Bootstrap Based Surface Reconstruction

RAMLI, AHMAD,LUTFI,AMRI,BIN

Bootstrap Based Surface Reconstruction

Ahmad Lutfi Amri Ramli

Department of Engineering and Computing Sciences

Durham University

A thesis submitted for the degree of

Doctor of Philosophy

May 2012

Abstract

Surface reconstruction is one of the main research areas in computer graphics. The goal is to find the best surface representation of the boundary of a real object. The typical input of a surface reconstruction algorithm is a point cloud, possibly obtained by a laser 3D scanner. The raw data from the scanner is usually noisy and contains outliers. Apart from creating models of high visual quality, assuring that a model is as faithful as possible to the original object is also one of the main aims of surface reconstruction.

Most surface reconstruction algorithms proposed in the literature assess the reconstructed models either by visual inspection or, in cases where subjective manual input is not possible, by measuring the training error of the model. However, the training error underestimates systematically the test error and encourages overfitting.

In this thesis, we provide a method for quantitative assessment in surface reconstruction. We integrate a model averaging method from statistics called bootstrap and define it into our context. Bootstrapping is a resampling procedure that provides statistical parameter. In surface fitting, we obtained error estimate which detect error caused by noise or bad fitting. We also define bootstrap method in context

of normal estimation. We obtain variance and error estimates which we use as a quality measure of normal estimates. As application, we provide smoothing algorithm for point clouds and normal smoothing that can handle feature area. We also developed feature detection algorithm.

Copyright

Declaration

The material contained within this thesis has not previously been submitted for a degree at the Durham University or any other university. The following publications were produced during the course of this thesis:

- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Distance based feature detection on 3d point sets", *In TPCG*, pages 53-56, 2009.
- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Bootstrap test error estimations of polynomial fittings in surface reconstruction", *In VMV*, pages 101-112, 2009.
- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Bootstrap-based normal reconstruction", *In Curves and Surfaces*, pages 575-585, 2010.

Contents

List of Figures

Chapter 1

Introduction

In today's technology, 3D has become one of the staple terms in daily life. 3D is offered in various forms - movies, games and photography. It can be widely defined either in reference to the object or the way our eyes observe it. A cartoon drawn on a piece of paper is seen in 2 dimensions. An animation made out of polygons is a 3D object but viewed in 2D on an ordinary computer or television. In popular culture, a movie which requires polarized 3D glasses or red-cyan glasses allows the viewer to see a 3-dimensional object in a 3-dimensional view, giving viewer more in-depth in distance and width, similar to what we experience in the real world.

As technology evolves, we can now have a 3D picture rather than an old-style 2D photo on paper. One may question which one is more aesthetically pleasing - either a normal 2D picture, or a 3D picture seen through special devices like 3D-glasses [39]. Whatever the case, a 3D representation of a real world object certainly offers more information.

A 3D supersedes 2D by giving extra information about an object. For in-

Figure 1.1: A 3D Scanner (Artec MHT 3D Scanner) can scan a real-life object to be represented as 3D model on a computer.

stance, taking a picture of a cube using a Polaroid camera may not offer in depth information of the length or width. A 3D camera, however, which uses 2 lenses (in some cases, just one) can give us depth and distance information.

A 3D model can also be made with combinations of polygon and molded in a 3D editing tool program such as Maya or 3DS Max. In such programs, a user can define an object the way that one intended. This can later be used for applications such as animation or modelling a design for engineering or manufacturing.

On the other hand, device such as 3D scanner will produce point cloud and needed to be processed further. In this context, a 3D object or model is defined as representation on a standard 3-dimensional Cartesian coordinate system where length, width and height is defined on x, y and z-axes. With this definition, we differentiate it from the norm to describe 3D as a viewing capability, and stick to how it is presented.

The input data for a surface reconstruction is obtained from measurements of real objects using optical devices such as 3D scanners or cameras producing image sequences. A 3D scanner (as in Figure 1.1) has not yet been made available widely as a commercially available user product. The model generated will provides more information about the object, because it can capture the object in its entirety. After reconstructed in the computer, one can rotate and view it from 360 degrees direction.

1.1 The modelling pipeline

We show the pipeline of surface reconstruction in Figure 1.2. As the procedure is not generic and one may choose different approaches, we acknowledge that this pipeline may be varied. Firstly, data acquisition procedure provides a set of point clouds from a real life object. Scanning may be done couple of times, positioning the model from different angles to assure that the scanner could capture all regions to provide few samples of point clouds. When this happens, registration needs to be done so it can combine the multiple point cloud that we obtained [19].

Once we obtain the point clouds, we may apply other pre-processing step such as outlier removal, simplification or denoising. Then, a surface representation is reconstructed from the set of point clouds. Post-processing steps may include some more denoising or smoothing if the data is still noisy, and remeshing. If the model contains sharp edges, one may want to make sure that the feature is well preserved. To use in animation or presentation, we may set up attributes and apply a texture on the model.

While this is only a basic idea, one may skip some parts and even go from

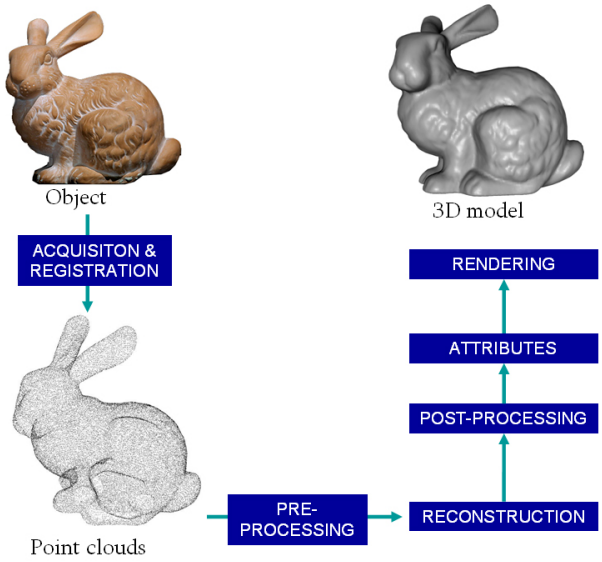

Figure 1.2: Pipeline of surface reconstruction. Scanning on a Standford bunny provides point clouds before being reconstructed into a 3D model.

point clouds to generating model by using a method such as implicit surface reconstruction which handles denoising, estimates surfaces, and generates triangulation without going through the steps shown in the Figure 1.2.

There is a variety of steps that can be done and these will be mentioned and discussed in Chapter 2.

1.2 Surface Reconstruction

Surface reconstruction is one of the main research areas in computer graphics. Its basic aim is to find the best approximate of surface representation of a real object. In many geometric modelling applications we want to create the 3D model of a real object, usually representing it by a surface describing the object's boundary. The process starts with data acquisition, where an optical device such as a 3D scanner

produces point sets describing the object. These point sets always contain noise. Moreover, in several applications, ranging from visualisation to Finite Element simulations, point sets are not suitable representations of the object's boundary, and surface representations such as implicit surfaces or meshes are preferable. The process of transforming a point set into a surface is called surface reconstruction.

Other than the raw data being noisy, there is also the problem of outliers: a point exist in a region that is quite far from the rest of the data due to inaccuracy in sampling procedure. Through the reconstruction process, we may lose the fine features of an object such as the sharp edges and the details of the boundaries. To obtain a good surface representation, we have to use a procedure that will eliminate or at least minimise these problems.

To tackle these problems we need a method for assessing the quality of the reconstructed surface. Such methods can be subjective, such as visual inspection, or objective, such as error estimation. The assessment of the final surface also depends on the rendering process.

While it is more common to create a polygonal mesh for the purpose of rendering, recent algorithms can simply do direct volumetric rendering. In this case, the extraction of a polygonal mesh is not required. Researchers have different opinions on the merits of each approach.

In all cases, human labour is still required to resolve the problems with the quality of the data [103]. A good surface reconstruction algorithm that would be able to deal with the mentioned issues and minimize the human involvement will obviously have immediate practical implications.

1.3 Motivation

This thesis will revolve around the development of surface reconstruction algorithms from data obtained from measurements of real objects, assuring that the model is as faithful as possible to the original object. As we have mentioned that the faithfulness of the model is not always the goal of surface reconstruction, the target applications of this thesis are those where the accuracy of the model is important. For example, in CAD/CAM applications the model will be used for the manufacturing of a physical object, and its faithfulness to the prototype is important. Cultural heritage and medical applications are other such examples. For instance, modelling an ancient statue or a human organ for the purpose of analysing it or just visualising it, requires us to be sure that the produced model is accurate.

Current surface reconstruction methods neglect the element of accuracy, either by assuming that there is no noise or by dealing indirectly with it: for example by smoothing the data with averaging operations during reconstruction. While many surface reconstruction procedure assume that the noise is uniformly distributed, Sun et al. [96] address that this is not the exactly the case. Our aim is to have the proposed algorithms deal explicitly with this matter and provide estimates of the data noise.

This is where a statistical method is useful. Statistical methods offer a vast literature regarding data mining, supervised learning and model assessment. In recent years, researchers have started to integrate this two fields to provide a method which can be assessed quantitatively and not just visually.

One of the model averaging methods which is bootstrap sparks our interest

as it could provide statistics such as variance and error estimate. In our case, we envisioned that the error value could be estimated to detect different noise level and be used for an algorithm such as denoising. Having such values like error could define the quality of our approach and make the assessment more quantitative, which relies on numerical value, rather than qualitative which relies on human observation through eyes.

Without a doubt, having such information could give much insight such as level of noise or quality of an approach, and require less involvement of checking it visually thus avoiding the subjectivity of one person. Bootstrap procedure will be the root of the thesis, and will be explored depending on the application of surface reconstruction.

1.4 Overview of Thesis

This thesis will explore a quantitative assessment of post-processing steps in surface reconstruction. As the field of statistics has already offered a vast literature regarding statistical measurement on collected data, we found that recently, researchers started to have interest in merging these two fields. This thesis will apply a model averaging method called bootstrap to provide a quantitative value which may tell the quality of our estimation. Our aim is:

1. To define the bootstrap method in a surface reconstruction setting. As this is a new approach, a verification procedure is expected.

2. To provide error and variance estimates for handling noisy point sets. Depending on the issue with which we are dealing; either smoothing or esti-

mating normal, we will choose the statistics accordingly.

3. To use the information obtained to recover the data. This will be used in either smoothing, feature preservation or normal estimate to demonstrate the benefit of having such statistics.

We will start by reviewing the literature related to this field in Chapter 2. Then, in Chapter 3, a discussion of the mathematical background will list methods that we use in our computation which had been used generally among researcher in related field. This will includes the discussion of type of error and polynomial fitting.

Chapter 4 will line out our test error estimation on a model based on bootstrapping. Test error estimates can be used to signify the noise level and quality of our estimate.

In Chapter 5, we come to a discussion regarding normal estimation. While the essence of bootstrapping is similar to that described in the previous chapter, we have to redefine the bootstrap approach on a normal estimation perspective. We use variance value to signify the quality of normal estimate on a model. Later in the chapter, an application is introduced to estimate a more accurate value of normal from noisy point sets. We also define test error estimates on normal approximation so that a comparison between different methods and neighbourhood can be done.

In Chapter 6, we will apply a bilateral filter which uses the error value that we obtained from the bootstrapping procedure to smooth out the noisy points and recover our model.

Chapter 7 explores detection on a feature area , using values from Principal Component Analysis and variance value from bootstrapping.

We conclude our thesis in Chapter 8, describing the contribution, limitation and future works.

Chapter 2

Literature Survey

In this chapter, we provide some relevant definitions and discussions related to surface reconstruction. We start by defining the setting of this research. Basic definition and data structures will be discussed in Section 2.1. Next in Section 2.2, we address about normal computation on surface. Normal estimates may be used in various step of surface reconstruction such as denoising and rendering, thus having an accurate estimate can be helpful. Further discussion will be continued in Chapter 5.

Section 2.3 presents denoising on meshes and point sets. As input data contains noise, denoising may recover feature area. Our proposed method in regards to denoising is presented in Chapter 6. Section 2.4 addresses surface reconstruction, which includes various approaches including implicit surface, Voronoi diagram, RBF and surface fitting. We also address statistical approaches that have been done in surface reconstruction before discussing the statistics in general in Section 2.5. Finally, we discuss about feature detection in Section 2.6.

2.1 Definitions and basic concepts

There are several available ways to generate and represent a 3-dimensional model in geometric modelling. The most common are polygonal meshes, point set representation, free form surfaces or implicit surfaces.

A mesh consists of vertices, edges and faces. A point is called a *vertex*. A vertex connects to an other vertex through an *edge* to form a polygonal shape. The vertices connected to form a polygon is called a *face*.

The point set representation is much simpler in definition. We need to define only the positions of the vertices. No information of connectivity is needed. Different ways of rendering are used for meshes and point sets. Throughout this thesis, there will be fewer mentions of how rendering works for both as it would not be our main focus. However, it will be taken into account when choosing between those two representations.

In many applications, it is important to know the neighbourhood of a vertex. We may use this knowledge to define features, run smoothing, or calculate normals. As the mesh representation includes the connectivity of points, we can use it to define the neighborhood as the set of vertices connected to that vertex with an edge. These points would be the *1-ring neighbourhood* of the point. It can be expanded to the two ring neighborhood for the considered point by including the vertices that are connected to the 1-ring neighbourhood. This k-rings neighbourhood concept is used widely in mesh processing and the considered point can be seen as a center of a dart board (the bulls-eye).

As there are no edges connecting the points in point-sets setting, we use the concept of K-nearest neighbourhood instead. Searching through the space of

points, we find points which are relatively close to a considered point. Using brute-force, we may find the distances of every single points to the considered point; and then simply find the K nearest points. However, it will be costly, as we have to repeat the procedure for each point, and our models often consist of thousands of points.

Data structures can be used to find and store the information of connectivity. As for mesh representation, one can use half-edge data structure [14; 64] to make the search for neighbouring points efficient. Half-edge data structure search the vertex and mark 2 faces connected to an edge in this process is done iteratively. For point sets representation, in searching for the K-nearest neighbourhood, see [20; 24; 108]. Commonly, the space are subdivided and *tree* is constructed making it easier to connect on point to the other base on its location in the subspace.

Researchers have different views on whether to use meshes or point sets. Mesh representation provides connectivity information, which makes it easier to compute neighbourhoods and apply a procedure in the neighbourhood (such as recovering sharp edges). On the other hand, rendering point sets may be better as [45] believe that it can be faster and more accurate than mesh rendering.

In this thesis, we will focus on point sets rather than meshes. As our approaches use statistical methods, we believe that point sets would suit our approach better as it is not limited to connectivity.

2.2 Normal Estimation

Given a surface, the normal of a point is a direction perpendicular to the surface at that particular point. This vector may be used in various steps of reconstruction.

For instance, when handling noisy data, one may wish to smooth the data and move it towards the direction of the normal. Another example is that the normal may be needed in rendering so as to compute lighting and shadow.

Normal value may be represented by vector $(\mathbf{i}, \mathbf{j}, \mathbf{k})$. Let $f(x, y, z) = 0$ be a function which define an implicit surface, the normal at point (x, y, z) is given by the gradient $\bigtriangledown f$. The normal direction is orthogonal to a tangent plane. In practice, both functions cannot be easily defined globally for an object, especially a complex looking one. Therefore, most of the methods are local and we find the normals in a small neighbourhood.

Given a 1-ring of a vertex of a triangle mesh with m faces, a simple normal can be estimated with

$$\sum_{i=1}^{m} n_i A_i$$

whereas n_i is the normal of i-th neighbouring face and A_i is the area for i-th face. This normal can be extended into k-rings by including weights in the above equation, such that the weights will be lower for various increment of k. Note that the k-rings neighbourhood does not directly provides information of spaces.

For point sets representation, given n points p_i's which is nearest to the vertex, using Principal Component Analysis [53], a normal can be estimated by finding the eigenvector for the smallest eigenvalue. These eigenvalues are derived from the covariance matrix of the distance of points to the centroid of the points, described as follows.

Given a neighbourhood of points $\{p_0, p_1, p_2, ...p_n\}$ centered at p_0, we construct a 3 by 3 covariance matrix $\{b_{jk} | j, k = 1, 2, 3\}$. For every vertex p, we calculate

the eigenvalues of the covariance matrix

$$b_{jk} = \sum_{i=1}^{n} (p_{i_j} - p_{0j})(p_{i_k} - p_{0k}) \tag{2.1}$$

where $p_i = (p_x, p_y, p_z)$ is the i-th neighbouring point. Let $a \leq b \leq c$ be the three eigenvalues of the matrix in Eq. 2.1. The respective eigenvector corresponding to a is the normal vector at p_0.

Alliez et al. [2] computes normals with confidence values based on the ratios of the eigenvalues of covariance matrix. The improved PCA normal estimation by Pauly et al. [86] and Alexa et al. [1] use the weighted distance instead. Hu et al. [52] propose a bilateral estimation of the normals. They decomposed the vertices from k-rings neighbours and divided it into multiple layers, based on the distances of the vertices to the considered vertex. By using Gaussian weights, they estimated the normals by taking every layer into account.

Mitra et al. [80] used least square fitting to estimate the normals. They also compute the optimal neighbourhood size to reduce error in normal estimation. Dey and Sun [30] approximate normals and features of a model using the medial axis. By creating a medial ball that is large enough with respect to a neighbourhood, the normal is estimated by the direction of the vertex from the center of the medial ball. This approach can adapt to noisy data sets, but seems to be expensive, as it considers global point sets instead of doing it locally.

As reliable normal estimations are essential for many geometric processing and rendering algorithms, the literature on normal reconstruction is extensive. The classic method proposed in [51] applies PCA on the k-neighbourhood of a point. The result is equivalent to computing the normal of the minimum least

square fitting plane for the same neighbourhood. Due to its simplicity and robustness, [51] is still widely used and considered to be part of the state-of-the-art. Pauly et al. [88] improve on PCA by computing a weighted least square fitting plane, with weights given by sigmoidal functions. Gopi et al. [44] use Singular Value Decomposition to compute a normal that minimises the variance of the dot products between itself and the vectors from the considered point to its nearest neighbours. Mitra et al. [80] apply least square fitting on the points inside a sphere of fixed radius. Hu et al. [52] combine several estimates obtained at different sampling rates to compute a more reliable normal. Dey et al. [28] compute the normal as the vector pointing to the centre of the largest Delaunay sphere passing through the considered point. Li et al. [72] create several random subsamples of the neighbourhood and uses them to estimate the local noise; it then uses the local noise estimate to compute a kernel function which is minimised by the tangent plane.

To improve normal estimation on noisy points, several approaches had been proposed. A bilateral filter for normal smoothing was proposed in [58]. There, the influence of the normal at d_j on the normal at d_i depends on the distance between d_i and d_j and also on the distance from d_i to the tangent plane at d_j. [113] had also use bilateral filter on meshes to smooth normals. The first weight is similar, which is distance-weight but the second weight is signal difference between d_i and d_j.

Most of the point set smoothing algorithms proposed in the literature focus on updating the positions of the points, while the normals are only implicitly updated. [91] uses a filter in the spatial domain, [89] filters in the frequency domain, [69] uses anisotropic diffusion, while [112] simultaneously optimises the

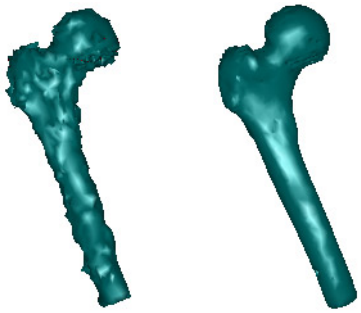

Figure 2.1: An example of a noisy model on the left. The model after denoising process is displayed on the right.

point set and its parametrisation by minimising an energy function.

2.3 Denoising

Generally, the input from a 3D scanner will inevitably produce noise (See Figure 2.1). The term *denoising* is literally interpreted as a process of removing the noise from the data. Going through the existing literature, this term is widely used interchangeably with *smoothing*. It is not in some cases, however, Sun et al. [95] have made a clear distinction between these two. In their terms, smoothing or fairing is a process to remove certain high-frequency information, while denoising is to preserve only genuine information at all frequencies.

2.3.1 Mesh denoising

One of the classic mesh denoising methods is Laplacian smoothing. Given a set S of vertices, Laplacian smoothing projected the considered vertex towards its average of neighbouring points.

Define p_i as considered vertex and q_i's are n numbers of the neighbouring

vertices connected through edges to p_i. By using Laplacian smoothing [101], the projected p_i is given by

$$p_i' = \alpha p_i + \frac{1-\alpha}{n} \sum_{j=1}^{n} q_i$$

This algorithm is done iteratively and is considered at a one-ring neighbourhood. α is a user-defined constant where $0 \leq \alpha \leq 1$. A larger value of α will require more iterations before the noisy model smoothes out. On the other hand, a smaller value of α may smooth out features rather quickly. Regardless of the α value, the model will lose its features and has a tendency to shrink after several iterations.

To avoid the shrinking effect, Taubin [100] used a signal processing approach which uses Fourier analysis. Compared to classic Laplacian smoothing, his algorithm avoids shrinkage by projecting vertices in opposite directions for every iteration. Similarly, HC algorithm [101] also push back the considered vertex towards its original position. The approach produces a better result in terms of avoiding shrinkage, but the properties of features and sharp edges may be lost along the way.

Desbrun et al. [27] introduces volume preservation to avoid shrinkage. Regardless of wherever the vertices are projected, the model will resize to match the original volume. Looking under differential equation setting, they pointed out that Laplacian smoothing and similar improved methods are using the concept of diffusion, which will result in shrinkage. Instead of using diffusion, they looked at the curvature flow of a model. The method used a similar approach of calculation as [100] by using signal processing approach. It manages to keep the features but tends to be expensive, as it needs more calculations.

Desbrun et al. defines curvature flow as

$$\frac{\partial p_i}{\partial t} = -k_i \mathbf{n_i} \ .$$

k_i is the mean curvature while $\mathbf{n_i}$ is normal for vertex p_i. The differential equation represents the process of moving the vertices along its normal direction of an amount proportional to its curvature. By solving the equation we will be able to smooth out the noisy surface and still preserve the features. In practice, this equation has to be discretized first, as the equation is in continuous setting. After discretization, we will have a sparse linear system, and solving the system would not be difficult.

Eickstein et al. [32] made a direct improvement of Desbrun's method by incorporating the volume preservation into the differential equation, instead of applying it separately as was previously done. The curvature flow is redefined as

$$\frac{\partial p_i}{\partial t} = -L k_i \mathbf{n_i} \ .$$

whereas L is the *volume-controlled deformation-filtering* operator. This operator preserves the volume locally, thus maintaining it globally as well. Hildebrandt et al. [50] used anisotropic curvature flow to smooth the surface while preserving the features.

Fleishman et al. [37] and Jones et al. [57] adapt 2-dimensional bilateral image filtering to 3-dimensional meshes. The algorithm works at its local vertex, moving the vertex in its normal direction using Gaussian filtering. It is affected by the variance of the k-ring neighbourhood. If the variance is high (for example, at sharp edges) the weights will be decreased. The particular region will not be

smoothed out, therefore preserving the features and sharp edges. Under the same approach, Sun et al. [95] used normal filtering and vertex position updating. The quality of features preservation are dependant to the filtering method.

Guskov et al. [48] use signal processing methods on meshes. Their methods can be used in mesh smoothing, enhancement, subdivision and multiresolution editing. For an edge, they compute two directions of normals affected from the two faces it shares. The difference between two normals is equivalent to the second order differential equation, and the meshes may be smoothed out by defining a relaxation operation which minimizes this value. We show this idea in Figure 2.2 in a two dimensional setting. Minimizing the difference of two normals may smooth out the meshes. The implementation also takes other variables into account, for example, weights are given depending on the distances to the considered edge.

For experimental purposes, instead of taking raw data, we usually take a 3D model and add some noise on it. We can then compare the effectiveness of our method by making a comparison of the original model and the smoothed one. Besides visual inspection, one may use Metro Tool by [23] to estimate the error produced by a method, by comparing the model before and after it has gone through the denoising process.

2.3.2 Point set denoising

Point set denoising is different than mesh denoising in terms of absence of connectivity. While mesh denoising uses mesh representation, point set denoising deals with points in Euclidean space.

Moving least squares (MLS) method is one of the widely used approaches in

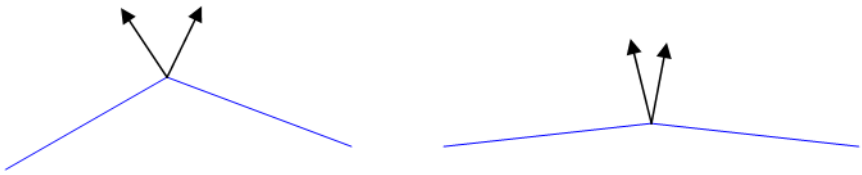

Figure 2.2: Relaxation operation in 2D setting. The arrows represent the normals for their respective lines. Minimizing the difference for two normals may smooth out the sharp part.

point set denoising. Following [1; 6], we give a brief summary of MLS. Note that while Amenta and Kil [6] define the error in the form of an energy function, both methods are very similar.

Let a new point q_i move along a normal direction from considered point r_i. The PCA normal is used in this method. Given a local frame of point sets, we find the value of q_i that will minimize the weighted distance of the points from the tangent plane as

$$\min \sum_{j=1}^{n} D(p_j)\theta(p_j, q_i)$$

$D(p_j)$ is the nearest distance of points p_j to the tangent plane where q_i lies. $\theta(p_j, q_i)$ is the weight based on the distance of p_j to q_i (Refer to Figure 2.3). Usually, the Gaussian function is used such that if the distance is larger, then the weight is lower.

Then we will find the polynomial fitting such that it will minimize the following equation

$$\sum_{j=1}^{n} (d(f_j - p_j))^2 \theta(p_j, q_i)$$

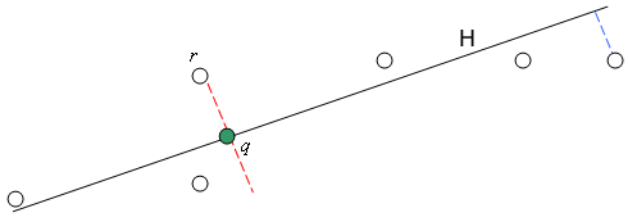

Figure 2.3: The new point q is moved along the normal direction represented by the red line. The tangent plane H will move according to q. The blue lines represent D(p_j).

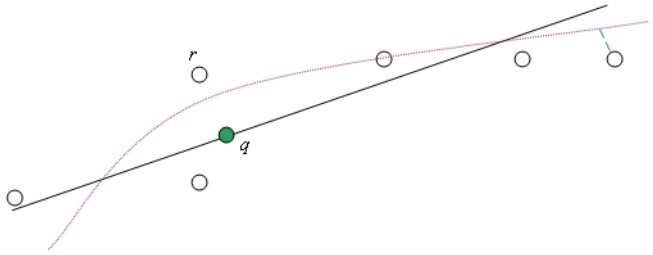

Figure 2.4: The polynomial approximation is represented by the purple line. We try to minimize the weighted distance of the points p_j to corresponding value of f_j. The green line represent $d(f_j - p_j)$.

whereas $d(.,.)$ is the distance between two points. The function F is an n-order polynomial fitting. $f_j \in F$ is the corresponding polynomial approximation that aligns with its respective point p_j on the normal direction. We will then move the considered point r_i so that it lies on the polynomial surface defined. After an iteration, points r_i's will be moved and the model will be smoothed.

Levin [71] shows that the local error of MLS approximation is bounded. Lipman et al. [74] used a different projection, which does not require a local parameterization. Bremer and Hart [13] find sampling conditions that guarantee well-defined normals on neighbourhoods of the surfaces, which lead to well-defined MLS surfaces.

Pauly and Gross [85] propose an algorithm for point set denoising based on filtering the frequency domain. Scheidegger et al. [94] performs triangulations on the points sets. As the points may be non-uniformly dense and noisy, they have developed a triangulation that can adapt noises. Schall et al. [92] took a slightly different approach by incorporating the concept of probability in model smoothing. The vertices are moved towards the high probability density area. This method manages to diminish the outliers. Then, they adapts a non-local filter originally developed for image processing [93]. The filter is based on a measure of similarity, which is used to find similarity weighed averages across the whole set.

Qin et al. [91] use the bilateral filtering for point sets denoising. Their approach is very similar in concept to Fleishman et al. (which was discussed in the previous subsection). While Fleishman et al. use bilateral filtering for mesh smoothing, Qin et al. apply it on point set smoothing. Here, a local tangent plane is estimated from the PCA normal, and the filtering is applied to the points in the K-nearest neighbourhood.

2.4 Surface Reconstruction

In surface reconstruction, researchers look for effective methods of processing raw point set data obtained by optical acquisition devices into surface representations. Possible preprocessing steps may include point set denoising, while mesh denoising can be seen as a postprocessing step of surface reconstruction.

Most of the early work in surface reconstruction does not explicitly deal with noise in the data. As a result, the distinction between training error and test

error is only implicit, and data pre-processing, or model post-processing, is recommended for dealing with the problem of overfitting noisy data. [51; 68] and the Voronoi-based methods [4; 10] are early papers that influenced the development of the field.

2.4.1 Implicit surface

Implicit methods, pioneered in [51], are a major branch of surface reconstruction algorithms, which have demonstrated several advantages in terms of both robustness and computational efficiency [17; 18; 83].

In a typical pipeline, an isosurface is defined throughout the space. Then, an octree can be constructed, implicit function is computed with its component defined on the octree cells. Finally, water-tight triangle mesh is extracted with Marching Cube [114]. In many implicit reconstruction approaches the issue of noise is dealt with indirectly, through a user defined variable tuning the algorithm to the level of noise present in the data. For example, in [83] this parameter is the tolerance of local training error of the algorithm, while in [84] a regularization parameter is used.

2.4.2 Voronoi diagram

Voronoi diagram and Delaunay triangulation have been used by [4; 5; 7; 10] to construct surface from unorganized points. By heuristic arguments, the triangles are determined whether they should belong to the surface or not. This approach will naturally eliminate noisy data and outliers. If the noise level is close to the density of the sample data, however, the algorithm might not work as it cannot

differentiate the noise from the data.

Under the same motivation that the previous methods related to Voronoi diagram and Delaunay triangulation is not able to deal with noisy points, Mederos et al. [78] use the power crust algorithm to handle noisy point sets.

Voronoi based methods have been further extended, resolving issues such as holes in the surface [29], noise outliers [66] and finding conditions for provably good samples [3; 9].

2.4.3 Surface fitting

Surface reconstruction based on local polynomial surface fittings is becoming increasingly popular, especially in the form of Moving Least Square surfaces [71]. Most of these approaches do not explicitly deal with the noise [1; 46], or the algorithms require a user defined parameter indicating the level of noise [74].

One classic surface reconstruction paper is by Hoppe et al. [51]. Given a set of unorganized points, their method can create a surface reconstruction with no prior knowledge of topology nor boundary. The surface is reconstructed by minimizing the distance of the points to the unknown surface M. Then, through a contouring procedure, the meshes are built giving the reconstruction as a simplicial surface, here a triangle mesh. The Marching Cubes algorithm [75] is used in the contouring. By estimating a function that minimizes the signed distance of the points to the surface, we can loosely say that the approach is similar to polynomial regression in statistics. This approach is prone to overfitting.

Kobbelt et al. [65] regenerate polygonal meshes to transform meshes with arbitrary connectivity to a subdivision connectivity. Using a shrink wrapping

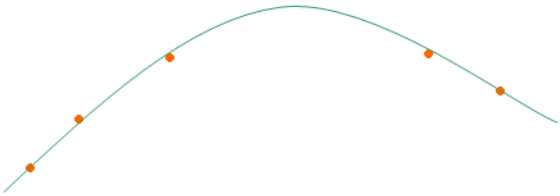

Figure 2.5: Example of the curve generated from an extrapolation. The missing data may be recovered.

approach, the base mesh is goes through subdivision until it produces appropriate models after iterations. The concept is like wrapping a plastic membrane around an object, producing the same object but with better connectivity. An example of bad connectivity is when we apply the marching cube algorithm straightforwardly on a model. A wrinkle may appear due to the different level of intensity of points in an area.

