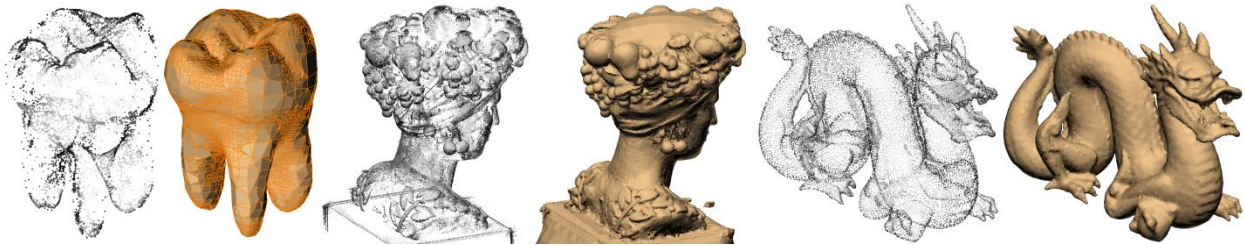


BOURNEMOUTH UNIVERSITY

Surface Reconstruction from Point Clouds

Master Thesis



Navpreet Kaur Pawar

M.Sc. Computer Animation and Visual Effects

Supervisor: - Jon Macey

15-AUG-2013

ABSTRACT

Recent advancement in the three-dimensional acquisition techniques provides huge input of unorganized three dimensional point data sets. With an increase in the application areas making use of point clouds, there is a growing demand to reconstruct a continuous surface representation that provides an authentic representation of the unorganized point sets and render the surface for visualization.

The main goal of the project is the study of various reconstruction algorithms and the creation of a 3d model of an object from a point cloud. The project was started with the use of Point Cloud Library to create 3d polygonal meshes and understanding its usage and experimenting with the various algorithms used in PCL (Point cloud Library) for reconstructing meshes from the point clouds. PCL makes use of many algorithms like Marching Cubes, Grid Projection, Greedy Projection, etc. After experimenting with the PCL's Grid Projection algorithm, the project focused on creating the mesh using Marching Cubes algorithm. The data for Marching Cube has been created from the Point Cloud using Principal Component Analysis and Hermite Radial Basis Function.

ACKNOWLEDGEMENTS

I would like to thank my project supervisor Jon Macey for all his support to the project and his NGL library. Also I would like to thank Mathieu Sanchez for his advices on various technical subjects and his inputs led to the completion of my project.

I would also like to thank my second supervisor Hammadi Nait-Charif for explaining the concept of Surface reconstruction during the early stages of my project.

CONTENTS

- 1- Introduction
- 2- Problem Statement
- 3- Thesis Structure
- 4- Important Terms
- 5- Tools and Technologies used
- 6- Background Research
 - a. Various Algorithms and classifications
 - b. Basic Steps of Reconstruction from Point Cloud
- 7- Greedy Projection Algorithm
- 8- Marching Cubes Algorithm
- 9- Kdtree – Nearest Neighbor Approximation
- 10- Surface Normals Estimation using Principal Component Analysis
- 11- Hermite Radial Basis Function
- 12- Project Structure and Flow
 - a. STAGE-1: Initial research and experiments with PCL and OpenCV.
 - b. STAGE-2: Understanding the Greedy Projection Algorithm Implementation and experimentation
 - c. STAGE-3: Final Project Structure and Flow
- 13- Results and Comparisons
- 14- Problems Faced
- 15- Conclusion
- 16- References

CHAPTER 1: INTRODUCTION

Surface Reconstruction is a challenging problem in the field of computer graphics with a wide range of application areas like medical imagery, virtual reality, video games, movies, e-commerce and other graphic applications. The unorganized point clouds derived from varied inputs like laser scanner data, photogrammetric image measurements, motion sensors etc. pose a tough problem of reconstruction, not completely solved and challenging in case of incomplete, noisy and sparse data.

The reconstruction of satisfactory models which can fulfill the high modeling and visualization demands of such application areas is a nontrivial problem which has not been completely solved because of incomplete, noisy and sparse input data at hand. The raw input points are often unorganized, lacking inherent structure or orientation information.

Surface reconstruction from raw geometric data has received increasing attention due to the ever broadening range of geometric sensors (for example Microsoft Kinect), laser data scanners and vision algorithms that provide little to no reliable attributes. The unavoidable presence of noise and outliers makes the challenge even greater and any progress in this direction can also directly benefit reconstruction from point data set input or the point clouds with attributes.

The goal is to create the model of an object which best fit the reality. Polygonal meshes are the commonly accepted graphic representation, with the widest support from existing software and hardware.

The wide range of applications from which the data may emerge (e.g. manufacturing, reverse engineering, cultural heritage, architecture) implies that the data can have quite different properties that must be considered in the solution of the surface interpolation problem.

The goal is always to find a way to create a computer model of an object which best fit the reality. Polygons are usually the ideal way to accurately represent the results of measurements, providing an optimal surface description. While the generation of digital terrain models has a long tradition and has found efficient solutions, the correct modeling of closed surfaces or free-form objects is of recent nature, a not completely solved problem and still an important issue investigated in many research activities. Many methods have been developed [Menc1, 2001] to create a regular and continuous (triangular) mesh representation from a point cloud.

CHAPTER 2: PROBLEM STATEMENT

The goal of *surface reconstruction* is to determine a surface S from a given set of points P , sampled from a surface in R^3 such that the points of set P lie on S . The surface S approximates the set of points P .

From a mathematical point of view, a surface in the Euclidean three-dimensional space R^3 is defined as a two-dimensional manifold that is compact, connected and contains information about face orientation. In other words, we might say that a surface is a “continuous” subset of points in R^3 which is locally two-dimensional.

A surface may have a border, when the boundary is not empty, or it may be closed, when the boundary is empty. The problem of surface reconstruction can therefore be formalized as follows:

Given a set of points $S = \{ P_i = (x_i, y_i, z_i) / (x_i, y_i, z_i) \in M \subset R^3, i=1\dots k, M \text{ surface in } R^3 \}$

Find a surface M' which interpolates or approximates M , using the data set S .

CHAPTER 3: THESIS STRUCTURE

The thesis outlines the introduction and background of Surface Reconstruction in Chapter 6. The chapter explains the various classifications of algorithms available for the reconstruction and gives an outline of the popular methods used for the problem it poses.

Further Chapter 7 and 8 detail the algorithms used in the research of the project. Marching Cubes Algorithm used in the project is discussed in Chapter 8 while the algorithm that was initially researched and experimented with is discussed in Chapter 7. The implementation of Greedy Projection could not be completed due to lack of time, complexity of the algorithm and missing details in the paper (Rusu , 2009).

Chapter 9, 10, 11 discuss the various methods used to process the point cloud data for reconstruction.

Chapter 9 and 10 detail the nearest neighbor search using Kd-tree and Principal Component Analysis (PCA) to help in the calculation of Surface Normals.

Chapter 11 details the Hermite Radial Basis Function (HRBF) which provides the corresponding value for the point cloud position in the marching cube's unit cube.

Chapter 12 details the Project Structure and Flow and the evolution of Project Pipeline.

Chapter 13 discusses the Results and comparisons

Chapter 14 details the problems faced in the project and what solutions were devised keeping in mind the time period of the project.

The last chapter 15 concludes the project with the lessons learnt and future work.

CHAPTER 4: IMPORTANT TERMS

Modeling: the (mathematical) construction and computer implementation of an object, by defining points in a 3 dimensional array. This array is based on the X, Y and Z axis of geometric space. Then, different sets of these points are mathematically 'joined' by say, lines, to create polygons and the polygons joined to create objects. The simplest result is usually displayed as a wireframe model.

Rendering: is referred to the usually realistic drawing of 3D objects using computer technologies. In order to render an object, certain properties like transparency, colour, diffuse or specular reflection and refraction must be assigned to it and to the surrounding environment.

Mesh: it is a collection of triangular (or quadrilateral) contiguous, non-overlapping faces joined together along their edges. A mesh therefore contains vertices, edges and faces and its easiest representation is a single face. Sometimes it is also called TIN, Triangulated Irregular Network

Surface: a compact, connected, orientable 2D or 3D manifold, possibly with boundary. A surface without boundary is a closed surface. A surface can be geometrically represented in implicit form (locus of the solution of the function $F(x,y,z)=0$) or parametric form (a collection of parametric patches properly joined together). Surfaces, which cannot be described in an analytic form, are called free-form surfaces.

IsoSurface: An isosurface can be defined as follows. Given a scalar field $F(P)$, with F a scalar function on R^3 , the surface that satisfies $F(P) = \alpha$, where α is a constant, is called the *isosurface* defined by α . The value α is called the *isovalue*. In practice, isosurface extraction usually involves generation of an approximate, piecewise isosurface (usually composed of a collection of triangles) on a sampled scalar field

Incremental Surface-Oriented Construction (MencI, 1998): the idea of surface-oriented construction is to build-up the interpolating or approximating surface directly on surface-oriented properties of the given data points. For example, surface construction may start with an initial surface edge at some location of the given point set, connecting two of its points which are expected neighbors on the surface. The edge is successively extended to a larger surface by iteratively attaching further triangles at boundary edges of the emerging surface.

