Regular Grids: An Irregular Approach to the 3D Modelling Pipeline

DAVID PAUL KAYE

ABSTRACT

The 3D modelling pipeline covers the process by which a physical object is scanned to create a set of points that lay on its surface. These data are then cleaned to remove outliers or noise, and the points are reconstructed into a digital representation of the original object.

The aim of this thesis is to present novel grid-based methods and provide several case studies of areas in the 3D modelling pipeline in which they may be effectively put to use.

The first is a demonstration of how using a grid can allow a significant reduction in memory required to perform the reconstruction. The second is the detection of surface features (ridges, peaks, troughs, etc.) during the surface reconstruction process.

The third contribution is the alignment of two meshes with zero prior knowledge. This is particularly suited to aligning two related, but not identical, models. The final contribution is the comparison of two similar meshes with support for both qualitative and quantitative outputs.

Regular Grids: An Irregular Approach to the 3D Modelling Pipeline

David Paul Kaye

A thesis submitted towards the degree of

Doctor of Philosophy.

School of Engineering and Computing Sciences
Durham University
United Kingdom

September 2016

CONTENTS

LIST OF TABLES

LIST OF FIGURES

DECLARATION

The work in this thesis is based on research carried out in the Innovative Computing Group, School of Engineering and Computing Sciences, Durham University. No part of this report has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Parts of this work have been published in the following conference proceedings:

- D. Kaye and I. Ivrissimtzis, Implicit Surface Reconstruction and Feature Detection with a Learning Algorithm. *Theory and Practice of Computer Graphics, Eurographics Association*, 2010, 127–130.
- D. P. Kaye and I. Ivrissimtzis, Memory Efficient Surface Reconstruction Based on Self Organising Maps. *Theory and Practice of Computer Graphics, Eurographics Association*, 2011, 25–32.
- David Kaye and Ioannis Ivrissimtzis, Mesh Alignment Using Grid Based PCA. *GRAPP, Springer.* 2015, 64–76.

COPYRIGHT

ACKNOWLEDGEMENTS

Balancing a project of this magnitude with a job is a difficult task, and without the understanding and guidance of my supervisor, Ioannis Ivrissimtzis, I would not have been able to complete this work. Not only has he provided direction and focus to what would otherwise be a disorganised (and likely incomplete) set of experiments, he has done so in his own time in order to ensure I have not had to work in isolation.

The help and support from my friends Emma Lyle and Juliet Pouteaux, have gone above and beyond what I could ever have hoped for, offering sympathetic ears and kindness even in the most trying of circumstances. Your kindness humbles me.

My intellectual sparring partner, Guy Hutchinson, helped me find the source of and solution to innumerable problems, often with no more than a well placed "Are you sure?" as I explained my current predicament.

There are countless others to whom I owe my gratitude, including Richard Parker, Tom Francis, Sarah and Stephen Howell, and Rebecca Papworth, for their understanding, advice, and consideration, particularly when times were tough and deadlines were tight.

Finally, I would like to offer my deepest thanks to my parents, Arthur and Pauline Kaye, for their unconditional love and timely reminders that I could be doing some work right now. Were it not for their endless patience and (appropriately questioning) support for all my endeavours, I would not be where I am today.

1. INTRODUCTION

The 3D modelling pipeline is the term used to describe the various stages and algorithms through which data must pass on their route from a physical object to a representation of that object on a computer screen. The data could be geometric in nature, describing the size and shape of the object; topological, describing fundamental properties of the object and its self-connectivity; or visual, pertaining to its colour or appearance (e.g material, reflectance).

A typical route for creating a 3D model from a physical object would be to scan the object using a laser, which would generate a set of points in 3D space (a *point cloud*), potentially alongside other data such as colour and reflectance. Since the scanning is usually done in several stages, resulting from rotating the object or moving the scanner, the data must be transformed to bring them into a consistent coordinate system.

This point cloud is then processed to remove any noise and spurious data that could have arisen during the scan. After this, the clean point cloud is then used as input for an algorithm that generates a representation of the 3D surface describing the boundary of the object. The representation of the object's boundary may not be optimal for the intended purpose; it can therefore be smoothed or have additional data attached to it (e.g. colour or a quality measure) as part of a post-processing stage.

The final stage is to actually render the object on-screen; this is generally done by creating a light source and mapping reflections, but there are other techniques that can be used depending on the desired result. Figure 1.1 shows a simplified overview of whole process.

This thesis will focus on creating the surface representation (chapters 5 and 4) and on comparing and aligning meshes as part of a post-processing stage (chapters 6 and 7).

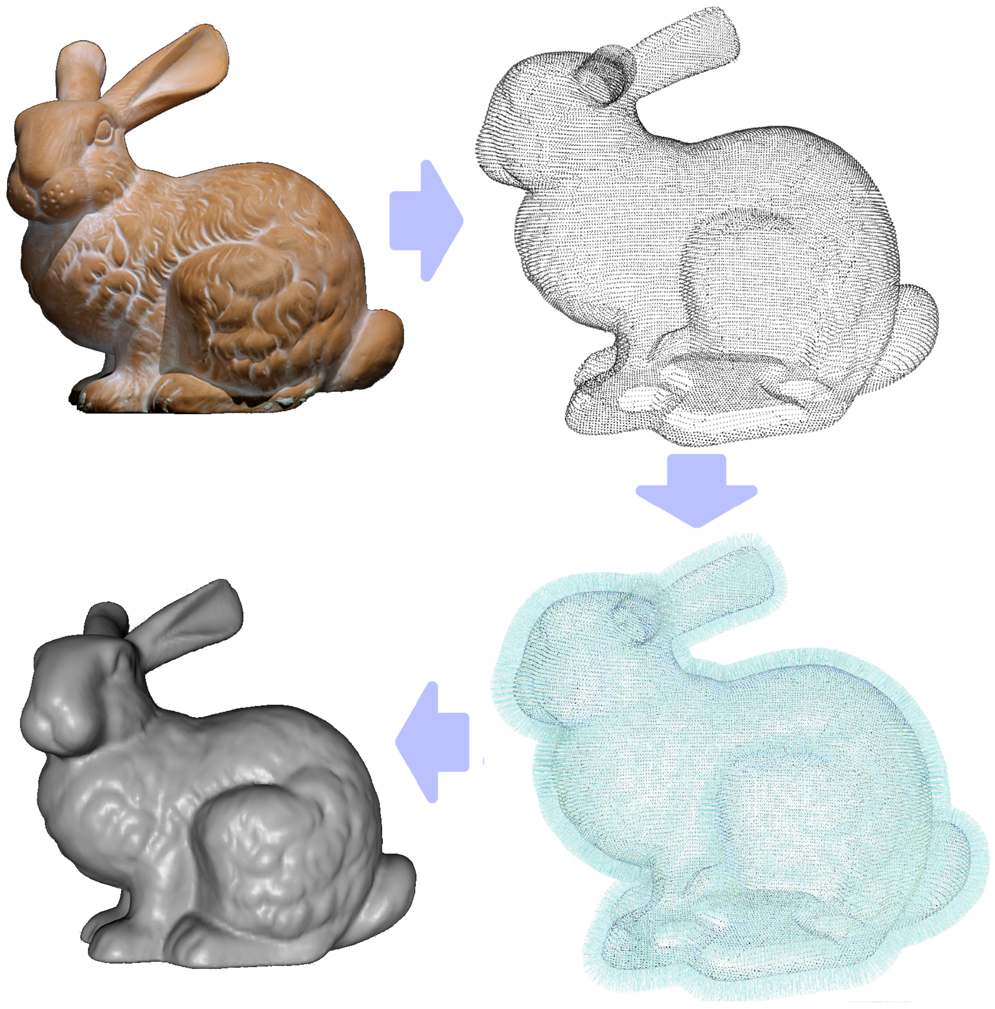

Fig. 1.1: *Top left:* physical object to be modelled.
Top right: point cloud produced by scanning the object.
Bottom right: normals estimated from the point cloud.
Bottom left: final model after surface reconstruction.

1.1 Background

The acquisition of 3D object data and the creation of models from these data has a wide range of applications, from understanding our cultural heritage and the interpretation of medical data to military terrain scanning and civil engineering.

For example, in a medical context, a patient awaiting brain surgery will undergo a CT scan, and a 3D representation of the brain will be created to allow the surgeon to plan the operation in advance. For industrial situations, one example of its use is viewing and analysing internal fractures in solids.

An engineer wanting to investigate how a bridge would react under particular weather conditions would likely use a mesh of the bridge as a starting point for Finite Element Analysis. To model the bridge's behaviour, the engineer would define material properties of the mesh, and approximate the physical conditions to be modelled using linear equations. These linear equations would then be solved for each face of the mesh (the finite elements of the name), and combined to form a global solution that accurately reflects the bridge's behaviour.

Understanding our cultural heritage is not only an academic pursuit; there are also non-profit organisations such as CyArk, whose aim is to create a library of freely available models of artefacts found at cultural heritage sites. The sooner and more accurately this can be done, the smaller the risk of damage and loss of irreplaceable information about our past. The digital representations would then be a form of preservation for these artefacts, made available online for future generations (in its own words) "before they are lost to natural disasters, destroyed by human aggression or ravaged by the passage of time."

1.2 Key Concepts

In this section, concepts fundamental to the thesis are presented alongside a high-level explanation.

Self Organising Maps are the basis for all algorithms presented in this thesis. We use a specialised form that allows for the detection of surface features as part of the reconstruction (discussed in chapter 5). It also allows for a considerable reduction in memory footprint, allowing modest hardware to process huge amounts of data (chapter 4).

In chapters 5 and 4, a surface representation must be extracted from the Self Organising Map, and for this we use the Marching Cubes algorithm. Our variant is adjusted to include information on any surface features that have been detected during the reconstruction.

In chapter 6, we present an algorithm to align two meshes and validate it against the common alternative of Principal Components Analysis. The Iterative Closest Point algorithm is the most widely used mesh alignment algorithm. It does not directly compare with the algorithm presented in chapter 6, since it places restrictions on the input data, but is discussed here to give context.

1.2.1 Self Organising Maps

An artificial neural network comprises several layers of nodes. Within each layer, each node performs some small amount of processing, which influences the nodes to which it provides input.

For instance, Figure 1.2 shows a simple artificial neural network divided into three layers. Data passes in from the left and the three layers process it from left to right, with the rightmost layer providing the final output. This architecture is based on *supervised learning*, where an updates the network depending on whether the processing was deemed successful.

A Self Organising Map (SOM) is special case of an artificial neural network that adapts itself via *unsupervised learning*. This is essentially the network training itself, and as such it does not have layers in the sense of the networks described above.

Figure 1.3 shows an example of an SOM (the black grid) adapting to some input data (the blue cloud). In this situation, a point from the blue cloud is selected at random, and the closest node in the SOM moved towards that data point. Neighbouring nodes are also moved towards that point, but to a lesser extent, and the procedure is repeated many times. SOMs work by minimising a so-called “energy function”, in this case, what is being minimised is the median distance from a point in the blue cloud to the SOM itself.

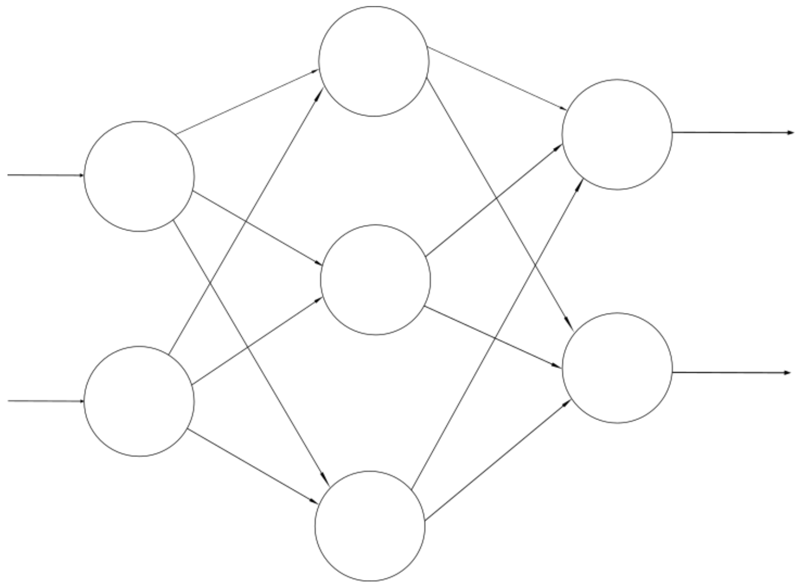

Fig. 1.2: A simple artificial neural network: input is provided on the left, and the processing flows from left to right. The final output is given by the two nodes in the rightmost layer.

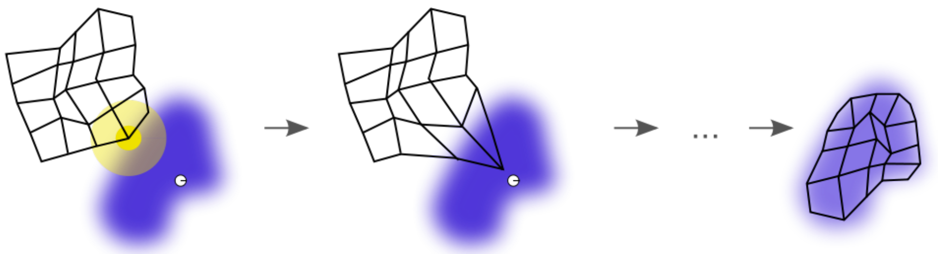

Fig. 1.3: Example of a 2D SOM adapting to training data (©Dan Stowell).

It is common to have a global dampening such that, with each iteration, the amount that the closest node moves is slightly diminished. This leads to an increase in stability when, after many iterations, the movement of each node is negligible, or at least within acceptable limits. At this stage, the SOM can be used either to approximate the data directly, or it can be provided as input for another application or algorithm.

1.2.2 Marching Cubes Algorithm

The Marching Cubes algorithm is used to create a 3D representation of an object. The space containing the surface is divided into a regular 3D grid, and each corner within that grid is evaluated to determine its shortest

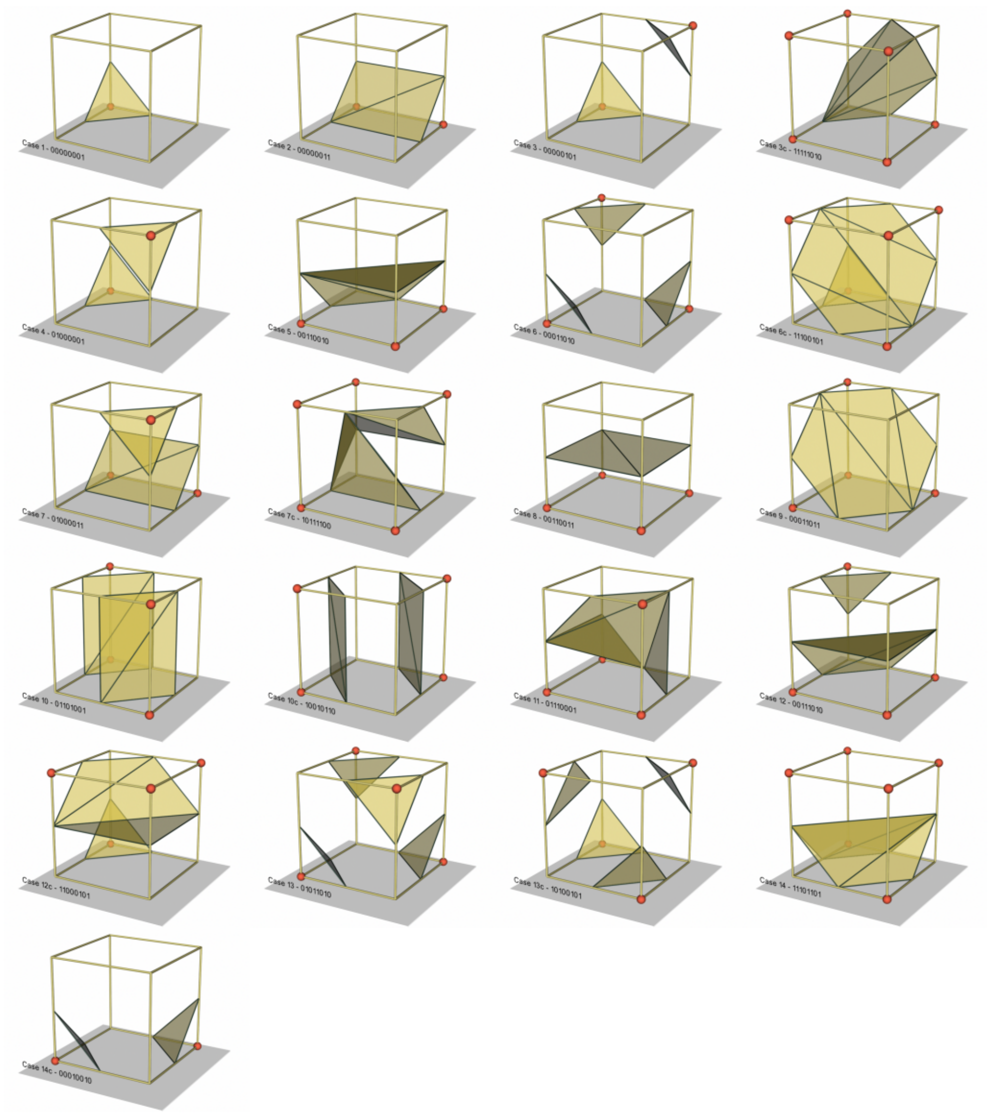

Fig. 1.4: Marching Cubes: the different cube configurations.

distance to the surface (negative values indicating that the corner lies inside the surface.

If the function is outside the surface at one corner, and inside at another, it must be the case that the surface passes between them. Since each corner can be either inside or outside the surface (ignoring the infinitesimal chance of it lying precisely on the surface), there are $2^8 = 256$ possible combinations of corners being inside/outside the surface.

When various symmetries (e.g. rotational) are taken into account, the number of unique combinations is reduced to just 15, which are shown (with some duplications for clarity) in Figure 1.4. After reconstructing

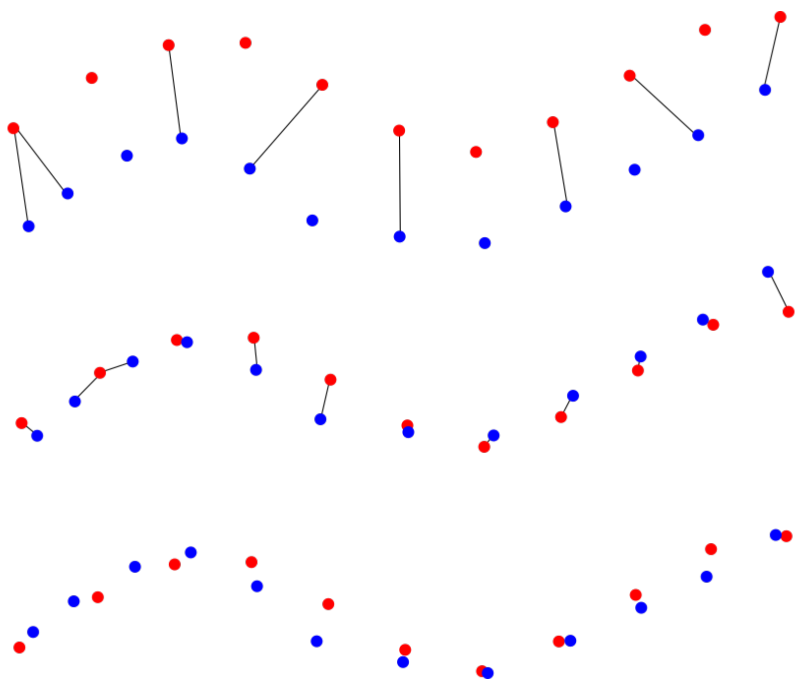
Fig. 1.5: Iterative Closest Point algorithm applied to a series of points.

the surface within each smaller cube, the results can be stitched together to provide a global reconstruction of the surface.

1.2.3 Iterative Closest Point Algorithm

The Iterative Closest Point algorithm can align two point clouds, two meshes, or a combination of the two. It is widely used, but requires an "adequate" initial guess in order to provide good alignment, otherwise it can get stuck and report success in cases of poor alignment.

First, points in one set of input data are matched to the closest point in the other. Then, the error for this initial alignment is computed, based on the mean squared distance between all pairs of points. One set is chosen to be the reference set, the other (the source) is translated and rotated in order to bring it into better alignment with the reference set (an alignment is deemed better if it reduces the alignment error computed earlier). The new errors are computed, new sets of points are matched, and the procedure is repeated until the computed error is stable.

Figure 1.5 shows how the ICP algorithm might work on two sets of points.

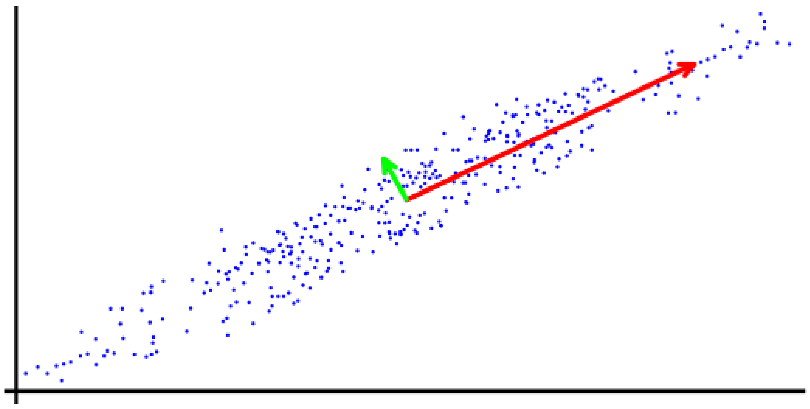

Fig. 1.6: Principal components of a set of data.

1.2.4 Principal Components Analysis

Principal Components Analysis is a method by which the directions of greatest variation in a set of data can be determined. These directions are known as the principal components, and can be used to understand the character of the data. For instance, when representing 3D data in two dimensions, taking the largest two principal components as the axes will ensure that the representation is as expressive as possible.

Figure 1.6 shows the two principal components of a set of data. It is clear that, in order to represent the data as expressively as possible in one dimension, we should describe them in terms of their distance along the red arrow (the largest principal component).

1.3 Motivation

In spite of the relevance and range of applications, the 3D modelling pipeline suffers from several long-standing issues. Among these are the resources for processing/creating highly detailed models, the computational resources required for searching for nearby points, and the complexity of some of the data structures that must be employed. Many of the stages involve finding solutions to problems that are *ill-posed*, meaning that a general solution for all cases (an *analytic* solution) does not exist. However, different circumstances can benefit from different solutions, so advances can still be made.

Algorithms that exploit regular 3D grids (*lattices*) have the potential to improve several stages of the pipeline. The regular structure of a lattice allows for nodes close to another to be found in constant time using simple arithmetic operations on that node's index. By contrast, an algorithm that uses nodes whose position can change over time must query the position of all nodes before determining proximity. This is computationally expensive, and, if nodes are added over the course of the algorithm, cannot be done in constant time. The simple, regular nature of a lattice also allows for algorithms that are simple, yet potentially effective.

1.4 Objectives

I propose that the regular structure of 3D grids allows simple algorithms to be developed that solve or mitigate otherwise challenging problems in the 3D modelling pipeline. There are three principle areas in which I will investigate their utility.

1.4.1 Memory Efficiency

The regular layout of nodes in a 3D grid naturally suggests subdivision for processing subsets of the input.

The accuracy of 3D scanning methods is ever-increasing, and with that increased fidelity comes a corresponding increase in the hardware required to process these data. Keeping up with ever-increasing requirements can be costly, but not doing so can lock researchers and interested parties out of working on large datasets.

Therefore, if a SOM could be trained one section at a time, the minimal hardware requirements for working with large amounts of data would drop. Taking a layer-by-layer approach to training approach is intuitive, and indeed fits very well with the way data are produced by some devices (such as CT scanners), making it natural to pair the two. Correspondingly, the number of people and institutions able to work on such data would increase, potentially by a large amount.

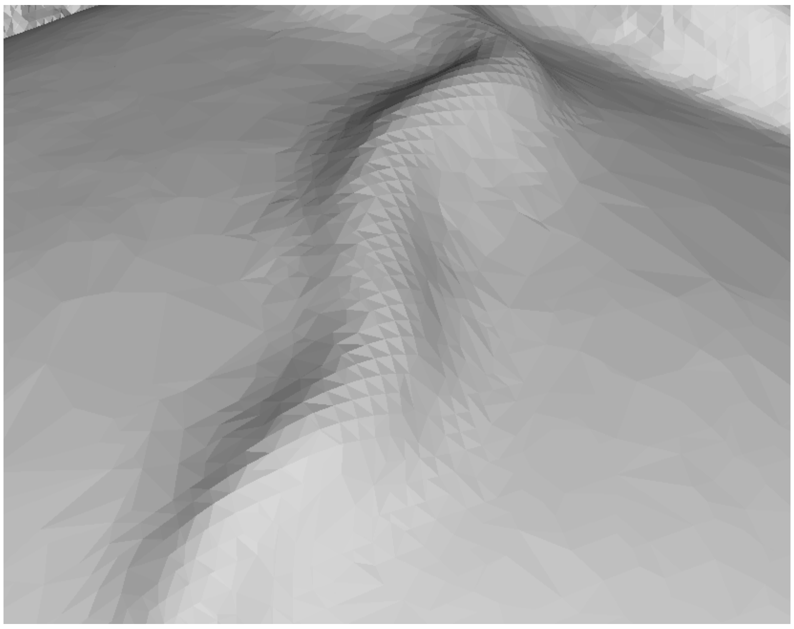

Fig. 1.7: Example of a surface ridge.

1.4.2 Feature Detection

Does the training history of a Self Organising Map afford any opportunity to improve the reconstruction quality?

Current SOM methods only use the final value of the SOM's nodes, discarding all previous states. The training history can give insight into the training data. For instance, nodes that have a long training history are likely in more densely populated areas of the input cloud.

Knowledge of the point cloud density in one area could allow a measure of confidence to be assigned to those nodes' values. This could be used to determine the future execution of an algorithm, or provide additional information for post-processing.

Surface ridges, valleys, and peaks (shown in Figures 1.7, 1.8, and 1.9 respectively) are likely to show training data from different points interfering, which should become clear with a simple analysis of the training history.

1.4.3 Alignment

Current mesh alignment algorithms take one of two approaches. Either they use PCA directly on mesh vertices, or use a variant of the ICP

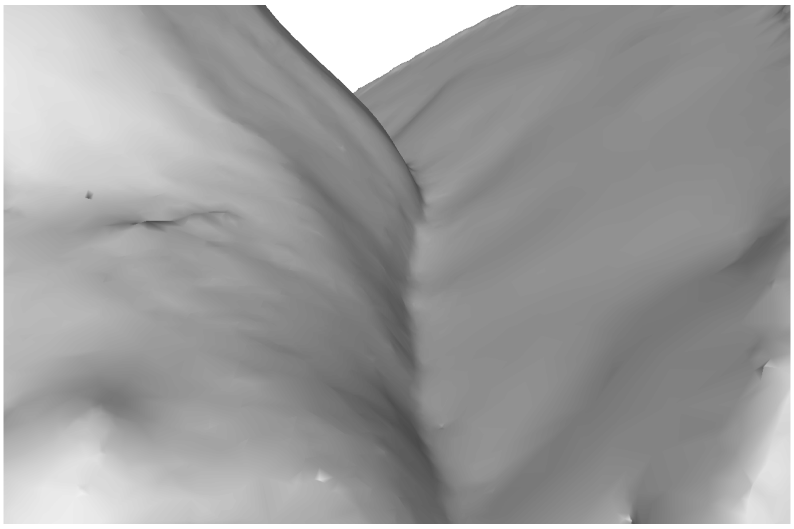

Fig. 1.8: Example of a valley.

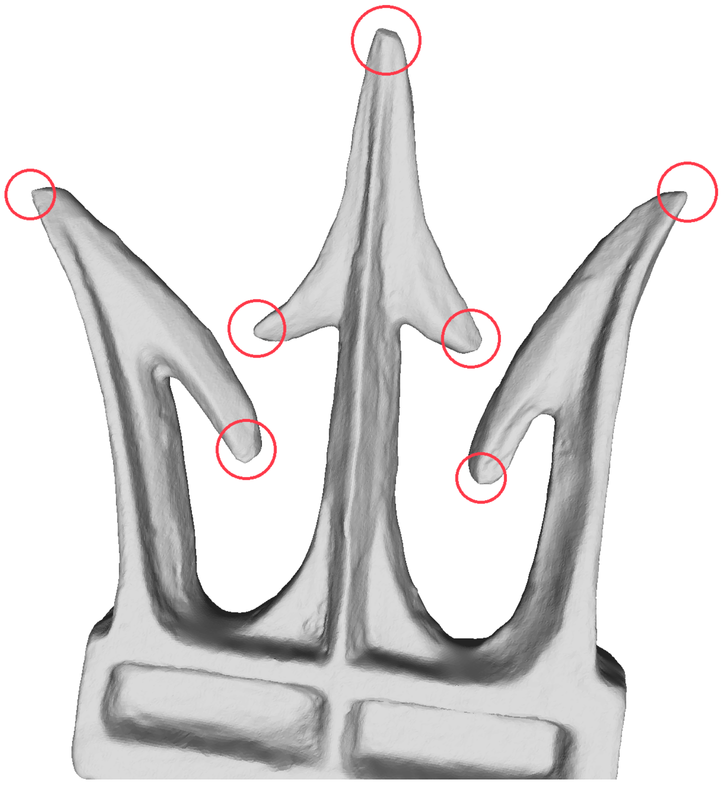

Fig. 1.9: Examples of peaks on a surface.

algorithm.

Vertex PCA inherits the existing issues with PCA, namely that it is sensitive to equivalent meshes with differing vertex distributions. As such, it is not viable for aligning similar meshes. In that context, PCA can be seen as overfitting its analysis to the specific mesh.

ICP can align two meshes, but requires that they have some vertices in common. Some variants even require one input mesh to be a subset of the other. ICP is therefore inherently unable to align two similar meshes that do not have vertices in common. This could be the case after a remeshing or if one input is a scaled form of another.

We will investigate the extent to which a 3D grid could be used to align a coarse representation of a mesh. This would provide a practical third option that avoids the pitfalls of both PCA and ICP.

1.4.4 Comparison

The comparison of meshes in most works is done by adjacent snapshots of the meshes in question, or sometimes just the points of interest. Depending on how the results are viewed, this carries the possibility of masking potentially significant differences.

Some comparison algorithms provide a single numerical output for their results (perhaps the volume of the space between the meshes). Without an idea of how the differences are distributed however, this number in isolation is not necessarily helpful when attempting to interpret the result.

Other methods are not symmetric, that is, comparing mesh A to mesh B provides a different result to comparing mesh B to mesh A – a result that is at least counter-intuitive. We will investigate the use of an SOM-based approach to mesh comparison, with the intention of providing a representative visual indication of differences (for intuitive viewing of results) alongside a solid foundation on which a number of metrics could be computed.

This would bridge the gap of both forms of comparison: non-visual but quantitative measures, and visual qualitative ones.

1.5 Research Questions

Implicit representations are frequently used for surface reconstruction but have not yet been exploited in other stages of the 3D modelling pipeline. Moreover, even in the surface reconstruction stage, the data contained within the SOM from the training process is discarded once the mesh is extracted.

In this thesis we argue that the trained grid contains a great deal of valuable information that can be exploited at several different stages of the pipeline. For instance, the information embedded in the grid from its training can be used to detect surface features and extract alignment information. Further, the regular structure also allows for easy comparison of inputs and compartmentalised processing for more efficient handling of large datasets.

Explicitly, this thesis seeks to answer the following questions.

1. Reducing the footprint of surface reconstruction.
   - 1.1 Can the structure of a regular 3D SOM be used to increase the performance of surface reconstruction?
   - 1.2 To what extent can the structure of a regular 3D SOM be exploited to work with large datasets?
2. Feature detection using Self Organising Maps.
   - 2.1 Can the training history of an SOM be used to detect surface features?
   - 2.2 How early can this be integrated in order to make the information available to more stages of the pipeline?
3. Alignment of dissimilar meshes.
   - 3.1 Under what circumstances would an SOM be suitable for aligning two meshes?
   - 3.2 To what extent would the regular structure be beneficial, and what limitations would it impose?
   - 3.3 How would such an algorithm compare to standard techniques?
4. Mesh Comparison.

4.1 Could an SOM, being external to two meshes, be trained to detect their differences?

4.2 Does SOM-based comparison offer any benefits over existing techniques?

1.6 Overview

The rest of the thesis is organised as follows. Chapter 2 presents an overview of each stage in the 3D modelling pipeline, before reviewing the algorithms used at each stage. Chapter 3 discusses implementation details (in particular, file formats) and describes the software developed as part of my research.

Subsequent chapters present research performed and the corresponding results. Chapter 4 demonstrates a modified surface reconstruction algorithm with a focus on reducing the memory footprint, such that very large meshes can be processed on modest hardware. Chapter 5 presents a surface reconstruction algorithm with integrated feature detection. Chapter 6 presents a novel method of approximately aligning two different (yet similar) meshes. Chapter 7 presents work on a method of comparing two meshes.

Finally, chapter 8 discusses the results of the preceding chapters and suggests potentially fruitful avenues for future research in each area.

1.7 Contributions

The contributions of the work presented in this thesis are as follows.