Carr et al. [18] used radial basis functions (RBF) in his approach. The radial basis functions are used as functions that approximate the signed distances in the neighbourhood of given initial data; the approach makes use of normal information, which is either given with the point set data, or estimated from them. By using an error minimisation technique, the local approximations are combined into a smooth surface. One of the advantages of this technique is that it may recover lost data. A simpler example is as shown in Figure 2.5, where a curve may estimate the value on the top part based on the given points by extrapolating the data.

Improving the approach of radial basis functions, Ohtake et al. [82] used compactly supported functions to produce a simpler and faster method. Explic-

itly they mentioned that this method is more robust than the one by Carr et. al, avoiding globally supported basis functions is one key point of improvement. Another improvement includes adaptively selecting the support of the RBF according to the properties of the data [84]. In this case, the centre of the RBF is chosen. Chen and Cheng [21] use an algorithm that can recover incomplete models and preserve sharp edges by marking the feature region and iteratively adjusting face normal on the mesh. Tagliasacchi et al. [99] introduced rotational symmetry axis (ROSA) and used skeletons and normal direction to construct missing data, reconstructing incomplete point clouds. Other papers that use RBF to recover defective or missing data are [31; 81; 106].

Xiao et al. [105] recover missing data by using texture synthesis. By mapping the texture on the existing data, the algorithm propagates the pattern over the missing data and reconstructs the area. [98] use volumetric priors to recover lost data, which works well on highly incomplete scans data, but does not work well with models with sharp edges.

Using MLS surface, [36] constructed piecewise smooth surface to preserve sharp edges. [41] applied local shape priors to preserve the sharp edges. Firstly, they defined a library of local shape priors where a feature is defined for a given shape of points (Refer to Figure 2.6). Then, in an augmentation process for a particular shape of points, they are able to reference a similar match from the library. The algorithm can simply be seen as a search and match procedure, but is reliable as it also includes some criteria in the decision making process, such as the directions of the normals. This method is effective in preserving the sharp edges but is very time consuming. They stated themselves that this method is very expensive for highly dense samples; besides, defining a library for local shape

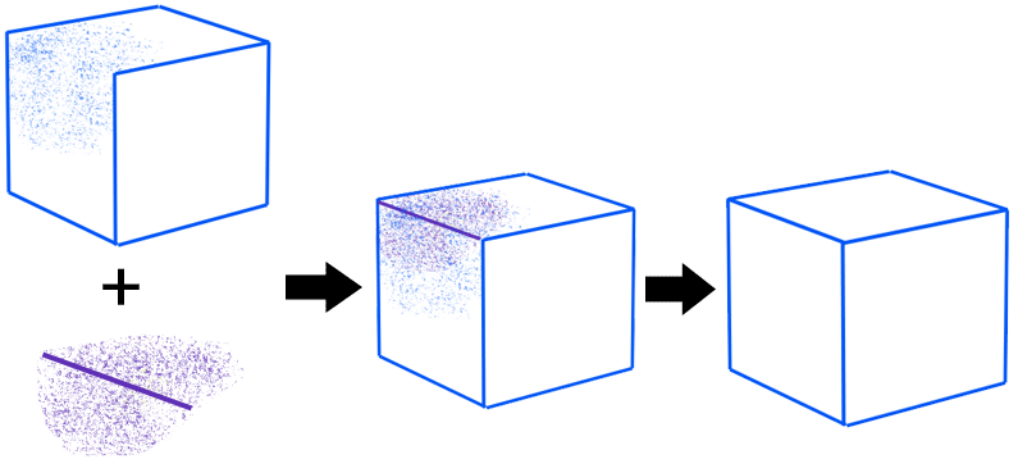

Figure 2.6: The purple figure is an example of local shape prior. The cube is going through an augmentation procedure where a potential sharp edge is matched with a local shape prior, therefore producing a final result where the sharp edge is preserved.

priors is also a tedious work.

Ohtake et al. [83] defined an implicit model called Multi-level Partition Unity (MPU) which approximates the surface using repeated subdivisions of the region of space. This approach is good at preserving features but very sensitive to the quality of input data. [104] improves MPU by adapting the method on high curvature regions and around feature areas.

Other surface fittings include B-spline surface [68] which is used widely in modeling industry, but has several disadvantages. It performs poorly for complex surfaces, is more expensive, and does not handle sharp edges and holes very well.

Surface reconstruction algorithms can be used for point set denoising by projecting the data points on the reconstructed surfaces. Other point set denoising techniques use spatial [91] or spectral filtering [89], and anisotropic diffusion [69].

2.4.4 Statistical approach in surface reconstruction

Statistical methods for surface reconstruction deal with the noise more directly. In [56; 90] a Bayesian approach is employed for surface reconstruction and smoothing, with the level of noise required as a user input. In [109] a variational Bayesian approach is employed to automatically estimate the level of data noise. In [109] comparisons between models are possible through the computation of the log evidence.

Kahzdan et al. [63] applied the Poisson formulation in their work. By incorporating noisy points with normals into the Poisson equation, the solution will approximate a smooth surface. Instead of solving each point in its own local neighbourhood, this method works globally by solving on all points at the same time. However, if there is missing data, it will try to connect the region instead of leaving it for a recovery process to fix.

Pauly and Gross [85] showed how the uncertainty can be incorporated into a geometric model. However, the assumption is that the amount of noise (variance) is known. [60; 61] developed a hierarchical method for representation and fast rendering of models incorporating uncertainty by using probabilistic models.

Using Bayesian statistics, Jenke et al. [56] define a probability distribution over the points. The noisy points have different probability depending on the location and the points which lie on the surface should have a higher probability than the outliers. Bayes' rule is used to estimate the points that belong to the model and to smooth out the noise at the time of reconstruction. To preserve the sharp edges, they applied the augmentation procedure based on a curvature-based edge detector. The procedure is incapable of detecting edges if the noise

level is high, however.

Qian et al. [90] used the same approach with Bayesian statistics to construct meshes from noisy point data. However, their approach did not emphasize sharp edge preservation as Jenke et al. did.

Yoon et al. [110] use a model averaging method called *ensemble* to deal with outliers. They define the general framework of ensembles on surface reconstruction. A sample data set is run repeatedly to create different models. After going through a deterministic algorithm which analyses the error, the models are combined to produce an ensemble result. While cost-efficiency is one of their concerns for improvement, it may also be useful to integrate a method which can preserve features and sharp edges.

Ivrissimtzis et al. [54] relate mesh representation with the concept of neural network. These neural meshes are constructed iteratively using Growing Cell Structures [40], which are similar to coarse-to-fine interpolation as in [82]. The new insertion of vertices to form the neural meshes is influenced by the probability density of the input signals to minimize the error. [77] improved the previous method to produce a more accurate construction around concave structures in a model.

Regarding statistical approaches in the wider area of graphics and computer vision, Bubna et al. [15] compared statistical approaches in this area. By experimenting with various model selection criteria on 3D models, they suggested using Bayesian criteria and bootstrapped variants for model averaging. Akaike information criterion increases bias for higher order polynomial fitting. So, it should be avoided.

The statistical approach is effective when it comes to outliers and estimating

error. We are still, however, seeking a model which is not only statistically good, but also visually correct. Thus, the issue of feature preservation should also be incorporated in statistical approaches.

2.5 Statistics

Traditionally, the researchers in geometric modelling try to construct a good model by removing the noise heuristically, using geometric methods . It may be interesting to point out that in statistics, there are various approaches in understanding and dealing with noisy data sets.

Lee et al. [70] points out that there is not much integration between these two fields. However, there have been a number of researchers using statistical approaches in order to solve surface reconstruction problems. These papers will be included in next subsection.

Note that while discussing the following methods, there is not much existing research on how they can be applied specifically to the area of graphics. Most of them are generally defined while we need it specifically to our setting.

In statistical modelling we try to find the best model that fits the data. We then estimate the error to assess a model. It has the similar principal as surface reconstruction, which finds the best surface representation assuring that the model is as faithful as possible to the original object.

There are two types of error: *training error* and *test error*. These type of error are discussed much further in Section 3.1. These two errors are brought up for discussion, as most smoothing algorithms, such as Laplacian or MLS smoothing, are using training error for their error estimation. Minimizing training error as a

way to find the best model may still produce a good result visually, but should be regarded as a bad assessment because it may not be faithful to the data. It may lead to overfitting, which favours complex models and do not generalise well for independent data.

2.5.1 Model averaging and model selection

Given N points of a data sample, we may calculate statistical measurements such as mean, variance or standard deviation of the sample. [33] had introduced a model averaging method called *bootstrap method* to estimate sampling distribution. Using bootstrap procedure, we are creating multiple data sets from the same sample we used previously. Each new data set consists of N points taken randomly from the data sample, allowing repetition of any point. We may combine the statistical measurement of the data sets and see the probability distribution for the sample.

Cabrera and Meer [16] uses bootstrap method to fit ellipses on 2D data. In a 2D setting, they presented methods to estimate splines from the noisy data using bootstrap method. Comparing the result with least square fitting, they showed visually that it is more accurate to use the bootstrap method. Using the same motivation, we will apply the same approach in a 3D setting. The ensemble technique, which has several similarities with the bootstrap method, was used for surface and normal reconstruction in [110].

In contrast, our focus is on test error estimation and overfitting detection. In [70] the test error of implicit models is locally estimated through extra sample validation. The method there is similar to the leave-one-out bootstrap estimator,

which, as we will see, overestimates the test error. The estimated test error is guiding a sophisticated algorithm for building a multi-scale implicit model, and no actual test error values are reported.

Another example of model averaging method is bagging. By aggregating bootstrap samples, this method may be used to lower the variance [11; 49]. This method may increase the accuracy of model prediction and avoid overfitting. Existing model prediction faces the bias-variance dilemma: that is increasing bias while decreasing the variance and vice versa. In geometric computing, we may take the classic example of Laplacian smoothing where bias increases while variance decreases after every iteration. Even recent approaches using curvature flow suffer from the bias-variance effect. In that case, the decreasing of bias is countered by volume preservation. Breiman [12] claimed that he managed to reduce bias and variance simultaneously by using iterated bagging.

2.6 Feature Detection

Given its importance in geometric modelling applications [76], feature detection and feature extraction has received considerable research interest, both for 3D point sets and polygonal meshes.

Regarding feature detection on point sets, Yang et al. [107] use PCA to compute principle curvatures on neighborhoods of varying size. The curvature information can then be used for feature detection. Similarly, Gatzke and Grimm [42] use quadratic fitting to compute a curvature map, which is then used to detect and classify features. Basdogan and Oztireli [8] focus on the application of point set registration. Its main descriptor for feature detection uses the distance

between a point and the centre of mass of its neighbourhood. Li and Guskov [73] first smooth the point set with the MLS projection and then define as features those areas where the difference between normals of neighbouring points is at maximum. Gumhold et al. [47] analyse the eigenvalues and eigenvectors of a neighbourhood and computes the probability that a point is a certain type of feature, such as sharp edge, boundary or corner. Demarsin et al. [26] use normal computations to obtain a segmentation of the point set and then detects sharp features between the segments. Joel et al. [25] identify as features the points with the highest error in a RMLS polynomial fitting. Chica [22] produces a voxelised model of the point data, which is then segmentised using visibility computations. The features are extracted from the segmentised voxelisation. [79] used Voronoi-based method and computed covariance matrices on Voronoi cells. The matrices provide curvature information and feature is detected by thresholding area with relatively higher curvature.

Considerable research effort has also been directed into the computation of features on polygonal meshes. The techniques applied there are similar to those for point sets. Yoshizawa et al. [111] use polynomial fittings to compute curvature information which is then used for detecting feature lines on triangle meshes. Sun et al. [97] use eigenanalysis on a tensor, which is computed as an average weighted by the geodesic distance of tensors of neighboring normals. Wang [102] analyses angles between normals to detect features and then uses bilateral filtering to reconstruct them. Ohtake et al. [84] detect features by first constructing an implicit model of the data and then computing curvature information from the implicit model.

Feature detection is an ill posed problem. Methods are usually evaluated with

visual inspection, and detection on point sets is more difficult than detection on meshes.

2.7 Summary

In studying surface reconstruction, researchers have come up with various type of approaches. Many stand-alone surface reconstruction methods are available, but they may not be able to handle issues like feature preservation or outliers. This is why some researchers handle this issue separately. For instance, computation of normal or denosing is done separately as a pre- or post-processing step, depending on where it falls on the surface reconstruction pipeline.

While there is a vast amount of literature regarding surface reconstruction, it is only recently that statistics have been brought into geometric computing. While statistics itself had developed a solid base in error estimation and model approximation, it had not been applied much in this field. The terms are not well-defined in our context.

Integrating the statistical approach into geometric modelling should never be seen as starting from scratch. We aim to apply existing methods, which are useful but complex, and transform them into a form suitable for geometric computing, creating accurate and robust algorithms.

Chapter 3

Background

In this chapter, we will define and summarise some of the terms that we will be using in this thesis. We will start with the definition of error and the way that we will be using in this thesis. Secondly, bias-variance decomposition will be discussed. Bias-variance decomposition is a reaction due to different variables used to estimating data. Both of these are already well-defined in statistical areas. Here, however, we extend the discussion into the context of use in the research presented. Our input data is in $\mathbb{R}^3$ and we use surface polynomial fitting which requires a 2.5 dimensional setting with plane and height function. In final part of this chapter, we apply *localisation* for this purpose.

3.1 Types of error

In surface reconstruction, our input data comes in the form of point clouds from the acquisition procedure. The data are given in $\mathbb{R}^3$. However, in this section, we will mostly discuss the definitions in 2D for the sake of simplicity. The terms

are often defined in a 2D setting before going into higher dimensional setting.

In statistics, a *prediction model* is a function which estimates input data. Points scattered on 2D plane with a Cartesian coordinate system can be fitted using numerous functions such as polynomial fitting, exponential series and Fourier series. The aim is to choose a function or model which best represents the data; this procedure is known as *model selection*. A typical way to select a good model is to rely on the value of error.

There are two main types of error; *training error* and *test error* which is also called generalisation error. In a 2D setting, given a set of samples $(x_i, y_i) \in \mathbb{R}^2$ for $i = 1, 2, ...n$, a function $f(x_i) = Y_i$ is estimated to fit the data. Training error of the model is the average loss over the sample

$$\frac{1}{n}\sum_{i=1}^{n}|Y_i - y_i| \tag{3.1}$$

Test error is defined as the average loss over independent test sample and generally it is more difficult to estimate. For example, if we have a large number of samples, we could exclude some for the purpose of verification. Given a sample y_i with $i = 1, 2, ..., n-1, n, n+1, ..., m-1, m$, we use the first n samples to estimate fit a model and estimate training error. The remaining $m - n$ samples could be our inputs for test error as follow

$$\frac{1}{m-n}\sum_{i=n+1}^{m}|Y_i - y_i| \tag{3.2}$$

In practice, instead of taking the first n samples, we will use n samples in a random sequence instead of an incremental sequence.

Training error usually underestimates the expected error of the model on an

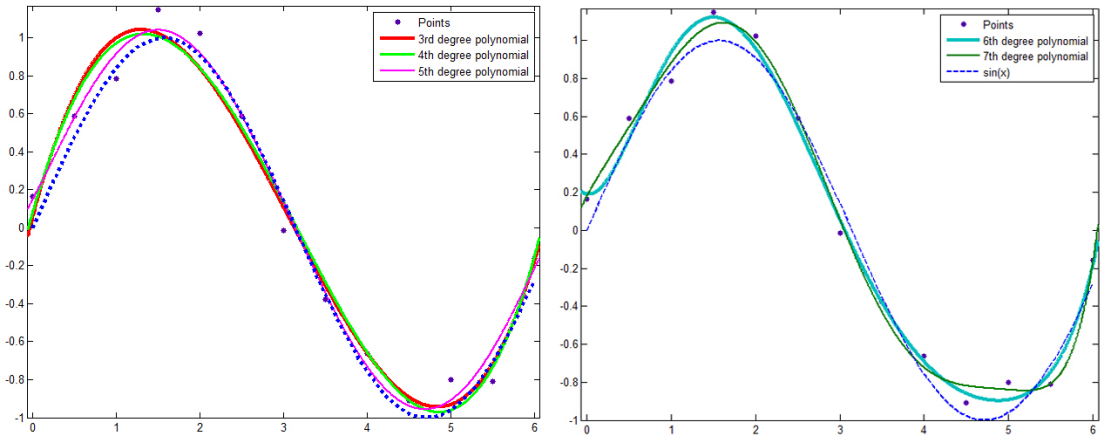

Figure 3.1: Example of overfitting

independent sample. If we use training error as our error estimation, we may obtain a smaller value of error than we are supposed to have. This is due to a phenomenon called optimism of the training error. This phenomenon is related to the concept called *overfitting,* in which the data is fitted very closely to the samples, which may reduce the training error.

We illustrate overfitting in an example. We generate points from a sinus function with added noise, which is a maximum of random value up to 0.2 in positive or negative direction as shown in Figure 3.1. In practice, one may choose cubic or quartic fitting to fit the given points, rather than a degree 6 or higher which intuitively will overfit the points. Referring to Figure 3.1(left), we can see that cubic, quartic and quintic fittings estimate the points rather well. The mean square errors (which are forms of training error) for cubic, quartic and quintic are 0.1386, 0.1444 and 0.1273 respectively. As the training error for quintic is the lowest, it may indicates that the quintic is the best fitting for this data. Without referring to the actual function before noise, which is represented by the dotted

blue line in the figure, we can assume that quintic polynomial fits better as it passes through a lot of points.

However, relying on the value of training error may misguide us. In Figure 3.1(right), we show the fitting of sixth and seventh order polynomial and the mean square errors are 0.1026 and 0.0883 respectively. Without a visual aid, one may assume that the seventh order polynomial fitting is better but the figure proves otherwise. Overfitting of this estimate make it a worse choice. Although quintic fitting has a higher training error, we may favour it as opposed to the seventh order polynomial fitting. As mentioned before, one would not normally choose such a high order for a polynomial fitting, but here is an example where we may question if we can actually rely on the training error.

Another term we will use in this thesis is *ground truth error*. We often assume that samples came from a smooth model with some noise. Assuming that the actual model is known and (x_i, y_i^t) are points of the model, we introduce the term *ground truth error,* which is defined by replacing y_i^t in Equation 3.1. For the previous example, this would be data values before the noise is added. While this may not be made possible for data acquired in our study, the discussion and comparison to this value is important for verification of our method. In this case, $y = sin(x)$ will be used to obtained the ground truth error. We run through different polynomial fittings and provide the training error and ground truth error as shown in Table 3.1. As we can see, training error is not a good estimate of the ground truth error. Thus, for a more reliable model assessment, we need to develop methods for estimating the test error.

For this research one of our aims is to use a method which can estimate test error in such a way that our reliance on visual assessment of the fitting can be

minimized. While it is relatively easy to check for a 2-dimensional case, it is more tricky when the data is 3-dimensional.

Notice that some smoothing algorithms, such as MLS smoothing, are using training error for their error estimation. Minimizing training error as a way to find the best model may still produce a good result visually, but may result in overfitting. This favours complex models which do not generalise well for independent data.

Degree of Polynomial fitting	Training error	Ground truth error
3rd	0.1386	0.0786
4th	0.1444	0.0744
5th	0.1273	0.0490
6th	0.1026	0.0988
7th	0.0883	0.1817

Table 3.1: Training error and ground truth error for the polynomial fitting of different degree. The data is $\{sin(x) \pm rand[0, 0.2], x = 0 : 0.25 : 6\}$

3.2 Bias variance decomposition

To extend the discussion regarding training error, we would like to discuss about *bias-variance decomposition*.

Following what had been described in [49] and also described in [43], test error can be decomposed into different parts given by

$$\sigma^2 + \text{Bias}^2(f(x)) + \text{Var}(f(x)) \tag{3.3}$$

where σ^2 is variance of the new estimate, which is irreducible error. $\text{Bias}^2(f(x))$ is defined as mean squared error, which is the squared value of training error given

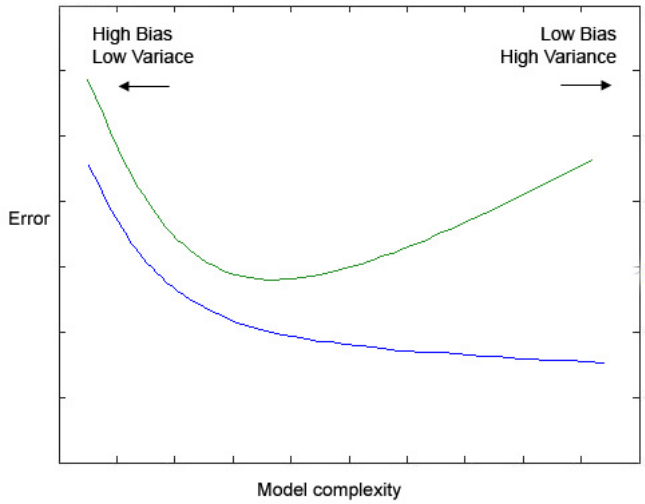

Figure 3.2: Error estimation from bias variance decomposition (green curve) and training error (blue curve).

in Equation 3.1. $\mathrm{Var}(f(x))$ is the variance as generally defined in common literature as $E[f(x)^2] - (E[f(x)])^2$ where $E(x)$ is mean of the samples x_i's.

Model complexity is the ability of the model to adapt to a more complicated data. In geometric modelling, a complex model is one with more parameters. For example, in splines, increasing the number of control points will increase model complexity. In our case, when we use polynomial fitting, the complexity is the order of the polynomial. Looking at Figure 3.1 as an example, we can see that a cubic fitting may give a better fit for the data as it able to adapt the sinus shape in the given range, compared to a quadratic or linear fitting.

Figure 3.2 shows the behaviour of the issue that we dealt with in the previous section. When we increase the model complexity (in previous case it was polynomial degree), training error, which is referred in this case as bias, will decrease. This is why it is not reliable to depend on training error to find the quality of an estimate. While bias reduces, variance increases and affects test error. As showed

by the green curve, a low test error might be obtained when we are able to find the middle ground of bias and variance.

In geometric computing, [55] defined an inaccurate model as having a high value of bias, while truly model-free inference may cause a high variance. A simple example can be discussed in the case of classic Laplacian mesh smoothing. After a few iterations, the model will be smoothed out, which can be seen as an action of reducing the variance of the model. However, it will also shrink and produce an inaccurate model hence having a higher value of bias.

While bias-variance decomposition would not be a part of our research, this concept will be of assistance in understanding the property of test error in later chapters.

3.3 Localisation

One of our main tools in most of the proposed algorithms is to fit a polynomial surfaces on a selected neighbourhood of point clouds. Although we are given 3-dimensional data, we fit the surface in 2.5-dimensions such that $z = f(x, y)$ is a height function for given point (x, y). As all of the methods are applied locally, we assume that the projection of the surface on the tangent plane gives a good local parameterisation. This is a standard assumption in all local method in surface reconstruction, simplifying the model from 3D to 2.5D. Without local parameterisation, numerical instability may occur when we apply polynomial surface fitting on a neighbourhood of points.

It is vital to select a coordinate system of points so that the points expand over the domain rather than scatter vertically around the z direction. Hence we

are going through a process which we called *localisation*.

Our data sets are given in 3-dimensional coordinates $(x, y, z) \in \mathbb{R}$, and all of our approaches use a local neighbourhood of points instead of dealing with all the points in a model. To estimates a model, we will be using surface polynomial fitting, and next we will define the steps before fitting a polynomial surface.

Given a point set of $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ with N points, we search for K-nearest neighbourhood for each point, such that for d_i, we obtain a neighbourhood of $P_i = \{p_1^i, p_2^i, p_3^i, ..., p_K^i\}$. When there is no room for ambiguity, we may drop out the superscript i in P_i. We may also refer this i-th neighbourhood as $\mathbf{P}$. We will use the neighbourhood $P_i = \{p_1, p_2, p_3, ..., p_K\}$, to compute the normal at d_i and obtain $\mathbf{n}_i = (n_1, n_2, n_3)$ which will give the tangent plane of the surface at d_i. We generally use Principal Component Analysis to estimate our normals. Note that these normals are not oriented and existing literature such as [67] can be followed to orientate normals on a model.

To fit a surface polynomial, we orientate P_i from the Cartesian coordinate system $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$-axes centered at $(0, 0, 0)$ to $(\mathbf{X}', \mathbf{Y}', \mathbf{n}_i)$-axes centered at d_i. We follow the orientation step as described in [38] .

Note that these two set of axes contains vectors where $\mathbf{X} = (1, 0, 0)$, $\mathbf{Y} = (0, 1, 0)$ and $\mathbf{Z} = (0, 0, 1)$, while $\mathbf{X}' = (u_1, u_2, u_3)$ and $\mathbf{Y}' = (w_1, w_2, w_3)$ are arbitrary chosen vectors which satisfy the condition that $(\mathbf{X}', \mathbf{Y}', \mathbf{n}_i)$-axes is orthornomal. Rotation matrix M which will transform the $\mathbf{X}'\mathbf{Y}'\mathbf{n}_i$ to $\mathbf{XYZ}$ is given

by

$$M = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ w_1 & w_2 & w_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

We wish to transform $\mathbf{XYZ}$ to $\mathbf{X'Y'n}_i$ instead. As $M\mathbf{X'} = \mathbf{X}$, so $\mathbf{X'} = M^{-1}\mathbf{X}$ whereas $M^{-1} = M^T$, where M^T denotes the transpose matrix of M. Hence, for any point p_i that we wish to transform to $(\mathbf{X'}, \mathbf{Y'}, \mathbf{n}_i)$-axes centered at d_i, we can obtain p'_i such that

$$p'_i = M^T \begin{bmatrix} p_{i,x} - d_{i,x} \\ p_{i,y} - d_{i,y} \\ p_{i,z} - d_{i,z} \\ 1 \end{bmatrix} \tag{3.5}$$

by neglecting the 1 in the last row.

As a note, after we have gone through polynomial surface fitting, we should orientate back to the $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$-axes, which is given by

$$p_i = M \begin{bmatrix} p'_{i,x} \\ p'_{i,y} \\ p'_{i,z} \\ 1 \end{bmatrix} + \begin{bmatrix} d_{i,x} \\ d_{i,y} \\ d_{i,z} \\ 0 \end{bmatrix} \tag{3.6}$$

and again neglecting the 1 in the last row.

Chapter 4

Bootstrap Error Estimates

Background for this chapter has been discussed in Section 2.5. We aim to have an objective assessment method to study based on estimations of test error. Hence, we define a statistical method called bootstrap in context of surface reconstruction. We will test the algorithm on various models including scanned data.

Section 4.1 - 4.2.1 have been presented in

- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Bootstrap test error estimations of polynomial fittings in surface reconstruction", *In VMV*, pages 101-112, 2009

4.1 Bootstrap surface reconstruction

In this section we briefly describe the bootstrap method and the several error estimators we can compute. The bootstrap method is based on repetitive random resampling of the data and the averaging of the results obtained from each sample. The reuse of the data as a result of the repetitive resampling is particularly helpful

in cases where the available data are sparse, as is the case with local surface reconstruction.

For further details on the bootstrap method, we refer to a statistical learning handbook such as Hastie et al. [49].

4.1.1 Creation of bootstrap sets

Let the set of training data $\mathbf{P} = \{p_1, p_2, \ldots, p_K\}$ be of size K where p_i is a 3D point. We use principal component analysis to estimate a tangent plane for $\mathbf{P}$, and we locally parametrize $\mathbf{P}$ over it as in Section 3.3. That is, each point p_i is written as $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is its projection on the tangent plane and y_i is its distance from the tangent plane.

We randomly sample $\mathbf{P}$ with replacement to draw a bootstrap sample S_b with the same size as $\mathbf{P}$. As we sample with replacement, the expected number of distinct elements in S_b is lower than K. In fact, we can easily calculate that the expectation for the number of distinct elements in S_b is

$$K \cdot (1 - \frac{1}{e}) \approx K \cdot 0.632 \tag{4.1}$$

The sampling procedure is repeated B times and the bootstrap samples

$$S_1, S_2, \ldots, S_B \tag{4.2}$$

are generated. The choice of B is a trade-off between accuracy and computational cost. The more bootstrap sets we process, the higher the reliability of the estimates.

Algorithm 1 Algorithm for bootstrap error estimation

Algorithm `bootstrap_error()`

Given point sets of $S_N = \{d_1, d_2, ..., d_N\}$, the algorithms estimate the bootstrap error for neighbourhood of d_i.

Define Err_{boot} and $\hat{Err}^{(1)}$ as zero

Get $\mathbf{P}^i$, the K nearest neighbourhood for d_i

Estimate PCA normal

Localize $\mathbf{P}^i$ to a 2.5D centered at d_i

for j=1:K

Run bootstrap as follow

for b=1:B

Generate a random subsampling

Fit the polynomial on the bootstrap samples

$|f^{*b}(\mathbf{x}_j) - y_j|$ is computed and added to Err_{boot}

if the subsample does not include p_j

$|C_i| = |C_i| + 1$

$|f^{*b}(\mathbf{x}_j) - y_j|$ is added to $\hat{Err}^{(1)}$

end if

end for

Delocalize the points and obtain estimated projection from bootstrap

Obtain training error, bootstrap error Err_{boot} from Eq. 4.3 and leave one out error $\hat{Err}^{(1)}$ from Eq. 4.4

end for

4.1.2 Bootstrap error estimations

On each bootstrap set S_b we fit a model f^{*b}. In our context, that means a polynomial surface of order M is fitted on the locally parametrized bootstrap set using a distance minimization algorithm.

For a neighbourhood of points, the bootstrap error estimation Err_{boot} is the average distance between the predictions of the bootstrap models on the data and the actual values of the data

$$Err_{boot} = \frac{1}{B}\frac{1}{K}\sum_{b=1}^{B}\sum_{j=1}^{K}|f^{*b}(\mathbf{x}_j) - y_j| \tag{4.3}$$

As one would expect, the bootstrap error, generally, underestimates the test error because some data points are used both for surface fitting and for error estimation. To solve this problem, one can compute the *leave-one-out* bootstrap estimate of the test error

$$\hat{Err}^{(1)} = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{|C_j|}\sum_{b\in C_j}|f^{*b}(\mathbf{x}_j) - y_j| \tag{4.4}$$

where C_i is the set of indices of bootstrap sets that do not contain the point p_i and $|C_i|$ denotes the size of the set C_i. Implementation of bootstrap error and leave-one-out error is shown in Algorithm 1.

Unlike the bootstrap error Err_{boot}, the leave-one-out error $\hat{Err}^{(1)}$, generally, overestimates the test error. This is due to the training-set-size bias phenomenon, which is related to the fact that $\hat{Err}^{(1)}$ uses training sets of size smaller than K. To solve this opposite problem, we follow [34]. One can use the *.632* estimator

$$\hat{Err}^{(.632)} = .368 \cdot \bar{err} + .632 \cdot \hat{Err}^{(1)} \tag{4.5}$$

which is a weighted average between the underestimating training error

$$\bar{err} = \frac{1}{K}\sum_{j=1}^{K}|f(\mathbf{x}_j) - y_j| \tag{4.6}$$

where f is the model fitted to the whole data set $\mathbf{P}$, and the overestimating leave-one-out error $\hat{Err}^{(1)}$. The .632 estimator reduces the upward bias of the bootstrap error. The .632 constant is related to the probability of a given data point to be member of a certain bootstrap sample (See Equation 4.1).

[35] improved the .632 estimations by the .632+ estimator

$$\hat{Err}^{(.632+)} = (1-w)\cdot \bar{err} + w \cdot \hat{Err}^{(1)} \tag{4.7}$$

with

$$w = \frac{.632}{1 - .368 \cdot R} \tag{4.8}$$

with

$$R = \frac{\hat{Err}^{(1)} - \bar{err}}{\gamma - \bar{err}} \tag{4.9}$$

where γ is the *no-information error rate* given by

$$\gamma = \frac{1}{K^2}\sum_{j=1}^{K}\sum_{k=1}^{K}|f(\mathbf{x}_j) - y_k| \tag{4.10}$$

Intuitively, the .632+ error estimate is also a weighted average of the underestimating training error and the overestimating leave-one-out error. Unlike the .632 error, however, the weights are not fixed, but depend on the amount of overfitting on the original data, as this is estimated by the *relative overfitting rate* R in Eq. 4.9.

