3D Surface Approximation from Point Clouds

Abhishek Pandey

Undergraduate Dissertation
Computer Science and Electronics
School of Informatics
University of Edinburgh

2019

Abstract

3D Visual perception has recently become very popular because of the advancements in 3D sensor hardware and development of powerful computational devices. Today, 3D sensory information is being extensively used to automate navigation and motion planning in autonomous systems like self-driving cars. Visual perception can be very useful for locomotion in robots. More specifically, a 3-dimensional representation of the surfaces present in a robot's environment can be helpful in motion planning tasks such as foot step planning for legged robots. However, since the information produced by 3D sensors can be very large, it is required to create models that can efficiently represent this information and derive useful features from it. In this project, various models for representing 3D sensory information in the form of surfaces were analysed. Based on the analysis, a representation model was chosen. Later, using the chosen model a system was designed, implemented and tested. More specifically, the designed system uses B-spline surfaces for point cloud approximation. The surfaces generated are C^2 continuous and therefore can be used to apply optimisation algorithms to find optimal locomotion trajectories and patterns. The system provides an easy way to extract the information needed by such algorithms. The surface approximations produced by the system are accurate and meet all the criteria for evaluation.

Acknowledgements

I would like to thank my supervisor, Mr. Zhibin Li, and my PhD. mentor, Iordanis Chatzinikolaidis for all the help and support they have provided during the course of this project.

Table of Contents

Chapter 1

Introduction

1.1 Goal and Motivation

The motivation behind this project is to enable the use of visual perception in robots for locomotion. Visual perception is the ability to represent and extract useful information from visual sensory data. Recently, 3D visual sensors such as LIDAR and RGB-D cameras are being widely used in mapping, navigation and motion planning. [7] This is because such sensors can produce 3D point-clouds very efficiently and accurately. Unlike 2D images, 3D point clouds capture the spatial characteristics of an environment very well. This makes them ideal for mapping or performing self localisation in an environment.

However, points are too abstract of a representation for an environment. A better representation is a continuous surface since most objects in an environment are continuous surfaces with certain characteristics like the slope or height. Information about such characteristics is very useful when performing locomotion. For example, Tasks such as walking or climbing stairs require finding of planar surfaces (see figure 1.1).

Research in cognitive science also suggests that encoding of surfaces is an elementary and indispensable aspect of perception. Surface representation forms an important

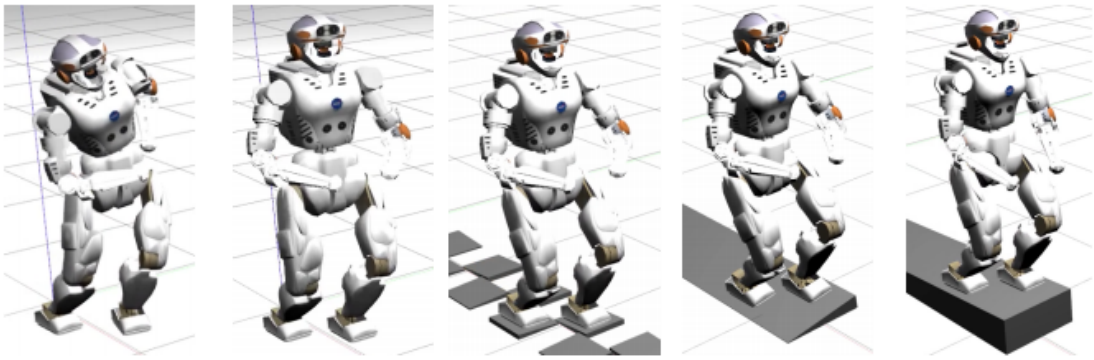


Figure 1.1: The Valkyrie robot walking trail using updated parameters.[25]

intermediary stage for advance visual processing tasks like object recognition and locomotion in humans [15].

The goal of this project is to create a system to approximate 3D surfaces from point clouds. This system is expected to produce C^2 continuous surface approximations of a point cloud. The generated representation is intended to be used by other robot control systems to run optimisation algorithms that can calculate optimal trajectories for locomotion using second order derivatives. Therefore, system has to be designed for easy use and integration.

1.2 Summary of Work Done

- Literature Review
  - Based on the objectives of the project, a criteria for evaluation of various surface representation schemes was defined. This criteria is described in section 3.2.
  - A thorough analysis of two popular surface representation schemes was conducted. These schemes included Polygon Meshes and Parametric surfaces. The analysis of each model defined in these representation schemes included a description of the model and an evaluation based on above mentioned criteria. To summarize the conclusion of the analysis, Rational B-Spline curves were chosen to perform surface approximation in the system.
- Problem Definition
  - The problem of surface approximation from point clouds was formally defined and a least squares solution to the problem was proposed.
  - Various optimisation objectives for the least squares solution were analysed, implemented, tested, and evaluated.
- System Design
  - The requirements of the system and its expected outputs were captured. This was done by consulting with a PhD student at the University of Edinburgh who was to use this system in his research.
  - Some key software requirements of the system were identified. These were primarily influenced by ease of integration with other systems and the ability to efficiently implement and test various parts of the least squares fitting algorithm. An Object Oriented design of the system was chosen.
- Implementation
  - Various useful libraries were identified.
  - The designed system was implemented as a python package.
- Evaluation

- The system was evaluated for its speed and accuracy of fitting.
- The various optimisation objectives used were also evaluated.

1.3 Dissertation Structure

This document captures the course of the complete project. The document has been structured in a way that covers the various aspects of the project incrementally, making them easier to follow.

Chapter 2 first describes how point clouds are created and the various sensor technologies used for the purpose. This is followed by a description of algorithms used for modelling an environment using point clouds.

Chapter 3 covers the various ways to surface representation schemes. First, the need for a surface representation is described. Second, an evaluation criteria for the evaluation of representation schemes is created. This is followed by an analysis of the representation schemes. Finally, the chapter is concluded by choosing B-Spline surfaces as the mode for representation for this project.

Chapter 4 formally defines the problem of fitting B-Spline surfaces to a point cloud. This is followed by a description of various optimisation objectives used for fitting. Finally, the surface fitting algorithm is defined and various important aspects of it are described.

Chapter 5 describes the design and implementation of the system.

Chapter 6 shows the results of fitting on some test cases. This is followed by an interpretation of the results.

Chapter 7 summarizes the results of evaluation and discusses the possible future work.

Chapter 2

Point Clouds

In this chapter we analyse how point clouds are created and briefly examine various technologies available for the same. Further we discuss algorithms used to combine point clouds captured from different poses and angles to form a single point cloud as a representation of the environment. Finally, we discuss some noise-removal techniques.

2.1 How to make your Point Cloud

Point clouds are 3-Dimensional data points collected by scanning an environment using a depth-sensor. This is similar to taking a simple photograph where we use visible electromagnetic waves reflected by the objects in the environment to create a representation of the environment. Photographs however, project this information onto a 2D plane losing the depth information which encodes the information about the distance of these objects from source(camera). This where point clouds are different from photographs.

A 3D depth sensor emits a signal and detects distortion in copies of the signal received by the sensor due to reflections from external objects. This distortion can be in the form of change in phase or amplitude of the signal and can be used to estimate the distance of the object that created this distortion from the sensor (Figure 2.1)[7]. Depending on the kind of signal used, other information like the colour of the object can also be gathered along with spatial(horizontal and vertical displacement) and depth information. For a typical 3D Depth sensor, these estimations are done for millions of points per second which results in a large collection of 3D points or point-clouds.

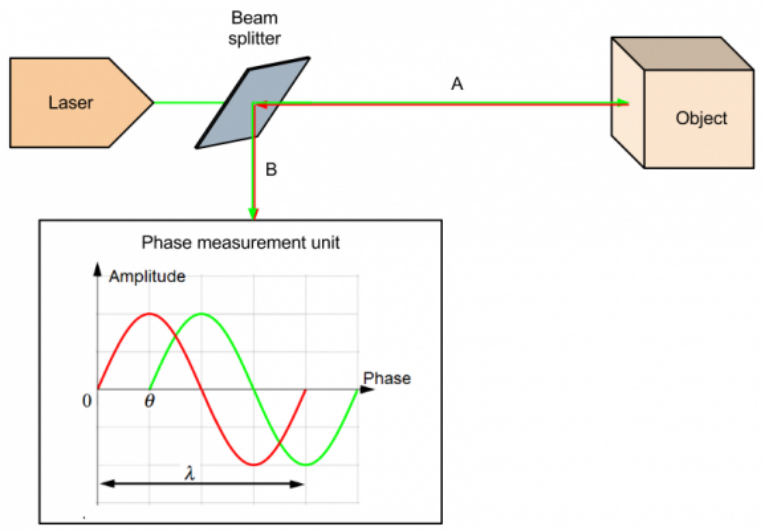


Figure 2.1: The figure shows a diagram describing the detection of an object using lasers. The distortion used for detection is phase shift caused by reflection. [1]

2.2 3D Sensors

2.2.1 RGB-D Sensors

RGB-D sensors consist of two camera sensors. One of which is a typical RGB camera and the second one is a specialised sensor which projects infrared scatter patterns on to the environment to observe it. The sensory information from these two sensors are combined through epipolar geometry and triangulation [7]. The disadvantages of these senors are that there is a lot of post processing involved in extracting the actual depth information. Also, the 2 sensor apparatus suffers from occlusion problems where the performance of the sensor drops when an object is only visible to one of the cameras. Asus XionPro is an example of such sensors (fig. 2.3a).

2.2.2 Time-of-Flight (TOF) Sensors

Time of flight refers to the time taken by an emitted signal to return to its source after reflection with objects. The time of flight value is then used to estimate the distance of an object from the source. This principle is used with different kinds of signals. The quality of the estimates depend on the kind of signal used [7]. Some TOF sensors include:

- **Ultrasonic Sensors**: These sensors use sound waves as the signal for TOF detection. These sensors are very cheap to highly available. However, due to the diverging property of sound waves, the distance estimates are not very accurate for long ranges.

[1] https://home.roboticlab.eu/en/examples/sensor/lidar

(a) Asus XionPro RGB-D sensor: Used for data collection.

(b) Image showing various sensors in a RGB-D camera and the output. [20]

Figure 2.2: RGB-D cameras.

- **TOF-Cameras**: These sensors use an array LEDs or laser diodes to illuminate the environment. Like a camera, a lens and an image sensor is used to measure the TOF of the received signal. Theoretically these sensors have a range in Kilo Meters. However, most common cameras like the Microsoft Kinect V2 have a range in tens of meters with a resolution of 1cm. The advantages of such sensors is that the error in modelling increases linearly with distance. Which is much better than the quadratic increase seen in RGB-D sensors [7].
- **LIDAR: Light Detection and Ranging.** LIDAR employs beams of lasers and a scanner to perform TOF sensing. Using beams of laser provides these sensors with range of a few kilo meters. Based on the scanning mechanism these sensors can produce a surrounding view from the source mapping 360°. However, such sensors generally have slower frequency of operation and like the Velodyne [8], can be very expensive.

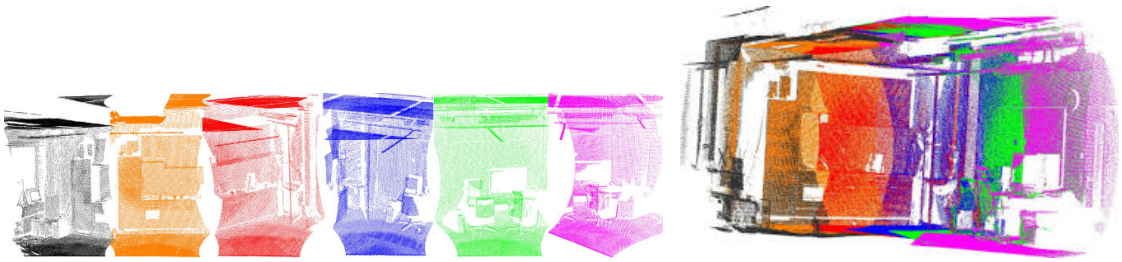
2.2.3 Noise

As discussed above, 3D sensors depend on signals for gathering data about the environment. This involves reliable detection of the reflected signals. Since the signals used may already be present in the ambient environment, many signal processing techniques are used to perform detection reliably. The quality of detection is quantified using a metric called Signal-to-Noise Ratio (SNR).

2.3 Environment Modelling

The sensors discussed above can be used to gather a large amount of observations(point clouds) of a target environment. However, depending on the structure of the environ-

ment and the capabilities of the sensor, these observations when taken individually may only represent a small part of the target environment. This gives rise to the problem of Point Cloud Registration which is the process of producing a single representation of the environment by aligning and combining multiple observations of an environment.



(a) Multiple scans of the same region. (b) Result of registration.

Figure 2.3: An example of point cloud registration. (Images taken from PCL website[2]

2.3.1 Point Cloud registration

Point Cloud registration generally involves computation of features capable of uniquely identifying a point(eg. SIFT[3] and FAST[4]). This is done for each point in the whole point cloud dataset after which a feature matching algorithm is used to find correspondences between two point clouds. A correspondence refers to finding the same point in both the point clouds at different location. The approach to find matches can consist of brute force matching or random sampling methods such as Random Sample Consensus[5] algorithm.

Later, a transformation tensor is computed such that applying the transformation (rotation and translation) on a source point cloud would perfectly align the correspondences between it and the target point cloud. This method is an offline method. Therefore, it does not utilise additional information such as the position of the sensor when the observations were collected.

2.3.2 Simultaneous Localisation and Mapping

Simultaneous Localisation and Mapping or SLAM is an online mapping process which involves an agent present in an unknown environment. Since the environment is unknown, the agent is uncertain about its own position in the environment. The SLAM approach tries to solve the problem of mapping the agent's environment by simultaneously estimating the agent's position with respect of its environment(tracking) and producing a representation of the environment with respect to the agent's observations(mapping).

