Surface Reconstruction in 3-D

Sudarshan Ghonge 2011B5A7230H

Ria Agarwal 2012A7PS152H

Prakhar Gupta 2012A7PS059H

Under the supervision of

Dr. Tathagata Ray

Assistant Professor, Dept. of Computer Science

and

Dr. Chandu Parimi

Assistant Professor, Dept. of Civil Engineering

Submitted in partial fulfillment of the requirements of

CS F376 : Design Oriented Project

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

HYDERABAD CAMPUS

May 2, 2015

Contents

List of Figures

1 Introduction

Surface reconstruction is a method by which a mesh is generated given some point cloud dataset in any dimension. Imagine we just have some points in 3D space, and there is a way to generate a 3-D model which represents the cloud-set same as it can be visualized by a human brain. Today, in medical analysis, deformations and many other fields, we get a point set through Kinect and many other sources. To imagine this point cloud dataset as some meaningful presentation, we need to reconstruct the original surface from where the point set was obtained from. A point cloud is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by X, Y, and Z coordinates, and often are intended to represent the external surface of an object. Point clouds may be created by 3D scanners. These devices measure a large number of points on an object's surface and often output a point cloud as a data file. The point cloud represents the set of points that the device has measured. There are many techniques for converting a point cloud to a 3D surface. Some approaches, like Delaunay triangulation, NN Crust, ball pivoting, Cocone, while other approaches convert the point cloud into a volumetric distance field and reconstruct the implicit surface so defined through a marching cubes algorithm [**?**]. In the last part of our report, we discuss another technique known as the Poisson Surface Reconstruction which gives out an implicit form of the surface.

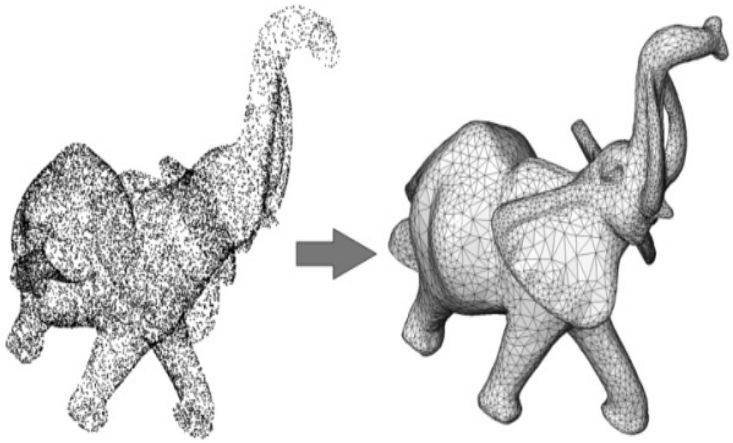

Figure 1: Image Source: http://doc.cgal.org/latest/Surface_reconstruction_points_3/

In the current project Simulation of Traumatic Brain Injury, the step is an important one to visualize the mesh formed collected using the point cloud dataset extracted during sudden brain injuries. The project aims at simulating the observed surface so that medical analysis of brain can be done at another level.

2 Problem of Surface Reconstruction

The Surface Reconstruction Problem takes as input a Point Cloud $\mathcal{P} \subset \mathbf{R^3}$ which are points lying on a closed Surface $\mathcal{S}$ and gives as output a mesh $\tau_{\mathcal{S}}$ which tessellates $\mathcal{S}$. The process involves two parts

1. Deriving an approximation of the Surface $\tilde{\mathcal{S}}$ of $\mathcal{S}$.
2. Using $\tilde{\mathcal{S}}$ to get a mesh $\tau_{\mathcal{S}}$.

To ensure that no parts of the surface go amiss, a certain minimum density of sampling is favourable. This idea is quantified by the sampling parameter $\epsilon(< 1)$. $\tilde{\mathcal{S}}$ should be Homeomorphic to and be Geometrically as close as possible to $\mathcal{S}$. Furthermore, $\tau_{\mathcal{S}}$ should meet the quality requirements of a *good* mesh so that it can be used in Finite Element Method (FEM) analysis. Our main focus in this project is to investigate Delauney Based methods to reconstruct the Surface. In order to understand these methods, we need to understand what are known as Voronoi Diagrams and Delauney Triangulation

2.1 Voronoi Diagram

The Voronoi Cell, V_p of a point $p \subset \mathcal{P}$ is defined as the set of points in $\mathbf{R^3}$ whose closest point from $\mathcal{P}$ is p. The collection of such Voronoi cells of each point belonging to $\mathcal{P}$ is called the Voronoi Diagram of $\mathcal{P}$. Voronoi Cells intersect at faces. Two Voronoi Cells intersect in facet (2-D face). Three Voronoi Cells intersect on an edge(1-D face) and four Voronoi Cells intersect at a point(0-D face). Voronoi cells can be definite or can extend into infinity.