According to [35], the value of R is zero when there is no overfitting ($\hat{Err}^{(1)} = \bar{err}$). In this case, .632+ error is the same as the .632 estimator. When the *no information rate*, γ approaches the value of leave-one-out error (see Equation 4.9), R will become 1 and the .632+ error will become the leave-one-out error.

This happen when there is no information whether overfitting occurs or not, deriving the term of no-information.

4.2 Results

We tested the proposed bootstrap test error estimation method on synthetic and natural data. For synthetic data, starting from a smooth model, we simulate raw data by adding to each point a random displacement in the direction of the previously computed normal. A noise level equal to ξ refers to displacements of magnitude uniformly and randomly sampled from the interval $[0\,, \xi h]$, where h is the average, throughout the model, of the distance between a point and its nearest neighbour. The original points are used as the ground truth against which we measure the error of the estimated normals.

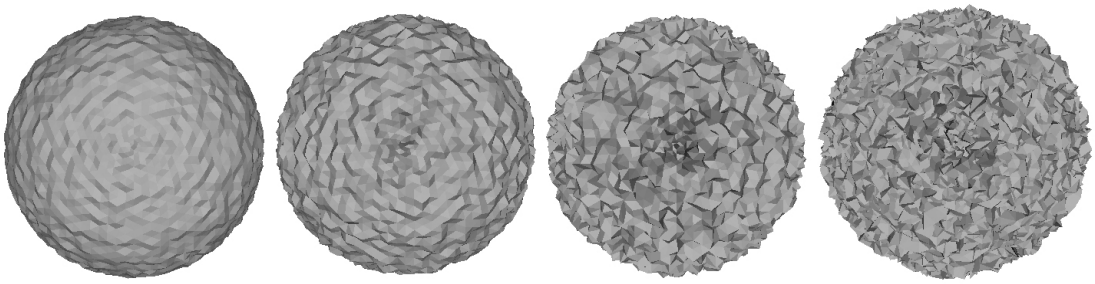

Figure 4.1: Sphere with different noise levels. From left to right, the noise level is 0.25, 0.50, 1.0 and 1.50.

The validation of test error estimations in the context of surface reconstruction from noisy point sets is a challenging task, as it is difficult to obtain reliable independent estimates of the test error to compare against.

In a first experiment, we used a set of 26,010 points lying on a sphere, added a constant amount of noise to them, and then computed several error estimators on 50 and 100 nearest point neighborhoods for each point. Fig. 4.1 shows the sphere model at various levels of noise. As the underlying surface is simple and regular one expects most of the test error to be the result of the data noise. However, as the initial model is smooth, the data noise is approximately equal to the added noise, which is a known quantity. Apart from the data noise, a second source of test error is the possible overfitting or underfitting. As the initial model is simple and regular, any underfitting and overfitting would be related to the size of the local neighborhoods and the order of the polynomial fitting, which are controllable quantities, rather than the features of the underlying surface, which in the case of the sphere do not exist.

Table 4.1 (columns 2-6) shows the average error estimations on the sphere for polynomials of order three and four and neighborhoods of size 50 and 100. We report the training, leave-one-out, .632 and .632+ errors. We also report a leave-one-out error estimate for the center only of the neighborhood, using the equation

$$\hat{Err}_p^{(1)} = \frac{1}{|C_p|} \sum_{b \in C_p} |f^{*b}(\mathbf{x}_p) - y_p| \tag{4.11}$$

where C_p is the set of indices of bootstrap sets that do not contain the central point $(\mathbf{x}_p, y_p)$.

In all experiments, the unit distance is the average throughout the model of

the distance between a point and its nearest neighbor. In all experiments, the number of bootstrap sets was $B = 50$.

	Noise	$e\bar{r}r$	$\hat{Err}_p^{(1)}$	$\hat{Err}^{(1)}$	.632	632+	Mean γ	Max γ
Cubic	0.0	0.0011	0.0013	0.0016	0.0014	0.0014	0.0014	0.0014
	0.5	0.4206	0.5085	0.5735	0.5172	0.5725	0.5710	0.5514
N=50	1.0	0.7270	0.8915	0.9899	0.8931	0.9894	0.9886	0.9599
	2.0	1.2126	1.5945	1.6602	1.4955	1.6568	1.6555	1.6073

	Noise	$e\bar{r}r$	$\hat{Err}_p^{(1)}$	$\hat{Err}^{(1)}$	.632	632+	Mean γ	Max γ
Quartic	0.0	0.0010	0.0015	0.0019	0.0016	0.0016	0.0016	0.0016
	0.5	0.3897	0.5479	0.7088	0.5914	0.7084	0.7079	0.6575
N=50	1.0	0.6806	0.9312	1.2332	1.0298	1.2332	1.2322	1.1518
	2.0	1.1463	1.6010	2.1309	1.7686	2.1306	2.1302	1.9790

	Noise	$e\bar{r}r$	$\hat{Err}_p^{(1)}$	$\hat{Err}^{(1)}$	.632	632+	Mean γ	Max γ
Cubic	0.0	0.0015	0.0015	0.0018	0.0017	0.0017	0.0017	0.0017
	0.5	0.4509	0.4942	0.5154	0.4917	0.5065	0.5048	0.5001
N=100	1.0	0.7725	0.8538	0.8834	0.8426	0.8806	0.8786	0.8677
	2.0	1.2799	1.4827	1.4676	1.3985	1.4657	1.4642	1.4483

	Noise	$e\bar{r}r$	$\hat{Err}_p^{(1)}$	$\hat{Err}^{(1)}$	.632	632+	Mean γ	Max γ
Quartic	0.0	0.0012	0.0013	0.0015	0.0014	0.0014	0.0014	0.0014
	0.5	0.4358	0.5037	0.5440	0.5042	0.5363	0.5325	0.5221
N=100	1.0	0.7501	0.8511	0.9360	0.8676	0.9346	0.9324	0.9136
	2.0	1.2479	1.3784	1.5637	1.4475	1.5625	1.5607	1.5336

Table 4.1: Bootstrap error estimations.

The comparison between cubic and quartic *neighborhood error estimates* shows that quartics have smaller training error, fitting the data more closely, as expected. However, in neighborhoods of size 50, for any amount of added noise, the leave-one-out, .632 and .632+ errors of the quartics are significantly higher than those of the cubics, showing that the smaller training error was a result of overfitting. We can see this effect shown in Figure 4.2. While training error for both cubic and quartic are relatively low, the .632+ errors shows a higher value

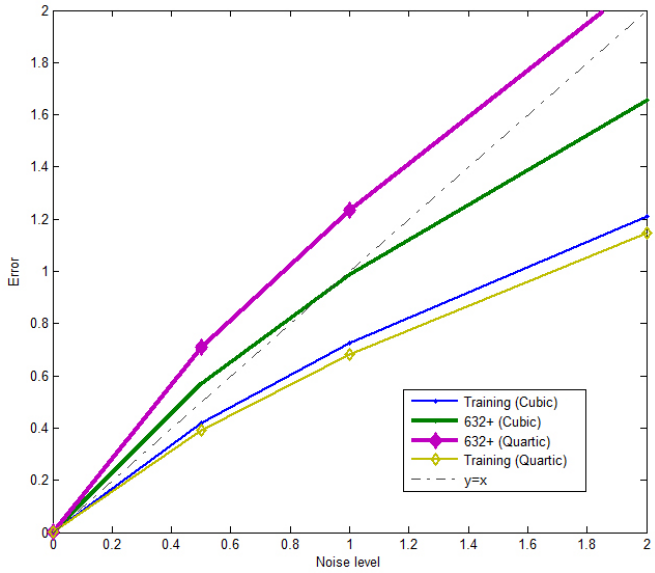

Figure 4.2: Training error and 632+ error on cubic and quartic fitting for different noise level. The neighbourhood size is 50.

of error. Although the training error is lower for quartic as compared to cubic, the .632+ error is higher for quartic because of overfitting.

In neighborhoods of size 100, the neighborhood error estimates on cubics and quartics are comparable. Thus, for neighborhoods of size 100 the use of quartics can be justified as they do not significantly overfit the data.

The comparison between cubic and quartic *point error estimates* shows little evidence of overfitting. That means that even in relatively small neighborhoods of size 50, near their center there is enough geometric information that even a quartic polynomial does not overfit. As a result of the better quality of information in the center of a neighborhood, we also notice that the point error estimates are lower than the neighborhood error estimates.

The comparison between neighborhoods of size 50 and 100 shows that the corresponding errors are comparable, except of the quartic leave-one-out, .632

and .632+ errors which are significantly smaller for the larger neighborhoods of size 100. The fact that the error of the quartics drops while the amount of data is increasing is another confirmation of the overfitting of the smaller neighborhoods of size 50 by the quartics. Obviously, small neighbourhood are easily overfitted.

A final observation is that all estimators tend to underestimate very large amounts of noise. We believe that this is an artifact of the validation experiment. In particular, the added noise is reported as a displacement in the normal direction of the initial smooth data. In contrast, the test error is computed by distance measurements in the normal direction of the noisy data. For large amounts of noise, the difference between the two normals becomes significant, affecting the correspondence between the added noise and its estimates.

4.2.1 No-information error rate

The values of the .632+ error in Table 4.1 are very close to the values of the leave-one-out error, indicating an average relative overfitting rate R near 1. In fact, several times the validation experiment returned values of R that exceeded 1. In these cases the value of R was capped to 1 to be inside the interval [0,1] of its theoretical range.

To explain the experimental behavior of R, we notice that the denominator of Eq. 4.9 tends to zero when γ approaches $e\bar{r}r$ and the computation of R becomes unstable. In our case in particular, the local parameterization uses the tangent plane. The numerical difficulties exist because in our setting, we might encounter a planar surface resulting to $\gamma = e\bar{r}r$. When this happen, R would not serve its purpose of detecting overfitting, thus the model is non-informative.

We propose two alternative computations of γ which result in more robust computations for R. The first method, instead of computing the γ on the training set, uses the mean of γ over all bootstrap samples. The second method uses the maximum of γ over all bootstrap samples. Notice that, in the second case, the γ used in the computations is systematically higher than the expected value of the γ in Eq. 4.10. In this case, however, we are not interested in the value of γ itself, but in the value of R and larger values of γ which lead to more robust computations for R.

The resulting .632+ error estimates are shown in Table 4.1 (columns 7-8). As expected, the use of the maximum γ gives .632+ estimates further away from the leave-one-out error; however, in this experiment on the sphere, there is no sufficient numerical evidence to verify that these are more accurate estimates of the test error.

As a second experiment, we compare our approach with [49]. Fig. 4.3 shows colormaps of R computed with the three different values of γ. The colormap of the values of R computed as in [49] verifies that the overestimation of R is restricted in the flat areas of the model. The computations of R using the mean and the maximum γ show an improved behavior of R, with a larger number of neighborhoods returning values smaller than one.

4.2.2 Validation on natural models

To validate the relevance of the technique when dealing with natural models, we computed .632+ error estimates on the Bimba model with different amounts of added noise. The model has 74,764 points. We still assume that the model is

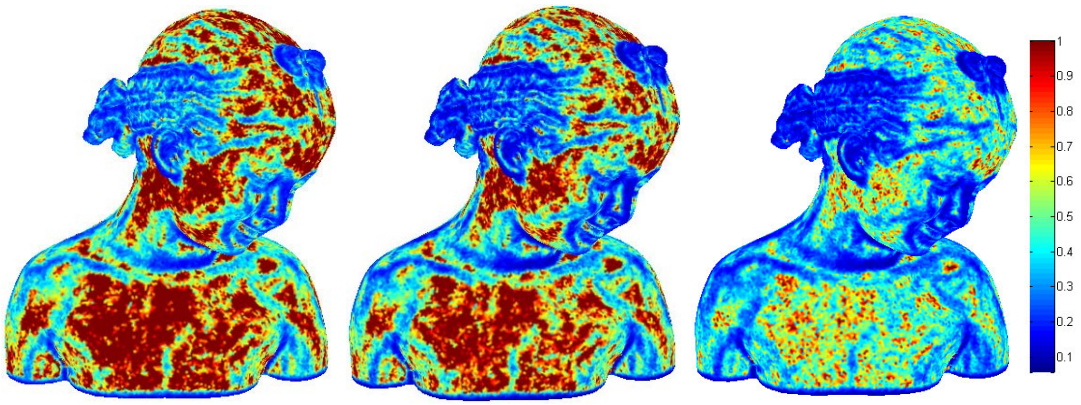

Figure 4.3: Three different computations of the value of R. The first follows [49]. The second and the third use the mean and the maximum values of γ computed over all bootstrap samples, respectively.

smooth and the source of noise is only from the added noise. The average error estimations are summarized in Table 4.2. We notice that the training error is again smaller than the test error, while the .632+ error, being in between the training error and the leave-one-out error, is expected to be more accurate. The no-information error rate was computed as an average over all bootstrap samples.

Noise	$\bar{err}$	$\hat{Err}^{(1)}$	.632	.632+
0.00	0.08	0.11	0.10	0.10
0.25	0.25	0.35	0.31	0.33
0.50	0.45	0.62	0.56	0.61
0.75	0.66	0.89	0.80	0.88
1.00	0.85	1.15	1.04	1.15
2.00	1.21	1.66	1.50	1.66

Table 4.2: Cubic polynomial fitting on the Bimba model on neighborhoods of size $N = 50$.

As expected with a natural model, the test error estimated on the model without any added noise can be significant. Fig. 4.4 shows the original Bimba model and the .632+ test error estimations on it. We notice that the high test error

values concentrate on the sharp features of the model, indicating the inability of polynomials of low degree to capture such features.

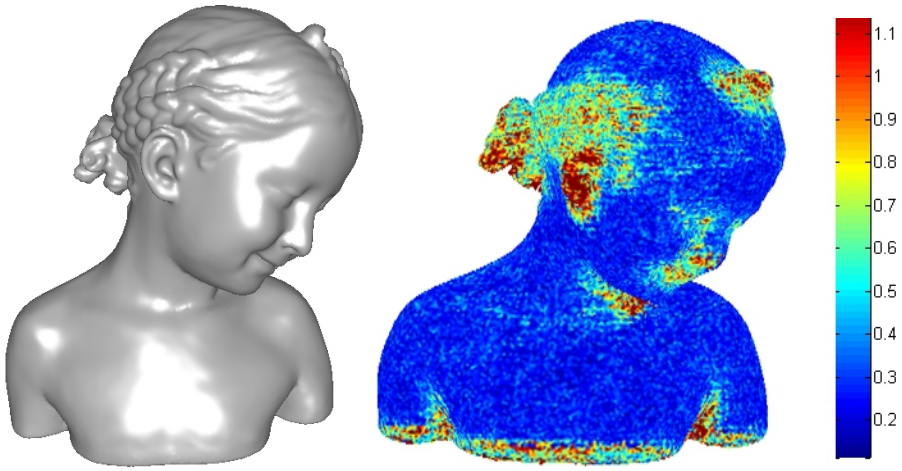

Figure 4.4: The Bimba model and the .632+ error estimations colormap.

To validate the accuracy of the method on models with varying amounts of noise, following [109], we tested the method on the Bimba model with continuously varying noise from 0.0 to 1.0. The results are shown in Fig. 4.5. The colormap of the .632+ error indicates that it can adapt nicely to a varying amount of noise. This is better observed in the smooth parts of the model where the added noise is the main source of test error. We also see that the model features are again captured by the .632+ estimator, this time as a second source of test error.

To validate the general ability of the bootstrap methods to cope with natural and CAD objects, we tested them on the Bunny and the Fandisk models. In the Bunny, we notice that the error concentrates in the ear tips, which are notoriously difficult to model. In the Fandisk, the error concentrates on the sharp features as well as on the thin part of the model where two different surface sheets come close to each other.

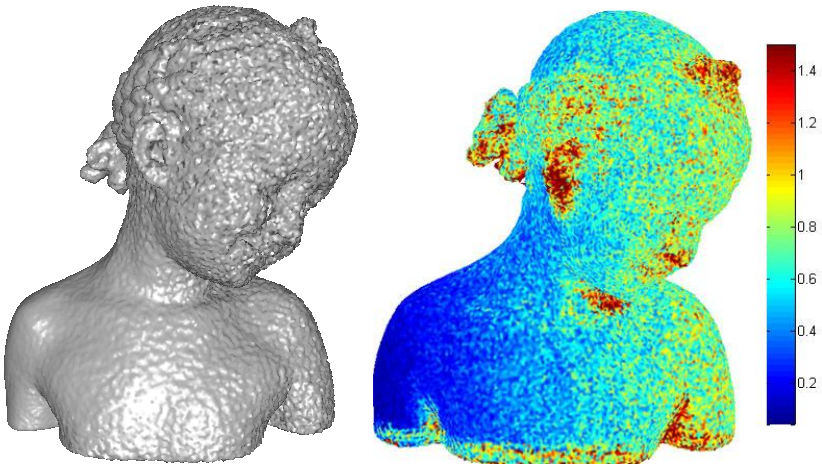

Figure 4.5: The Bimba model with continuously varying added noise from 0.0 to 1.0 and the colormap of the .632+ test error estimations.

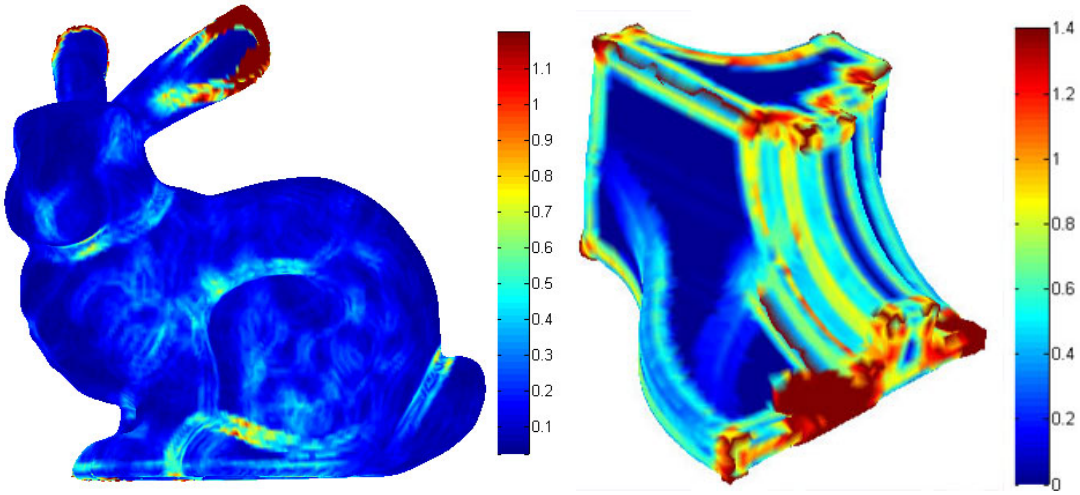

Figure 4.6: .632+ error estimations colormaps.

4.2.3 Tests on scanned models

We run the bootstrap procedure on two scanned models to see the effect of our algorithm in real life data. In this experiment, we do not add any noise as in previous cases, and the source of noise is merely from the scanning procedure. Thus in this case, there is no independent way to validate the estimation as in previous case. The models chosen are Fertility model with 241607 points and Ramses with 826266 points as shown in Figure 4.7. The model (yellow) is displayed next to its error estimation (grayscale). The darker area signify the higher error estimated at that particular area. We display the errors in Table 4.3.

Noise	K	$\bar{err}$	$\hat{Err}^{(1)}$	.632	.632+
Fertility	100	0.1281	0.1489	0.1412	0.1420
Ramses	50	0.1764	0.2470	0.2210	0.2343

Table 4.3: Cubic polynomial fitting on the Fertility model on neighborhoods of size $N = 100$.

Visually, our bootstrap error is faithful in showing high error on feature and noisy areas. As we can see from the Fertility model, there is a long visible line on the right hand-side of the model (first and third figure in top row). In its error estimation counterpart, the colourmapping display a long dark line which indicates high value of error (grayscale picture, second and fourth figure in top row). In this case, the high error value is caused by the feature area. We can also see that the black lump on its side as shown on the grayscale picture is caused by noisy area as seen on the model.

The Ramses model has average higher noise in comparison to Fertility model. However we can see that the error detected nicely around edges as well as the

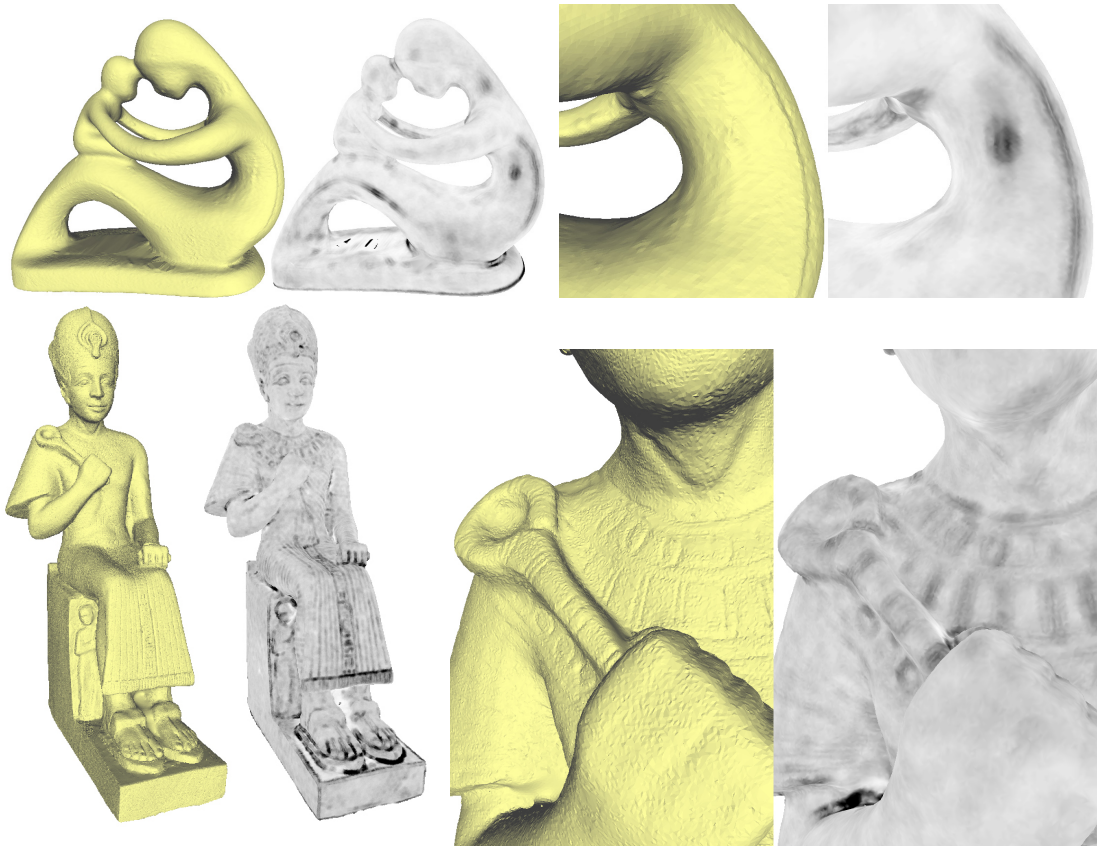

Figure 4.7: The model and colourmap of Fertility (top row) and Ramses (bottom row). The model is shown in the first and the third column. Second and fourth column show the colourmapping of the .632 error. The error is capped at 1 error level for visualisation purposes. Fertility Model is provided courtesy of Frank ter Haar by the AIM@SHAPE Shape Repository. Ramses model is provided courtesy of the AIM@SHAPE Shape Repository.

decorative feature under its neck area.

4.3 Summary

We proposed the use of bootstrap based techniques for test error estimation in the context of surface reconstruction. Validation experiments on synthetic and natural smooth models with local polynomial surface fitting indicate that the .632 and .632+ error estimates are the most reliable, as their values lie between the training error, which underestimates the test error, and the leave-one-out, which overestimates it.

The experiments also show that some commonly used settings of polynomial fitting, such as quartic polynomials on neighborhoods of size 50, exhibit a small but quantifiable amount of overfitting. This shows that overfitting detection not only has academic interest, but also may provide assistance in making choice of a type of fitting in a set of neighbourhood.

To the best of our knowledge, this is the first method computing test error estimates in the context of surface reconstruction from large noisy point sets. However, assuming that the main source of test error is data noise, our work has similarities with the Bayesian data noise estimator proposed in [109]. There is a relation between the two papers due to the fact that the distribution of the values of the bootstrap models $f^{*b}(\mathbf{x}_i)$ approximates the posterior of the model under non-informative priors [49]. That is, the creation of the bootstrap models can be seen as a direct, naive approach to the estimation of the posterior distribution of the model. As bootstrap is much simpler than the direct Bayesian approach, it can be easily extended to more complex geometric modelling settings.

Chapter 5

Normal estimates on point clouds

5.1 Introduction

Several of the state-of-the-art surface reconstruction algorithms require accurate estimations of normal vectors associated with the points of the set. The process of estimating such normals is known as *normal reconstruction*.

In this chapter we propose a method for normal reconstruction based on *bootstrap*. In our context of normal reconstruction, the initial data set is the K nearest neighborhood of a point d_i of the point set. The model fitted in each bootstrap sample is an estimated tangent plane computed with Principal Component Analysis (PCA). From each tangent plane we compute a normal vector; the average of these normals is our normal estimate at d_i. The variance of the angles of the bootstrap normals is also computed and treated as a confidence value for the normal at d_i. Later, we will also introduce error estimation method for a normal estimate.

Our experiments show that the bootstrap normals are comparable with the

commonly used PCA normals, and thus their main advantage lies in the confidence values associated with them. As an application, we construct a bilateral Gaussian filter for normal smoothing. In this filter, the influence of d_i on d_j depends not only on the distance between d_i and d_j, but on the normal bootstrap variance at d_i as well.

In summary, the main contributions of this chapter are:

- A bootstrap-based method for normal reconstruction which produces normals with confidence values based on variance.
- A bilateral Gaussian filter for normal smoothing utilising the confidence values of the bootstrap normals.
- An error estimation method for normal estimates.

In this chapter, we will apply bootstrapping to a normal estimation. Given a point set, the normal of each point is computed from Principal Component Analysis. We apply bootstrapping to a normal estimation to estimate the reliability of a calculation. The background of bootstrap has been discussed in Chapter 4, and we saw that the bootstrapping procedure generates samples which may be used to estimate error, variance, probability density and other statistical values.

There are two statistical values which will be presented. Firstly, we will estimate the variance of normal estimates. We will study the variance of the angle of the normals generated from the bootstrap samples. The value itself signifies the reliability of our estimation based on this property: If the average angle of the samples is small, it signifies that the normal is reliable. Comparing variance value for normal estimates across point sets can give an indication of reliability and quality of the estimate.

Secondly, we will implement a test error estimation method based on the bootstrap procedure. We will show that an appropriate adjustment is needed, in contrast to what had been implemented in Chapter 4.

Section 5.2 - 5.3.2 have been presented in

- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Bootstrap-based normal reconstruction", *In Curves and Surfaces*, pages 575-585, 2010.

5.2 Bootstrap normal reconstruction for PCA

Following the definitions given in Chapter 4, we will define bootstrap variance. If $f(\mathbf{P})$ is any quantity computed from the data $\mathbf{P}$, we can use the bootstrap samples in Eq 4.2 to estimate any aspect of the distribution of $f(\mathbf{P})$. In particular we can estimate its mean by

$$\bar{f}(\mathbf{P}) = \frac{1}{B}\sum_{b=1}^{B} f(S_b) \tag{5.1}$$

and its variance by

$$\widehat{Var}[f(\mathbf{P})] = \frac{1}{B-1}\sum_{b=1}^{B}(f(S_b) - \bar{f}(\mathcal{D}))^2 \tag{5.2}$$

Given a sufficiently large number of bootstrap samples B, the bootstrap statistics in Eq. 5.1 and Eq. 5.2 are good approximations of the true mean and variance of $f(\mathbf{P})$, because the distribution of the $f(S_b)$ follows the distribution of $f(\mathbf{P})$ under non-informative priors (See [49]).

5.2.1 Algorithm

Applying the bootstrap method to the problem of normal reconstruction, let the input of the algorithm be an unorganised 3D point set $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$. For a given point $d_i \in \mathcal{D}$ we find its K-neighbourhood

$$P_i = \{p_1, p_2, \ldots, p_K\}, \quad p_i \in \mathcal{D}, \quad i = 1, 2, \ldots, K \tag{5.3}$$

Following the bootstrap sampling procedure, we generate a set of bootstrap samples $\{S_1, S_2, \ldots, S_B\}$ consisting of B subsets of P_i. On each of the S_i we apply PCA and compute a tangent plane. For a consistent orientation of the normals of these planes, we first apply PCA on the whole dataset $\mathbf{P}$ and, by choosing an arbitrary orientation, we compute a normal $\mathbf{n_P}$. Then, we obtain the bootstrap normals $\{\mathbf{n}_1, \mathbf{n}_2, \ldots, \mathbf{n}_B\}$ by choosing for $\mathbf{n}_i$ the orientation that minimises the angle between $\mathbf{n_P}$ and $\mathbf{n}_i$. This will consistently orientate the normals from bootstrap samples. Note that this does not globally orientate normals for all of the points in the model.

Our final estimate for the normal at d_i is the average of the bootstrap normals computed on P_i

$$\mathbf{n}(d_i) = (\sum_{b=1}^{B} \mathbf{n}_b)/|\sum_{b=1}^{B} \mathbf{n}_b| \tag{5.4}$$

This statistic can be seen as the result of Eq. 5.1, where the function f defined on the subsets of P_i returns the PCA normal of a subset.

The second bootstrap statistic we compute for d_i is the variance of the angles between the bootstrap normals and the normal of the first bootstrap sample.

That is, we compute the variance

$$\nu_i = \widehat{Var}\{< \mathbf{n}_1, \mathbf{n}_i > | i = 1, \ldots, B\} \tag{5.5}$$

whereas $< \mathbf{n}_1, \mathbf{n}_i >$ is the angle between $\mathbf{n}_1$ and $\mathbf{n}_i$. This statistic can be seen as the result of Eq. 5.2, where the function f defined on the subsets of P_i returns the angle between the PCA normal of a subset and $\mathbf{n}_1$.

This second statistic will be used as a confidence value for our normal estimate $\mathbf{n}(d_i)$. Notice that other bootstrap statistics could also be treated as confidence values for the normal estimates. In particular, one could treat the PCA tangent planes as the fitted models, compute distances between points in P_i and these tangent planes, and use them to compute one of the several bootstrap error estimates proposed in the literature (See [49]). However, we think that processing angles between normal vectors, rather than distances between data points and tangent planes, is a more direct approach and thus more likely to give reliable confidence values.

5.2.2 Normal reconstruction

We tested the proposed normal reconstruction method on synthetic models. Starting from smooth triangle meshes, we used their connectivity to compute reliable normals at the vertices. These normals were used as the ground truth against which we measured the error of the estimated normals.

To reiterate from chapter 4, a noise level equal to ξ refers to displacements of magnitude uniformly randomly sampled from the interval $[0, \xi h]$, where h is the average throughout the model of the distance between a point and its nearest

neighbour.

Table 5.1 summarises the results of the bootstrap normal reconstruction. The Sphere and the Bunny models were tested at noise levels of 0.25, 0.5, 1.0 and 1.5, and with neighbourhood sizes of 15 and 30. The reported numbers are the average angle differences (in radians) with the ground truth normals. We highlighted the comparison with blue text for one that has a smaller angle than the other. We can see that it is not definitive to say that the PCA normal is better than the bootstrap normal. Hence, we conclude that the bootstrap normal reconstruction and the PCA normal reconstruction produce comparable results. However, the bootstrap normal reconstruction also provides confidence values which are not available with PCA normal. We also note that the bootstrap method is computationally more intensive, taking 529 seconds to process the 11146 vertices of the Bunny model on a low-end commodity PC.