An SOM-based, memory-efficient, and scalable surface reconstruction algorithm (chapter 4). This algorithm, presented along with a simple modification to the Marching Cubes algorithm, allows for the reconstruction of very fine triangle mesh representations of large quantities of input data.

A Self Organising Map (SOM) that stores its training history and demonstrates that this history can be used for feature detection (chapter 5.

A deterministic, fully automated alignment algorithm based on performing PCA on a point sample on a regular grid (chapter 6). The algorithm can process point clouds and meshes as inputs, and can even align a point cloud with a mesh.

A comparison algorithm that can not only detect the differences between two meshes, but also meshes and point clouds, and even two point clouds. For clean inputs, the results match up well to the standard methods, which lends credence to the algorithm's validity.

1.8 Limitations

There are some limitations to the work presented in this thesis.

The algorithm presented in chapter 4 requires pre-processing the data to sort them by z-coordinate. Such sorting requires pre-determined granularity, and the optimal value is not always known in advance. It is also difficult to model mathematically, making it difficult to derive provable properties of the reconstructed surfaces.

When detecting features as described in chapter 5, the optimal resolution of the analysis is not known in advance, which can hinder fully-automated processing efforts. The results are also not effective when attempting to detect features in a noisy point set, where corrupted data points are often incorrectly flagged as features.

The alignment algorithm in chapter 6 depends on PCA, and so inherits some of its problems, such as the lack of suitability for inputs with high levels of rotational symmetry, and the difficulty of modifying it to work on non-geometric data, such as colour. Another potential issue is that the method is less accurate than performing PCA on the vertices when the inputs are identical (or almost identical). However, this larger error is a trade-off for increased robustness, and the errors are usually in a tolerable range for most applications.

The comparison algorithm presented in chapter 7 is resolution-dependent, and, like the feature detection algorithm, the optimal resolution is not known in advance. Once the appropriate resolution is known however, it is valid for all future comparisons with that mesh. The use of RMS to compute the difference between meshes does carry some problems – partially trained nodes can skew the results.

2. LITERATURE SURVEY

In this chapter I will first present an overview of the 3D modelling pipeline as a whole in order to provide a lens through which the remainder can be read. After this I will delve into details of the algorithms and research relating to each stage, providing context for the research undertaken and its place within the existing body of work.

In particular, the pre-processing stages are important since the algorithms presented in chapters 5 and 4 both require normal data as part of the input. Since normal data is not always available, it is important to know not only how this data can be estimated (section 2.3.3), but also how to attempt to control for – and mitigate – the confounding factors in this process, such as outliers and noise (sections 2.3.1 and 2.3.2 respectively). Since chapters 5 and 4 both describe surface reconstruction algorithms, it is important to understand the context in which they are presented, and as such a detailed survey of existing algorithms is given in section 2.4.

Similarly, methods for aligning meshes are discussed in section 2.5.2, including why PCA is generally not used for mesh alignment. This helps to show why the use of PCA for mesh alignment, as presented in chapter 6, is novel.

Other sections, such as Data Acquisition (section 2.2) are presented purely for context.

2.1 Stages in the Pipeline

2.1.1 Data Acquisition

There are a multitude of sources for 3D data, ranging from CT scans of brains[71] and laser scans of archaeological finds[30], buildings and trees[40]. There are a number of different methods and tools available for

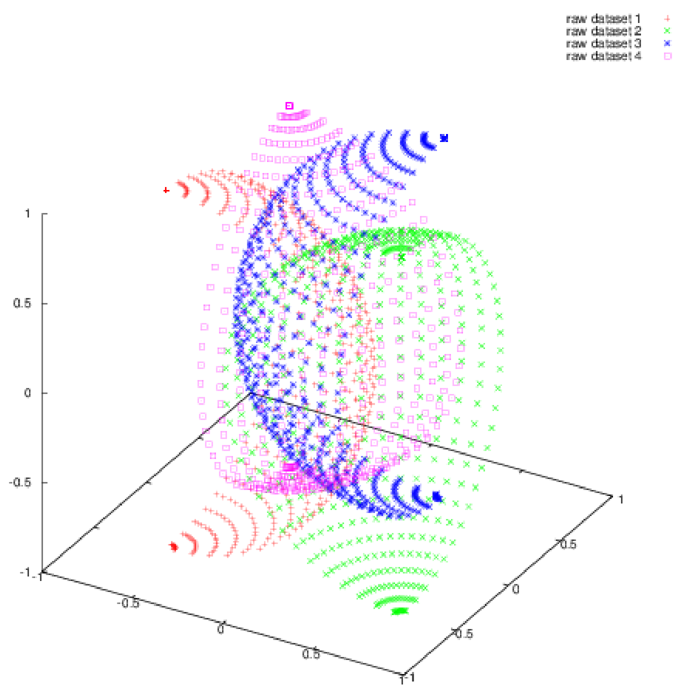

Fig. 2.1: Raw scan data.

scanning objects, each with their own strengths and weaknesses, which often relate to the type of noise and errors to which they are susceptible. For instance, objects with areas of different colours and reflectivity can give rise to significant errors with laser scanners[10]. In the context of cultural heritage, modern methods of data acquisition (such as laser scanners) provide significantly more reliability over the previous methods (photography, wax-rubbing, and free-hand drawing) that have been the mainstay for a long time[30]. Practical considerations often require the data to be acquired in several distinct sets, for instance, to allow the object to be rotated. Figure 2.1 shows what a collection of these raw datasets might look like.

2.1.2 Registration

Since the datasets do not necessarily have a consistent coordinate system, they must undergo some processing to determine if and where they overlap, and how they relate to each other. Following this analysis (which

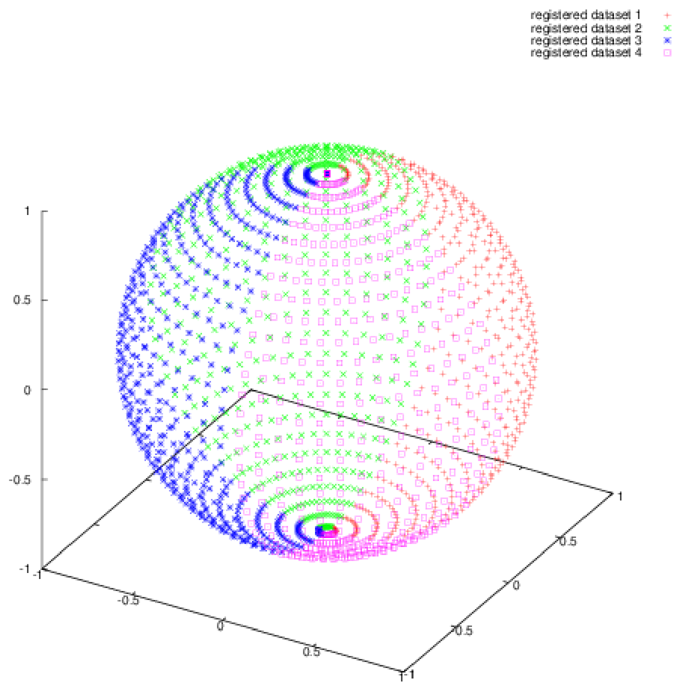

Fig. 2.2: Registered point cloud.

typically involves additional information being collected in the data acquisition phase), systematic errors are detected and removed, and the necessary coordinate transformations are applied to bring each dataset into a unified coordinate system. This whole process is known as *registration*. For instance, the Long Meg rock art was actually scanned in 102 distinct sections, which needed to be registered before the data as a whole could be worked with. Figure 2.2 shows what the previous datasets would look like having gone through this process.

2.1.3 Pre-Processing

After the data has been scanned and registered, it is often preprocessed to clean remove (or smooth out) noise in order to provide clean input data for the surface reconstruction algorithm. It can be used to remove outliers for sensitive algorithms[94], estimate noise in particular regions or simply to ensure correct file formats. There is some overlap between preprocessing and registration, but the registration phase focuses primarily on systematic errors rather than noise. Depending on the acquisition method, normal data for the point cloud may not be available. Since many algorithms are dependent on this data, it can be estimated at this stage.

2.1.4 Surface Reconstruction

Whilst it is possible to represent a surface by a point cloud directly, this comes with limitations such as difficulty finding intersections and deciding whether a point is located within the surface. Consequently the next stage is generally is turning a point cloud into a *mesh*; a representation of the object's surface from which the data were collected. Such a representation is usually deemed accurate if it has no protrusions/features that are not present in the original object, and there are no features/protrusions on the original that are not present in the representation. However, sometimes such inaccuracies are unavoidable. A multitude of algorithms exist for this process and the selection of an algorithm depends on the input, level of noise present, and desired output format. For example, in engineering applications it is sometimes imperative that the output be a watertight solid[27], or that the process is noise tolerant[107]. Figure 2.3 shows a surface reconstructed from the preprocessed data. Most algorithms focus on reconstructing a surface from data with no prior knowledge (e.g. no sorting of the input data along one axis), since this is the most general case. However, knowledge of pre-existing structures in the data – for instance, knowing that there are no holes – can be exploited for significant increases in speed[23] and robustness. During the process of reconstructing the surface, it may be possible to attempt to detect geometric surface features. If a feature is detected then it can affect the future direction of the algorithm, possibly by reducing the smoothing applied to that area in order to avoid dulling the edges of a corner.

2.1.5 Post-Processing

The mesh produced by the surface reconstruction algorithm can be smoothed to reduce the appearance of ridges and discontinuities or to remove other artefacts. Spurious holes can also be detected and filled, though this is a non-trivial problem[14] compared to smoothing.

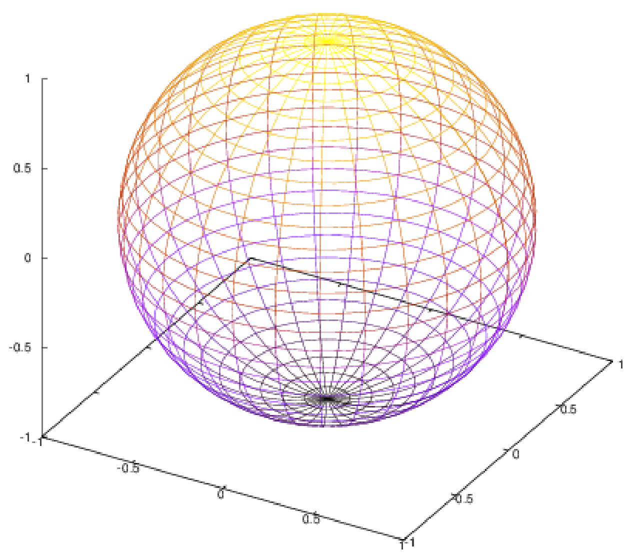

Fig. 2.3: Reconstructed surface.

Once the surface has been processed as required, it may be desirable to compare it to a known-good or theoretical model of the same object. Such a comparison could be used to evaluate the efficacy of the surface reconstruction algorithm, or, by controlling for the effects of the algorithms at other stages of the pipeline, to evaluate the method of data acquisition.

2.1.6 Rendering

Meshes are rendered by creating light sources and calculating where and how the light will be reflected. If surface normals are not provided with the mesh then they must be approximated or calculated before the surface can be rendered. Such approximations can be informed by data collected during the post-processing stage. For instance, if a region of a mesh has been flagged as being low quality by a comparison algorithm (perhaps because the scanned object was damaged in that region), this could be highlighted. The final rendered surface is shown in figure 2.4

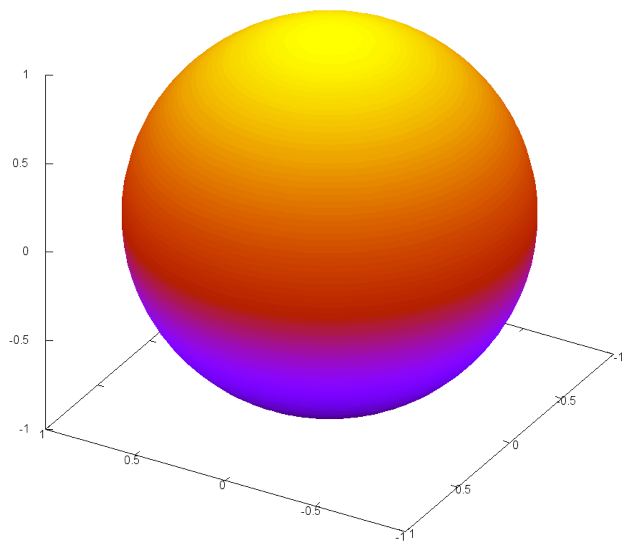

Fig. 2.4: Rendered surface.

2.2 Data Acquisition

The first step in the pipeline is to acquire data to process. This can vary in scale and complexity from scanning a room to identify whether a person present, to mapping an entire landscape. A common factor is that nothing is done to the object or environment being scanned; the process is entirely non-invasive. This stands in contrast to motion tracking, which requires markers to be attached to the target to achieve high accuracy.

2.2.1 Technologies

The number of technologies that can be involved in data acquisition is too broad to discuss exhaustively. Instead, an overview of the most popular methods and a selection of their applications are presented.

Time of Flight

Direct time-of-flight records the time taken for a pulse of light to return after being transmitted. Since the speed of light is constant, the distance from the transmitter (whose position is accurately known) to the object is simply the speed of light multiplied by the time difference. This requires high-end clocks, which are generally too expensive to include in hardware that must be affordable for a large number of consumers.

Indirect time-of-flight methods transmit a continuous wave and measure the difference between the transmitted and received amplitudes, which are less costly to measure. The distance is then computed using equation 2.1, where $\Delta\psi$ is the phase shift, f is the frequency, and c is the speed of light.

$$d = \frac{\Delta\psi}{4\pi f}c \tag{2.1}$$

One of the most popular forms of time-of-flight scanning is LiDAR. LiDAR stands for *Light Detection And Ranging*, and commonly uses a laser as the light source. By combining multiple LiDAR scanners, shadowing can be minimised. A typical setup involves a laser emitting pulses at a high frequency and adjusting the position of the beam by a small, known amount in between each pulse.

LiDAR mapping has been used in landscape surveys for some time, starting with mapping different types of vegetation by their different radiance under red and infra-red light in forested environments[99], and developing into automated, multi-spectral analysis of suburban areas[84].

It has also been used in archaeological preservation[30], and inner-city mapping, where, with some appropriate seed data and a GPS device, it can achieve accuracy levels better than 30mm based on a single drive-by with an appropriate setup[41].

Ultrasound Mapping

Ultrasound mapping works by sending a stream of ultrasound waves into the object to be scanned and performing indirect time-of-flight measurements. Knowing the position of the transceiver allows the operator to infer the distance from the scanner to a change of structure (where the waves will be partially reflected). This change of structure could be between the surface of an object and the air surrounding it, but crucially, it could also be the internal surface of a hollow object.

This ability of ultrasound – to pass through and map multiple surfaces at once – has made it a valuable tool for scanning objects and features that are not easily accessible, or even impossible to access. This non-invasive, deep scanning has found applications ranging from mapping cracks inside rails[52], to detect and pre-empt wear on the rails, which can cause increased maintenance and rougher journeys.

Another application to make particular use of ultrasound mapping's particular strengths is the *in-situ* scanning of plastic pipes and their surroundings[112]. This is particularly important as it is not simply the surface of the pipe itself that must be inspected for cracks. Many problems are caused by the subsidence of material on which the pipe rests, so identifying and filling these voids can improve safety (for gas pipes), but can also allow the type of preventative maintenance that can prevent service interruption (in the case of water pipes).

Synthetic Aperture Radar

In its simplest form, Synthetic Aperture Radar (*SAR*) creates a 2D map by moving a transmitter along one axis (the *along-track* distance) and tracking the return time of a radar pulse transmitted orthogonally to its motion (the *cross-track* distance).

SAR provides its own illumination, making possible the scanning of terrain at night and even from space. The basic form suffers from the problem that scanned locations are often distorted when compared to the 2D view, due to a parallax-type error illustrated in Figure 2.6.

This can be overcome by using two spatially-separated receivers. By having two receivers, the phase difference of the radar pulse received at both gives a third measurement for each point, which, when combined with knowledge of the transmitter and receiver geometry, can be used to compute a 3D reconstruction[89]. Whilst this bears a resemblance to stereoscopy, it is SAR uses combination of one transmitter and two receivers as part of a single system. Stereoscopy on the other hand uses a pair of systems, each of which consists of one transmitter and one receiver.

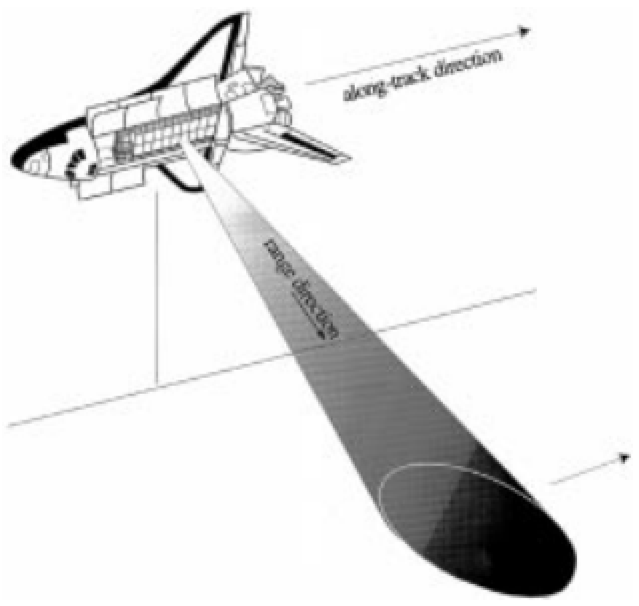

Fig. 2.5: Synthatic Aperture Radar scanning.

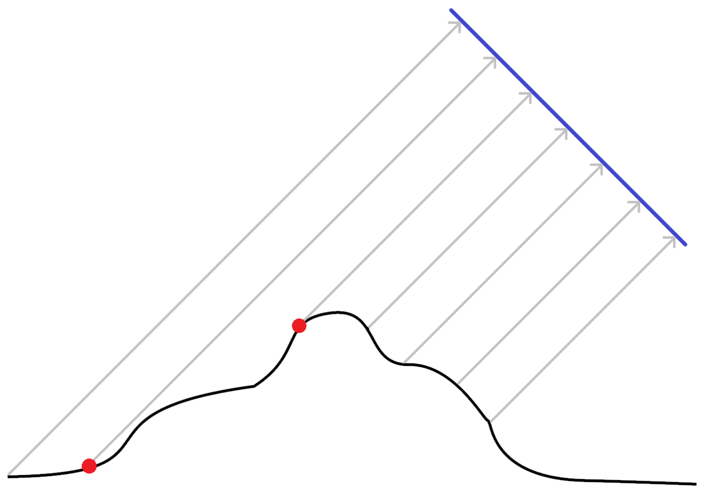

Fig. 2.6: SAR parallax error. The points represented by the grey arrows appear equidistant in the output. Consequently the two red points appear nearby, despite being far apart.

Even this form of SAR is not without its issues however, as it still suffers from shadowing (lack of data due to one part of the scenery obstructing another) and layover (superposition) of data, though scanning from multiple viewing angles can significantly increase the accuracy, and reduce the effect of shadowing[95].

2.2.2 Commodity Scanners

Scanners need not be costly in order to be effective – for instance, the Microsoft Kinect is commodity hardware priced at a consumer level. The original Kinect used a known pattern of light ("structured light") to determine the position of objects and players in the room. It projected a number of rows of dots into the room (with the position of dots in each row being unrelated to the previous row). Their depth into the room was computing by comparing the difference between a point's apparent position and its reference (calibration) position. Using this method, distances of up to nearly eight metres were able to be reliably measured[111].

The follow-up, Kinect v2.0, contains one RGB camera, one Infra-Red (IR) camera, and three IR projectors, and uses indirect time-of-flight to measure distances. Three different modulation frequencies are used to distinguish between the signals, making the second-generation Kinect more robust to operating in sunlight, and allowing it record more accurate depth values[111]. Measurements can be carried out at 30Hz, with 70° horizontal and 60° vertical angles, and an operating range of 0.5–4.5 metres. The software development kit includes a tool called "Kinect Fusion", which uses standard surface reconstruction algorithms to create 3D meshes when the Kinect is slowly moved around and object[63].

Most data acquisition involves moving a scanner around a fixed object map a point in 3D space. Reversing the perspective however, we can use a set of fixed scanners and a mobile object to map the location of that object in 3D space. Filonenko *et. al.* used a set of ultrasound emitters attached to a walls to locate a smartphone on a building floor[32]. Their motivation was that outdoor positioning methods work poorly indoors (e.g. GPS) and current indoor positioning methods are unreliable. For instance, fingerprinting an accurate location based on wireless network and other signal strengths requires dense fingerprint value collection, which must be performed multiple times in order to filter out noise.

Since the speed of sound is significantly slower than the speed of light, expensive hardware is not needed in order to take measurements accurate enough to perform direct time-of-flight calculations with sound. Indeed, the microphones found on commodity smartphones are able to detect frequencies above the audible range. If the positions of the emitters are known, the location of the smartphone can be computed to a high level of accuracy ($< 10cm$). Significantly, this did not require knowing the distance from the smartphone to each of the microphones, only the differences in the distances to each. Given four emitters, this gives three distance differences, which can be used to compute the 2D position of the smartphone on a floor of the building.

Bosse *et. al.* attached a 2D scanner to a moving body (generally a spring), calling their system "Zebedee". The vibrations of the moving body cause irregular scanner movement, which effectively creates a 3D scanner that is simple, mobile and cheap[12]. Their implementation used a 2D time-of-flight laser scanner, but the idea is compatible with 3D scanners. Data are assigned to a time-window, so the cloud is not a snapshot of the scene, though depending on motion within the scene, this may not be important. If increased resolution is required, the rotation can be limited, increasing the frequency with which areas are scanned (and therefore the resolution).

2.2.3 Scanning Objects and Scenes

A famous example of a laser-scanned object being digitised is the Stanford Digital Michelangelo Project[67]. A laser was used to scan Michelangelo David at a 0.25mm scale; the original reconstructed model had approximately 56 million triangles, and was based on a subset of the data. Many years later however, Brown and Rusinkiewicz were able to register all the scan data[16] and a new model was created at full resolution. The full-resolution reconstruction has nearly 1 billion polygons, and at the time was likely to be the largest ever geometric model of a scanned object.

At the other end of the scale is the well-known example of the Stanford Bunny. It is a clean, simple model with approximately 65'000 triangles, and few holes. This makes it a good early test case for algorithms in the early stages of development, before the techniques are refined an applied to more complex models. It was originally created by Turk and Levoy, who used a modified ICP algorithm to provide the initial alignment of the surface fragments[101].

Zhu *et. al.* used an airborne laser to scan terrain for later reconstruction and feature detection[113]. They created a pipeline for reconstructing CAD building models without requiring regularisation, and were able to achieve an accuracy of approximately 0.8 points/m^2. The low density of points made road-edge detection challenging; it was determined that this would require approximately 8 points/m^2.

De Reu *et. al.* applied this pipeline to the field of Archaeology, since future study of the growing archive of site structures and drawings are biased by their 2D nature, which makes reconstruction difficult[88]. Better 3D imaging mitigates the destructive effect of excavation, not only destroys the original source, but also the context of artefacts within a site. It also helps to raise public awareness and participation, However, given the increasingly digitised nature of the field and the need for remote analysis, the methods and equipment used must be both fast and accurate. This allowed them to reduce the amount of manual processing required to achieve the best possible results.

2.3 Preprocessing

2.3.1 Outlier Removal

Motion quantisation, multiple reflections and object occlusion often corrupt (outdoor) point clouds with significant outliers and noise, which require multiple scans for handle. In the case of archaeological sites or streets, it may not be possible to close the area, resulting in *ghost geometry*, perhaps from where a person has moved in between scans. Outlier detection is non-trivial in the absence of prior knowledge of the surface (which would be able to guide the classification). It is also hindered by the unknown distribution of outliers across the surfaces, and by geometric discontinuities, which can falsely suggest that points laying on such discontinuities are outliers.

Kanzok *et. al.* noted that testing if other scanners can “see through” an object detected by one scanner was equivalent to asking if the points from such an object cast a shadow in other scans of the same region[53]. If an object only casts a shadow/obstructs a view in one scan, it is more likely to be an artefact. Their algorithm assigned a confidence to each point, and allowed setting a threshold on this confidence, below which points would be removed as outliers.

Given a noisy set of points containing outliers, Wang *et. al.* defined the k-distance, of a point $p \in P$ $(k_d(p))$to be the distance from p to the k^{th}-farthest point $q \in P$, and the k-neighbourhood of p:

$$N_{kd}(p) = \{q \in P | d(p,q) \leq k_d(p)\}. \tag{2.2}$$

A distance-based deviation factor $(\omega(p))$ was defined for each point, based on the the relative deviation of its local neighbourhood compared to the average deviation of the points in the neighbourhood. The points with the smallest value of $\omega(p)$ were taken as seeds for regions, and the nearest three neighbours were added to each each region (with conditions to prevent the added points being too distant). This was repeated until no more points could be added to any region, at which stage P was partitioned into regions and the smallest regions treated as outliers (since the conditions on adding distance points would prevent their regions growing)[104].

2.3.2 Noise Estimation

Raw data from 3D scanners is rarely directly usable. There are three distinct types of information in meshes created from scanners.

- Connectivity – how the vertices are connected within the mesh (introduced as a side-effect of mesh creation).
- Geometry – the positions of the vertices themselves.
- Topology – how the mesh as a whole connects to itself (e.g. self-intersections or holes).

Connectivity noise is unimportant to us since it is a by-product of mesh creation, therefore no particular connectivity map can be said to be more correct than another. Geometric noise is produced by errors in the data acquisition: errors in measurement and sampling, and is the type of noise that will be considered throughout the rest of this thesis.

Topological noise (incorrect self-intersections or holes) is produced by the mesh generation algorithm. Artefacts from topological noise can be significant, for instance in the case of CT scans the resulting model will have an incorrect representation of how biological structures are connected to each other[105].

Most denoising algorithms perform the same conceptual steps[86]:

- Apply a transformation to move the noisy signal to a domain where the signal and noise are cleanly separated.
- Use assumptions about the effect of transform on the noisy signal to remove the noise.
- Apply the reverse transformation.

Even if the noise cannot be completely removed, it can be beneficial for certain algorithms for it to be smoothed in order to achieve a more uniform distribution over the whole data set.

Many methods of noise estimation assume that it is evenly distributed across a surface, though this is not always the case. Yoon *et. al.* approximated a neighbourhood of points by a uniform B-spline[107]. Their method uses a variational Bayesian algorithm to estimate the quantity and variance of the noise. A new B-spline (with more control points) is then computed, taking the noise estimation into account.

The algorithm was tested on meshes that had varying amounts of noise added to them and performed very well. The presence of features interfered with the estimates of noise (something inherent in the algorithm) but this did not cause serious problems. Predictions for the locations and amount of noise agreed with visual inspection, and the most appropriate lattice size was found for each area.

2.3.3 Normal Estimation

Principal Components Analysis

The goal of Principal Components Analysis is to extract meaningful relationships from a cornucopia of noisy data. The relationships between the data are assumed to be linear, but this is rarely an issue and greatly simplifies the analysis. We therefore attempt to find an optimal basis for expressing the data (i.e. one that will highlight the relationships) by using a linear combination of the original basis vectors.

Given a matrix X representing a dataset, each column corresponds to the set of measurement types and each row of X represents all the data of an individual measurement type $(x_0, x_1, x_2, \ldots)$. To represent the X in a new basis (P), we transform it like so:

$$\mathbf{PX{=}Y.} \tag{2.3}$$

We assume that after this transformation, the most important correlations will occur in the directions of greatest variance. To get a measure of the redundancy in our dataset we calculate the covariance matrix of our new dataset Y like so:

$$C_Y = \frac{1}{A} Y Y^T. \tag{2.4}$$

C_Y is a square, symmetric matrix. The diagonal elements are the variance of the different measurement types, the off-diagonal elements are the covariance of the measurement types with respect to each other. By our earlier assumption, a large on-diagonal element indicates that a relationship is significant, whereas a large off-diagonal element indicates a large degree of redundancy in the measurements. If we express C_Y as ABA^{-1}, where B is a diagonal matrix of the its eigenvalues, then A will be a matrix of its mutually-orthogonal eigenvectors.

These new eigenvectors, ordered by eigenvalues, are called the principal components of Y, and are centred on the component-wise mean of the original dataset represented by X. If necessary, the dimension of the data is reduced by removing the least significant principal components. Principal Components Analysis can be used for normal estimation because when the data represents a surface, there is far more variation parallel to the surface than orthogonal to it, so the least significant component can approximate the normal to the surface.

Robust Tangent Plane Estimation

Li *et. al.* developed a new method of normal estimation, since in their view Principal Components Analysis is, whilst efficient, too susceptible to noise. They noted that the scales on which one must look in order to determine the local noise level and to estimate the local tangent plane are different – a large scale is needed for noise determination, and a small scale for accurate tangent plane (normal) estimation[68].

For each point in a neighbourhood (the size of which must be manually chosen), three other non-colinear points are selected to estimate the tangent plane, and the residuals of all other points (their shortest distance to this plane) are computed. Three different points are then chosen, a new plane defined, and all points have their residuals computed and sorted in ascending order for this new plane. This process is repeated for each point in the set, so that for N points in the neighbourhood, we have N potential planes, each of which has an associated (and sorted) list of residuals stored. The plane with the lowest k^{th} residual is used to estimate the noise scale from its list of residuals. The value of k is chosen according to the feature size: the smaller the value of k, the smaller the features that could be detected. Li *et. al.* found that taking k to be 20% of the neighbourhood size gave satisfactory results. The algorithm requires that the noise distribution be Gaussian however, and the normals produced are unoriented.

Wang *et. al.* took a different approach. If a point p is not near a sharp feature, take three non-colinear points to estimate the normal plane. If p is near a sharp feature, there could be several nearby surfaces that could be used, so a clustering algorithm is used to determine the most appropriate. For a set of n points near a sharp feature, m planes are defined, each by three non-colinear points. The residual of each point to the plane is then computed, and the list of all such residuals is sorted in non-decreasing order. The top-k list of a point i is the list of k planes for which i has the smallest residual (effectively, the first k planes in its sorted residual list)[104].

It is intuitive that if two points lie on the same substructure, their top-k preference lists will be similar. Kendall's Tau is used to determine the "distance" between top-k preference lists, but it is inverted to give a similarity measure. Given a point p and its neighbouring point set Q, we compute in turn the similarity of each $q_i \in Q$; if q_i and p are above a threshold of similarity, then q_i is added to the set of p's set of consistent points.

Least Squares

Mitra *et. al.* sampled a 2D surface and added noise to simulate a noisy dataset from which they intended to estimate normals[75]. The inputs to their algorithm are the dataset D, a user-defined radius r and an initial number of neighbours to look at, k_0. For each point $p \in D$, the algorithm finds all points inside the r-sphere centred on p. The density is then estimated by:

$$\rho = \frac{k}{\pi r_{old}^2} \tag{2.5}$$

and used to approximate the local curvature. This curvature is then used to compute a new value of r, which is in turn used to compute a new value of k, the number of neighbours to use in the estimation calculation, by:

$$k_{new} = \lceil \pi \rho r_{new}^2 \rceil . \tag{2.6}$$

The preceding operations are performed a predefined number of times, at which point the least squares plane of the k-nearest neighbours provides an estimate of the surface's normal at p.

Sheung *et. al.* selected a subset $\{p_i\}$ of the point cloud, then for each point p_i, n points are selected from its neighbourhood and a quadratic surface fitted to them using a least squares method[97]. The sum of the residuals of each point is computed, with a lower value taken to indicate a greater probability of the quadratic surface being accurate.

This process is repeated many times, and the quadratic surface with the lowest sum of residuals is used as the surface estimator. The points are then "pulled back" onto this surface estimator, and their normals are assigned to be the normal of the quadratic surface at their new position.

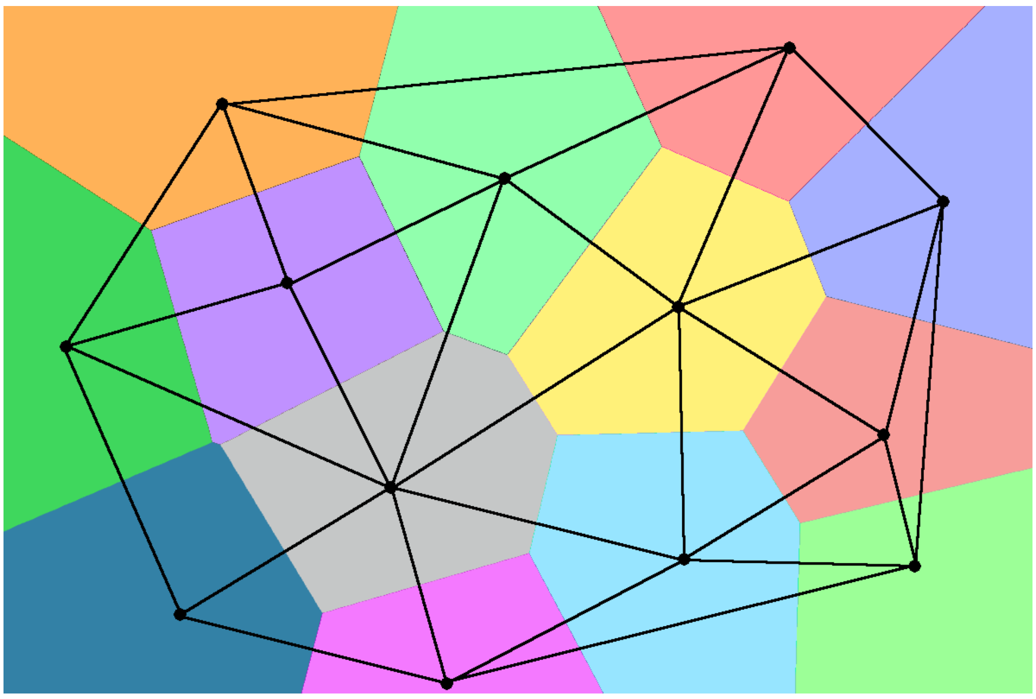

Fig. 2.7: A 2D Voronoi diagram (coloured shapes), with the Delauney triangulation overlaid in black.

