CS4089 Project Report

Surface Reconstruction

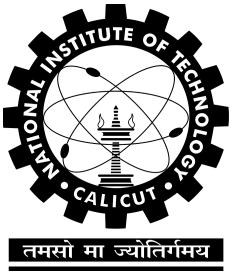
Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Submitted by

Roll No	Names of Students
B130055CS	Ashok Dhungana
B130056CS	Prabhav Adhikari
B130487CS	Suraj Yadav

Under the guidance of
Subhasree M and **K Muralikrishnan**



Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India – 673 601

Winter Semester 2017

Department of Computer Science and Engineering

National Institute of Technology Calicut

Certificate

This is to certify that this is a bonafide record of the project presented by the students whose names are given below during 2016-17 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

Roll No	Names of Students
B130055CS	Ashok Dhungana
B130056CS	Prabhav Adhikari
B130487CS	Suraj Yadav

Subhasree M
(Project Guide)

Anu Mary Chacko
(Course Coordinator)

Date: 27^{th} June, 2017

Abstract

Given a finite sampling $P \subset R^3$ of an unknown surface S, Surface Reconstruction is concerned with the calculation of a model of S from P. In this report we present a non-parametric algorithm that generates a manifold triangular mesh S from a set of unorganized points and compare it with various existing algorithms for Surface Reconstruction. We base our algorithm on the observation that the Euclidean Minimum Spanning Tree provides a frame to the triangular mesh that approximates the surface. To this frame, we add triangles to approximate the surface. The running time comparison with the current state of art algorithms shows that our algorithm is at par in terms of speed while generating comparable models.

Contents

Chapter 1
Introduction

1.1 Problem Definition

Given a finite sampling $P \subset \mathbb{R}^3$ of an unknown surface F, Surface Reconstruction is concerned with the calculation of a model S of F from P. S is represented by triangular mesh.

1.2 Motivation

Surface Reconstruction has a horizon of applications in Computer Aided Designing, Computer Graphics, Computer Animation, Computer Vision, Medical Imaging etc. Most cases of reconstruction involve sampling of points using scanners on a model for input data. Depending on the scanner, we have a wide variety of input types like with/without normal information, with/without color information etc. The large number of possibilities of an approximate surface for a given input Point set makes the problem ill-posed. Further to be able to reconstruct a model without any other details other than the co-ordinates of the points and be able to capture the features of the original model like its concavity, convexivity, sharp edges, holes, smoothness etc is a challenge in itself.

1.3 Preliminaries

1.3.1 Convex Hull

Given a set of points P, convex hull is defined as a closed convex polygon that includes all the points in P with the least area.

1.3.2 Triangulation

Triangulation of a set of points P is the subdivision of the convex hull of the points into simplices such that any two simplices intersect in a common face of any dimension or not at all and such that the set of vertices of the simplices are contained in P. Every point set has a triangulation.

1.3.3 Delaunay Triangulation

Delaunay triangulation of P is a triangulation of P in which every triangle is Delaunay. A triangle is delaunay if its vertices are in P and its open circumdisk is empty. For a 2D point set, there always exists a delaunay triangulation, and is unique if and only if no four points in P lie on a common empty circle. However, these conditions are not guaranteed for higher dimensions. s

1.3.4 Euclidean minimum spanning tree

The Euclidean minimum spanning tree or EMST is a minimum spanning tree of a set of n points in the plane, where the weight of the edge between each pair of points is the Euclidean distance between those two points.

1.3.5 Medial Axis

The Medial Axis (MA) of a surface bounded by curve C, is defined as the locus of the centers of circles (in $\mathbb{R}^2$), spheres (in $\mathbb{R}^3$), that are tangent to curve C in two or more points, where all such circles (or spheres) are contained in the surface. The medial axis together with the associated radius of the inscribed discs is called the Medial Axis Transform (MAT).

1.3.6 Closed Neighbourhood

In a graph G, closed neighbourhood of a vertex $v \in G$, denoted by $N_G[v]$, is the set of vertices adjacent to v in G along with v.
$N_G[v] = \{u | u \in G, u = v \vee (u, v) \in E(G)\}$

1.4 Literature Survey

There are several algorithms that tackle this problem. The Powercrust[1] method on 3D point set produces a surface mesh and an approximate medial axis. It approaches by first approximating the medial axis transform (MAT) of the object then using an inverse transform to produce the surface representation from the MAT which is watertight.

The Ball Pivoting Algorithm(BPA)[2] works in this way: Three points form a triangle if a ball of a user-specified radius ρ touches them without containing any other point. Starting with a seed triangle, the ball revolves around the edge while keeping in contact with the edges endpoints until it touches another point, forming another triangle. The process continues until all reachable edges have been tried, and then starts from another seed triangle, until all points have been considered.