Sphere	k=15		k=30	
Noise	PCA	Bootst.	PCA	Bootst.
0.25	0.0546	0.0547	0.0296	0.0297
0.50	0.1141	0.1151	0.0586	0.0591
1.00	0.2818	0.2847	0.1306	0.1311
1.50	0.4881	0.4949	0.2601	0.2599
Bunny	k=15		k=30	
Noise	PCA	Bootst.	PCA	Bootst.
0.25	0.1388	0.1397	0.1732	0.1739
0.50	0.1754	0.1748	0.1880	0.1882
1.00	0.3175	0.3152	0.2471	0.2463
1.50	0.4660	0.4670	0.3278	0.3264

Table 5.1: Average angle difference between the ground truth normals and the estimated normals in radians. The number of bootstrap samples is always $B = 50$. A lower value is highlighted in blue colour text.

5.2.3 Bootstrap variance

In a second experiment, we examined the claim that normals with higher bootstrap variance are generally less accurate. Working on the Bunny and Fandisk models at various noise levels, we split the set of vertices into a high variance subset, containing the vertices with the highest 15% variances, and a low variance subset, containing the rest of the vertices. We computed and reported the average normal error of the two sets separately. The neighbourhood size K is fixed to 50 and the results are summarised in Table 5.2.

Bunny	0.25	0.50	1.00	1.50
Low Var.	0.1707	0.1768	0.2082	0.2489
High Var.	0.4731	0.4920	0.5168	0.5927
Fandisk	0.25	0.50	1.00	1.50
Low Var.	0.0590	0.0745	0.1146	0.1818
High Var.	0.5813	0.5746	0.5850	0.5911

Table 5.2: The average normal error on the high and low variance subsets of the Bunny and the Fandisk models.

We notice that, as expected, the high variance set has normal estimates with significantly higher average error. Moreover, this is the case at all noise levels, meaning that the method can handle error from both sources, that is, the model features and the added noise. We note that the method works for fixed neighbourhood size, which means that to compare the confidence level of normal estimation on one point to the other, we have to use a fixed K to compute normals on $\mathcal{D}$. This is because higher or lower values of K would respectively naturally decrease or increase the variance, without necessarily changing the accuracy of the normal estimates.

Fig. 5.1 shows colourmaps of the bootstrap variance for the Bunny and Fan-

disk models at various levels of noise. We notice that the high variance areas concentrated on the features of the mesh. We also notice that this pattern degrades slowly as the added noise increases.

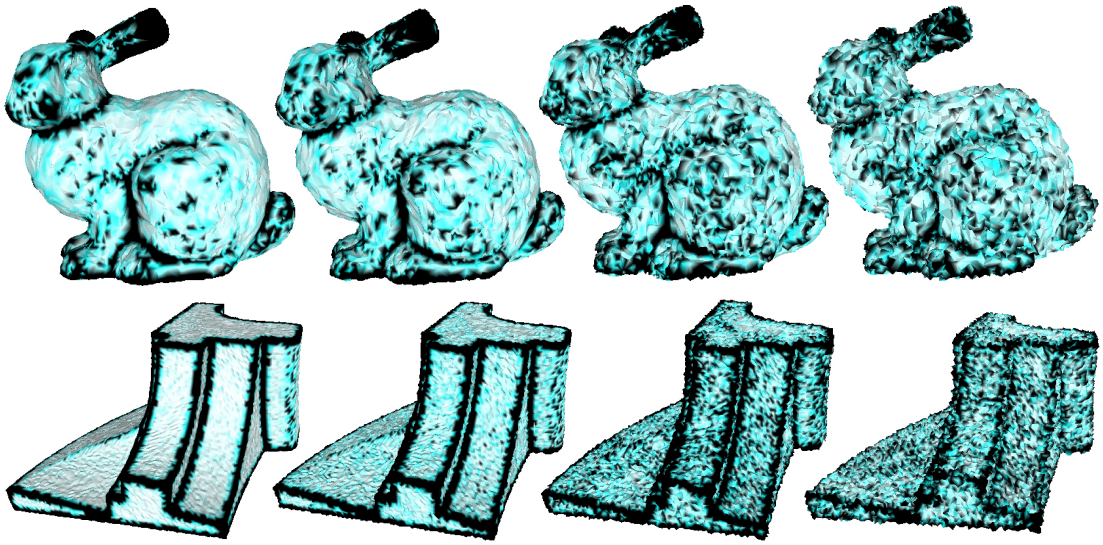

Figure 5.1: From left to right: Colourmaps of the bootstrap variance at noise levels 0.25, 0.5, 1.0 and 1.5, respectively. The darker colours signify higher variance.

5.3 Bilateral Gaussian filter for normal smoothing

In previous section we showed that normal estimates with lower bootstrap variance are generally more reliable than those with higher variance. In this section, we will use the bootstrap variance as a confidence value and propose a bilateral Gaussian filter for normal smoothing. In each filtering iteration, normal at a point d_i is updated as a distance weighted average of the normals at the neighborhood of d_i, while the confidence values are used to reduce the influence of the

less reliable normals.

5.3.1 Bilateral Gaussian filter

One iteration of the proposed filter updates the normal $\mathbf{n}(d_i)$ at a data point $d_i \in \mathcal{D}$ by

$$\mathbf{n}(d_i) \rightarrow (1-\alpha)\mathbf{n}(d_i) + \frac{\alpha}{c(d_i)} \sum_{p \in N_i} \mathcal{G}(\nu_j, \sigma_\nu)\mathcal{G}(||d_i - p||, \sigma_d)\mathbf{n}(p) \tag{5.6}$$

In Eq. 5.6, $\mathcal{G}$ is the Gaussian function with zero mean and standard deviation b given by

$$\mathcal{G}(a, b) = e^{-a^2/b^2}, \tag{5.7}$$

$||\cdot||$ denotes Euclidean distance, and $c(d_i)$ is a normalisation factor that makes the sum of the weights equal to 1, that is

$$c(d_i) = \sum_{p \in N_i} \mathcal{G}(\nu_j, \sigma_\nu)\mathcal{G}(||d_i - p||, \sigma_d) \tag{5.8}$$

The variances σ_ν and σ_d of the two Gaussians in Eq. 5.6 are user defined parameters. A smaller value of σ_ν would mean normals with high variance, that is, less reliable normals, which will have less influence on the smoothing process. A smaller value of σ_d would mean that the influence of a point decays more rapidly with distance. In our experiments, we used a σ_d equal to h which is the average throughout the model of the distance between a point and its nearest neighbour. Finally, α is a user defined parameter controlling the strength of the filtering operation.

5.3.2 Evaluation

We tested the algorithm on the Cube and the Fandisk point sets with 0.5 level of added noise, $\sigma_\nu = 10^{-12}$ and 1.1×10^{-12}, respectively, and $\alpha = 0.1$. As in Section 3, we used the normals obtained from the underlying meshes before the addition of noise as the ground truth.

The results and a comparison with the single Gaussian filter corresponding to $\sigma_\nu = \infty$ are shown in Fig. 5.2. The computed normal errors are shown separately for the feature areas where $\nu_i \geq 0.04$ (left), and the non-feature areas (right). In all cases, the proposed bilateral filter outperforms the single Gaussian filter. We also notice that in the non-feature areas the normal error increases from the very first iteration of the single Gaussian filter. The reason is that the inaccurate normal estimates in the feature areas corrupt the more reliable estimates in the smoother areas. This is a serious limitation of the single Gaussian filter, which is overcome by the proposed bilateral filter.

Fig. 5.3 shows the normals of the Cube model after applying the bilateral filter and after applying the simple Gaussian filter. We conclude that the superiority of the bilateral filter, as demonstrated by the graphs in Fig. 5.2 (top), is visually significant.

Notice that the above validation of the filtering algorithm is mainly concerned with the accuracy of the bootstrap normals compared to a ground truth; here, the normals were obtained from a smooth triangle mesh whose vertices were used in the construction of the test data sets. One of the primary uses of point sets is the fast, high quality rendering of 3D models. We do a quantitative comparison, while in the literature, qualitative comparisons based on the quality of rendering

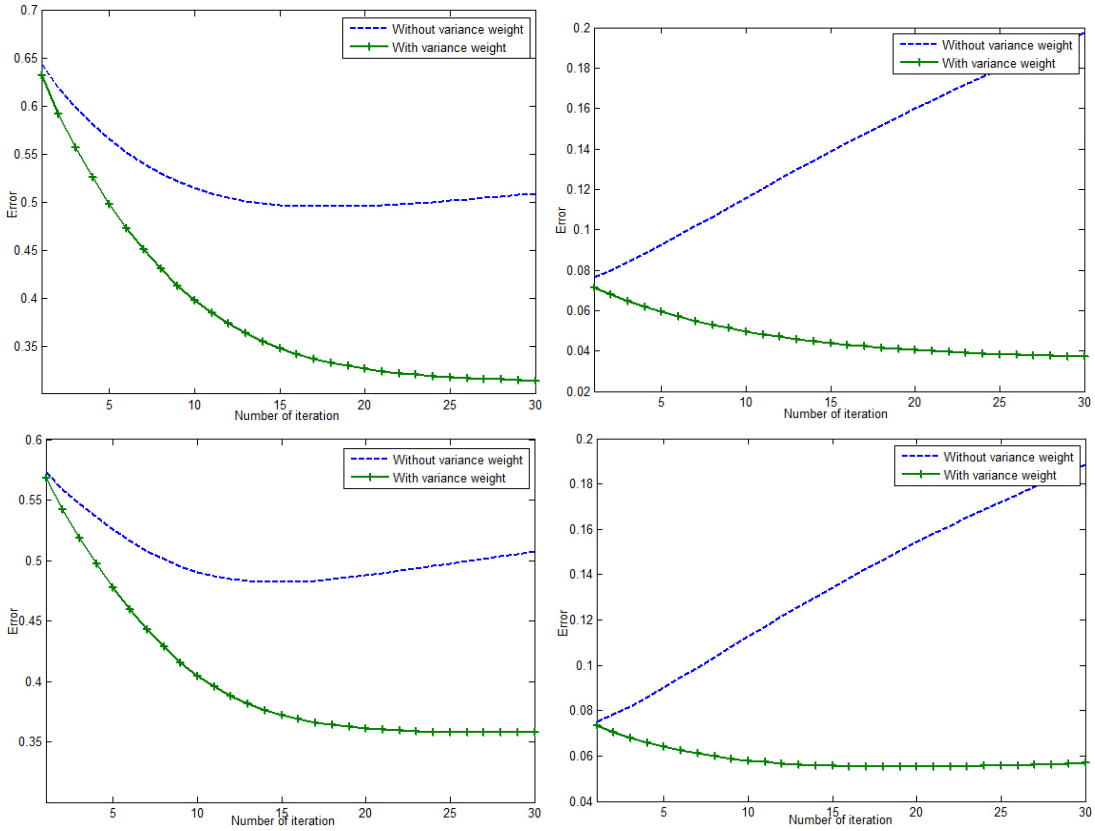

Figure 5.2: Comparison between the single Gaussian filter (blue dotted line) and the proposed bilateral filter (green crossed line). **Top:** Cube with 0.5 noise level. **Bottom:** Fandisk with 0.5 noise level. **Left:** Feature areas. **Right:** Non-feature areas.

are most common (See for example [58]).

5.4 Normal estimation with higher order

5.4.1 Introduction

In previous sections, we estimated normals using Principal Component Analysis, which is equivalent to a least square fitting of a planar surface. Normal computation from the gradient value obtained from a non-planar surface had been done

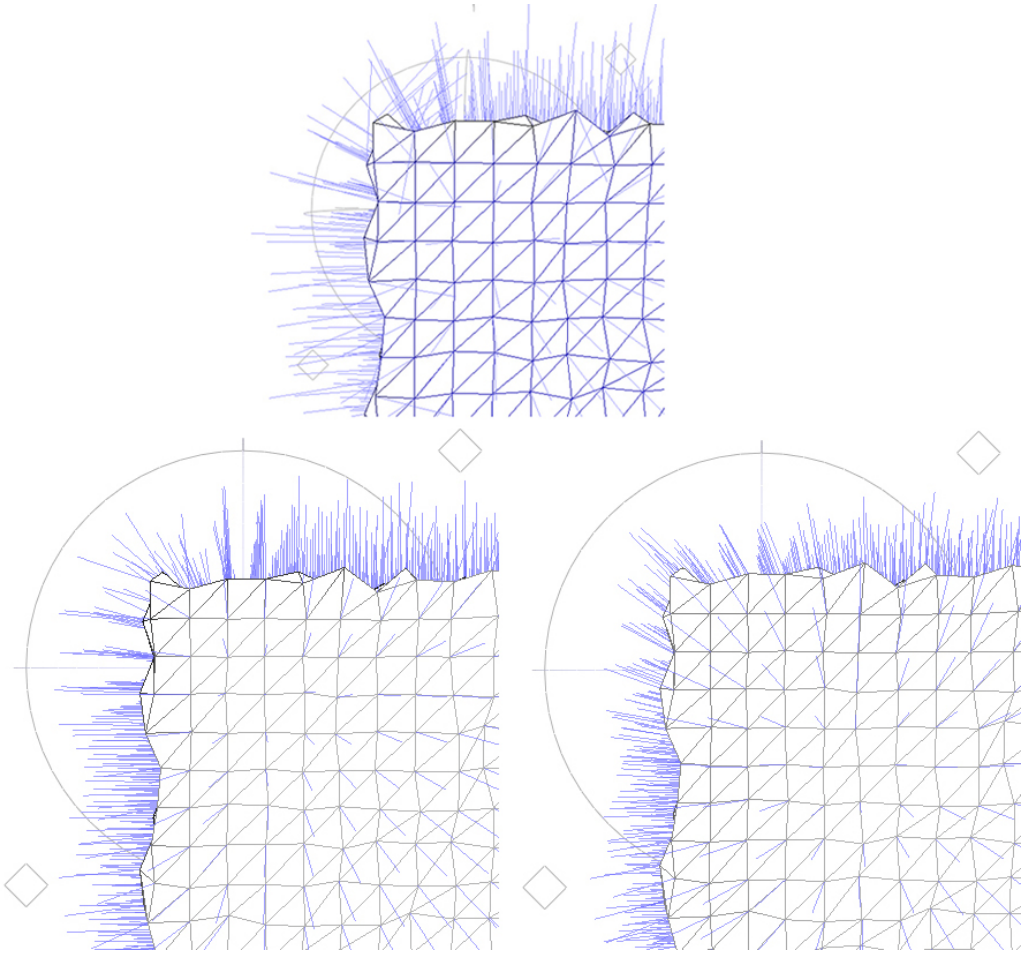

Figure 5.3: **Top**: Normals before smoothing is applied. **Bottom Left**: The proposed bilateral filter. **Bottom Right**: Simple Gaussian filter. In all figures, the angle of view is centered at the circle.

such as in [88].

In this section we define normal computation from a polynomial fitting and apply bootstrap methods on a quadratic surface fitting. We intend to see if the bootstrap method can supersede a single fitting as well as if higher degree polynomial can outperform PCA normal.

While we define a general formula for any given polynomial degree, we limit the discussion to linear (PCA normal) and quadratic in order to see the comparison more easily. A higher degree polynomial, such as a cubic, needs more points, and we then have to deal with other issues such as overfitting.

5.4.2 Algorithm

We repeat the bootstrap method on a normal construction, using polynomial surface fitting instead of PCA normal. For a neighbourhood of $P_i = \{p_1, p_2, p_3, ..., p_K\}$ we will fit a polynomial surface of degree M.

Again we firstly localize $(x, y, z) \in \mathbb{R}^3$ to parameterise a 3-dimension to (x', y', z') in 2.5-dimension to allow a fitting of a height function $z' = f(x', y')$ on a plane (x', y') from point sets $(x, y, z) \in P_i$ (The function for polynomial fitting surface is available in Appendix A.

Given that a parametric surface $\vec{r} = (x', y', f(x', y'))$, a normal at point (x', y') is given by

$$\mathbf{n} = \frac{\vec{r_x} \times \vec{r_y}}{|\vec{r_x} \times \vec{r_y}|} \tag{5.9}$$

whereas $\vec{r_t}$ is a partial derivative $\partial f / \partial t$, $\times$ is a cross vector product and $|\,.\,|$ is a norm for the vector.

As the points p_i's had been localized to 2.5-dimensional, the normal that we obtained has to be transformed back to 3-dimensional. Then, the bootstrapping procedure follows similar steps to those defined in Section 5.2.

Algorithm 2 Algorithm for bootstrap normal reconstruction using higher degree polynomial

Algorithm bootstrap_normal_higher()
n = 0;
for `i=1:N`

Estimate normal in a single fitting.
for `j=1:B`

Create bootstrap samples
Fit a polynomial surface
Compute normal $\mathbf{n}_j$(by (5.9))
Make orientation of normal consistent with $\mathbf{n}_1$
Compute the angle difference between $\mathbf{n}_j$ and $\mathbf{n}_1$

end for
Compute variance of angle difference
Compute mean of normal

end for

5.4.3 Result and Discussion

We had run this method on the previous model that we used, starting with a sphere. Similar to the previous section, the normal value produced via bootstrapping does not improve the one that we compute without bootstrapping (which we will call *single fitting*). This result is shown on Table 5.3. Table 5.4 shows the comparison between bootstrapping on a PCA normal and quadratic fitting for a

sphere.

	K=15		K=30	
Noise	Single fit	Bootstrap based	Single fit	Bootstrap based
0.25	0.0449	0.0483	0.0228	0.0232
0.50	0.0901	0.0958	0.0445	0.0450
1.00	0.2138	0.2248	0.0932	0.0936
1.50	0.3713	0.3956	0.1791	0.1825
	K=15		K=30	
Noise	Single fit	Bootstrap based	Single fit	Bootstrap based
0.25	0.1063	0.1022	0.1421	0.1390
0.50	0.1329	0.1340	0.1485	0.1459
1.00	0.2421	0.2525	0.1836	0.1841
1.50	0.3758	0.3947	0.2436	0.2463
	K=15		K=30	
Noise	Single fit	Bootstrap based	Single fit	Bootstrap based
0.25	0.1008	0.1009	0.1118	0.1117
0.50	0.1356	0.1390	0.1262	0.1266
1.00	0.2342	0.2437	0.1628	0.1642
1.50	0.3940	0.4169	0.2373	0.2402

Table 5.3: Angle difference (in radian) of ground truth normal and estimated normals (single fit on quadratic based or bootstrap aggregation) of sphere (top), bunny with 11146 points (middle) and fandisk with 24664 points (bottom).

As in the previous subsection, we display the angle difference of the normal estimates with the ground-truth normal. Thus, a lower value signifies a better estimation. In every case except 1.50 noise level, a quadratic fitting would produce a better estimate of normals than PCA-normal. When we increase the size of the neighbourhood from 15 to 30, we could see that the estimation of normal improves as well for both cases.

We could simply claim that the normal obtained from quadratic fitting estimates better than PCA normal on a sphere. This is simply because higher order polynomial such as quadratic fitting imitates the a shape of sphere in a local

neighbourhood, thus giving a better normal straightaway.

Having said that, when we consider a neighbourhood of points from a natural model, the neighbourhood tends to be curvy rather than planar. This is especially true for models Bunny and Bimba.

	K=15		K=30	
Noise	PCA Bootst.	M2 Bootst.	PCA Bootst.	M2 Bootst.
0.25	0.0542	0.0483	0.0295	0.0232
0.50	0.1146	0.0958	0.0592	0.0450
1.00	0.2487	0.2248	0.1292	0.0936
1.50	0.3947	0.3956	0.2280	0.1825

Table 5.4: Angle difference (in radian) of ground truth normal and the estimated normals (PCA based or quadratic using bootstrap aggregation) of sphere. We use a K nearest points for our neighbourhood and number of bootstrap samples B is 50.

	PCA		M2	
Noise	Single fit	Bootstrap	Single fit	Bootstrap
0.25	0.2142	0.2152	0.1839	0.1827
0.50	0.2216	0.2228	0.1871	0.1861
1.00	0.2526	0.2537	0.2038	0.2036
1.50	0.2959	0.2982	0.2313	0.2318
	PCA		M2	
Noise	Single fit	Bootstrap	Single fit	Bootstrap
0.25	0.0915	0.0924	0.0978	0.0970
0.50	0.0974	0.0985	0.1009	0.1002
1.00	0.1185	0.1196	0.1124	0.1121
1.50	0.1536	0.1555	0.1341	0.1343

Table 5.5: Angle difference (in radian) of ground truth normal and the estimated normals (PCA based or bootstrap aggregation) of bunny with 11146 points (top) and 34835 points(bottom). We use a K=50 nearest points for our neighbourhood and number of bootstrap samples B is 50.

We furthered our test by comparing PCA and bootstrap method on Bunny as displayed in Table 5.5. We used two different Bunny models with 11146 and

38435 points using $K = 50$ and $B = 50$. First is the comparison between single fitting and bootstrap. In all cases, both results are comparable, as bootstrapping may give a better approximate in one case and worse in the other case. However, we can clearly see, comparing between PCA normal and quadratic fitting, the latter give a better estimate than the former.

In this case, bootstrapping does not play a significant role, as one can simply use a single fitting which is most cost-effective in this case. However, if a quality of estimation is in question, the bootstrap method may offer one as shown in the earlier section.

5.5 Bootstrap error estimation on normals

In this section, we will address limitations to using variance as our statistic and present bootstrap error estimates on normals.

5.5.1 Variance effect on different neighbourhoods

In practice, the chosen value of K nearest neighbourhood to compute normals is user-defined. One may choose a neighbourhood appropriately depending on total number of points and the sparseness of the distribution. Usually it should not be too small, such that noise may affect the computation, or too large, such that features may become a factor.

It is one of our interests to provide an automatic selection of neighbourhoods such that it will give us the best estimates of normal and optimum values to be used in other post-processing step. However, we have to analyse whether variance information could be relied on in such a matter.

Note that in comparing the variance value of one point to the other, a lower variance signifies a better estimates. We should note, however, that the value of K is fixed, thus the comparison is possible. We will see now whether the variance value can signify the quality if the size of the neighbourhood is a variable we want to compare.

Table 5.6 shows us the angle difference between ground truth normals and estimated PCA normals (which we labelled as *ground truth error*) for different neighbourhoods used in computing normal on Bunny with 0.5 noise level. Although the variance of a neighbourhood of 100 is lower than a neighbourhood of 50, the ground truth error shows the opposite. As we increase the size of the neighbourhood, the variance of an area will reduce. Thus, for a different choice of neighbourhood, one cannot generalise that a low variance value will imply a low error value. Thus, one should not use a variance value in determining a good selection of K.

Estimating a good K cannot not be done from variance due to a phenomenon known as bias and variance decomposition, which had been addressed in Section 3.2. Error can be decomposed to a component of bias and variance as follows, $Error^2 = Bias^2 + Variance$. While increasing the number for K, the complexity decreases and therefore the variance will become lower (as showed in Table 5.6).

Up until this section, we have been comparing the quality of the model with a fixed K thus intentionally ignoring the information of bias as the complexity remains the same. This would be different when comparing different K as complexity varies.

We show another example in Figure 5.4. Mean variance is plotted for different

Number of Neighbourhood, K	30	100
Mean Variance	0.0205	0.0141
ground truth error	0.1880	0.2866

Table 5.6: Angle difference (in radian) of real normal and estimated normals for different neighbourhood in bunny with 0.5 noise level. Mean variance of the normal estimation is also displayed.

neighbourhood sizes for a cube model in the region highlighted in the figure. The neighbourhood starts to include the feature area once the neighbourhood is more than 50 points (which is represented by blue and black coloured regions).

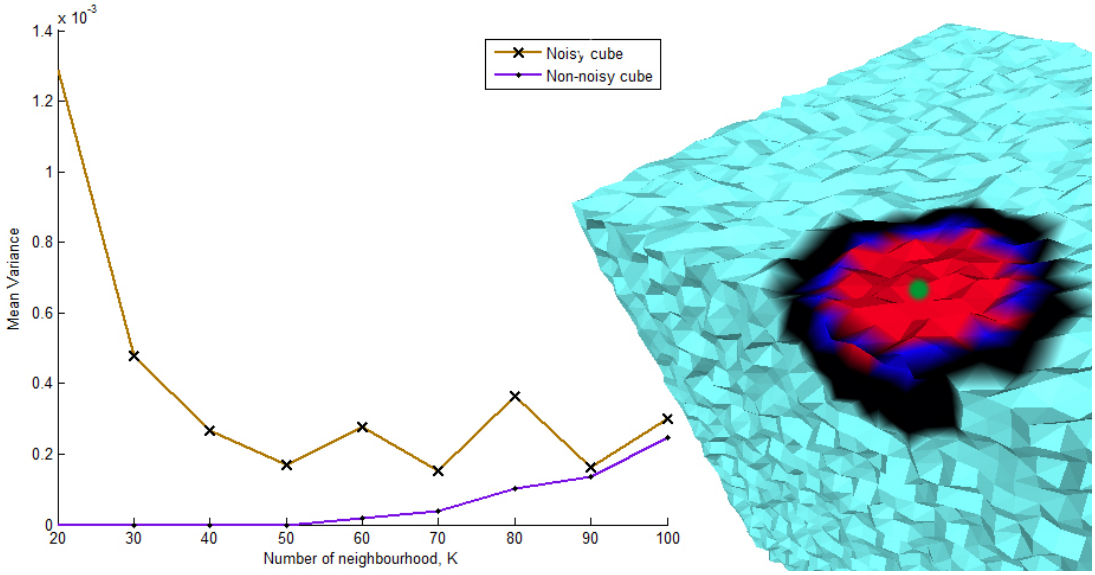

Figure 5.4: Mean variance for different sizes of neighbourhood. The point is centered at the green dot, the 50 nearest neighbourhood is coloured red, the 70 nearest neighbourhood expands to the blue region and 150 nearest neighbourhood expands to the black region. The model used is a cube with 6146 points and 0.5 noise level.

As the neighbourhood size increases, variance decreases. The effect can be seen when we increase the neighbourhood from 20 to 50. The staggered pattern after that is caused by feature area which is verified by the purple line when we

tested it on a non-noisy cube.

The result is an example of bias-variance decomposition. Complexity is affected by the chosen number of neighbourhoods, thus making it impossible for us to rely on the quality of normal estimation based on variance value. Similarly, if we use a different polynomial degree, it will also alter the complexity of the estimation on a model. So, when we compute variance as our measure of confidence value on a model, a comparison of quality cannot be made when a variable presents that alters the complexity of the computation.

5.5.2 Conceptualisation

Here, we propose using error estimation on normal using the bootstrap method. The bootstrap method has a straightforward definition concerning the estimation of statistical values, but we have to define and conceptualise it accordingly to our case.

We had implemented error estimation in the model fitting in Chapter 4. As a discussion, we revisited our definition of training error. We had defined training error as the average distance between the input training data and the predictions of the model. So, if a prediction results in a point that is further away from the sample, the training error will be higher.

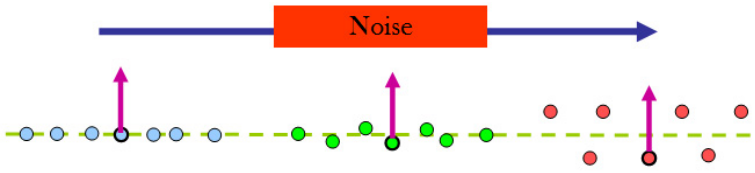

Figure 5.5: Different noise level and normals.

For normal estimation, this would not necessarily be the case.Although in practice noise does affect normal estimation, we may still estimate a good normal even though the noise level is higher, as illustrated in Figure 5.5. Thus, to define error in the context of normal estimates, we cannot simply used the error defined in Chapter 4.

We had defined our ground truth error as the angle difference between the normal computed from the samples and the normal from a smooth model. As we have seen that this definition has been a valid way to represent a measure of quality in our normal estimation, we will further define *training error for normal estimates* as the average angle difference between the normal estimates of input training data and normal estimates of the predictions of the model. This definition will be used when we apply the bootstrap methodology to find a test error.

5.5.3 Algorithm

As our data is generated by adding some noise from a smooth model, we may have certain normal values beforehand which may be used for the purpose of verification. This obviously would not be the case in a real-life problem, where we are required to estimate the normals ourselves.

In this case, if we could have an error estimate which provides us with something similar to what we had defined as the ground truth error, it would be a very beneficial tool to check the quality of estimates. We will see that the bootstrap method can be a tool to solve this issue.

We will apply bootstrapping for a normal estimation by starting with PCA

normals before extending our case into higher degree polynomials.

The bootstrap error estimation Err_{boot} is defined in Equation 4.3. In this case, our $\mathbf{Z}^{*b}$ remains the same as in Chapter 4, which is a K-nearest neighbourhood of point at d_i. However, the part of the equation $|f^{*b}(\mathbf{x}_i) - y_i|$ has to be defined accordingly to what we want to achieve. As we have shown in previous section that training error uses angle difference as parameters, we will use this same value for this method as well.

While $|f^{*b}(\mathbf{x}_i) - y_i|$ is the distance between predictors and their sample points, we redefine this whole term as the angle difference between two vectors. To differentiate it in the context of normal estimates, we will note this as $\theta(\mathbf{w}_1, \mathbf{w}_2)$ as the angle difference vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. Thus, for a PCA normal estimate $\mathbf{g^b}(x_i)$ on bootstrap samples in neighbourhood of point x_i, the Err_{boot} is given by

$$Err_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{K}\theta(\mathbf{g^b}(x_i), \mathbf{n}_i). \tag{5.10}$$

Given a point d_i with K-neighbourhood $Z = \{p_1, p_2, ..., p_3\}$, bootstrapping procedure works as described in Algorithm 3. Technically if $\mathbf{n}_i$, normal at point x_i for the model is made available in the acquisition process, we can use the given value in our estimation. As the purpose of this mechanism is to estimate error without any initial information, however, we would estimate $\mathbf{n}_i$ using PCA normal on a certain neighbourhood for point x_i. Note that this neighbourhood may be different from K and may include points which are not included in Z. This statement have a greater impact once we are comparing the error between different neighbourhoods, and in this case $\mathbf{n}_i$'s should be constant and not recalculated using different K's.

Algorithm 3 Algorithm for bootstrap test error estimate on normals

Algorithm `testerror_normal()`
sum=0;
for b=1:B

Random resampling on Z to obtain Z^b

for i=1:K

Compute $\mathbf{g^b}(p_i)$
Make orientation of normal consistent with $\mathbf{n}_1$
Compute $\theta(\mathbf{g^b}(p), \mathbf{n}_i)$
sum+= $\theta(\mathbf{g^b}(p_i), \mathbf{n}_i)$

end for

end for
Compute mean of bootstrap normal, $\Sigma\mathbf{g^i}(p_i)/B$
Compute error estimates as in (5.10)

PCA normal does not necessarily keep the consistency of normal direction, but when we are estimating normals from bootstrap samples, we orientate it accordingly, so that the angle difference is minimum and Equation 5.10 is estimated correctly.

When we estimate the normal on each points $\{p_1, p_2, ..., p_K\}$ in a particular neighbourhood around point d_i, it is worth mentioning that, when we try to compute $\mathbf{g}(x_i)$ at $x_i = p_i$, the neighbourhood that we use is still Z and not the K-nearest neighbourhood for point p_i.

The consequence of this is that even for different p_i's, the value of $\mathbf{g^b}(p_i)$ will always be the same as $\mathbf{g^b}(d_i)$, as PCA normal creates a plane for neighbourhood Z, thus, giving the similar normal value for its neighbourhood as well. Of course, if this is extended to a different method of normal computation such as normal from quadratic fitting, $\mathbf{g^b}(d_i)$ may be different.

5.5.4 Result and Discussion

As usual, our result is validated on a simple model, sphere and cube. As this is a new approach, we feel that it is necessary to do a verification procedure. For the verification, we would see the error estimation on different value of K-nearest neighbourhood.