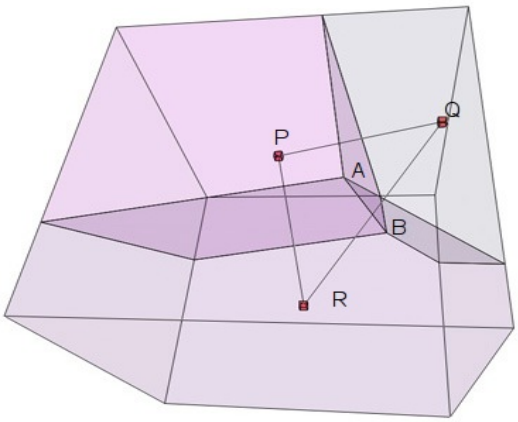

Figure 2: The above image shows the Voronoi Cells of the three points $P, Q, R \subset \mathcal{P}$. Each pair of the cells intersect at a phase. All 3 cells intersect at the Voronoi Edge AB. Image Source: http://www.wblut.com/blog/wp-content/2009/04/voronoi3db2.jpg

2.2 Delauney Tetrahedralization

The Dual Structure to Voronoi Diagram is the Delauney Tetrahedralization. It is defined as the Simplical Complex Convex hull of the points belonging to $\mathcal{T}$ where $T \subset \mathcal{P}$ such

that the intersection of the all the points in $\mathcal{T}$ is not a null set. Hence, the dual to a Voronoi Cell V_p is the point p itself. If the intersection V_p and V_q is a facet, then pq is its dual and is called a Delauney Edge. If the intersection of V_p, V_q, V_r intersect at a Voronoi edge, then the triangle pqr is its dual and is called a Delauney Triangle. If V_p, V_q, V_r V_s intersect at a point (also called as a Voronoi Vertex) then its dual is the tetrahedron $pqrs$ and is called a Delauney Tetrahedron.

It is these Delauney Tetrahedrons that will populate our space and describe the closed surface volume.

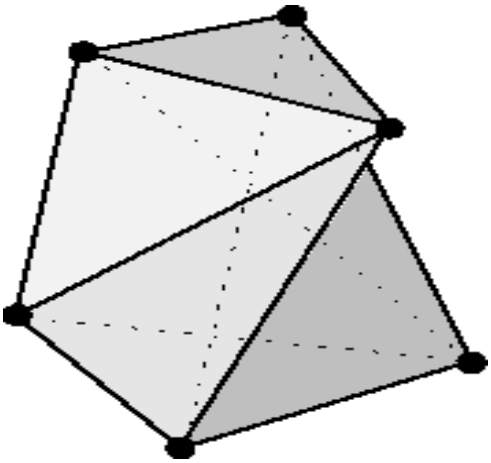

Figure 3: The above figure shows the delauney tetrahedrons for a small set of points. Image Source: http://www.iue.tuwien.ac.at/phd/fleischmann/img242.gif

3 Literature Survey

The diagram below explains the procedures followed by nearly all the surface reconstruction algorithms.