CHAPTER 5: TOOLS AND TECHNOLOGIES USED

Open GL: OpenGraphics Library is cross-language, multi-platform application programming interface (API) for rendering 2D and 3D computer graphics. The 3D graphics programmer interface was initially design by Silicon Graphics Inc. (SGI) and now developed by several companies, to improve the performances of graphical hardware supporting the Open GL standard.

QtCreator: Qt Creator is a cross-platform C++ integrated development environment which is part of the SDK for the Qt GUI Application development framework. It includes a visual debugger and an integrated GUI layout and forms designer. The editor's features include syntax highlighting and autocompletion, but not tabs. Qt Creator uses the C++ compiler from the GNU Compiler Collection on Linux and FreeBSD.

Eigen: Eigen is a high-level C++ library of template headers that implements linear algebra and related matrix operations. Since it is a header library, there is no need for any installation.

NGL: graphics library by Jon Macey (NCCA).It comprises of a number of C++ classes useful for graphics programming.

For research and understanding:

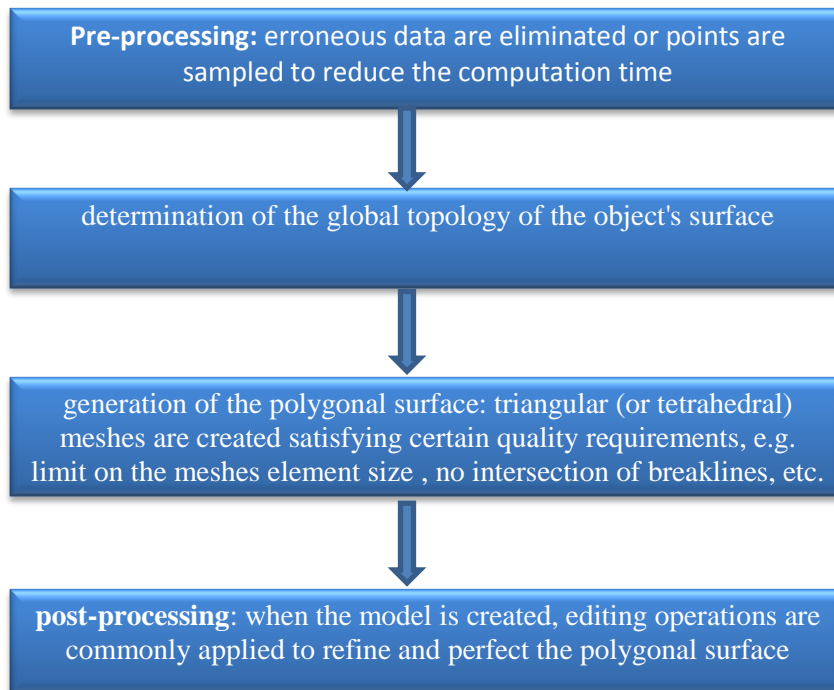
PCL: The Point Cloud Library (**PCL**) is a standalone, large scale, **open project** for 2D/3D image and point cloud processing. It implements a set of algorithms designed to help work with 3-D data, in particular point clouds.

OpenCV: Open Source Computer Vision Library is a cross-platform library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It focuses mainly on real-time image processing.

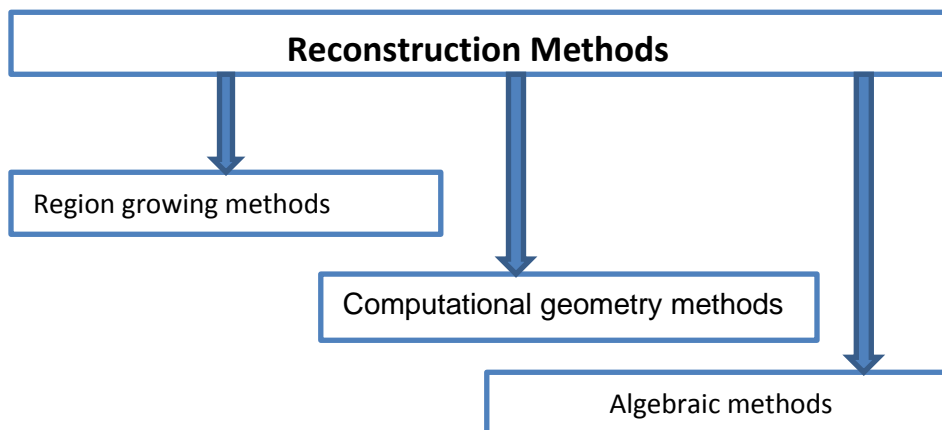
CHAPTER 6: BACKGROUND

The research literature on surface reconstruction is vast and there exist many different algorithms and approaches for the problem.

The main steps involved in any algorithm are as stated below:



There are various reconstruction algorithms and it is a difficult task to discuss all of them. The following table shows one classification of reconstruction methods. We will be discussing only a few of each category.



Region Growing Methods: propagate information and thus reconstruct surfaces in a progressive style.

Hoppe(1992) provides an estimate of the surface normal for the point by fitting a plane to the neighborhood around each data point.

Bernardini's (1999) Ball-Pivoting algorithm grows a mesh from an initial seed triangle that is correctly oriented. A ball of specified radius is pivoted across edges of each triangle bounding the growing mesh. If the pivoted ball hits vertices that are not yet part of the mesh, a new triangle is instantiated and added to the growing mesh. Non-manifold constructions are avoided in the above process.

Computational geometry methods: depend on mechanisms, such as Delaunay triangulation and Voronoi diagram. Such methods interpolate the original points and basically they are sensitive to the presence of noise. Two most successful examples of this classification are Alpha Shapes and the Crust algorithm

In Alpha shapes, the shape is carved out by removing simplexes of the Delaunay triangulation of the point set. A simplex is removed if its circumscribing sphere is larger than the alpha ball.

In the Crust algorithm, Delaunay triangulation is performed on the original set of points along with Voronoi vertices that approximate the medial axis of the shape. The resulting triangulation distinguishes triangles that are part of the object surface from those that are on the interior because interior triangles have a Voronoi vertex as one of their vertices.

Algebraic methods: try to fit a function to the data points. They avoid creating noisy surfaces by fitting a smooth function, and by not requiring that the function pass through all data points. Most of the signed distance based and implicit function based methods fall under this category.

Dinh (2002)proposes a reconstruction approach based on a summation of non-polynomial basis functions whose domain is a scalar value obtained from the distance between sample points.

Fang and Gossard (1995) reconstruct piecewise continuous parametric curves. Fang and Gossard show examples using Hermite basis.

CHAPTER 7: GREEDY PROJECTION ALGORITHM

As explained by Rusu(2009) in his paper, the Greedy Projection algorithm works by maintaining a list of points from which the mesh can be grown (“fringe” points) and extending it until all possible points are connected. Triangulation is performed locally, by projecting the local neighborhood of a point along the point’s normal and connecting unconnected points.

The algorithm is based on incremental surface growing principle (Mercl,1998), following a greedy type approach. The algorithm starts by creating a starting triangle and keeps on adding new triangles until all the points in the cloud have been considered or there no more valid triangles which can be connected to the resultant mesh.

Algorithm Flow:

1. Nearest neighbor search: For each point ‘ p ’ in the point cloud, a k -neighborhood is selected. This neighborhood is created by searching the reference point’s nearest k -neighbors within a sphere of radius r . The radius is defined as $\mu.d$, where d is the distance of the point p from its closest neighbor and μ is the user-specified constant to take into account the point cloud density. (Discussed in Chapter: 9) To find the nearest neighbors for the given point in the point cloud, Kdtree nearest neighbor search has been used.
The points in the cloud are assigned various states depending on their interaction with the algorithm: *free*, *fringe*, *boundary*, and *completed*.
 - a. Initially, all the points in the cloud are in the **free** state and free points are defined as those points which have no incident triangles.
 - b. When all the incident triangles of a point have been determined, the point is referred to as of **completed** state.
 - c. When a point has been chosen as a reference point but has some missing triangles due to the maximum allowable angle parameter, it is referred to as a **boundary** point.
 - d. **Fringe** points are the points that have not yet been chosen as a reference point
2. Neighborhood projection using tangent planes: the neighborhood is projected on a plane that is approximately tangential to the surface formed by the neighborhood and ordered around p .
3. Pruning: The points are pruned by visibility and distance criterion, and connected to p and to consecutive points by edges, forming triangles that have a maximum angle criterion and an optional minimum angle criterion. The points in the point cloud are pruned depending on many criterions.
 - a. Pruning by distance criterion: a distance criterion is applied to prune down the search for candidate adjacent points in the spatial proximity of current reference point using kd-tree.
 - b. Further points which lie outside the sphere of influence centered at reference point are rejected. The chosen points are referred to as the candidate points.
 - c. Choice of projection plane: the candidate set of points obtained after applying the distance criterion are projected on the the approximate tangent plane.