The Robust Cocone [3] algorithm handles noisy sample points. This algorithm is further extended to compute homeomorphic surface interpolating the subset of out sample points residing on outer or inner bid Delaunay balls.

Poisson surface reconstruction [4] creates watertight surfaces from oriented point sets. The points are incorporate as interpolation constraints, which allows the problem to be interpreted as a generalization of the underlying mathematical framework to a screened Poisson equation.

Chapter 2

Work Done

2.1 Algorithm

The algorithm initializes a Graph G to the Minimum Spanning Tree obtained from the Delaunay Triangulation (DT) of the input Point Set P. We then add edges to G, essentially adding more triangles to G, eventually creating a Triangular Mesh approximating the original Surface. The details for the process of adding triangles is discussed in the following sections.

2.1.1 Definitions

Definition 2.1. Surface Graph
Surface graph G represents the triangular mesh approximating the surface. S denotes the set of triangles in G. G keeps changing during the course of the algorithm.

Definition 2.2. Score
Score of a triangle is defined as the minimum angle of the triangle.

Definition 2.3. Face
A face is a triple of the form (score, edge, opposite vertex) for a triangle.

Definition 2.4. Hole Graph
$H = \{e \in G \mid facedegree(e) \leq 1$, where $facedegree(e)$ denotes the number of faces sharing the edge e in G. H is computed afresh in the beginning of each iteration of the algorithm, but remains unchanged during the iteration.

Definition 2.5. Overlapping condition
An edge is said to overlap a triangle if the projection of the edge on the plane containing the triangle is inside the triangle.

Definition 2.6. Maximum Neighbour Distance
For a vertex $v \in G$, Maximum Neighbour Distance is the maximum length of the edge adjacent to v in G. This value is returned by $maxNbrDist(G, v)$

Definition 2.7. Stitching Edges
Stitching edges are the edges in $DT \setminus G$ that connects two non-adjacent vertices belonging to a single connected component of the *Hole Graph* H (Def 2.4), satisfying the following additional conditions:

- Length of the edge is less than $2 \times maxNbrDist$ (Def 2.6) of both the vertices of given edge in G.
- The edge does not satisfy the *Overlapping Condition* (Def 2.5) with faces in G that are adjacent to the vertices of the edge.

Definition 2.8. Possible Faces
Given an edge (u,v) the possible faces are each (u,v,w) for w $\in N_G[u] \cup N_G[v]$

Definition 2.9. PQ
PQ is a max heap of faces that uses *score* (Def 2.5) as key

Definition 2.10. Edge Condition
A face $f \in DT \setminus G$ satisfies *Edge Condition* if the number of edges of f whose length is less than the length of the longest edge in the MST is greater than or equal to edge condition number denoted by the variable *edgeCondition*, maintained by the algorithm.

Definition 2.11. Face Replacement Condition
A given face $f \in DT \setminus G$ satisfies the face replacement condition if there exist faces f_a and f_b in G that share an edge with f such that $angle(f, f_b) < angle(f_a, f_b)$. Then f can replace f_a, and f_a is called a **replacable face**. Similarly, if $angle(f, f_a) < angle(f_a, f_b)$ then f can replace f_b, and f_b will be a replacable face.

Definition 2.12. Face Acceptance Condition(FAC)
A face $f \in DT \setminus G$ satisfies the face acceptance condition if

- Either each edge e of f has $facedegree(e) < 2$ or f satisfies the *Face Replacement Condition*. Here $facedegree(e)$ denotes the number of faces sharing the edge e in G.
- Edges of f should not overlap (Def 2.5) with the faces adjacent to the neighbouring vertices of the vertices in f other than the replacable faces
- f satisfies the *Edge Condition* (Def 2.10)

2.1.2 Description of Algorithm

The algorithm is implemented using the function *reconstruct*() that takes a point set P as an input and returns a set S of triangles which approximates the surface.