[2]http://pointclouds.org/documentation/tutorials/registration_api.php
[3]https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
[4]https://en.wikipedia.org/wiki/Features_from_accelerated_segment_test
[5]https://en.wikipedia.org/wiki/Random_sample_consensus

To put the process in context of this project, An agent may be legged robot observing its environment using one of the 3D depth sensors discussed above. Hence the observations generated are point clouds. Since point clouds typically consists of points in the order of thousands, the observations generated are considered dense and the problem is called Dense SLAM.

Most solutions to the SLAM problem proposed by robotic researchers use probabilistic models like the Kalman Filter [23]. In such approaches, the tracking and mapping phases are tightly linked together. The representation of the environment is stored in the agent's state. As the agent makes more observations, the state of the agent is updated. These updates take into account the observation errors which is modelled by maintaining a co-variance matrix of the state [14] [10]. Research done in [10] demonstrates such a system that performs dense SLAM using the Extended Kalman Filter.

However, advancements in the field of parallel processing has prompted researchers to decouple the tracking and mapping phase in favour of enhanced computational power offered by GPUs. This approach was first demonstrated by [11] in an attempt to use SLAM in context of Augmented Reality.

The authors of [11] argued that probabilistic models based solutions to SLAM originating from robotics research are not optimal in situations where the agent is a handheld device like in the case of augmented reality. This is because the movements of a handheld devices are quick and random while robots typically have a more constrained movement. ElasticFusion [24] and KinectFusion [16] are examples of some algorithms that run tracking and mapping processes parallel to each other on a GPU to provide very good realtime surface mapping results.

2.4 Data collection

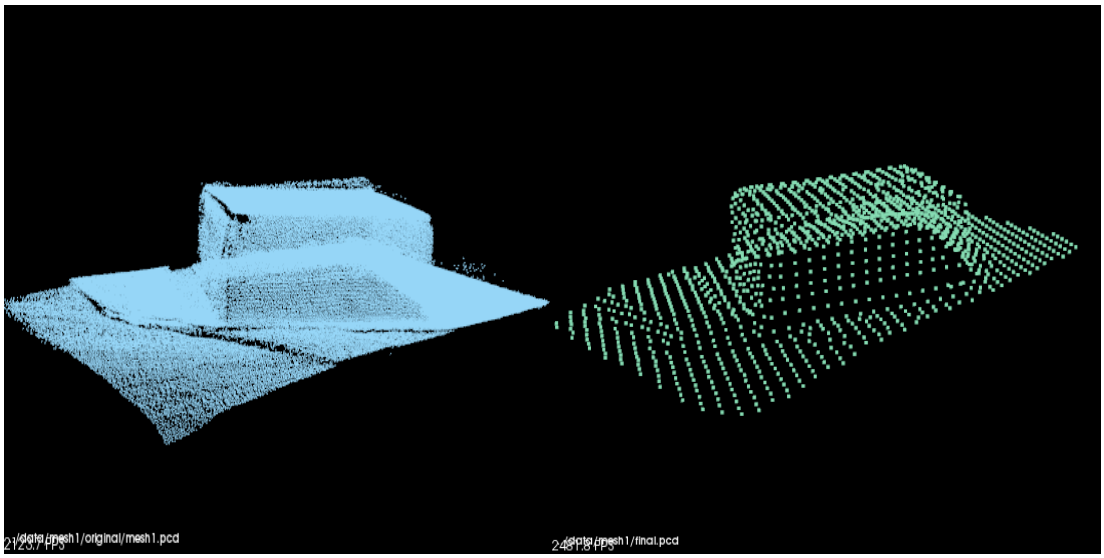
The Asus XionPro was used to perform data collection experiments. This was because a complete apparatus with the device set up with ElasticFusion[6] was readily available at the University Of Edinburgh. Figure 2.4 and Figure 2.5 show the point clouds collected for experimentation. Boxes and books were used to create scenarios consisting of surfaces with different levels of curvature.

Note: The data collection experiments were not conducted by me. I was given the point clouds along with a description of the device and algortihm used.

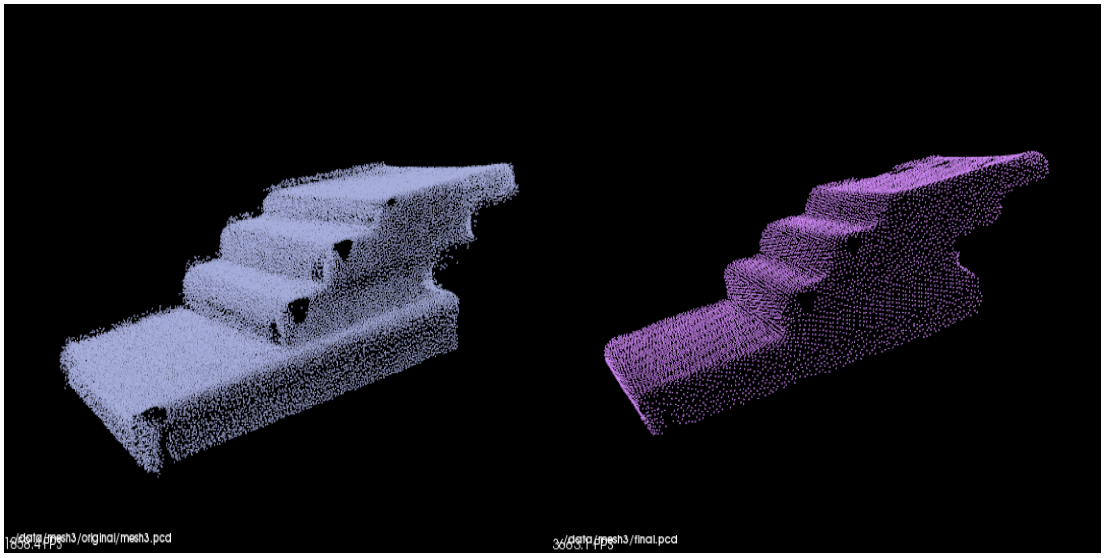
2.5 Noise Removal

The point-clouds generated using the above methods are noisy. The source of the noise can be misaligned point cloud registration and low SNR of observations collected by

[6] https://github.com/mp3guy/ElasticFusion

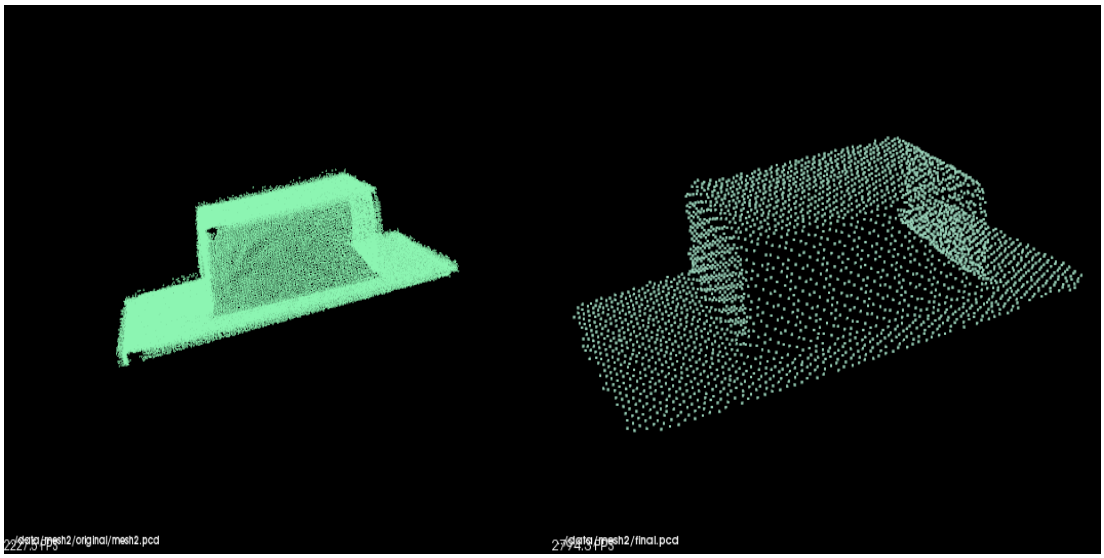


(a) Unprocessed-1 (b) Processed-1

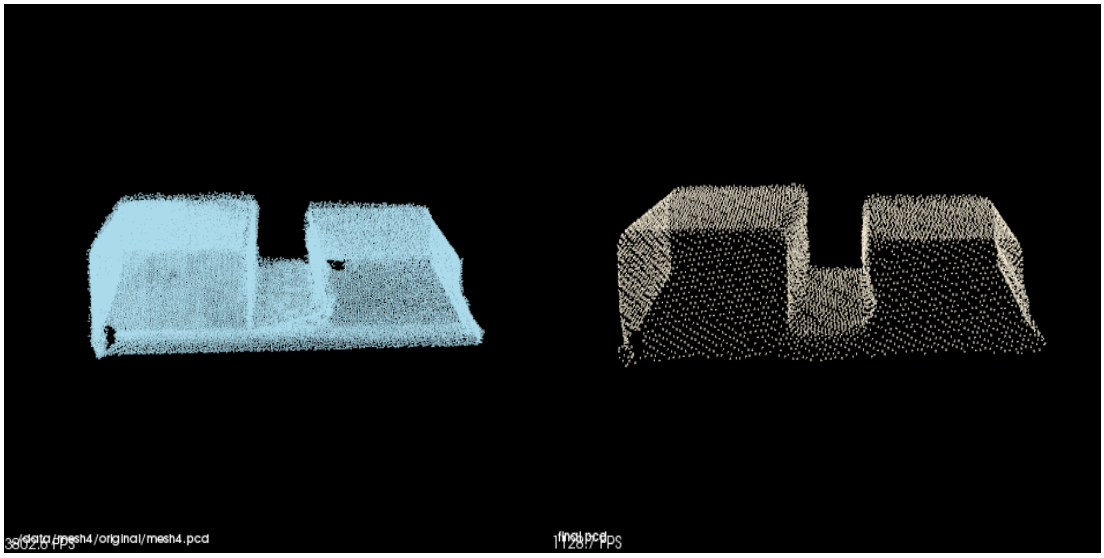


(c) Unprocessed-2 (d) Processed-2

Figure 2.4: Point clouds: Left figures show unprocessed point clouds and the right figures show the corresponding processed point clouds. (a) total points: 130261 (b) voxel-size = 0.02; total points: 1447 (c) total points: 52240 (d) voxel-size 0.005; total points: 10467

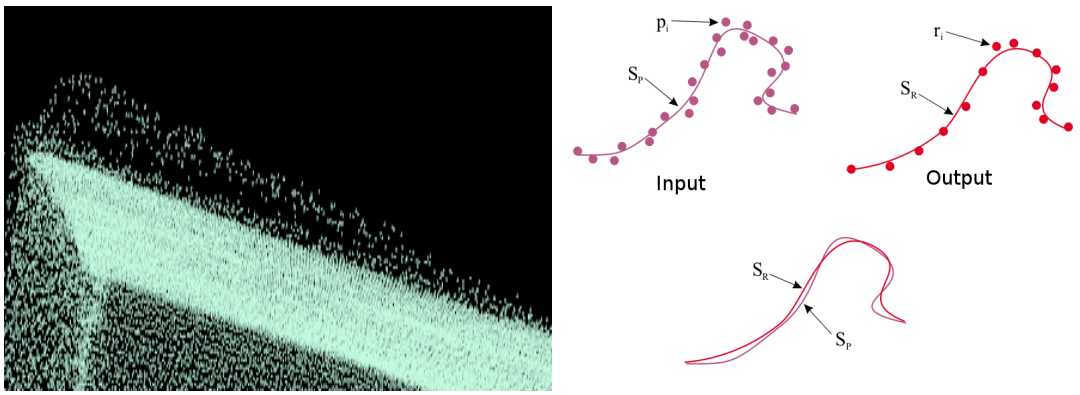


(a) Unprocessed-3 (b) Processed-3



(c) Unprocessed-4 (d) Processed-4

Figure 2.5: Point clouds: Left figures show unprocessed point clouds and right figures show corresponding processed point clouds: (a) total points: 67535 (b) voxel-size = 0.01; total points: 3204 (c) total points: 54825 (d) point cloud filtered using voxel-size=0.008; total points: 6411



(a) A noisy point cloud (b) An illustration of least squares fitting. [2]

Figure 2.6

the 3D sensors. The presence of such noise can lead to inaccuracies in the approximation of the surface. In the worst case, presence of outliers can deform the shape of the approximating surface or keep the approximation schemes used from converging.

Moreover, A point cloud can consist of more than 100K data points. Although, more data points mean that the fitting surface will better approximate the point cloud, large number of data points take longer time to process and usually result in very large system of equations that is needed to be solved for optimisation. To address the point clouds are pre-processed before performing any fitting.

2.5.1 Statistical Noise Removal

In statistical noise removal, a small locality of points (usually of fixed size) is chosen. Then the mean and the variance of the locality is used to filter the locality. This process is repeated until the whole point-cloud has been processed. This method is very effective against outlier removal.

2.5.2 Downsampling

Downsampling or Voxelisation is used to reduce the size of a point cloud. In this process, the point cloud space is divided into small cubical volumes called voxels. Then, for each voxel in the space, the points present on the inside of the voxel are used to compute a single point using a designed metric. This metric usually is the mean of all the points. The point computed is then added to the voxel and all the constituent points are removed. Large voxel sizes result in sparse point clouds.

Although very effective for size reduction, downsampling can lead to loss of information such as curvature of the point cloud. This introduces additional noise to the system. Therefore, the voxel size used for downsampling is influenced by a trade-off between speed and accuracy.

We explore this trade-off by performing tests on point clouds of different sizes generated by using different voxel size.

2.5.3 Re-sampling using Moving Least Squares Reconstruction

Another source of noise can be misalignment of two point clouds (fig.2.4a). Misalignment usually results in sections of the point cloud having redundant points representing the same underlying surface (figure 2.6a). In such scenarios a filtering technique called moving least squares [2] is used to re-sample these points from a local approximation of that section. The local approximation is generated by fitting a polynomial on to a small locality of section using least squares. The operation is performed on multiple overlapping localities sampled from the noisy section to ensure continuity.