- d. Angle ordering: a new local coordinate system is defined with the reference point as the origin and the plane projection of the previous step serves as the xy-plane. All the points in the candidate set are projected this plane. Ordering around the ref. point is based on the angle (Θ) between the x-axis of the local coordinate system and the vector from origin to the projected candidate point.
- e. Visibility: the points which potentially form a self-intersecting mesh are discarded. The algorithm defines two edge types for checking this condition:
 - i. Boundary Edge: an edge with only one triangle incident on it. These edges connect fringe and/or boundary points.
 - ii. Internal Edge: connect the completed points with any other points.

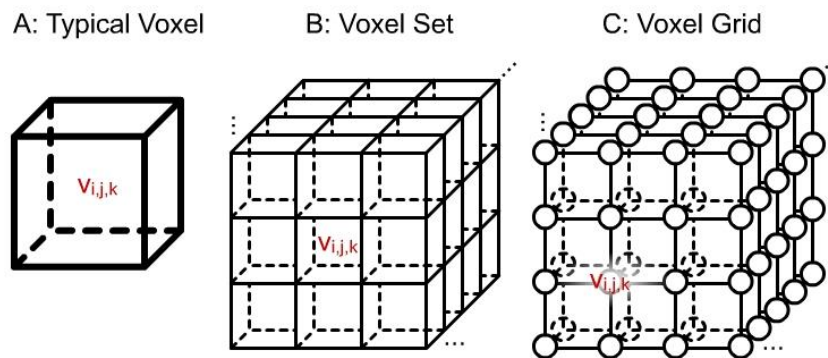
The plane is projected using the reference point, candidate set of points and the boundary edges. In case, the line of sight from the reference point to a candidate vertex is obstructed by an edge, the point is occluded.

The input to this algorithm is a point cloud with estimated surface normals and the curvature data calculated using Principal Component Analysis discussed in Chapter 10.

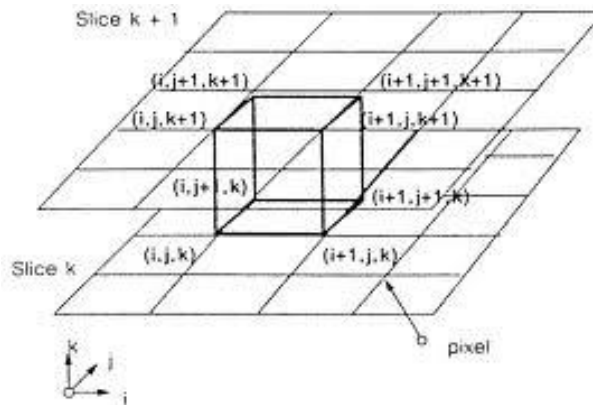
CHAPTER 8: MARCHING CUBES ALGORITHM

The Marching Cube (MC) algorithm is the most used for the isosurface reconstruction. This algorithm has been designed by William E. Lorensen and Harvey E. Cline in 1987 to generate a 3D model.

The standard MC algorithm, as originally described by Lorensen and Cline (1987), takes as its input a *regular scalar volumetric data set*. Such a data set has a scalar value residing at each lattice point of a rectilinear lattice in 3D space. The algorithm creates subdivides the region of space into 3D array cubes, also known as voxels.

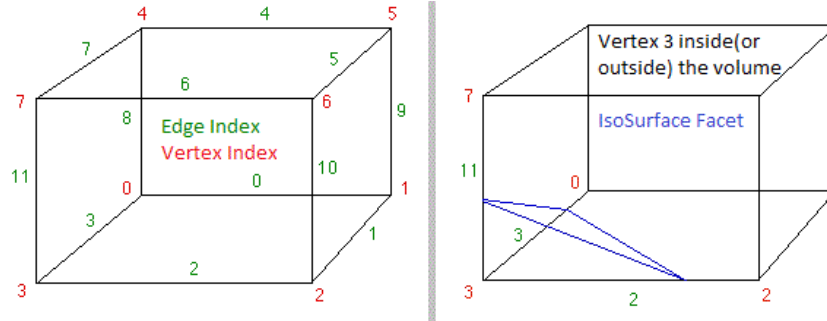


The basic cube is called a voxel and is formed by 8 vertices and 12 edges.



The MC processes the volumetric data set by considering the “cubes” that make up the volume. Each lattice point is a corner vertex of a cube. The cubes are defined by the volume's lattice. The algorithm instructs to 'march' through each of the cubes while testing the corner points and replacing the cube with an appropriate set of polygons. The sum total of all polygons generated will be a surface that approximates the one the data set describes.

Each vertex and each edge of the cube is indexed for lookup in the tables. By determining which edges of the voxel are intersected by the isosurface, we can create triangular patches that divide the cube into different regions that are within the isosurface and the regions that are outside. By connecting the patches from all voxel on the isosurface boundary, we get a surface representation.



Left figure shows the indexing order followed in the algorithms implementation in the project and right figure shows a case when vertex 3 is either below or above the isosurface value and the isosurface facet cuts through the edges 2, 3 and 11. The exact position of the vertices of the triangular facet depends on the relationship of the isosurface value to the values at the vertices 3-2, 3-0, 3-7 respectively.

In case, one vertex is above the isosurface and an adjacent vertex is below the isosurface, then the position at which the isosurface cuts the edge is linearly interpolated. The ratio of the length between the two vertices will be the same as the ratio of the isosurface value to the values at the vertices of the grid cell.

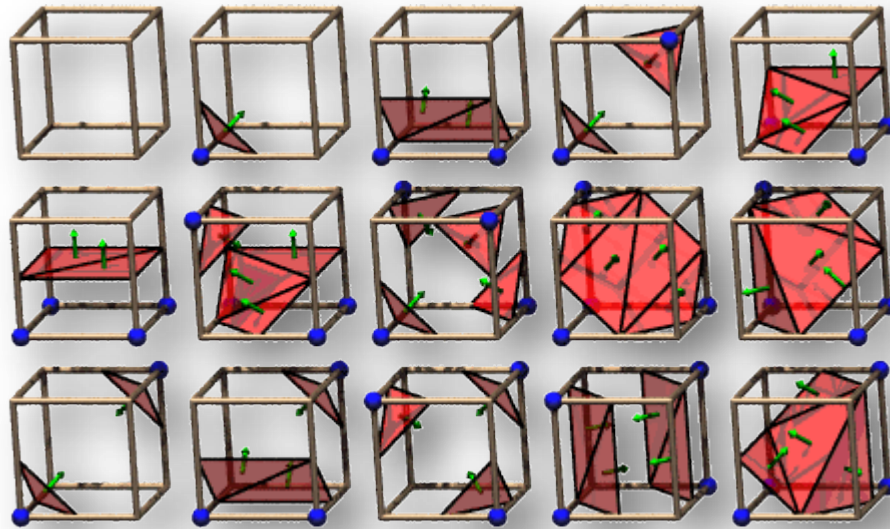
The intersection points can be calculated by linear interpolation. If P_1 and P_2 are the vertices of a cut edge and V_1 and V_2 are the scalar values of each vertex, the intersection point P is given by the following equation:

$$P = P_1 + (isovalue - V_1)(P_2 - P_1) / (V_2 - V_1)$$



Since each of the eight vertices of a cube can be either marked or unmarked, there are 256 (2^8) potential combinations of the corner status. These combinations are simplified by taking into account cell combinations that duplicate under the following conditions:

- Rotation by any degree over any of the 3 primary axis
- Mirroring the shape across any of the 3 primary axis
- Inverting the state of all corners and flipping the normals of the relating polygons.[Sharman]

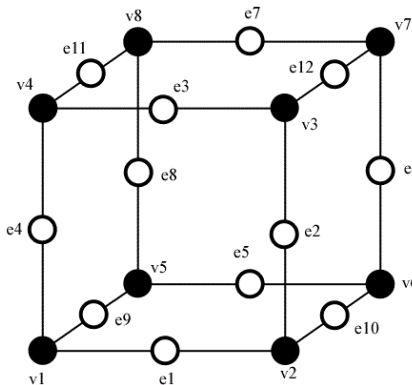
These conditions reduce the original 256 combinations of cell state to 15 combinations as shown below:



The 15 Cube Combinations

-  Blue sphere denote the vertices that have been tested as 'Inside' the shape
-  Green arrows denote the surface normal of respective generated triangles.

For each case, an index is created based on the state of the vertex. Using the vertex numbering in Figure 4, the eight bit index contains one bit for each vertex. This index serves as a pointer into an edge table that gives all edge intersections for a given cube configuration.



Using the vertex numbering in Figure 4, the eight bit index contains one bit for each vertex. This index serves as a pointer into an edge table that gives all edge intersections for a given cube configuration.

Looking up the edgeTable returns a 12 bit number, each bit corresponding to an edge

$$\text{Index} = \overline{v8}v7v6v5v4v3v2v1$$

In the final step of the algorithm, the normal for the faces are calculated for correct rendering of the surface.