Delauney Balls

Given a set of points

$$P = \{p_0, p_1, \ldots, p_N\} \in \mathbb{R}^2 \tag{2.7}$$

the *Voronoi cell* of a point p_i is the region of $\mathbb{R}^2$ that lies closer to p_i than any other point in P. The *Voronoi diagram* is the decomposition of $\mathbb{R}^2$ into Voronoi cells. A 2D Voronoi diagram is shown with the coloured shapes in Figure 2.7. The corners of Voronoi cells are called *Voronoi vertices*.

Each Voronoi vertex is equidistant from exactly three members of P (in N dimensions, they are equidistant from $N + 1$ members of P). If we connect these three points together then we get the *Delauney triangulation*. The Delauney triangulation of is shown in black in Figure 2.7. The Delauney triangulation generalises to three dimensions, where we get Delauney Tetrahedrons.

A Delauney ball is simply a ball circumscribing the four vertices of a Delauney tetrahedron. Dey *et. al.* proposed a method that uses

Delauney balls centred on Voronoi poles (the farthest Voronoi vertices from a point) to approximate normals[29]. Each point in the sample set has two Delauney balls; one with its centre inside the surface, the other with its centre outside. These are used to approximate (unoriented) normals at the sample points.

They showed that even in noisy samples (which can severely restrict the size of the Voronoi cells, and hence Delauney balls), large Delauney balls still exist and can be used to give a good estimate of normals. Estimates can be made not only for the sample points the Delauney balls are incident to, but also to nearby points that have no large, incident Delauney ball.

Orientation

Estimating normal data is not just a question of the direction of the normal (i.e. magnitude of the normal vector's components). The orientation of the normal in that direction must be accurate if these data are to be relied on when reconstructing or rendering a surface.

Given N unorganised points lying on (or near) a surface, with normals of unknown orientation, Liu *et. al.* developed a method of orienting them as a precursor to reconstructing a surface[70]. First, a set of covering spheres is created and a weight assigned to each point. In areas of high sampling density, each point has a low weight, in areas of low sampling density, each point has a high weight.

The covering spheres are expanded until the residual error crosses a threshold, at which point a rough triangulation of the surface is generated. This triangulation need not to be geometrically accurate: only the orientation of the normals is important, not their values. For all input points p_i, the closest point on the surface M is found (c_m), the direction of c_m's normal is then assigned to p_i.

2.4 Surface Reconstruction

2.4.1 Early Algorithms

Contour-Based Methods

Contour-based methods rely on triangulating the space between adjacent contours, then joining all such strips together to form the final surface. The contours are typically dense, parallel cross-sections of an object. However, if multiple scans are taken there can be multiple contours per slice, resulting in ambiguities. There are other pitfalls, many of which can be attributed to the fact that the methods throw away data[71]. Contour-based reconstructions have taken a back seat to other types of algorithm in recent years, but do still see some work. For instance, Barequet *et. al.* studied reconstruction from a set of sparse, non-parallel cross-sections[5]. This stands in contrast to the earlier papers, which looked at parallel, often dense cross-sections.

The Marching Cubes Algorithm

Probably the most famous algorithm for surface reconstruction is the Marching Cubes algorithm, developed by Lorensen and Cline[71]. It is simple yet powerful, and remains in frequent use long after its initial development.

A scalar function $f(x, y, z)$ is defined over a 3D space D (taken to be a cube, without loss of generality) containing the object. A value of f (usually 0) is then chosen to represent the surface. D is subdivided into many smaller cubes[1] and f is sampled at each corner of these smaller cubes. By the Intermediate Value Theorem, if one corner is inside the surface ($f \leq 0$) and another is outside ($f \geq 0$), then the surface ($f = 0$) must pass between them.

Each corner can either be inside the surface or outside[2], therefore there are $2^8 = 256$ possible combinations of corners being inside/outside the surface. When reflective, rotational and internal/external inversion symmetries are taken into account the number of unique combinations is reduced to just 15, which are shown (with some equivalent combina-

[1] In spite of the name, there is no requirement to use cubes instead of cuboids.
[2] Ignoring the case where the surface passes precisely through a corner.

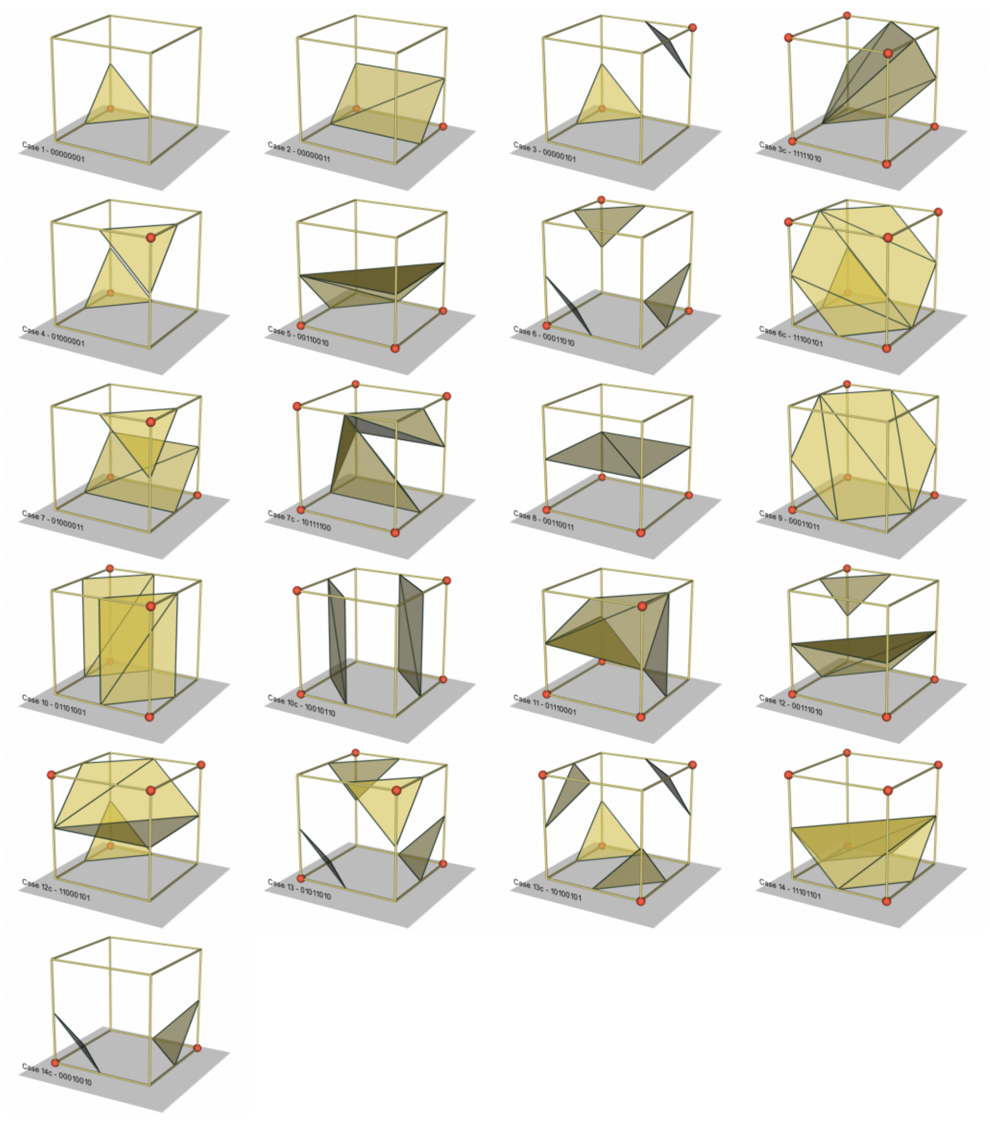

Fig. 2.8: The different cube configurations.

tions duplicated for clarity) in Figure 2.8. By creating an index for each unique intersection cube we can create a lookup table for each cube and reconstruct the surface with ease.

To give an example of this for a single triangle, figure 2.11 shows the output for a cube where v_5 is inside, and all others are outside (or vice versa). Point i_1 lies on the surface (i.e. $f(i_1) = 0$), its position along edge e_6 is determined by linear interpolation between $f(v_2)$ and $f(v_6)$. Lorensen and Cline investigated quadratic interpolation of the distance of the intersection along the cube's edge, but found it to offer no significant improvement in accuracy.

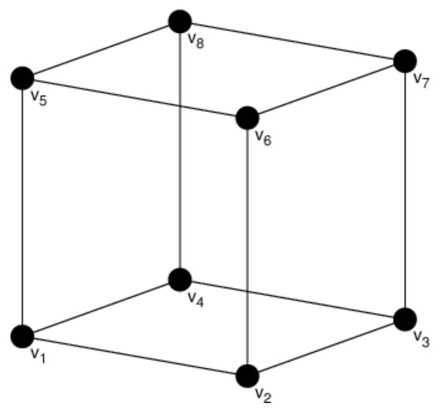

Fig. 2.9: The function f is sampled at each of the cube's vertices.

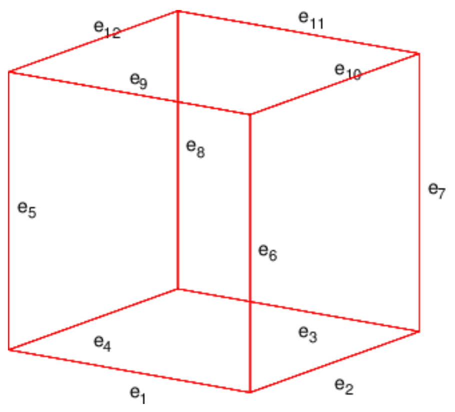

Fig. 2.10: The labelled edges of the cube.

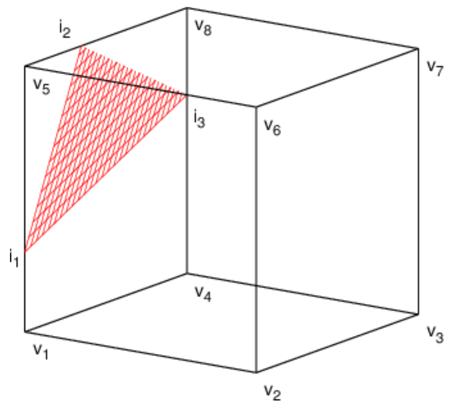

Fig. 2.11: Points i_1, i_2 and i_3 denote intersection of surface with edges.

Marching Cubes Modifications

Due in part to the simplicity of its implementation, Marching Cubes has become the *de-facto* standard algorithm for isosurface extraction. Unfortunately it has a tendency to smooth out sharp features and as such modifications have been proposed to counteract this. One method is to use a *directed* signed distance field; *i.e.* rather than storing the scalar distance of a vertex to the surface, the distance from the vertex to the surface along the direction of each axis is stored separately[60]. This allows the computation of a more accurate intersection of the surface with the cube, giving sharper corners and better-reconstructed features.

Schaefer and Warren created the Dual Marching Cubes algorithm to address the smoothing of sharp features[93]. The dual of a mesh is created by switching faces with vertices, and connecting any two vertices if the faces from which they were originally created shared an edge. Whereas the Marching Cubes algorithm runs on a regular 3D grid (or an octree), their modification runs on a grid dual to this. The dual grid conforms more closely to the features of the implicit function and therefore allows features to be extracted more accurately whilst using fewer polygons. This dual-graph method of reconstructing sharp features was also used by Sheung and Wang[97].

Another shortcoming is the production of low-quality triangles, where one edge is significantly shorter when compared to the other two. First, the border of each polygon within a cube is computed, and vertices with bad (i.e. small) angles are detected. The two vertices adjacent to these bad angles are then connected, to isolate its impact (if other vertices were connected, the angle would be subdivided, creating more bad triangles). For polygons with a circumference of four or more edges, a new vertex is placed inside (subject to user-defined thresholds), close to the smallest edge (thus minimising the angle created by this new vertex with the edge This results in a mesh consisting of triangles with significantly better side length and angle ratios[64].

Marching Tetrahedra

The Marching Tetrahedra algorithm follows the same idea as Marching Cubes, but once the input has been partitioned into cuboids, it partitions

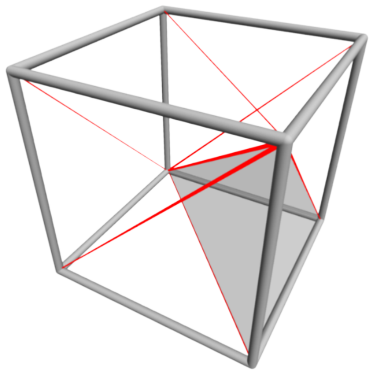

Fig. 2.12: Splitting a cube into 6 tetrahedra, one of which is shaded.

each cuboid into six tetrahedra. These tetrahedra are created by cutting diagonally through each pair of cuboid faces.

Doing this consistently across the whole dataset ensures that intersection points can be shared between cuboids. Consequently there are nineteen potential points of intersection in the cuboid, instead of twelve with Marching Cubes (though the point on the main diagonal is entirely contained within the cuboid). Each tetrahedron is evaluated to one of the following cases:

- No surface intersection.
- Intersection resulting in one triangle (one vertex in/outside).
- Intersection resulting in two triangles (two vertices in/outside).

The additional intersections increase the resources require to run the algorithm, but leads to a more accurate representation of the isosurface. It also resolves an ambiguity in some cube combinations in the Marching Cubes algorithm.

2.4.2 Explicit Algorithms

In this section we explore algorithms that directly create an explicit surface from a point cloud.

Basic Concepts

The *convex hull* of a set of points in 2D is a curve enclosing all points in such a way that no part of the curve is concave. This concept extends

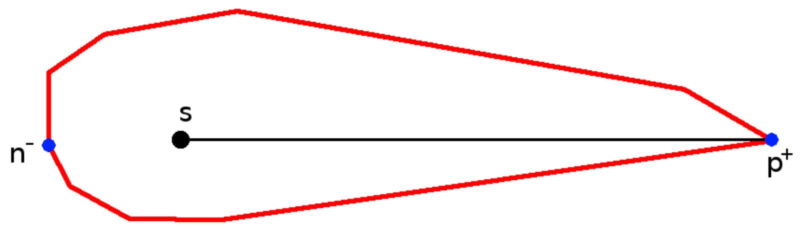

Fig. 2.13: Determination of p_s^+ (and therefore n^+) for a point s not on the convex hull.

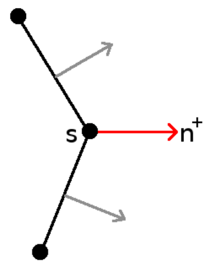

Fig. 2.14: Determination of p_s^+ (and therefore n^+) for a point s on the convex hull. The grey arrows indicate the normals of adjacent triangles.

to 3D, with the equivalent condition that no point on the surface may have negative curvature. If a point is in the convex hull, its Voronoi cell is unbounded.

Given a sample point s, we define the following in its Voronoi cell:

- p_s^+: If s is not on the convex hull, this is the farthest Voronoi vertex from s, shown in Figure 2.13.
- n_s^+: if s is on the convex hull of S, this is the vector sp_s^+. Otherwise it is the average of outer normals of adjacent triangles, shown in Figure 2.13.
- p_s^-: the Voronoi vertex whose negative projection onto n_s^+ is farthest from s.

p^+ and p^- are called the *poles* of s.

Crust

In 1998, Amenta *et. al.* developed an algorithm that uses the Voronoi diagram and Delauney triangulation. No experimentally determined parameters were given as input; they were all calculated locally[1].

The algorithm starts by computing the Voronoi diagram of the sample set S. P is defined to be the set of all poles p^+ and p^- and the Delauney triangulation is calculated for the the set of all points lying in $S \cup P$. Finally, any triangles with a vertex in P are removed.

The output was found to need more filtering in order to guarantee a good reconstruction, so triangles whose normals differed too much from n^+ or n^- were thrown out, after which the output normals then converged to the surface normals. The algorithm rests on the assumption that the Voronoi cells are long and thin, but around sharp edges they are much fatter, leading to greater variability in the normals. This may lead to desirable triangles being deleted near sharp edges/features.

Undersampling caused holes to appear, but by moving all poles a fixed fraction closer to their corresponding sample point, the holes appeared in different places. Taking a union of the modified output and the original output sometimes gave a perfect reconstruction. Noise also presented a problem: when the level of noise was roughly equal to the sampling density the algorithm broke down and was unable to reconstruct any surface. It was suggested that there may be a thick surface algorithm that would be able to tolerate a higher level of noise.

Power Crust

The *medial axis* is the set of points that have more than one closest point on the surface. It may be divided into several distinct sections, each of which may be inside or outside the surface. In two dimensions, the positions of the Voronoi vertices approximate the medial axis.

A medial ball is a sphere centred on the medial axis or a centre of curvature of the surface, and that has as large a radius as possible without containing any sample points. The medial axis transform is the union of all such balls.

Power Crust is a modification of Crust that was designed to create watertight surfaces[2]. It uses the poles as defined in section 2.4.2 to approximate the medial axis transform. Each ball has its centre labelled as either inside or outside the surface, and the “power crust” is defined to be the boundary of the union of all the internal medial balls.

Cocone

Given a point s with pole p^+, the vector sp^+ provides an estimate of the normal at s. The cocone of a point s is the complement of a solid double cone centred on s and aligned with sp_s^+[3]. The angle of the cone (θ) is given.

The Cocone algorithm[24, 25] selects triangles from the Delaunay triangulation whose Voronoi edges intersect the cocones – these are the candidate triangles. If the sampling density is sufficiently high, the candidate triangles lie near the surface and have normals that are nearly the same as their vertices' normals. A continuous surface is then extracted from this set of candidate triangles.

In well-sampled regions, the Voronoi cells are long and thin, in undersampled regions they are short and fat. The Boundary algorithm detects undersampling by testing how skinny the Voronoi cells are and whether their elongation is close to that of their cocone neighbours. It is unknown whether intended boundaries can be recognised at the same time as filling holes that have arisen due to undersampling.

Tight Cocone

Tight Cocone[27] is a modification of Cocone that produces watertight surfaces. Cocone is modified to call the Boundary algorithm and allow only those triangles not marked as undersampled to be selected as candidate triangles. This removes bad triangles and gives a surface that usually contains holes.

Sample points are labelled good if their incident triangles form a topological disk, otherwise they are labelled bad. All infinite tetrahedra are marked as out. A stack of "good" point and "out" tetrahedron pairs (p, σ) is initialised with a point and an infinite tetrahedron and maintained thereafter. All tetrahedra connected to σ and p are walked through, without crossing the surface triangles incident to p. These tetrahedra are all marked as out. When a vertex/tetrahedron pair q, σ' (incident to p) is reached, (q, σ') is added to the stack if q is good and unexplored. p, σ is then popped off the stack and the next pair is explored.

When there are no more pairs on the stack, a "peeling" algorithm works through a stack of triangles built in the previous stage and removes any that do not form part of the surface.

The performance bottleneck is the 3D Voronoi calculation. In the worst case, Tight Cocone is quadratic in the number of sample points, though this was not seen in practice. Issues with the algorithm include its fundamental inability to reconstruct internal voids and its inability to construct a surface above a particular noise threshold.

Other Cocone Variants

Cocone has two more variants, each suited to different tasks. RobustCocone[28] computes a surface by interpolating a subset of the sample points and is better suited to noisy datasets (assuming that the sampling density is high by comparison to the local feature size). There is also SuperCocone[26] which is designed to handle very large data sets. It does so by using octree subdivision to divide the data into subsets, applying the Cocone algorithm to these subsets, then matching surface sections from adjacent sets to create the final mesh.

Covering Spheres

A set of covering spheres are generated and all points are labelled as uncovered. An uncovered point is selected at random to be the centre of a new sphere. An error function is defined for each sphere, which takes its minimum value at x_m).

If x_m is inside S_i then x_m is marked as the auxilliary point of S_i. Otherwise, the centre of S_i is used as its auxilliary point. All the points contained within the sphere are then projected onto the tangent plane of all the points in the sphere. On this plane, the convex hull of the points is computed, and any points not lying on the boundary of the complex hull are marked as covered. The process is then repeated until all points are covered. A Radial Basis function method was then used to reconstruct the surface[70].

2.4.3 Implicit Algorithms

Implicit surface reconstruction methods usually require points with normals as inputs [18, 100, 82]. Using a standard technique, the input data are assigned a scalar value of 0, the ends of the normals pointing to the exterior of the surface are assigned value 1, while the opposite normal ends are assigned value -1. These values are interpolated or approximated by local or global scalar fields, which are then blended to create a single global scalar field approximating the signed distance to the surface.

Finally, the surface is extracted as the zero-level set of the scalar field. Implicit methods perform well when presented with poorly sampled data, but interpolating algorithms are not robust in the presence of noise. Implicit methods also have significant difficulty representing surfaces with boundaries.

Hoppe's Method

A set of points is "ρ-dense" if any sphere centred on a sample point, with radius ρ, contains at least one other sample point. Hoppe *et. al.*[43] assumed the sample points to be "δ-noisy", i.e. $||\text{noise errors}|| < \delta$.

The tangent space of each point x_i is estimated using its k-nearest neighbours and the centroid (barycentre) of these used to determine the centre of the tangent plane o_i. Principle components analysis is then used to determine the normal vector, n_i. A graph is created of the centroids and the consistent alignment is solved as an optimisation. The point with the largest z component is defined to have a normal pointing outside the surface, and this "outward pointing" label is propagated to neighbouring normals.

The distance of each point from the surface is estimated by evaluating $f(x_i)$ and weighting it with the distance (projected onto n_i) from the point to its centroid, like so:

$$\text{estimated distance} = f(x_i)(x_i - o_i).n_i. \tag{2.8}$$

If the estimated distance is larger than $\delta + \rho$ then the point is discarded.

The zero-set of the distance function is approximated in each voronoi region, giving a discontinuous global approximation. Post-processing was used to ensure that long and thin triangles were not produced. The algorithm automatically discovered the topology of the surface and performed well with respect to surface geometry, it also generalises to higher dimensions allowing surface attributes to be modelled as well as the physical dimensions.

Radial Basis Functions

If a function depends only the distance of its parameter(s) from the origin, it is known as a radial basis function. Ohtake *et. al.* used radial basis functions to interpolate a point set. First, a hierarchy of point sets $P = \{P^1, \ldots, P^m\}$ is created by placing P in parallelepiped and octree-subdividing. Then, the centroid of each cell is computed and assigned a normal vector, which is computed by averaging the normals of all the points in the cell and normalising the result[81]. The support radius of the RBFs is taken to be three-quarters of the average diagonal length of the cells in the parallelepiped.

The interpolating functions are recursively defined from the previous set like so:

$$f^k(x) = f^{k-1}(x) + o^k(x), \tag{2.9}$$

the initial function is taken to be a constant at -1. o^k is an offset function - it is a correction applied to the previous level's interpolating function. The support size decreases by a factor of two with each recursion. The surfaces given by this method can be used for a variety of implicit surface operations, including morphing and cutting. The algorithm method gave good performance and works with irregularly and badly sampled data, though since it interpolates points, it is not robust with respect to noise.

Carr *et. al.* used RBFs fitted to a subset of the point cloud to reconstruct the surface[18]. These points are assumed to lie on the surface, and each is augmented by two off-surface points whose positions are given by $x_i + n_i$ and $x_i - n_i$, where n_i is the point's normal, which may be given or estimated. The problem is therefore that of finding a global function f such that

$$f(x) = 0 \text{ where } x \text{ is on-surface} \tag{2.10}$$

$$f(x) = ||n_i|| \text{ where } x \text{ is external off-surface} \tag{2.11}$$

$$f(x) = -||n_i|| \text{ where } x \text{ is internal off-surface} \tag{2.12}$$

The points to which an RBF is being fitted are classified as either near or far to the RBF's centre. Near points are evaluated individually, whereas far points are clustered and the the cluster's influence as a whole is approximated. This results in a dramatic speedup that makes the use of RBFs feasible for large point sets.

A greedy algorithm is used to reduce the number of interpolation nodes necessary to represent the surface to the same level of accuracy. Starting with the initial number of interpolation centres, if the error for a particular RBF is above a threshold, then a new RBF centre is added nearby until all fitting errors are below the threshold.

The isosurface is then extracted using a variant of the marching tetrahedra surface extraction algorithm that only retains those vertices that are on the "wavefront" of the algorithm, and so reduces the memory requirement. It also has the advantage that the computational cost increases with the square of the resolution, not the cube. This approach was developed further in [19], where low-pass filtering was used to allow the reconstruction of surfaces from noisy point clouds with large undersampled or unsampled regions.

Partitions of Unity

Partitions of unity are used to combine many local approximations into a single global approximation, inheriting various properties like maximum error. The basic idea is to split the domain into several regions and solve the problem in each subdomain. These local solutions can then be added together with small local weights which sum to 1 at all points in the domain.

Ohtake *et. al.* used octree-based subdivision to create the subdomains. The weight function of each cell was given a support radius that was a multiple of the diagonal of the cell. If this did not contain enough sample points then the radius was allowed to grow by a fixed proportion[80].

For each cell they allowed three possible approximations: a general 3D quadric, a bivariate quadric in local coordinates, and a piecewise quadric surface. The general 3D quadric is used for larger areas that could be unbounded or contain more than one sheet. The bivariate quadric in local coordinates is used to approximate local smooth patches. Finally, the piecewise quadric surface performs several feature tests to determine the most appropriate type of approximation, making it effective for edges and corners. If there are many points in a cell then one of the first two is used: if the average normal deviation is greater than $\frac{\pi}{2}$ then the first is used, otherwise the second is.

The method can also be adjusted for interpolation by simply giving each point its own cell. The reconstructions are robust with respect to variations in point density and sharp features are reproduced. The algorithm worked well with poorly sampled data, especially when compared to [82], which was more sensitive to density variations.

Nagai *et. al.* extended the partition of unity methods, smoothing the local approximations whilst attempting to preserve features[76]. Each octree was assigned a spherical support with a local signed distance function. Principal Components Analysis was used to calculate the smallest eigenvector, which was compared to a vector from the centre of the sphere and the centroid of all points contained in it. If the sampling density changed in the sphere, this angle was large and so the points were assigned a low confidence (often indicating an outlier). The method is robust to noise and able to cope with random normal rotations of up to 60 degrees, with some manual parameter tuning.

2.4.4 Least Squares Methods

Basic Least Squares

Least square methods are all variants on, or extensions to, the basic least squares method of minimising a cost function. Given a set of points $\{x_i\}$ and corresponding values $\{f_i\}$, the goal is to construct a global function $f(x)$ such that the difference between $f(x_i)$ and f_i is as small as possible. Therefore a cost function, G_{LS}, is defined like so:

$$G_{LS} = \sum_i \|f(x_i) - f_i\|^2. \tag{2.13}$$

In order to minimise the error, the coefficients and constants in $f(x)$ are chosen such that G_{LS} is minimised. This is done by computing the partial derivatives and solving for zero.

Weighted Least Squares

The weighted least squares method is an improvement on this since it (unsurprisingly) includes a weight function. It defines a set of weighting centres $\{\bar{x}_i\}$ with corresponding local approximations. Errors close to a weighting centre are treated as "worse" that errors further away[77]. The error is:

$$\begin{aligned} G_{WLS} &= \sum_i \Theta(d) \| f(x_i) - f_i \|^2 && (2.14) \\ d &= \| \bar{x} - x_i \|. && (2.15) \end{aligned}$$

A local approximation is calculated for each of the weighting centres, and a global approximation created from these using a partition of unity.

The choice of $\Theta(d)$ is situation-dependent and often includes a spacing parameter h to smooth out small features and irregularities. Popular examples include a Gaussian function:

$$\Theta(d) = \exp\left(-\frac{d^2}{h^2}\right), \tag{2.16}$$

with its non-compact support, and the Wendland function:

$$\Theta(d) = \left(1 - \frac{d}{h}\right)^4 \left(\frac{4d}{h} + 1\right), \tag{2.17}$$

which is well-defined on $d \in [0, h]$ and has the convenient properties that $\Theta(0) = 1$ and $\Theta(h) = 0$ as well as C^2 continuity ($\Theta'(h) = 0$ and $\Theta''(h) = 0$). A judicious choice of $\Theta(d)$ can dramatically change the result of the fitting process. For instance,

$$\Theta(d) = \frac{1}{d^2 + \epsilon^2} \tag{2.18}$$

with $\epsilon = 0$ forces interpolation rather than approximation. The partition of unity functions $\psi_j(x)$ are given by:

$$\psi_j(x) = \frac{\Theta_j(x)}{\sum_{k=1}^{N} \Theta_k(x)} \tag{2.19}$$

with the requirement that

$$\sum_j \psi_j(x) \equiv 1 \tag{2.20}$$

everywhere. The global approximation is then given by:

$$f(x) = \sum_j \psi_j f_j(x) \tag{2.21}$$

where $f_j(x)$ is the local approximation for the weighting centre $\bar{x}_j$.

Moving Least Squares

Moving least squares is a logical extension to the weighted least squares method whereby a local approximation is calculated for every point in the dataset, *i.e.* for every x_i, an $\bar{x}_i = x_i$ is created. This approach has some significant benefits. Levin *et. al.* used moving least squares to approximate an $N-$dimensional function[66]. The error was bounded by the error of the best local polynomial approximation. Kolluri used moving least squares and developed an algorithm that, given sufficient sampling density and bounds on noise, could produce a provably-good reconstruction (both topologically and geometrically)[62].

Mederos *et. al.* clustered the sample points, then computed a point on the moving least squares surface that was representative of the cluster[74]. These representative points were triangulated by taking the nearest neighbour r_j of a point r_i and inserting an edge, then selecting a third point r_k in order to maximise the angle $r_j r_i r_k$.

Algorithms based on the MLS projecions are the a very successful case of point-set surfaces, where the surface is implicitly defined as the set of the fixed points of a projection.

2.4.5 Neural Algorithms

Fundamentals

A neuron is an object that consists of N inputs (x_i), an output (o), an *activation function* (f), and a *learning rule.* Each input is assigned a weight (w_i), which can be positive or negative and (almost) always lies in the range $[-1, 1]$. The weighted sum of the inputs is called the *activation* (a) of the neuron. The activation function takes a as a parameter and gives the output of the neuron.

$$a = \sum_{i}^{N} w_i . x_i, \tag{2.22}$$

$$o = f(a). \tag{2.23}$$

The activation function may produce an output based the current activation or on some combination of the current activation and past activations. Over time the weights are typically modified in order to give more precedence to some inputs than others. The manner in which these modifications are made is defined by the learning rule.

A neural network is simply a number of connected neurons. Each neuron may take its inputs from the output of other neurons or from external inputs (or any combination of the two). Each neuron has only one output value, which may be used as input for many other neurons. The connections between neurons are called synapses.

There are several different ways that a neural network can be trained, each suited to different situations. *Supervised learning* requires an external teacher that knows the desired response from the network. The teacher updates the weights after every response in order to move the network's response closer to the desired output.

Hebbian learning updates the weight of the synapse between any two connected neurons according to the correlation between their activations. If both tend to be activated concurrently then the weight increases over time (the converse is also true).

Two neurons are neighbours if they are directly connected by a synapse. It is common however, to loosen this definition so that two neurons are neighbours if one can move from one to the other across fewer than m synapses (where m is a constant chosen by the user).

Self-Organising Maps

Self-organising maps (SOMs) are a particular type of neural network. They were developed by Teuvo Kohonen as 2D neural networks used to express complex non-linear relationships between data into simple geometric relationships between the nodes[61]. The use of SOMs is not confined to geometric modelling: they have shown themselves to be useful in areas such as pattern recognition and shape indexing[98].

When applied to surface reconstruction, the SOM is placed within the data to be analysed and its nodes have their weights randomly initialised over their nearest N neighbours. Each node's position is then updated to the weighted mean of all its neighbours' positions. The nearest N neighbours are then recalculated and the process begins again. It is often the case (but not guaranteed) that the SOM will converge on a good representation of the data. In this context, good representation means higher node density in areas of higher data density, and node values providing reasonable approximations of data values.

Several surface reconstruction algorithms are based on SOMs and their variants. They frequently use neural networks with 2D connectivity, and the result of the training is an explicit model of the surface data, such as a triangle mesh or the control grid of a uniform bivariate spline. Implicit SOM methods for surface reconstruction were introduced in [106]. SOMs have been used for grid fitting in [6] and for surface reconstruction in [110]. In [34, 46], special types of SOMs called *Growing Cell Structures*, that dynamically create edges between nodes, are used for the same problem. Unfortunately, the growing cell structures required the entire point cloud to be sampled several times in order to achieve a stable result.

Neural Pre-Processing

An interesting application for neural networks/SOMs is as a form of pre-processing. For instance, to generate a Bezier surface we require a 2D grid of control points. Hoffmann *et. al.* used surface data to train an SOM, the final state of which was to be used as input to a surface reconstruction algorithm[42].