Figure 2.6b illustrates such an example. The original points(purple) contain redundant or noisy points. The purple line forms the manifold of the underlying curve. The red points show a re-sampled representation of the purple curve. The manifold formed by the re-sampled points (red line) is the resultant approximation of the curve.

The re-sampling process removes redundant points from the curve and results in to a smoother and more uniformly sampled representation of the section.

2.5.4 Processed Point clouds

The techniques described in previous sections were applied on the point-clouds collected through experiments. This was done using the PCL library [19] which has an implementation of all the above filtering techniques.

Notice the difference between the unprocessed and processed pairs of point clouds in figure 2.4 and figure 2.5. The filtering process has removed most of the noise and produced smoother point clouds. Applying re-sampling has reduced the noise caused by misalignment in figure 2.4a. Further, down-sampling has reduced the size of the point clouds at least by a factor of 5 while maintaining essential curvature information for accurate approximation.

Chapter 3

Surface Representations

This chapter presents an analysis of various methods used to represent a surface. First the need for modelling an environment using a collection surfaces is discussed. This is followed by a discussion on the properties of a surface and the requirements a surface should satisfy to model an environment successfully. Later we analyse techniques used to represent surfaces and evaluate them against these properties.

Please note that the list of models analysed here is in no way an exhaustive list of surface representations. The representations discussed here are currently the most popular in fields like CAD and computer graphics.

3.1 Need for surfaces

In previous chapter we discussed how point clouds are created. Although a dense point cloud may contain over a million points, they are not very useful to derive useful properties of an environment. This is because: Point clouds are discrete.

A point cloud is essentially a set of discrete 3D points. Therefore the model of the environment it presents is also discrete and full of discontinuities. But since an environment is continuous, such a representation is not useful. Hence, a more continuous form of representation is needed.

A better representation for an environment are Surfaces. Surfaces are continuous and can be used to derive properties like slopes and normals at any point in the environment. Knowledge about such properties can be used to make better decision while performing complex tasks like locomotion.

3.2 Representation requirements

For the aim of this project, an ideal surface model should have the following attributes:

1. **Compact Representation.** A point cloud typically consists of hundreds of thousands of points. Hence, the ability to approximate an environment made up of such high number of points using a smaller number of parameters will be useful.

2. **Continuity.** One of the most important properties of a surface is its continuity. In mathematics, the continuity of a function is defined by the number of continuous higher order derivatives that exist for that function. A function with C^n continuity has continuous derivatives until the n^{th} order derivative.

   This is important for this project because one of the biggest advantages that visual perception brings to locomotion in robots is the ability to extract terrain properties like gradients of various regions of an environment. For instance, motion and path planning algorithms for legged robots use various optimisation methods [1] that require first or second order gradients for finding optimal locomotion trajectories.

   Some other uses of gradient information may be, distinguishing regions containing flat patches from regions containing patches with higher slope. Similarly, information about the rate of change of gradient(2^{nd} order derivatives) can be used to differentiate cliffs from slightly sloped regions.

   Hence, choosing a continuity constraint is critical for useful modelling of the environment. For this project, the continuity constraint was set to C^2 continuity. C^2 continuity ensures the existence of second order derivatives at all point of the surface. This also ensures that second order optimisation algorithms can be used with these representations.

3. **Support for High variation in curvatures**. Since the modeled environment may consist of surfaces that have localities of high variation in curvature, like edges of stairs, It is important that the surface model used is able to model such high variances robustly.

4. **Local Support.** Local support means that the a change occurring in a particular section of the modelled surface should not affect other sections of the surface. This property is desirable for applying local optimisations during surface fitting without deforming the surface.

3.3 Polygon Meshes

Polygon Meshes are widely used to represent surfaces. The mesh based representation defines a surface with a collection of vertices and polygons, usually triangles. Figure 3.1 shows a surface represented as mesh.

A key characteristic of mesh based representation is that they are constructed by extracting polygons directly from the input point-cloud avoiding any kind of optimisation algorithms like least squares. This allows them to represent surfaces with arbitrary topology.

[1] https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Dolphin_triangle_mesh.png

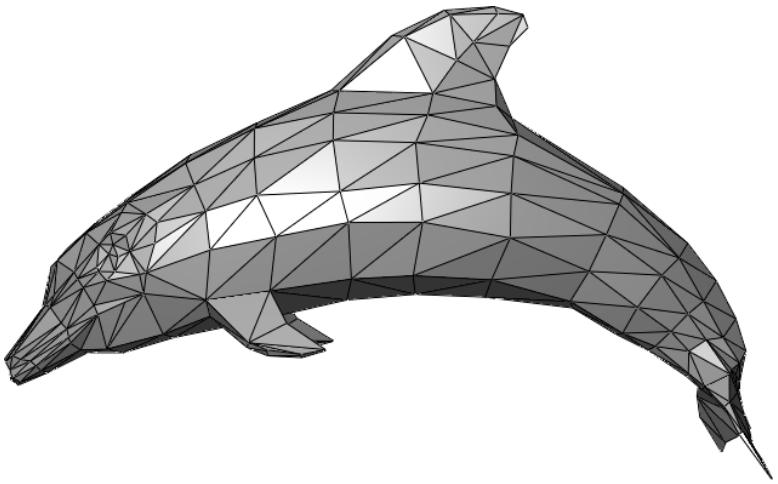


Figure 3.1: A mesh representation of a dolphin. [1]

Since meshes use polygons faces to represent surfaces, they are not inherently smooth. Therefore, after generating a mesh, repetitive smoothing operations are performed until desired level of smoothness is achieved. In this section we will briefly discuss algorithms used for generation of a mesh from a point-cloud. Later we discuss some smoothing techniques and analyse the continuity of the generated smooth surfaces.

3.3.1 Mesh Generation

Some popular methods to create meshes from a set of points are:

1. Delaunay Triangulation.
2. Marching Cubes [13].

In recent years, Delaunay Triangulation has been widely used for automatic mesh generation. As the name suggests, the algorithm uses triangulation to group the input points into a collection of triangular faces adhering to certain conditions.

In order to keep this analysis short we will not discuss the various generation algorithms. Interested readers can use the mentioned references[2] as a good place to start.

3.3.2 Smoothing

Smoothing of meshes is done using the subdivision operation. The subdivision operation recurrently divides the polygon faces in a mesh into smaller faces making the surface a little smoother after each iteration. Figure 3.2 demonstrates the subdivision operation. Applying subdivision operation on mesh infinitely guarantees a smooth limit surface. There are many subdivision algorithms. These algorithms differ from

[2]https://people.eecs.berkeley.edu/ jrs/meshpapers/delnotes.pdf

each other is in the way that they define rules for subdivision. Some of the popular subdivision algorithms are:

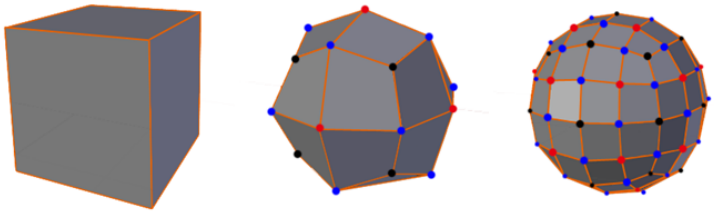


Figure 3.2: Effect of Catmull-Clark subdivison on a cube.[6]

1. Catmull-Clark Surfaces [5]
2. Loop Subdivision [12]

Interestingly, Catmull and Clark in [4] prove that the subdivision algorithm they invented results in a bi-cubic B-Spline surface. We discuss these kind of surfaces in the next section. Similar methods are used to show that most subdivision algorithms result in a B-Spline derived final surface.

3.3.3 Drawbacks

Some drawbacks of subdivision surfaces are:

- Although they are proven to demonstrate C_2 continuity [5]. There are certain points on the surface called the extra-ordinary points that only show C_1 continuity.
- The drawback of these algorithms is that they lack most of the mathematical background of the other approaches. As a result error modeling is much less rigorous, and heuristics are used to model the error. [9]

3.4 Mathematical models for Surfaces

Surfaces are well studied in mathematics. This has resulted in multiple mathematical models to represent a surface. In this section we discuss some of these models. The focus of the section will be to define parametric surfaces and analyse the various types of such surfaces. The analysis will be based on flexibility, efficiency and the continuity of the resultant surfaces.

Implicit Surface Equation

The most well known mathematical model is the implicit equation of a surface given below.

$$f(x,y,z) = 0 \tag{3.1}$$

The implicit equation tightly couples all the coordinate points on the surface(x,y, and z) together into one equation. This model works for simple and well structured surfaces like a sphere of radius r,centered at the origin:

$$x^2 + y^2 + z^2 - r^2 = 0 \tag{3.2}$$

However, for more complex surfaces, the implicit equation form is very restricting. An environment may consist of complex surfaces having high variability in different sections of the surface. To model such an environment mathematically, a more flexible model is required.

Parametric Models

Parametric models are another popular mathematical representation of a surface. To simplify the discussion we will focus on the parametric form of curves. After analysing the parametric curves, we will discuss **Tensor Product Surfaces** scheme, which allows us to represent a surface by using two parametric curves. The analysis done on curves will remain valid since the tensor product surface generated using these curves inherits their properties [17].

$$C(u) = (\mathbf{x}(u), \mathbf{y}(u), \mathbf{z}(u)) \quad u \in (a,b) \tag{3.3}$$

Equation 3.3 shows the parametric equation of a 3D curve, C(u), which is a vector-valued function of a parameter u. In parametric form, each of the coordinates of a point on the curve is represented by a separate function of the independent parameter, u. Here, the curve coordinate points x, y and z are represented by explicit functions $\mathbf{x}(u), \mathbf{y}(u), \mathbf{z}(u)$ respectively.

The parametric surface models are well covered in the NURBS book [17]. Some of the most common forms are parametrisation are discussed below.

3.4.1 Power Basis Curves.

The power basis curves are the most simplest form of Parametric curves.

An n^{th} degree power basis curve is defined as

$$C(u) = \sum_{i=0}^{n} \mathbf{a_i}\, u^i \qquad 0 \le u \le 1 \tag{3.4}$$

where, $\mathbf{a_i} = (x_i, y_i, z_i)$ are vectors. Therefore,

$$\mathbf{x} = \sum_{i=0}^{n} x_i\, u^i \qquad \mathbf{y} = \sum_{i=0}^{n} y_i\, u^i \qquad \mathbf{z} = \sum_{i=0}^{n} z_i\, u^i \tag{3.5}$$

Hence all the explicit functions are polynomials of the parameter u.

Matrix Representation

Generally, for a curve with degree n, the n + 1 functions $\{\mathbf{u}^i\}$ are called the basis (blending) functions. The set $\{\mathbf{a}_i = (x_i, y_i, z_i)\}$ are called geometric coefficients. It can be seen that the curve is a linear combination of the basis functions weighted by the geometric coefficients. This is an interesting way of looking at the parametric surfaces because this allows to represent eq. 3.4 as a matrix multiplication.

$$C(u) = \begin{bmatrix} u^0 & u^1 & .. & u^n \end{bmatrix} \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ & .. & \\ x_n & y_n & z_n \end{bmatrix} \tag{3.6}$$

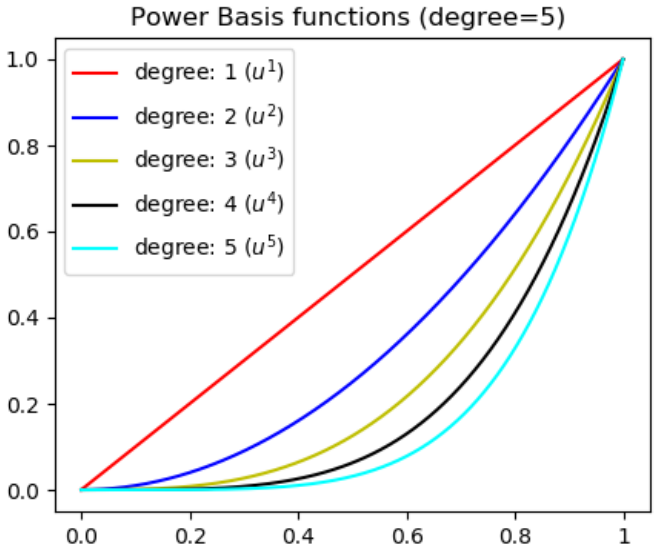


Figure 3.3: A plot showing the value of power basis functions from degrees 1 to 5.

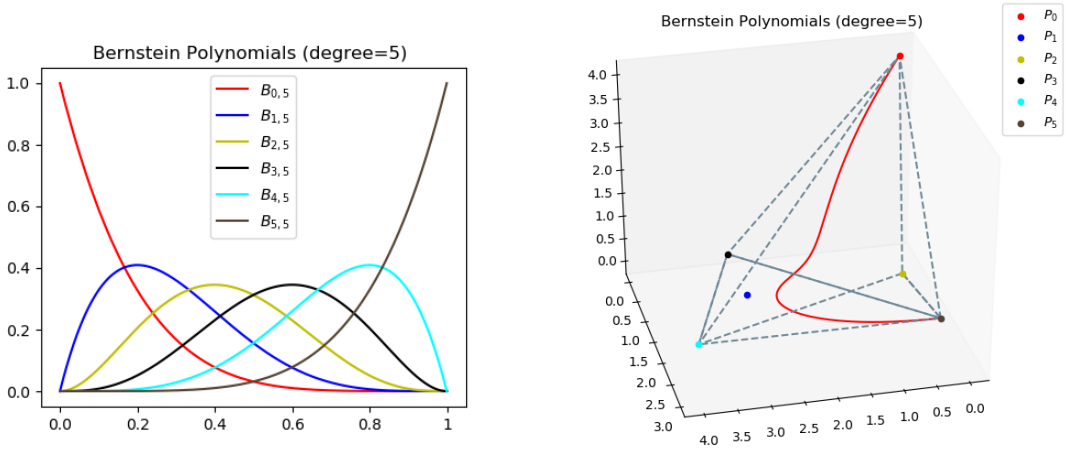
It can be observed that eq. 3.6 is of the form $\mathcal{C}(u) = \mathbf{u}^\mathbf{T}\,\mathbf{a}$. Therefore, for fixed basis function vector $\mathbf{u}$ and a target curve $\mathcal{C}$, optimisation methods can be used to find a set of geometric coefficients $\hat{\mathbf{a}}$, such that the residuals $|\mathcal{C} - \mathbf{u}^\mathbf{T}\,\hat{\mathbf{a}}|$ is minimised. The optimisation process is discussed in chapter 4.