The steps involved in the implementation of the algorithm can be summarized as follows:

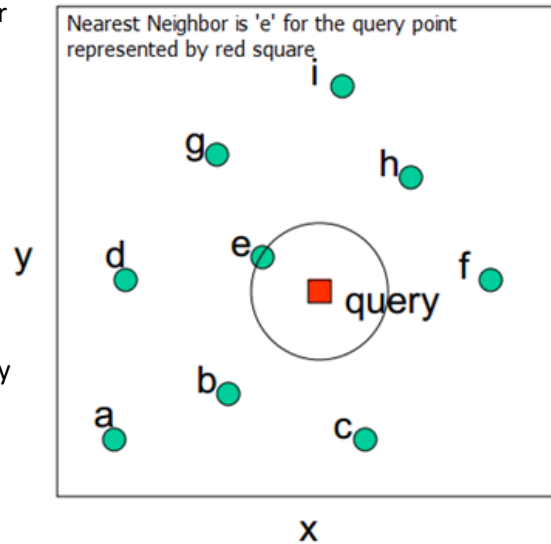
- STEP 1:** define a cube and number its vertices according to the Paul Bourke convention.
- STEP 2:** determination of the index and use the index as a pointer in the Lookup table, which defines the set of intersections of the interested surface by the cube edges
- STEP 3:** Calculate the intersection points on the cube edges by linear interpolation.

CHAPTER 9: NEAREST NEIGHBOR SEARCH PROBLEM

Nearest Neighbor Search Problem Definition: Given a set S of points in a n -dimensional space, construct a data structure which given any query point Q finds the point in S with the smallest distance to Q .

The easiest approach to find the nearest neighbor to the point Q will be to compute the distance from Q to each point in P . This linear scan approach is tolerable for small data sets but for large data sets of point clouds, it is very expensive. So we are using Kd-Trees by Jon Bentley, 1975 to solve this problem.

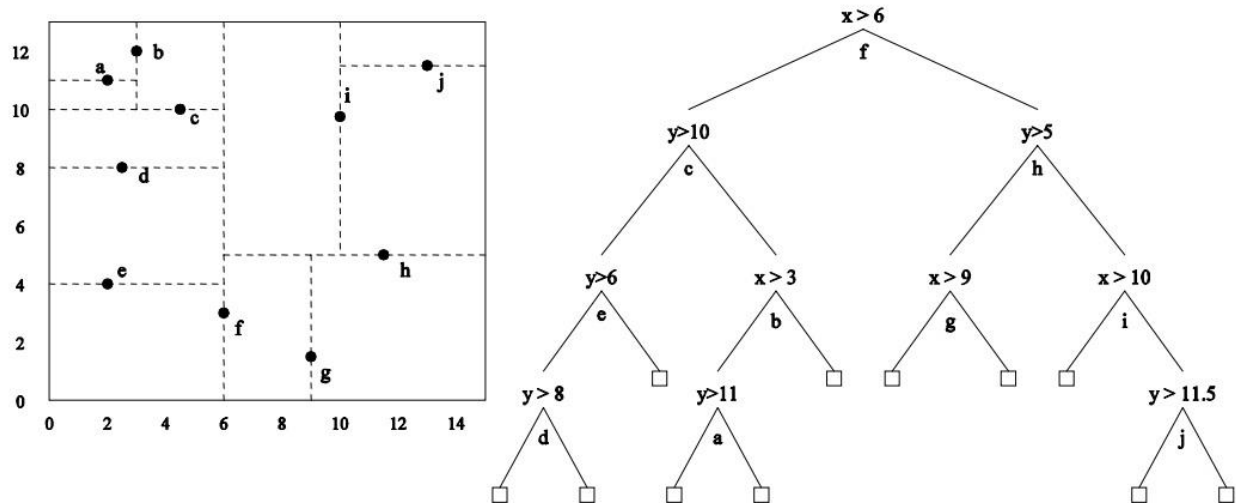
The key point of the problem formulation is that dataset S is considered fixed. The query point may vary from request to request, but S remains unchanged. Kdtree preprocesses the dataset and build the tree data structure which accelerates processing.



The Kd-trees

Kd tree or a k -dimensional tree is a space-partitioning data structure for organizing points in a k -dimensional space. The k -d tree is a generalization of binary search trees in which every node is a k -dimensional point. Every non-leaf node generates a splitting hyperplane that divides the space into two parts, known as half-spaces.

Each node in the tree is defined by a plane through one of the dimensions that partitions the set of points into left/right (or up/down) sets, each with half the points of the parent node. These children are again partitioned into equal halves, using planes through a different dimension. Partitioning stops after $\log n$ levels, with each point in its own leaf cell. The partitioning loops through the different dimensions for the different levels of the tree, using the median point for the partition. kd-trees are known to work well in low dimensions but seem to fail as the number of dimensions increase beyond three.



A kd-tree is similar to a decision tree except that we split using the median value along the dimension having the highest variance. Every internal node stores one data point, and the leaves are empty.

kd-trees allow to efficiently perform searches with problem statements like "find k -nearest neighbors of Q " or "find all points at distance lower than R from Q ".

Algorithm: (Bentley, 1980, Chapter - 3.3 Nearest Neighbors)

- 1) locate the location where the point would be located if it were added to the tree :
Starting with the root node, the algorithm moves down the tree recursively, taking decisions to follow left or right node depending on whether the point is less than or greater than the current node in the split dimension.
- 2) Once the algorithm reaches a leaf node, it saves that node point as the "current best". As the tree is traversed the distance between the point and the current node should be recorded.
- 3) The algorithm goes back up the tree evaluating each branch of the tree that could have points within the current minimum distance i.e. it unwinds the recursion of the tree, performing the following steps at each node:
 - a) If the current node is closer than the current best, then it becomes the current best.
 - b) Check the other side of the hyperplane for points that could be closer to the search point than the current best. To check this, intersect the splitting hyperplane with a hypersphere around the search point that has a radius equal to the current nearest distance.
 - i) If the hypersphere crosses the plane, there could be nearer points on the other side of the plane. The process is repeated in this branch.
 - ii) If the hypersphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.
- 4) Search completes when the algorithm reaches the root node.

CHAPTER 10: Principal Component Analysis for Normal Estimation

Principal component analysis (PCA) involves a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible (Smith, 2002).

In computational terms the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the co-variance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on.

Let A be a $n \times n$ matrix. The eigenvalues of A are defined as the roots of:

$$\text{determinant}(A - \lambda I) = |j(A - \lambda I)| = 0$$

where I is the $n \times n$ identity matrix.

This equation is called the characteristic equation (or characteristic polynomial) and has n roots.

Let λ be an eigenvalue of A . Then there exists a vector x such that:

$$Ax = \lambda x$$

The vector x is called an eigenvector of A associated with the eigenvalue λ .

To find a numerical solution for x we need to set one of its elements to an arbitrary value, say 1, which gives us a set of simultaneous equations to solve for the other other elements. If there is no solution we repeat the process with another element. Ordinarily we normalise the final values so that x has length one, that is

$$x \cdot x^T = 1$$

In our case, we are dealing with 3d data which gives us $(n \times 3)$ matrix of x . which gives us a 3×3 covariance matrix as $A_{n \times 3} \times A_{3 \times n}$ gives us a matrix of size (3×3) $A_{3 \times 3}$.

So, a 3×3 matrix A with eigenvectors x_1, x_2, x_3 , and eigenvalues $\lambda_1, \lambda_2, \lambda_3$ so:

$$A x_1 = \lambda_1 x_1 \quad A x_2 = \lambda_2 x_2 \quad A x_3 = \lambda_3 x_3$$

Each eigen value corresponds to an eigen vector. For this project the maximum eigen value is being used to find its corresponding eigen vector. This is a column vector which stores the (x,y,z) coordinates of the normal position.

Method for Normal Estimation

In the computer graphics, the surface normals are used for generating the correct shadows based on the lightning. In the surface reconstruction, the normals define the direction of the polygon faces. The normal estimation of the geometric surface is usually a trivial task, but the points in the point cloud do not have any surface, they only represent one. For this reason, the task is nontrivial.

There are basically two different approaches for the normal estimation.

- 1 - Reconstruct the surface, which the points represent, and calculate the normals from that.
- 2 - Approximate the normal data directly from the point cloud.

The input point cloud in the project takes only the position values of the point cloud, and we are using the second method stated above to calculate the surface normal from the point cloud.

For each point p in cloud P :

1. Get the nearest neighbors of point P using the nearest neighbor search using kd-tree discussed in chapter 9.
 - i. Compute the mean vertex for the neighborhood and subtract the mean from all the points.
 - ii. Using the mean vertex of the neighborhood, calculate the centroid for the neighborhood.
 - iii. The centroid is used for calculating the covariance matrix.
 - iv. Calculate the Eigen Vector and the Eigen Values of the Covariance Matrix.

2. A local coordinate system is created by taking a cross product of the eigen vector normal with its unit orthogonal.

```
LocalV = eigenNormal.unitOrthogonal();  
LocalU = eigenNormal.cross(LocalV);
```

The normalized localU gives us the surface normal.

3. Check if n is consistently oriented towards the viewpoint and flip otherwise

CHAPTER 11: HERMITE RADIAL BASIS FUNCTION

The marching cube algorithm works on implicit surfaces. To use the algorithm, one needs to provide a function that can provide the point of interest in the marching cube's voxelized space.