The size of the SOM is chosen in advance and its weights randomly initialised around the average of all input coordinates. A sample point s is extracted from the point cloud and the closest neuron n gets moved towards s. The neurons in the neighbourhood of n are also moved towards s, but to a lesser extent. Over time, the amount of movement and the neighbourhood radius are decreased to allow the SOM to become finely-tuned to the data.

Ordered and scattered points can be interpolated/approximated by the same type of surface with a wide variety of input conditions. The static size of the SOM can cause problems however; if too few neurons are used then some points lie away from the output grid, whereas if too many are used then speed suffers.

A dynamic neural network solves this problem by inserting new rows/-columns into the SOM where necessary until all points are interpolated or a predefined number of neurons are reached[102]. The number of training iterations required decreases dramatically as does the number of vertices in the grid. Insertion is not an expensive process and so the algorithm is actually faster than [42]!

Neural Networks

Martinetz *et. al.* used a 2D neural network to represent a surface[73]. A sample was extracted from the point cloud and used as input, and the neuron with the highest activation was trained. Its neighbours were also trained (albeit to a lesser degree), and Hebbian learning was used to create new synapses. When only the neuron with the highest activation was trained, the neurons specialised. When the points were sufficiently dense, the synapses that developed were the edges of the Delauney triangulation (the dual of the Voronoi diagram).

Barhak *et. al.* took a slightly different approach: they declared all neurons to be mobile or static, and active or inactive. A sample point is then taken from the point cloud, and the nearest neuron declared the winner. Once this neuron's position was updated, the position of its activated and mobile neighbours were also updated[7]. The boundary of the network was trained first by extracting only the outermost points from the point cloud. These boundary neurons were then declared static and the inner neurons trained using the remainder of the point cloud.

After a fixed number of steps the network is reparametrised, using the current parametrisation as the base for the next, in order to achieve more uniform parametric density. Unfortunately it is sensitive to the training parameters and can produce self-intersecting surfaces.

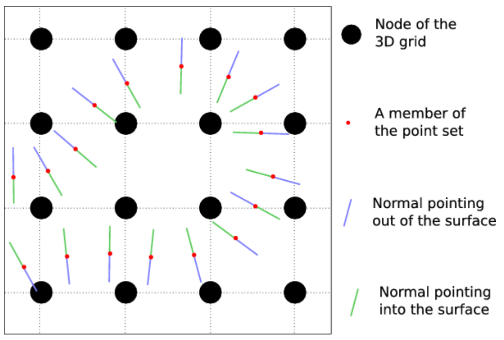
Node of the 3D grid
A member of the point set
Normal pointing out of the surface
Normal pointing into the surface

Fig. 2.15: A set of points (with normals) lying inside a 2D SOM.

Ivrissimtzis *et. al.* created a self-organising map with connectivity of a regular 3D grid where each of the nodes stored their signed distance from the surface[106]. The input is a set of points with normal data that are assumed to lie on the surface. The normals are used to assign values to the nodes that are representative of that node's distance from the surface. A 2D sample is show in figure 2.15.

Once trained, the grid therefore gives a discrete, implicit representation of the signed distance function from which a triangle mesh can be extracted using the Marching Cubes algorithm. The error is estimated by splitting the input points into two sets: one for training and one for validation. It is calculated periodically, and, if the ratio of current error to previous error is below a threshold, the algorithm stops. This procedure caught overfitting correctly without imposing severe time penalties.

In spite of the assumption that all the input points lie on the surface, the algorithm is noise tolerant because the nodes take values from several input points. Artefacts near sharp edges are avoided and a good level of accuracy can be achieved; better than partition of unity methods but worse than radial basis functions.

Growing Cell Structures

The predetermined, regular structure of an SOM can be limiting; in many cases a different shape can provide a more optimal solution. A growing cell structure can change its structure dynamically, allowing it to learn the most appropriate shape. In this case the neural network directly represents the mesh and so the terms mesh and network are used interchangeably, as are vertex and neuron.

In a simple example, the point cloud is sampled the neuron nearest to the sample moved slightly closer to it. Each neuron has an activity counter that is incremented every time it is the winning neuron, after which all activity counters decay slightly, so that more recent matches count for more than old ones[35]. The training is deemed complete when all neurons have equal probability for matching a randomly chosen input. After a predefined number of steps the neuron with the highest signal counter is split. After a few iterations the topology has been determined and the main source of change is adding new neurons, not moving old ones. Cells that are barely active are removed.

When growing cell structure surface reconstruction algorithms are in a training equilibrium, restricted Voronoi cells (the intersection of Voronoi cells with the surface) tend to have equal area, and the number of synapses converge[36]. Increasing the frequency of connectivity changes increased the overall error but distributed it more uniformly. Ensembles and forgetting were then used to further improve the learning.

The approach of Ivrissimztis *et. al* is similar to [35], but the learning rate can be modified[46]. A high rate increases mesh mobility but increases the likelihood of convergence to a local error minima. The vertex split distributes synapses evenly between the two resulting vertices and generalises to higher dimensions. Using an edge-collapse preserves topology, and allows for more aggressive removal of the least active vertices. In the meshes produced, 95% have a valency of 5, 6 or 7, which is useful when dealing with data from a scanned object. As the input surface is sampled, different inputs types (implicit, point cloud, . . .) can be treated uniformly and the running time is independent of the input size.

This was extended in [47] by the use of a non-constant counter decay: as the number of vertices increased, the signal counter decay slowed. Neurons were no longer removed only in a particular step, but whenever their signal counter fell below a threshold. Topology changes (from boundary merging and triangle removal) were performed rarely and with decreasing regularity. The modified algorithm stops when the mesh contains a certain number of vertices.

It is simple to increase the resolution of a growing network; one need only leave the algorithm running longer (perhaps decreasing the frequency of vertex removals). In contrast, an algorithm built around a static network would need to be re-run from the beginning, wasting time and computing resources. Growing cell structures are generally best used when the input's geometry and topology are unknown, since they will naturally learn them both.

2.4.6 Statistical Algorithms

The Bayesian Approach

Given a random variable X that takes values $\{x_0, x_1, \ldots, x_N\}$, the classical approach to statistics concerns itself with evaluating the probability that X takes the value x_i, denoted $P(X = x_i)$. This is an interpretation of probability as a relative frequency; the idea being that we can test X as often as we like and $P(X = x_i)$ is the relative frequency of $X = x_i$.

Using the timeless example of a biased coin toss, the classical approach is; "We have tossed the coin 100 times, it has turned up heads 75 times. It therefore appears that $p(\text{heads}) \approx 0.75$. If we continue to toss the coin then the relative frequency of heads converges to the probability of a heads being tossed."

$$\frac{\text{heads}}{\text{total tosses}} \to p(\text{heads}). \tag{2.24}$$

By contrast, the Bayesian approach interprets probabilities are a quantification of uncertainty. Rather than having a set of outcomes and approximating the probability of each, we take a set of results and use this to calculate the most likely probability distribution over the outcomes[9]. Informally, this could be phrased as: "given that this is the outcome, what is the likelihood of this probability distribution being correct?".

Using our coin example, the Bayesian approach says; "We have tossed the coin 100 times; it has turned up heads 75 times. Given this, what is the probability that $p(\text{heads}) = x$?" This is usually repeated in order to give a probability distribution over x. Given a set of outcomes D and a set of unknown parameters r, Bayes' Theorem states:

$$p(r|D) = \frac{p(D|r)p(r)}{p(D)} \tag{2.25}$$

Where $p(D|r)$ is the probability of the observed outcomes, given a particular set of parameter values. The $p(r)$ are called the prior probabilities, they encode our assumptions about the parameters before data are collected (*i.e.* without regard to D). The $p(D)$ term is the probability of the observed outcomes without regard to r. The $p(r|D)$ are dubbed the posterior probabilities, and are the probabilities of r, given the outcomes D. They are a quantification of our uncertainty in the parameters having observed the data.

Statistical Methods

In spite of the potential benefits, Bayesian methods are rarely directly employed in surface reconstruction. However they are often employed in related areas. For instance, Jenke *et. al.* took samples from a point cloud, added (Gaussian) noise and attempted to reconstruct the original point cloud using a Bayesian method[49]. In their method, the surface is assumed to consist of piecewise smooth patches connected by sharp boundaries; an assumption that works well for man-made objects, but natural objects are not so cleanly constructed. Prior probabilities were used to identify which artefacts are taken to be noise; one for density, one for smoothness and one for estimating sharp features.

The density prior is used to estimate the surface area, which in turn allows an expected distance between points to be computed. Minor holes can be filled automatically, but above a noise threshold it is impossible to identify edges of the point cloud. The algorithm is slow, but robust, as objects of arbitrary topology can be reconstructed.

Schall *et. al.* defined a set of local functions that give the likelihood that a point lies on the surface[94]. The maxima of the likelihood functions were found using a method akin to gradient-ascent (find the direction of greatest increase then move in that direction) with an adaptive step size.

Each point in the kernel of a likelihood function (its most fundamental expression) is then moved to the area of maximum likelihood: points corrupted by noise are “pulled back” into the most likely correct position. Outliers converge to a set of isolated points lying away from the surface, which are easy to remove by thresholding due to their very low sampling density. In this way the point cloud is cleaned and filtered, and noise-sensitive algorithms (such as Delauney-based ones) were shown to benefit from this.

Ensemble Techniques

An ensemble is a collection of objects, each of which is assigned a probability. In this context the objects are surfaces and the probability indicates the chance of the surface being an accurate reconstruction. The basic ensemble technique is to run a probabilistic algorithm on a set of data many times, putting the outputs into an ensemble, and then to combine the outputs into a single model[65]. This approach can be applied quite generally. For example, Ivrissimtzis *et. al.* used ensembles of neural meshes to make the reconstruction robust against noise[48].

The recombination of candidate surfaces into a final reconstruction requires a good averaging method. If a supervised recombination is not possible or desirable then the mean of several surface positions can be taken. Taking the mean of surface positions that are close to the median was been shown to be more robust than a simple mean over all the positions[48]. To estimate normals the tangent plane of each point must be estimated, then a consistent orientation of tangent planes must be determined.

To create an ensemble, random samples of the initial dataset are taken (with overlapping permitted), each of which is then run through a deterministic algorithm. The outputs are combined using an averaging method and the Marching Cubes algorithm used to construct a triangle mesh. To reconstruct the normals the same method is used, with the exception that the subsets are required, not just allowed, to overlap (to get the consistent orientation).

As the number of samples in each subset increases, error and speed decrease, regardless of initial noise[108]. Unfortunately due to the way the error is formed, an ensemble method can only reduce (but not eliminate) errors. The combination of normal and surface ensemble technique was shown to be very effective.

2.4.7 Other Techniques

Surface reconstruction is still an active research area, with many algorithms employing a variety of geometric, statistical and signal theoretic techniques. Nehab *et. al.* combine separately acquired positional and normal information[78]. Kil *et. al.* process dense point sets obtained from multiple scans using a variant of the image processing technique of super-resolution[59]. A Poisson equation can also be solved to reconstruct a surface[57, 11], which is used for comparison in chapters 4 and 6.

2.5 Post-Processing

2.5.1 Feature Detection and Extraction

Feature detection is a problem closely related to surface reconstruction, and the two will be examined together in chapter 5. In many cases, the two problems are solved concurrently by a feature-preserving surface reconstruction algorithm[33]. However, features can also be detected on the input point set as part of a pre-processing analysis[85], or on the reconstructed surface as a post-processing analysis of the obtained model [109].

There are three classes of feature that may be of interest to extract or identify; low–, mid– and high–level[37]. Low-level features are pixel-properties such as colour or texture. Mid-level features are geometric, and are the most relevant to this thesis, and include ridges, corners and points. High-level (semantic) features are those that require additional context/information to interpret.

Gumhold *et. al.* extracted features as a pre-processing step[39]. They created a nearest-neighbour graph and assigned to each edge a probability that described how unlikely it was to make up a feature. A sub-graph was created of probable-feature edges and this was filtered to leave only the edges most likely to constitute surface features.

The likelihood of a particular structure being a surface feature (as opposed to an artefact) is influenced by various factors[50] such as the difference in facet normals (in the case of ridges) and the number of valent edge-features (in the case of vertices). These factors are often in competition with one another and so thresholds are often used to select the most probable structures.

Ohtake *et. al.* developed a post-processing method of feature detection[79]. Radial-basis functions with compact support (i.e. that are 0 outside their domain) are used to reconstruct the surface, then the vertices of the mesh are projected onto the reconstructed surface. The first- and second-order derivatives of the curvature are then calculated for each mesh vertex, and curvature maxima and minima are detected along the edges.

Pauly *et. al.* used the idea of surface variation to estimate features[85]. The surface variation is defined via the principal components, like so:

$$\sigma = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3,} \text{ where } \lambda_1 < \lambda_2 < \lambda_3. \tag{2.26}$$

The scale on which to look for features is a provided as a manual input, which is translated into the number of neighbours used in the calculations. To look for features on a large scale, a larger number of neighbours are included, the converse is also true. The more neighbours of a point with a surface variation above a threshold, the higher the confidence that the point belongs to part of a feature.

All the points that are highly likely to constitute features are then connected via a minimum spanning tree. Isolate clusters are joined by those points that have a relatively high feature confidence (but were not included in the feature-point set). The tree is then cleaned, and smoothed to produce a final feature map.

2.5.2 Mesh Alignment

It is often useful to compare two meshes, two point clouds, or a point cloud and a mesh. For example, in shape recognition one might compare two meshes or two point clouds asking whether they belong to the same category. Such a comparison of usually starts with the alignment of the meshes, which will be investigated in chapter 6.

It should be noted that the exact way in which two meshes are best aligned is not clearly defined. In fact, the best alignment can be application-dependent, and mesh alignment should be seen as an ill-posed problem. Nevertheless, a good alignment algorithm is expected to be able to align a mesh with a version that has undergone common mesh processing operations such as smoothing, simplification or remeshing.

Alignment is typically done by computing a translation, a scaling and a rotation, which are then applied to one of the meshes to align it with the other. The computation of the translation is usually done by aligning barycentres, while the scaling is done by aligning bounding boxes or bounding spheres [96]. Translation and scaling are both considered less challenging to compute than the rotation. In some fields such as medical imaging, the registration process requires more than this simple pose normalisation, for instance, an alignment between certain parts of the two models [87].

Most of the work on mesh alignment focuses on and enhances the *Iterative Closest Point* (ICP) algorithm, which takes a set of points common to both input meshes and iteratively rotates the mesh until the common points are aligned as closely as possible. A variety of modifications and enhancements to the original ICP algorithm have been developed and studied. For instance, to make it geometrically stable[38] and to make it work with approximate nearest neighbouring points or with added noise[72]. Other ICP based methods require an explicitly defined initial guess, which prevents the method being used in a completely automated manner[8].

In reality, many practitioners use PCA directly on the vertices of the input meshes in order to provide an alignment that is good enough to work with. PCA is efficient, since it is essentially a quadratic optimisation problem based on variance maximisation. A technique that is similar in spirit, called Independent Component Analysis (ICA)[44], is based on quartic optimisation and has been used for 3D object recognition[92].

Despite its popularity, PCA has been reported to perform poorly when aligning meshes for 3D model recognition and this has been cited as motivation for developing rotationally invariant mesh descriptors[58]. Nevertheless, several important shape descriptors, such as[17], shape histograms[4] and descriptors based on higher order moments[31], are not rotationally invariant and thus require alignment.

Extensions to PCA to overcome its shortcomings include PCA performed on the normals of a surface[83], and a continuous version of PCA applied to whole mesh triangles rather than just their vertices[103]. The latter is independent from the distribution of vertices within the mesh and thus, overcomes some of the limitations of PCA the same way the method proposed here does. However, it requires a triangle mesh as input and has no obvious extensions to point clouds.

2.5.3 Mesh Quality

Yu *et. al.* improved mesh quality in a post-processing step by exploiting the fact that given any two triangular meshes there are only three operations needed to transform one into the other: edge collapse, edge swap and edge split[110].

If the meshes are both 2D manifolds with the same number of edges and vertices then only the edge swap is needed. When scanning objects, concave features can cause long, thin triangles to be formed by the reconstruction algorithm. These triangles are detected and replaced via an edge-swap.

A measure, “deviation”, is defined to determine which edge to swap, with swaps being accepted if the new edge has a lower deviation than the original. Two types of swap are tried before abandoning the attempt: single swap and, should this prove unsuccessful, double swap. A double swap is simply two consecutive single swaps where both the swapped edges belong to the same original triangle.

3. SURFACE REPRESENTATIONS

3.1 Mathematical Definition

Intuitively, a surface is an object that, when viewed sufficiently closely, looks like a two-dimensional plane. Formally, a surface is an object for which the neighbourhood of every point is the image of a smooth map from an open subset of $\mathbf{R}^2$[69]. This means that it can have no self-intersections (as expected), but also that there may not be any discontinuities. Examples of surfaces are hollow spheres and tori.

Examples of non-surfaces are a hollow cylinders, hollow cones and 2D curves. The hollow cylinder fails because of the circular edges at each end. If we look at a point lying on this edge, its neighbourhood is not similar to a plane; discrete operations are required to transform this neighbourhood into a plane. Similarly, a hollow cone fails for both its base edge and its tip; try as one might, there is no continuous operation to map the neighbourhood of the tip to the plane. A curve is an intrinsically one-dimensional object, and so cannot look like a plane anywhere. It is also clear that any object incorporating a non-surface entity, cannot itself be a surface.

In this thesis, an object will be considered a surface if its constituent components each meet the mathematical definition of a surface. Using this definition a hollow cylinder is a surface, since the curved edge and two circular ends all meet the mathematical definition of a surface.

3.2 Explicit Representations

Explicit/parametric surface representations are a map f operating on a domain $A \in \mathbf{R}^2$ such that $f : A \to B \in \mathbf{R}^3$. For all but the simplest of surfaces, this would be an immensely complex (if not impossible) task. Consequently the domain A is divided into N subdomains, each with a corresponding map (dubbed *patches*).

In order to allow efficient processing the patches are generally taken to be polynomials, since these may be calculated using elementary methods and can approximate any smooth function to any desired precision (provided they are of sufficiently high order). Given an infinitely differentiable function, it can be approximated via a polynomial of degree p, with intervals of length h that has an approximation error of the order h^{p+1}.

Given this (and that $h < 1$) there are two apparent methods for improving the approximation. The order of the polynomial, p, can be increased, or the interval size, h, can be decreased (and more intervals used in total). This also applies for polynomials of more than one variable, where h is simply promoted $h \to \mathbf{h}$).

It is very unusual to increase p to improve the accuracy for two reasons: 1) the continuity conditions between the surface patches can become quite difficult to satisfy at higher orders, and 2) it is often more efficient to perform a large number of simple calculations than a smaller number of complex ones[15]. By far the most commonly used representations are piecewise linear, which requires only that neighbouring surface patches meet, and the most common of these is the triangle mesh, which will be assumed from now on. A good trade-off between accuracy and speed can be obtained by varying the density of the triangles according to the curvature of the surface. Areas with low curvature need relatively few triangles to describe them to the same level of accuracy as an area with wildly varying curvature.

When dealing with scanned objects it is often considered bad to have a large number of long, thin triangles; something that is often indicative of trying to reconstruct an object from a sparse set of data.

Explicit representations are useful because they can be modified with relative ease and are easily rendered. Unfortunately, finding out whether a point is inside or outside the surface is a costly operation, as is working out its distance from the surface. Collision detection (finding out if one part of the surface meets another) is also computationally expensive.

3.3 Implicit Representations

To represent a surface implicitly, a function is defined that maps each point in 3D space to a real number. Any point that is mapped to a negative number is defined to be inside the surface, and any point mapped to a positive number; outside. If a point is mapped to 0, then it lies precisely on the surface. Formally:

$$f : \mathbf{R}^3 \to \mathbf{R} \tag{3.1}$$

$$S = \{\mathbf{x} \in \mathbf{R}^3 : f(\mathbf{x}) = 0\} \tag{3.2}$$

$$O = \{\mathbf{x} \in \mathbf{R}^3 : f(\mathbf{x}) > 0\} \tag{3.3}$$

$$I = \{\mathbf{x} \in \mathbf{R}^3 : f(\mathbf{x}) < 0\} \tag{3.4}$$

The set S represents the surface itself, O the space outside the surface, and I the space contained inside the surface. It is important to note that given an appropriate definition of f these sets will partition $\mathbf{R}^3$. There is no ambiguity since, by the Intermediate Value Theorem, S will always lie between O and I and no point can belong to more than one set. It is simple to see that the choice of 0 is arbitrary in the above definitions, and that any level set would satisfy these criteria.

Given that f tells us whether a point is inside or outside the surface the most natural choice of f is a signed distance function; a function defined such that the value of $f(\mathbf{x})$ is the distance of $\mathbf{x}$ from the surface. Even if 0 is not being used as the level set, then a trivial calculation will allow values of $f(\mathbf{x})$ to be mapped to distances from the surface.

Due to the extreme difficulty of finding a mathematical description that matches the surface with sufficient accuracy, the domain of f is divided into subdomains, with each subdomain A_N having a corresponding function f_N. In order to maintain a smooth surface across subdomain boundaries, a weighted sum of the contributing functions is often used.

If a partition has few or no sharp features then it is often inefficient to sub-partition it as finely as areas with many sharp features. Using such an adaptive partitioning scheme can lead to significant memory savings[13].

Implicit surface representations do not have the same flaws as explicit representations; inside/outside calculations are trivial and distance calculations are dramatically simplified. Unfortunately, they are not so easily rendered and are unable to reproduce boundaries such as the edge of a hemisphere.

3.4 Conversions

A mesh conversion involves moving between an explicit representation and an implicit one. There are many well-established algorithms for such conversions, but the process necessarily involves the loss of information.

3.4.1 Explicit to Implicit

Conversion from an explicit to an implicit representation amounts to approximating the signed distance function of the surface. This involves calculating the nearest triangle to a given point, then calculating the point's distance from that triangle. It must also be determined whether a point is inside or outside the surface. This results in a piecewise linear distance field, which, whilst not the most accurate, is certainly sufficient for most needs (since the signed distance field of an implicit model is not always smooth).

Given a point p near the surface, whose closest point on the surface is c, which lies in a triangle with normal n, the angle between $(p - c)$ and n can be used to compute the signed distance. Unfortunately this method is susceptible to noise in the normal data as a misaligned normal would at best cause an inaccurate distance to be calculated. In the worst case would this could place a point on the wrong side of the surface, resulting in the formation of artefacts (spurious features).

3.4.2 Implicit to Explicit

This conversion is properly called *isosurface extraction* since it extracts the level set (isosurface) of the implicit function f. The *de-facto* standard algorithm is Marching Cubes[71], which uniformly divides the signed distance field into a regular grid. It "marches" through all the cubes in the grid, performing some simple operations to work out approximately where the surface lies.

To determine the location of the surface within a cube, f is evaluated at each of the eight corners. If the sign of f differs between two adjacent corners then it follows that the surface must pass between them. Triangles are then created that partition the cube into internal and external corners.

4. MEMORY–EFFICIENT SURFACE RECONSTRUCTION

This chapter is based on material originally published in the 2011 proceedings of the Theory and Practice of Computer Graphics[56].

We propose a memory efficient, scalable surface reconstruction algorithm based on self organising maps (SOMs). Following previous approaches to SOM based implicit surface reconstruction, the proposed SOM has the geometry of a regular grid and is trained with point samples extracted along the normals of the input data. The layer-by-layer training of the SOM makes the algorithm memory efficient and scalable as at no stage there is need to hold the entire SOM in memory. Experiments show that the proposed algorithm can support the training of the very large SOMs that are needed for richly detailed surface reconstructions.

The algorithm presented in this chapter works in a similar manner to SOMs. Unlike traditional SOMs, the nodes themselves to not move, however, they still work to minimise an energy function. In this case the energy function is the second derivative of the gradient near the surface. That is, the nodes update their estimate of the distance to the surface in order to minimise the discontinuity in the gradient near the surface.

4.1 Introduction

The popularity of implicit surface reconstruction algorithms is in no small part due to their robustness. Indeed, implicit methods seem to be particularly well suited to deal with the noisy, unevenly sampled point sets that are the typical outputs of optical scanning devices. Moreover, intensive research activity on implicit methods has yielded some very fast, computationally efficient algorithms.

On the other hand, the extra third dimension of the implicit surface representation may increase the memory requirements of an implicit surface reconstruction algorithm. Memory efficiency problems are dealt with by employing flexible data structures, such as adaptive octrees, however, these complicate the algorithms and increase the implementation overheads. A second drawback of the implicit approach is that the required global optimisation may affect the scalability of the algorithm. Scalability issues are ameliorated by making the locally fitting implicit functions have compact support. However, even though their compact support means that, in principle, the global optimisation problem can be solved locally, in a small neighbourhood of the data, it is nontrivial to implement this in a computationally efficient way.

In this paper we extend the work in [106, 54], proposing an implicit reconstruction algorithm based on a self organising map. The SOM has the connectivity of a regular 3D grid. Its nodes can be seen as a regular, discrete sample from the inside of a bounding box of the input point set. Each node stores a scalar value representing the signed distance between the node and the surface, and it is trained with data sampled from the normals of the input point set.

The proposed algorithm extends the work in [54] by having the size of the SOM adapt itself to the data provided. Most importantly, the SOM is trained layer by layer, and never stored entirely in memory at any given time, see Fig. 4.1. Given the ordered, rather than random nature of the training, fewer samples need to be taken from the point cloud to ensure a smooth reconstruction. While the work presented in [54] was able to handle large quantities of input data by sampling it, rather than processing it globally, storing the SOM itself entirely in memory could give rise to problems with scalability.

Any trained layer can be passed directly to the Marching Cubes algorithm for isosurface extraction or saved to disk without needing to wait for the completion of the SOM training. As a result, the algorithm is memory efficient without needing an adaptive data structure, and it is scalable without needing a technically involved localisation of a global optimisation problem. Taking this approach one step further, each layer of the SOM can also be trained in stages, in this case line by line, leading to further memory efficiencies at the expense of higher computational

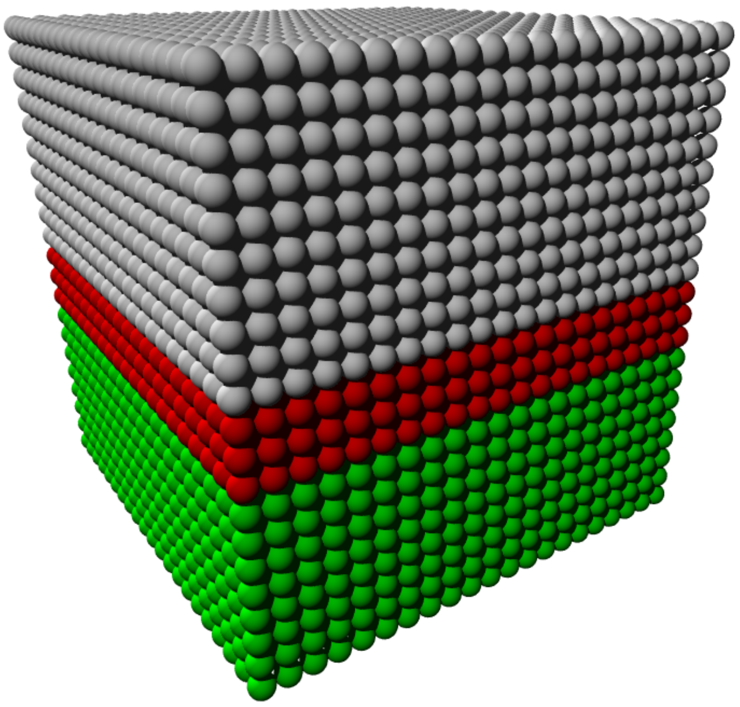

Fig. 4.1: The SOM is trained layer by layer, starting from the bottom and going up. The already-trained nodes are shown in green. The nodes currently being trained are shown in red. The nodes to be trained are shown in grey.

costs.

4.2 Layered Algorithm

In this section we describe the main algorithm and discuss some implementation details. The input of the algorithm is a point set with normals.

The SOM is arranged in the form of a regular 3D grid with the nodes on the lattice $\mathbf{Z}^3$. That is, each node has integer coordinates and the length of each edge is 1. Each node stores an estimation of its signed distance from the surface, $\bar{d}$. The edges provide no information and can be completely ignored since neighbourhood relations for the nodes of the grid are obtained by direct means such as distances between nodes.

The SOM has a band of active layers, in which each node stores a list L of weighted distances from the surface¡ obtained through training. The distance estimation, $\bar{d}$, is computed as a weighted average of the elements of L and represents the current estimate for the value of the implicit function at that node. Only nodes in this active band are trained. The active band moves from the base to the top of the SOM, training it.

When fully trained, the SOM represents a discrete implicit description of the surface that can be triangulated using the Marching Cubes algorithm [71]. Even though each node only passes its value $\bar{d}$ to the Marching Cubes algorithm, the list of weighted distances L provides information about earlier states of the SOM, which can be used for fine-tuning or analysis of the results, as demonstrated in [54]. It also provides some robustness to noisy input data or misaligned normals of the type that are common when processing data from optical scanners.

4.2.1 Data Alignment and Sorting

In the first step of the algorithm, we find a tight rectangular bounding box for the input point set and align it to the SOM. We perform PCA on the input data and use the three principal components as the axes of the box. By an affine transformation followed by scaling, we map the bounding box and the data into the convex hull of the SOM grid. In the labelling of the axes we choose the z-axis to be the the largest principal component ensuring that the base of the SOM is as small as possible, affording us the smallest memory footprint. The point cloud is analysed and its maximal and minimal x and y values found, and the SOM then configures its size accordingly. Finally, the point set is sorted by z-coordinate in ascending order.

4.2.2 Training Step

The basic training step of the SOM runs as follows:

1. A sample point, s, is extracted from the point set.
2. Nine training data are created as shown in Fig. 4.2, which extend a distance of ± 2 units from s. These training data store their distance from the sample, d_s, and a corresponding weight computed with equation 4.1.
3. For each training point, the nearest SOM node is found. This node has the weight and distance of the training point added to its list of distances, L.

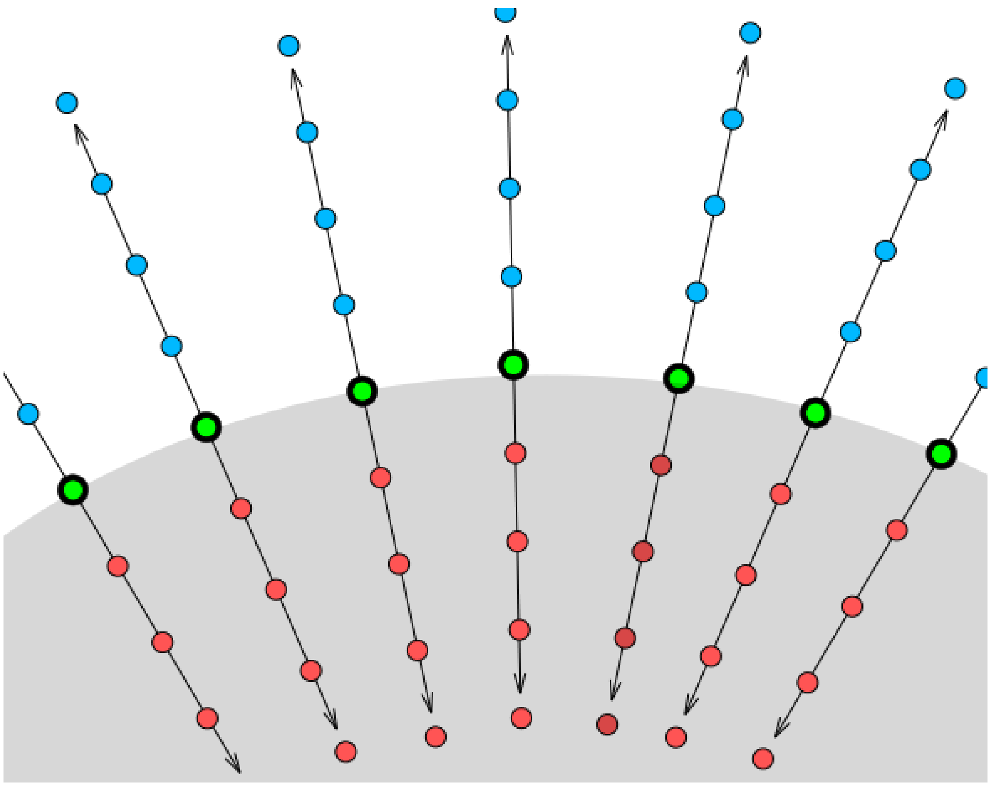

Fig. 4.2: The grey area represents the interior of the surface. Red points are training data with negative distances from the surface, blue points are those with positive distances, and the green points have distance zero.

The weight of a training point represents our confidence that its distance from the surface is accurate. This might not be the case if two areas of the surface are close to each other, or if there is another sample point closer to the training point than s. Other methods of computing the weight were tried, but the equation 4.1 was found to give good results whilst remaining computationally inexpensive.

$$w(d_s) = \frac{1}{1 + d_s^2} \tag{4.1}$$

Samples are extracted sequentially and are assumed to lie on the surface, *i.e.* we assume that their distance from the surface is 0.