Flexibility.

The basis functions of this representation are polynomials, which can potentially represent very complex surfaces for a sufficiently high value n, the degree of the curve. However, in practise, to model complex curves using power basis functions requires very high values of the degree. This makes this modelling technique very inefficient.

Continuity. Using basis functions of degree $\mathbf{n}$ results in a C^n continuous curve.

Drawbacks. Some drawback of using power basis curves for representation are:



(a) Bernstein Polynomials of degree 5.

(b) Bezeir curve of degree 5.

Figure 3.4: Plots showing properties of Beizer Curves

- The parametrisation of the power basis curves is not very intuitive. It is difficult to reason about the effect of moving a vector $\mathbf{a_0} = (x_0, y_0, z_0)$ on the curve C(u).
- The algorithm used for evaluation of such curves (Horner's algorithm[3]) suffers from instability such as rounding errors for high degree evaluations [17].

3.4.2 Beizer Curves

An n^{th} degree Beizer Curve is defined as:

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u)\ \mathbf{P_i} \tag{3.7}$$

Here, $\mathbf{B_{i,n}(u)}$ are bernstein[4] polynomials. These are defined by the equation (Figure 6.4d):

$$B_{i,n}(u) = \frac{n!}{i!\ (n-i)!}\ u^i\ (1-u)^{n-i} \quad u \in [0,1] \tag{3.8}$$

The geometric coefficients used in this representation, $\{\mathbf{P_i}\}$, are called control points. For a 3D curve, the control points are a set of 3D points in space of the form (x_i, y_i, z_i). The polygon formed by these control points is called the control polygon.

This parametrisation is better than that of power basis form because it provides information about geometric properties of the curve it represents. A non-exhaustive list of properties are shown in Figure 3.4b and discussed below:

- Points $C(0) = P_0$ and $C(n) = P_n$. This provides information like the starting and ending point of the curve.

[3] https://en.wikipedia.org/wiki/Horner%27s_method
[4] https://en.wikipedia.org/wiki/Bernstein_polynomial

- The resultant curve lies inside of the convex hull of the control polygon.

Flexibilty

Since beizer curves also use polynomial basis function, they are capable of representing all the curves a power basis form can represent.

Continuity

The continuity of the beizer curve depends on the positions of the control points $\{\mathbf{P_i}\}$ [17]. Therefore, the continuity of a Beizer curve is highly variable.

Drawbacks All the control points in a Beizer curve influence all the points on the curve. This means, a change in one of the control points can change the shape of the curve completely. This introduces some problematic properties like:

- They require high degree polynomials for representing a highly variable curve.
- The continuity of the curve is highly variable. Even if uniform continuity is achieved, it is not robust against factors like, introduction of new control points or change in position of a control point.

These problems are addressed by B-Splines which are discussed below.

3.4.3 B-Splines

B-Splines are an extension of piece-wise polynomial curves called Splines. A spline divides its domain, say, $[t_0, t_m]$ into $m-1$ intervals. The values in each interval $[t_i, t_{i+1})$ is mapped to a value using a polynomial P_i (eq. 3.9).

$$S(t) = \begin{cases} P_0(t) & \text{for } t \in [t_o, t_1) \\ P_1(t) & \text{for } t \in [t_1, t_2) \\ \ldots \\ P_{m-1}(t) & \text{for } t \in [t_{m-1}, t_m] \end{cases} \tag{3.9}$$

It can be observed in equation 3.9 that two intervals $[t_i, t_i+1)$ and $[t_j, t_{j+1})$ are isolated. Changing the polynomial P_i does not affect the polynomial P_j. This property of isolation is exploited by b-splines to provide local support.

Since B-Splines are complex models, we first define and get familiar with some critical attributes and terminology related to a b-spline.

A B-Spline uses a vector $\mathcal{U}$ to represent the division of the input domain into intervals. If $\mathcal{U} = \{u_0, ..., u_m\}$, then the vector $\mathcal{U}$ is called the **Knot Vector**, the values u_i, such that $i \in [0, m]$ are called **knots**, and an interval $[u_i, u_{i+1})$ is called a knot span.

A knot vector has monotonically increasing values and it can also contain duplicates. For example, $\mathcal{U} = \{0,0,0,1,2,3,4,4,4\}$ is a valid knot vector. The number of times

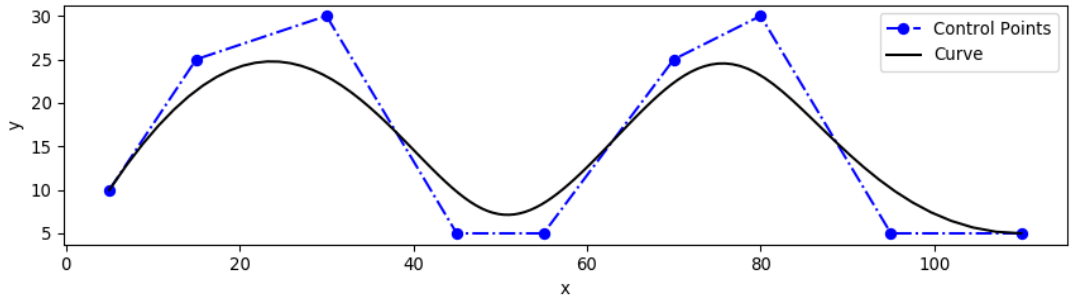


Figure 3.5: A Spline Curve and its control points.

a knot value occurs in the knot vector is called the multiplicity of that value. For example, the knot multiplicity of 0 and 1 in $\mathcal{U}$ is 3 and 1 respectively.

A knot vector uniquely identifies a b-spline curve with degree p [17]. Hence, the parameters of a b-spline curve are a knot vector $\mathcal{U}$ and the degree of the curve **p**.

A p^{th} degree B-Splines curve is defined by the equation:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)\ P_i \quad u \in [a,b] \tag{3.10}$$

$$N_{i,0} = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & otherwise. \end{cases} \tag{3.11}$$

$$N_{i,p} = \frac{u-u_i}{u_{i+p}-u_i}\ N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}\ N_{i+1,p-1}(u)$$

Where $N_{i,p}$ is the b-spline polynomial basis function and $P_i \in P$ is the set of control points. $[\mathbf{a},\mathbf{b}]$ represents the interval over which the knot vector $\mathcal{U}$ is defined. Usually, $[\mathbf{a},\mathbf{b}]$ is normalised to $[0,1]$. The b-spline basis function $N_{i,p}$ can be calculated recursively using the equation(s) 3.11.

Although, eq. 3.10 looks very similar to eq. 3.7, it is important to note the difference between the two representations.

In Beizer curves (eq. 3.7), variable n is equal to the degree of the curve and is fixed. Consequently, the number of control points that can be used in beizer curves are also fixed. Moreover, the value of the berstein polynomial functions depend on the degree **n** and the parameter value **u**. Since the degree of a curve is fixed, the value of the basis functions is always the same for a parameter and degree value.

In the case of B-Splines on the other hand, variable p is used to denote the degree of the curve and n is used to represent the total number of control points which has a minimum value of p+1 and can be increased as required. This gives more flexibility to the shape of the control polygon created.

Further, like the bernstein polynomial functions, the value of the B-Spline basis functions depend on the parameter u, and the degree p of the curve. However, the value of the b-spline polynomial also depends on the knot span, $[u_i, u_{i+1})$ that the parameter u belongs to (eq. 3.11). This results in an interesting property:

- The value of $N_{i,p}(u) = 0$ if u is outside the interval $[u_i, u_{i+p+1})$.

  This is significant because this doesn't allow a change made in a span $[u_i, u_{i+1})$ to affect more than **p** other adjoining segments. This gives b-splines the local support property.

The continuity of the surface represented by a B-Spline depends directly on the continuity of the b-spline polynomial basis functions, $N_{i,p}$ [17]. If we observe eq. 3.11, we see that the b-spline basis polynomial of degree **p** is a summation of two b-spline basis polynomials of degree $\mathbf{p-1}$.

Further, it can be proved using induction that the all the b-spline basis functions are non-negative [17]. Therefore, it implies that within a knot span(excluding the knots), the p^{th} degree b-spline basis function can be differentiated **p** times. Consequently, a **p** degree b-spline has C^p continuity inside of the knot spans.

The continuity at the knot points for the same curve is defined as C^{p-k}. Here, **k** is the multiplicity of the knot and **p** is the degree of the curve. Therefore to attain C^2 continuity using b-splines, the degree of the curve must be set to $2+k$, where **k** is the maximum multiplicity of a knot. The value of k is usually 1. Therefore, cubic b-splines are sufficient to represent any curve with C-2 continuity.

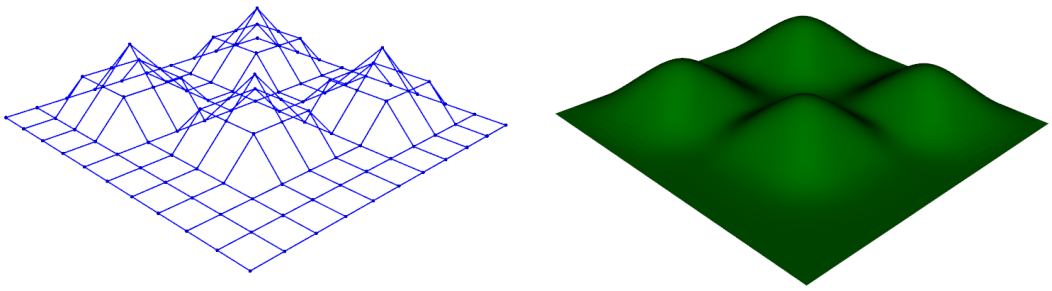
Non-Uniform Rational B-Spline

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)\ w_i\ P_i}{\sum_{i=0}^{n} N_{i,p}(u)\ w_i} \quad u \in [0,1] \tag{3.12}$$

NURBS are a generalisation of B-Splines that have slightly better properties. The main differences between the two are:

1. NURBS are Rational. This means that instead of polynomial basis functions, NURBS basis functions are a ratio of polynomials (eq. 3.12). This gives them the ability to represent closed form surfaces like spheres and cylinders. [17]
2. NURBS work in homogeneous coordinate system. This means that each control point in a NURBS curve has a weight associated to it. The weights can be used to increase or decrease the influence of a control point over the resultant curve. Setting these weights to 1 results in a Rational B-Spline curve.
3. NURBS usually have non-uniformly distributed knots. This can be used to match the distribution of the target points. This helps in providing local support.

From here on, when we refer to B-Splines, it will implicitly be a Rational B-Spline, or a NURBS with weights = 1.



(a) Control Points (Geometric Coefficients) of a B-Spline surface.

(b) Resultant B-Spline Surface.

Figure 3.6: An example of B-Spline Tensor Surface.

3.4.4 Tensor Product Surfaces

As discussed earlier, The parametric curve C(u) is a vector-valued function of one parameter. It maps a straight line segment into 3D euclidean space using explicit functions.

Similarly, a surface is vector-valued function of two parameters u and v. It maps a region $\mathcal{R}$ in the **uv** plane to the 3D euclidean space. Hence a surface can be represented using the following equation:

$$S(u,v) = (\mathbf{x}(u,v), \mathbf{y}(u,v), \mathbf{z}(u,v)) \quad (u,v) \in \mathcal{R} \tag{3.13}$$

Such a surface can be constructed using the tensor product scheme. The tensor product scheme is widely used and is a quite popular approach to create parametric surfaces [17].

$$S(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n} f(u)\, g(v)\, b_{i,j} \quad \text{where} \quad \begin{cases} b_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \\ 0 \leq u, v \leq 1 \end{cases} \tag{3.14}$$

A Tensor Product Surface is essentially a bidirectional curve. Similar to the curves discussed above, these are created using basis functions and geometric coefficients. The basis functions used are bivariate and can be constructed by taking the product of two univariate basis functions. The resultant surface has $n \times m$ geometric coefficients where n and m are the total number of geometric coefficients for each of the constituent curves.

Figure 3.6 shows a b-spline surface generated using using the tensor product scheme. Notice that the control points are arranged in a grid structure. This is because the surface is represented as a tensor product of two curves(one for each axis).

3.5 Evaluation

This section presents an evaluation of the discussed representations based on the requirements stated in section 3.2.

3.5.1 Meshes

Compaction

Polygon meshes use polygons generated directly from the input point-cloud to represent a surface. Since this technique requires to hold information like the vertices of the surface and the relationship amongst them (edges of a polygon). The final representation of the surface is not compact.

Continuity

As discussed, meshes are not inherently smooth. Although the limit surfaces created by subdivision are provably smooth, such surfaces do not show strict C^2 continuity.

Support for High Variation in curvatures.

Meshes are capable of modelling surface of any topology. However, they do not robustly capture the curvature of a surface smoothly.

Local Support

Since a region of the surface represented by a mesh only depends on the polygon faces belonging to that region, meshes provide excellent local support.

3.5.2 Parametric Surfaces

The discussion in section 3.4 suggests that B-Splines are the most powerful form of parametric surfaces. Hence, we evaluate Parametric Surface representation based on the properties of B-Splines.