The Hermite radial basis function (HRBF) implicit method computes an implicit function which interpolates scattered multivariate Hermite data (unstructured points and their corresponding normals). Differently from previous radial basis functions (RBF) approaches, HRBF implicit do not depend on offset points to ensure existence and uniqueness of its interpolant (Macedo, c.a. 2007).

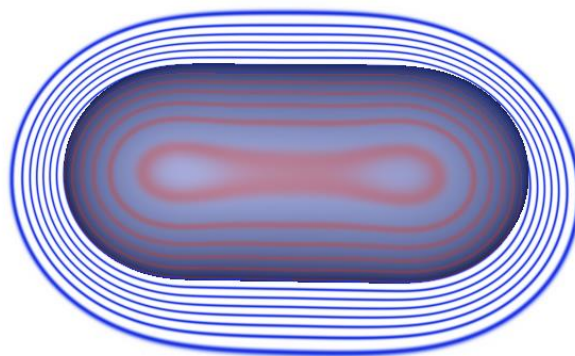
This function should be able to perform the computation of implicit surfaces that approximate or interpolate scattered point cloud data.

As described by Vaillant (2007) in his tutorial the surface is represented by a scalar field $f:R^3 \rightarrow R$. The function interpolates the given set of N points along with their normal $(\mathbf{p}_i, \mathbf{n}_i)$ to reconstruct the surface.

The project makes use of the code provided by Vaillant (2007) along with the tutorial.

Function f returns values ranging from $[-\infty; +\infty]$ and has increasing values from the inside to the outside.

$f(\mathbf{p}) = 0$ on the surface
 $f(\mathbf{p}) < 0$ (red) for curves inside the surface
 $f(\mathbf{p}) > 0$ (blue) for curves outside the surface



The GRBF calculates the normal also but we are not going to use the normal from this function, but will be using the normal of the surfaces generated in marching cube. The gradient return a vector which contains in each component the partial derivatives of f :

$$\nabla f = (\partial f(x, y, z) / \partial x, \partial f(x, y, z) / \partial y, \partial f(x, y, z) / \partial z)$$

$$\begin{aligned} \text{The function } f: \mathbb{R}^3 \rightarrow \mathbb{R} \quad - \quad f(\mathbf{x}) &= \sum_i^N \alpha_i \varphi(\|\mathbf{x} - \mathbf{p}_i\|) + \beta_i \cdot \nabla[\varphi(\|\mathbf{x} - \mathbf{p}_i\|)] \\ &= \sum_i^N \alpha_i \varphi(\|\mathbf{x} - \mathbf{p}_i\|) + \mathbf{a}_i(\mathbf{x})^T \boldsymbol{\beta}_i \end{aligned}$$

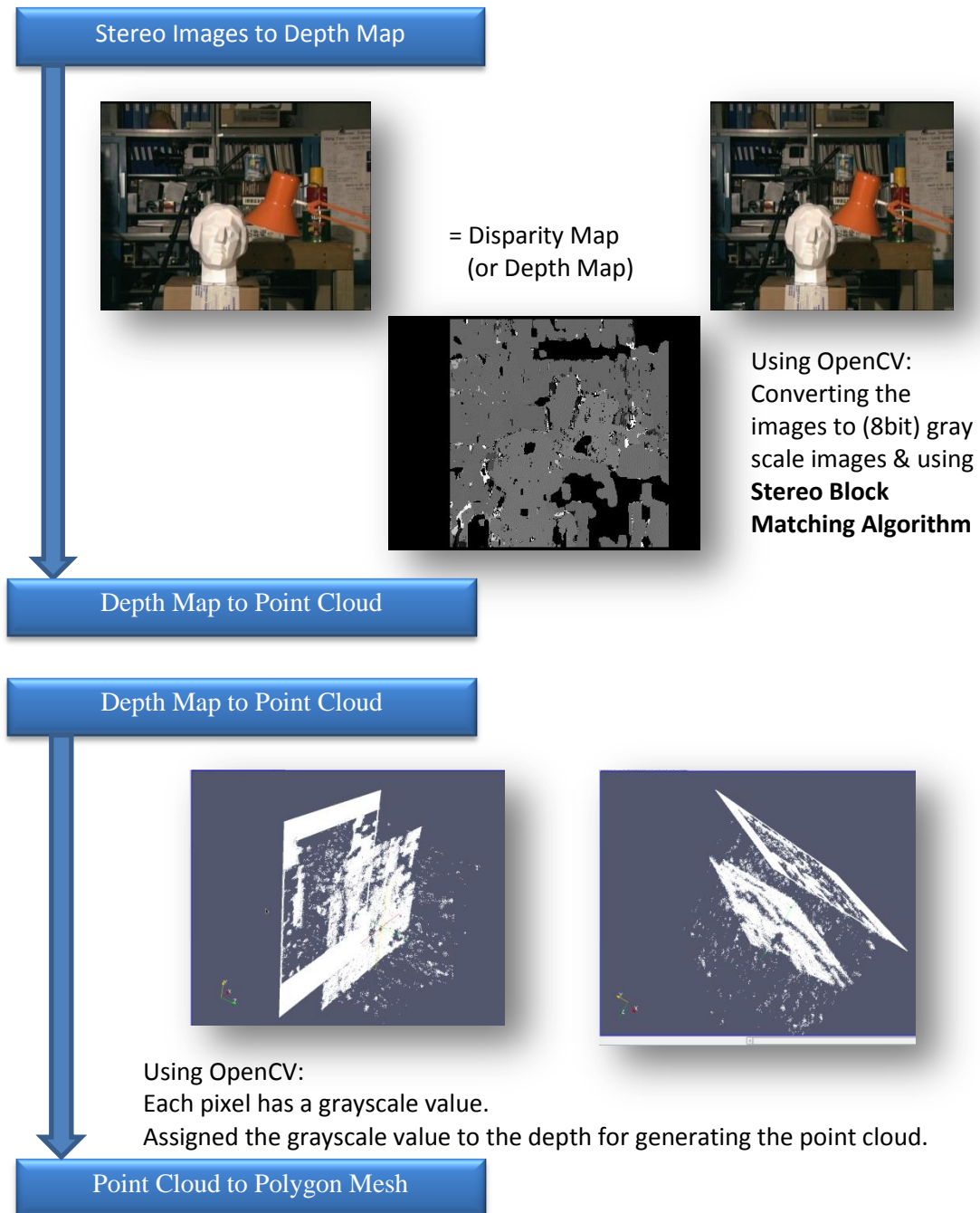
Now, with have $\mathbf{x} \in \mathbb{R}^3$ the evaluated position and \mathbf{p}_i the samples/points to be interpolated. The function $\varphi: \mathbb{R} \rightarrow \mathbb{R}$ is called a radial basis function defined as $\varphi(x) = x^3$ for our project.

To compute the value of f we will have to find the N values of $\alpha_i \in \mathbb{R}$ and $\boldsymbol{\beta}_i \in \mathbb{R}^3$, to get value we need for sampling our point cloud in the marching cube for reconstruction.

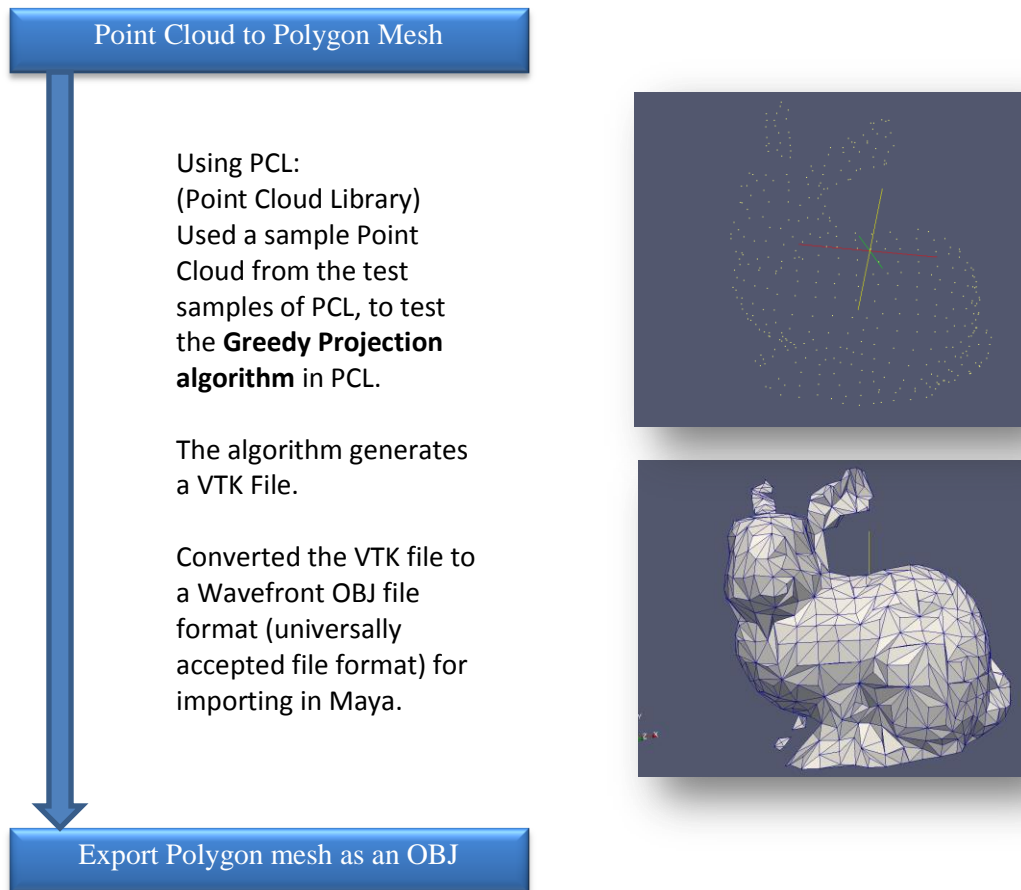
CHAPTER 12: PROJECT STRUCTURE AND FLOW

STAGE 1:- Initial research and experiments with PCL and OpenCV.