Initializations

- $DT \leftarrow$ Delaunay Triangulation of P
- $G \leftarrow$ MST of DT
- $PQ \leftarrow \phi$
- $edgeCondition \leftarrow 2$

Loop Invariants

- For each edge e $\in G$, $facedegree(e) \leq 2$ i.e., an edge is shared by at most 2 faces
- Each face $f \in G$, satisfies *Edge Condition* (Def 2.10)

Termination Condition

- G does not change in an iteration

In each iteration of the loop we begin by getting *Stitching Edges* (Def 2.7) by calling $getStitchEdges()$. This function takes G and DT as input, computes a *Hole Graph* H (Def 2.4) from G and returns the set *Stitching Edges*. The set *Stitching Edges* is passed to the function $getFacesFromEdges()$ which outputs the set *Possible Faces* (Def 2.8). This is a potential set of faces that can be added to G and hence are inserted into the *Priority Queue* PQ (Def 2.9) . The function $addFacesToSurface()$ extracts faces one by one from PQ and checks whether each face satisfies the *Face Acceptance Condition* (Def 2.12). If a face satisfies the condition then it is added to G. For each face f thus added, the edge set of f is passed to $getFaceFromEdges()$ which identifies new faces in $DT \setminus G$ to be added to PQ. If a face extracted from PQ fails only the *Edge Condition* (Def 2.10) in the *Face Acceptance Condition* then such faces are kept aside and added to PQ of the next iteration. This completes an iteration of our algorithm. The iterations continue until an iteration fails to change G.

The *postProcess()* step computes *leftOverHoles*, defined as
$\{e \in G \mid facedegree(e) = 1\}$. Subsequently for each simple cycle in *leftOverHoles*, faces are add to G with an algorithm similar to ear-clipping algorithm [5].

Finally the algorithm extracts the set of triangles S from G and returns S.

Algorithm 1: Surface Reconstruction using Euclidean MST

Function *reconstruct(P)*
  $DT \leftarrow$ Delaunay triangulation of P
  $G \leftarrow$ MST of DT
  $PQ \leftarrow \phi$
  $edgeCondition \leftarrow 2$
  repeat
    $G` \leftarrow G$
    $stitchEdges \leftarrow getStitchEdges(G, DT)$
    $PQ \leftarrow PQ \cup getFacesFromEdges(G, sticthEdges)$
    $(G, PQ) \leftarrow addFacesToSurface(G, PQ, edgeCondition)$
    $edgeCondition \leftarrow edgeCondition - 1$
  until $G` = G$;
  $PostProcess(G)$
  $S \leftarrow$ Extract triangles from G
  return S

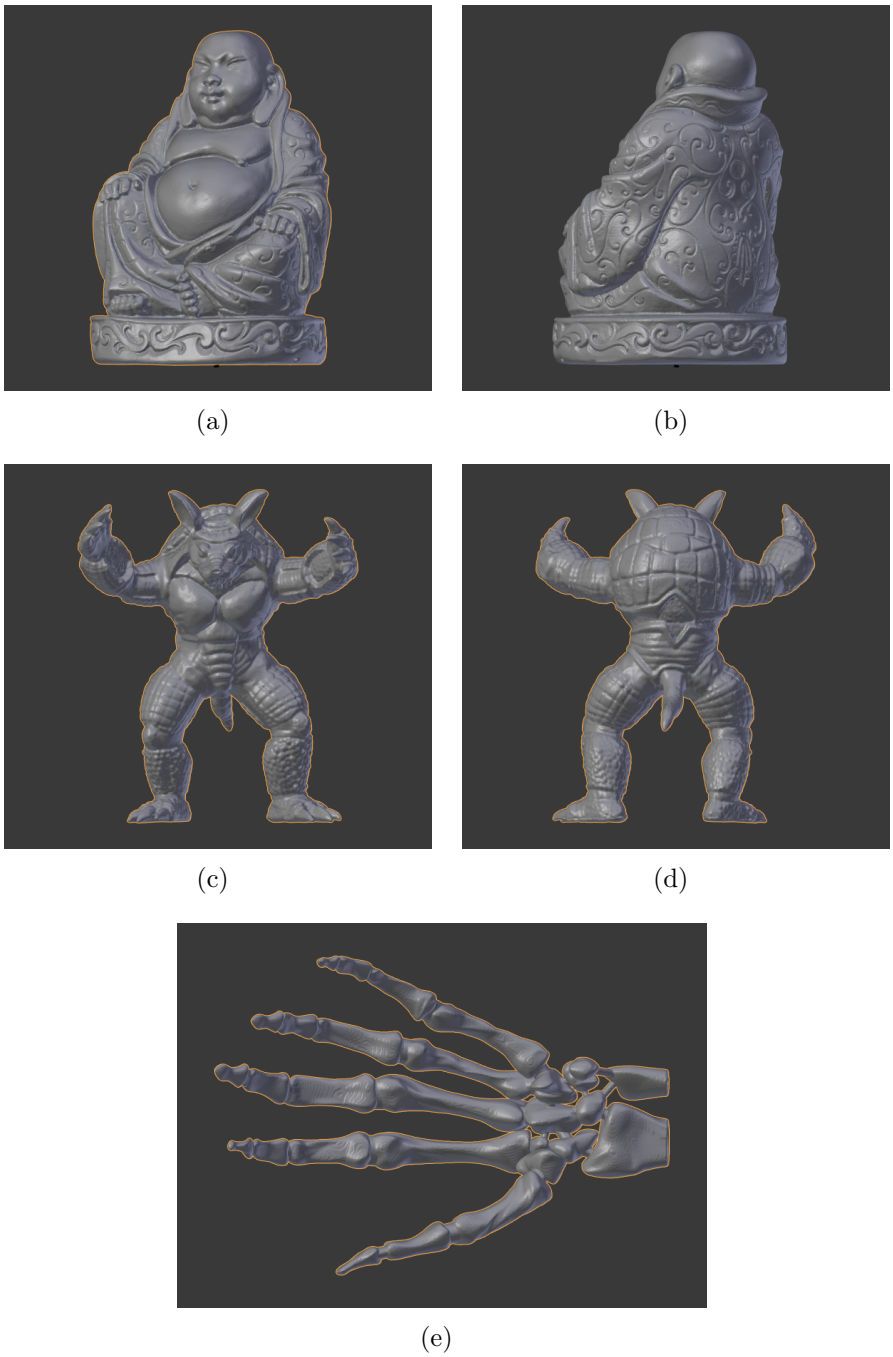
Function *edgeOverlap(e,G)*
  return *true* if e has projection on any of the faces adjacent to vertices of e in G
  return $false$ otherwise