Compaction

Since parametric surfaces define a surface as a linear combination of certain basis functions, a surface can be efficiently represented by set of geometrical coefficients. This results in a compact representation.

Continuity

B-Splines use piece-wise higher order functions to represent a surface. Since the continuity of a b-spline surface depends upon the continuity of the b-spline basis functions, the constraints on the continuity of the modelled surface are directly handled by the underlying mathematical model. In other words, choosing an optimum degree for the surface guarantees certain continuity properties.

Support for High Variation in curvatures.

The piece-wise nature of b-splines can capture regions of high-variations by refining the knot vector. One drawback of this approach is that refinement of the knot vector increases the number of control points (increasing the geometric coefficients needed for representation).

Local Support

For a degree p surface, only p+1 control points influence a region of the curve belonging into a knot span. This enables local support in b-splines.

3.5.3 Conclusion

Based on the analysis of the various surface representation techniques, (Rational) B-Splines were chosen to implement the system for approximating surfaces.

Chapter 4

Surface Fitting

This chapter presents a description of surface fitting problem. First we define the fitting of B-Splines to a point cloud as an optimisation problem. This is followed by a discussion on the optimisation objectives commonly used for surface fitting. Finally the core surface fitting algorithm is described.

4.1 Problem Definition

As discussed in section 3.4.1, parametric surfaces are represented using basis functions and geometric coefficients. They can be represented in vector form as :

$$\mathcal{S} = \mathcal{B}(u,v)^T\ \theta \tag{4.1}$$

Where, $\mathcal{B}(u,v)$ is a vector function generating basis function values using parameters **u** and **v**, and θ is a vector containing geometric coefficients(control points). The surface is a linear combination of $\mathcal{B}(u,v)$ weighted by θ.

In the case of B-Splines, the parameters **u** and **v** are normalised to $[0,1]$. Therefore, for constant knot vectors ($\mathcal{U}_u$ and $\mathcal{U}_v$), the value of the b-spline basis function always remains the same for the same parameters (u_i, v_i). Hence, the equation 4.1 can be re-written as:

$$\mathcal{S}(\theta) = \mathcal{B}^T\ \theta \tag{4.2}$$

Now, Given a point cloud $\mathcal{X}$ of size N, with points $\mathcal{X}_i$, $i \in [0,N)$ we want to find geometric coefficients θ of a parametric surface $\mathcal{S}$, such that the term $\mathcal{E}$, represented by eq. 4.3 is minimised.

$$E(\theta) = \frac{1}{N}\sum_{i=0}^{N} \mathcal{D}(X_i, \mathcal{S}(\theta))^2 \tag{4.3}$$

4.1.1 Least Squares Fitting

In equation 4.3, the term $\mathcal{E}$ is the mean squared value of an objective function $\mathcal{D}$, that represents the distance of the point X_i from the surface $\mathcal{S}$. The reason behind using this notation is to have the flexibility of choosing an appropriate objective function for optimisation.

Since $\mathcal{E}$ is the summed square of the distance, the value of $\mathcal{E}$ can be minimised using least squares approximation. Using normal equation to solve the least squares approximation for $\mathcal{E}$ results in solving the system of equations, $\mathbf{B}\,\hat{\theta} = \mathbf{X}$. The solution to which is:

$$\hat{\theta} = (\mathbf{B}^T\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{X}) \tag{4.4}$$

where $\mathbf{B}$ are the b-spline basis functions, $\mathbf{X}$ is the point cloud and $\hat{\theta}$ are the new control points.

In the next sections we discuss the various optimisation objectives that are used for surface fitting.

4.2 Optimisation Objectives

4.2.1 Point Inversion

Earlier, We defined $\mathcal{D}$ as the function measuring the distance between a 3D point X_i and the surface $\mathcal{S}$. To calculate this distance, we need a point X_i^s, on the surface $\mathcal{S}$, that is closest to the point X_i. We will refer to X_i^s as the foot-point of X_i on $\mathcal{S}$.

The process of calculating X_i^s, requires finding of parameters (u_i, v_i) such that evaluating $\mathcal{S}$ at (u_i, v_i) using θ results in X_i^s. This is called Point Inversion.

$$X_i^s = \mathcal{S}(\theta) = \mathcal{B}(u_i, v_i)^T\ \theta \tag{4.5}$$

We use newton iteration to find the parameters (u_i, v_i) such that the distance $|X_i - \mathcal{S}(u_i, v_i)|$ is minimised. The algorithm to do so is described in the NURBS book [17].

Once the parameters (ui, vi) are computed, they can be used to find the tangents and normal at the point X_i^s. This is required for some of the optimisation objectives discussed below.

4.2.2 Point Distance Minimisation (PDM)

Figure 4.1 shows one of the most common ways to define $\mathcal{D}$ is to define it as the euclidean distance between the points X_i and X_i^s.

$$\mathcal{D}(\theta) = ||X_i - X_i^s|| \qquad \text{where } X_i^s = \mathcal{S}(u_i, v_i, \theta) \tag{4.6}$$

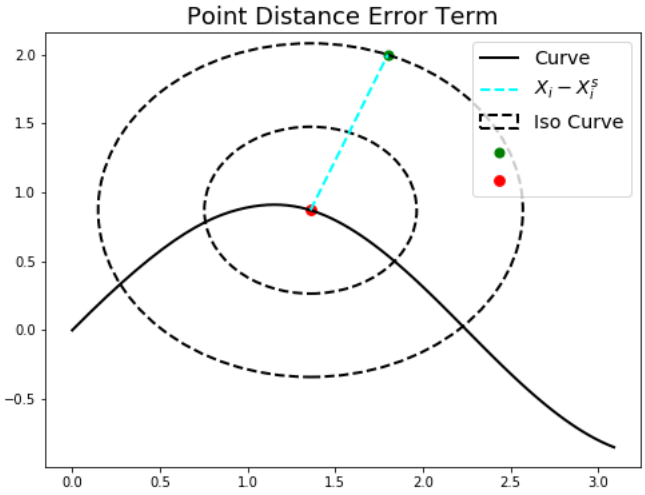


Figure 4.1: Example of Point Distance Estimation

Equation 4.3 now becomes:

$$E(\theta) = \frac{1}{N} \sum_{i=0}^{N} || X_i - X_i^s ||^2$$

This optimisation objective is called Point Distance Minimisation(PDM). This objective is simple to implement and consequently, is very popular in CAD industry [22]. However, Wenping Wang in [22] argued that, although simple to implement, the point distance function does not capture the distance between the points accurately. This is because PDM is only based on the foot-point X_i^s, and X_i^s is dependent on θ (eq. 4.5). Since θ is subject to optimisation, X_i^s is variable and changes after every iteration.

Therefore, the optimisation converges slowly.

4.2.3 Tangent Distance Minimisation (TDM)

In Tangent Distance Minimisation(TDM), the distance function, $\mathcal{D}$ is defines as:

$$\mathcal{D}(\theta) = (X_i - X_i^s)^T N_i \tag{4.7}$$

Where, N_i represents the unit normal to the surface, $\mathcal{S}$ at point X_i^s.

Essentially, TDM tries to minimise the distance between the point X_i and the moving line L_i, which is the tangent to surface $\mathcal{S}$ at point X_i^s (figure 4.2). TDM usually converges faster than PDM, however it is highly unstable in regions of high curvature.

In [22], the authors show that the instability of the TDM is due to the fact that it is an approximation of Gauss-Newton[1] iteration with an excessively large step size. The

[1] https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm

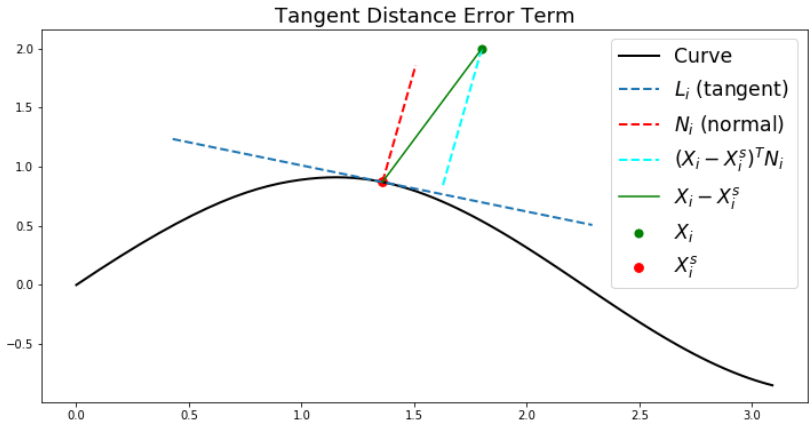


Figure 4.2: Example of Tangent Distance Estimation

large step size is caused due to omission of important curvature related information in the true Hessian Matrix. The authors later propose an extension to the TDM objective function that incorporates the curvature information. The proposed extension is called Squared Distance Minimisation and it is described below.

4.2.4 Squared Distance Optimisation (SDM)

In [22], Wang Weping and others proposed squared distance optimisation as an algorithm which converges faster than PDM but is also more stable than TDM in regions of high curvature. They do this by introducing curvature information into the TDM objective function by modifying it as shown in eq. 4.8. The term ρ represent the radius of curvature of the surface at $\mathcal{X}_i^s$ and the term d represents the signed distance between $\mathcal{X}_i^s$ and $\mathcal{X}_\rangle$, where the sign is positive if the curvature ρ and $\mathcal{X}_i$ are in the same direction and negative if the opposite.

$$\mathcal{D}(\theta)^2 = \begin{cases} \frac{d}{d-\rho}[(X_i - X_i^s)^T T_i]^2 + [(X_i - X_i^s)^T N_i]^2 & d < 0 \\ [(X_i - X_i^s)^T N_i]^2 & 0 \leq d < \rho \end{cases} \tag{4.8}$$

$$\mathcal{D}(\theta)^2 = \beta\ [\ (X_i - X_i^s)^T T_{u,i} + (X_i - X_i^s)^T T_{v,i}]^2 + [(X_i - X_i^s)^T N_i]^2 \tag{4.9}$$

Considering the complexity of the term and keeping in mind that the above function was created and tested for curves (not surfaces), for which calculation of ρ is straightforward, during implementation, the factor $\frac{d}{d-\rho}$ was replaced by a constant weight, β. eq. 4.9

4.3 Core Algorithm

In this section, the core algorithm to fit a B-Spline surface to a point cloud is built and discussed. Algorithm 1 shows the pseudocode of the core algorithm highlighting important aspects of the algorithm as functions. Each of these aspects are described below:

Algorithm 1: Surface Fitting Algorithm for B-Spline Surface

Result: B-Spline fitted to the input point cloud
input: A point cloud $\mathcal{X}$ with N points
input: Error Tolerance γ
input: Max iterations k
parameter: $\mathcal{S}$: B-Spline Surface
parameter: θ: control points
parameter: loop: iteration count
parameter: ε: error

```
1  S ← InitialiseSurface(X);
2  ε ← inf;
3  loop ← 0;
4  while ε > γ & loop ¡ k do
5      θ ← GetControlPoints(S);
6      for i ← 0 to N − 1 do
7          X_i ← X [i];
8          //section 4.2.1
9          X_i^s ← PointInversion(S,X_i);
10         // add point constraint to system of eqs based
11         // on one of eq. 4.6, 4.7, 4.8
12         addConstraint(X_i,X_i^s);
13     end
14     θ̂ ← Solve(); // solve system of eqs.
15     θ ← UpdateSurface(θ̂);
16     ε ← CalcError(S(θ),X); // eq. 4.3
17     loop ← loop + 1;
18 end
```

4.3.1 Surface Initialisation

When fitting B-Spline surface to a point cloud, the rate convergence is affected by the initial position of the surface control points [22][17]. Therefore, it is important that the surface is initialised at a good position. Initialising a B-Spline surface consists specifying the knot vectors $\mathcal{U}_u$ and $\mathcal{U}_v$ and the control points θ.

Initialisation of control points

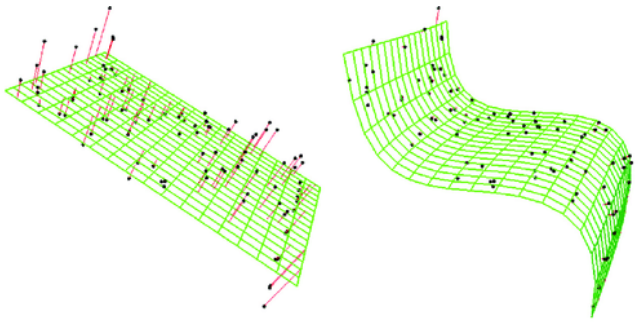


Figure 4.3: (left)A B-spline curve being initialised using PCA decomposition of a point cloud (black points). (right) The resultant B-spline surface after fitting. [18]

The most intuitive way to perform initialisation would be to have the surface aligned with the axes of maximum co-variance of the point cloud. This can be done by performing the PCA decomposition of the input point cloud and choosing the eigenvectors, v_1 and v_2, with maximum corresponding eigenvalues. The initial control points, θ can then be initialised in a grid along the axes v_1 and v_2. This intialisation works under the assumption of the existence of two major axes in the point cloud. This assumption is valid in the case of modelling an environment since this is true for most naturally existing environments. [18]

The number of control points that the initial surface has depends on the type of the fitting algorithm. Typically, there are two types of algorithms, one starts with minimum number of control points (degree of surface + 1) and increases the total number of control points using knot insertion until convergence. The other type starts with total number of points much greater than the minimum value and decreases the number of control points using knot removal in each iteration until the error value has increased over a certain threshold (or min number of control points has reached)[17].

In this project, the initial number of control points are set to the minimum value and incrementally increased upto a threshold of approximately 1200 points.

Initialisation of knot vectors

The params (u,v) of a B-Spline are normalised to $[0,1]$. Theoretically, the degree of a curve p, total control points, $n+1$ and total knots $m+1$ are constrained with the equation : $m = n+p+1$ [17].