4.2.3 Separation Calculation

First, the separation (estimate of signed distance to the surface) of each node is computed by calculating the weighted mean of all the distances in its training history.

$$\text{separation} = (\sum_i w_i d_i)/(\sum_i w_i). \tag{4.2}$$

If the node has had its separation calculated previously, its separation is updated to

$$\sigma_u = a\sigma_c + (1-a)\sigma_n \tag{4.3}$$

Where σ_c is the node's current separation, σ_n is the newly-computed separation. If the node has not had its separation computed previously, then it is set to the value computed in equation 4.2. The speed of learning is controlled by a, the learning rate parameter, where $(0 < a \leq 1)$. A higher value increases the training speed, but makes it more susceptible to corruption by noise. A lower value trains the SOM more slowly, and favours slow convergence to a single value.

4.2.4 Smoothing

We define a value z_s, which initially holds the z-coordinate of the first sample taken. In subsequent sample extractions, if the new sample's z-coordinate is greater than z_s plus a predefined threshold, z_s is updated to this value and the SOM is smoothed. We continue use the natural L_0 metric of the grid, where $L_0(n)$ denotes nodes with an L_0 distance of exactly n units from a given point.

The nodes to be smoothed are then subjected to the following procedure:

1. The $L_0(1)$ neighbours of node n are found. If n has no trained neighbours, it is not smoothed. Otherwise, the mean of the $\bar{d}$'s of the trained neighbours, m_1, is computed.

2. Similarly to above, the $L_0(2)$ neighbours of node n are found. If there are less than two trained $L_0(2)$ neighbours then n is not smoothed. Otherwise, the mean of the trained neighbours, m_2 is computed.

3. Finally, n has the distance $0.65m_1+0.35m_2$ added to its distance list with a weight of 1.0. Weights of 0.65 and 0.35 were experimentally determined to work well for a variety of data.

The bottom two layers of the active band are not smoothed because the computed distance would not include contributions from the $L_0(1)$ and $L_0(2)$ neighbours with the lowest z-coordinates and so would lead to biased smoothing. The smoothing is also not applied to layers within 2 units of z_s, since these likely still trained directly by new data. Similarly to the restriction at the bottom of the active band: layers very close to the top would be unlikely to have trained upper neighbours, which could lead to biased smoothing.

4.2.5 Storing

If the z-coordinate of any sample point is within 2 units of the top of the active band then it triggers a dumping of the bottom 2 layers. The data for these layers (node coordinates and their separation, $\bar{d}$) are then saved to a file, but could be passed directly to the Marching Cubes algorithm. Following this, the active band moves 2 layers in the positive z direction. The memory for the two formerly active layers is then freed, helping to keep the memory consumption within reasonable bounds.

4.2.6 Parameter Choice

Setting the height of the active band to 20 nodes results in good quality surfaces without using large amounts of memory. Different values of the learning rate parameter a in Eq. 4.2 were tested and a value of 0.9 was found to give a good balance between convergence speed and numerical stability.

The training data extend 2 units from their sample point in the direction of the normal, and in the same distance the opposite direction. This distance was chosen because only nodes close to the surface will have any effect on its geometry when the Marching Cubes is run. Training nodes further away would therefore increase the memory consumption for no benefit. This is also the distance to the top of the active band that triggers storing since to have a sample closer than 2 units to the top of the active band would mean its training data extending beyond the top.

Nine training data were created per sample, and spanned a range of ± 2 units from each sample point. This was to ensure that nodes were consistently trained but not over-trained. A more sparse set of training data would lead to gaps (and thus inconsistent distances) and a denser set would result in each node being trained with multiple inconsistent distances.

The length of the weighted distance list is constrained to provide a bound on the memory that can be used. Each node can store a maximum of 100 weighted distances. This was chosen to be long enough to tolerate noise (because the effect of the other distances dwarfs that of the noise) but short enough to keep the memory footprint within sensible limits.

4.3 Results

To validate the results, two algorithms were selected for comparison. The first was proposed in [54], it is similar, but has some notable differences. For instance, it does no preprocessing on the data in order to determine the optimal size of the SOM or sort the data. Instead, it keeps the entire SOM in memory and samples the data many times, relying on an overfitting heuristic to terminate the process. The second was the commonly-used Poisson reconstruction[57]. It should be noted that [57] produced smoother meshes than the proposed algorithm and [54], but this is normal for Poisson reconstructions.

We first validated the proposed algorithm by testing it on point sets obtained by stripping the connectivity from smooth meshes. We used the neptune, turbine blade, happy buddha and dragon meshes. By comparing the re-reconstructed meshes with the original meshes (which serve as the ground truth for the underlying surface of the point data) we are able to gauge the accuracy of the method. Figure 4.3 shows the obtained reconstructions and Figure 4.5 shows close-ups of the reconstructions.

Next, we tested the surface reconstruction algorithm on unprocessed point sets from raw range scan data, in particular the Bunny data from Stanford repository and the Ramesses data from AIM@SHAPE. The normals for the Ramesses model were computed from the raw mesh also provided by AIM@SHAPE (using MeshLab) as the weighted average of incident face normals. Figure 4.6 shows the obtained reconstructions. Figure 4.7 shows a close-up of the Ramesses reconstruction.

To validate the memory efficiency claim, memory consumption was monitored during the reconstruction of the models (whose sizes are shown in Table 4.1). The average and peak memory use for each model are displayed in Tables 4.2 and 4.3 respectively. A $\times$ in any table indicates that the algorithm was not able to run to completion on a PC with 4GB of RAM.

A detailed breakdown of timings for the algorithm is shown in Table 4.4 and the total run-time for each model and method are shown in Table 4.5. The Marching Cubes implementation used was not able to extract the isosurface of the huge Neptune model due to insufficient memory. The number of triangles in each model after reconstruction by each method is shown in Table 4.6.

The Neptune model was reconstructed at a variety of scales, with the timing recorded. The results are shown in Figure 4.8 as a function of the volume of the point cloud's bounding box, or equivalently, the number of SOM nodes. All other parameters were kept constant for these reconstructions to ensure that only the scale affected the results. As can be seen, the timing scales almost linearly with the volume of the bounding box.

model	bounding box	points
buddha	$140 \times 122 \times 300$	543'652
dragon	$185 \times 235 \times 299$	437'645
turbine	$495 \times 463 \times 598$	882'954
neptune	$302 \times 694 \times 1001$	2'003'931
huge neptune	$2112 \times 4858 \times 7004$	2'003'931
bunny scans	$130 \times 209 \times 210$	362'272
ramesses	$224 \times 318 \times 645$	826'266

Tab. 4.1: The size of each point cloud's bounding box in the x, y and z directions, along with the number of points in each cloud.

As can be see in table 4.2, our results compare very favourably to the the alternative algorithms. For instance, the worst case comparison is against Ohtake *et. al.*, which peaks at using more than five times the memory of the proposed algorithm. The performance is significantly better than [54], which kept the whole SOM in memory, so the larger

model	average (proposed)	average [54]	average [57]
buddha	16	933	173
dragon	20	1260	81
turbine	48	×	144
neptune	54	×	205
huge neptune	1633	×	×
bunny scans	18	623	73
ramesses	29	×	19

Tab. 4.2: The average memory use of the layer by layer reconstructions in Megabytes.

model	peak (proposed)	peak [54]	peak [57]	peak [80]	peak [18]
buddha	23	1209	173	442	291
dragon	24	1721	253	210	306
turbine	71	×	384	-	-
neptune	112	×	312	-	-
huge neptune	1772	×	×	-	-
bunny scans	20	885	178	110	-
ramesses	38	×	90	-	-

Tab. 4.3: The peak memory use of the layer by layer reconstructions in Megabytes. A dash indicates that the data was not provided by the paper in which the algorithm was proposed.

differences here are to be expected. Excluding these results however, the algorithm is favourable in terms of resources used when compared against the alternatives, and differences of an order of magnitude can see seen in several places.

4.4 Line-by-Line SOM training

If further memory efficiency is required then the SOM can be modified to be trained line by line, as shown in Figure 4.9. After the initial pre-processing and sorting, the points within the range $(z, z+1)$, for integral z, are sorted in ascending order (left to right) by their y-coordinate. In this case the active band becomes an active line, which has fixed size in both the y and z directions.

model	pre-processing time (s)	training time (s)	polygonisation time (s)
buddha	12	45	5
dragon	10	38	6
turbine	19	227	40
neptune	50	302	57
huge neptune	51	∼8 hrs	×
bunny scans	9	21	3
ramesses	20	120	19

Tab. 4.4: Timings for the different stages of the proposed layer by layer algorithm.

model	recon. (s) proposed	recon. (s) [54]	recon. (s) [57]
buddha	62	142	163
dragon	54	159	220
turbine	286	×	366
neptune	409	×	400
bunny scans	33	98	50
ramesses	159	×	39

Tab. 4.5: The total run-time for each algorithm, including any pre-processing and isosurface extraction.

4.4.1 Implementation

Inactive nodes are stored in temporary files, and like the layer-by-layer reconstruction, cannot be trained. If the z-coordinate of any sample point is too close to the top of the SOM, the SOM dumps these layers and moves in the positive z direction.

Similarly, if the y-coordinate of any sample point is too close to the rightmost edge of the active line, the leftmost 2 rows of nodes are stored in temporary files (including their distance list) and the active line moves to the right. When the SOM has been trained with all the sample points in a layer (indicated by $y_{n+1} < y_n$) the SOM stores the current state of all active nodes in the temporary files and the active line jumps back to the left.

The temporary files are read to determine the state of the nodes when they were last active. The training then continues as before, but whenever the SOM moves right, it reads the states of the now-active

model	triangles (proposed)	triangles [54]	triangles [57]
buddha	182’421	343’501	629’208
dragon	247’751	436’873	856’976
turbine	1’600’242	×	1’359’064
neptune	1’070’264	×	1’403’528
bunny scans	128’884	221’166	211’930
ramesses	577’923	×	111’980

Tab. 4.6: The number of triangles for each method.

nodes from the temporary files.

Using an active line with a height and width of 30 nodes resulted in good quality surfaces and low memory use. A larger value was used compared to the height of the active band in section 4.2 to take into account that smoothing the active line would propagate the training information less than the active band. Smoothing can be triggered by z-coordinate changes, as in the layer-by-layer reconstruction, or by analogous y-coordinate changes.

4.4.2 Results

The current implementation of the line-by-line SOM training is basic and has not been tested on large input data sets. However, proof of concept results on small data sets show significant memory savings. For example; when the Ramesses model (scaled to $112 \times 159 \times 322$) was reconstructed, the mean and peak memory consumption were only 2.4MB and 2.5MB respectively. On the other hand, the time taken to complete the reconstruction was 44 minutes (148’495 triangles). The reconstructed model is shown in Figure 4.10.

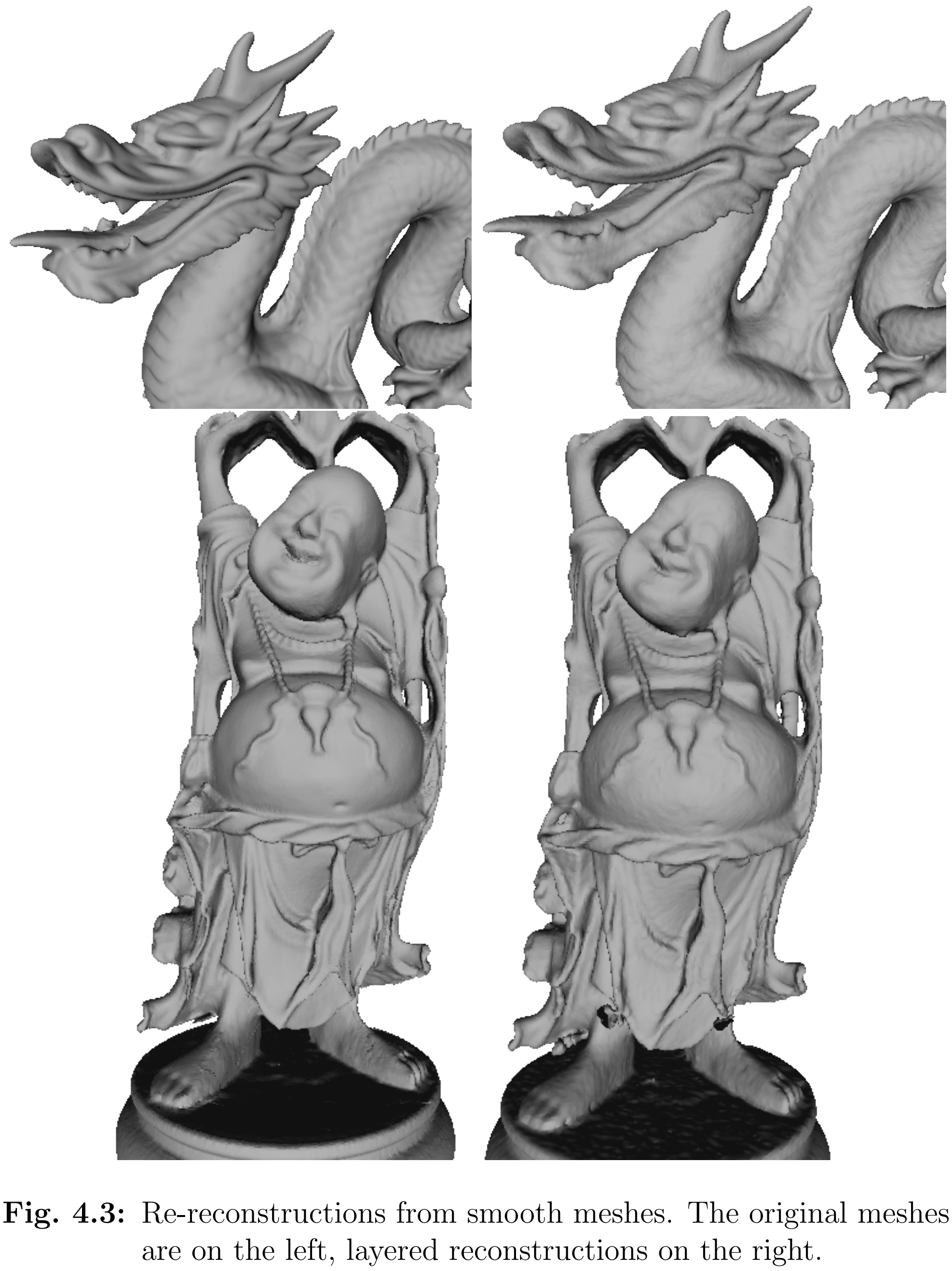

Fig. 4.3: Re-reconstructions from smooth meshes. The original meshes are on the left, layered reconstructions on the right.

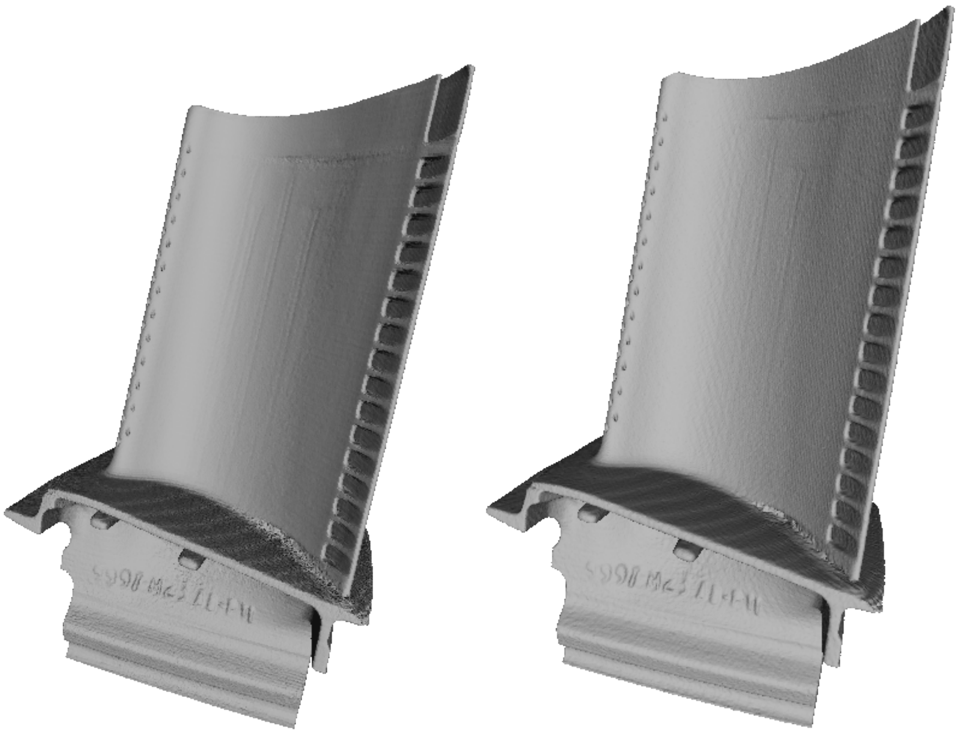

Fig. 4.4: Re-reconstructions from a smooth mesh. The original mesh is on the left, the layered reconstruction on the right.

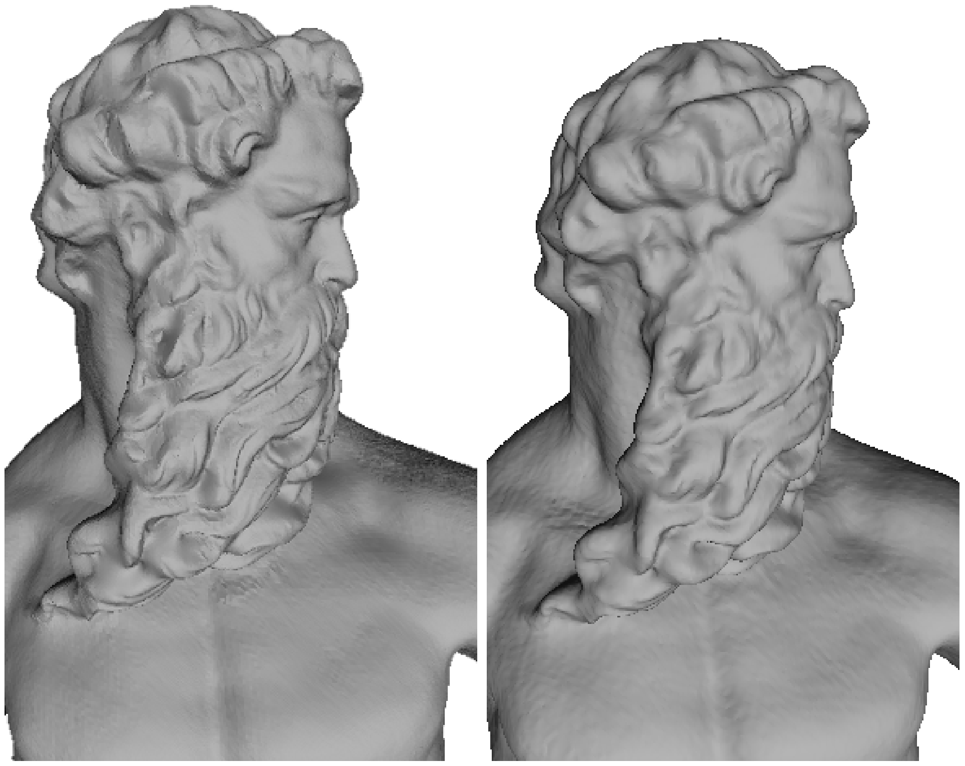

Fig. 4.5: Close-up of Neptune's face, layered reconstructed from a mesh.

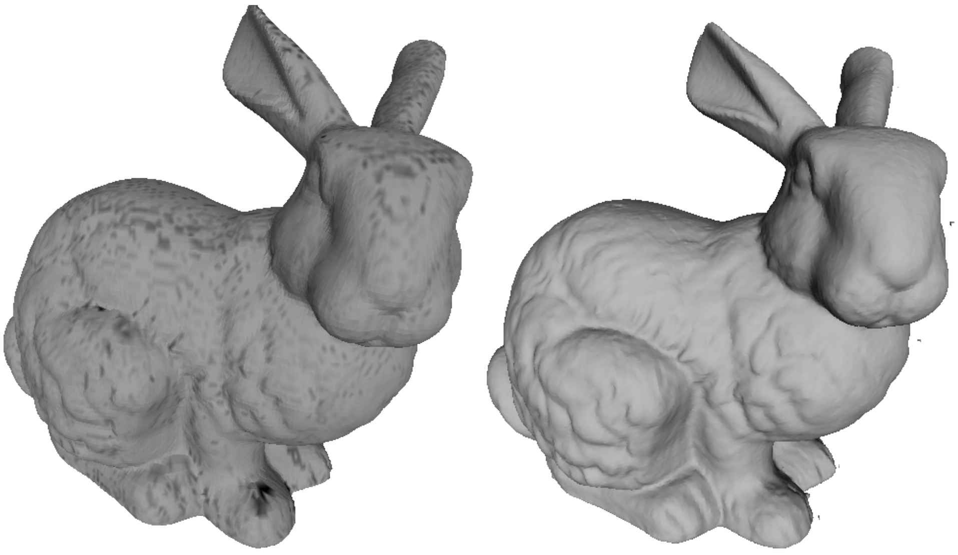

Fig. 4.6: Reconstruction from scan data. The mesh supplied from the Stanford 3D Scanning Repository is on the left, the layered reconstruction is on the right.

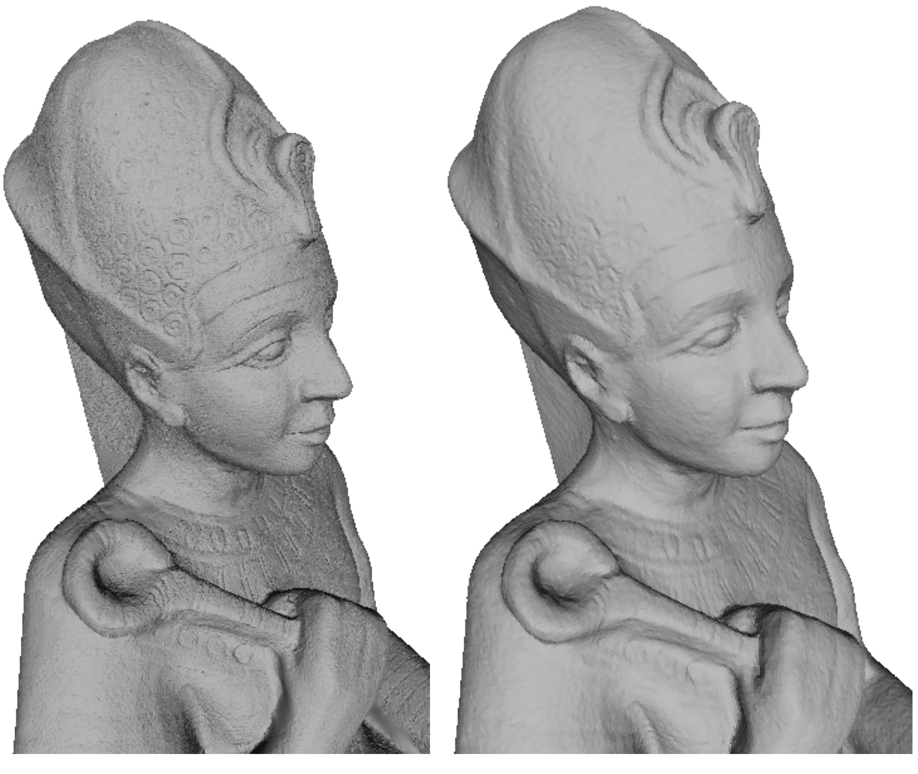

Fig. 4.7: Reconstruction from scan data. The mesh supplied from AIM@SHAPE is on the left, the layered reconstruction on the right.

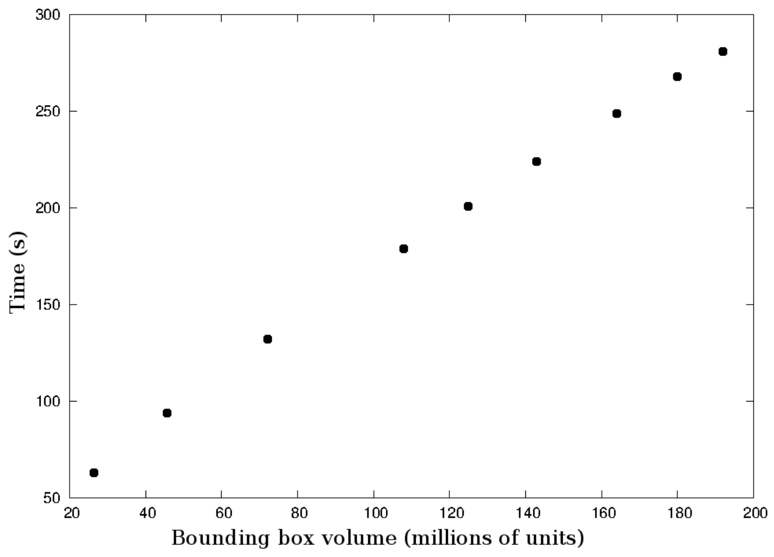

Fig. 4.8: Time taken to reconstruct the neptune model by layers vs. the volume of its bounding box.

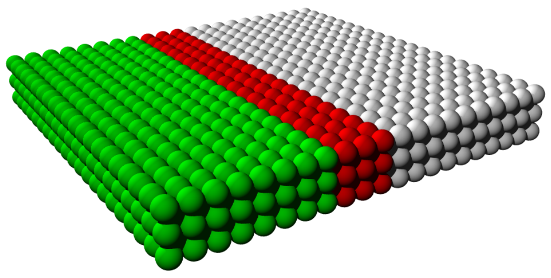

Fig. 4.9: In a recursive application of the layer by layer training principle, a layer can be trained line by line.

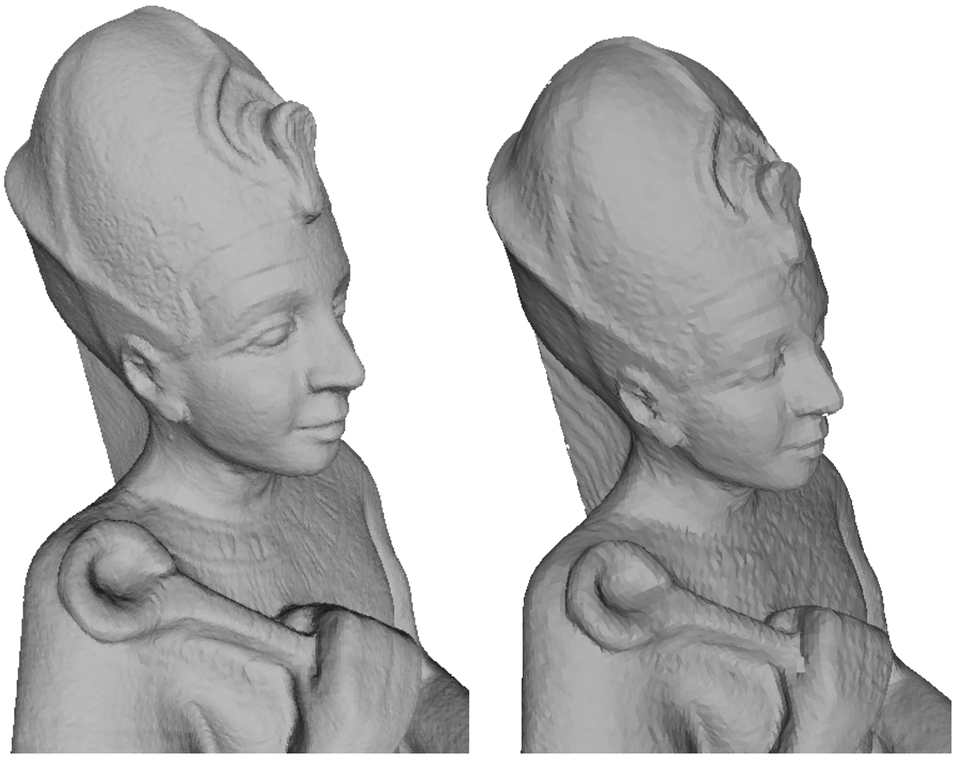

Fig. 4.10: Line-by-line reconstruction from scan data. The layer-by-layer reconstruction is on the left (layer height 20), and the lower-resolution line-by-line reconstruction on the right (line width and height 30).

5. FEATURE DETECTION

This chapter is based on material originally published in the 2010 proceedings of the Theory and Practice of Computer Graphics[54].

In this chapter, we propose a new algorithm for feature detection. The algorithm is based on a self organising map with the connectivity of a regular 3D grid that can be trained into an implicit representation of surface data. The implemented self organising map stores not only its current state but also its recent training history, which can be used for feature detection. Preliminary results show that the proposed algorithm can detect various types of feature on simpler data sets.

As in chapter 4, the presented SOM minimises the second derivative of the gradient by updating each node's estimate of its distance from the surface (rather than by updating its position).

5.1 Introduction

One of the main challenges in surface reconstruction is the detection of surface features. In this context, we are interested in geometric features, and therefore define a feature as a point on the surface for which the curvature is significantly different than the most points in its neighbourhood. This definition neatly classifies spikes, corners, creases, and ridges as features, whilst allowing that features may comprise multiple points on the surface.

The ill-posed nature of the surface reconstruction and the feature detection problems means that the use of machine learning techniques can be advantageous as they can handle the uncertainty of the data better than their equivalent geometry based techniques. In this chapter, we use a 3D SOM with the connectivity of a regular grid, which is trained to implicitly represent the reconstructed surface [106].

In [106], and all other previously proposed SOM-based surface reconstruction algorithms, the SOM learns the shape of the input data through a training process that alters the values stored at the SOMs nodes and, sometimes, its connectivity. In each training step, only the current state of the SOM is to be stored. Of course, as the evolution of the trained SOM is gradual, the current state does contain information related to previous states, however, in general, the previous states of the SOM can not be fully retrieved.

In contrast, the SOM based algorithm presented in this chapter explicitly stores information not only on its current state but on previous states as well. That is, it stores the *training history* of the SOM. This training history can be used to infer surface feature information, under the assumption that the well-defined at areas of the surface are likely to have a stable training history. Flat areas are expected to exhibit low variance of the SOM node value between different states. Conversely, the less well-defined feature parts of the surface are expected to have a more unstable training history, that is a higher variance of the SOM node value between different states.

As the implementation stores not only the current state of the SOM but also some of its training history, memory efficiency becomes a primary concern. To solve this problem, the implemented SOM does not have the shape of a full 3D grid, but considers only nodes that are near the training samples and thus near to the reconstructed surface. Other differences between the implemented implicit SOM and the one proposed in [106] are discussed in section 5.2.

The algorithm proposed in this chapter uses an implicit SOM surface reconstruction. After training, the isosurface is extracted using a slightly modified form of the Marching Cubes algorithm.

The results show that the training history can be used for feature detection. Since the data is a by-product of the surface reconstruction, it adds very little overhead on top of processing that must be done anyway.

5.2 Algorithm

As input, the algorithm takes a set of 3D points with normals, either from a static file or a stream source. The output is a triangle mesh with any potential features highlighted.

5.2.1 Surface Reconstruction

The SOM is trained using the algorithm and parameters described in chapter 4. In the original algorithm up to 20 surface distance estimations were stored, with the oldest being discarded as new entries are added. Tests were run with up to 100 entries being stored, but no discernible difference was found in the reconstruction or efficacy of the feature detection.

The focus of this algorithm is not on the reconstruction itself, but on the metadata created during the reconstruction process. After the computing of the final separations is complete instead of discarding this data and extracting the isosurface, we first attempt to analyse it. In doing so we attempt to estimate areas of high curvature (and thus, features).

5.2.2 Feature Detection

After the training is complete, we cycle through the list of SOM nodes and examine their training history. The weighted variance β of the node's training history is calculated as

$$\beta = (\sum_i \gamma_i^2)/(\sum_i w_i) \quad \text{with} \quad \gamma_i = w_i(d_i - s) \tag{5.1}$$

where w_i is the weight of distance d_i, and the sums run over all the weighted distances in the node's training history.

If β is above the 95^{th} percentile threshold then the node is flagged as having a high distance variance and suspected of being close to a surface feature. A high value of β could be caused by features such as a spike, a crease, a corner, or two parts of the surface lying sufficiently close so that the training data for each part interferes with the other. It could also be caused by inaccurate training data caused by spatial and normal noise, or by incorrectly oriented normals. The thresholding on the variance percentile was experimentally determined to provide a good balance between detected features and false positives.

5.2.3 Isosurface Extraction

In the final step, the surface is extracted using a variant of the Marching Cubes algorithm [71], for which the regular arrangement of the SOM nodes is ideal. If a vertex is created between two nodes that are both flagged as having a high distance variance, then it flagged up as a suspected feature vertex.

We require both nodes to have the flag set in order to cut down on the number of surface areas falsely detected as features (which can happen due to random variation in the surface).

5.3 Results

The algorithm was tested against a variety of simple meshes and complex meshes. Meshes had normals computed and then their connectivity (and therefore face data) stripped in order to provide input point clouds from clean data. To test robustness against noise, vertices were randomly displaced by 0.25 units along their normals. New source data was also created by applying three rounds of Laplacian smoothing to the meshes before computing normals and stripping their connectivity.

Note that due to the nature the diagrams, it is strongly recommended to view them in colour.

5.3.1 Simple Meshes

First, the algorithm was tested on simple models: a Cube, the Fandisk, Bunny and Horse models.

Cube

Being an analytic model, the Cube shows the results most clearly, with the edges being correctly detected on the regular model, and the flag faces not having any vertices flagged as potential features. The smoothed model displays similar results - there are no false positives, though both the regular and smoothed mesh do display some false negatives (the blue vertices on the edges).