Function *getStitchEdges(G, DT)*
  $H \leftarrow \{(u, v) \in G \mid faceDegree(u, v) \leq 1\}$
  $stitchEdges \leftarrow \phi$
  Initialize $u.covered = false \ \forall \ u \in G$
  foreach $e = (u, v) \in DT \setminus (H \cup G)$ *(sorted by length)* **do**
    if $u, v \in$ *same component in* H ***AND***
     $distance(u, v) \leq 2 \times \min(maxNbrDist(u), maxNbrDist(v))$ ***AND***
     $\neg edgeOverlap(e, G)$ ***AND*** *(*$\neg u.covered$ ***OR*** $\neg v.covered$*)* **then**
      $sticthEdges \leftarrow sticthEdges \cup \{e\}$
      $u.covered \leftarrow True$
      $v.covered \leftarrow True$
    end
  end
  return sticthEdges

```
Function addFaceToSurface(G,PQ,edgeCondition)
    tempPQ ← φ
    while PQ ≠ φ do
        f ← PQ.extractMax()
        if f satisfies Face Acceptance Condition then
            G ← G \ replacable(f) // (Def 2.11)
            G ← G ∪ {f}
            PQ ← PQ ∪ getFacesFromEdges(G, f)
        end
        if f only fails Edge Condition then
            tempPQ ← tempPQ ∪ {f}
        end
    end
    return (G, tempPQ)


Function replacable(f)
    return {f_a | f_a is a face in G and f can replace (Def 2.11) f_a}


Function getFacesFromEdges(G, edgeSet)
    faces ← φ
    foreach e = (u, v) ∈ edgeSet do
        nbrSet ← N_G[u] ∪ N_G[u] \ {u, v}
        foreach w ∈ nbrSet do
            faces ← faces ∪ {scoreof{u, v, w}, e, w}
        end
    end
    return faces


Function PostProcess(G)
    H ← {(u, v) ∈ G | faceDegree(u, v) = 1}
    foreach simple cycle ∈ H do
        repeat
            edge_0, edge_1 ← pair of adjacent Edges in cycle with min angle between
              them
            G ← G ∪ {edge_0, edge_1}
            cycle ← cycle \ {edge_0 ∩ edge_1}
        until |cycle| ≤ 3;
        G ← G ∪ cycle
    end
```