Since a B-Spline curve is a tensor product of two curves, and parameters u and v are independent of each other. The knot vectors $\mathcal{U}_u$ and $\mathcal{U}_v$ can be generated by dividing the parameter space of u and v, respectively, into $m+1$ knots uniformly.

4.3.2 Adding Constraints

The function *addConstraints* represents the optimisation objectives discussed in section 4.2. This function essentially adds an constraint to the system of equations mentioned in section 4.1.1 based on the definition of the distance function $\mathcal{D}$.

For a point $\mathbf{X}_i$ in the point cloud, the corresponding constraint equation is given by:

$$\mathbf{B}_i \, \hat{\theta} = \mathbf{X}_i$$

where $\mathbf{B}_i$ is the vector containing the values of the b-spline surface basis function at the parameters (u_i, v_i) computed using Point Inversion of $\mathbf{X}_i$.

4.3.3 Updating Surface

Solving the equation 4.4 gives the control points $\hat{\theta}$, such that the value $\mathcal{E}$ from eq. 4.3 is minimised. The control points θ can be updated in the direction of $\hat{\theta}$ to minimise error. For a step size α, the update equation is as follows:

$$\theta = \theta + \alpha \, (\hat{\theta} - \theta) \qquad \alpha \in (0, 1] \tag{4.10}$$

Setting $\alpha = 1$ replaces θ with $\hat{\theta}$.

Chapter 5

System Design and Implementation

This chapter documents the design and implementation of the Surface Approximation System built in this project.

5.1 Design

5.1.1 Core System

The main design requirement for the core system was the ability to easily implement, test and evaluate various aspects of the core algorithm (Alg. 1) independently.

It was observed that the core system could be divided into multiple objects that interacted with each other in a fixed way. These objects were:

- **Surface**: A class that represents the surface models that can be fitted to a point cloud. All aspects of algorithm 1 relating to the parametric surface could be abstracted into the Surface class. The main task of this object would to compute a surface based on given set of parameters ((u, v) and θ).
- **Distance Metric**: A class that could represent the various optimisation objectives that could be used for surface fitting. This object of this class could use the Surface object and a given point cloud to create a system of equations(eq. 4.4) using one of the distance functions discussed in section 4.2.
- **Solver**: A class to hold all the logic related to the solving of the above generated system of equations. An object of this class would represent a solution such as LU decomposition or SVD decomposition.
- **Optimiser**: A container object that could perform the optimisation process using the above objects.
- **Configuration**: To make performing experiments easy, a configuration object could be created that specified the inputs to the Algorithm 1 and the kind of

objective function to use. The Optimiser object could then use this configuration object to run the optimisation process.

Dividing the system in this way made it easier to isolate the different aspects of the system. This made it very easy to implement various parts of the system and conduct experiments targeted at individual parts.

5.1.2 Integration Requirements

Since other robot control systems(clients) were supposed to interact with the core system. It was essential to make these interactions easy. Moreover, since the motivation behind this project was to enable visual perception, it was also important to extend the core system to add support for extracting certain attributes about the surface that the client systems needed.

To understand these requirements better, a PhD student at the University of Edinburgh was consulted. The final design requirements for the extended system were:

- Support for MATLAB. This was because the optimisation algorithms that were to use the system were running in MATLAB.
- Ability to specify tolerable error in estimates.
- Ability to extract the following attributes of the surface:
    1. The height map
    2. The tangents and normals at each point
    3. The higher order derivatives at each point.
    4. An estimated value of error in the estimate.

5.1.3 Libraries

These libraries were used during the development process. **Point Cloud Library**

The Point Cloud Library (PCL)[19] is an open-source large scale project for point cloud processing. It is completely written in C++ and contains state of the art algorithms for point-cloud filtering, feature estimation, point cloud registration and surface reconstruction.

Open3D

Open3D is another open-source library that contains point cloud processing algorithms for filtering, reconstruction and registration. The library exposes both c++ and python API and is designed for rapid development of 3D point cloud related software. [26]

NURBS-Python (geomdl)

NURBS-Python (geomdl) is an object-oriented, open-source, B-Spline and NURBS evaluation library written in pure python. [3]. The library contains implementations of numerous algorithms mentioned in the NURBS book [17], making working with B-Spline surfaces very convenient. This library also provides rendering routines to visualize the b-spline surfaces.

Numpy. Numpy is a linear algebra library for python. [21]

5.2 Implementation

The initial experiments were performed using the PCL library which is written in C++. However, the final implementation of the project was decided to be done in python. This was motivated by the ease of using python packages in MATLAB and the availability of well developed and light-weight libraries for B-Spline Surfaces [3], point-cloud processing [26], and linear algebra [21].

5.2.1 Surface Module

B-Spline Surface

The NURBS.Surface class of the NURBS-Python library was used to represent a B-Spline surface. This made the process of evaluating the 3D point, the normal, the tangents and, the higher order derivatives for a set of parameters (u, v) very convenient.

Surface Initialisation

A separate module was created for for surface initialisation. This module consisted of an abstract class that defined an interface for surface initialisation. This class was extended to implement the initialisation scheme discussed in section 4.3.1. This was done to make it easy to implement and use other initialisation schemes.

5.2.2 Solver module

A solver object was created to implement the system of equations that could be constrained and solved for optimisation. The object essentially consisted of two matrices, **B** and **X** (eq. 4.4) and an interface to add constraints to the matrices. Numpy's linear algebra functions are used to solve the system of equations.

5.2.3 Distance Metrics

The distance metric module consists of the implementations of PDM, TDM and SDM optimisation objectives discussed in section 4.2. The structure of this module was adapted from the PCL library [19] which has an implementation of the PDM and the TDM objectives.

During the implementation of the module, it was noticed that the implementation of the TDM objective in the PCL library was incorrect. Instead of optimising the equation 4.7, the implementation optimised the following equation:

$$\mathcal{D}(\theta)^2 = [\ \beta\ [(X_i - X_i^s)^T T_{u,i} + (X_i - X_i^s)^T T_{v,i}] + [(X_i - X_i^s)^T N_i]\ ]^2 \tag{5.1}$$

Which is very similar to the SDM objective ($a^2 + b^2 \neq (a+b)^2$). It was assumed that this was a new optimisation objective that was a hybrid of SDM and TDM. This function was also used in evaluation was referred to as PCL-TDM.

During evaluation it was noticed that building the constrained system of equations was a bottleneck for the whole algorithm. To remove the bottle neck, the code was extensively refactored by dividing the module into simpler routines that could be parallelized. This is discussed further in evaluation.

The biggest performance boost was achieved by parallelizing point inversion across all(8) CPU cores. The run-time of the system was reduced to a third of its original value.

Configuration

A configuration class was created that was used to specify the parameters of the optimisation process. These parameters included:

- The maximum tolerable error of the fit.
- Maximum number of iterations
- Type of optimisation objective to use
- Type of Solver to use. (This specified the algorithm to use for solving. example: LU decomposition)

Creation of this class made performing evaluation experiments easier. It also makes the system more configurable.

5.2.4 Optimiser

A class was created to contain the whole optimisation process described in Algorithm 1. This class uses an instance of the configuration object (mentioned above) to initialise the algorithm (A1) and perform optimisations.

The class also contained definitions of routines to collect metrics such as the run-time and the fitting errors. Utility methods for saving/loading objects were also implemented.

Query

The same class as above was extended to include methods that served as an interface to query the final surface. All the attributes mentioned in section 5.1.2 was supported through these functions. The client system can specify an array of 2D (x,y) or 3D

(x,y,z) points and get the information about the height (z), tangents and normal to the surface at these points. A cache was implemented to avoid running expensive computations for query consisting of the same points that had already been processed.

Chapter 6

Evaluation of System

In this chapter, the performance of the system on the collected test cases are evaluated. The evaluation is based on the convergence rate and the approximation error of the resultant surface. First the evaluation setup is described. This is followed by a series of figures showing the results of surface fitting algorithm on all the test cases. This is followed by a discussion on the results collected.

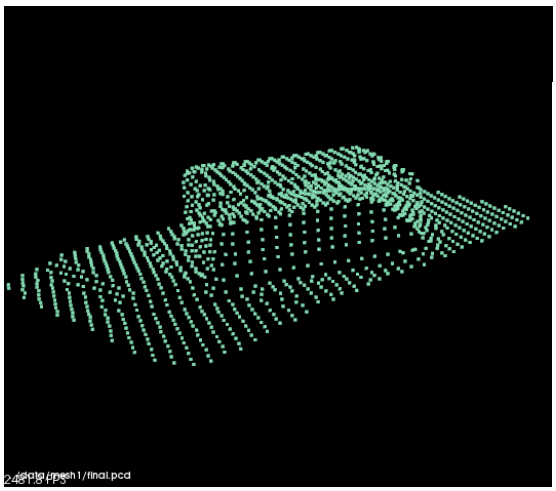
6.1 Evaluation Setup

As discussed in section 4.3.1, the algorithm is designed to start with minimum number of control points and increases the number of points over the iterations until desired error level is reached. During the evaluation phase, the maximum number of allowed control points were set to 1225. This was due to memory limitations. All experiments were conducted on a machine equipped with 8 core Intel Core i7 processors and 8GB of ram.

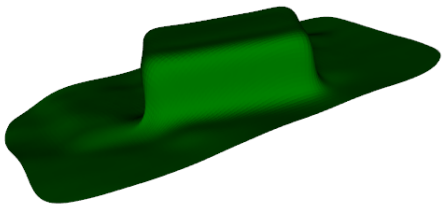
For evaluation, the system was run on 4 processed point clouds (figure 2.5) using all 4 optimisation objectives and bi-cubic b-spline planes. All experiments were run for 15 iterations.

6.2 Results

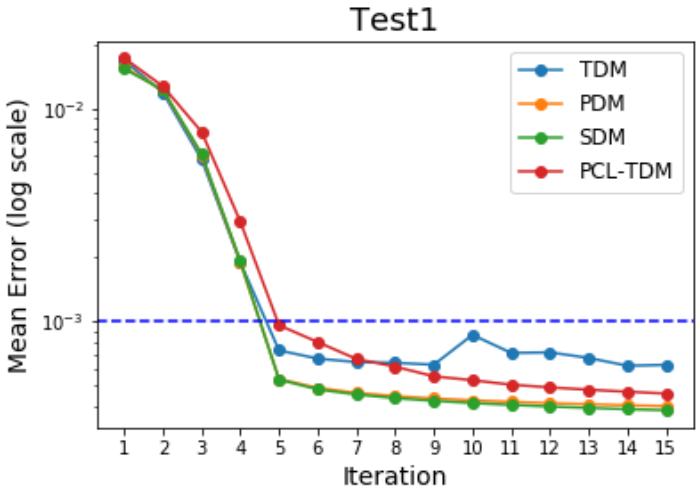
This section consists of a series of figures showing the results of performing surface fitting on the point clouds seen in figure 2.5 and 2.4. For each test case, there is an image of the input cloud, the resultant surface and plots showing comparison between the fitting error and run-time of all the optimisation objectives that were tested.



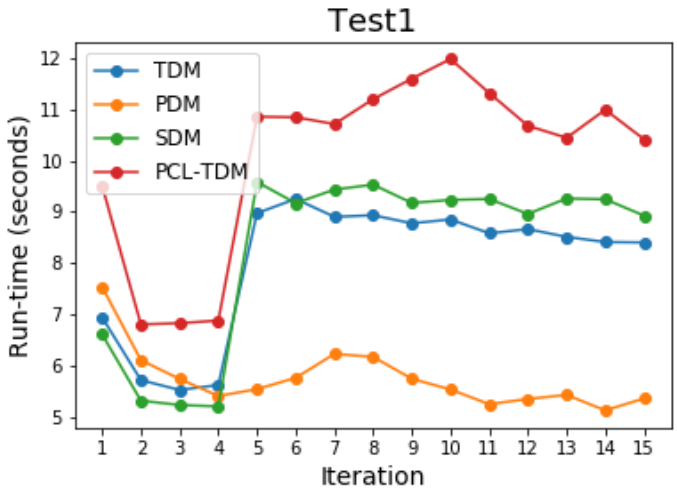
(a) Input Point Cloud



(b) Fitted Surface

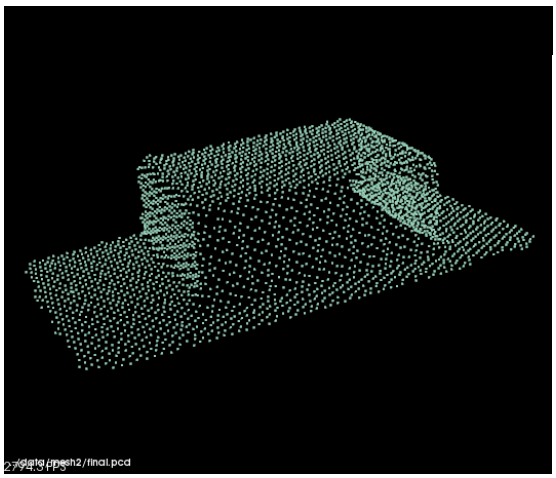


(c) Error of the objective functions

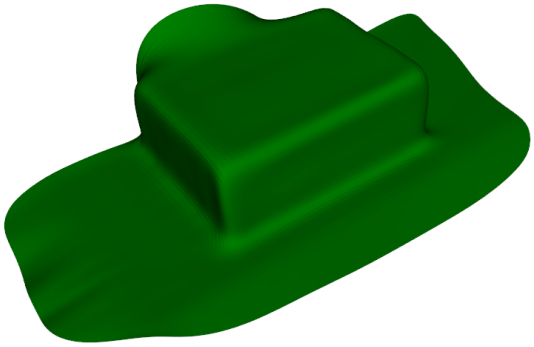


(d) Runtimes of the algorithms

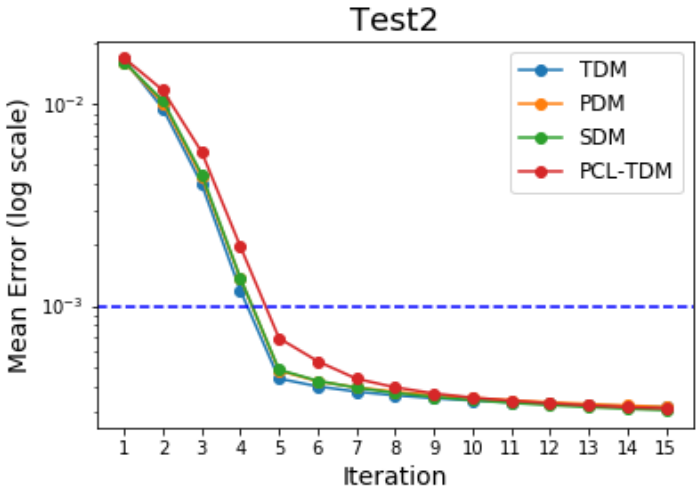
Figure 6.1: Input, output, and performance: Test-case 1. (1447 points)