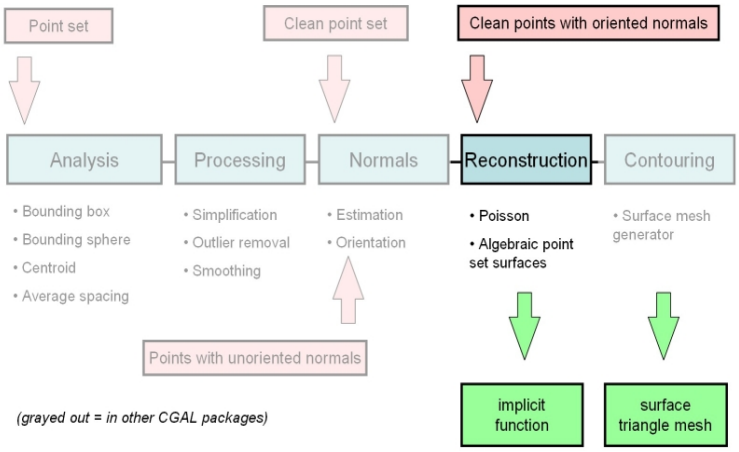

Figure 4: Surface Reconstruction Flowchart

The various algorithms that have been applied till date over or on the construction of a Surface in 3D/2D using point cloud data is as follows:

- Hoppe Algorithm (1992):
  It takes as input an unorganized set of points $x_1, x_2, x_3, x_4, ., x_n$ subset of $\mathbf{R^3}$ on or near an unknown manifold $\mathcal{M}$, and produces as output a simplicial surface that approximates $\mathcal{M}$. Neither the topology, the presence of boundaries, nor the geometry of $\mathcal{M}$ are assumed to be known in advance all are inferred automatically from the data. This problem naturally arises in a variety of practical situations such as range scanning an object from multiple view points, recovery of biological shapes from two-dimensional slices, and interactive surface sketching.

- NN Crust 2D(1998):
  The algorithm is explained in depth in Surface Reconstruction in 2D section

- Ball Pivoting Algorithm - BPA (1999): It is a non-voronoi based interpolating algorithm. It uses a rolling ball to construct triangles one by one and add them to the surface. A triangulated mesh is generation using point cloud. One disadvantage of the algorithm is that it produces holes if point set is not sufficiently sampled [**?**].

- Cocone (2000):
  The algorithm is explained in Surface Reconstruction in 3D section.

- Powercrust SuperCocone (2001):
  It is a voronoi based approach and can handle very large datasets. It partitions space into clusters using octree subdivision and then apply cocone on individual cluster [**?**].

- Tight Cocone (2003):
  It is voronoi based approach and a modification of Cocone. In this, A boundary algorithm is used as a preprocessor to detect under-sampled regions in input set [**?**].

- Robust Cocone (2004):
  It is a voronoi based approach and can handle noisy datasets. It reconstructs a surface interpolating through a subset of the points. Robust Cocone does not smooth data. It interpolates the data [**?**].

- Poisson (2006):
  Given a set of 3D points with oriented normals (denoted oriented points in the sequel) sampled on the boundary of a 3D solid, the Poisson Surface Reconstruction method solves for an approximate indicator function of the inferred solid, whose gradient best matches the input normals. The output scalar function, represented in an adaptive octree, is then iso-contoured using an adaptive marching cubes [**?**].

4 2D Curve Reconstruction and Meshing

To start with, we first went through the paper Size Optimal Delaunay Meshing Directly from Point Cloud Data. The paper has an upper hand over previous work as the curve is not approximated using the point cloud data as a whole at the start of the algorithm, but is constructed as and when required as the algorithm progresses. Since in the method of subsection 5.2, the whole curve is not triangulated and local triangulation is used for finding each intersection, the computational intensity is drastically reduced. This entertains the possibility that, depending on the input (maybe the curve is too densely sampled) some points are not at all used for approximation, potentially saving significant execution time. This of course is dependent on the value chosen for the thresholds.

Following is a short description of the Algorithm

1. Take as input the Point cloud $\mathcal{P}$ which is ϵ sampled.
2. Select an initial set of points S to start off with (4-6 points).
3. **Phase 1: Initialization**
   Delaunay triangulate S to get τ_S. Also find the dual Voronoi diagram.
4. **Phase 2: Topology Extraction**
   Check whether any Voronoi edge intersects the approximated curve more than once, or is tangential to it. If so, add the farthest intersection point of the Voronoi edge with the curve to the set S. Update τ_S. Go back to Phase 1.
5. Once above condition is satisfied, check whether each point in S has exactly 2 Delaunay edges incident on it. If not, the farthest intersection of the curve approximation and the Voronoi edges of the dual Voronoi cell of that Delaunay point from the point itself is added to the set S. Recalculate τ_S.
6. **Phase 3: Geometric Approximation**.
   Compare angles between the Restricted Delaunay edges on each Delaunay vertex to see whether it is greater than user specified value, α. If not, take dual Voronoi edge of longer Delaunay edge incident on that vertex, and add its intersection to S. Update τ_S. Go back to Phase 2.
7. **Phase 4: Mesh Refinement**.
   Mark all triangles outside the domain so that they are not processed in this phase.
8. For all triangles, find ratio of shortest edge to circumradius. If it is greater than user given value ρ_0, add the triangles circumcenter to S and update τ_S. Go back to Phase 1.