Results obtained using OpenCV:



Results obtained using PCL:



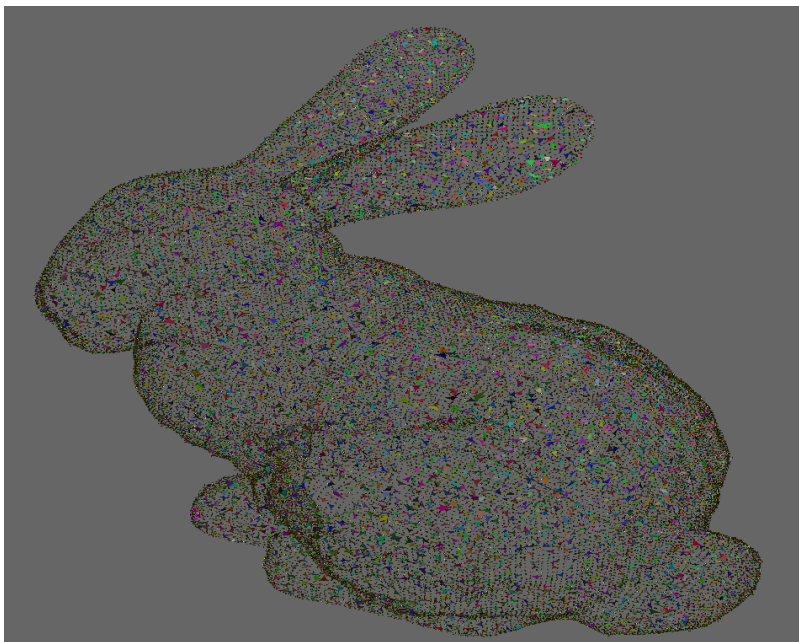
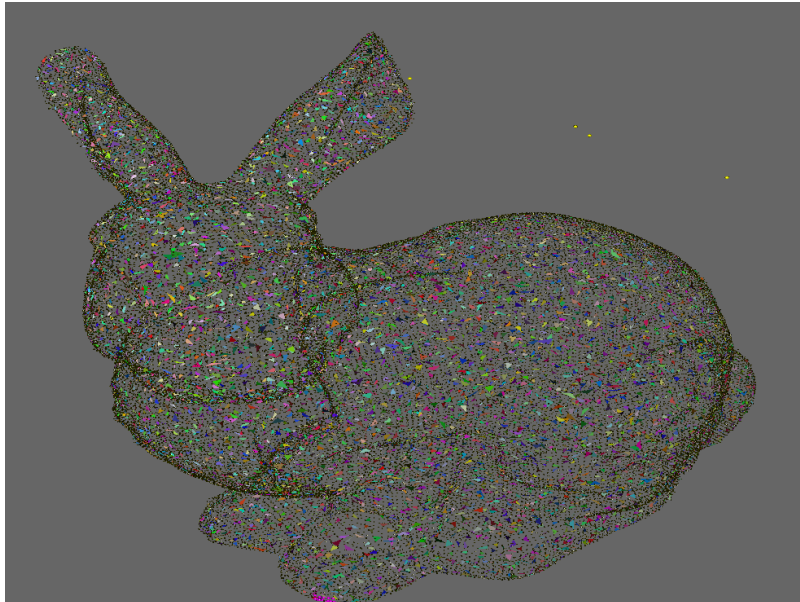
The project was started with the installation and usage of libraries – PCL and OpenCV and the results were discussed during the mid-term project. The project pipeline was changed after the advice of tutors to concentrate on one part of surface reconstruction than to keep a broad pipeline and completing it with the use of libraries.

So the project was restructured to concentrate on the generation of polygonal mesh from input point cloud data. After experimenting with PCL's Greedy projection algorithm and looking at promising results, the implementation of the Greedy projection was studied.

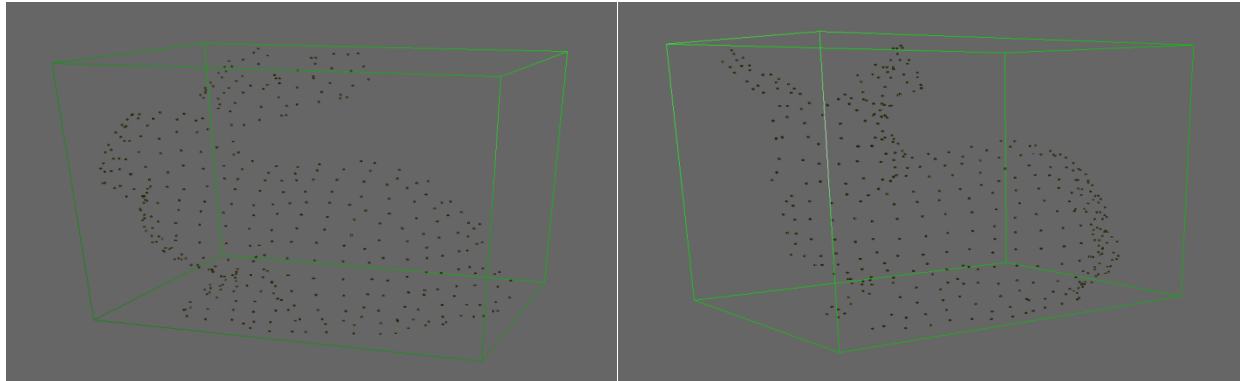
STAGE 2:- Understanding the Greedy Projection Algorithm Implementation and creating the program on similar lines as PCL's implementation for better understanding of the algorithm.

The algorithm is based on the technical paper by Rusu R.B.(2009) which has been implemented in PCL. It is a lengthy code of 2000+ lines running in multiple while loops and taking care of numerous conditions.

I was able to complete initial process of algorithm to simplify the point cloud and calculating the local projections of the tangent planes to estimate surface normal. Using the projection of local coordinate system, the angle between neighbors with respect to the reference points was calculated. The first triangle for all the k-neighborhoods was created using this projection data.



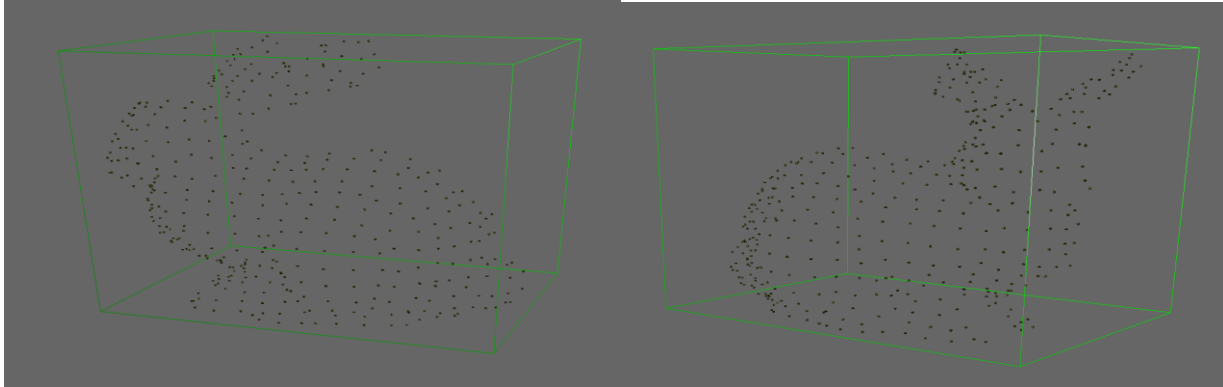
The tests were created by creating the code on similar lines of the algorithm's implementation in PCL. Figure below shows the unified and simplified input point cloud.