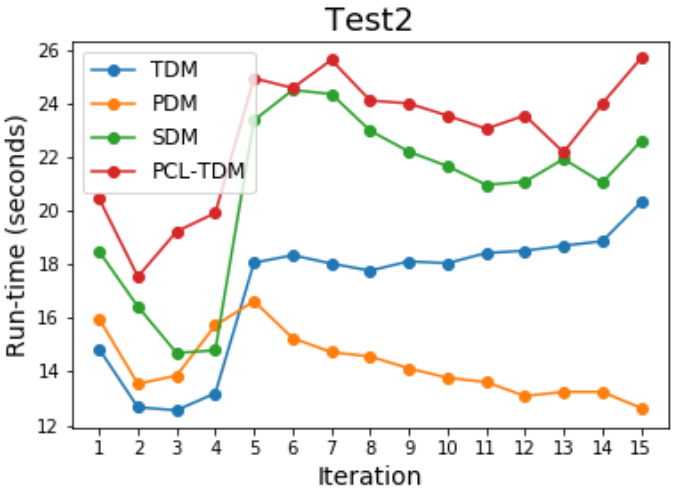
(a) Input Point Cloud



(b) Fitted Surface

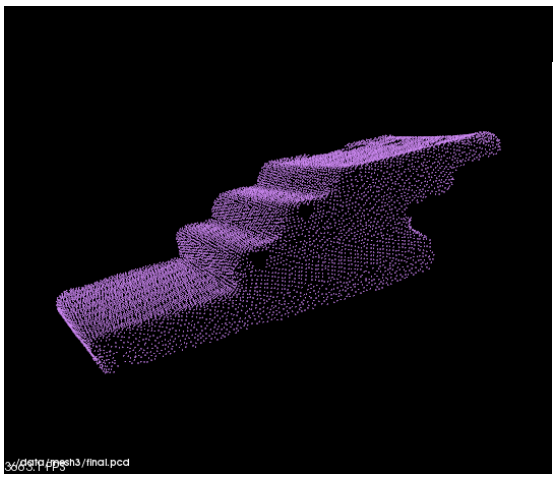


(c) Error of the objective functions

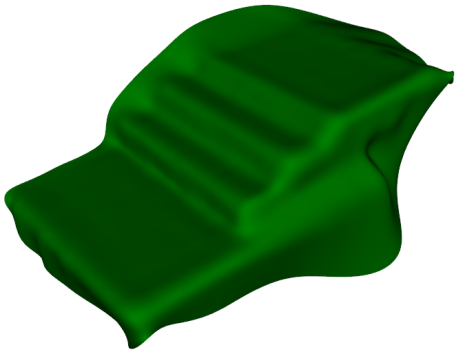


(d) Runtimes of the algorithms

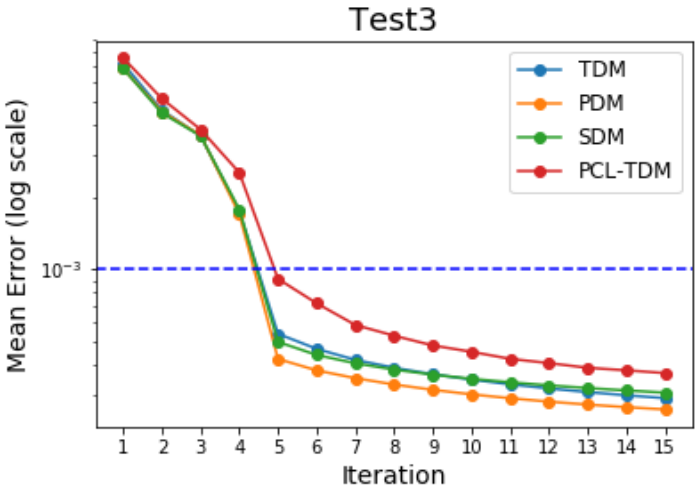
Figure 6.2: Input, output, and performance: Test-case 2. (3204 points)



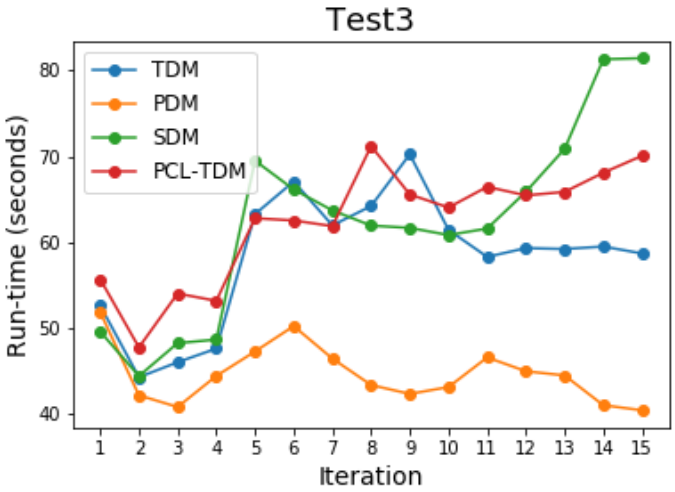
(a) Input Point Cloud



(b) Fitted Surface

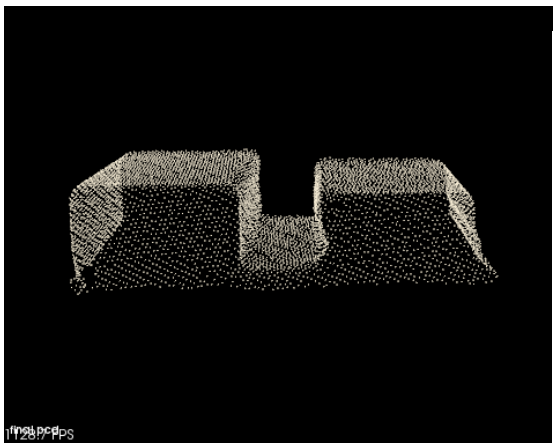


(c) Error of the objective functions

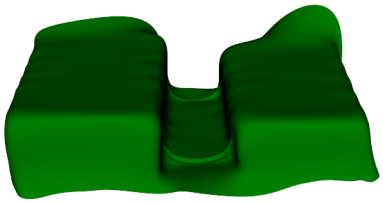


(d) Runtimes of the algorithms

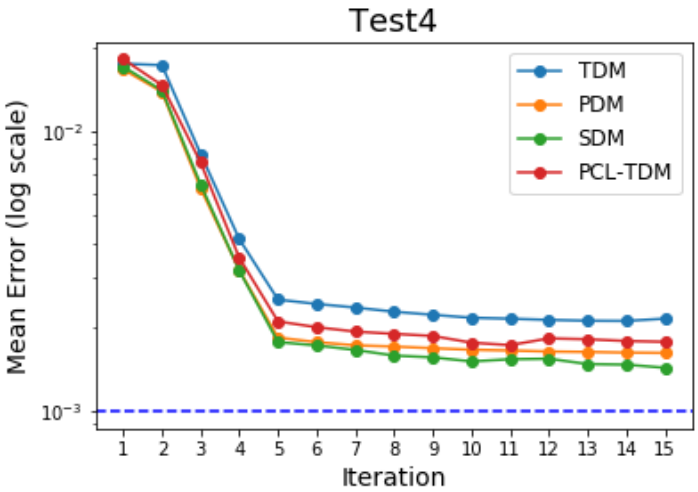
Figure 6.3: Input, output, and performance: Test-case 3. (10467 points)



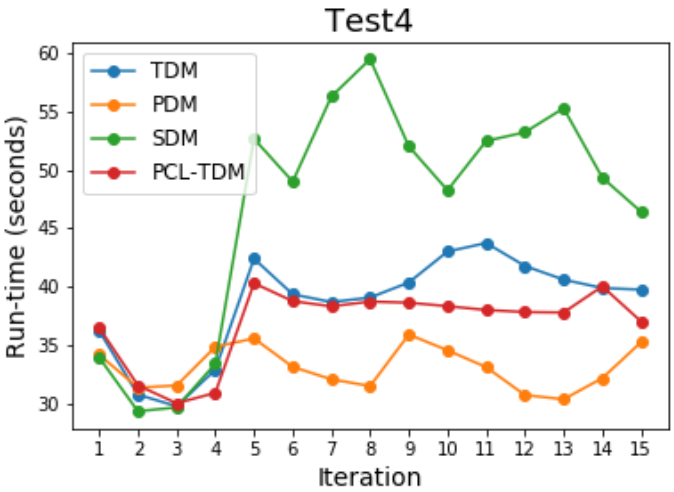
(a) Input Point Cloud



(b) Fitted Surface



(c) Error of the objective functions



(d) Runtimes of the algorithms

Figure 6.4: Input, output, and performance: Test-case 4. (6411 points)

Distance Metric	Test Case	Avg. run-time (secs)	Min. Fitting Error	Total points
PDM	Test-1	**5.75**	0.00040	1447
TDM	Test-1	8.00	0.00062	1447
SDM	Test-1	8.27	**0.00038**	1447
PCL-TDM	Test-1	10.07	0.00046	1447
PDM	Test-2	**14.27**	0.00032	3204
TDM	Test-2	17.10	0.00031	3204
SDM	Test-2	20.74	**0.00030**	3204
PCL-TDM	Test-2	22.83	0.00031	3204
PDM	Test-3	**44.68**	**0.00026**	10467
TDM	Test-3	58.27	0.00029	10467
SDM	Test-3	62.40	0.00031	10467
PCL-TDM	Test-3	62.32	0.00037	10467
PDM	Test-4	**33.15**	0.00162	6411
TDM	Test-4	38.59	0.00210	6411
SDM	Test-4	46.72	**0.00143**	6411
PCL-TDM	Test-4	36.90	0.00172	6411

Table 6.1: Table summarizing results of all the experiments.

6.3 Discussion

Table 6.1 summarises the results of fitting experiments performed using the system. It can be seen that surface approximated using SDM objective has slightly better fitting error than other algorithms.

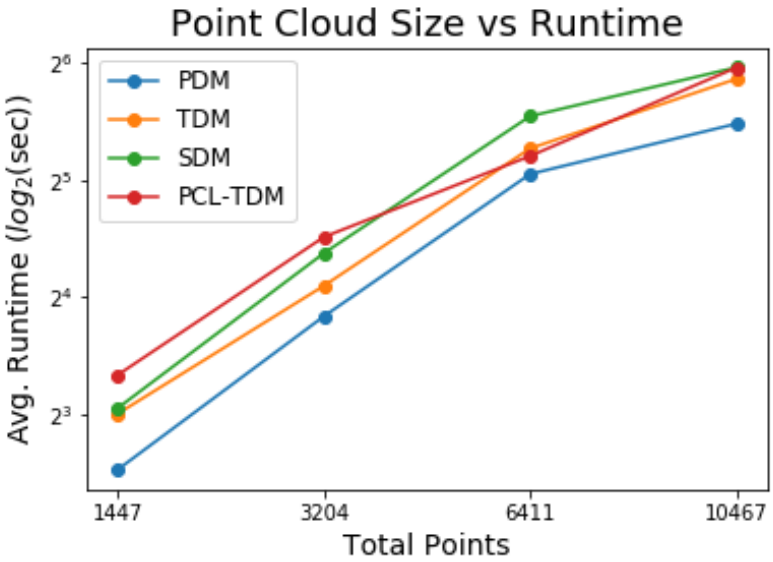
6.3.1 Fitting Error

The system was consistently able to approximate surfaces with an error less than the value of 0.01. It was seen that SDM performed overall slightly better than other optimisation objectives. One interesting observation was that even though none of the optimisation objectives performed well on the fourth test case, SDM performed a lot better than other optimisation objectives while TDM performed the worst.

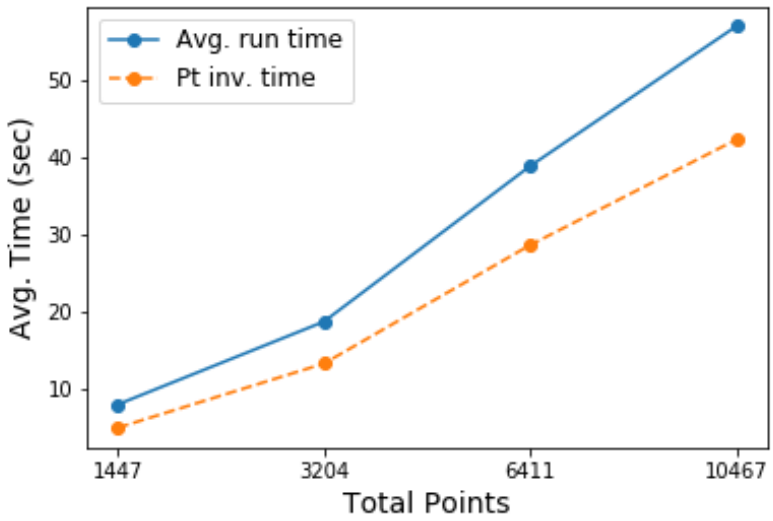
This behaviour can be attributed to the curvature associated with the fourth test case and hence the claim made by Wenping Wang and other in [22] about the instability of TDM near regions of high curvature due to lack of curvature in formation can be verified.