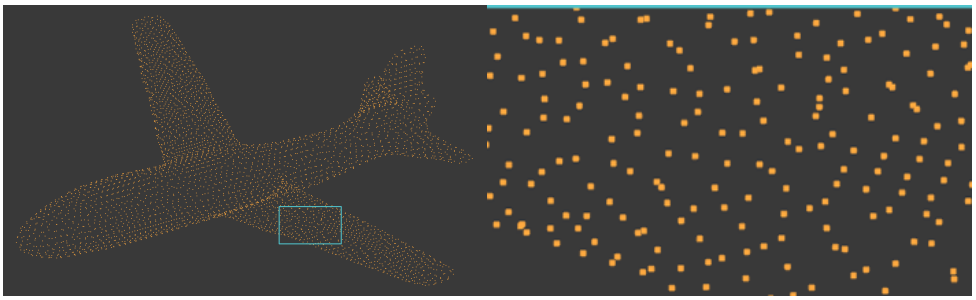
Chapter 3

Illustrations

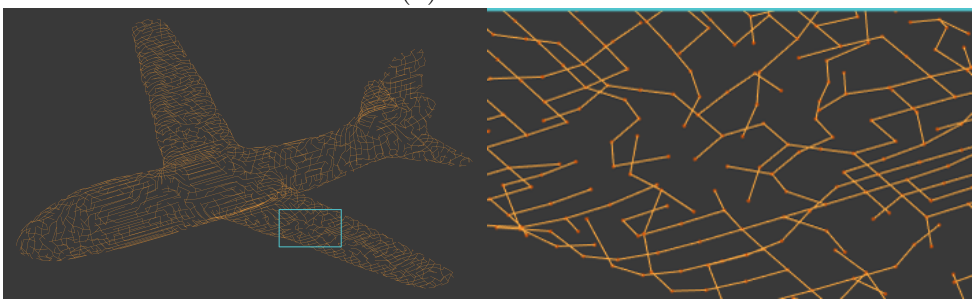


(a) (b) (c) (d) (e)

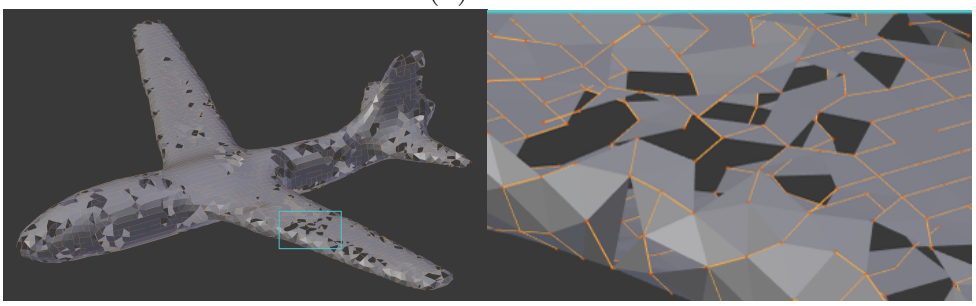
Figure 3.1: (a),(b) Reconstructed Model with 757490 vertices and (c),(d) with 172974 vertices. (e) Rendered Image of a reconstructed surface with 327323 vertices.