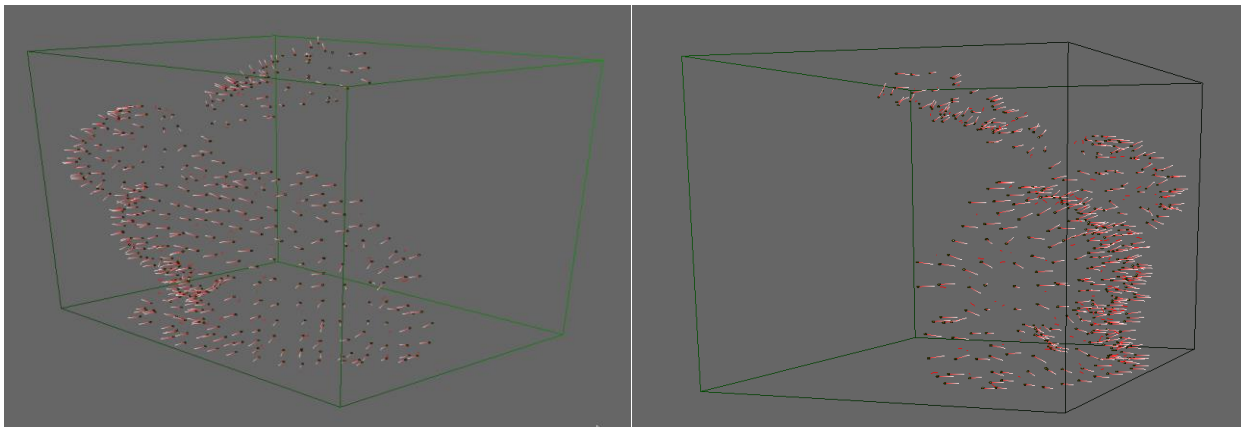
The 2000+ line code running in multiple while loops was difficult to understand and the accompanied paper for the implementation did not list all the conditions for further adding triangles using incremental approach. Due to running short on time for the project, I had to discard the paper and look for an alternate approach as discussed in the next step of Project Flow.

STAGE 3:- Final Outputs and the flow of project is discussed as follows:

STEP 1: Process the input Point cloud containing only the (x, y, z) position coordinates in 3D space. The input is a text file containing the (x, y, z) position values.



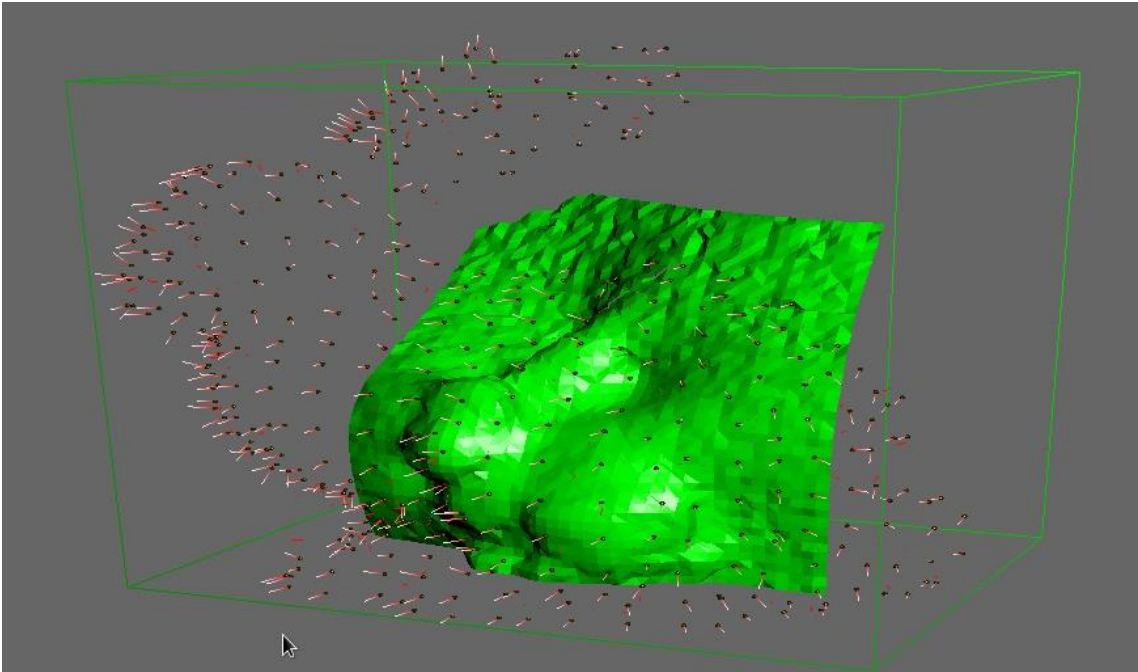
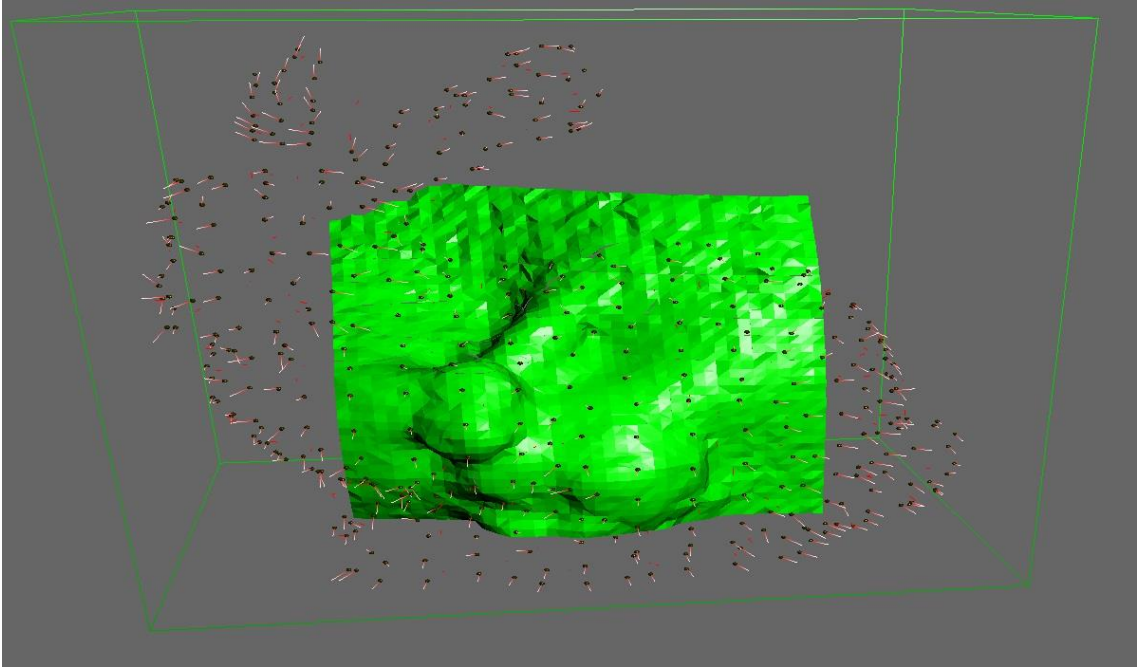
STEP 2: Estimate the surface normals using the Principal Component Analysis discussed in Chapter 10. The resulting surface normal can be seen in the output below:

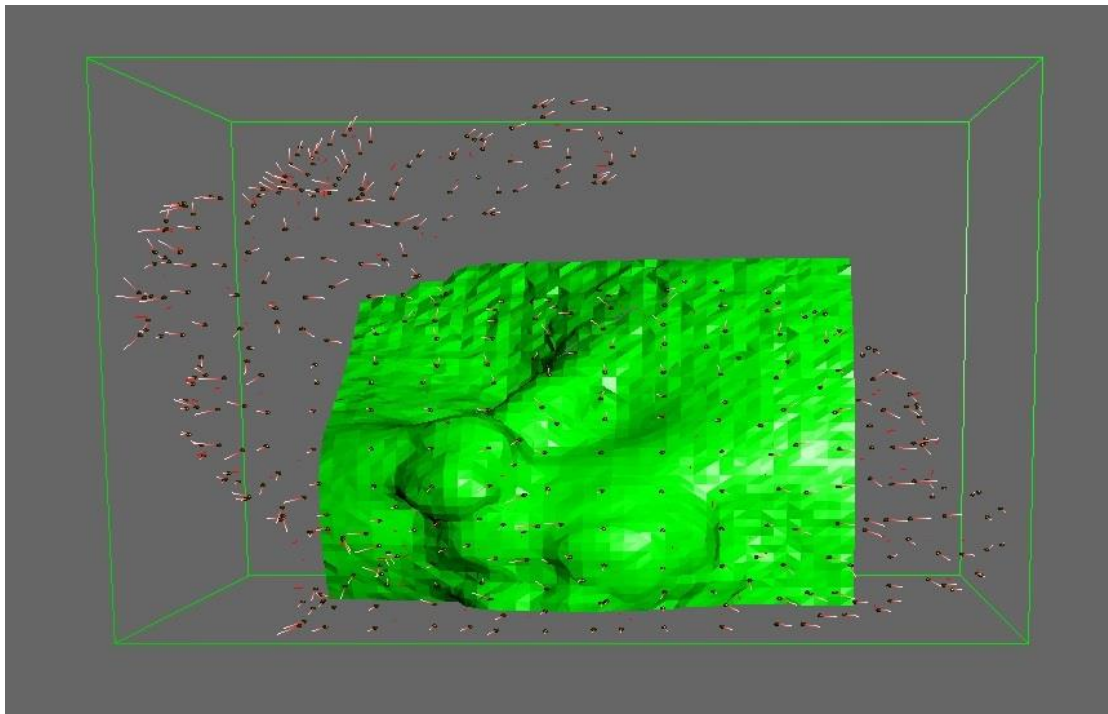
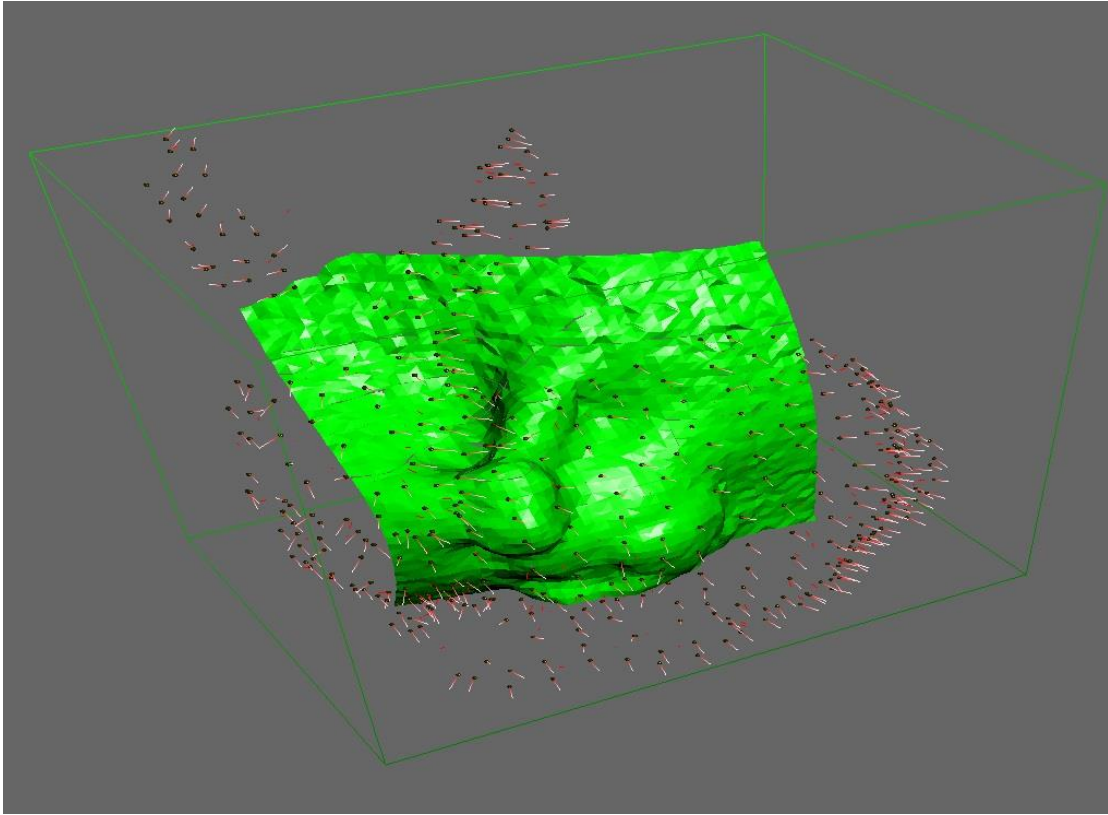


STEP 3: The point cloud position and the calculated surface normal data is fed into the Hermite Radial Basis Function to calculate the sample value for the point cloud, to be used as input to the Marching Cube Algorithm in STEP 4.

STEP 4: Generation of surface mesh using Marching Cubes algorithm as discussed in Chapter 8, using the evaluation of HRBF at each cube of the marching cube's voxel grid. The surface is generated for all the positions (returned by the HRBF evaluation) in the unit cube of the marching cube algorithm.

The code is based on the tutorial by Paul Bourke (1994). The project uses the tables and the basic structure of the tutorial code but is changed to take care of the point cloud input. The main loop for the marching cube that defines the voxel grid runs from the maximum and minimum extents of the Point Cloud Bounding Box.





RESULTS AND COMPARISONS:

The results obtained using the PCL library were

	PCL	GREEDY PROJECTION IMPLEMENTATION EXPERIMENT	PCA+HRBF+MC
Surface Generated	Regular surface generated which retains the mesh boundary	Could generate only first triangle per neighborhood	Regular mesh generated but with some artifacts and the geometry gets extended because of HRBF's property of creating enclosed surfaces.
Speed	Extremely fast because of greedy approach	Fast	Very slow because of marching cubes trying to access the HRBF evaluated output at each cube vertex of the grid.
Algorithm's complexity	The implementation is extremely difficult with so many conditions for pruning.	Due to the non-trivial algorithm and incomplete nature of the paper (Rusu, 2009) the implementation was not fully completed.	The process flow could be understood in a procedural way and hence the implementation could be listed in procedural steps.
Process Adaptability	The implementation is enclosed in while loops which are difficult to break in steps and hence difficult to combine with other methods.		On completion of the process, I could infer that the usage of HRBF was making the process slow. The process can be adapted to include some other method like RBF(Radial Basis Function) or signed Distance Functions could be used and experimented with.
Data set Size handled	The robustness of the project allows the algorithm to process large point clouds.	Steps to unify and simplify the point cloud were implemented which helped in reducing a mesh of 30k points to 4k points.	HRBF implementation is using Eigen library for matrix calculations which makes it difficult to use large data sets.

Problems faced:

- A lot of time was devoted to the installation of libraries and learning the local installation on lab machines using cmake. Initial results were obtained using libraries.
- The project demanded huge time for research and narrowing down on a single algorithm for implementation from the vast resource pool in the area of reconstruction.
- Alteration of project pipeline changed the focus of the project from creating high level detail models to creating simple meshes to understand the core of the problem.
- For the final implementation, very little time was left and hence could not improve upon it. The Final implementation runs slow due to HRBF.

Future Work and Conclusion:

- Use Signed Distance or Radial Basis Function instead of HRBF for non-closed meshes
- Use the HRBF on enclosed meshes by slicing the mesh into parts to take care of surface normal.
- Reduce the run time for program.

REFERENCES:

1. Marton, Z.C., Rusu, R.B., Beetz, M. 2009. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan.
2. Fabio, R. FROM POINT CLOUD TO SURFACE: THE MODELING AND VISUALIZATION PROBLEM. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV-5/W10, Zurich, Switzerland.
3. Lorensen, W. , Cline., H. 1987. MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM. Computer Graphics (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 163-169.
4. Bourke, P. 1994. Polygonising a scalar field. University of Western Australia. Available from : <http://paulbourke.net/geometry/polygonise/>
5. Sharman, J. The Marching Cubes Algorithm. Available from: <http://www.exaflop.org/docs/marchcubes/>
6. Nguyen, T. Nearest Neighbor Search. Oregon State University. Available from: http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf
7. Christopher S. 2011. k-D Tree Nearest Neighbor Search. University of Akron, US. Available from: <http://www.christopherstoll.org/2011/09/k-d-tree-nearest-neighbor-search.html>
8. Bentley, J. Multidimensional Divide-and-Conquer. Carnegie-Mellon University. . Available from : <http://www.cs.uiuc.edu/class/fa05/cs473ug/hw/p214-bentley.pdf>
9. R. Mencl and H. Muller. Interpolation and approximation of surfaces from three-dimensional scattered data points. In: State of the Art Reports, Eurographics '98, 1998, pp. 51–67.
10. H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. 1992. Surface reconstruction from unorganized point clouds. In ACM Siggraph pp 71–78.
11. M. Gopi and S. Krishnan. 2002. A Fast and Efficient Projection-Based Approach for Surface Reconstruction. In: SIGGRAPH '02: Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing, 2002, pp. 179–186.
12. Lecture 15: Principal Component Analysis. In: DOC493: Intelligent Data Analysis and Probabilistic Inference Lecture 15. Imperial College London. Available from: <http://www.doc.ic.ac.uk/~dfg/ProbabilisticInference/IDAPILecture15.pdf>
13. Smith, L. 2002. A tutorial on Principal Components Analysis. Available from : http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
14. Vaillant, R. 2007. Recipe for Implicit surface reconstruction with HRBF. Available from : <http://www.irit.fr/~Rodolphe.Vaillant/?e=12> .
15. Macedo, I. Gois, J.P., Velho, L. Hermite Interpolation of Implicit Surfaces with Radial Basis Functions. Vision and Graphics Laboratory, Instituto Nacional de Matematica Pura e Aplicada (IMPA), Rio de Janeiro, RJ, Brazil. Available from: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.150.1019>.
16. F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taublin. 1999. The ballpivoting algorithm for surface reconstruction. IEEE Trans. on Visualization and Computer Graphics, 5(4):349–359.

17. H. Edelsbrunner and E. Mücke. 1994. Three-dimensional alpha shapes. *ACM Trans. on Graphics*
18. N. Amenta, M. Bern, and M. Kamvysselis. 1998. A new voronoi-based surface reconstruction algorithm. *SIGGRAPH*, pages 415–421,
19. H. Dinh, G. Turk, and G. Slabaugh. 2002. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*.
20. L. Fang and D. Gossard. 1995. Multidimensional curve fitting to unorganized data points by nonlinear minimization. *Computer-Aided Design*, 21(1):48–58.