6.3.2 Runtime

Figure 6.5 shows how the average runtime of the optimisation objectives behave with increase point cloud size.



(a) Plot of average runtime/iter vs the point cloud size.



(b) Plot showing relationship between total runtime and point inversion time per iteration.

Figure 6.5: An analysis of effect of point cloud size on runtime

It can be seen that although SDM produces slightly better approximations, it takes almost 1.5 times the time to run one iteration of fitting than that taken by the PDM. Keeping this in mind and considering that PDM sometimes outperforms SDM and the average difference between the two is only 0.000045, the use SDM objective function over PDM cannot be considered viable.

Further, it is observed that the runtime of the algorithm increases approximately linearly with the increase in the size of the dataset.

The low values of the run-times of all the objective functions during the first five iterations is explained by the control point initialisation scheme employed. Initially, the number of control points are set to the minimum allowed value. Through iterative knot insertion [17], the number of control points reached the defined limit in 5 iterations.

Figure 6.5b shows the relationship between the total run-time and time taken by point inversion in an iteration. It can be seen that the total runtime constitutes majorly of the point inversion time. This shows the bottleneck created by performing newton iteration for point inversion in this algorithm.

6.3.3 Convergence

It is interesting to note that although the average runt-imes of the algorithms are large for large points cloud like the test case number 3, the fitting algorithms are well within the error value of 0.001 within 5 iterations. The value of the error does not decrease much after that.

6.3.4 Query API

The figure 6.6 shows plots of the results from querying the normal and tangents at 5 randomly selected points on the fitted surfaces. The same functionality was tested by importing the module into MATLAB. The module was successfully imported and could be seamlessly used for fitting surfaces to point clouds and querying.

6.3.5 Curvature

The Figure 6.7 shows the first and second derivatives of the Z values w.r.t the parameter u and v of the surface. Since the axes u and v may not align with the axes x and y, the shape of the plots will be slightly different from the plots seen until now. The purpose of these plots is to demonstrate proof of C2 continuity.

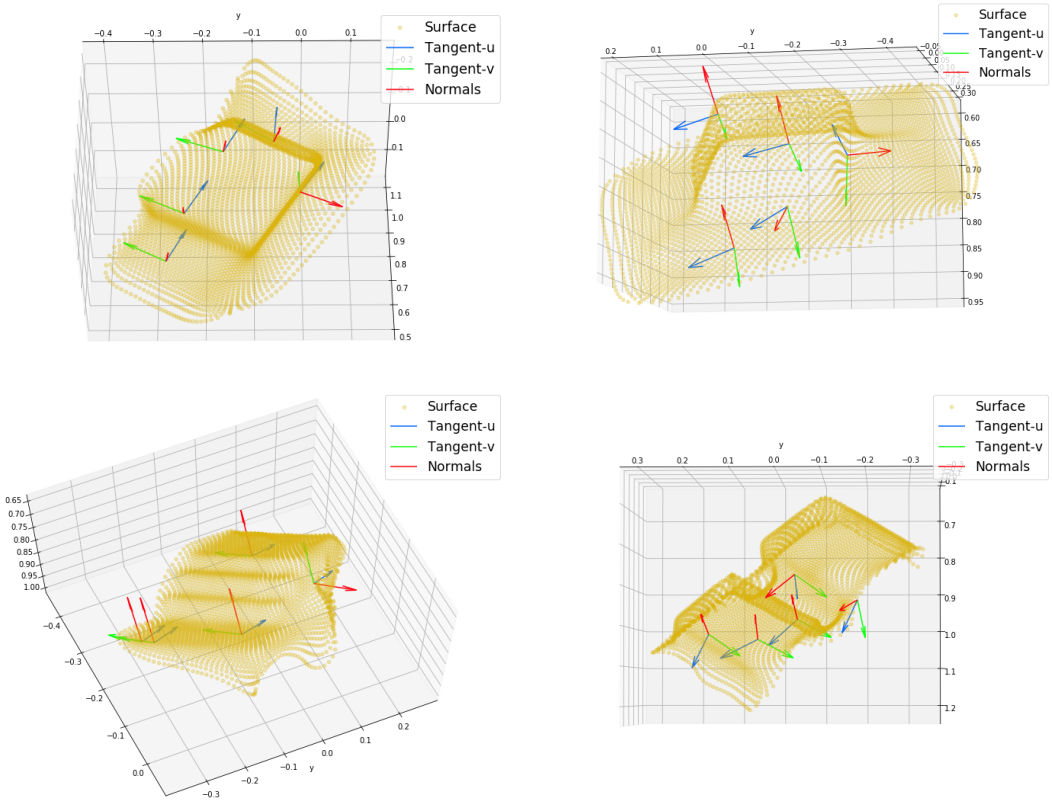
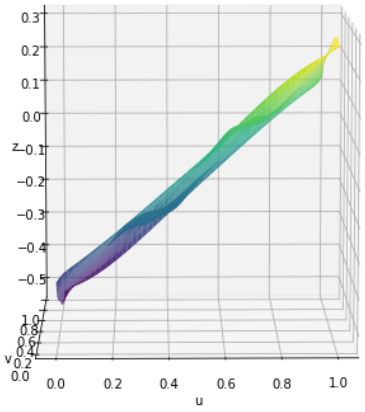
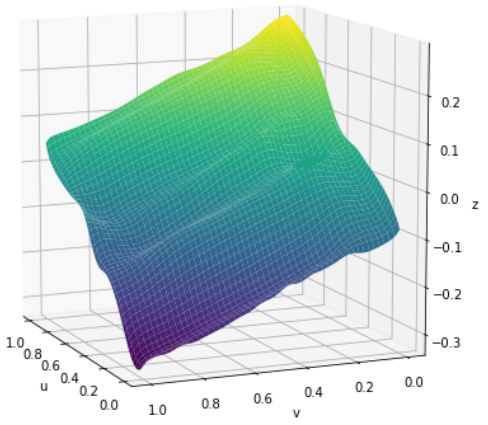


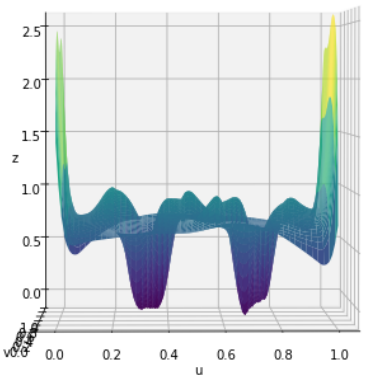
Figure 6.6: Plots of results for querying 5 random points on the fitted surfaces: The red lines show the estimated normal, the blue lines show the tangent corresponding to the curve represented by the parameter u, and the green lines show the tangent corresponding to the curve represented by the parameter v.



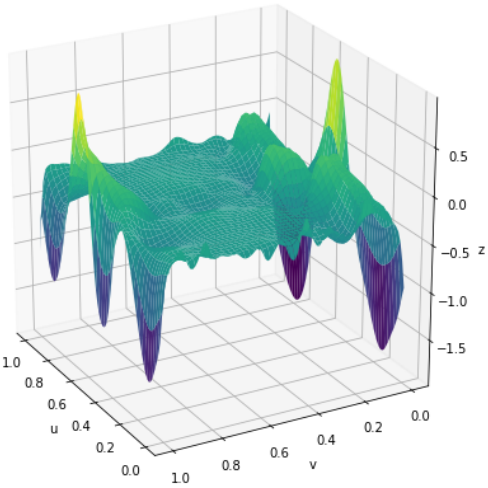
(a) Non differentiated z values (u facing)



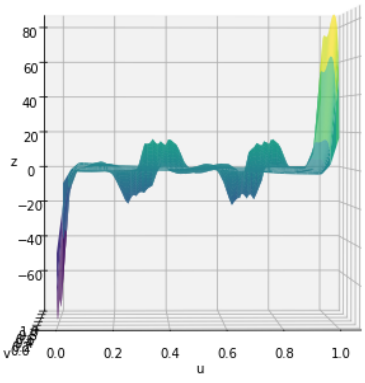
(b) Non differentiated z values (v facing)



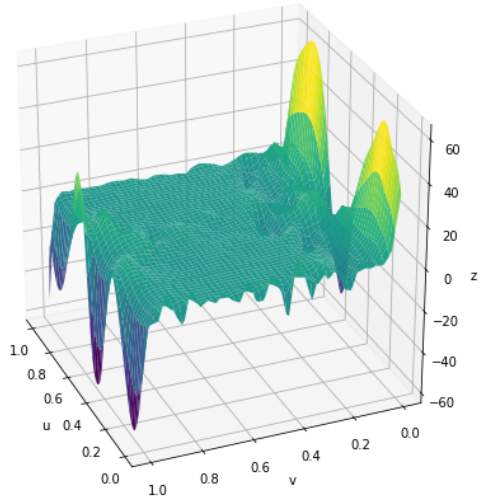
(c) First derivative of z w.r.t u



(d) First derivative of z w.r.t v



(e) Second derivative of z w.r.t u



(f) Second derivative of z w.r.t v

Figure 6.7: Derivatives of the z values of a modelled surface w.r.t. u and v params

Chapter 7

Conclusion and Future work

7.1 Summary of achievements

In this project, a system to approximate 3D surfaces from point clouds was successfully implemented. This was done by first understanding the need for such a system and subsequently identifying the steps that had to be taken to accomplish this task. These steps included:

- getting familiar with the technologies that enable such a system. Such as various 3D sensors and the methodology to create and process point clouds.
- carefully understanding the requirements that this system must fulfill to be successful. This included the constraints of C^2 continuity and general requirements needed for easy integration.
- a thorough review and evaluation of the literature available on surface representation to identify the best representation to use in the system.
- a thorough analysis of the tools available to build the system. This involved compiling and experimenting with various libraries like the PCL [19], PyMesh[1] etc.
- designing the system in a way that it captures all the requirements. This included a personal motivation to create a system that can be easily adapted to extend the work that has been accomplished in this project.
- Implementing the design and several iterations of refactoring to ensure optimal performance of the system.

[1] https://github.com/PyMesh/PyMesh

7.2 Summary of Evaluation

The results discussed in the last section indicate the system met all the evaluation criteria that were identified.

- The approximations generated by the system were smooth with an appropriately small value of fitting error.
- The surfaces produced were C^2 continuous which satisfied a key objective of this project. Using b-splines made it possible to add this constraint into the design of the solution avoiding the need of any external optimisations to achieve this.
- The maximum number of control points that were allowed were 1225, and yet the system was able to approximate a point cloud with approximately 10000 points very accurately. This satisfies the compaction criteria we created.
- The system was successfully used and tested from within the MATLAB environment.

7.3 Criticism of work done

Although, the fitting accuracy of the system is good and the properties of the resulting surfaces adhere to all the requirements, the runtime of the algorithm is still not optimal. This is because, even though parallelizing the execution of newton iteration for point inversion increased the performance of the system, as the size of the point cloud grows this method of point inversion still becomes a bottleneck in the algorithm. In order to make the system more scalable, it is important to find an alternative, more efficient method to perform point inversion.

7.4 Future Work

Since the motivation behind this project is visual perception and visual perception usually happens in real-time, exploring ways to make the surface approximation process work in real-time comes as a natural extension to this project.

Bibliography

[1] Optimization-based locomotion planning, estimation, and control design for atlas. *Autonomous Robots*, 40(3), 2016.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, Jan 2003.

[3] Onur Rauf Bingol and Adarsh Krishnamurthy. NURBS-Python: An open-source object-oriented NURBS modeling framework in Python. *SoftwareX*, 9:85–94, 2019.

[4] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350 – 355, 1978.

[5] Edwin E. Catmull. A subdivision algorithm for computer display of curved surfaces. 1974.

[6] Yueyue Cheng, Henri Palleis, and Wolfgang Hhl. Simple 3d low poly modeling tool with intuitive gui, 07 2015.

[7] Jongmoo Choi. *Range Sensors: Ultrasonic Sensors, Kinect, and LiDAR*, pages 1–19. Springer Netherlands, Dordrecht, 2016.

[8] Frost and Sullivan. Lidar: driving the future of autonomous navigation, 2015.

[9] Technische Hochschule, Andreas Hubeli, and Markus Gross. A survey of surface representations for geometric modeling. 04 2000.

[10] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. volume 2, pages 1403–1410, 01 2003.

[11] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[12] Charles Teorell Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, January 1987.

[13] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[14] Michael Montemerlo, Sebastian Thrun, Daphne Roller, and Ben Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 1151–1156, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[15] K. Nakayama, Z. J. He, and S. Shimojo. *Visual surface representation: a critical link between lower-level and higher level vision*, pages 1–70. M.I.T. Press, 1995.

[16] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[17] Les Piegl and Wayne Tiller. *The NURBS Book (2Nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1997.

[18] Andreas Richtsfeld, Thomas Mrwald, Johann Prankl, Michael Zillich, and Markus Vincze. Learning of perceptual grouping for object segmentation on rgb-d data. *Journal of Visual Communication and Image Representation*, 25:6473, 01 2014.

[19] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[20] Shengjun Tang, Qing Zhu, Wu Chen, Walid Darwish, Bo Wu, Han Hu, and Min Chen. Enhanced rgb-d mapping method for detailed 3d indoor and outdoor modeling. *Sensors*, 16(10):1589, Sep 2016.

[21] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[22] Wenping Wang and bullet . Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics*, 25:214–238, 05 2006.

[23] Greg Welch and Gary Bishop. An introduction to the kalman filter. 1995.

[24] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[25] Kai Yuan, Iordanis Chatzinikolaidis, and Zhibin Li. Bayesian optimization for whole-body control of high degrees of freedom robots through reduction of dimensionality. *IEEE Robotics and Automation Letters*, 2 2019.

[26] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.