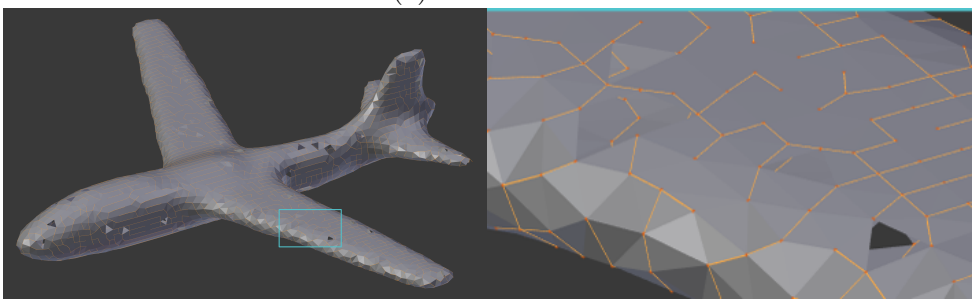
(a) Point Set



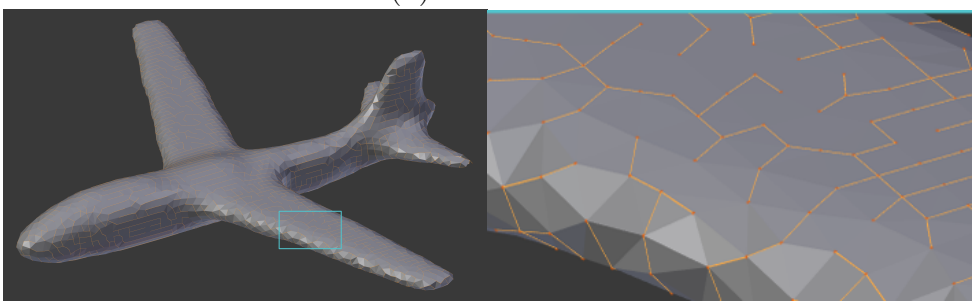
(b) MST



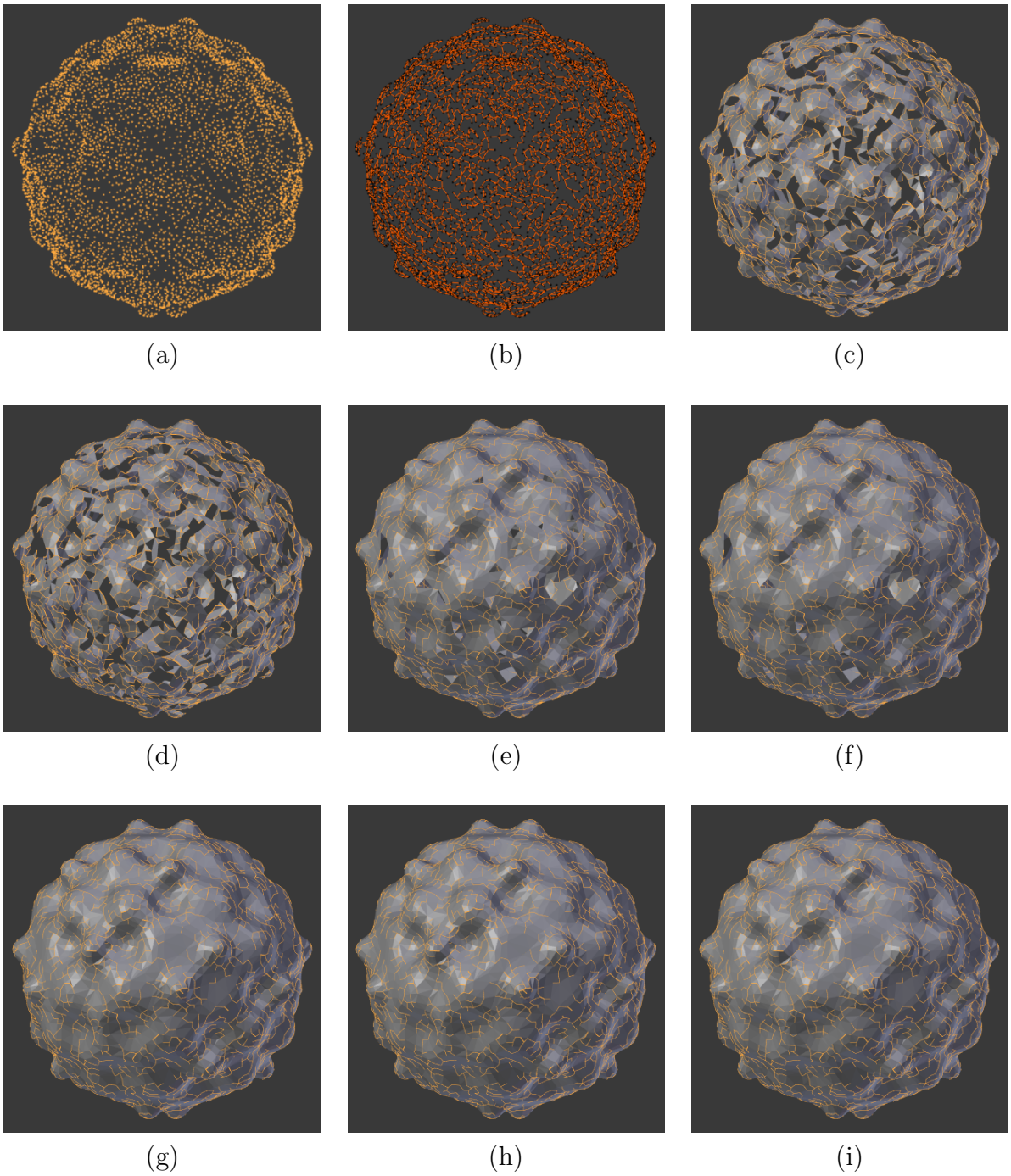
(c) Iteration 1



(d) Iteration 3



(e) After Post Processing



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 3.2: Surface during Iteration of bumpy Sphere.

Chapter 4

Comparisons

The paper "A Benchmark for Surface Reconstruction" [6], provides a class of shapes containing many interesting characteristics such as multiple scales of detail, nontrivial topology, and sharp features.

We compared our algorithm with Screened Poisson Software[7], Powercrust[8], Ball Pivoting Algorithm [9] and RobustCocone[10].

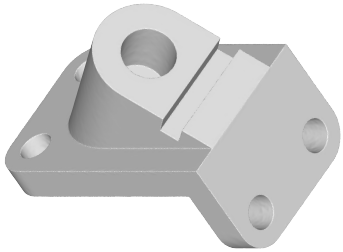
For comparison, we considered the running time of the algorithm.

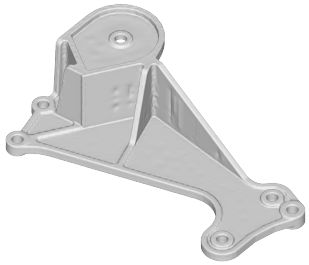
Similar to L^2 [11], we use a L^3 error norm for comparing two models.

$$L^3error = \frac{volume((O - S) \cup (S - O))}{volume(O)}$$

where O is the original Surface and S is the model to be compared with. For L^3 error calculation both O and S should be watertight. This norm reflects the similarity between two models, with 0 denoting that the two models are identical. For our purpose, where we lack an original surface , we have set our reconstruction as the original and compared others with it.