Our expectation of error estimates is as follow: for a sphere with some noise, a normal estimation would be better if we use more points in the neighbourhood. Thus, when we increase K, we would expect the error to decrease. For cube, the error would be different depending on the location of the point itself. Assuming a point is close to the sharp edges of a model, the PCA normal computed would be good as long as it does not include the point on the other side of the cube. When K includes the points in the feature area, we would assume that the normal computation would be relatively worse than before it includes the feature. So, when we increase the value of K, the error should decrease and increase immediately when it touches the feature.

We start with a test on a sphere of 4098 points with different noise levels as shown in Figure 5.6.

We will rely on the value of the estimated bootstrap error and compare it with the ground-truth error. We can see that a similar pattern of approximation is given by our error estimation, although the value is higher than the ground truth error for every different noise level.

Although the error estimation is not exactly the same as the ground truth, we can see that the values are comparable for different noise levels. Let us say that we choose our neighbourhood of K to be 50. The error estimation and the

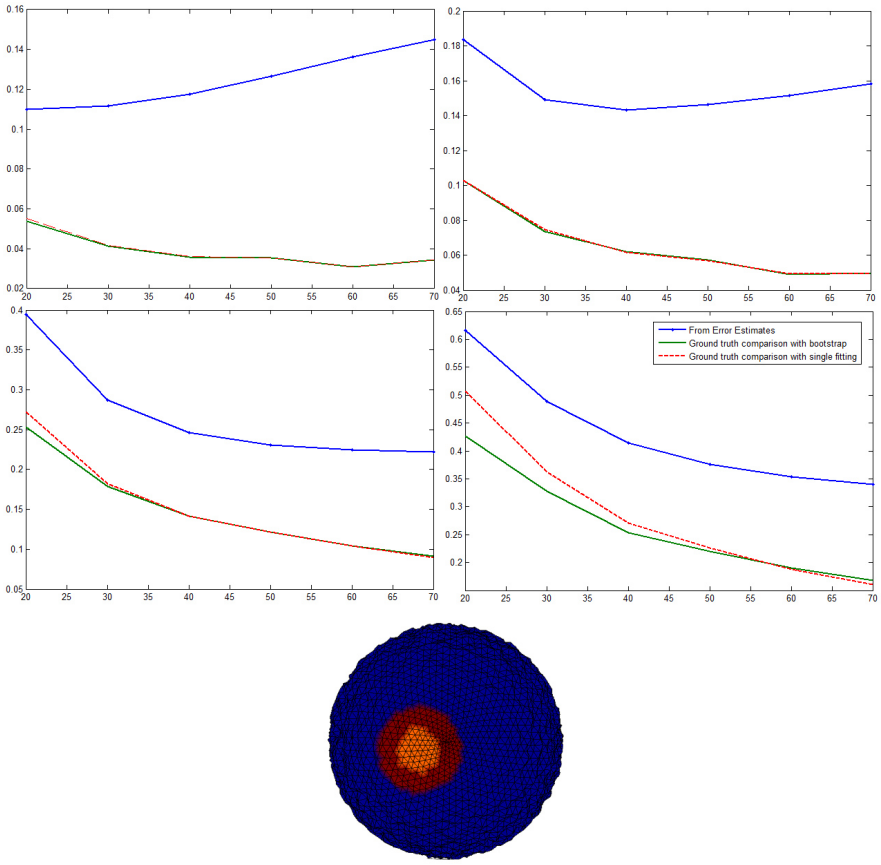

Figure 5.6: Error estimation from different sizes of neighbourhoods on a selected region for a sphere with 0.25, 0.5, 1.0 and 1.5 noise level. The result shown is an average for the selected region (orange and dark red coloured) as in the figure below, which includes 100 points.

ground truth for the bootstrap method and the single fitting are given in Table 5.7 .

Table 5.7 is an extract of Figure 5.6 in a neighbourhood of 50. Here, we can see that our estimates gives a good approximation to compare the quality of normal estimation for different noise level.

Going back to Figure 5.6 and looking at each graph individually, we try to see if we can choose in which neighbourhood the normal estimate is at its best. For a

	Estimated error	Ground truth error	
Noise level		Bootstrap	Single fitting
0.25	0.1264	0.0354	0.0354
0.50	0.1467	0.0564	0.0569
1.00	0.2304	0.1216	0.1213
1.50	0.3769	0.2234	0.2260

Table 5.7: Estimated error and ground truth error (for bootstrap and single fitting) on a region (as shown in Figure 5.6) for a sphere at neighbourhood of 50. Values are shown for different noise level.

1.5 noise level (bottom right), we can see that the blue line (error estimates) goes down as we increase the size of the neighbourhood K. This pattern would imply that choosing a larger K would be better, and this pattern follows the ground truth error (both bootstrap and single fitting). The same pattern applies to the 1.0 noise level.

Noise level	K increase from 20 to 70
0.25	Error increase from 0.10 to 0.15
0.50	Error decrease from 0.18 to 0.14 and increase again to 0.16
1.00	Error decrease from 0.40 to 0.25
1.50	Error decrease from 0.60 to 0.40

Table 5.8: Characteristic of the error estimation for different noise level. The value used had been rounded for discussion.

Table 5.8 is built up from Figure 5.6 to summarize the behaviour of the error estimates. We can see that a lower noise level produces lower error estimates in its range. Although the graph of 0.25 and 0.5 does not follow this pattern, we can see that the range of increment is relatively small. While the increment of 0.05 and 0.02 are shown for 0.25 and 0.5 noise level respectively, the decrement of 0.15 and 0.20 shown for 1.0 and 1.5 noise level which is relatively bigger.

If we choose to use bootstrap error estimates to choose an optimum value of K, it may be worth noting the difference between the error values. However, we

would like to note that bootstrap test error on normal estimates can certainly help us, giving us a better idea of our normal estimates. The test error estimates improves significantly compared to variance estimates as well as shows a nice pattern on our normal estimates. In practice, the green and red lines in the graph showing the ground truth error are not available at all, making this bootstrap test error stands on its own.

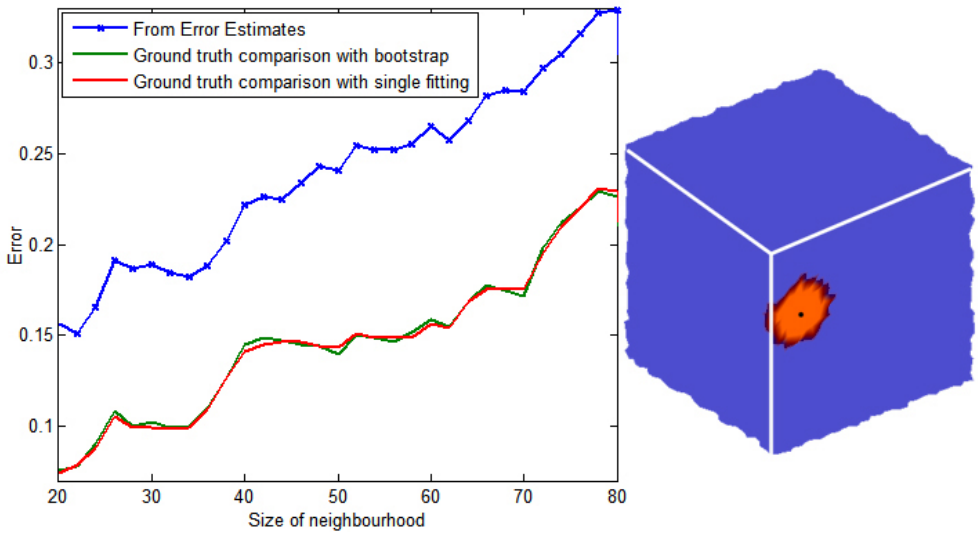

Figure 5.7: Error estimation from different sizes of the neighbourhoods. The cube (right) shows the considered point (black dot), its 50 nearest neighbourhood coloured in orange, and the rest of the neighbourhood up to 80 coloured in dark red.

Figure 5.7 displays the error estimation of angle difference for different sizes of a neighbourhood on a particular point (the considered point is marked as a black dot in the figure to the right) on a cube of 6146 points with 0.5 error level. We increased the size of the neighbourhood ranging from 20 to 80 in steps of 2 and displayed three different errors. The first one (red line) is the angle difference between PCA normal and the ground truth normal. To reiterate, we define

ground truth normal from the initial non-noisy model which is obtained from the underlying connectivity. The second one (green line) is the angle difference between bootstrap normal and the ground truth normal. The third one (blue line) is the bootstrap error estimate, Err_{boot} that estimated from the bootstrap procedure.

One of the motivations driving us to obtain error estimations is to be able to compute the optimum neighbourhood size for normal estimation. Unlike previous sections where we displayed the statistics for the whole model, here we only select a patch or a section on the model to see a certain behaviour when we increase the size of the neighbourhood. An optimum neighbourhood size should vary for different properties of the surface. For instance, in an area where surface features are not present, we should include more points in the computation, thus including a larger neighbourhood size. On the other hand, if the neighbourhood starts to include the feature area, the error may be high, therefore indicating lower neighbourhood size.

We run the same algorithm on the Bunny in the neighbourhood around its body as shown on Figure 5.8. We can observe the same pattern that we have seen in the Sphere model previously, as the area contains no feature.

Figure 5.9 displays the error estimates around the tail of the Bunny which include feature area when we increase to a certain neighbourhood size. Referring to the red and green line which display the ground truth normal can give us an idea of the reliability of the normal estimates in that particular area. For 0.25 and 0.5 noise level, the estimates are better for $K \leq 40$, which we assume is when it reaches the feature area. We can see that the blue line indicates a constant increment, which indicates that the estimates would be better if we use a smaller

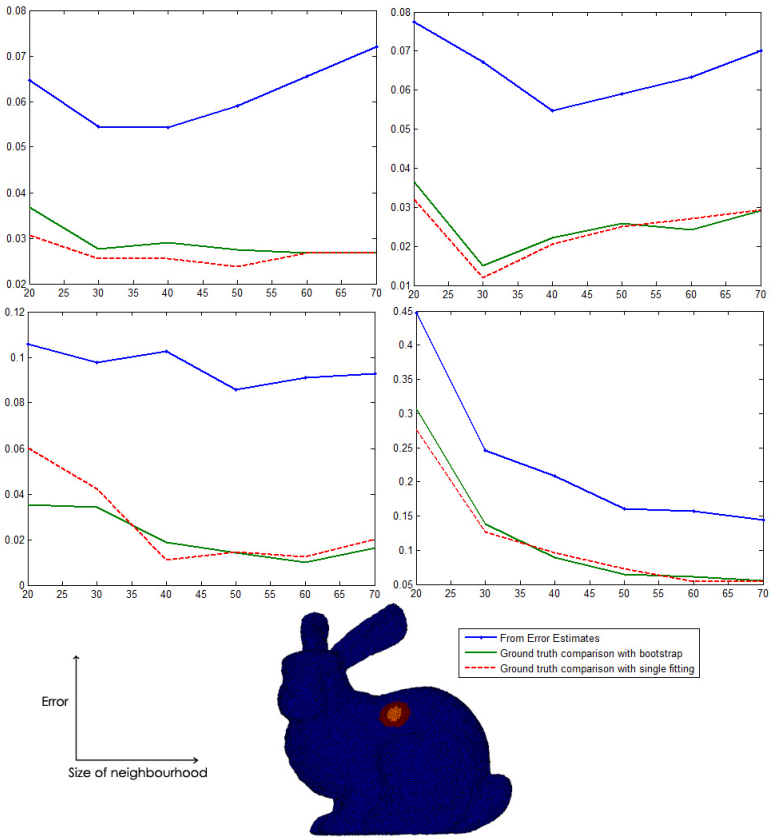

Figure 5.8: Error estimation from different size of the neighbourhood on a region of the Bunny of 34835 points with 0.25, 0.5, 1.0 and 1.5 noise level (top left, top right, bottom left and bottom right respectively). The result shown is an average for the selected region (orange and dark red coloured), as in the figure in the bottom, which includes 70 points.

neighbourhood rather than a larger, as opposed to what we had in Figure 5.8.

As for 1.0 and 1.5 noise level, the ground truth error shows that increasing the size of the neighbourhood will result in better normal estimates, which is captured nicely by our error estimates (blue line).

We move on to include another variable, which is the different order of fitting. Here, we wish to see if we can verify that a fitting is better than the other by relying on error estimates. Figure 5.10 shows the test error estimates on nor-

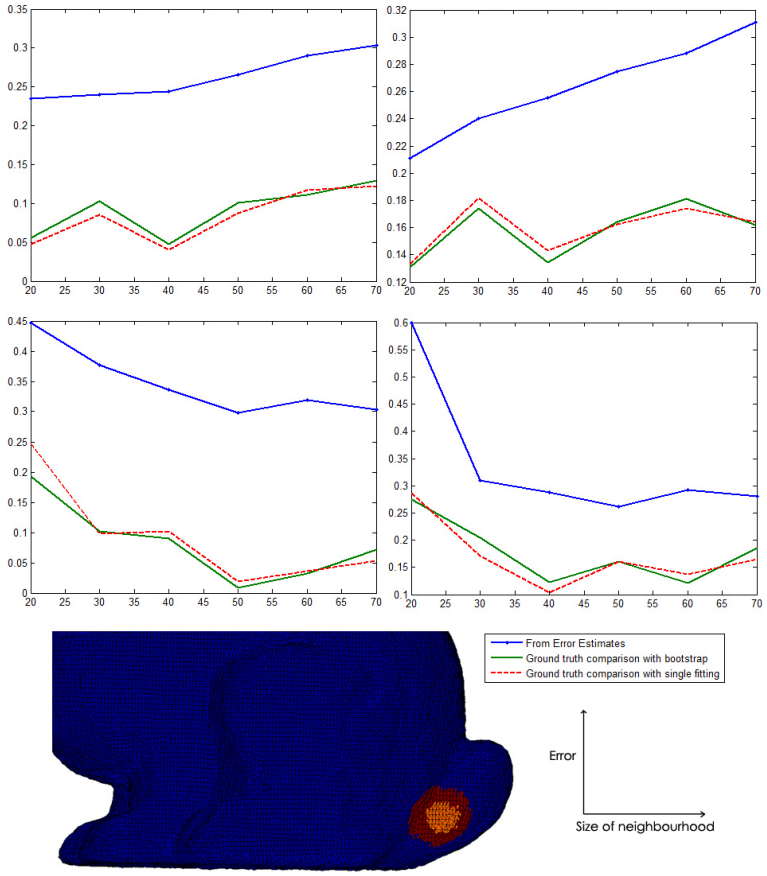

Figure 5.9: Error estimation from different size of the neighbourhood on a region of the Bunny of 34835 points with 0.25, 0.5, 1.0 and 1.5 noise level. The result shown is an average for the selected region (orange and dark red coloured) which includes 200 points, as shown in the bottom of the figure .

mal estimates using different orders of polynomial surface fitting. We display the results for the selected region on a sphere for different levels of noise. Our error estimates shows a higher error for normal, based on fitting in quartic, cubic, quadratic and PCA normal. One exception is 1.50 noise level, where test error estimates for normal based on quadratic fitting and PCA normal are both overlapping one another (dotted blue and red line).

From the ground truth error, we can see that the error estimates give quite

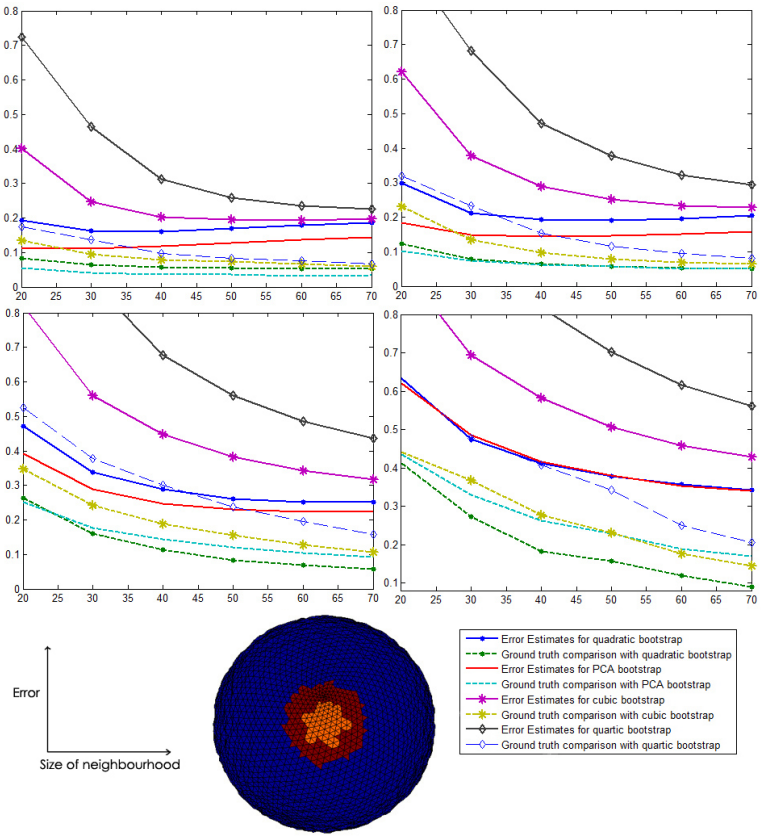

Figure 5.10: Error estimation for different sizes of neighbourhoods on a region of a sphere with 0.25, 0.5, 1.0 and 1.5 noise level, using bootstrap on PCA, quadratic, cubic and quartic fitting. The result shown is an average for the selected region (orange and dark red coloured) which includes 100 points as shown in the bottom of the figure.

reliable estimates, but further caution needs to be taken into consideration. While there is not much issue in comparing quartic and cubic cases, error estimates for PCA normal does not react exactly as we wanted.

Figure 5.11 details the observation from Figure 5.10. On a smaller scale, the error estimates do not contradict the ground truth error, except for 1.0 noise level. We had noticed that there seems to be a small range where we cannot conclude that one method is better than the other, or as in previous case, a neighbourhood

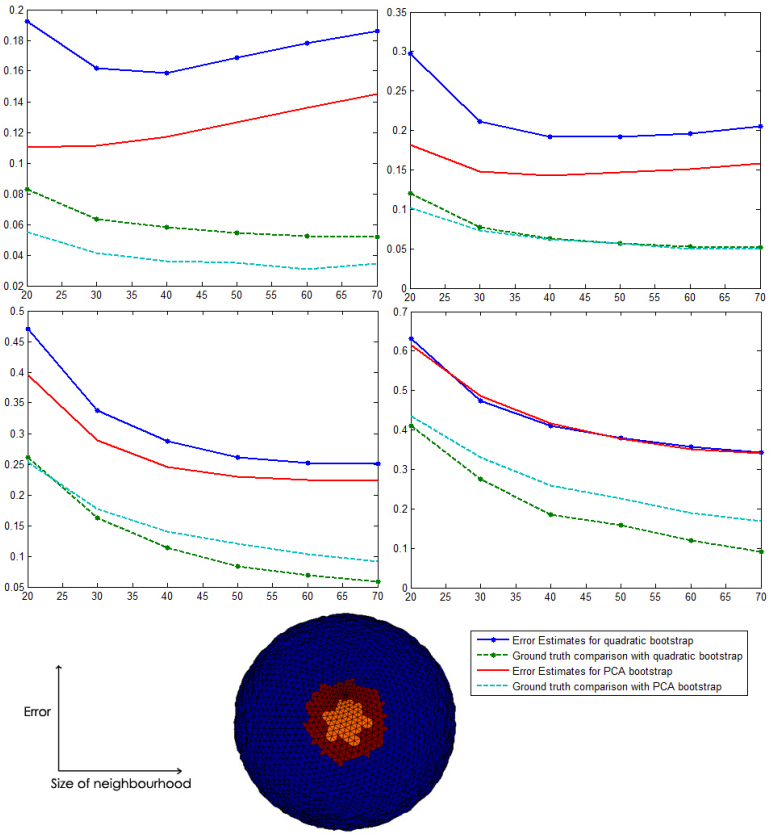

Figure 5.11: Error estimation for different sizes of the neighbourhoods on a region of a sphere with 0.25, 0.5, 1.0 and 1.5 noise level, using bootstrap on PCA and quadratic fitting. The result shown is an average for the selected region (orange and dark red coloured)which includes 100 points as shown in the bottom of the figure.

gives better estimates than the other.

While it has been our motivation to obtain an optimum neighbourhood size for a particular fitting, we have not managed to point an exact value and choose to define a range from visual inspection. This limitation of evaluating the range of reliability of error estimates should be noted in further research. However, we have shown that bootstrap error estimates on normal has provided an insight which has not been available to us before. As in real life, ground truth error is

not something that can be obtained; bootstrap error estimates can help us to understand the property of point sets without relying on visual inspection.

5.6 Summary

In this chapter, we explore variance and error estimates in the context of normal estimation on noisy point clouds. Using variance, we can obtain a confidence value for our estimates and later use it to smooth noisy normals.

To check the quality of our estimates for different neighbourhood sizes, we can not rely on the variance. We had shown this issue as it relates to bias-variance decomposition. Bootstrap error estimates had been proposed and defined accordingly to normal estimates, which differ from our approach in Chapter 4. The result shown that bootstrap error estimates can offer a qualitative insight for normal estimates.

Next, we point out our observation on how orientation may affect normal estimates from bootstrapping, which can be a direction for future research.

5.6.1 Discussion of orientation issues

The bootstrap normals are computed by Eq. 5.4 as averages of B normals. As PCA gives unoriented vectors, the orientations of these B normals were computed separately. Table 5.9 shows the error of the bootstrap normal estimation when different normal orientation methods are used.

The columns under Bootst.(1) show the results when an orientation consistent with the ground truth normal is selected. In this case, the bootstrap normal estimation outperforms PCA. Of course, as the ground truth normal is not known,

Sphere	k=15			k=30		
Noise	PCA	Bootst.(1)	Bootst.(2)	PCA	Bootst.(1)	Bootst.(2)
0.25	0.0546	0.0545	0.1644	0.0296	0.0296	0.1403
0.50	0.1141	0.1137	0.2348	0.0586	0.0591	0.1664
1.00	0.2818	0.2456	0.4633	0.1306	0.1290	0.2665
1.50	0.4881	0.3752	0.6582	0.2601	0.2242	0.4178

Bunny	k=15			k=30		
Noise	PCA	Bootst.(1)	Bootst.(2)	PCA	Bootst.(1)	Bootst.(2)
0.25	0.1388	0.1370	0.2100	0.1732	0.1696	0.2331
0.50	0.1754	0.1664	0.2600	0.1880	0.1817	0.2547
1.00	0.3175	0.2776	0.4172	0.2471	0.2330	0.3206
1.50	0.4660	0.3826	0.5781	0.3278	0.2970	0.4066

Table 5.9: Average angle difference between the ground truth normals and the estimated normals in radians. In Bootst.(1), the normals generated from the bootstrap samples were oriented consistently with the ground truth normal. In Bootst.(2) the orientation was random. The number of bootstrap samples was always $B = 50$.

the method cannot be used for normal estimation; however, the result shows that bootstrap normal estimation combined with an accurate vector orientation algorithm could potentially outperform PCA. The columns under Bootst.(2) show the results when a random orientation is chosen for each normal. In this case, the results are clearly worse than PCA, showing again that a good normal orientation algorithm is essential for accurate bootstrap normal estimations.

Finally, we notice that the consistent orientation of a set of normal estimates which correspond to the same data point set is a problem that has attracted considerably less research interest than the problem of consistent normal orientation over a point set (see for example [51]).

Chapter 6

Bilateral Smoothing

In Chapter 4, we discussed the implementation of bootstrap error estimates on point sets. In this chapter, we will use the obtained information to perform smoothing on noisy models.

Section 6.2 has been presented in

- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Bootstrap test error estimations of polynomial fittings in surface reconstruction", *In VMV*, pages 101-112, 2009

6.1 Overview of bootstrap estimates

Bootstrap error estimation is a statistical method of averaging to predict the error value of a fitting. Given a point cloud of $\mathbf{Z} = \{z_1, z_2, z_3, ..., z_N\}$, bootstrap samples are generated from a random resampling of $\mathbf{Z}$. We fit the bootstrap samples to a function and repeat the process B times. Bootstrap error underestimates the real error, as it uses the training samples for the estimation of error. To improve the

error computation, we use leave-one-out error estimation instead, which mimics the process of cross-validation. An averaging of training error and leave one out error will produce .632 error. We use polynomial surfaces for our fitting, as explained in Chapter 4. A further estimation uses the no-information error rate to detect overfitting and produce one that we call .632+ error. As the estimation of the relative overfitting rate is unstable, we wish to use .632 error instead.

Given a noisy model of a point cloud, error is estimated on each point using bootstrap error estimation. To reiterate, in our experiment, noise is randomly generated in normal directions from an existing model. The function that we use in bootstrapping is a polynomial fitting degree 3 or 4. Generally, we can use any relevant functions for the purpose of bootstrapping.

Figure 6.1 shows example of the bootstrap error estimation on the Bunny with 1.0 noise level and on Bimba with 0.5 noise level. For the Bunny, the area with less features is estimated around 1.0, with a bit of cyan and yellow representing around 0.9 and 1.1 respectively. As for the Bimba, the non-feature area remains around 0.5 while the feature area has a higher value. Note that both error values in the figures are capped for visualisation.

There are three parts in this chapter. Firstly, we will address a straight forward smoothing where values are obtained from the bootstrapping procedure. To overcome the limitation of it, we will later introduce a smoothing algorithm which takes some other factors into account. In particular, this is a smoothing algorithm which relied on factors from distance and error value from neighbouring points. Finally, we will take into account the density of the model to avoid oversmoothing.

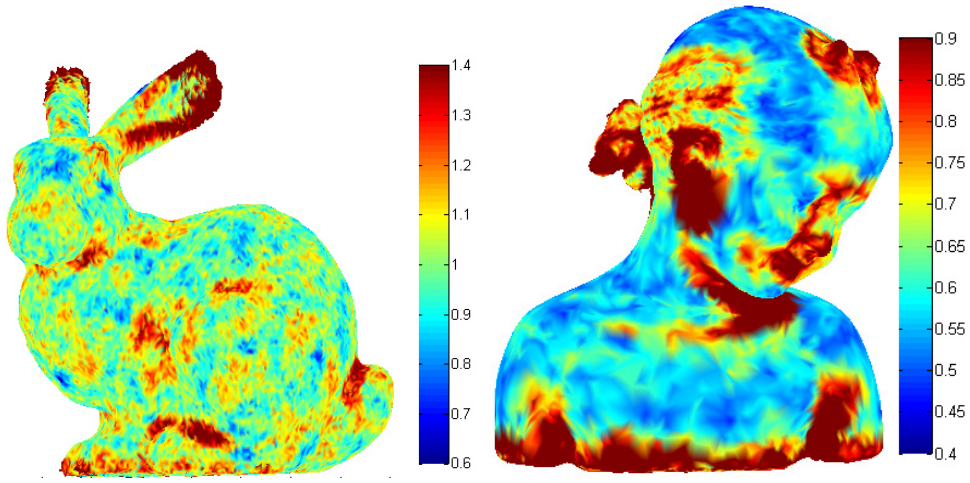

Figure 6.1: Bootstrap error estimation on the Bunny with 1.0 noise level (left) and Bimba with 0.5 noise level (right). This is the colour mapping of the error. The polynomial fitting of 50 neighbouring points was used for the bootstrapping. For visualisation, the values of error are capped in between [0.6,1.4] for Bunny and [0.4,0.9] for Bimba.

6.2 Naive smoothing

In a simple application of the bootstrap method for test error estimation, we use the fitted bootstrap surfaces to denoise the point set. In particular, we project each point to the average of the values at that point of all the bootstrap models, that is

$$(\mathbf{x}_i, y_i) \rightarrow (\mathbf{x}_i, \frac{1}{B}\sum_{b=1}^{B} f^{*b}(\mathbf{x}_i)) \tag{6.1}$$

Fig. 6.2 (top) shows the Bimba model with 0.25 added noise, before and after denoising.

As is the case with all point set denoising algorithms based on projections to locally fitted surfaces, a low quality surface fitting (caused for example by a poorly estimated tangent plane) may actually increase the noise. An example of this noise increase can be seen in Fig. 6.2, near the ear of the Bimba model.

In the case of bootstrap surfaces, we can use the test error estimates to detect poor quality fittings and avoid using them for denoising. Assuming that the main source of test error is surface features rather than noise, the proposed algorithm can remove the noise while avoiding surface degradation near the features. Fig. 6.2 (bottom) shows the results of the denoising algorithm: the projection of Eq. 6.1 is applied only if the .632+ error is below a threshold.

From Table 4.2 we see that the average .632+ error for this model is 0.48. We show the results for thresholds of 0.6 (bottom left) and 0.5 (bottom right), and we notice that the problems caused by poor quality local fittings have been largely resolved.

Fig. 6.3 shows the results of denoising the Cube model. We notice a smooth surface is generated on each side of the cube.

The value of error in a fitting is increased if the area is high in noise or if the fitting is bad. Consider Bunny in Figure 6.1: the feature area which is around the tip of its ears causes higher value of error. In this case, this is the result of a bad fitting, due to the fact that polynomial fitting that we have used in that particular area does not estimate well. For the non-feature area, the error remains around 1.0, which is caused by the noise. The cause of error is not distinguishable. It will either be from the noise, due to the feature, or both. While this is a limitation, it should not be a major issue in smoothing. Our aim is to do a projection based on the information provided from error estimation. While a high value of error could not define the cause, a low value of error implies that both the fitting of the chosen function is good and the noise is small. Hence a straightforward projection is possible and should produce a fine result.

An example is shown in Figure 6.3. From a noisy cube (top left), we estimate

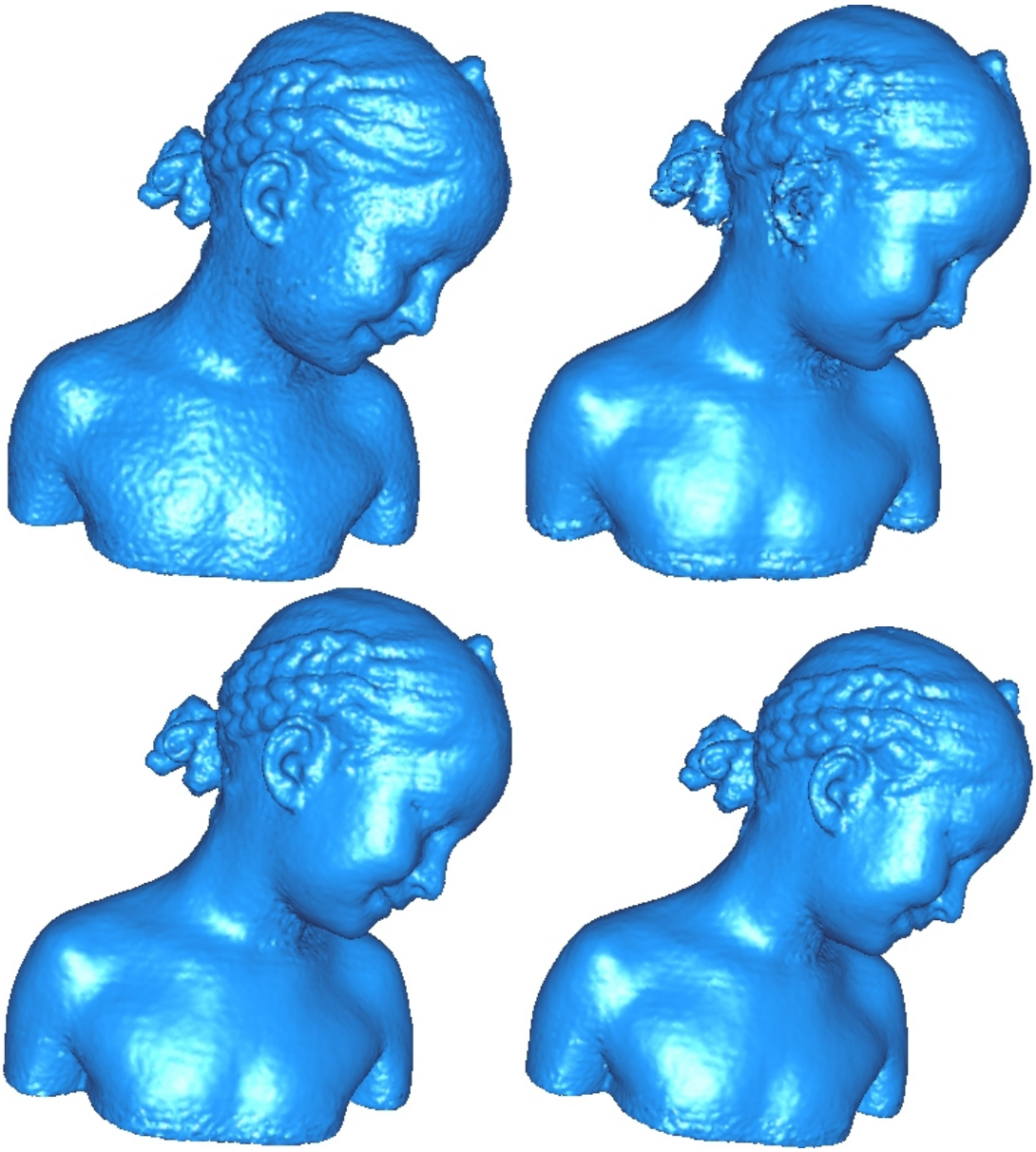

Figure 6.2: **Top:** The Bimba model with 0.25 added noise, before and after naive bootstrap denoising. **Bottom:** The Bimba model denoised by projections to good quality local fittings only. The error thresholds are 0.6 and 0.5, respectively.