As expected, the noisy noisy point cloud, there is noticeably more variation in the learned distances for each SOM node. Consequently the 5% of nodes whose learned distances exhibit the highest variance are more evenly distributed across the model, with many false positives appearing inside the flat, featureless faces. There is, however, still a higher concentration of flagged vertices along the edges.

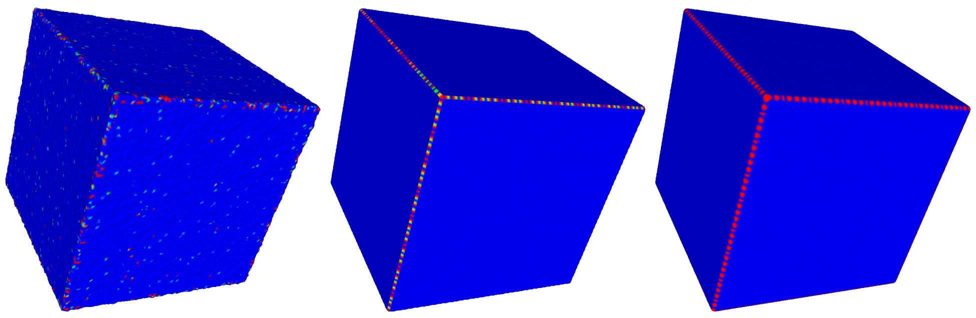

Fig. 5.1: Cube results: *left*: noisy, *centre*: original, *right*: smoothed.

Fandisk

Following in this vein, as a relatively simple geometric model, the algorithm also gives good results on the Fandisk. Once again, the edges are correctly detected as edges in the standard model (with some false positives on the larger curved areas).

The noise model has the expected wider variation, as shown by the wider range of colours in the high-vertices. Increased noise also leads to greater variation in the larger curved areas. Smoothing the point cloud leads to the expected result of poorer feature detection along the edges, though they are still the main area that is picked out.

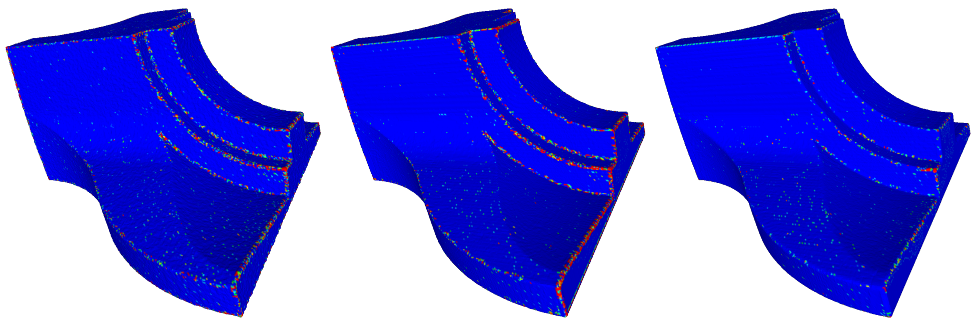

Fig. 5.2: Fandisk results: *left*: noisy, *centre*: original, *right*: smoothed.

Bunny

As a more complex shape, the results for the Bunny are not quite so impressive. The algorithm has correctly picked out the primary contours around the ears as being feature-like. The head, knee and joint around the tail are also picked out in all cases, though the smoothed model, as expected, has fewer vertices flagged.

Unfortunately for all variants, the natural texture of the model surface has led to a certain amount of false positives in the middle of some of the larger areas. It is noticeable however, able that these tend to be flagged blue, indicating that these vertices were near nodes with lower variance that the red vertices.

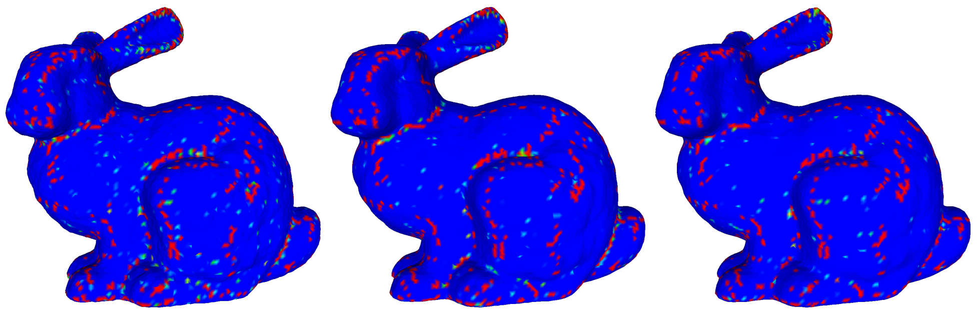

Fig. 5.3: Bunny results: *left*: noisy, *centre*: original, *right*: smoothed.

Horse

The Horse model exhibits similar results to the bunny - obvious features such as the ears and thinner areas on the legs are highlighted in red, whilst lower-variance vertices are still highlighted in the larger areas. The smoothed model shows some reconstruction artefacts around the ears, likely as a result of the smoothed model causing the training data to pass through very thin areas of the ears and interfere with nearby nodes. Aside from the wider distribution of highlighted vertices in the noisy model, the results are similar for each variant.

5.3.2 Complex Meshes

The algorithm was then tested on more complex models: the Blade, Happy Buddha, and AIM@SHAPE Neptune model.

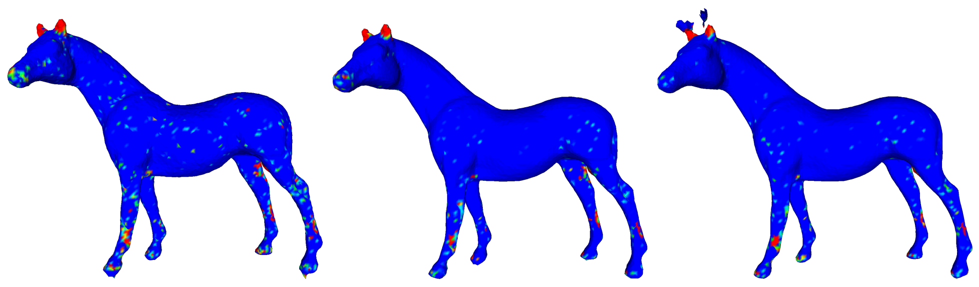

Fig. 5.4: Horse results: *left*: noisy, *centre*: original, *right*: smoothed.

Blade

The Blade results are good: the ridge along the bottom is correctly highlighted in all three variants, as are the holes on the left hand side and the top right corners and the worn area at the join between the vertical and horizontal components. Feature-flagged vertices can also be seen in the holes along the right hand side.

Once again the noisy mesh shows a wider distribution of flagged vertices, and both this and the smoothed model have minor artefacts, most likely due to training data crossing surface boundaries and interfering. Overall the results are good though, with the larger, flatter areas having few false positives flagged in both the smooth and original reconstruction.

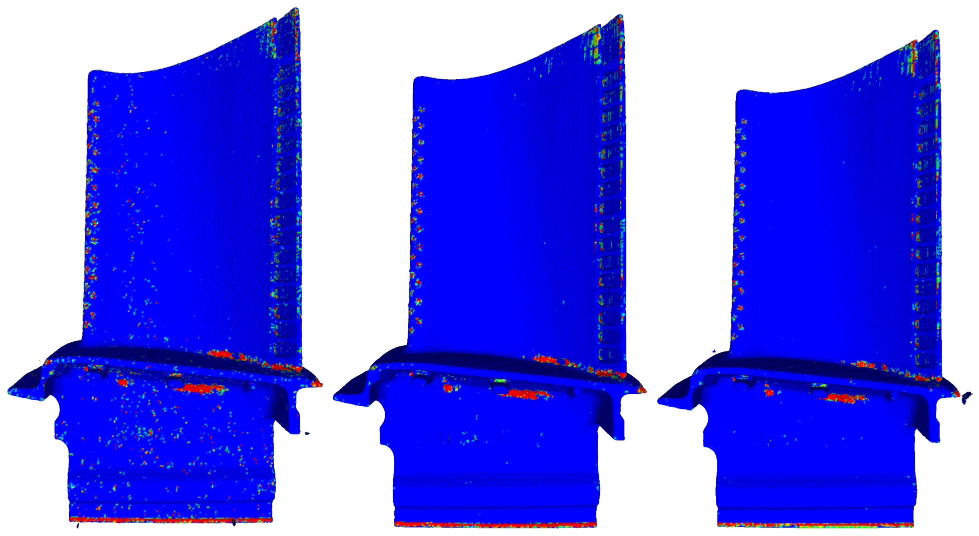

Fig. 5.5: Blade results: *left*: noisy, *centre*: original, *right*: smoothed.

Happy Buddha

The Happy Buddha is a challenging model due to the large number of feature-like areas, due in part to the number of thin areas of the surface and folds of the robe. The noise added to the Happy Buddha resulted in a large number of artefacts, which are visible around the edge of the model, though the original and smoothed models were not able to escape a small number of artifacts from very thin surface regions. High variance in training data is clearly visible across apparent surface features and larger, flatter areas.

The regular and smooth models have a number of areas correctly identified. For instance, the edges of the necklace are identified, though the front face is not (since the training data will not overlap here, this is not entirely surprising). The lips and many folds of the robe are detected as having high training variance, and therefore likely to be features.

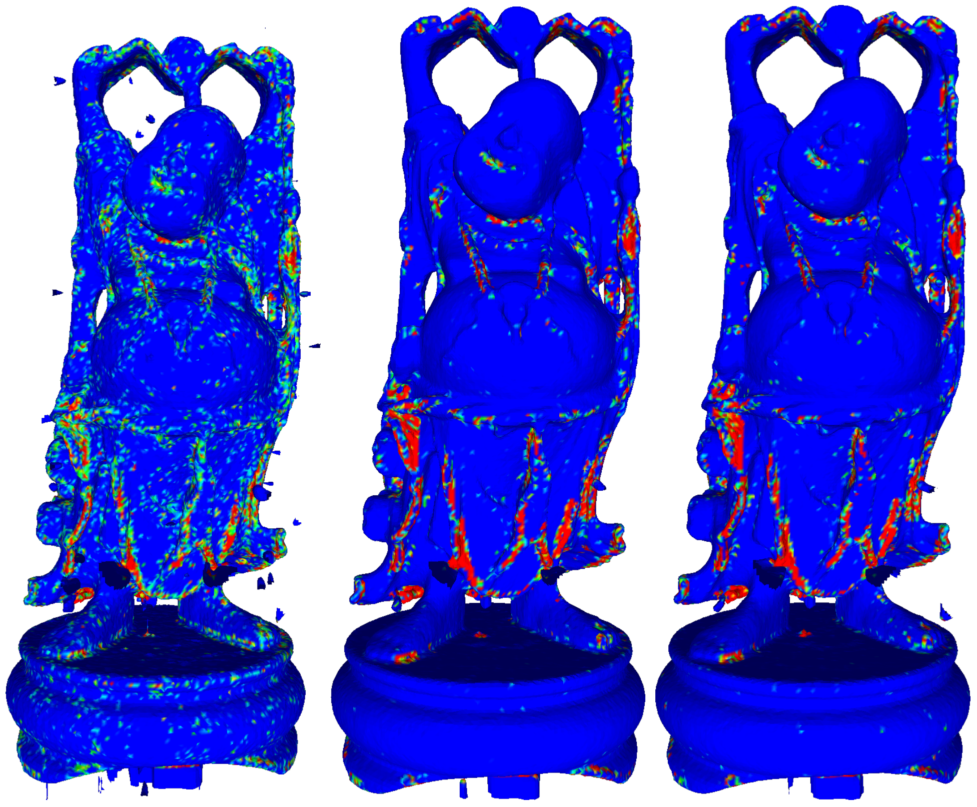

Fig. 5.6: Happy results: *left*: noisy, *centre*: original, *right*: smoothed.

Neptune

The AIM@SHAPE Neptune model is also challenging, due in part to the curls of the beard. Many of these are flagged as features in each of the variants, but it is difficult to pick them out.

Noise corruption once again came with artefacts in the thin areas, and caused many areas to be incorrectly flagged as features. Sevel large areas are (in the noisy model) flagged almost entirely as featureful: the beard, head of the trident, the fish and the sides of the base. This is understandable, particularly since the source data not only contains genuine features, but are also rough even in apparently smooth areas, which have a similar effect to the noise we added; the variation in training data is increased.

In the original and smoothed variants the head of the trident has some areas flagged as features, notably the thin areas running along the length of each spike.

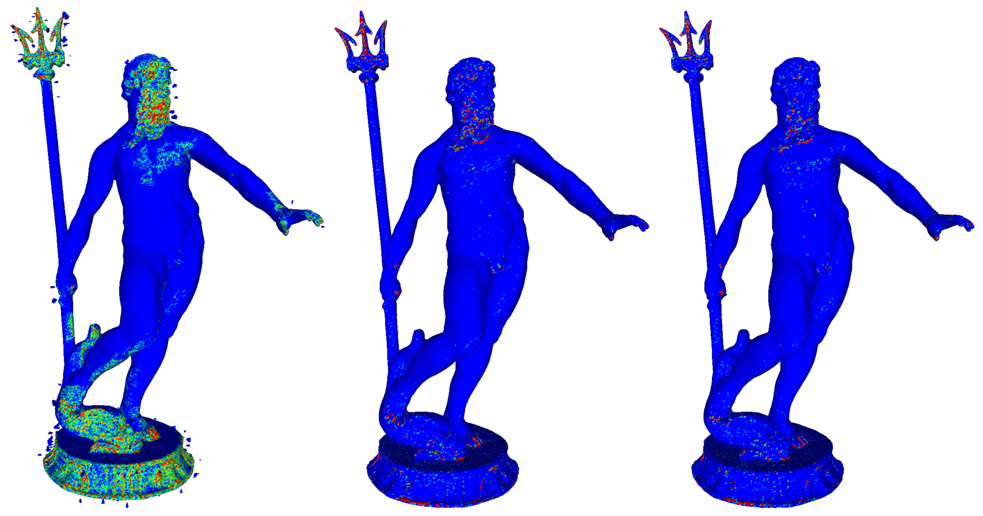

Fig. 5.7: Neptune results: *left*: noisy, *centre*: original, *right*: smoothed.

5.4 Discussion

Whilst our result have been focused on feature detection as applied to a surface being reconstructed, the algorithm is quite general. The feature detection is, in effect, performed on the point cloud directly. Though we have reconstructed the surfaces, this is not a required step.

If the point cloud itself were tagged with this information, which would then be passed to a modified reconstruction algorithm, which could alter its properties or parameters for data flagged in such a way. One way to make use of this flagged data could be to decrease the distance from a tagged sample point at which we create training data. This could reduce the conflicting training data and lead to more accurate reconstructions.

Increasing the density of the point cloud (by normal-respecting surface subdivision) and increasing the resolution of the analysis (by scaling) were found to have no significant effect on the results.

The feature detection is not directly sensitive to the estimated distance from the surface (since it could also be applied without surface reconstruction, this should not be surprising). It is however. sensitive to noise, which causes false positives; non-features being flagged and highlighted as though they were features.

The algorithm performs best at detecting thin surface sections (for instance, the Happy Buddha's robes) or sharp corners (like the edges of the Fandisk).

6. MESH ALIGNMENT

This chapter is based on material originally published in the 2015 proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications[55].

6.1 Introduction

Translation and scaling are both considered less challenging to compute than the rotation. The simplest and most widely used method for the rotational alignment of two meshes aligns the principal axes of the mesh vertex sets. Despite its popularity, it is well documented that in demanding applications such as shape recognition the results of this alignment method may not be satisfactory. This is especially true when the mesh undergoes processing that potentially disturbs the distribution of mesh vertices such as simplification and remeshing.

One way around this problem is to voxelise the mesh and then apply an alignment algorithm for volumetric data. However, such a method can be computationally demanding, and the cost of the voxelisation cannot be fully justified if it is used for mesh alignment only. A second approach is to apply PCA not on the mesh vertex set, but on a more uniform point set produced by a mesh sampling method. However, such a method would depend on the quality of the triangulation. For example, a large number of long thin triangles in the mesh could cause problems.

This chapter describes a solution in between the above two approaches, that is, a sampling method which, without being a fully volumetric method, is based on creating a subset of the nodes of a regular grid and then performing PCA on that point set.

Meshes are typically aligned using a variant of the ICP algorithm. However, since these require one input to be a subset of the other (and possibly some manual intervention), they are not directly comparable to the algorithm described in this chapter.

6.2 Alignment Algorithm

We begin with two meshes (A and B), assuming that mesh B has been obtained from mesh A after a rotation by an unknown angle around an unknown axis, and possibly subjected to some kind of mesh processing operation. The operations considered here are smoothing, simplification, remeshing, and corruption by random displacement of vertices. Each mesh is centred on the origin as a pre-processing step. The translation can be stored and the reverse operation applied at the end of the procedure. The basic alignment algorithm first creates a regular grid around each mesh, then computes the subset of the grid nodes that are near to the mesh, and finally applies PCA to this subset of nodes.

6.2.1 The Basic Algorithm

For each mesh M, we first create a regular 3D lattice, L_M, around the mesh M. The size of the grid is given by the user and trades-off the speed of the algorithm against the accuracy of the alignment. We then perform the following for each face $f \in M$:

- Calculate the smallest rectangular subgrid, P_f in L_M that completely contains f.
- To increase robustness, P_f is expanded by one node in each direction along each axis, for example, a $2 \times 2 \times 3$ subgrid becomes $4 \times 4 \times 5$.
- For each lattice node, $n \in P_f$, determine the shortest distance from n to f.
- If the distance from n to f is less than 2 times the edge of a grid cell, export the node to list I_M (the 'imprint' of the mesh M on the lattice).

For each mesh imprint (from Figure 6.1; the collection of green nodes from every face in the mesh), we perform PCA on the nodes' coordinates

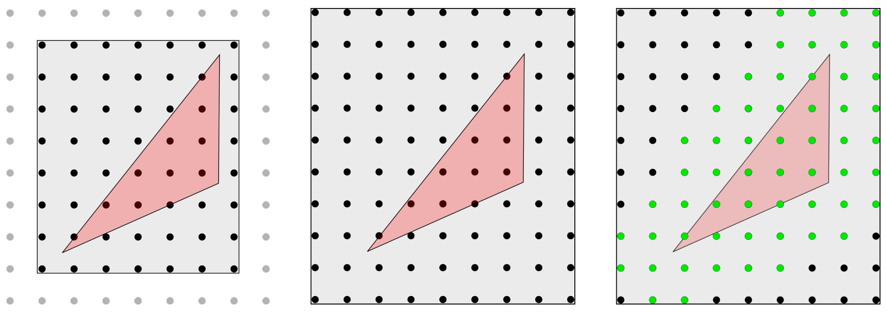

Fig. 6.1: *Left:* the black nodes are the smallest subgrid that completely contains the red face. *Centre:* the smallest subgrid is extended to decrease discontinuities. *Right:* the nodes highlighted in green are the imprint of the red face on the lattice.

and sort the output eigenvectors in decreasing order of eigenvalue magnitude. Note that a more sophisticated implementation of the algorithm would apply a weighted PCA, with the weight of each node derived from its distances to the mesh triangles that pushed it in I_M. However, we have found experimentally that this would not have a significant effect on the results and thus, we opted for the much simpler unweighted PCA.

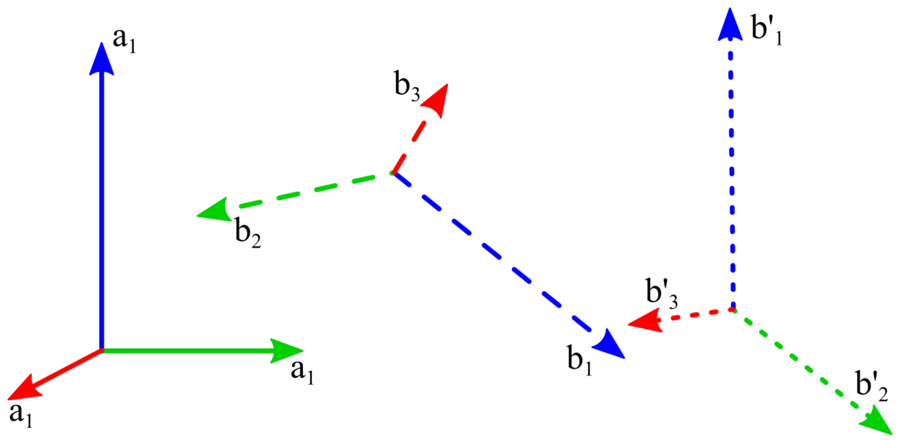

Fig. 6.2: *Left, solid:* eigenbasis of mesh A. *Centre, dashed:* eigenbasis of mesh B. *Right, dotted:* eigenbasis of mesh B'.

Between the two eigenbases (one for each mesh), pairs of eigenvectors are formed based on them having the largest, middle, or smallest eigenvalue magnitude (blue, green, and red correspondingly in Figure 6.2). Since PCA does not provide oriented principal components, we have to ensure that the two eigenbases are consistently oriented. Where an inconsistent orientation was detected, as we discuss in Section 6.2.3, the sign of the eigenvector with the smallest eigenvalue was flipped in one of the meshes.

For the actual mesh alignment, we start with the first pair of principal components, a_1, b_1, (those with the largest eigenvalues; blue). The rotation aligning a_1 with b_1 (blue) is computed and applied to mesh B to produce mesh B'. Lattice imprinting and PCA is then performed on B'. The rotation around a_1 (or, equivalently at this point, b_1) aligning a_2 with b'_2 (green) is then computed and applied to B' to produce a mesh B'' which is in alignment with A.

Note that it would have been possible to work out both rotations (or even, a single rotation) from the initial PCA. However, this is likely to be less accurate, as the imprints of the meshes B and B' are different. By imprinting for a second time, the alignment of the second eigenvector uses a dataset that is closer to B'.

6.2.2 Iterative Algorithm

The basic algorithm can be repeated on mesh A and the mesh B'' (which has been aligned with A). The procedure usually leads to a closer alignment, but the decrease is not monotonic, and in some cases it can even lead to poorer alignment.

We believe that the reason for the non-monotonic decrease is the discrete nature of the grid relative to the mesh itself. This means that even a tiny rotation, which can change the position of any mesh vertex by no more than an arbitrary small distance ϵ, may nevertheless change the position of a grid node marked for processing by a distance equal to the edge of a grid cell.

6.2.3 Eigenvector Orientation

In order for the method to work, the principal components of meshes A and B must be consistently oriented. However, PCA does not define a consistent orientation. In order to align the two principal components a_i and b_i consistently we evaluated a sum of distances function on the two extreme mesh vertex projections on the principal axis. For each of these points, the distance to every data point was summed. Then the principal axis was oriented in the direction of the point with the largest associated sum.

Consistent orientation of principal components is a particularly challenging problem. In our experiments we noticed instances where the method was not successful, causing large deviations (approximately 180 degrees) from the true alignment.

6.2.4 Input Types

The algorithm can, with minimal alteration, be used to align a mesh with a point cloud, or even to align two point clouds. Each point in the cloud is simply interpreted as a face with zero area. Here, the benefit of using the lattice instead of performing PCA directly on the point cloud is the increased robustness of the calculations due to the external reference. Which allows, for example, the alignment of a point cloud and a simplified version thereof.

6.3 Results

In the first experiment, each mesh had its principal components computed by mesh imprinting and was rotated by a known angle around the largest principle component. The proposed algorithm was then used to recover the rotation angle, this was repeated using vertex PCA and the results were compared. The results are summarised in Table 6.1.

Mesh	Mesh Imprint	Vertex PCA
Bunny	0.45407	0.00648
Armadillo	0.18480	0.01150
Fandisk	1.11274	0.00075
Blade	0.93554	0.00650
Statuette	5.62708	0.01968

Tab. 6.1: Mean errors (in degrees) when recovering angles from a set of known rotations.

The angle recovered from the Statuette is much larger than for vertex PCA. The primary reason for this is that the Statuette is a highly-detailed model with significant rotational symmetry. Consequently, imprinting onto a coarse mesh loses some of this finer detail, exacerbating the problem of rotational symmetry, since the fine details are lost.

Since mesh alignment is an ill-posed problem, in a second experiment we evaluated the visual relevance of the reported errors by rotating the test meshes by an unknown angle around an unknown axis. The algorithm was used to bring them back into alignment. The results for the Armadillo and Statuette, with the smallest and largest mean error respectively, are shown in Figures 6.3 – 6.4.

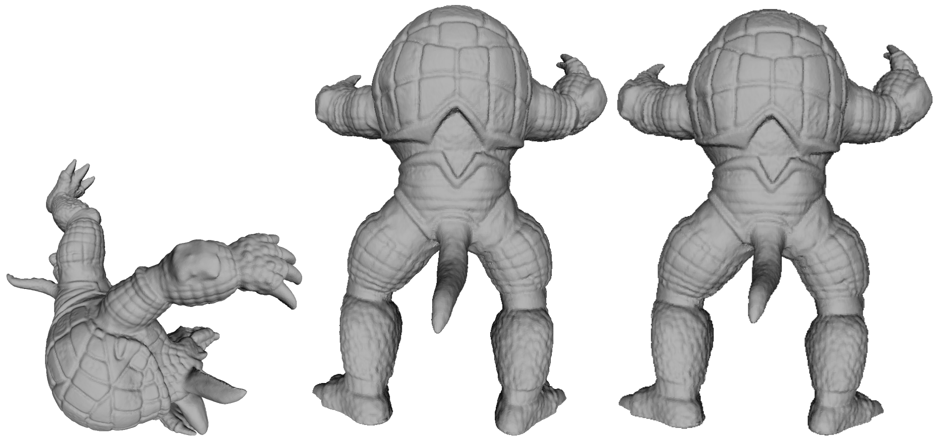

Fig. 6.3: Standard Armadillo results. Left: initial rotation, middle: original, unrotated mesh, right: four iterations.

6.3.1 Robustness Against Mesh Processing Operations

Smoothed versions of the models were obtained by applying three iterations of Laplacian smoothing (updating the position of the vertices based on the position of their direct neighbours). Simplified versions were obtained by using clustering decimation with a cell size of 1% of the diagonal of the bounding box. The decimation results are shown in Table 6.2. Noisy meshes were obtained by randomly displacing vertices by 1% of the bounding diagonal. For remeshed models, surfaces were reconstructed using the Poisson method[57] with 10 octree subdivisions.

The algorithms were then run against each mesh and the processed variants thereof. For each method, the angular deviation between corresponding principal components of the original and the processed mesh

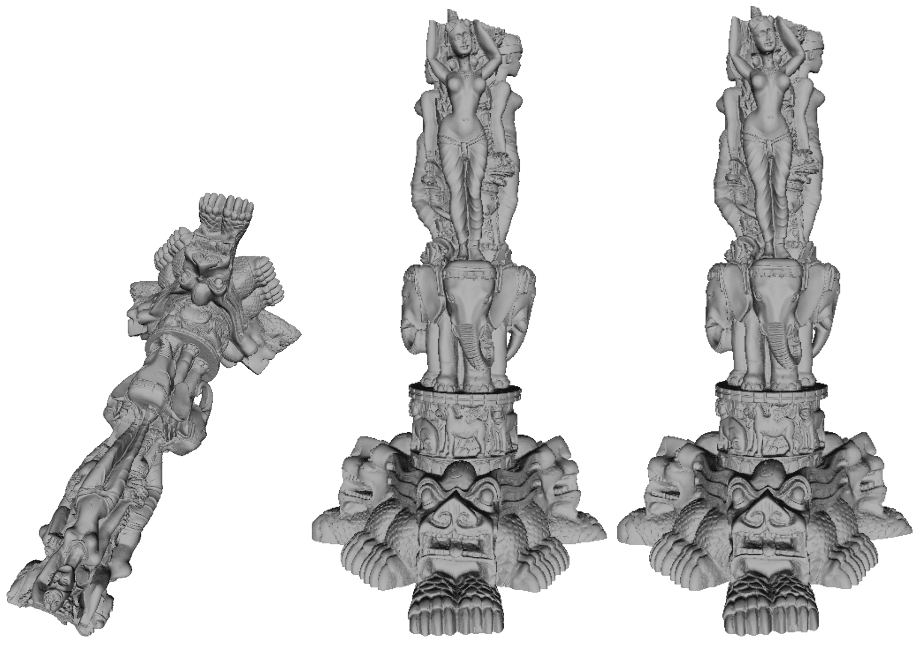

Fig. 6.4: Standard Statuette results. Left: initial rotation, middle: original, unrotated mesh, right: four iterations.

Mesh	Original faces	Decimated faces
Bunny	35,947	9,588
Armadillo	172,974	7,540
Blade	1,765,388	16,088
Statuette	10,000,000	18,330

Tab. 6.2: Number of faces in the original and decimated meshes.

were computed. The results are summarised in Tables 6.3 - 6.6. As expected, mesh imprinting did not give such good results on the highly rotationally-symmetric Statuette model.

The proposed algorithm performed well on the remeshed and simplified variants, but the noisy and smoothed variants were better served by vertex PCA. This is in line with our expectations as remeshing and simplification are likely to have a larger impact on vertex distribution than uniformly applied noise and smoothing. Poorer performance after smoothing or adding noise is added is not a huge problem, as these are less likely to be done in a practical context where this algorithm might be used. Simplification and remeshing however, are common operations, and so better results here are more practically significant.

Remeshed	Mesh Imprint	Vertex PCA
Bunny	1.17170	2.53230
Fandisk	0.47195	0.49521
Armadillo	0.20151	0.83864
Blade	0.05844	1.06610
Statuette	3.61364	15.16333

Tab. 6.3: Average deviation of principal components (in degrees) when models were remeshed.

Simplified	Mesh Imprint	Vertex PCA
Bunny	0.48455	1.80539
Fandisk	1.40759	6.68831
Armadillo	0.13906	0.45590
Blade	0.28992	0.46369
Statuette	1.02004	17.24127

Tab. 6.4: Average deviation of principal components (in degrees) when models were simplified.

Noisy	Mesh Imprint	Vertex PCA
Bunny	0.07809	0.01908
Fandisk	0.00269	0.01375
Armadillo	0.03226	0.04337
Blade	0.04314	0.02045
Statuette	0.28631	0.01530

Tab. 6.5: Average deviation of principal components (in degrees) when mesh vertices had noise added.

Smoothed	Mesh Imprint	Vertex PCA
Bunny	0.73757	0.00974
Fandisk	0.38102	0.00458
Armadillo	0.21526	0.04584
Blade	0.01547	0.00613
Statuette	0.08497	0.00000

Tab. 6.6: Average deviation of principal components (in degrees) when meshes were simplified.

6.3.2 Iterative Algorithm

The performance of the iterative algorithm is shown in Figures 6.5 and 6.6. We notice that generally, the iterative algorithm has improved accuracy in each successive iteration and that most of the improvement materialises in the first three or four iterations. Since the computation of principal components is rotationally invariant, there is no significant improvement in alignment in subsequent iterations. Once again, it is clear that imprinting a highly rotationally-symmetric model onto a coarse mesh gives less accurate results.

Given the number of vertices in some of the models however, it seems likely that the accuracy improvements in subsequent Vertex PCA iterations were a result of the accuracy limitations of floating point arithmetic.

Of particular note are the simplified Fandisk results in Figure 6.5. Whilst Vertex PCA appears to instantly converge to a highly accurate result, in reality this was a very poor alignment. The simplified Fandisk had many large, thin triangles, which significantly altered the distribution of vertices in the mesh, which in turn significantly changed the initial computation of the principal components. Consequently, all subsequent comparison of the original principal components with those computed after the remeshing were invalid. The Mesh Imprint results on the simplified mesh are a true representation of the alignment, as are the Vertex PCA results for the standard and smoothed Fandisk.

6.3.3 CAD Meshes

The proposed method is particularly well suited for CAD meshes that have undergone mesh processing operations.

The Room 215 model shown in Figure 6.7 is a hand-made replica of an office created using CAD software, it has 171,711 faces and significant variance in vertex density. For instance, large areas of walls are represented by huge triangles, but tiny triangles are used to pick out the detail and high-curvature of the radiator grills and chairs. In its simplified form it has 16,080 faces The simplification will have a large effect on vertex distribution as the highly-detailed areas, such as the radiator, will lose many of their triangles, which removes a large number of the vertices in that area. By contrast, the large, flat areas are already almost as simple as they can be, consisting, as they do, of small numbers of large triangles, so the relative sparseness of vertices in these areas will be unaffected.

When the principal components of each were computed and compared, Mesh Imprinting proved very effective, with a maximum deviation of 0.493 degrees across all principal components, compared to a minimum deviation of 6.302 degrees for vertex PCA.

The same test was run against a simple house model and a remeshed form thereof shown in Figure 6.8. The original mesh had 1,396 faces, the remeshed model had 98,818. The remeshing operation significantly affected vertex distribution by "filling in" larger triangles with many smaller, consistently-sized triangles. Areas of high curvature or that already had a high vertex density may have had their vertices shifted a little or the triangles resized to achieve greater uniformity, but this will have had little effect on the global vertex distribution. The maximum deviation between the standard house and the remeshed form thereof was 4.297 degrees. Vertex PCA however, had a minimum deviation of 6.875 degrees.

The model/dressed models in Figure 6.9 are a pair of models that both depict a human figure. This figure is nude in the regular model, but has long hair and is wearing bulky/baggy clothes in the second. Both were analysed and had their principal components compared. The computation was reasonably stable for both models, showing only small deviations between the two meshes, but with up to five times smaller deviations being produced by the mesh imprint. The maximum deviation between principal components computed by imprinting was 0.126 degrees, and the minimum computed by vertex PCA was 0.688 degrees.