4.1 Calculation through method 5.1

The Delaunay triangulation of the entire point set is found. As and when an intersection point is required, use a threshold about the Voronoi edge, and run the NN-crust on the points that fall inside it. Then take the point of intersection of the edge with the result of the NN-Crust. Intersection calculation through method 5.2 No global Delaunay triangulation is used. Two thresholds are taken around the Voronoi edge, an inner one and

an outer one. The points that fall within the outer threshold are Delaunay triangulated. NN-crust is run on this triangulation, and the point of intersection is found by iterating through all the edges of this crust.

4.2 Calculation through method 5.2

No global Delaunay triangulation is used. Two thresholds are taken around the Voronoi edge, an inner one and an outer one. The points that fall within the outer threshold are Delaunay triangulated. NN-crust is run on this triangulation, and the point of intersection is found by iterating through all the edges of this crust.

4.3 NN Crust

Once the Delaunay triangulation is ready, the NN-Crust algorithm simply gets those edges from it which approximate the curve. These edges are the ones whose dual Voronoi edges intersect the curve exactly once. The algorithm is that for each Delaunay vertex; take the smallest edge incident on it. This is part of the NN-Crust. Inevitable gaps are then filled by taking the shortest edge which makes an angle of more than 90^o with the already selected one.

5 Cocone

Cocones are points in $\mathbf{R^3}$ that satisfy the following property: The set $C_p = y \in V_p : \angle_a(\vec{py}, \mathbf{v_p}) \geq \frac{3\pi}{8}$ is called the cocone of p, where V_p is the Voronoi cell of a point $p \subset \mathcal{P}.\mathbf{v_p}$ is the vector from p to the pole v. In plain english, the cocone is the complement of the a double cone that is clipped within V_p. This double cone has p as the apex the vector v_p as the axis and an opening angle of $3\pi/8$ with the axis.

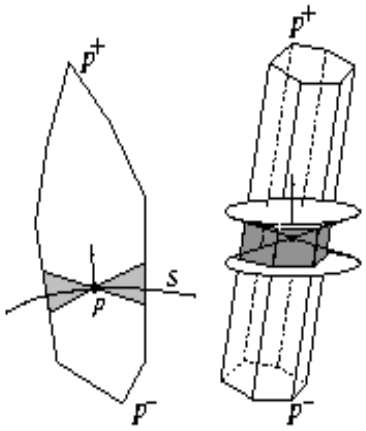

Figure 5: The above figure shows the Cocone of the Voronoi Cell. Image Source: http://web.cse.ohio-state.edu/~tamaldey/cocone.gif

The cocone of a Voronoi Cell helps compute the *Cocone Triangles*. Essentially, those Delauney Traingles who dual Voronoi edges intersect the cocone, can be shown to lie very close to the Surface that we wish to reconstruct [**?**].

The Algorithm is to compute an approximation of the surface using the concept of cocone is given below

1. Compute Voronoi Diagram $V_\mathcal{P}$ of Point cloud $\mathcal{P}$.
2. For each Voronoi edge $e \in V_\mathcal{P}$ check if this edge belongs to the cocone of all the Voronoi cells adjacent to it. If it is, add the dual Delauney Triangle to the set of Cocone Triangles
3. Extract the surface from the set of Cocone Triangles.

6 3-D Surface Reconstruction

The 3D curve reconstruction is an extension of its 2D equivalent described above, except for the fact that there is no local method to calculate the intersections of Voronoi edges with the surface. Hence, the entire point cloud is meshed and the Voronoi diagram is extracted. The Algorithm is as follows:

1. Mesh the point cloud and extract Voronoi diagram.
2. For each point p, find the point farthest from it in its Voronoi cell. We can this point v the *pole* of the Voronoi Cell.
3. Find all possible points lying on the edges of that Voronoi cell which make an angle of nearly $\pi/2$ with the ray to the pole at the point p. Mark any edge containing such a point.
4. Find all such edges that have been marked by all 3 Voronoi cells containing it.
5. Calculate the dual of the marked edges. This will be the crust of the surface.