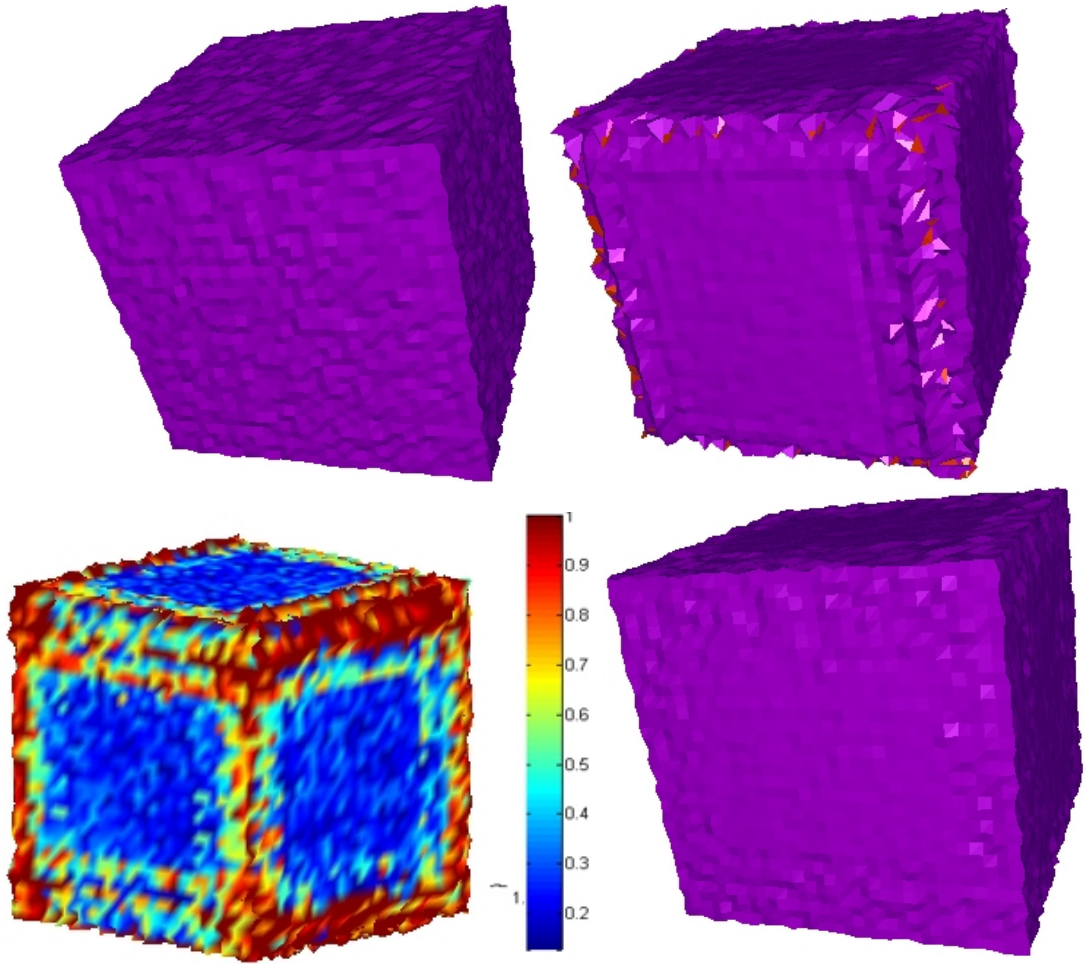

Figure 6.3: **Top:** The Cube model with 0.25 added noise, before (left) and after naive bootstrap denoising (right). **Bottom:** The colormap of the .632+ errors (left), and the model denoised by projection to good quality local fittings only (right).

the error (bottom left). When we project the value straight away, we will get the result as shown in top right of the figure. Around the edges of the cube, we observe that the high value of estimated error is actually caused by the bad fitting due to the feature area and, in this case, not from the higher level of noise in that particular area.

We proposed a naive smoothing of a noisy model. For the area with low estimated error, we project each point to the fitting we obtained from the bootstrap procedure. This seems to work for the model we used, as the error is relatively low (in this case it is 0.25 noise level). Our confidence level of the projection is the same as the estimated error, and thus we omit the projection of the area with a higher value of error.

However, the limitation of this method is evident when the noise level is generally high, such as in Bunny in Figure 6.1. In this case, the naive projection would not work, as our confidence level is the same as the estimated error. Thus the model will remain noisy.

We should also note that in practice, it is quite common to have models with a small amount of noise, as we seen in scanned data in Section 4.2.2 where Fertility and Ramses model have errors less than 0.25. It is also more challenging to denoise point sets rather than meshes. As an additional note, we only use the connectivity for visualisation, not in the computation.

6.3 Bilateral Filtering

The fact that each point is handled separately results in a projection that still retains the noise. As the error is estimated in a neighbourhood of points, we wish

to project all points in that neighbourhood with the polynomial fitting we used in the bootstrapping step. This method is adaptive, as we rely on the value of the error.

We wish to combine with the idea applied in MLS smoothing, which is to give weight based on the distance of neighbouring points to the central point of the neighbourhood. The estimated error of a point corresponds to the central point, so we would project less if the points are relatively far from the central point.

The essence of the the naive smoothing, as in the previous subsection, is to do the projection if the noise is low. So this is a factor to contribute in doing the smoothing.

Given the point cloud of $\mathbf{Z} = \{z_1, z_2, z_3, ..., z_N\}$, we run the bootstrapping of the polynomial fitting to obtain the estimated projection z_i^B and its error W_i (from Eq 4.5).

For a neighbourhood of K nearest points $\{z_{i,1}, z_{i,2}, z_{i,3}, ..., z_{i,K}\}$ around the considered point $z_{i,0}$, then we would project each point as following

$$z_{i,j} \rightarrow (1 - w_{i,j})z_{i,j} + w_{i,j}z_{i,j}^B \tag{6.2}$$

given that $w_{i,j} = \theta_1(\parallel z_{i,j} - z_{i,0} \parallel)\theta_2(W_i)$. The subscript of i, j for any variable $\Theta_{i,j}$ represents that the value of the variable $\Theta = \{w, z\}$ at the j-th nearest point from z_i. Function $\theta_k(x) = \mathcal{G}(x, h_k)$ is a weight function, which is Gaussian function that has been defined in Equation 5.7. In this case h_k is user-defined value according to its θ_k.

6.3.1 Results

We tested the proposed normal reconstruction method on synthetic models. After stripping off the connectivity, we added some noise as in previous chapters. We have run the Algorithm 4 on Bimba with 0.5 noise level.

Algorithm 4 Algorithm for bilateral smoothing

Algorithm `bilateral(K,B,degree)`
The algorithms run through every point estimating the bootstrap error. Then, it will project the points using the error values and distance as weights.
1. For every point $\mathtt{z}_{i,0}$, `i=1:N`

    1.1 Let $\mathtt{w}_i$`=0` for all i's. Find the K-nearest points $\mathtt{z}_{i,j}$'s for the considered point, j=1:K.

    1.2 Compute the error

        1.2.1 Find the normal for the vertices, $\mathtt{normal}_i$. In practice we compute the PCA-normal.

        1.2.2 Localise the K-nearest neighbourhood of the point to transform it to a planar space.

        1.2.3 Run the bootstrapping algorithm, `bootstrap` and obtain $\mathtt{W}_i$.

2. Get the average of $\mathtt{W}_i$'s to estimate the error of the model.
3. For every point $\mathtt{z}_{i,0}$, `i=1:N`

    3.1 Fit the surface polynomial on the neighbourhood and obtain the fitted values $\mathtt{z}^*_{i,j}$.

    3.2 For the point in the neighbourhood $\mathtt{z}_{i,j}$, `j=1:K`

        Project the point and its neighbourhood to $\mathtt{z}^*_{i,j}$ with weight $\mathtt{w}_{i,j}$ such that $\mathtt{z}_{i,j}\mathtt{=(1-w}_{i,j}\mathtt{)z}_{i,j}\mathtt{+w}_{i,j}\mathtt{z}^*_{i,j}$

Figure 6.4 shows the 0.5 noise level Bimba and its bootstrap projection (on top row). Although the bootstrap projection on its own has improved the noisy model to make it smoother, we will show how a bilateral filtering would provide a

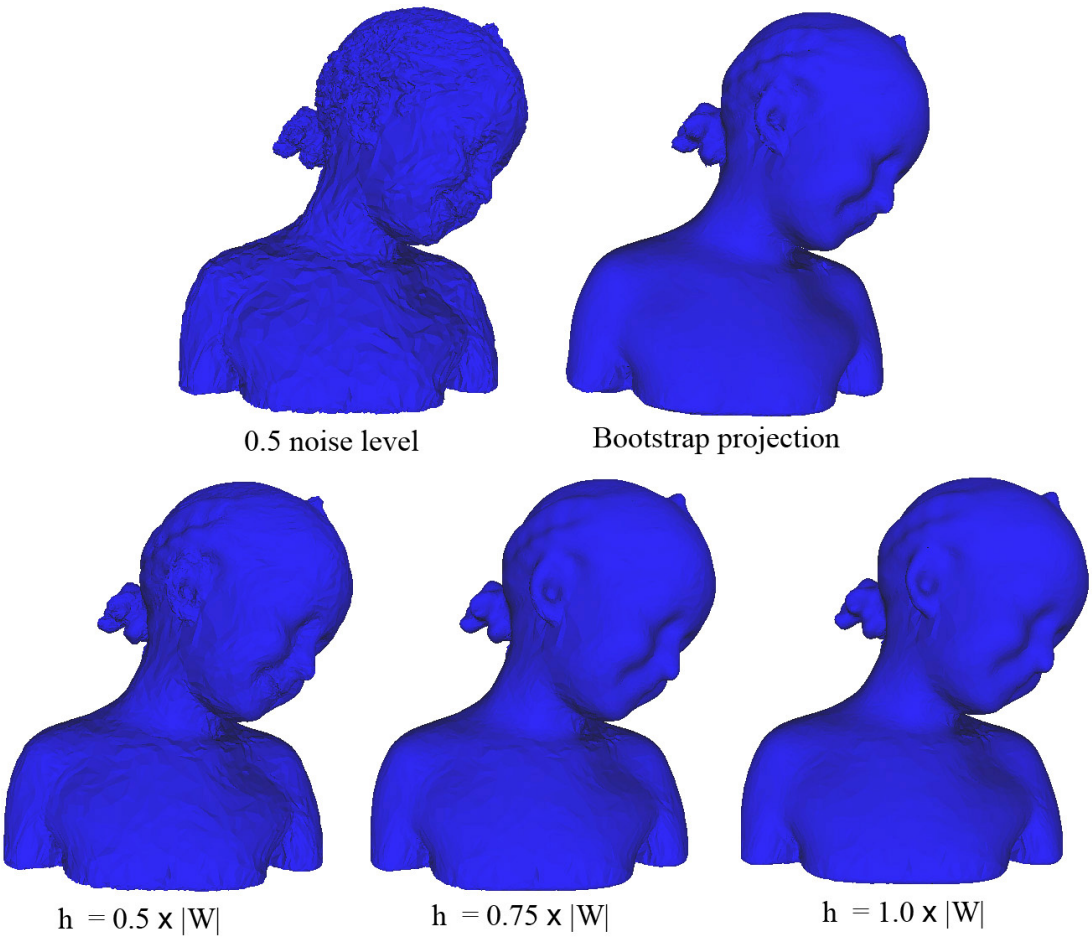

Figure 6.4: Bilateral smoothing on the Bimba with 0.5 noise level. Figure shows the noisy model (top left) and its bootstrap projection (top right). The bottom rows display the bilateral smoothing with different h_2 values. |W| is the expected value of w_i .

better result, especially around the feature area. In the bottom row of the figure, the bilateral smoothing with different h_2 values are shown. We could see that the use of different h_2 provides different degrees of smoothness. Given any value of h_2, the area which has the error value significantly higher than the h_2 would be smoothed less. As shown in the bottom row of Figure 6.4, the model is still noisy when h_2 is valued at half of the model error. Increasing h_2 to 0.75 of the model error, or to the full value of model error would give us a better smoothing.

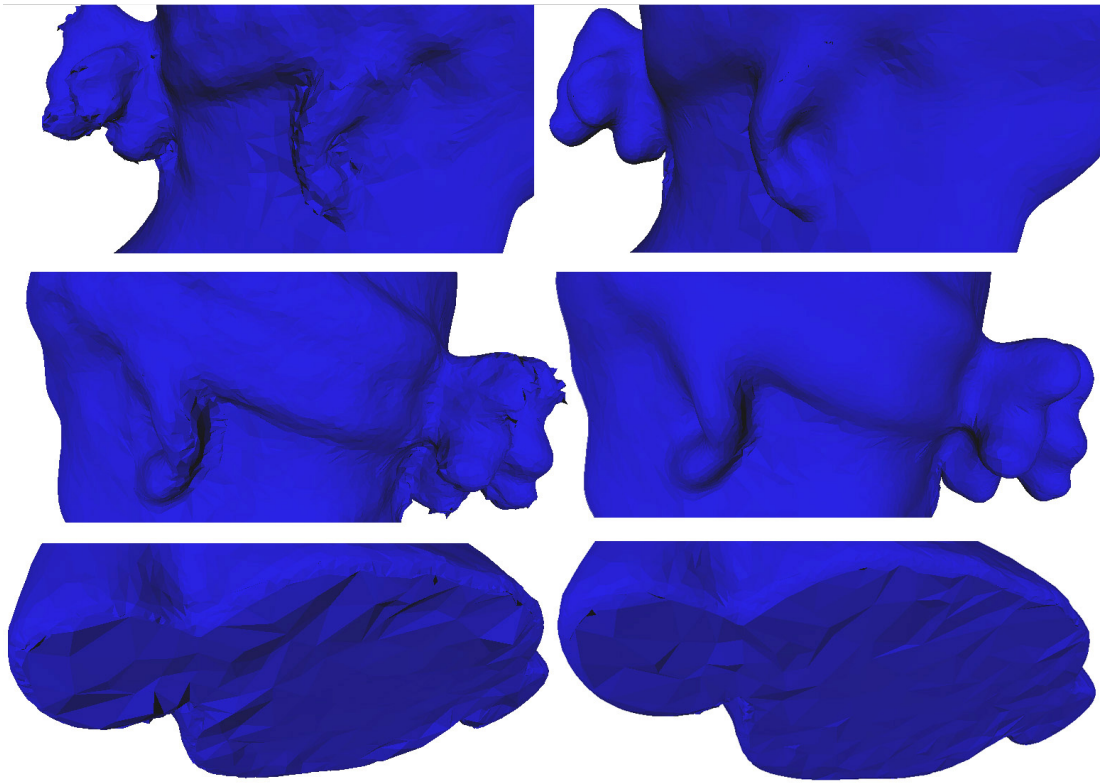

Figure 6.5: Comparison between bootstrap projection (left column) and bilateral filter (with $h_1 = d$, $h_2 = 0.75|W|$) (right column).

Figure 6.5 shows the comparison between the naive projection from the bootstrap averaging of the polynomial fitting and the bilateral filtering, as to show the improvement made by the proposed bilateral filtering. Bootstrap estimation fit a

neighbourhood individually and the quality of the fitting is given by its error estimates. We could see that the feature area of the Bimba such as the ear, hair and bottom edges is badly estimated in Figure 6.5(left). On the other hand, Figure 6.5(right) shows that the area has been smoothed after a bilateral smoothing.

6.3.2 Weights influence on smoothing

We show the effect of using different values of h_1 and h_2 in Figure 6.6. We also display the reflection lines of the model in Figure 6.7 to assist us in looking at the smoothness of the model.

If the variance for distance weight is increased, neighbouring points will have more influence on a considered point. So, when the value of h_1 is increased, the model generally becomes smoother. We can see this effect in both figures as h_1 is increased after each column. While a single filter would smooth out the whole model, a bilateral filtering would avoid smoothing the feature area thus preserving the feature. We should note that by increasing h_2 to infinity, we would be left with a single filter using the distance parameter only. As we can see from Figure 6.6, for $h_1 = 1.5d$ the ear of the Bimba started to deform when the h_2 value increased, which means that we start to lose the weight of the error parameter.

We run the bilateral filtering on the Bunny with 0.5 noise level. The result for different h_1 and h_2 is shown in Figure 6.8 and its reflection lines in Figure 6.9.

6.4 Multi-pass smoothing

There are a few things that we should note from the implementation of Algorithm 4. The smoothing procedure is only done once, but each point is updated before

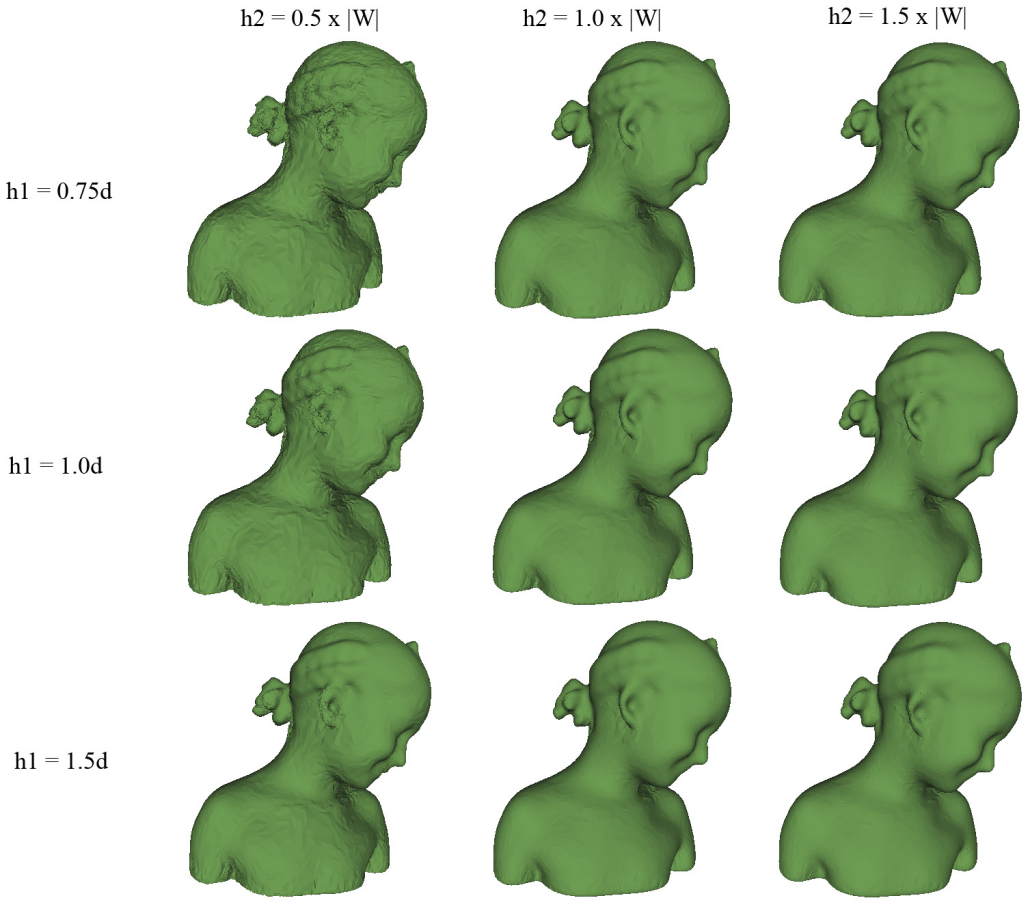

Figure 6.6: The effect of bilateral smoothing for different values of h_1 and h_2.

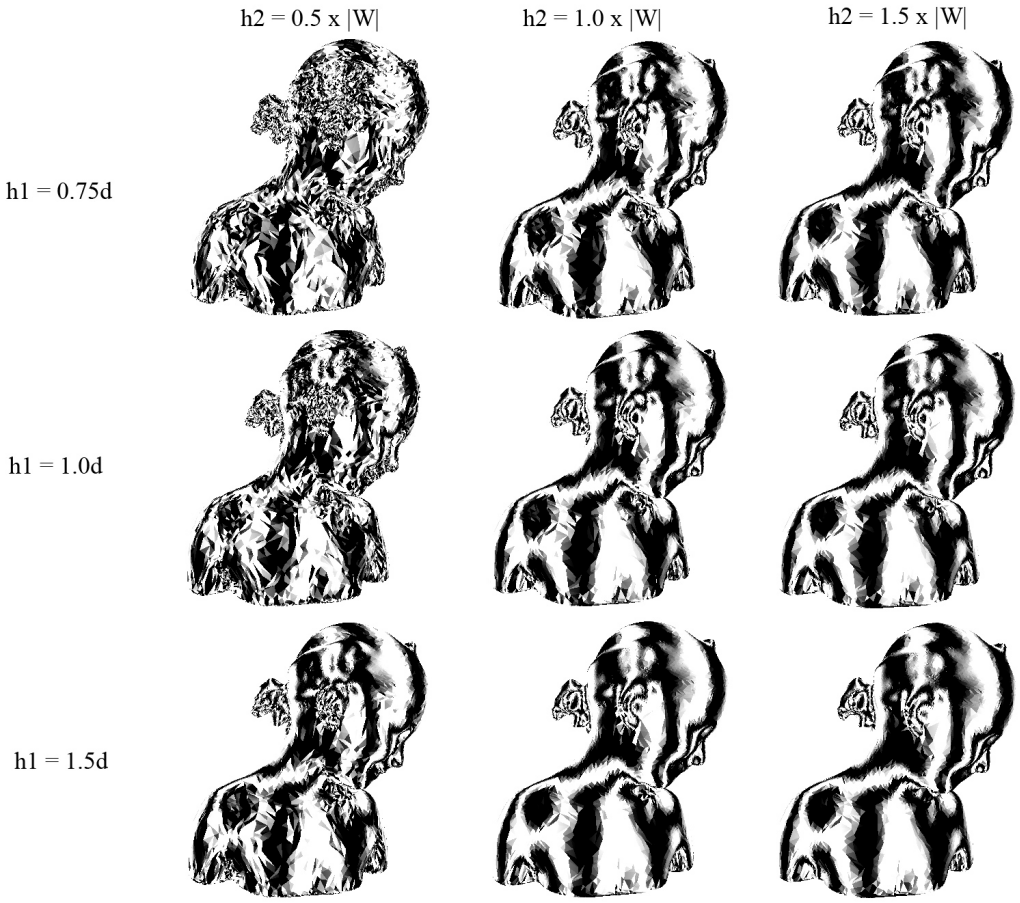

Figure 6.7: Reflection lines for the Bimba with respect to Figure 6.6.

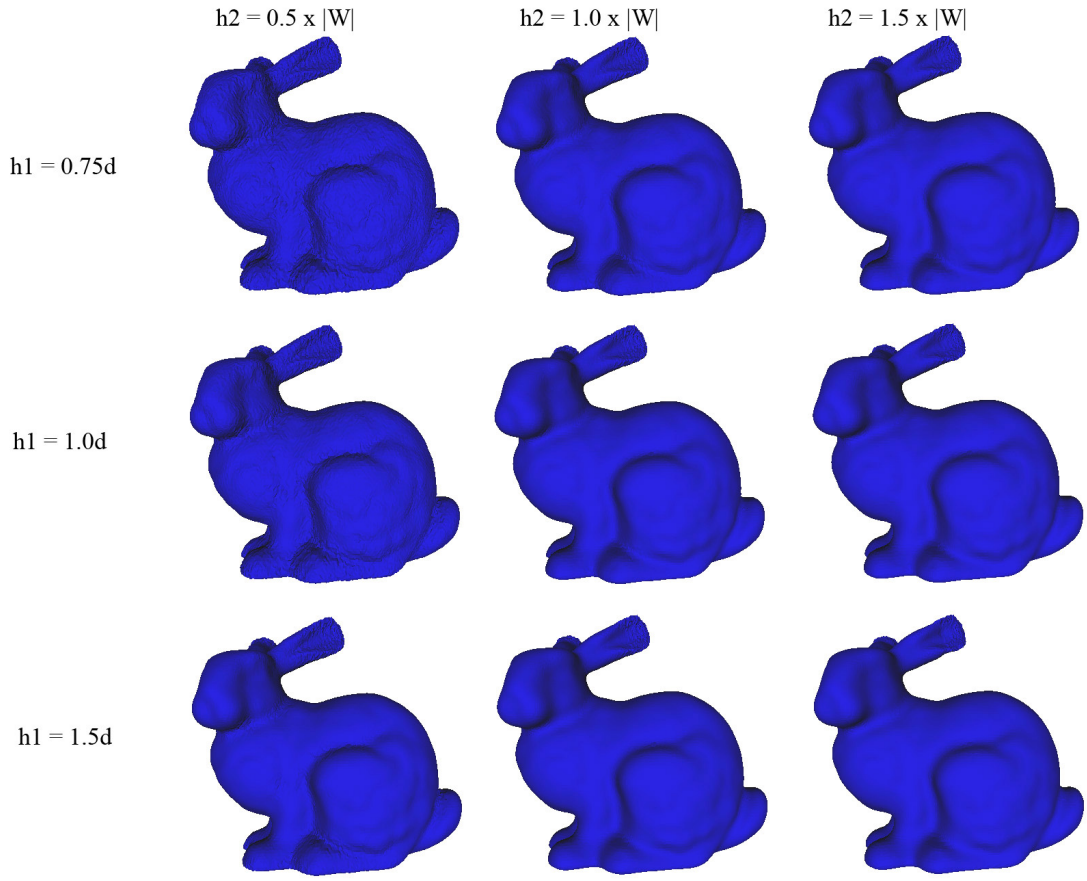

Figure 6.8: The effect of bilateral smoothing for different values of h_1 and h_2 on the Bunny.

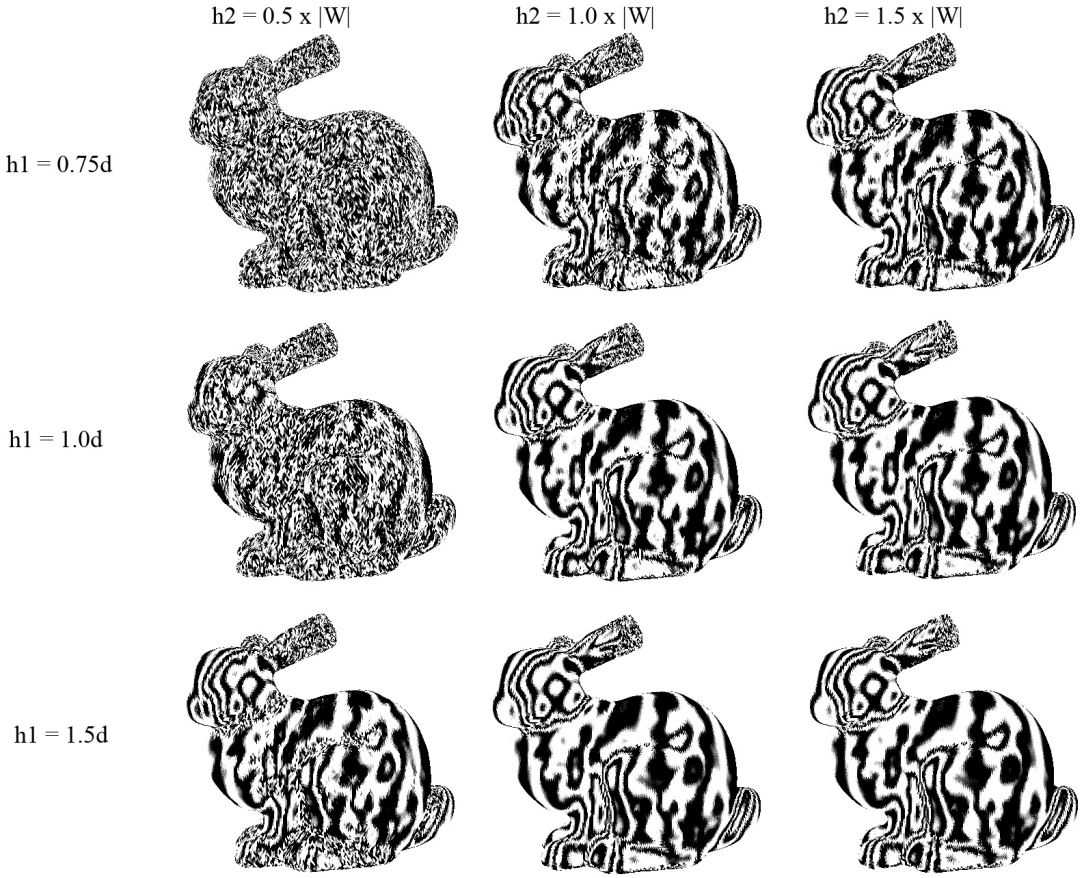

Figure 6.9: Reflection lines for the Bunny with respect to Figure 6.8.

we move on to the next one. So, if i-th and j-th point are neighbouring points in a small vicinity, after the i-th point has been projected through bilateral filter, the computation on the j-th point will include the projected i-th point value instead of the original noisy one. This accelerates the smoothing process and can be run in one pass. However, we suggest the next improvement to be made with regards to other issues that may arise.

Firstly, the size of neighbourhood, K, is user-defined. In practice, for a model with 10,000 to 50,000 points, choosing $K = 100$ seems to be an appropriate value. As the smoothing is an extension of the bootstrap error estimates, we use the same value of K in the smoothing procedure. It is fine in practice to keep a constant K for bootstrap error estimates, as it will detect the overfitting and provide us with information such as higher error around the feature area or wherever it overfits.

In Algorithm 4, using a bilateral filter, we simply project a point onto the polynomial surface constructed with K points, whereas K is defined earlier on. While we can see that Bunny could be smoothed nicely, Bimba does not retain much of its features because of oversmoothing.

Note that one of the main differences between our bilateral filtering method and some other smoothing method is that a point is not projected with respect to the neighbouring points. Instead, depending on the fitting we defined earlier, a point and its neighbourhood will be projected towards the fitting.

As we rely on the distance weight in our bilateral filter, the second issue that we should consider is the density of the model.

We propose an improvement of the bilateral filter by applying multiple iterations in our smoothing procedure as well as adapting to the density of the model,

Algorithm 5 Algorithm for multipass bilateral filtering

Algorithm `bilateral_multi(K,B,degree,iteration,varp,varh)`
The algorithms is similar to Algorithm 4 but with multiple iteration in smoothing step.
for i=1:N

Let $\mathtt{w}_i$=0 for all i's. Find the K-nearest points $\mathtt{z}_{i,j}$'s for the considered point, j=1:K.