This is not too surprising, as the two figures have the same pose and the changes from the regular model to the dressed model are relatively rotationally symmetric between the two minor principal components. The largest principal component is along the height of the model, and the proportions do not change sufficiently to make much difference to this. This result is significant as it demonstrates that the algorithm can be used to good effect not only on alternative forms of the same mesh, but on meshes that are related in a manner that goes beyond pure geometric processing and into high-level contextual processing.

Mesh	Mesh Imprint	Vertex PCA
Room 215	0.3707	6.7913
House	3.4744	12.0224
Model	0.1896	0.7047

Tab. 6.7: Mean angular deviations (in degrees) between the simplified and standard Room215, standard and remeshed House, and standard and dressed Model.

6.3.4 Effect of Resolution

We experimented on grids at 10%, 50%, 100%, and 150% of their original size. The initial size (100%) of each grid is shown in Table 6.8. In addition to the meshes used earlier, the Sphere and Vase models (Figure 6.10) were also used.

Mesh	Volume
Bunny	$78 \times 77 \times 60$
Armadillo	$127 \times 151 \times 115$
Fandisk	$121 \times 131 \times 67$
Blade	$352 \times 598 \times 274$
Statuette	$235 \times 396 \times 203$
Sphere	$105 \times 108 \times 105$
Vase	$55 \times 101 \times 55$

Tab. 6.8: Initial grid sizes.

Predictably, lower-resolution analyses usually produced alignments that were not so accurate as higher-resolution analyses. However for the Bunny and Armadillo models the 150% resolution alignments were actually slightly less accurate than the 100% resolution analyses.

The Vase, Sphere and Statuette all highlighted a limitation of the eigenvector orientation method; they produced inaccurate results because one principal component was incorrectly aligned. This occurs when the input meshes have high levels of rotational symmetry. When run at an appropriate resolution the algorithm correctly orients the principal components, leading to a successful alignment. However there appears to be no universally optimal resolution for the lattice in this regard.

6.4 Discussion

The presented algorithm proved robust in the face of the most significant mesh processing operations that are likely to be performed when attempting to align meshes. By its nature (being based on PCA, and using a coarse imprint) it is best suited to meshes that do not have high levels of rotational symmetry. As expected, it significantly outperformed vertex PCA when operations were performed that altered the vertex distribution of the input.

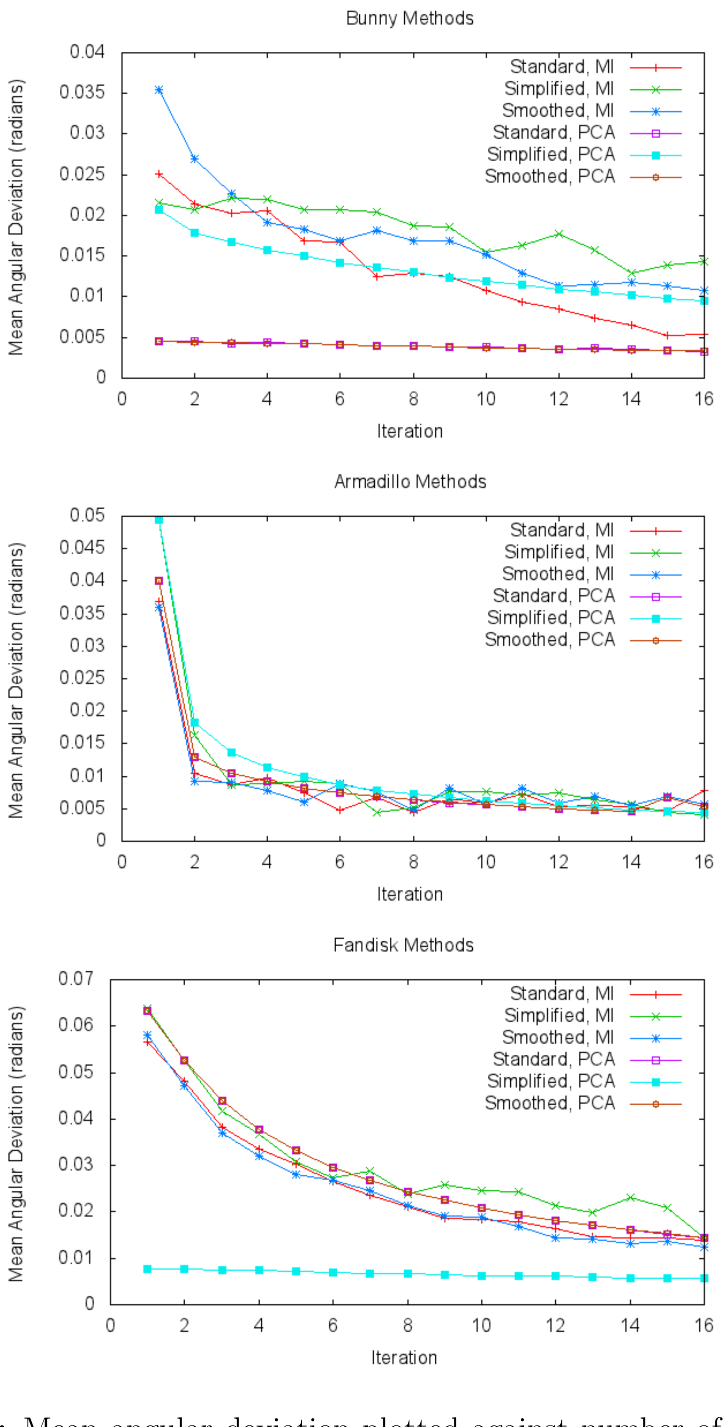

Fig. 6.5: Mean angular deviation plotted against number of iterations for the Bunny, Armadillo and Fandisk.

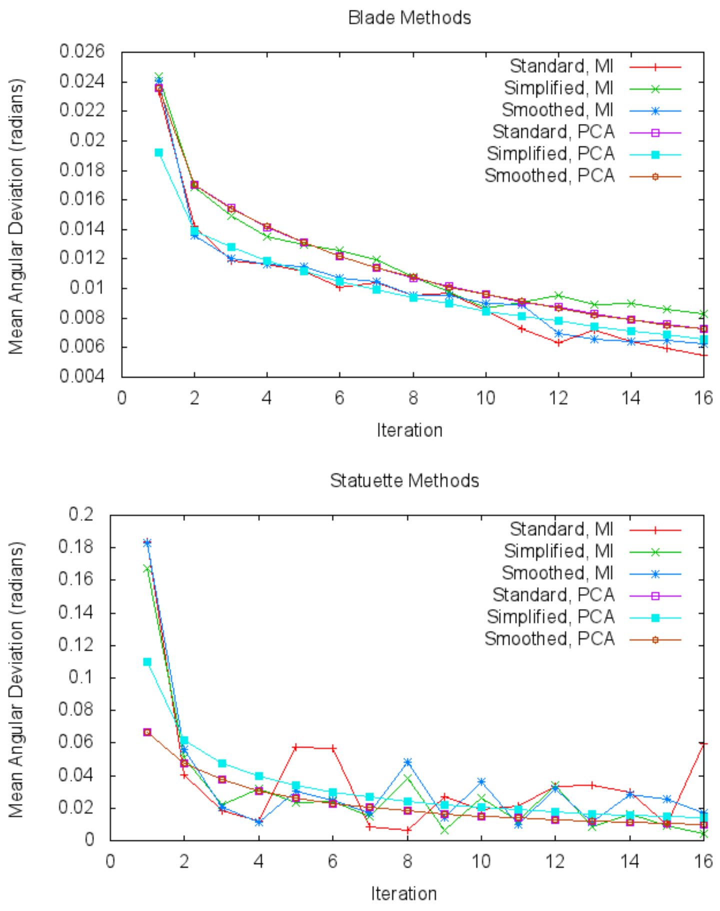

Fig. 6.6: Mean angular deviation plotted against number of iterations for the Blade and Statuette.

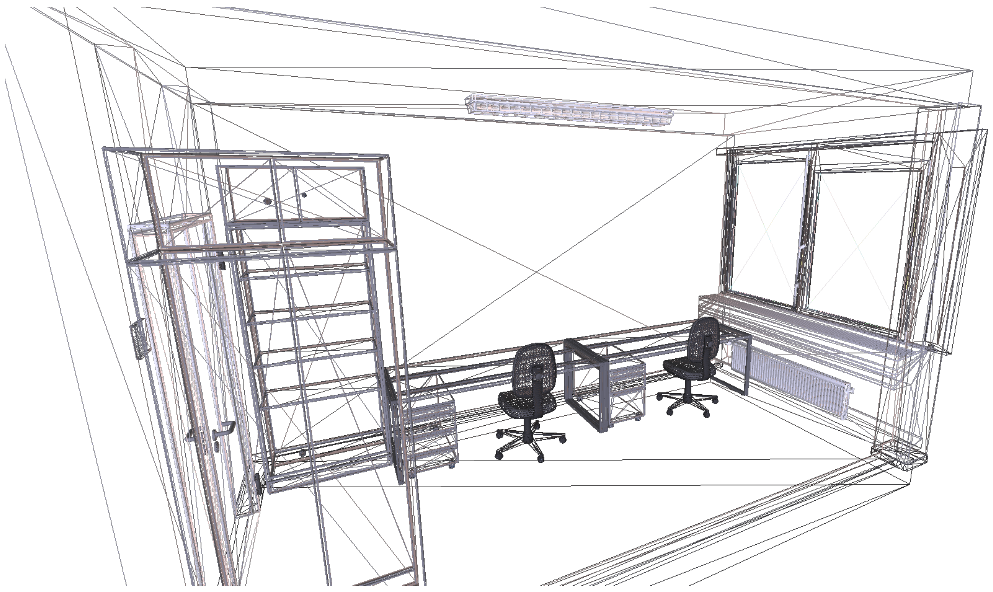

Fig. 6.7: Wireframe view of the Room 215 model. Areas of high curvature have more triangles and appear as solid colours.

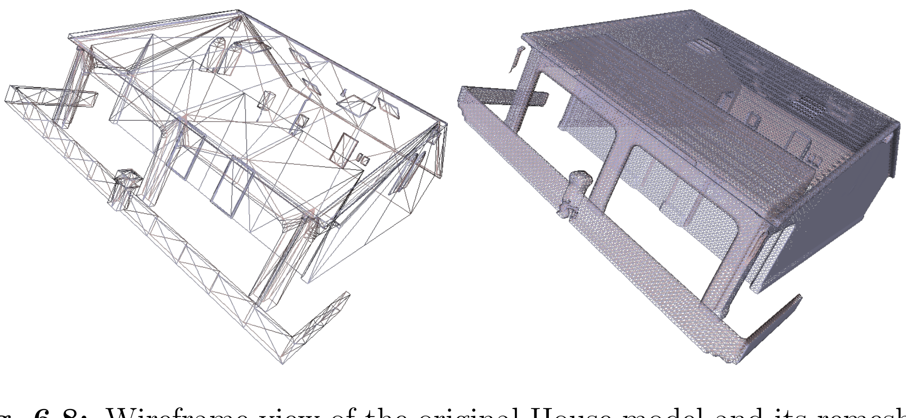

Fig. 6.8: Wireframe view of the original House model and its remeshed form.

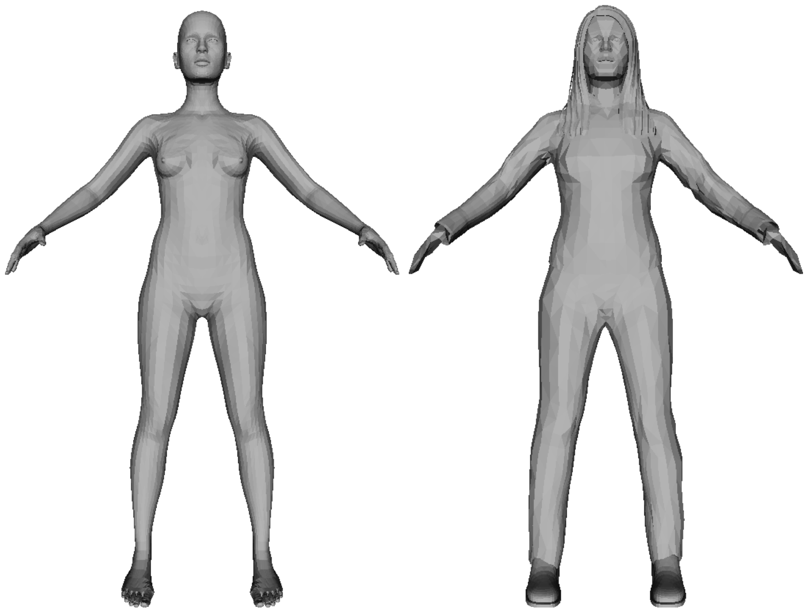

Fig. 6.9: Model/dressed model.

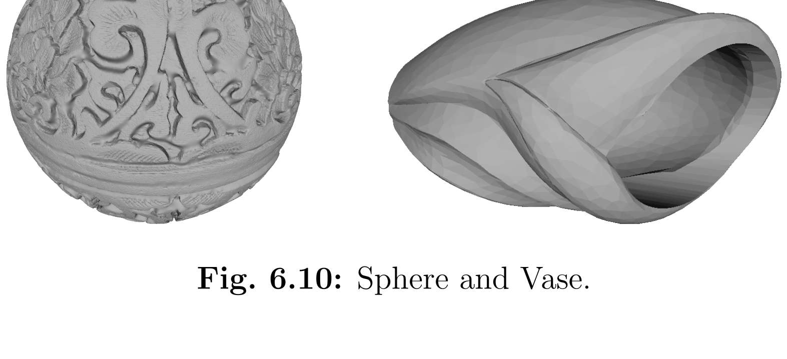

Fig. 6.10: Sphere and Vase.

7. MESH COMPARISON

7.1 Introduction

Some works compare their test mesh to a highly detailed, pre-existing mesh by placing images of each adjacent to each other. This has the drawback that on black and white or low quality printers, subtle variations can be masked, obscuring the results. Even on high quality printers, significant differences can be difficult to observe.

Unfortunately, there are not many objective mesh comparison algorithms available, and those that do exist do not always produce output in the clearest way. Providing the difference in volume gives a global measure of error, but does nothing to elucidate where the error lies, which can make a huge difference to the interpretation of the result.

Other methods are not symmetric, that is, comparing mesh A to mesh B provides a different result to comparing mesh B to mesh A – a result that is at least counter-intuitive. In this chapter we present an objective, symmetric and easy-to-use algorithm that we believe lays the groundwork for this gap to be filled, whilst allowing a simple and immediately clear presentation of mesh differences.

7.1.1 Related Work

The presented algorithm differs from [90] in that the points of comparison are the nodes of a regular lattice that is independent of each input mesh. Like [90], the comparison can be done on a purely geometric basis, or by user-defined attributes (as long as these are defined over the whole input).

The Hausdorff distance between two meshes is the maximum value of set of minimal distances between the two. So, if we have two meshes, A and B, then for each vertex in mesh A, we compute the minimum distance from that vertex to mesh B. The Hausdorff distance from A to

B, $d(A, B)$, is the largest of these distances.

Note that this distance is not symmetric, i.e. $d(A, B) \neq d(B, A)$, this makes sense if we consider the case where A is a subset of B, though this is often forced by setting

$$d_s(A, B) = max|d(A, B), d(B, A)|. \tag{7.1}$$

The Hausdorff distance can be set by a single outlier (consider the case of a single vertex lying a long way from the other mesh).

The main tool used in practice is Metro[22], which works by sampling vertices, edges and faces, and colours of vertices by taking a mean of errors at sample points on adjacent faces. It is against this tool that we will validate the proposed algorithm.

Any significant deviations between the test and reference inputs would show up as a high level of localised error. Consequently if deviations are spread more uniformly over the lattice, this may be indicative of a higher degree of similarity, albeit with a systematic error (perhaps a translation error).

The structure of the lattice allows it to be trained layer by layer in order to improve memory efficiency, as shown in chapter 4. This could prove useful if the meshes/point clouds to be compared are very large. Since the error is defined as a function of a regular grid, it should be easier to apply analytical methods (which may require evenly spaced data).

The lattice structure allows neighbours to be found efficiently, without needing to perform any searches on the mesh. Since a common speed enhancement is to store data instead of repeatedly searching for it, this also helps to reduce the memory footprint.

7.2 Comparison Algorithm

7.2.1 Mesh Processing

Two meshes (A and B) are selected for comparison and scaled to fit inside a unit lattice. Since the lattice nodes have integer coordinates, the choice of scale is, in effect, the choice of the resolution at which the comparison will be performed.

The algorithm assumes aligned meshes as input; as with other measures, mismatched alignments would be detected as differences in the meshes themselves. Similarly, the meshes must be translated to share an origin in order for the comparison to be performed. This is not a troublesome requirement however, and can be met with a trivial preprocessing step. Differences due to noise corrupting the mesh are, in line with other algorithms, simply detected as differences in the meshes.

The presented algorithm is invariant under differing densities – since faces are interpolated to train the lattice the only limitation is the number of vertices in the mesh. In principle per-face textures could be created for each face that were painted according to the lattice's values near that point, but this is somewhat outside the scope of this work.

Each node on the lattice stores its closest distance to each mesh (d_A and d_B). If the input mesh as associated normals, these can be incorporated into the comparison.

A face is selected from the mesh, and the smallest bounding box subsection (parallelepiped P) of the lattice that contains the face is computed. Each node n in P has its shortest distance d_A to the face calculated (be that point inside the face or on the perimeter).

If the distance between the mesh and the node is less than the distance currently stored in n for the mesh in question (or if no distance is currently stored) then that node's d_A is updated. The distance between the mesh and the node is shown in blue in Figures 7.1 and 7.2. Once this has been done for all nodes in P, the next face in the mesh A is selected and the process is repeated.

This process is repeated for mesh B. The lattice nodes keep track of which mesh has trained them, so nodes that have been trained by one mesh and not the other are trivially identifiable. Such nodes are have their difference set to the maximum value, but are excluded from the variance computations to avoid biasing the results.

7.2.2 Difference Visualisation

In order to easily visualise the differences between the meshes we follow the standard practice of applying a changing hue to areas of significance. For each vertex v, we find the nearest node n and map the distance discrepancy of n to a hue according to its magnitude. This can be done on an absolute basis or relative to the other discrepancies.

We then paint v with the calculated hue, giving a colour-coded difference map painted onto the mesh. After this procedure we have 2 output meshes, one is A with the differences to B highlighted, the other is B with the differences to A highlighted.

7.3 Results

The initial analyses were performed on two analytic models - a cube and an icosahedron, wireframe views of each can be see in Figure 7.3. Four experiments were done in order to test the algorithm on clean data before testing it on real meshes.

7.3.1 Analytic Meshes

In order to provide a consistent basis for analysis, both were scaled to fit within a $150 \times 150 \times 150$ bounding box. The mesh volumes and surface areas were computed, and percentage differences compared to the other forms computed. The volumes were all unchanged up to 0.1% relative to the original.

Smoothed

In the first experiment, both meshes underwent three iterations of Laplacian smoothing. As can be seen, the edges are clearly highlighted correctly, with the most significantly highlights in those areas that are be affected most by smoothing.

The RMS difference for the Cube was 0.1755, with a 1.7% smaller surface area. The Icosahedron had an RMS difference of 0.1219 and a surface area and a 0.94% smaller surface area.

Remeshed

The second experiment was to compare each mesh to variants that had been remeshed using Poisson reconstruction. Each bounding box had a side length fixed at 1% of the bounding diagonal of the mesh. The distribution of differences is significant and distributed over large areas.

However, looking at the remeshed wireframe for the Icosahedron, we can see the algorithm detecting differences in the areas that might be expected. A simple threshold was applied that ignored differences of less than 0.1 units, which revealed that the mid-face deviations generally fell into the $0 - -0.1$ unit range. The horizontal bands that can be seen also line up the denser regions of vertices that can be seen in the triangles adjacent to the forward-facing triangle. The RMS difference was 0.3956, larger than for the smoothed mesh, with a 1% smaller surface area.

Similarly for the Cube, the edges have a higher vertex density, and the two visible, off-centre faces have discernible squares of differently-arranged triangles that are shown in the original, coloured form. The smaller differences near the edges are likely due to the Poisson reconstruction creating a slightly more rounded form of the mesh that nevertheless passes very close to the edges in a similar manner to a circle circumscribing a square. The RMS difference was 0.7461, again larger than for the smoothed form, which is in keeping with the Poisson reconstruction. The surface area was 2.5% smaller.

Simplified

In the third experiment, simplification was performed using the Clustering Decimation with a cell size of 1% of the value of the diagonal bounding box of the mesh. The simplified meshes do not show significant groupings of differences.

Looking at the simplified wireframe models, the larger detected differences in both meshes correspond to those areas with more movement of vertices. This is particularly clear for the Icosahedron, where the front-facing triangle has a larger difference. The simplification algorithm gave results on the faces that were axis-dependent, as can be see for the Icosahedron, where the off-centre faces have long, thin triangles (that have been detected as different at their vertices). Once again, a small threshold was then required in order to show up any difference, and these differences were confirmed to be very small, and therefore most likely as a results of minor numerical changes in the storing of the vertex coordinates.

The Cube had an RMS difference of 1.197 and 0.07% larger surface area. Similarly, the Icosahedron had an RMS error of 6.152 and a surface area increase of 0.31%. The RMS errors are large in this case despite the small differences in surface area. Further investigation suggests with both the Metro tool and thresholding suggests that this is a result of the technique used to determine the RMS difference for partially-trained nodes.

Noisy

For the fourth experiment, each vertex was displaced randomly by a vector with a maximum modulus of 0.5 units. As with the simplified and remeshed variants, there is no discernible pattern to the comparison, as would be expected given the random distribution of noise.

The RMS difference for the Cube was 0.3363 with a 15.9% larger surface area, and for the Icosahedron, the RMS difference was 0.5748 with a 21.7% larger surface area. Significant differences in surface area combined with a small RMS difference is in line with intuition for a noisy surface – the irregularity has little overall effect on volume and the differences in distance from nodes to the surface remain small.

7.3.2 Real Meshes

As with the analytic meshes, three rounds of Laplacian smoothing were performed before comparing the meshes. We focused on the smoothed meshes as changes from smoothing are intuitively clear, which eases assessment of the results. This explains the relatively smooth colour changes in comparison to the presented algorithm.

The Bunny and Fandisk models show the strongest visual results for the proposed algorithm, with the most significant deviations being in areas that would be expected. They are also consistent with the Metro output – detected areas of high and low differences coincide across all models. When RMS scores are analysed however, the Bunny had the worst RMS score of all the tested “real” meshes with a difference of 0.545. This may be due to the relative sparseness of the mesh causing a large number of nodes to only be partially trained. The Fandisk had the best RMS result of all meshes at 0.203.

The more complex Happy Buddha and Armadillo models show greater variation across the models as a whole, but with a higher concentration of deviation in near edges. Results are comparable to the result from the Metro tool – areas of more significant deviation are the same between most models (red with the proposed results, green from metro).

Whilst the output of the proposed comparison method shows greater variation across the output, these areas are generally isolated, whereas for the Metro output, they are typically merged into a larger band of colour. The Armadillo and Happy Buddha models had similar RMS differences of 0.382 and 0.377 respectively.

The optimum resolution for the comparison is not known in advance, and must therefore be experimentally determined. If the resolution is too low, large regions will be flagged as significantly different. If it is too high, then the results can become noisy, in a situation equivalent to over-fitting, since the hue of each point is determined by the difference percentile in which the node variances fall.

This is due to the fixed expansion of the parallelepiped around each face when computing the distances from the face. As the resolution increases, this fixed distance becomes smaller relative to the sizes of the models being compared. Consequently, more nodes will only have been trained by one surface, and so will be flagged as having the maximum difference.

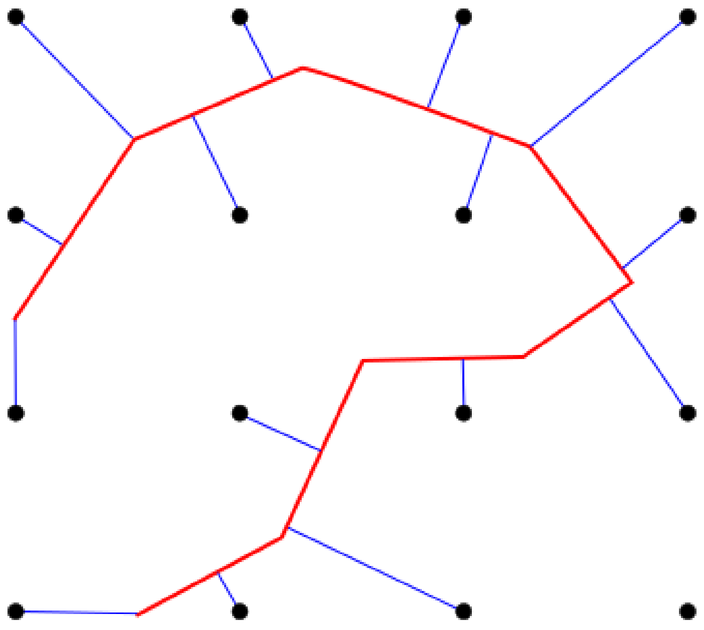

Fig. 7.1: Fitting a mesh to the grid. The shorted distance between each node and the mesh is shown by blue lines.

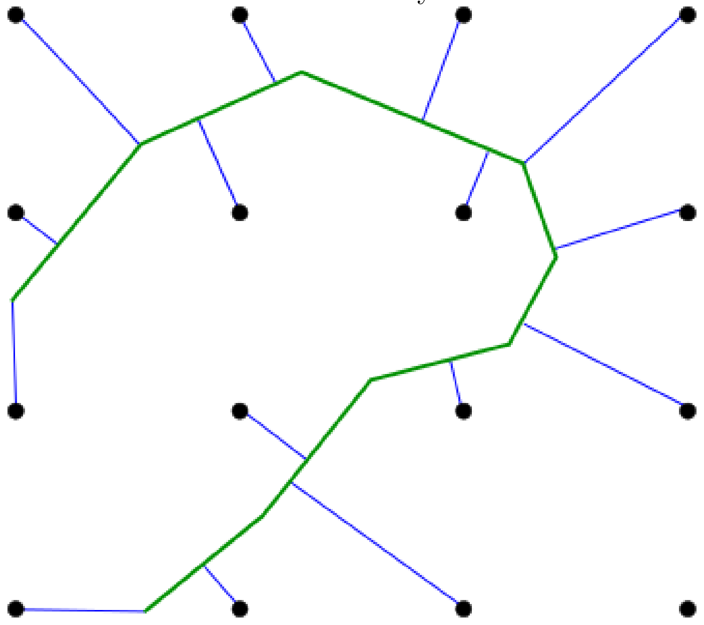

Fig. 7.2: Fitting a second mesh to the grid. Comparing the two meshes then amounts to comparing the magnitude of the blue lines in each case.

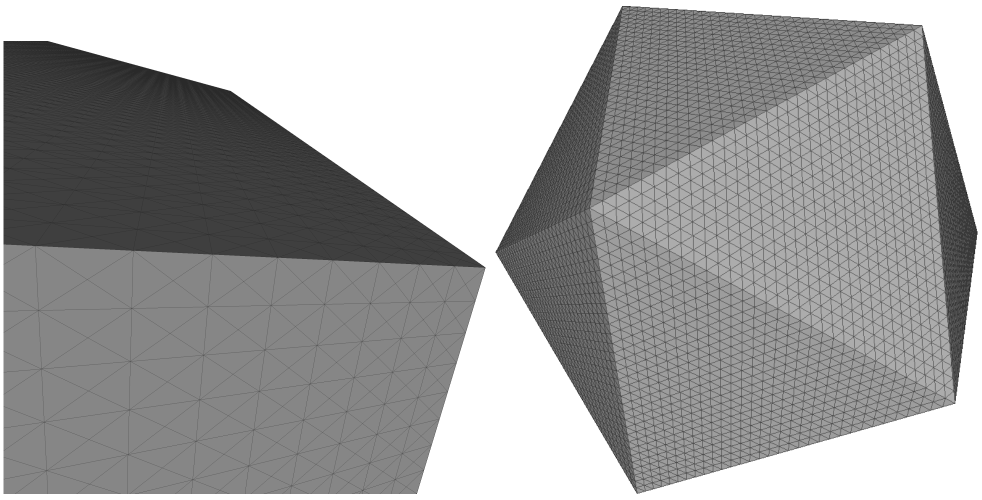

Fig. 7.3: Wireframe view of the cube and icosahedron.

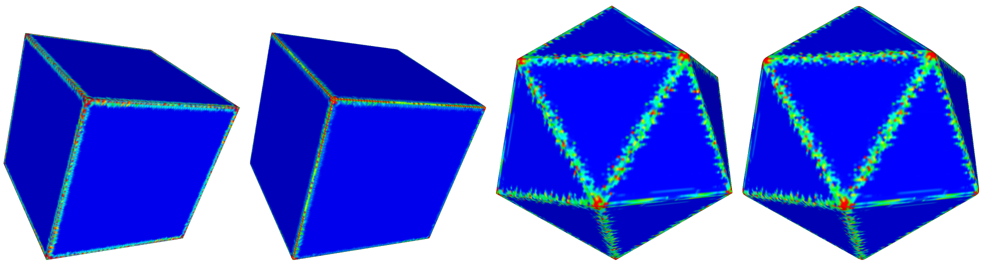

Fig. 7.4: Comparison of analytic meshes (left) to their smoothed counterparts (right).

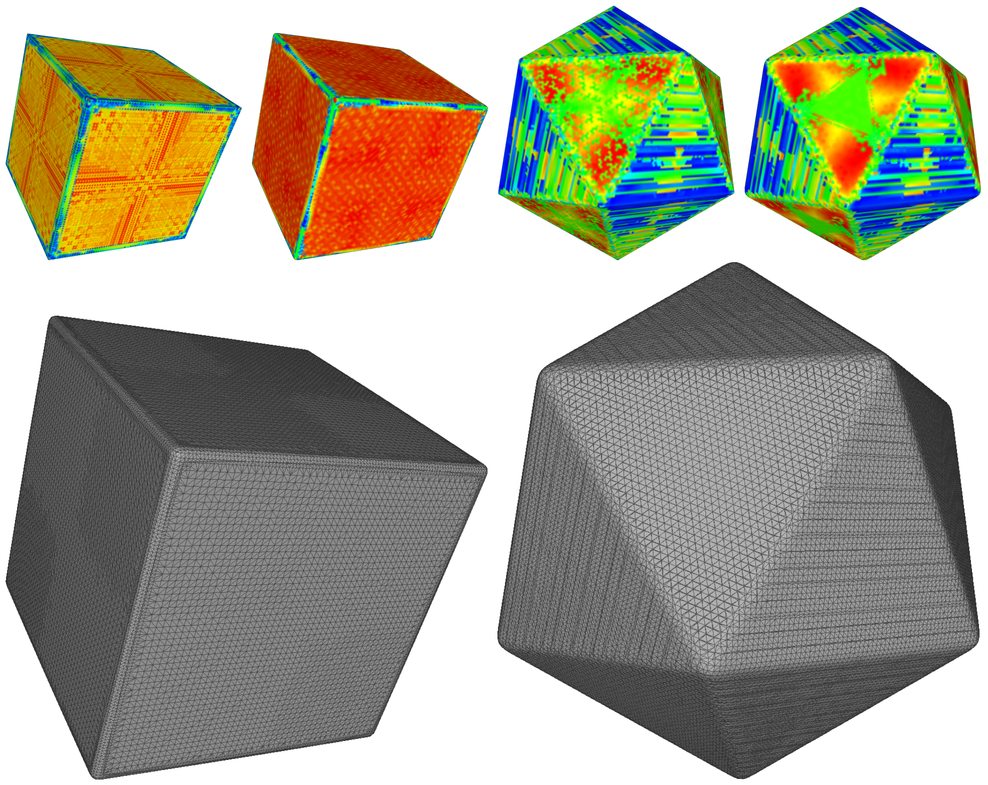

Fig. 7.5: Comparison of analytic meshes (left) to their remeshed counterparts (right). Wireframes of the remeshed forms are shown below.