Sharp edges are excluded from this simplical complex , these edges have incident triangles making angles greater than $3\pi/2$.

7 Implementation

7.1 DataSet

We have started looking for some datasets for implementation. We are considering datasets for simple objects currently like sphere. However, we went through a sample bunny dataset available freely on Stanford ply. https://graphics.stanford.edu/data/3Dscanrep/

7.2 Libraries

We are using C++ to implement the code. We are using CGAL library data structures to make the code optimized.

7.3 Code

The code is in a very initial stage and can be viewed in the following link. https://sudughonge@bitbucket.org/sudughonge/surface-reconstruction.git

8 Proposed Algorithm for Meshing

8.1 Introduction

Here, we use the point cloud data P and the reconstructed surface, let it be R. Similar to the 2D counterpart, there are four phases to mesh the enclosed volume.

8.2 Phase One

Select some seed points S from the point cloud data for starting the algorithm. These points are randomly selected from the set of points in P. A Delaunay tetrahedralization $touofS)oftheseedpointsSiscreated.Wearenotusinganylocalisationintheprocessandmakeuseoftheglob$

8.3 Phase Two

We must keep in mind that R is based on an $\epsilon-sampleofP, andRisthushomeomorphictotheactualsurfa$

8.4 Intersection Calculation

For each Voronoi edge e, a ray r is generated. Each triangle in the reconstructed surface R, is checked for intersection with this ray. All intersections are noted, and the ones that happen outside the limit of the line e are rejected. The remaining intersections are kept. If there are 0 or 1 intersections for that Voronoi edge, nothing is done and the next Voronoi edge is considered. If more than one intersection exists, then all of the intersections are calculated and the farthest one is added to the set S.

8.5 Phase 3

To make a mesh that closely resembles the original surface, we must make sure that the angle made between adjacent triangles in the mesh is large enough. If this angle is less than a given threshold value $\alpha atanypoint, thenwemustrectifyit.Thisisdonebythefollowingmethod.Thetriangl$

9 Poisson Surface Reconstruction

The algorithm follows 3 basic steps:

9.1 Finding Out Normals For Boundary Point-Set

1. Find the tangential planes for approximating the local surface: To achieve this the centroid of every vertex is computed as the average of all k-nearest neighbors. Then the corresponding normal is computed by its eigenvector.
2. Building the Riemannian graph:
The Riemannian graph is needed because it creates a structure where for every vertex the edges to his k-nearest neighbors are included what we need for the next step.
3. Using the minimum spanning tree algorithm from Kruskal on this Riemannian graph to weight the normals what results in different length for each of them.

The algorithm computes the normals as a weighted product over the k-nearest neighbors. So we get a very smooth model. In picture 1 we will try to illustrate this, where the red marked normal is computed using the six nearest neighbors of it. Assuming that the scanned points form the surface of the scanned object, the isosurface of the model is approximated from the normalfield that is in relation to the gradient of the indicator function describing the isosurface. This problem is considered as the solution of a poisson problem.

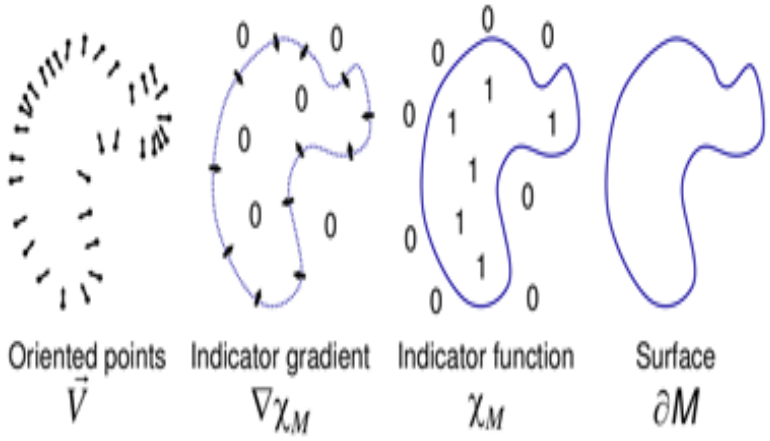

Figure 6: Extracting the ISO surface

At first the points with the oriented normals are taken and the gradient of the indicator function is approximated through the normalfield. After that the indicator function, which is zero everywhere except near the surface, is derived from this gradient and the surface of the object can be extracted.