Compute the bootstrap error (Algorithm 1, `bootstrap_error`) and obtain $\mathtt{w}_i$

end for
for i=1:N

Speculative neighbourhood selection by linear mapping and obtain number of neighbourhood $\hat{K}_i$ as in Equation 6.3

Compute local average distance between points, $\mathtt{d}_L$

Fit the surface polynomial on the neighbourhood $\hat{K}_i$ and obtain the fitted values $\mathtt{z}^*_{i,j}$.

end for
for k=1:number_of_iteration

for j=1:$\hat{K}_i$

Project the point and its neighbourhood to $\mathtt{z}^*_{i,j}$ with weight $\mathtt{w}_{i,j}$ such that $\mathtt{z}_{i,j}=(1-\mathtt{w}_{i,j})\mathtt{z}_{i,j}+\mathtt{w}_{i,j}\mathtt{z}^*_{i,j}$ as in Equation 6.2 but with $h_1 = a * d_L$ as defined in Equation 6.5

end for

end for

as in Algorithm 5.

In this procedure, we use heuristic neighbourhood selection as shown in Figure 6.10 based on the error given. We select different sizes of neighbourhood as a piecewise linear function with two components, defined as

$$\hat{K}_i = \max\{10, y\} \tag{6.3}$$

where $\hat{K}_i$ is the size of the neighbourhood for i-th point. If the error is high, we will penalise the size of neighbourhood to 10. This is because, if a fitting is bad, we wish to use a lower value of the neighbourhood, as we assume that the cause of high error is from the feature area. Reducing this number would exclude points where feature area present. But if the error is considerably low, in this case ranging from 0 to $mean(w_i)$, we select a y value from a linear equation

$$y = \frac{10 - K}{mean(w)} w_i + K \tag{6.4}$$

given that when error is zero, $x_1 = 0$, the size of the neighbourhood should be at its maximum, which in this case $y_1 = K$. When error $x_2 = mean(w_i)$, we let $y_2 = 10$, so that the size of the neighbourhood will decrease when error increases.

In Chapter 4, we can see that for non-feature areas, the error usually corresponds to the noise level. The choice of $mean(w_i)$ is based on that observation and can also be user-defined.

We would then compute the average distance between points in the local neighbourhood $\hat{K}_i$, denoted d_L. Using multiple iterations, we follow Equation 6.2 where weight is defined by

$$h_1 = a * d_L \tag{6.5}$$

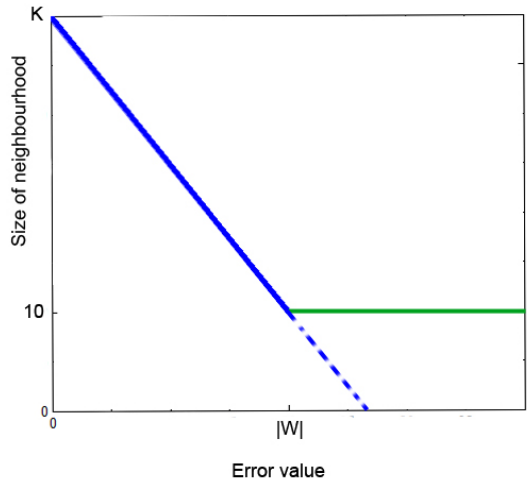

Figure 6.10: Heuristic neighbourhood selection by linear mapping.

where a is a user-defined value.

6.4.1 Implementation and Result

Figures 6.11 and 6.12 show the result of the Bimba model with 0.5 noise level after being smoothed with the bilateral filters. In this example, we choose $h_1 = 0.1d_L$ and $h_2 = 0.3|W|$ and run 300 iterations.

Compared to the previous approach, a multipass smoothing (Algorithm 5) produces a better result due to a couple of reasons. Firstly, a localised d_L enables us to adapt to the sparseness of the model. As the model is not distributed uniformly, the distance from one point to another is not a constant. Thus, for a certain neighbourhood, we may get a different d_L. As our variance h_1 relies on this factor, making it adaptable to the sparseness gives a different impact to the smoothing procedure.

Secondly, multiple iterations smooth the area slowly rather than quickly. This procedure disallow the area to be drastically oversmoothed by a high value of h_1

or h_2 .

Figures 6.11 and 6.12 show a significant improvement compared to Algorithm 4. Visually, the area around the hair is smoothed nicely and not oversmoothed as in Algorithm 4.

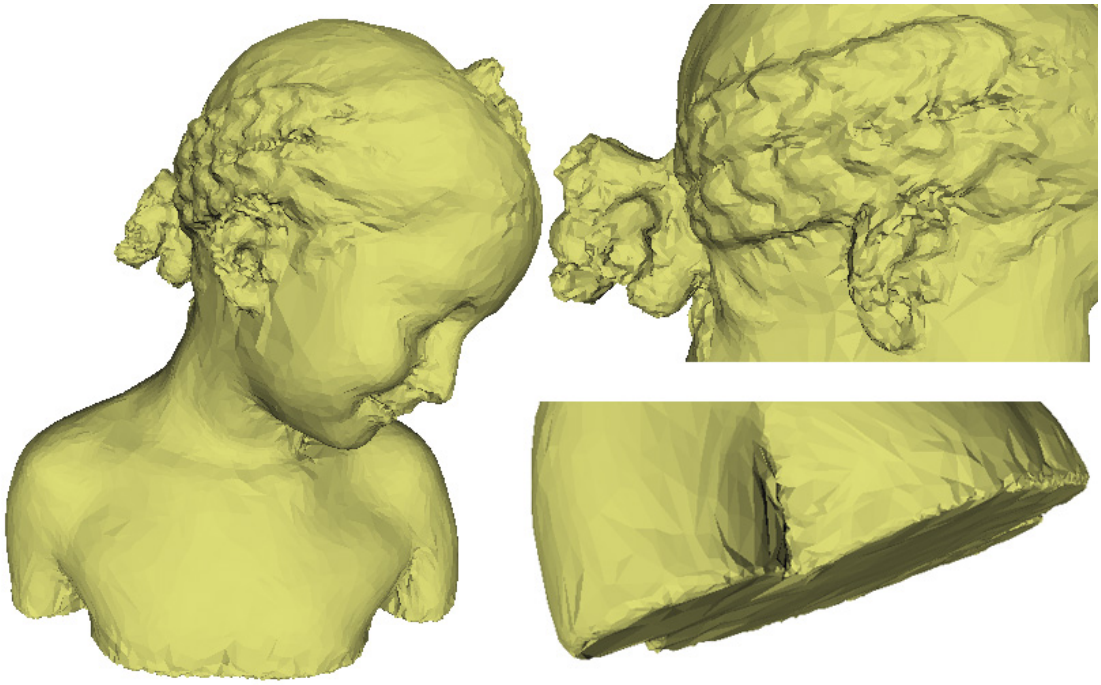

Figure 6.11: The effect of multipass bilateral smoothing with $h_1 = 0.1d_L$ and $h_2 = 0.3|W|$, after 300 iterations, b=10.

We compare this result with the other methods. Figure 6.13 compares the result of our method with Poisson Surface Reconstruction and AMLS. We can see that while the feature area is better preserved with our method compared to Poisson, AMLS supersedes our result in smoothness as well as feature preservation, for instance around the ear.

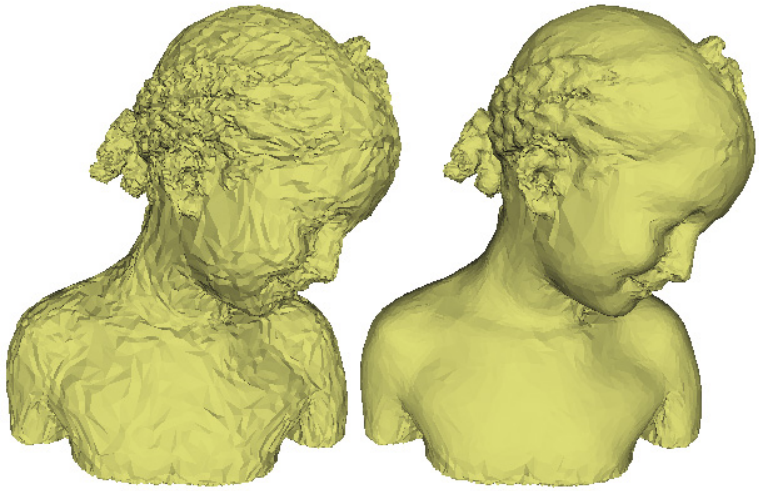

Figure 6.12: The effect of multipass bilateral smoothing with $h_1 = 0.1d_L$ and $h_2 = 0.3|W|$, after 300 iteration, b=10, before (left) and after (right).

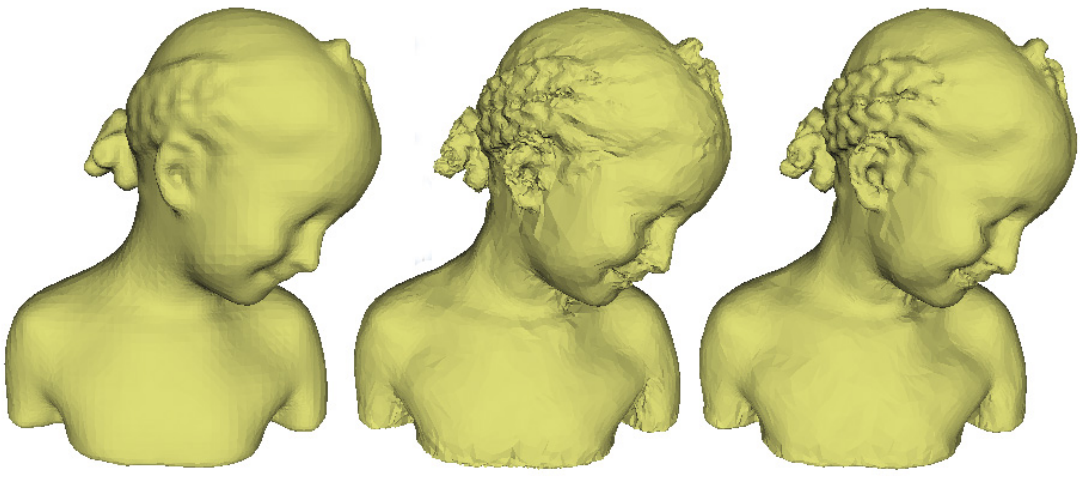

Figure 6.13: Comparison with other methods, from left: Poisson Surface Reconstruction, our method and AMLS.

6.4.2 Summary

Our multipass bilateral smoothing has similarities with MLS, which projects points to a fitting with certain weights. In our case, we used error values as one of the weights. While the method can smooth the model, one of the issues worth analyzing is to find automatically the value for the variances h_1 and h_2.

Another limitation is that this method may not be able to preserve sharp edges such as from Cube or Fandisk. This is a common limitation of point smoothing based on averaging of neighbouring points.

All of these can be directions for future research.

Chapter 7

Feature Detection on Point Clouds

Recovering feature areas is one of the challenging aspects in surface reconstruction. Having a knowledge of the feature on a particular model may help, either in visualising a model or in applying different geometric processing methods on different parts of the model if we wish to do so.

Given a model where noise is present, it may not be necessary to explicitly separate the feature and non-feature areas to apply a procedure such as smoothing. For instance, in sharp edge preservation on meshes, Hildebrandt and Polthier[50] use information derived from mean curvature to denoise a model and recover its sharp features.

As discussed in Section 2.6, there are various ways to differentiate feature areas such as by computing curvature values, normal comparison or error values comparison on the model.

In this chapter, we will discuss some new approaches that we used to detect

feature areas. Firstly, we will use properties of the Principal Component Analysis and later, using information that we have collected from bootstrapping procedures in previous chapters, we will make use of it to detect feature areas.

Section 7.1 has been presented in

- Ahmad Ramli and Ioannis P. Ivrissimtzis. "Distance based feature detection on 3d point sets", *In TPCG*, pages 53-56, 2009.

7.1 Feature detection from PCA

7.1.1 Algorithm

In this section we propose an implicit feature detection algorithm on point sets. At the first stage of the algorithm, the goal is to compute a function giving the distance between a point and its nearest feature. For each point p, we run *Principal Component Analysis (PCA)* on its K-neighborhood, for an increasing sequence of K's. We process the eigenvalues obtained from the PCA, trying to detect when the K-neighborhood has reached a feature. Then, we use the radius of the K-neighborhood as a measure of the distance between p and its nearest feature (See Fig. 7.1). At the second stage of the algorithm we smooth the distance function using bilateral filtering. Finally, the feature points of the point set are detected as the points at a near zero distance form their nearest feature.

The proposed feature detection algorithm is based on the PCA of neighborhoods of varying size. For every vertex p, we calculate the eigenvalues of the covariance matrix defined in Equation 2.1. As a repeat, $a \leq b \leq c$ are the three

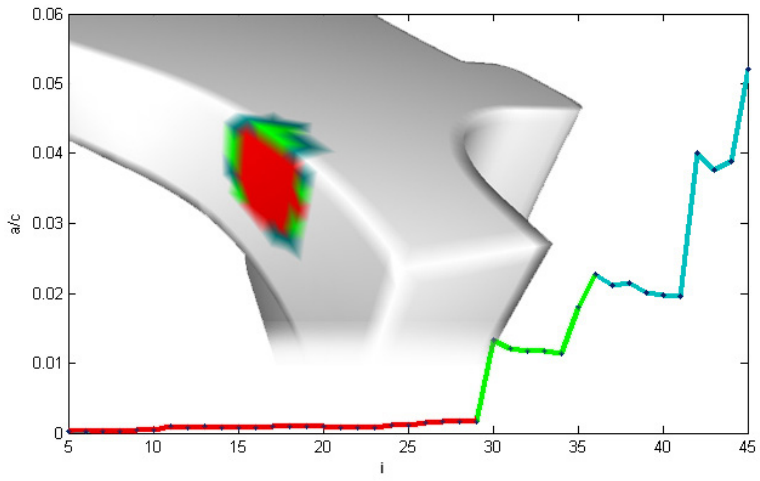

Figure 7.1: An expanding k-neighborhood of a point reaches a feature of the model. The graph shows the ratio of the smallest and largest eigenvalues of the PCA analysis.

eigenvalues of the matrix.

The ratio a/c is used as a measure of the how flat the neighbourhood is. A small value of a/c means that a is small with respect to c, indicating a flat area. [87] has a similar approach, considering the ratio $a/(a+b+c))$ instead.

We keep increasing the i-neighbourhood, creating a function $f_{a/c}(i)$ of values of a/c, until $f_{a/c}(i)$ exceeds a user-defined threshold $T_{a/c}$. Experimentally we found that a value $T_{a/c} = 0.1$ gives satisfactory results. Even though the thresholding of $f_{a/c}(i)$ is by itself a crude instrument for feature detection, we found it sufficient for the purpose of computing the distance function. If $T_{a/c}$ is not exceeded after a certain number of iterations the process will terminate.

Notice that we can take into consideration the derivative of $f_{a/c}(i)$ as well as its value. For example, we may use the function $f_{a/c}(i) + d \cdot f'_{a/c}(i)$ whereas $d \geq 1$ is a user-define value. As shown in Figure 7.2, by thresholding this function, we may compute a more accurate distance function on smooth models, as it shows

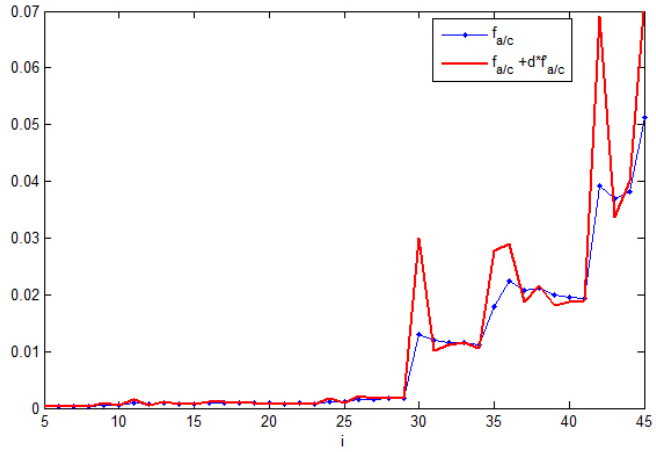

Figure 7.2: Comparison between $f_{a/c}(i)$ and $f_{a/c}(i) + df'_{a/c}(i)$.

a better indication of the jump when it reaches a feature.

However, as Fig. 7.3 indicates, that approach would be less robust in the presence of noise due to the fact that $f_{a/c}(i)$ is far higher than thresholding value.

When the neighborhood has been computed, we assign the radius of that neighborhood as the distance h_p between the point p and its nearest feature. This is the main difference between our method and previous approaches with varying neighbourhoods, where such information is discarded.

The next step is the bilateral filtering of the distance function h. The new value of h'_j of the distance value of at the vertex j is given by

$$h'_j = (1-\alpha)h_j + \alpha \frac{\sum_{i=1}^{n} g_a(\|p_j - q_i\|).g_b(\|h_j - h_i\|).h_i}{\sum_{i=1}^{n} g_a(\|p_j - q_i\|).g_b(\|h_j - h_i\|)} \tag{7.1}$$

where α is a user-define value and g_a, g_b are Gaussian functions. The standard deviation of g_a is the average over the whole point set of the distance between a point and its nearest neighbour, while the standard deviation of g_b was set to 0.01.

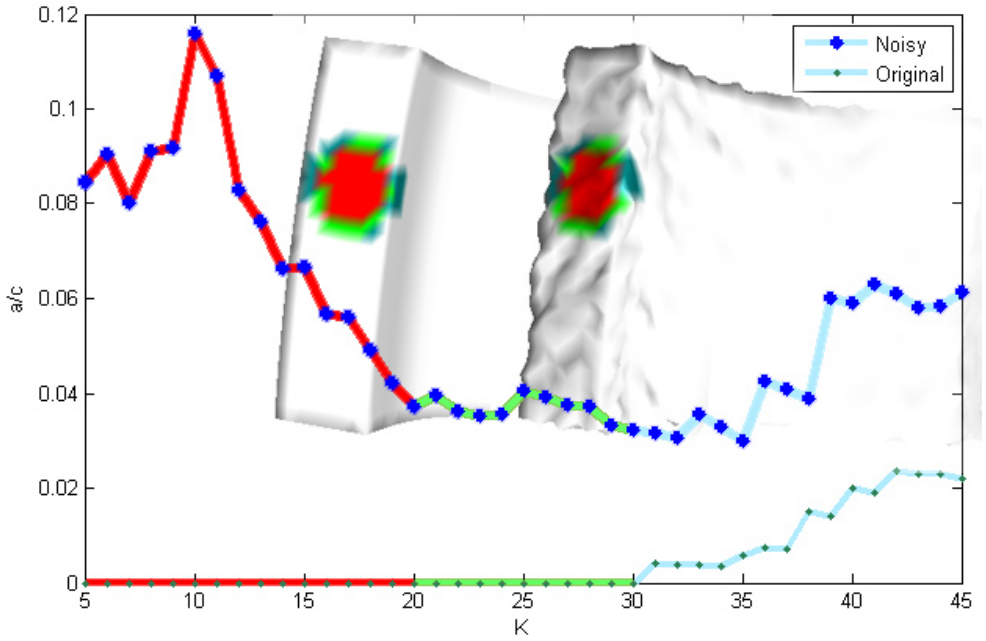

Figure 7.3: The computation of the function $f_{a/c}(i)$ on corresponding points of a smooth and a noisy model. The graph of $f_{a/c}(i)$ shows the problems associated with the direct application of PCA for feature detection.

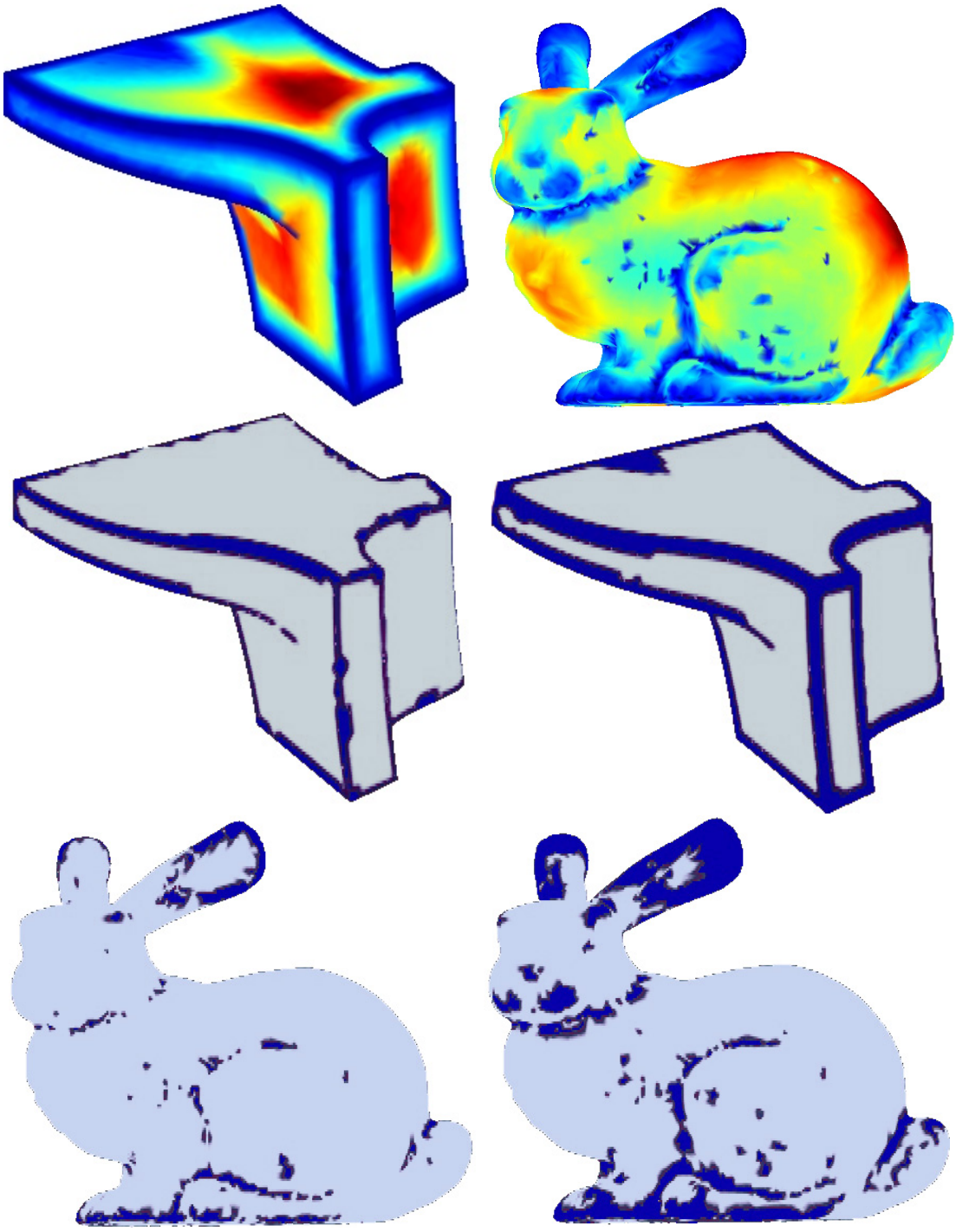

Figure 7.4: **Top:** Colourmaps of the distance function. **Middle:** Feature detection with the threshold value T_h set at 10% and 20% of the maximum value of h. **Bottom:** Feature detection with T_h set at 20% and 30% of the maximum value of h.

To increase the effectiveness of the method, before applying bilateral filtering we find the points with zero distance values, which are the detected features and some noisy points, and replace their values with $-max(radius)$, that is, the negative value of the maximum radius. This way, the features of the model are accentuated and better preserved by the filtering process. Note that otherwise the edge-preserving property of the bilateral filtering would not have been fully utilised, as the distance function is smooth.

After the smoothing process, the features of the point set are extracted by thresholding the distance function. The value of the threshold T_h is critical for the shape of the features. Having T_h as a user-defined parameter is typical in feature detection applications.

7.1.2 Validation

We tested the proposed algorithm on the Fandisk and Bunny models, as well as on the Fandisk model with added noise.

Fig. 7.4 shows the results on the two smooth models. Notice a small region on the flat area of the Fandisk where there seems to be an error in the computation of the distance function. The reason is that the model is thin there, and the two surface sheets that are close to each other are identified as features. However, the problem is solved with a suitable thresholding of the distance function. The results on the Bunny are comparable with [87].

Fig. 7.5 shows the results on the noisy Fandisk model. Note the considerable improvement after the bilateral filtering. We believe that this is indicative of the robustness of the proposed method and its potential compared to naive PCA

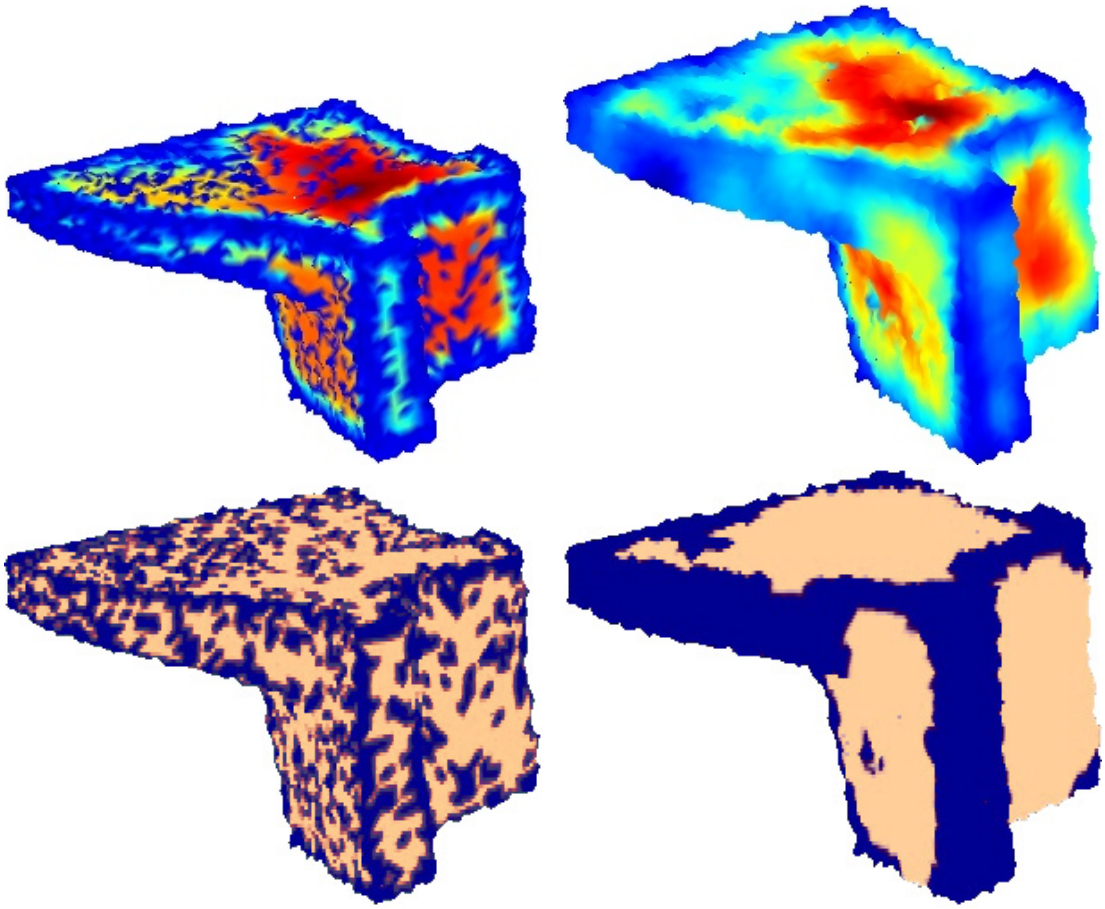

Figure 7.5: **Top:** Colourmap of the distance function before and after bilateral filtering. **Bottom:** Feature detection with $T_h = 0$, before and after bilateral filtering.

analysis.

The feature detection method closer to our approach is the one proposed in [88]. There, similarly to our approach, neighbourhoods of different size are analysed with PCA, however, there the extracted information corresponds to a probability map, while our algorithm creates a distance map. The difference is not just in the interpretation of the PCA results; our map encodes information about the size of the neighborhood, which is ignored in [88]. We believe that the analysis of the PCA results towards the creation of a distance map is a more robust approach because a distance map is expected to be smooth, allowing thus post-processing operations.

7.2 Variance-based feature detection

In this section we extend the discussion from Chapter 5 on the subject of feature detection. From Equation 5.2, we obtain the variance value for the normal estimation on a model. We will use this information to detect the feature area on a model.

7.2.1 Variance and feature area

In estimating normals using bootstrap methods, variance on a particular point gives an indication whether the estimate is relatively accurate or inaccurate in comparison to normal estimates on other points. As discussed in Chapter 5, a high variance is caused either by noise or the presence of a feature. Assuming that the noise level is uniform, a high value of variance will signify the feature area as shown in Figure 7.6. Using this understanding, a colour mapping of the

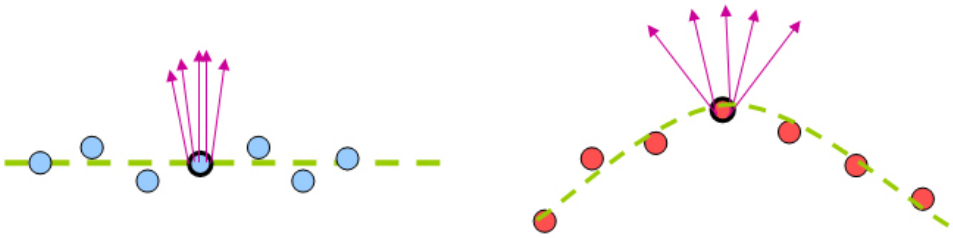

Figure 7.6: Visual representation of normal direction from bootstrap samples on a neighbourhood.

variance could show us the feature and non-feature areas.

Revisiting Figure 5.1, which shows a model with less noise, a straightforward colour mapping may indicate a feature area defined as

$$F = \{i | \nu_i > t_h\} \tag{7.2}$$

whereas F is a set containing the indices with respect to i-th point of the model, ν_i is the variance of the model and t_h is a threshold value to separate the region.

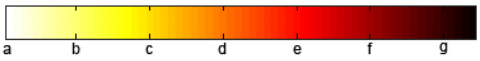

Figure 7.7: An example of color map.

In practice, thresholding is also important for visualisation. We show an example of a color map in Figure 7.7. If most of the variance values lie between a and b and few fall between f and g, a colour mapping would signify the difference of a model between a and b. For this example, thresholding the value at b would be recommended for visualisation. In Figure 5.1, as we define the same t_h value for different noise level, we experience similar effect as in previous examples and the visualisation starts to include non-feature area on a model with higher noise

level.

We tested different t_h values on 0.5 and 1.0 noise level Bunny, and the results are displayed in Figure 7.8 and 7.9 respectively. In these figures, the values which are larger than the threshold value are equalized to t_h and represented in a black colour region. We could see that although the model is noisy, choosing a larger value of t_h may differentiate feature and non-feature areas.