The following observations were made. 0* denotes the case where the volumetric difference between two models was very small.

Anchor (263286 points)	Algorithm	Runtime	L^3error
	Our Algorithm	43.412s	0
	Screened Poisson	15.536s	0.00130963
	PowerCrust	2m 24.156s	Mem Exceeded
	Robust Cocone	3m 23.892s	0.0000970
	BPA	8.031s	0*

Daratech (245836 points)	Algorithm	Runtime	L^3error
	Our Algorithm	49.316s	0
	Screened Poisson	29.023s	0.00190252
	PowerCrust	2m 3.79s	Mem Exceeded
	Robust Cocone	3m 44.312s	0.00026137
	BPA	9.053s	0*

Dancing Children (468022 points)	Algorithm	Runtime	$L^3 error$
	Our Algorithm	44.648s	0
	Screened Poisson	47.642s	Mem Exceeded
	PowerCrust	3m 36.024s	Mem Exceeded
	Robust Cocone	8m 16.432s	0.0000159339
	BPA	27.130s	Mem Exceeded

Gargoyle (481358 points)	Algorithm	Runtime	$L^3 error$
	Our Algorithm	1m 45.236s	0
	Screened Poisson	1m 6.947s	Mem Exceeded
	PowerCrust	4m 46.892s	Mem Exceeded
	Robust Cocone	8m 16.808s	2.12957e-05
	BPA	23.545s	Mem Exceeded

Lord Quasimoto (350000 points)	Algorithm	Runtime	$L^3 error$
	Our Algorithm	1m 11.784s	0
	Screened Poisson	29.944s	0.00033601
	PowerCrust	3m 21.192s	Mem Exceeded
	Robust Cocone	5m 51.044	0.00001050
	BPA	11.818s	5.18534e-06

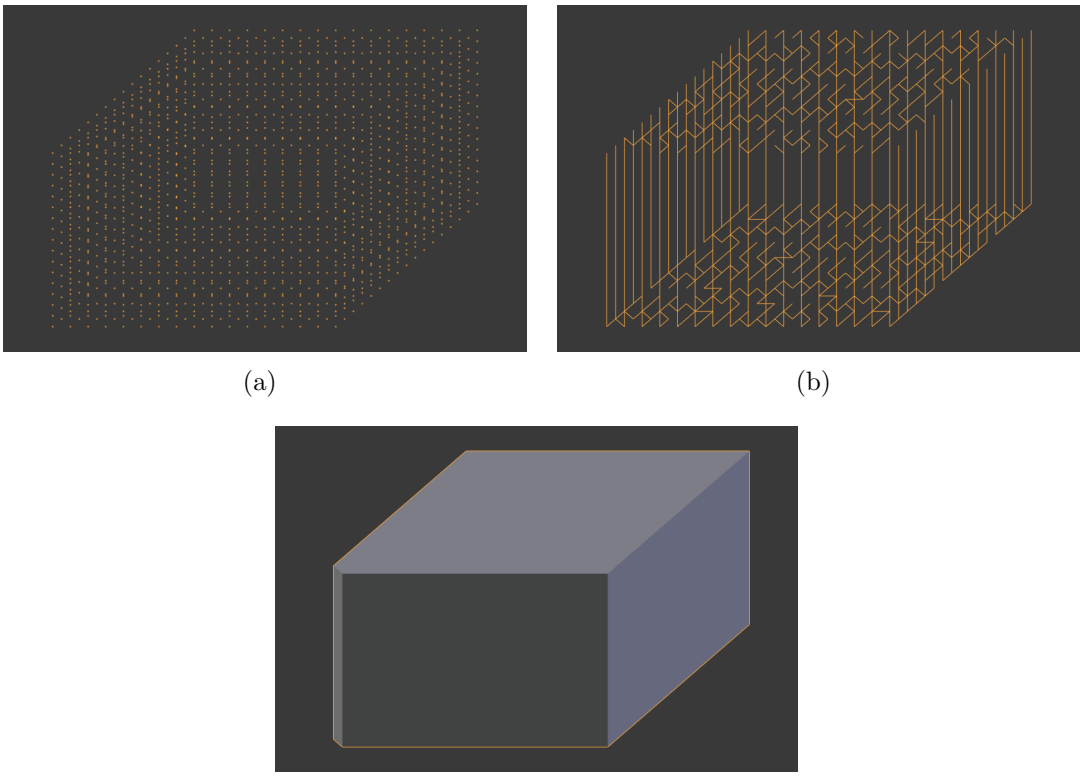
Chapter 5

Limitations and Future Work

Our algorithm, while being non-parametric, has been able to efficiently approximate a manifold triangular mesh. Although this has worked on our advantage in terms of running time in comparison to state of art algorithms like *Screen Poisson* and *Power Crust*, we observed that our reconstruction was inconsistent to original surface under certain cases.