9.2 Several Parameters for Poisson Surface Reconstruction

Depth: tree-depth that is used for the reconstruction. default: 8
SolverDivide: specifies depth at which a block Gauss-Seidel solver is used to solve Laplacian equation. default: 8
IsoDivide: specifies the depth at which a block iso-surface extractor should be used to extract the iso-surface. default: 8
SamplesPerNode: specifies the minimum number of sample points that should fall within an octree node as the octree construction is adapted to sampling density. For noise-free data: [1.0, 5.0], noisy data: [15.0, 20.0]
Scale: ratio between the diameter of the cube used for reconstruction and the diameter of the samples bounding cube. default: 1.25
Offset: an hacked offset value. if set to 1 there is no offset.
Confidence: enabling this flag tells the reconstructor to use the size of the normals as confidence information. When the flag is not enabled, all normals are normalized to have unit-length prior to reconstruction.
Especially the parameters depth and samples per node have a great influence on the generated mesh.

The higher the value for the octree-depth is chosen the more detailed results you will get. This is because the deeper the marching cubes algorithm goes the finer gets the granularity of the voxelgrid. On the other hand with noisy data (like our scanned point clouds) you keep vertices in the generated mesh that are outliners but the algorithm doesnt detect them as such. So a low value (maybe between 5 and 7) provides a smoothing but you will lose detail. The higher the depth-value the higher is the resulting amount of vertices of the generated mesh. The samples per node parameter defines how many points the marching cubes algorithm puts into one node of the resulting octree. A high value like 10 means that the algorithm takes 10 points and puts them into the node of the octree. If you have noisy data a high sample per node value provides a smoothing with loss of detail while a low value (between 1.0 and 5.0) keeps the detail level high. A high value reduces the resulting count of vertices while a low value remains them high.

9.3 Comparison between various Poisson Parameters

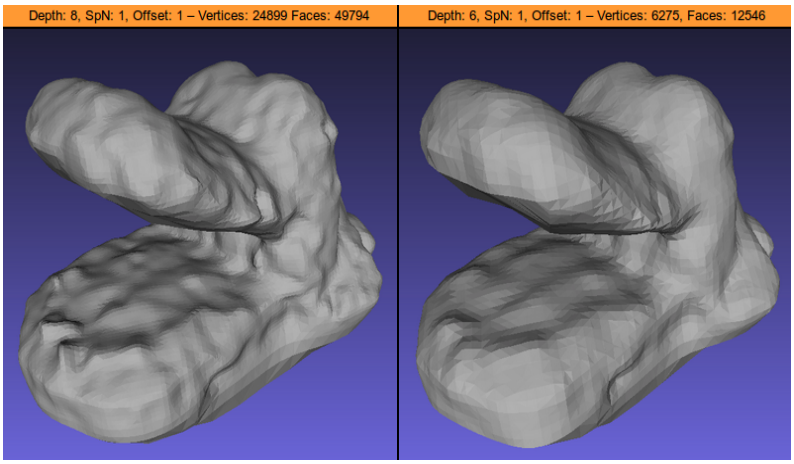

Figure 7: A fast computed result with an acceptable level of detail and a nicely shaped smooth surface is produced by using a depth value of only 6.

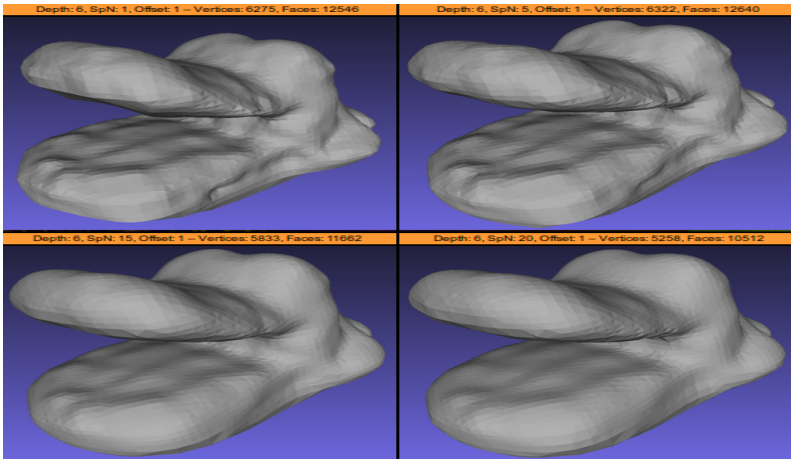

Figure 8: Now by increasing the value for the samples per node (with a constant depth value of 6 and offset 1) you can see that the surface gets even smoother

