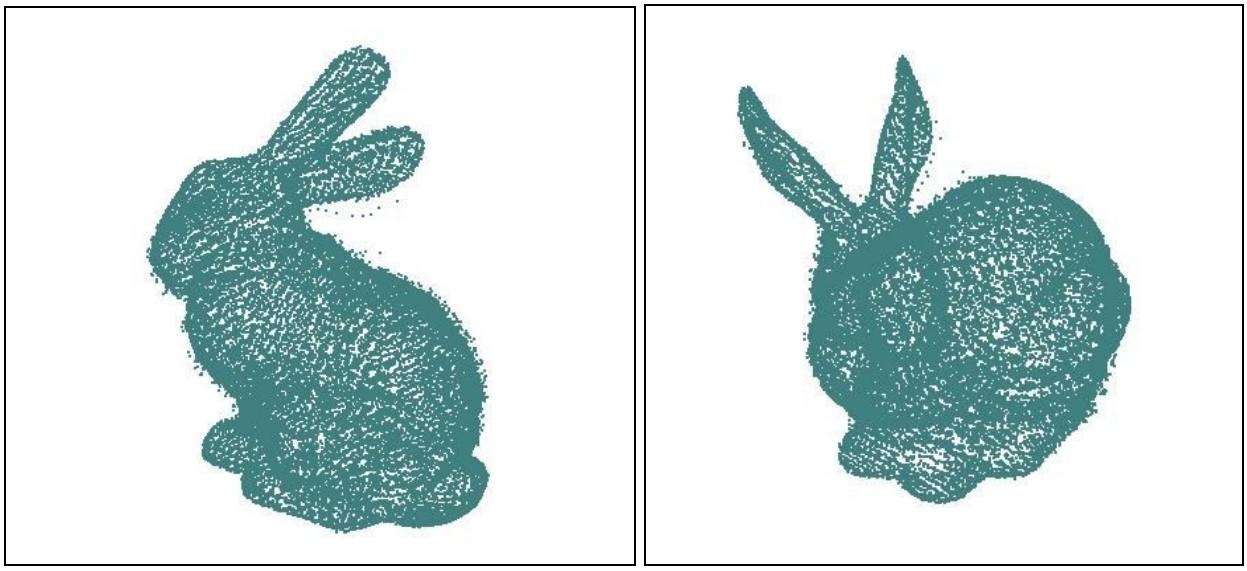
The idea came to us from the famous image format ‘jpeg’ which is almost universally used by the entire internet because of the compression levels it can achieve. JPEG format uses DCT, Quantization, Huffman Encoding and IDCT.

5.3. Results

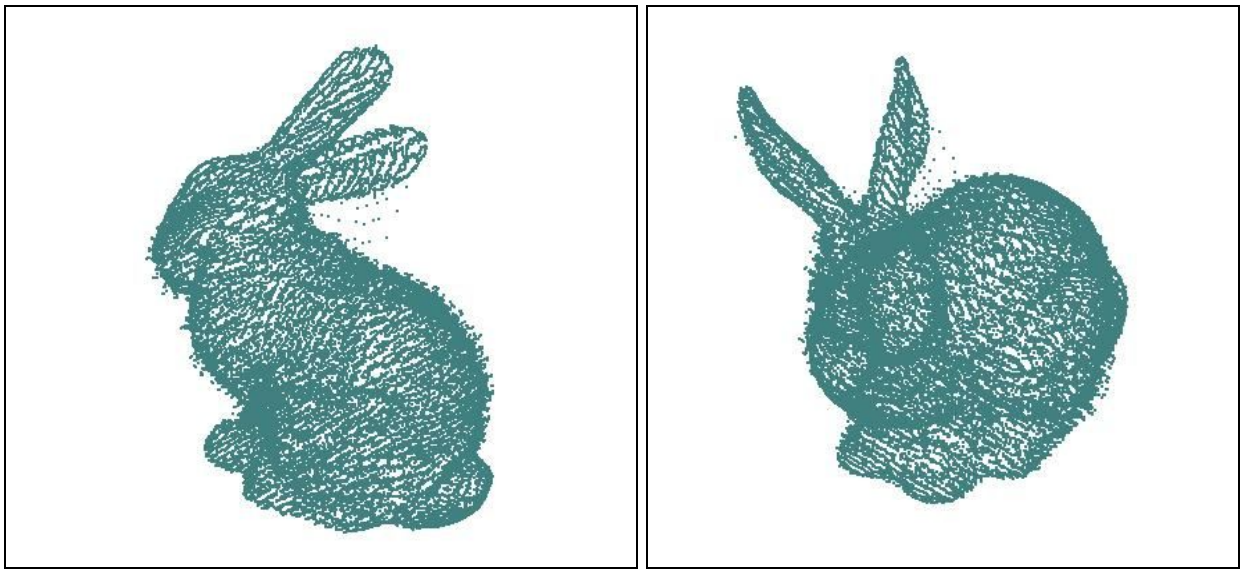
Compression Ratio (CR=1) (original point cloud):
File size: 398 KB



CR=2:
File size= 200KB



CR=4:
File size=100 KB



5.4. Future scope of work:

1. Implementing Huffman Encoding to compress the image further.
2. Using Lee Quantization for better rounding off techniques and reducing rounding off errors.
3. Use of wavelet transform rather than DCT for lesser loss of data for similar values of compression ratio.

6. References

1) Han H. et. al., Point cloud simplification with preserved edge based on normal vector, Optik, 2015.
2) BQ Shi, J Liang, Q Liu, Adaptive simplification of point cloud using k-means clustering, Computer-Aided Design, 2011
3) http://www.whydomath.org/node/wavlets/basicjpg.html (for JPEG compression)
4) http://graphics.stanford.edu/data/3Dscanrep/ (for sample point clouds)