As our algorithm is based on the EMST, any deviation in the EMST will affect the quality of reconstruction.

For eg. when the EMST overlooks certain features of the geometry, and thus they are not reflected in the reconstruction. In the case below, if the EMST constructed would have had horizontal lines along the sides, instead of the vertical ones, the sharp edge on the left would have been present.



(a) (b)

(c)

Figure 5.1: Rhombus (a) Point Set (b) MST (c) Reconstruction

Another is the case of noisy input set. The EMST constructed will deviate from the surface, as it covers the noisy points and thus the reconstruction will not be true in those regions.

Certain further improvements in the algorithm are possible.

- Considering the above case of EMST overlooking certain features, the EMST can be generated such that, if choice is present between several edges of similar length, we choose an edge which is not collinear with existing edges in EMST. This will force the EMST to from zigzag lines, instead of straight lines.
- In addition to the input point set, if we take the information about vertex normal, we might be able to form better EMST which will improve the reconstruction.

Acknowledgments

We would like to extend our sincere gratitude to our project supervisors Dr. Subhashree M and Dr. K Muralikrishnan for their valuable time and expertise. We were introduced to this interesting problem in Computational Geometry by Dr. Subhashree M and since then we had a great learning experience which eventually helped us design and implement a working algorithm from scratch. The Department of Computer Science and Engineering at National Institute of Technology Calicut supported us in infrastructure by providing a high-end Workstation(Intel Xeon(R) CPU ES-2630 v3, 16cores@2.40GHz, 16GB memory). We are also grateful to the availability of online research materials made by Digital Library at National Institute of Technology Calicut.

Ashok Dhungana
Prabhav Adhikari
Suraj Yadav
2016-17
National Institute of Technology Calicut

References

[1] N. Amenta, S. Choi, R. K. Kolluri, The power crust, in: Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications, SMA '01, ACM, New York, NY, USA, 2001, pp. 249–266. doi:10.1145/376957.376986.
URL http://doi.acm.org/10.1145/376957.376986

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, The ball-pivoting algorithm for surface reconstruction, IEEE Transactions on Visualization and Computer Graphics 5 (4) (1999) 349–359. doi:10.1109/2945.817351.

[3] T. K. Dey, S. Goswami, Provable surface reconstruction from noisy samples, in: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04, ACM, New York, NY, USA, 2004, pp. 330–339. doi:10.1145/997817.997867.
URL http://doi.acm.org/10.1145/997817.997867

[4] M. Kazhdan, H. Hoppe, Screened poisson surface reconstruction, ACM Trans. Graph. 32 (3) (2013) 29:1–29:13. doi:10.1145/2487228.2487237.
URL http://doi.acm.org/10.1145/2487228.2487237

[5] G. Mei, J. C. Tipper, N. Xu, Ear-clipping based algorithms of generating high-quality polygon triangulation, CoRR abs/1212.6038.
URL http://arxiv.org/abs/1212.6038

[6] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, C. T. Silva, A benchmark for surface reconstruction, ACM Trans. Graph. 32 (2) (2013) 20:1–20:17. doi:10.1145/2451236.2451246.
URL http://doi.acm.org/10.1145/2451236.2451246

[7] Kazhdan, Screened poisson surface reconstruction (2016).
URL http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version9.01/

[8] K. Clarkson, Robust cocone implementation (2001).
URL http://web.cs.ucdavis.edu/~amenta/powercrust.html

[9] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia, MeshLab: an Open-Source Mesh Processing Tool, in: V. Scarano, R. D. Chiara, U. Erra (Eds.), Eurographics Italian Chapter Conference, The Eurographics Association, 2008. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[10] S. Goswami, Robust cocone implementation (2004).
URL http://www3.cs.stonybrook.edu/~algorith/implement/cocone/implement.shtml

[11] M. Duckham, L. Kulik, M. Worboys, A. Galton, Efficient generation of simple polygons for characterizing the shape of a set of points in the plane, Pattern Recogn. 41 (10) (2008) 3224–3236. doi:10.1016/j.patcog.2008.03.023.
URL http://dx.doi.org/10.1016/j.patcog.2008.03.023