9.4 Experimentation

We ran the Poisson implementation of the algorithm and got the following outputs over POISSON DATABASE.We used Paraview to visualze the output ply file.
DataBase 1: Bunny Dataset from Stanford ply site Output file generated:

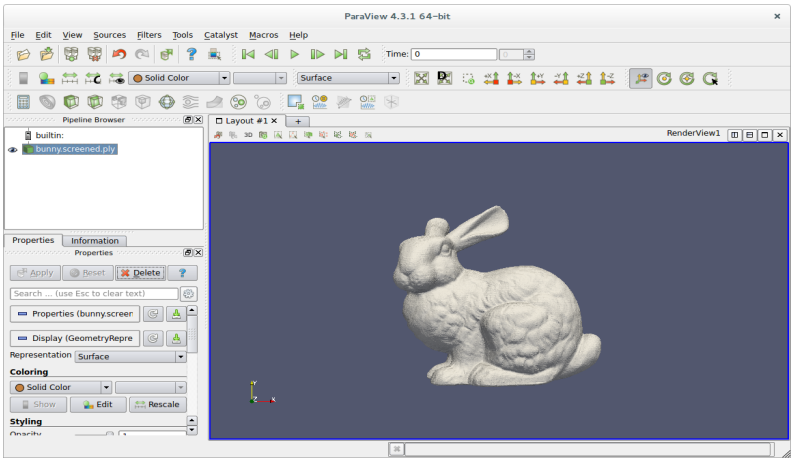

Figure 9: Time taken for output generation : 35.45 Seconds

DataBase 2: Horse Dataset from Stanford ply site Output file generated:

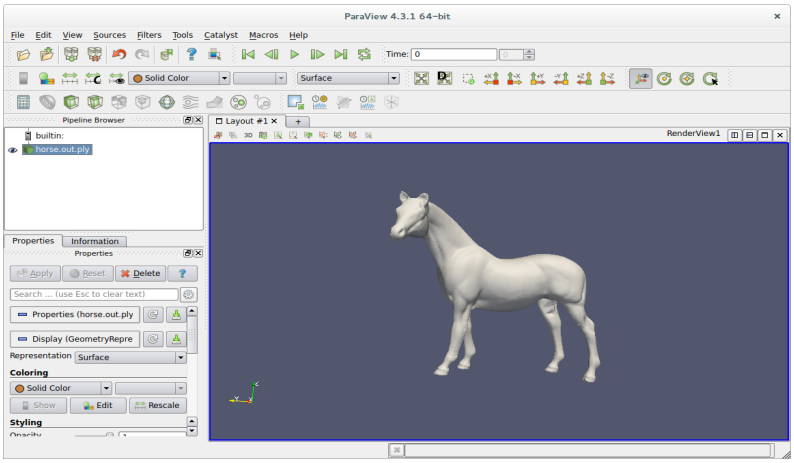

Figure 10: Time taken for output generation : 22.7 Seconds

Specifications of Machine Tested : Processor : Intel Core i3-2330M 2.2 Ghz RAM : 2GB Operating System : 64 Bit Linux Ubuntu Gnome 14.04 Graphic Card : Nvidia GeForce 630M

10 Conclusion

We have gone through the Curve Reconstruction implementation done in 2-D called the **PCMesh** algorithm which locally computes the curve using the NNCrust algorithm and adds the information provided by into the mesh by taking intersections with the Voronoi edge. We wish to extend this approach into 3-D and by borrowing concepts from the Cocone and localize our above described 3-D reconstruction method. We have also studied the Poisson Reconstruction Algorithm and it's inbuilt implementation as an alternative approach to the Delaunay based method.