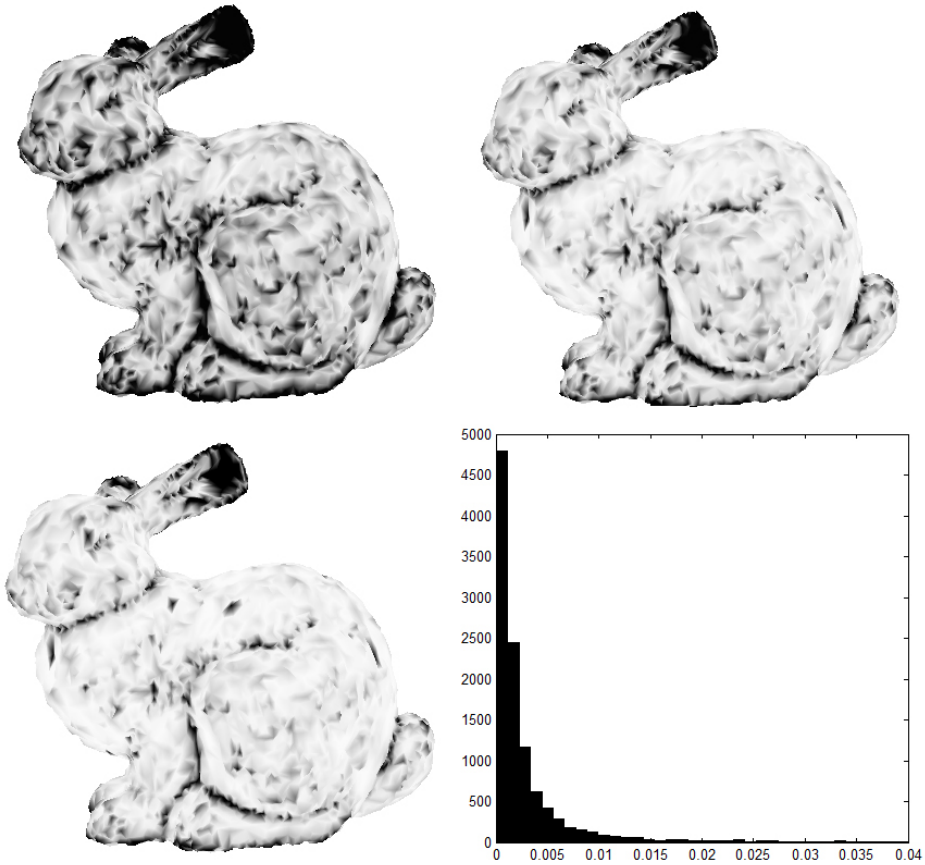

Figure 7.8: Variance mapping of Bunny 0.5 noise level with different value of thresholding. Thresholding value for top-left, top-right and bottom-left is 0.005, 0.010 and 0.015 respectively. Histogram of variance ν is displayed as well and is capped at $\nu_i \leq 0.04$ for visualisation.

It would be quite tedious to generate models and test them visually for each

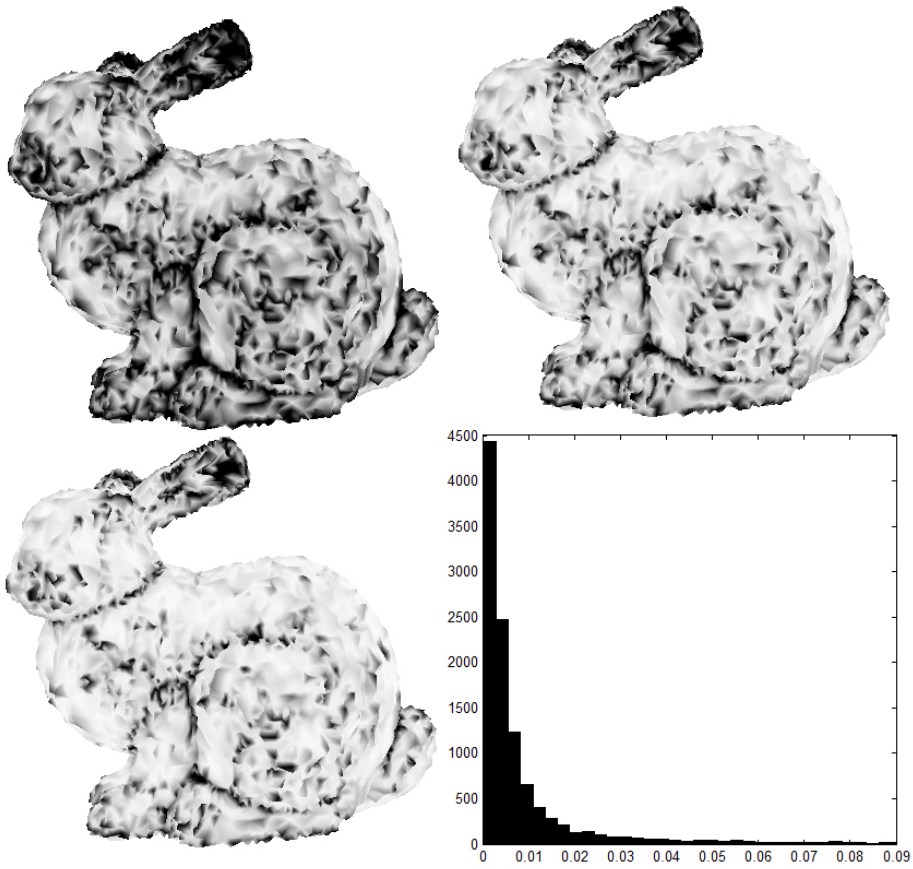

Figure 7.9: Variance mapping of Bunny 1.0 noise level with different value of thresholding. Thresholding value for top-left, top-right and bottom-left is 0.01, 0.02 and 0.03 respectively. Histogram of variance ν is displayed as well and is capped at $\nu_i \leq 0.09$ for visualisation.

value of t_h. So we explore the distribution of the variance to find a relation with the thresholding value t_h. In both figures, histograms for variance are displayed (bottom-right of both figures). We wish to use this information to select the thresholding value.

In Figure 7.8, we can see that by increasing the value of t_h, we are able to separate feature and non-feature areas. We can see that choosing $t_h = 0.015$ corresponds to the part of the graph where the height of the histogram bar starts

getting smaller by comparison. We assumed that most of the area is non-feature. Thus, looking at the pattern of the histogram, we may choose a threshold value at the start of the 'tail' of the distribution. A similar case is shown in Figure 7.9 where $t_h = 0.03$ seems to be a good choice for thresholding value.

Unfortunately, at this point, we rely on the observation of the histogram in order to choose a suitable value. But we believe that the histogram could be a good guide in finding the value automatically. This is possibility for a future research.

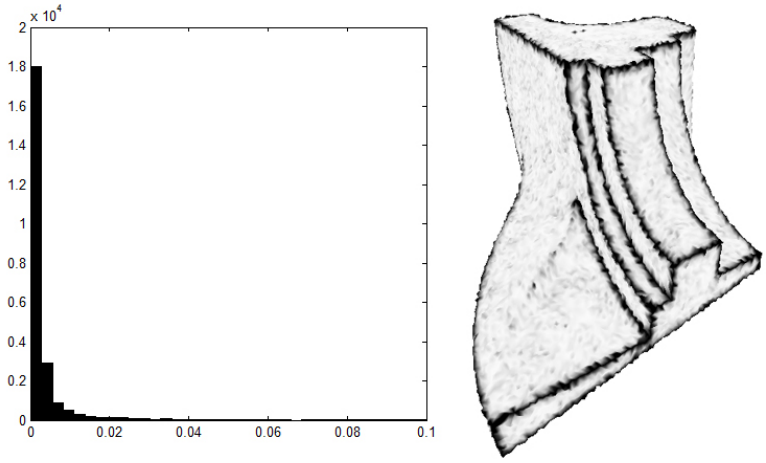

Figure 7.10: Variance mapping of Fandisk 0.5 noise level. Thresholding value is 0.02. Histogram of variance ν is displayed as well and is capped at $\nu_i \leq 0.1$ for visualisation.

Figure 7.10 shows the variance mapping of Fandisk with 0.5 noise level. We choose 0.02 as a thresholding value from the observation of histogram and display the variance mapping in the figure to the left. We can see that the feature area is well separated from the non-feature area.

The limitation of this approach is when the noise level is high. The value of variance on non-feature areas may be increased due to the noise level. Thus,

often the thresholding leaves some parts on non-feature areas to be marked as shown in Figure 7.9 (bottom-left). We believe, however, that even for 1.0 noise level, which is considerably high, feature area are still marked nicely.

7.3 Summary

This chapter describe two different methods of feature detection: an algorithm based on distance function giving the distance between a point and its nearest feature, and thresholding on variance values, which is an extension from Chapter 5. The end-product of each implementation is a set of points marked to be feature or non-feature areas.

We have shown that thresholding variance values as feature detection provides a reliable result, even on a noisy model. In this thesis, we often separate feature and non-feature areas for the purpose of verification, as when testing if our methods could handle different characteristics of a model.

In many practical applications, the detected features should be lines. Candidate methods for such an improvement are line growing approaches [25] and snakes [59; 62]. As feature detection and extraction are interdependent problems, we believe that any further improvement should be specific to the feature extraction method that will be used.

Chapter 8

Conclusion

We have presented bootstrap surface reconstruction. Throughout Chapters 4 to 7, we have covered topics regarding error estimates, normal estimation, smoothing, and feature detection.

8.1 Contributions

The main contribution of this thesis is a applying statistical method called *bootstrap* to provide a quantitative measure of quality for procedures in surface reconstruction. While the bootstrap method has been around for quite some time, it had not been applied in this particular area. While a few, including [16; 49] have discussed it in regards to the 2-dimensional problem, our bootstrap procedure has been applied to 3-dimensional data sets. In our approach, we reduce a 3D space to a 2.5D and recognize factors that may influence our estimation. For instance, the existence of feature area had affected our estimates of noise on a certain area, but in return revealed features that may or may not be shown by

visual observation.

We demonstrate bootstrap error estimates in Chapter 4, which emphasizes test error estimation for a fitting on point sets. In Chapter 5, we discuss the quality of fitting on normal estimation using variance and error estimates. While the basis of our methodology is bootstrapping, we have shown in those two chapters that different approaches should be used according to the property of the data. For instance, for normal estimation, variance property can signify the quality of normal estimates, while for the purpose of denoising, error estimates can offer more information about the fitting and the data itself.

Other than that, we have also made several other contributions:

1. In Section 6.2, we show how the value of error estimates directly relate to the quality of fitting. This is to show that the right error estimates (which is test error instead of training error) could be a legitimate valuation for quality of a fitting.

2. We then introduce a more advanced denoising step in Chapter 6. We make use of the error estimates as our confidence level of our approximation and run a smoothing procedure on the point sets.

3. In Section 5.3.1, we demonstrate a smoothing procedure on normal values which gave us a good result on and around feature area.

4. In Chapter 7, we present two different ways to detect features on point clouds. The method can also handle a considerable amount of noise.

8.2 Scope of the research

While we had presented some applications as noted in Section 8.1, a bootstrap estimate can be adapted to a post-processing method in surface reconstruction. In Equation 4.3, while we used polynomial surface fitting as our function $f(x)$ that estimates the points, we can also use any other functions as we wish.

The same goes for our normal estimation as described in Chapter 5. As the bootstrapping on variance had only required the normal value after the estimation (as shown in Equation 5.2), one may also choose a different approach in estimating normal values.

This is to reiterate that the bootstrap methodology can be generalised and used as a tool rather than being restricted to the particular approach which we have shown in this thesis.

While implying that the bootstrap method can be used for other approaches, we admit that we have only tested bootstrapping on polynomial surface fitting. We have used linear, quadratic, cubic, quartic and occasional quadric along our test.

While it is obviously not possible to try every different function available, our choice of limiting our work to only polynomial surface functions is due to the following reasons:

- Polynomial surface is one of the common functions used in the area of surface reconstruction. In normals computed using Principal Component Analysis (PCA), the largest eigen vector of the covariance matrix is similarly a linear plane constructed from least square minimization (which is equivalent to the first order of polynomial surface). MLS surface also used

polynomial surface in its projection.

- While quantitative assessment on surface reconstruction has only surfaced recently, and bootstrapping which provides test error estimation has not been used in this area, one of our areas of focus is on the verification of our procedure, and to show that the error that we estimated was test error and not training error. Thus, choosing a polynomial surface fits in nicely as its property can be well-understood in our research. For instance, a higher order of polynomial fitting may result in overfitting which can reduce the training error. Understanding this attribute allow us to observe our result, compare and understand how bootstrapping results in different degrees of polynomials.

8.3 Limitations

After using the bootstrap method, we had found few limitations of the method:-

- As the bootstrap procedure requires random sampling for B number of times, and we must repeat it for every single point, the procedure is generally slow.

- It is clear that error estimates from bootstrapping are unable to differentiate between noise and feature area. However, an error value gives the quality estimates of a function $f(z)$ on the data z's. If w is an average error value on a non-feature area, and we are using a function f such that $f(z) \leq w$, we will not be able to detect the feature area. However, we should note that if that was the case, it means that we have found a good surface representation

for our data.

To reiterate, if we are seeking the source of bad fitting, an error value may not differentiate noise in the data or high value due to feature area. However, how we make use of it is what matters.

8.4 Future work

In this thesis, we have defined an error estimation method and used the given information in estimating better normals and point fitting. There are a few improvement which could be made for future research.

As run-time is not a factor in our research, our algorithms and codings are not optimised for speed. Currently we recognise two factors which slow down the run-time. Firstly, the programming language that we used is Matlab, which is relatively slower than other programming languages like C++. Secondly, there are numerous nested loops due to a straightforward implementation of the formula such as bootstrapping. A tweaking of programming may provide us a faster algorithm.

We are also interested to use functions other than polynomial surface fittings as our estimator. In particular, spline surfaces may give a better approximation for certain area with features and sharp edges. The sharp corners of a CAD object may be better reconstructed with such function.

Finally, there are numerous other statistical methods which can be applied in our case. Other than the Bayesian Approach as used by [90], there are other model averaging methods such as Bagging and Boosting, which can be integrated into this field.

Throughout this thesis, we have shown how statistics and surface reconstruction can go side by side, and there is huge potential for research in merging these two fields together.

Appendix A

Appdx A

A.1 Normal in polynomial surface fitting

The height function $f(x, y)$ for different degree M is given as follow:-

For quadratic,

$$
\begin{aligned}
M &= 2, \\
f(x, y) &= c_1 x^2 + c_2 xy + c_3 x + c_4 y^2 + c_5 y + c_6. \\
\frac{\partial f}{\partial x} &= 2c_1 x + c_2 y + c_3, \\
\frac{\partial f}{\partial y} &= c_2 y + 2c_4 y + c_5.
\end{aligned}
$$

For cubic,

$$M = 3,$$

$$f(x,y) = c_1x^3 + c_2x^2y + c_3x^2 + c_4xy^2 + c_5xy + c_6x + c_7y^3 + c_8y^2 + c_9y + c_{10}.$$

$$\frac{\partial f}{\partial x} = 3c_1x^2 + 2c_2xy + 2c_3x + c_4y^2 + c_5y + c_6.$$

$$\frac{\partial f}{\partial y} = c_2x^2 + 2c_4xy + c_5x + 3c_7y^2 + 2c_8y + c_9.$$

For quartic,

$$
\begin{aligned}
M =& 4, \\
f(x,y) =& c_1x^4 + c_2x^3y + c_3x^3 + c_4x^2y^2 \\
&+ c_5x^2y + c_6x^2 + c_7xy^3 + c_8xy^2 \\
&+ c_9xy + c_{10}x + c_{11}y^4 + c_{12}y^3 \\
&+ c_{13}y^2 + c_{14}y + c_{15}. \\
\frac{\partial f}{\partial x} =& 4c_1x^3 + 3c_2x^2y + 3c_3x^2 + 2c_4xy^2 \\
&+ 2c_5xy + 2c_6x + c_7y^3 + c_8y^2 \\
&+ c_9y + c_{10} \\
\frac{\partial f}{\partial y} =& c_2x^3 + 2c_4x^2y \\
&+ c_5x^2 + 3c_7xy^2 + 2c_8xy \\
&+ c_9x + 4c_{11}y^3 + 3c_{12}y^2 \\
&+ 2c_{13}y + c_{14}.
\end{aligned}
$$

Bibliography

[1] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, **9**[1]:3–15, 2003. 15, 21, 25

[2] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *Proc. Symposium on Geometry Processing*, pages 39–48, 2007. 15

[3] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proc. Symposium on Computational Geometry*, pages 213–222, 2000. 25

[4] Nina Amenta, Marshall W. Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH*, pages 415–421, 1998. 24

[5] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Comput. Geom.*, **19**[2-3]:127–153, 2001. 24

[6] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Trans. Graph.*, **23**[3]:264–270, 2004. 21

[7] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH 95, Conference Proceedings*, pages 109–118, 1995. 24

[8] Cagatay Basdogan and A. Cengiz Oztireli. A new feature-based method for robust and efficient rigid-body registration of overlapping point clouds. *Vis. Comput.*, **24**[7]:679–688, 2008. 33

[9] J. D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *Proc. Symposium on Geometry Processing*, pages 9–18, 2003. 25

[10] Jean-Daniel Boissonnat, Olivier Devillers, and Monique Teillaud. A semidynamic construction of higher-order voronoi diagrams and its randomized analysis. *Algorithmica*, **9**[4]:329–356, 1993. 24

[11] Leo Breiman. Bagging predictors. *Machine Learning*, **24**[2]:123–140, 1996. 33

[12] Leo Breiman. Using iterated bagging to debias regressions. *Machine Learning*, **45**[3]:261–277, 2001. 33

[13] Peer-Timo Bremer and John C. Hart. A sampling theorem for mls surfaces. In *Proc. Symposium on Point-Based Graphics*, pages 47–54. IEEE, 2005. 22

[14] Hervé Brönnimann. Designing and implementing a general purpose halfedge data structure. In *Algorithm Engineering*, pages 51–66, 2001. 13

[15] Kishore Bubna and Charles V. Stewart. Model selection techniques and merging rules for range data segmentation algorithms. *Computer Vision and Image Understanding*, **80**[2]:215–245, 2000. 30

[16] Javier Cabrera and Peter Meer. Unbiased estimation of ellipses by bootstrapping. *IEEE Trans. Pattern Anal. Mach. Intell.*, **18**[7]:752–756, 1996. 32, 134

[17] F. Calakli and Gabriel Taubin. Ssd: Smooth signed distance surface reconstruction. *Comput. Graph. Forum*, **30**[7]:1993–2002, 2011. 24

[18] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. Richard Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*, pages 67–76, 2001. 24, 26

[19] Will Chang and Matthias Zwicker. Global registration of dynamic range scans for articulated model reconstruction. *ACM Trans. Graph.*, **30**[3]:26, 2011. 4

[20] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, **33**[3]:273–321, 2001. 13

[21] Chun-Yen Chen and Kuo-Young Cheng. A sharpness-dependent filter for recovering sharp features in repaired 3d mesh models. *IEEE Trans. Vis. Comput. Graph.*, **14**[1]:200–212, 2008. 27

[22] Antoni Chica. Visibility-based feature extraction from discrete models. In *Proc. Symposium on Solid and Physical Modeling*, pages 347–352. ACM, 2008. 34

[23] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. Technical report, Paris, France, France, 1996. 20

[24] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Trans. Vis. Comput. Graph.*, **16**[4]:599–608, 2010. 13

[25] Joel II Daniels, Linh K. Ha, Tilo Ochotta, and Claudio T. Silva. Robust smooth feature extraction from point clouds. In *Proc. Shape Modeling and Applications*, pages 123–136, Washington, DC, USA, 2007. IEEE. 34, 133

[26] Kris Demarsin, Denis Vanderstraeten, Tim Volodine, and Dirk Roose. Detection of closed sharp edges in point clouds using normal estimation and graph theory. *Comput. Aided Des.*, **39**[4]:276–283, 2007. 34

[27] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH*, pages 317–324, 1999. 18

[28] T. K. Dey, G. Li, and J. Sun. Normal estimation for point clouds : A comparison study for a voronoi based method. In *Proceeding of SoPBG*, pages 39–46, 2005. 16

[29] Tamal K. Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *Proc. Symposium on Solid Modeling and Applications*, pages 127–134, 2003. 25

[30] Tamal K. Dey and Jian Sun. Normal and feature approximations from noisy point clouds. In *FSTTCS*, pages 21–32, 2006. 15

[31] Huong Quynh Dinh, Greg Turk, and Gregory G. Slabaugh. Reconstructing surfaces using anisotropic basis functions. In *ICCV*, pages 606–613, 2001. 27

[32] I. Eckstein, Y. Tong, C.-C. J. Kuo, and M. Desbrun. Volume-controlled surface fairing. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 8, New York, NY, USA, 2007. ACM. 19

[33] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, **7**[1]:1–26, 1979. 32

[34] B. Efron. Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation. *Journal of the American Statistical Association*, **78**[382]:316–331, 1983. 48

[35] Bradley Efron and Robert Tibshirani. Improvements on Cross-Validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, **92**[438]:548–560, June 1997.

[36] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, **24**[3]:544–552, 2005. 27

[37] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, **22**[3]:950–953, 2003. 19

[38] James Foley, Andries van Dam, Steven Feiner, John Hughes, and Richard Phillips. *Introduction to Computer Graphics*. Addison-Wesley Professional, 1993. 43

[39] Jonathan Freeman and Jane Lessiter. Evaluation of partially sighted people's viewing experiences of 3d relative to 2d tv. Technical report, 2010. 2

[40] Bernd Fritzke. Growing cell structures–a self-organizing network for unsupervised and supervised learning. *Neural Networks*, **7**[9]:1441–1460, 1994. 30

[41] Ran Gal, Ariel Shamir, Tal Hassner, Mark Pauly, and Daniel Cohen-Or. Surface reconstruction using local shape priors. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 253–262, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. 27

[42] T. Gatzke and C. Grimm. Feature detection using curvature maps and the min-cut/max-flow algorithm. In *Proc. Geometric Modeling and Processing*, pages 578–584, 2006. 33

[43] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Comput.*, **4**[1]:1–58, 1992. 40

[44] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized Delaunay triangulation. *Computer Graphics Forum*, **19**[3]:467–478, 2000. 16

[45] J. P. Grossman and William J. Dally. Point sample rendering. In *Rendering Techniques*, pages 181–192, 1998. 13

[46] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Transactions on Graphics*, **26**[3]:23, 2007. 25

[47] Stefan Gumhold, Xinlong Wang, and Rob Macleod. Feature extraction from point clouds. In *Proc. International Meshing Roundtable*, pages 293–305, 2001. 34

[48] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 20

[49] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, August 2001. x, 33, 40, 46, 55, 56, 61, 65, 67, 134

[50] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum*, **23**:391–400, 2004. 19, 120

[51] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH*, pages 71–78, 1992. 15, 16, 24, 25, 96

[52] Guofei Hu, Jie Xu, Lanfang Miao, and Qunsheng Peng. Bilateral estimation of vertex normal for point-sampled models. In *ICCSA (1)*, pages 758–768, 2005. 15, 16

[53] Aapo Hyvarinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Wiley-Interscience, May 2001. 14

[54] Ioannis P. Ivrissimtzis, Won-Ki Jeong, and Hans-Peter Seidel. Using growing cell structures for surface reconstruction. In *Shape Modeling International*, pages 78–88, 288, 2003. 30

[55] Gareth M. James. Variance and bias for general loss functions. *Mach. Learn.*, **51**[2]:115–135, 2003. 41

[56] Philipp Jenke, Michael Wand, Martin Bokeloh, Andreas Schilling, and Wolfgang Strasser. Bayesian point cloud reconstruction. *Comput. Graph. Forum*, **25**[3]:379–388, 2006. 29

[57] Thouis R. Jones, Frédo Durand, and Mathieu Desbrun. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph.*, **22**[3]:943–949, 2003. 19

[58] Thouis R. Jones, Frédo Durand, and Matthias Zwicker. Normal improvement for point rendering. *IEEE Computer Graphics and Applications*, **24**[4]:53–56, 2004. 16, 73

[59] Moonryul Jung and Haengkang Kim. Snaking across 3d meshes. In *Pacific Conference on Computer Graphics and Applications*, pages 87–93, 2004. 133

[60] Aravind Kalaiah and Amitabh Varshney. Statistical point geometry. In *Proc. Symposium on Geometry Processing*, pages 107–115, 2003. 29

[61] Aravind Kalaiah and Amitabh Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Transactions on Graphics*, **24**[2]:348–373, 2005. 29

[62] Michael Kass, Andrew P. Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, **1**[4]:321–331, 1988. 133

[63] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. 29

[64] Lutz Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom.*, **13**[1]:65–90, 1999. 13

[65] L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, **18**[3]:119–130, 1999. 25

[66] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proc. Symposium on Geometry Processing*, pages 11–21, 2004. 25

[67] Sören König and Stefan Gumhold. Consistent propagation of normal orientations in point clouds. In *VMV*, pages 83–92, 2009. 43

[68] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96, Conference Proceedings*, pages 313–324, 1996. 24, 28

[69] Carsten Lange and Konrad Polthier. Anisotropic smoothing of point sets. *Computer Aided Geometric Design*, **22**[7]:680–692, 2005. 16, 28

[70] Yunjin Lee, Seungyong Lee, Ioannis P. Ivrissimtzis, and Hans-Peter Seidel. Overfitting control for surface reconstruction. In *Symposium on Geometry Processing*, pages 231–234, 2006. 31, 32

[71] David Levin. The approximation power of moving least-squares. *Math. Comput.*, **67**[224]:1517–1531, 1998. 22, 25

[72] Bao Li, Ruwen Schnabel, Reinhard Klein, Zhiquan Cheng, Gang Dang, and Shiyao Jin. Robust normal estimation for point clouds with sharp features. *Computers & Graphics*, **34**[2]:94–106, 2010. 16

[73] Xinju Li and Igor Guskov. Multi-scale features for approximate alignment of point-based surfaces. In *Proc. Symposium on Geometry Processing*, page 217. Eurographics Association, 2005. 34

[74] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics*, **26**[3]:22, 2007. (Proc. SIGGRAPH 2007). 22, 25

[75] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, pages 163–169, 1987. 25

[76] Kwan-Liu Ma and Victoria Interrante. Extracting feature lines from 3d unstructured grids. In *Proc. IEEE Visualization*, pages 285–292, 1997. 33

[77] João F. Mari, José Hiroki Saito, Gustavo Poli, Marcelo R. Zorzan, and Alexandre L. M. Levada. Improving the neural meshes algorithm for 3d surface reconstruction with edge swap operations. In *SAC*, pages 1236–1240, 2008. 30

[78] Boris Mederos, Nina Amenta, Luiz Velho, and Luiz Henrique de Figueiredo. Surface reconstruction for noisy point clouds. In *Symposium on Geometry Processing*, pages 53–62, 2005. 25

[79] Quentin Mérigot, Maks Ovsjanikov, and Leonidas J. Guibas. Voronoi-based curvature and feature estimation from point clouds. *IEEE Trans. Vis. Comput. Graph.*, **17**[6]:743–756, 2011. 34

[80] N. J. Mitra, A. Nguyen, and L. Guibas. Estimating surface normals in noisy point cloud data. In *special issue of International Journal of Computational Geometry and Applications*, **14**, pages 261–276, 2004. 15, 16

[81] Bryan S. Morse, Terry S. Yoo, David T. Chen, Penny Rheingans, and Kalpathi R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modeling International*, pages 89–98, 2001. 27

[82] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 292, Washington, DC, USA, 2003. IEEE Computer Society. 26, 30

[83] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, **22**[3]:463–470, 2003. 24, 28

[84] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 3d scattered data approximation with adaptive compactly supported radial basis functions. In *Proc. Shape Modeling International*, pages 31–39. IEEE, 2004. 24, 27, 34

[85] Mark Pauly and Markus Gross. Spectral processing of point-sampled geometry. pages 379–386, 2001. 23, 29

[86] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference*

on Visualization '02, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society. 15

[87] Mark Pauly, Richard Keiser, and Markus H. Gross. Multi-scale feature extraction on point-sampled surfaces. *Comput. Graph. Forum*, **22**[3]:281–290, 2003. 122, 126

[88] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, **22**[3]:641–650, 2003. 16, 75, 128

[89] Mark Pauly, Leif P. Kobbelt, and Markus Gross. Point-based multiscale surface representation. *ACM Transactions on Graphics*, **25**[2]:177–193, 2006. 16, 28

[90] Guiping Qian, Ruofeng Tong, Wen Peng, and Jinxiang Dong. Bayesian mesh reconstruction from noisy point data. In *ICAT*, pages 819–829, 2006. 29, 30, 138

[91] Hongxing Qin, Jie Yang, and Yuemin Zhu. Nonuniform bilateral filtering for point sets and surface attributes. *The Visual Computer*, **24**[12]:1067–1074, 2008. 16, 23, 28

[92] O. Schall, A. Belyaev, and H.-P. Seidel. Robust filtering of noisy scattered point data. *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics*, **0**:71–144, 2005. 23

[93] Oliver Schall, Alexander Belyaev, and Hans-Peter Seidel. Feature-preserving non-local denoising of static and time-varying range

data. In *Proc. Symposium on Solid and Physical Modeling*, pages 217–222, 2007. 23

[94] Carlos E. Scheidegger, Shachar Fleishman, and Cláudio T. Silva. Triangulating point set surfaces with bounded error. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 63, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. 23

[95] Xianfang Sun, Paul L. Rosin, Ralph R. Martin, and Frank C. Langbein. Fast and effective feature-preserving mesh denoising. *IEEE Trans. Vis. Comput. Graph.*, **13**[5]:925–938, 2007. 17, 20

[96] Xianfang Sun, Paul L. Rosin, Ralph R. Martin, and Frank C. Langbein. Noise analysis and synthesis for 3d laser depth scanners. *Graphical Models*, **71**[2]:34–48, 2009. 7

[97] Yiyong Sun, Joon Ki Paik, Andreas Koschan, David L. Page, and Mongi A. Abidi. Triangle mesh-based edge detection and its application to surface segmentation and adaptive surface smoothing. In *ICIP (3)*, pages 825–828, 2002. 34

[98] Andrea Tagliasacchi, Matt Olson, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. Vase: Volume-aware surface evolution for surface reconstruction from incomplete point clouds. *Comput. Graph. Forum*, **30**[5]:1563–1571, 2011. 27

[99] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. Curve skeleton extraction from incomplete point cloud. *ACM Trans. Graph.*, **28**[3], 2009. 27

[100] Gabriel Taubin. A signal processing approach to fair surface design. In *SIGGRAPH*, pages 351–358, 1995. 18

[101] J. Vollmer, R. Mencl, and H. Mller. Improved laplacian smoothing of noisy surface meshes. In *Computer Graphics Forum*, pages 131–138, 1999. 18

[102] Charlie C. L. Wang. Bilateral recovering of sharp edges on feature-insensitive sampled meshes. *IEEE Trans. Vis. Comput. Graph.*, **12**[4]:629–639, 2006. 34

[103] Tim Weyrich, Mark Pauly, Simon Heinzle, Richard Keiser, Sascha Scandella, and Markus Gross. Post-processing of scanned 3d surface data. In *Symposium on Point-Based Graphics*, pages 85–94, 2004. 6

[104] Chun-Xia Xiao. Multi-level partition of unity algebraic point set surfaces. *J. Comput. Sci. Technol.*, **26**[2]:229–238, 2011. 28

[105] Chunxia Xiao, Wenting Zheng, Yongwei Miao, Yong Zhao, and Qunsheng Peng. A unified method for appearance and geometry completion of point set surfaces. *The Visual Computer*, **23**[6]:433–443, 2007. 27

[106] Hui Xie, Kevin T. McDonnell, and Hong Qin. Surface reconstruction of noisy and defective data sets. In *IEEE Visualization*, pages 259–266, 2004. 27

[107] Yong-Liang Yang, Yu-Kun Lai, Shi-Min Hu, and Helmut Pottmann. Robust principal curvatures on multiple scales. In *Proc. Symposium on Geometry Processing*, pages 223–226. Eurographics Association, 2006. 33

[108] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. 13

[109] Mincheol Yoon, Ioannis Ivrissimtzis, and Seungyong Lee. Variational bayesian noise estimation of point sets. *Computers & Graphics*, **33**[3]:226–234, 2009. 29, 57, 61

[110] Mincheol Yoon, Yunjin Lee, Seungyong Lee, Ioannis Ivrissimtzis, and Hans-Peter Seidel. Surface and normal ensembles for surface reconstruction. *Comput. Aided Des.*, **39**[5]:408–420, 2007. 30, 32

[111] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. Fast and robust detection of crest lines on meshes. In *Proc. Symposium on Solid and physical modeling*, pages 227–232, New York, NY, USA, 2005. ACM. 34

[112] Lei Zhang, Ligang Liu, Craig Gotsman, and Hua Huang. Mesh reconstruction by meshless denoising and parameterization. *Computers & Graphics*, **34**[3]:198–208, 2010. 16

[113] Youyi Zheng, Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. Bilateral normal filtering for mesh denoising. *IEEE Trans. Vis. Comput. Graph.*, **17**[10]:1521–1530, 2011. 16

[114] Kun Zhou, Minmin Gong, Xin Huang, and Baining Guo. Data-parallel octrees for surface reconstruction. *IEEE Trans. Vis. Comput. Graph.*, **17**[5]:669–681, 2011. 24