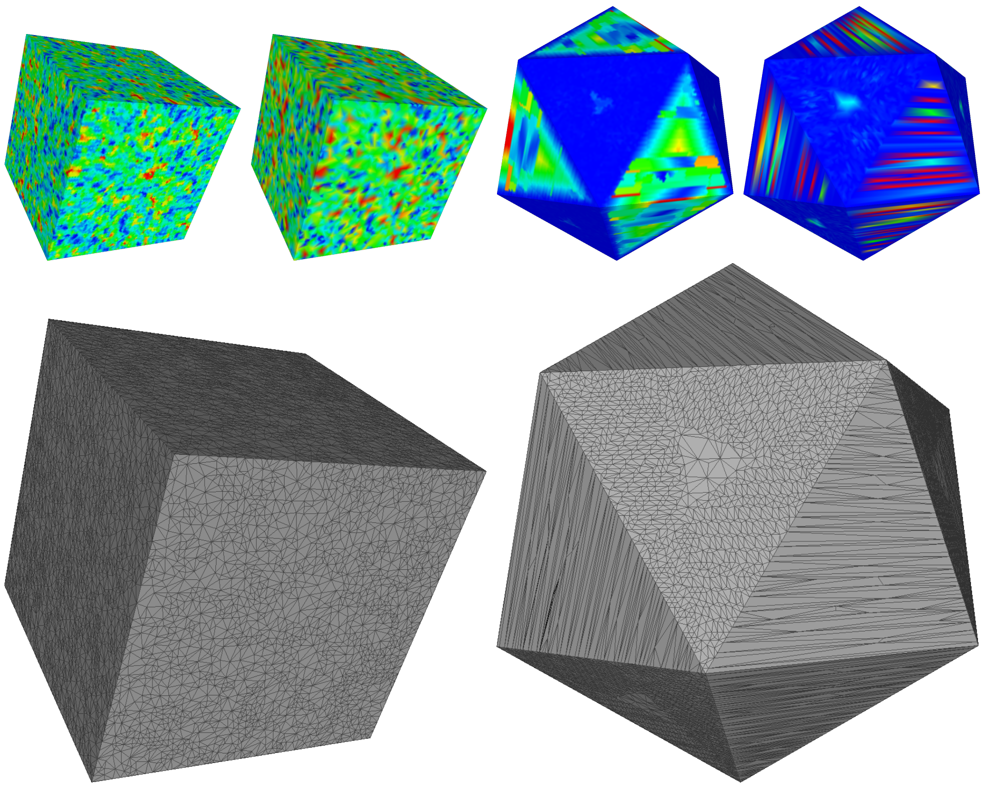

Fig. 7.6: Comparison of analytic meshes (left) to their simplified counterparts (right). Wireframes of the simplified forms are shown below.

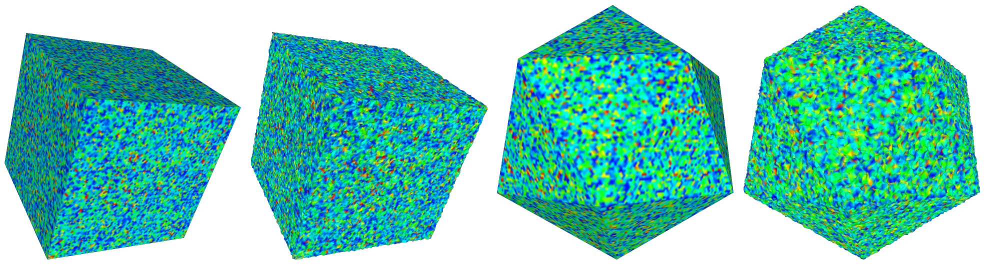

Fig. 7.7: Comparison of analytic meshes (left) to their noisy counterparts (right).

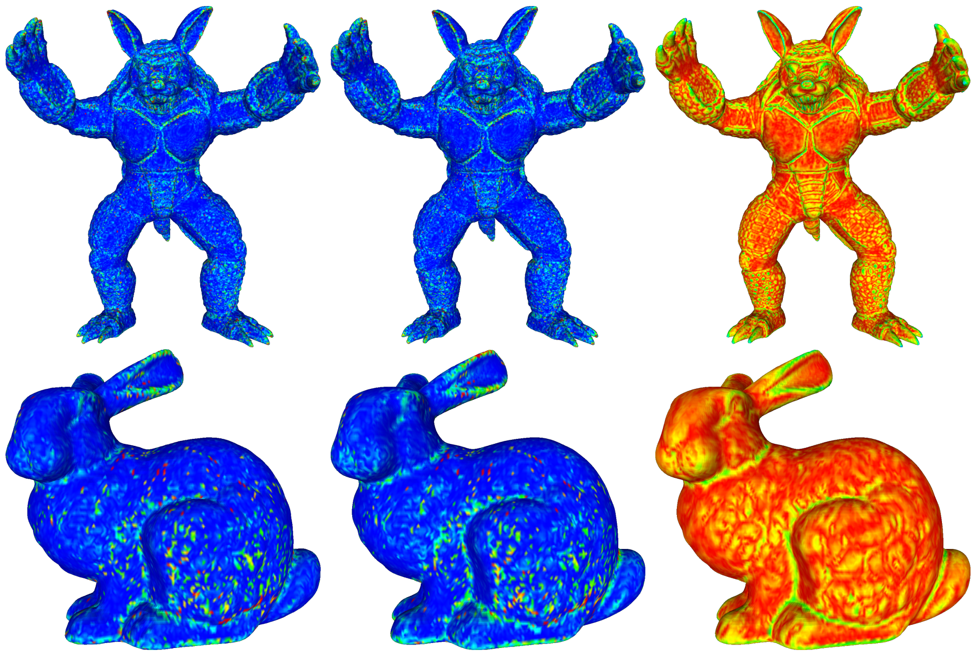

Fig. 7.8: Comparison of real meshes (left) to their smooth counterparts (center), Metro comparisons (right).

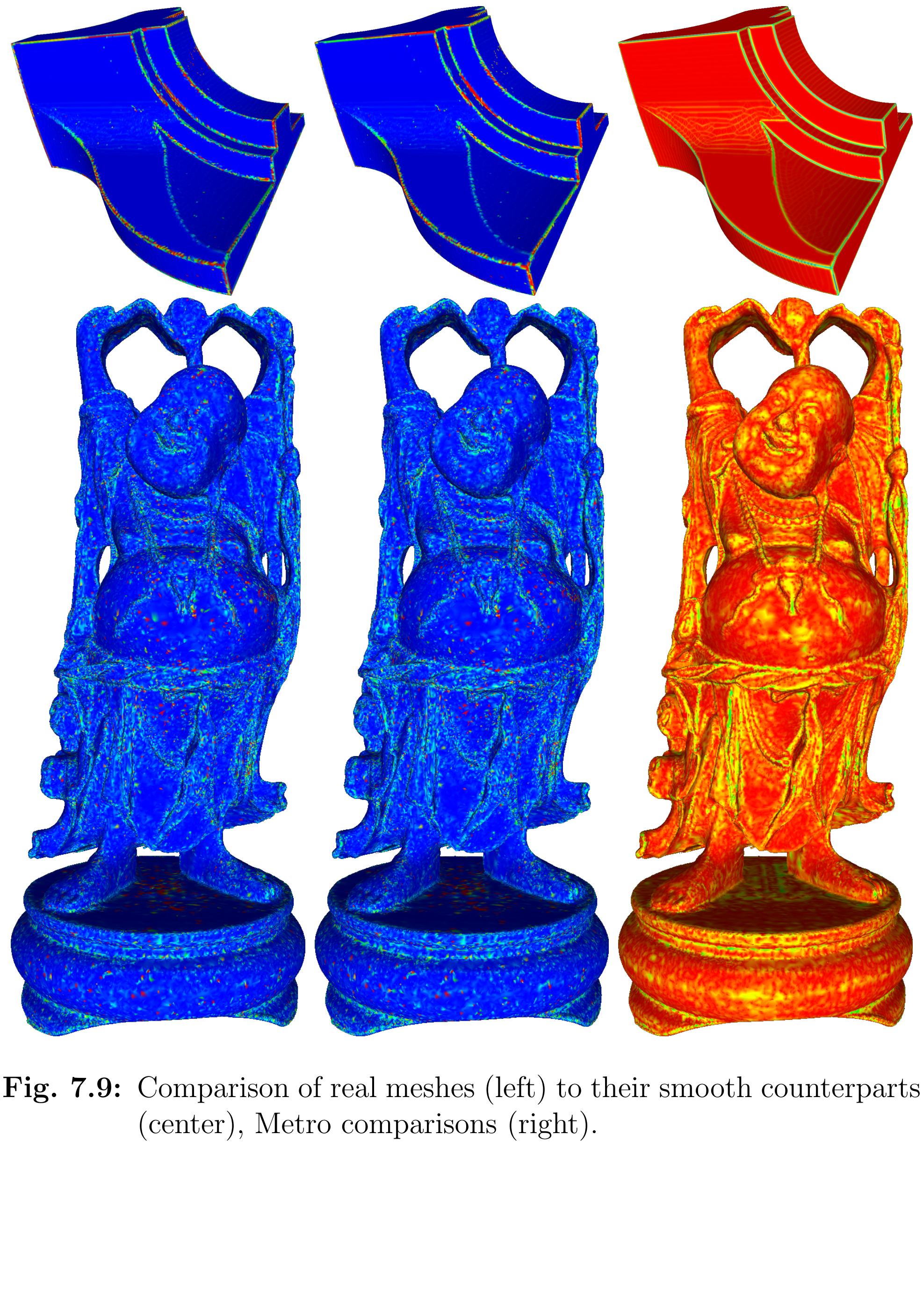

Fig. 7.9: Comparison of real meshes (left) to their smooth counterparts (center), Metro comparisons (right).

8. CONCLUSION

In this chapter a summary of the work performed is presented, along with discussion of the outcomes and limitations. The original hypothesis is then revisited in light of the undertaken experiments. Finally, we discuss areas for future investigations that follow naturally from the work presented.

8.1 Summary of Work

8.1.1 Memory–Efficient Surface Reconstruction

We proposed a memory efficient, scalable surface reconstruction algorithm based on SOMs.

The SOM has the geometry and connectivity of a regular 3D grid. The input data is preprocessed in order to sort it in order of increasing z-coordinate. Samples are taken sequentially from the data and training data created for each sample. Training data consist of regularly-spaced points laying along the normal of the original sample, each carrying an estimate of its distance to the surface. SOM nodes are trained storing a list of estimates, each with a weight factor, to indicate how far from the original sample they are.

If a sample is near the top of the active band, the active band is smoothed and the bottom layers stored. Each node's final distance estimate is a weighted average of these training data. The active band is then moved up by a small amount and the processing continues. This layer-by-layer training of the SOM makes the algorithm memory efficient and scalable, since at no stage is the entire SOM held in memory.

Taking this idea further, we demonstrated initial promising results for further memory footprint reduction by training this active band line-by-line.

8.1.2 Feature Detection

We proposed a new algorithm for feature detection that can be performed in tandem with surface reconstruction. The feature detection is an expansion of the algorithm described in chapter 4 intended to serve as a visual aid.

Before a node's data is stored and the active band moves on, the weighted variance of its training data is computed. Nodes with a high variance (we used the 95^{th} percentile) were flagged as being potentially adjacent to features. This limited the number of nodes that were flagged to only those for which we had a high level of confidence, and prevented the output becoming too noisy.

The variance data is passed to the Marching Cubes algorithm, which renders vertices surrounded by such nodes according to the mean of the nodes between which they lie. Higher variances led to higher deviations from the default mesh colour, increasing the prominence of the most likely features.

8.1.3 Mesh Alignment

We presented an algorithm for mesh alignment by performing PCA on a set of nodes of a regular 3D grid.

The nodes on which to perform PCA were determined by taking an "imprint" of the mesh to be aligned. To do this, the lattice was created around the mesh, and any node within a set distance of the mesh was flagged for inclusion in the calculation. These flagged nodes were then treated as the input to a PCA calculation, and these principal components were aligned with those of the target mesh's imprint.

By taking an imprint of the mesh (instead of just performing PCA on the vertices, as is common practice), the potentially negative consequences of large triangles are avoided. In this sense, the imprint captured the *character* of the input, and could therefore align meshes that are geometrically similar, but structurally very different.

The use of a 3D lattice external to both inputs increased the robustness of PCA, particularly when dealing with meshes of different and possibly uneven vertex density. The proposed algorithm was tested on meshes that have undergone a variety of standard mesh processing operations and it was found to perform well under most circumstances.

8.1.4 Mesh Comparison

We presented an algorithm for the symmetric comparison of two meshes.

As a preprocessing step the meshes are aligned and set to the appropriate scale. Each node in the lattice computes its shortest distance to each grid (as an optimisation, only nodes within a certain distance of the lattice are trained in this manner). The difference between a node's distance to each mesh is used as a measure of the deviation of nearby vertices. In order to smooth out extreme values, these differences are sorted into percentiles.

The meshes are then coloured, with nearby nodes being assigned a colour based on the difference percentile of their closest nodes' differences. Nodes that are far from one mesh (and have thus only been trained by one) are coloured as though their distance difference lay in the largest percentile. As a final step, the RMS difference between the separation of all nodes in the SOM are computed and used to give a quantitative value for the differences between them.

8.2 Outcomes

Throughout the various chapters of this thesis we have attempted to show that a trained grid can be used for many purposes aside from the typical one of computing an input for the Marching Cubes algorithm. Overall we have met this goal: the results have been good for most of the areas investigated; not just meeting the standards of existing techniques but in some cases surpassing them. In the cases where the results were not as good as could be hoped, the reasons for these shortcomings are well-understood, allowing future work to be planned to address them.

8.2.1 Memory–Efficient Surface Reconstruction

Overall this avenue was very successful, with both research questions being answered in the affirmative.

Can the structure of a regular 3D SOM be used to increase the performance of surface reconstruction?

We have found that a regular 3D grid offers notable benefits to the performance of surface reconstruction. The lack of search, querying and sorting simplifies what are often relatively complex operations.

To what extent can the structure of a regular 3D SOM be exploited to work with large datasets?

Experiments showed that the proposed algorithm can support the training of the very large SOMs that would be required for large data sets. The layer-based reconstruction provided a significant decrease in memory requirements compared to earlier algorithms, in some cases by an order of magnitude.

8.2.2 Feature Detection

The feature detection was a qualified success, though further work would be required for it to meet its full potential.

Can the training history of an SOM be used to detect surface features?

Preliminary results showed that the algorithm can detect various types of feature, and gives intuitively correct results with clean inputs.

How early can this be integrated into the pipeline in order to make the information available to more stages of the pipeline?

The algorithm would be well incorporated into a reconstruction that would use the data to inform later pipeline steps. For example, flagged areas could be skipped in the smoothing phase (either as part of a post-processing step, or visually in the rendering phase).

The overhead of this feature detection is very low, and can be implemented by a fast and simple modification to the surface reconstruction algorithm from chapter 4.

8.2.3 Mesh Alignment

In light of the results, the presented algorithm provides a valid third option for alignment after PCA and the ICP variants, particularly when the inputs have significantly difference vertex distributions.

Under what circumstances would an SOM be suitable for aligning two meshes?

If the vertex distribution between the meshes is significantly different, using an SOM to align them as in chapter 6 is robust. Correspondingly, if the meshes are “similar” in some sense, the algorithm can also align them successfully.

To what extent would the regular structure be beneficial, and what limitations would it impose?

In several cases the results indicate an improved robustness compared to performing PCA directly on mesh vertices. This was generally the case when the vertex distribution changed due to mesh processing operations (smoothing, remeshing, etc.).

The most significant problems were in the case of the Statuette, which is highly-detailed and rotationally-symmetric, since the coarseness of the imprint removes detail that can increase the accuracy of the PCA, which already faces some difficulty with rotationally symmetric inputs.

How would such an algorithm compare to standard techniques?

The results as applied to CAD meshes (section 6.3.3) were all better than Vertex PCA, and showed that the algorithm can be successfully applied to meshes that are not simply geometrically-modified forms of each other.

8.2.4 Mesh Comparison

The use of an SOM to compare two meshes was a qualified success. We achieved a proof-of-concept algorithm, but there are still some issues that would need resolving for the comparison to reach its full potential.

Could an SOM, being external to two meshes, be trained to detect their differences?

The algorithm was tested on analytic meshes that had undergone standard mesh processing operations, and performed as expected for smoothed and noisy variants.

Remeshed variants used Poisson reconstruction to rebuild the mesh, which resulted in large deviations between the two inputs. Given the nature of Poisson reconstruction these differences are not surprising, and the deviations are present and correctly detected. Simplified forms also presented some difficulties, but testing with thresholds suggested that these were tiny numerical differences.

Results for for real meshes were comparable to the results from the Metro tool, though given its sample-based method of comparison, its colour maps were smoothed.

Does SOM-based comparison offer any benefits over existing techniques?

In terms of mesh-to-mesh comparisons, whilst the results produced are suggestively similar, the lack of automatic resolution detection means that the current state of the SOM-basd comparison does not meet the same standard as the Metro tool.

It does however, offer a notable benefit compared to Metro, namely that the algorithm is capable of transparently handling comparisons of meshes to point clouds, and even point clouds to point clouds.

8.3 Discussion

8.3.1 Memory–Efficient Surface Reconstruction

The layer-by-layer training of the SOM is the main novelty, and means there is no need to store the entire SOM in memory at any point. The memory efficiency of the algorithm compared to [18, 82, 57] and [54] was demonstrated. Good sized SOMs, such as those used for the reconstructions of Neptune and the turbine, require about 100MB peak memory, while even the massive SOM used for reconstructing the huge Neptune

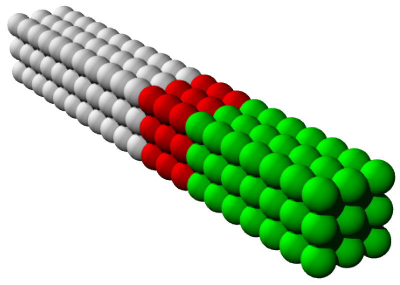

Fig. 8.1: A line could be trained thick point by thick point.

model can be accommodated in the memory of a commodity PC.

The second major advantage of our approach is its scalability. Not only can the training of the SOM be done layer by layer, but, in a recursive application of this principle, a layer can be trained line by line. As shown in figure 4.10, the preliminary results of this line-by-line training are promising, particularly as the memory required was approximately 10% of that required for the layered reconstruction, though at the expense of processing time. If further memory efficiency is needed, a line could be trained thick point by thick point, see Fig. 8.1.

We note that memory efficiency and scalability are natural features of our approach and can be achieved with minimal implementation overheads. In contrast, memory efficiency in other implicit reconstruction methods requires the implementation of complex data structures, such as adaptive octrees, or the use of special scalable algorithms for solving global optimisation problems.

The regular structure of the SOM employed by the proposed algorithm, and the very simple processing operations performed at each node, make the method particularly suitable for GPU implementation.

8.3.2 Feature Detection

The main novelty of the feature detection algorithm is that instead of only storing the current state of the SOM, the recent training history is explicitly stored and used for feature detection. Additionally, despite being performed in parallel with surface reconstruction, the latter is not a requirement – features would be able to be detected and flagged in point clouds by the same procedure.

The algorithm performed very well on analytic and simpler models, such as the Cube and Fandisk, but even gave good results on the Blade. Thin surfaces areas were well detected, like the thin robes of the Happy Buddha.

8.3.3 Mesh Alignment

In our implementation, we used the Point Cloud Library [91] for PCA, and MeshLab [21] for the various geometric operations we performed as part of our testing; smoothing, simplification, adding noise and remeshing. While processing large meshes can require large amounts of memory due to the sheer number of points that must be processed, our method (by virtue of performing PCA on fewer points) will naturally have a smaller memory footprint than many. Memory could also be saved by running the proposed algorithm in a layered fashion, as proposed in [56].

Since the proposed method aligned the meshes on a relatively coarse regular grid, the loss of accuracy compared to direct vertex PCA was noticeable. However, it was inside a range that would be considered tolerable in most applications, that is, around one degree if there were no problems caused by the rotational symmetry of the meshes, or by incorrectly-oriented eigenvectors. Note that these problems are common to both the proposed method and standard PCA on mesh vertices.

Mesh Imprinting shows its strengths when original inputs are poorly meshed. For instance, if they have many long, thin triangles, or an uneven distribution thereof. While long thin triangles are very rare in meshes that are acquired through physical optical devices such as laser scanners, they often dominate meshes produced by CAD software. In such cases, simplification and remeshing significantly affect the distribution of the vertices, causing Vertex PCA to produce highly inaccurate alignments, as discussed in chapter 6 in section 6.3.2.

8.3.4 Mesh Comparison

The algorithm can accept a point cloud as input and could therefore be used for more than just comparing a modified mesh to the original (in the case of simplification or smoothing). For example, it could be used to directly evaluate the effectiveness of a surface reconstruction algorithm or check that a subsampled point cloud still has sufficient density in important areas.

Signed distance to grid nodes was not measured as it would only be possible to measure whether two points lay on different sides of a grid node, which could give rise to misleading results. For instance, in figure 8.2, the red and blue surfaces are a fixed distance apart, however, the way the grid is positioned, the purple node will flag that it is in different sides of the surfaces, which could suggest that the surfaces are uneven.

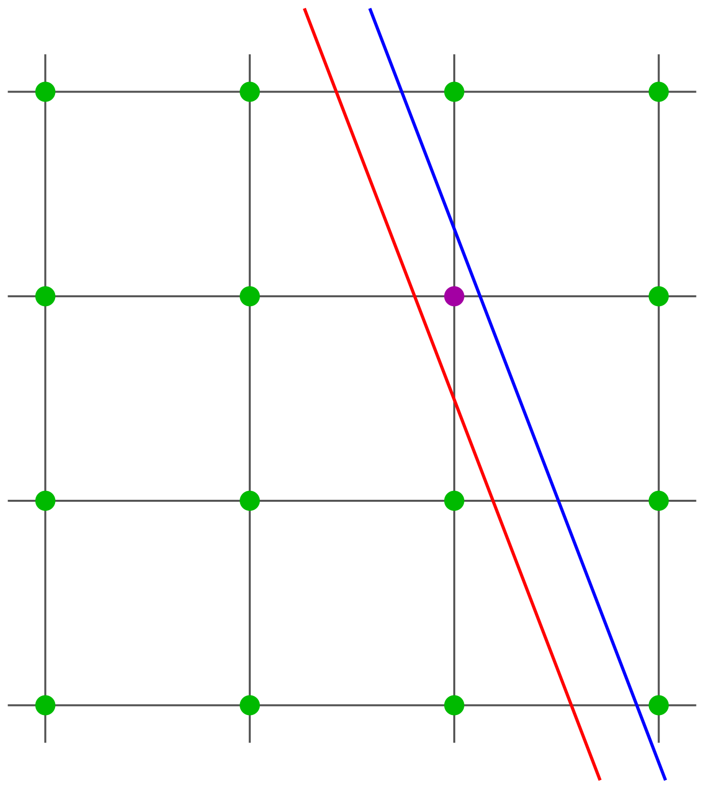

Fig. 8.2: The problem of using signed distance on a grid.

The two inputs laying in difference cubes can cause significant differences in the RMS calculation due to the way partially-trained nodes are handled. Figure 8.2 would result in partially-trained nodes on each side. One way to approach this could be with input-specific smoothing, for example, by looking at the distances of nearby nodes to that same input, then incrementing this by 1 unit.

Currently, the output is “noisy” in the sense that colour transitions on the mesh are often abrupt. It is possible that some form of smoothing could be applied to the SOM, but a balance between a smooth appearance and the loss of data must be found.

8.4 Limitations

8.4.1 Memory–Efficient Surface Reconstruction

As can be seen in chapter 5, the algorithm is sensitive to noise, which can cause artefacts to form. Such artefacts lay outside the surface however, and would be simple to remove with standard algorithms.

The layered reconstruction requires pre-processing for best results, and the optimum parameters for the pre-processing are not always known in advance.

The line-by-line reconstruction sacrifices processing speed for further gains in memory efficiency. Whilst the memory requirements are even lower than the layered reconstruction, the time required to process the Ramesses model was significantly increased.

8.4.2 Feature Detection

The algorithm is sensitive to noise and variability in the input data, which has the same effect on a node's training data as being near a surface feature, namely, a high variance. Such an effect results in false positives; vertices being incorrectly flagged as belonging to a surface feature.

High variability in the input data, for instance, Neptune's beard (which has a large number of ridges in a small area) does present problems, as there is a large amount of interference in the training data. Consequently, with the percentile threshold on training data variance, this prevented other features being flagged. The percentile could be changed in order to detect these features as well, but would come at the cost of false positives.

8.4.3 Mesh Alignment

The implementation of each algorithm was not optimised due to the wide variety of different techniques and circumstances under which each is possible and appropriate. Our implementations took the simple approach of reading the full file from the hard drive, processing the data entirely in memory, and writing the output back to the hard drive in a single execution thread. Since the algorithm is based on PCA, it is not suitable for datasets that are highly rotationally symmetric.

8.4.4 Mesh Comparison

The result from the algorithm is resolution-dependent, and the resolution at which the comparison is best performed must be determined experimentally for each model. However, once the appropriate resolution is known, it will be valid for all future comparisons.

The algorithm, due to highlighting all differences, can appear overly-sensitive if no minimum difference is specified. This can actually be a strength however, as it allows comparison and detection of even the smallest differences between two meshes.

The RMS calculation is sensitive to partially-trained nodes, and can give artificially-inflated values that do not properly reflect the differences between the two meshes.

8.5 Future Work

In this section we suggest potentially fruitful avenues for future research. Not only does each chapter's work contain potential for expansion and refinement, but an SOM may also be able to be used in more areas of the pipeline.

8.5.1 Memory–Efficient Surface Reconstruction

The regular structure of the SOM employed by the proposed algorithm, and the very simple processing operations performed at each node, make the method particularly suitable for GPU implementation. In the future, we plan to work on a GPU implementation of the algorithm which, together with existing GPU implementations of the Marching Cubes algorithm [51], could be a step towards the goal of real-time surface reconstruction.

The implementation of the line-based reconstruction algorithm is not mature. With further work the the memory requirements could be decreased even further than the layer-based reconstructions; the initial results suggest by up to an order of magnitude.

8.5.2 Feature Detection

A more sophisticated statistical analysis of the separation of a single node, or the separations of neighbourhood nodes, is expected to allow the extraction of more reliable feature information. Depending on the characteristics of the training history, it may even be possible to classify features in to different types, such as a ridge, valley or peak.

8.5.3 Mesh Alignment

In the future we plan a systematic analysis of the error of the standard PCA caused by vertex quantisation. Indeed, the small alignment error produced by our method is essentially a vertex coordinate quantisation error, which anyway may be present in the vertex coordinates, if for example the mesh had undergone lossy compression. By showing, as we conjecture, that the alignment error of our method and the vertex PCA error caused by vertex coordinate quantisation are comparable, we will further justify our approach.

8.5.4 Mesh Comparison

A natural extension to this work would be to implement input-specific smoothing such that the number of partially-trained nodes can be reduced. This could be achieved by increasing the size of the parallelepiped around each input in which we train the nodes, though this would come at a significant performance penalty. A more sophisticated statistical analysis of the SOM could also provide a more stable and informative measure of the difference between the inputs.

Further, we hope to investigate potential methods for automatically determining the best resolution at which to run the analysis. Even if the objectively best resolution cannot be proved, the existence of one or two methods that suggest good resolutions could save time for future experimenters.

8.5.5 Normal Estimation

It may be possible to make further use of a lattice and the predefined training procedures to estimate normals in a point cloud that is otherwise devoid of them. We have done some initial work on setting up such an algorithm. The basic step is that only the winning node is trained from sample point, however, the SOM is still smoothed (this is another case where the resolution of the analysis becomes important).

After the point cloud has been processed in this manner and the SOM is fully trained, focus passes to each axis of the SOM in turn. The partial derivative of the node separations all the rows parallel to the x axis is used as an estimate of the x component of each node's normal. After doing this for all rows parallel to the x axis (and thus computing all the x components of the normals), the procedure is repeated for the y and z axes.

For a sufficiently dense point cloud, this would reduce the complexity of the problem to the development of methods for determining the orientation of the normal components.

APPENDIX

A. MESH IMPLEMENTATIONS

In this appendix I provide details on how surfaces are realised within software as meshes. A mesh is a collection of polygons that describes a surface, these polygons (faces) are comprised of vertices and edges. Typically these polygons are triangles for simplicity. The represented surface may or may not have a boundary.

A.1 Data Structures

A.1.1 Indexed Meshes

Indexed meshes are named as such because their vertices are implicitly indexed by their position in a list. This has the advantage that editing the position of a single vertex updates all faces whose perimeters include that vertex. However, such a modification could invalidate the mesh's normal data. With indexed meshes, the only way to find the data relating to a vertex is to sequentially read the list.

It is not possible to skip sections or perform a binary search, since given an individual vertex, there is no way to know what its index is. This is clearly inefficient, particularly when attempting to find the immediate neighbours of a vertex (it's *1-ring*), which would require searching the entirety of the face data. If the mesh can be fully loaded into memory then this is not such an issue, though scattered memory access could result in decreased performance.

A.1.2 Non-Indexed Meshes

Informally referred to as a "triangle soup" (since the faces are almost always triangles), non-indexed meshes store redundant vertex information to reduce the number of linear searches. A set of three coordinates defines a vertex, and faces are implicitly defined by a collection of three vertices. This means that given a face to draw, all the necessary information is immediately to hand, but finding the 1-ring of a vertex is still inefficient.

A.1.3 Half-Edge Data Structure

The Half-Edge Data Structure only stores vertices and edges. Each edge is realised as two directed half-edges, each of which store the following:

1. The face bordered by the half-edge.
2. The vertex the half-edge is leading to (or equivalently, the vertex from which it emanates).
3. It's half-edge pair (the half-edge going in the opposite direction)
4. The next half-edge in the perimeter of the face.

Extra data is often stored to ease implementation, for example; by storing the vertex that a half-edge is coming from as well as the vertex it leads to.

Each vertex stores its 3D coordinates and the index of a half-edge leading to it. The face data is now implicitly encoded in the half-edge data structure, but it is often stored explicitly for ease of manipulation. It is also useful to store face data explicitly when extra information (e.g. normal/texture data) needs to be stored.

The half-edge data structure makes adjacency queries simple; by iterating various simple operations we can easily determine what faces are adjacent to a particular face. In C, such an operation might look like this:

```
do
{
      adjacent_faces[n] = current_edge.face;
      current_edge = current_edge.next;
      n = n + 1;
}
while( current_edge != start_edge );
```

A.2 File Formats

A.2.1 Object File Format

The Object File Format (OFF) has a simple structure, facilitating the development of new tools for its manipulation. It stores an indexed mesh representation. A standard OFF file is laid out as follows:

```
OFF                                        // file type
nF nV nE                                   // Number of faces, vertices and edges.
$x_0$   $y_0$   $z_0$                      // $x_n$ = x coordinate of vertex n
$x_1$   $y_1$   $z_1$                      // Position in the file gives an implicit index.
⋮                                          ⋮
$x_{nV-1}$   $y_{nV-1}$   $z_{nV-1}$       // end of vertex data
$nS_0$   $i_0$   $i_1$   …   $i_{nS-1}$    // $nS_X$ = number of faces that side X has
$nS_1$   $i_0$   $i_1$   …   $i_{nS-1}$    // $i_n$ = index of $n^{th}$ vertex in the perimeter
⋮                                          ⋮
$nS_{nF-1}$   $i_0$   $i_1$   …   $i_{nS-1}$   // end of face data
```

nF is the number of faces, nV; the number of vertices and nE; the number of edges. The number of edges, nE, must required to be present but is rarely used, and is not always accurate since many programs ignore it or just store 0 to comply with the file format specification. After the header comes the vertex data; each line of which consists of the 3D coordinates of a single vertex, implicitly indexed by its position in the file.

Next comes the face data, with one face being described per line. Each line starts with the number of sides of that face and is followed by a list of vertex indices that define the corners of the face. The vertices are listed such that they describe the perimeter of the face, making cyclic permutations and order reversals entirely equivalent. All information has now been provided (either explicitly or implicitly), so the file ends.

A.2.2 Object File Format Variants

There exist two optional extensions to the OFF, one or both may be used at any time. The first adds normal information to each of the vertices

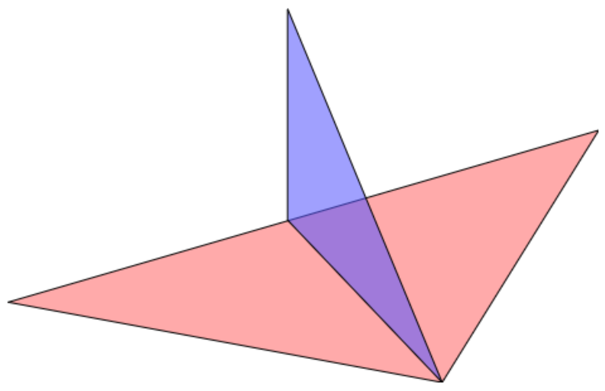

Fig. A.1: Invalid triangle configuration in STL format.

and declares itself to be a NOFF file. The header starts with “NOFF” instead of “OFF”, and each vertex line has the Cartesian components of the normal to the surface at that vertex appended. The face section remains unchanged.

The second extension adds RGB colour information to the faces, and files are declared to be COFF files, starting with “COFF”. If both extensions are used, then the files are CNOFF files, and begin as such.

A.2.3 Stereo Lithography Format

The STereo Lithography file format (STL) describes a single object. The file itself and begins with the word “solid”, and ends with “endsolid”. A facet (face) is begun with the word “facet” and finished with “endfacet”. If the facet has a normal then “normal n_x n_y n_z” follows the facet declaration. Normal information is optional as it can be losslessly generated from the face itself. Within each facet, the perimeter is described by listing the position of each of its vertices. Any two adjacent triangles must share two vertices, and so the triangle configuration shown in figure A.1 is not valid.

A.2.4 Polygon Format

The Stanford PoLYgon (PLY) file format was created in an attempt to unify the wide range of potential file formats that are often used for 3D models. In spite of this, it was not intended to be all-encompassing; it describes precisely one object and does not support a variety of features found in other formats, such as polygons with holes. Element types are declared along with the number instances of that element, and properties are added to describe all the necessary features.

The format was also designed to be extensible, with user-defined data being declared in the header of the file and appearing after the vertex and face subsections (which are identical to those of the OFF). Compatibility is maintained by requiring those data not understood by an interpreting program to be ignored or dropped without impact on the others.

A typical PLY file might start like so:

```
ply
format ascii 1.0
element vertex 35947
property float x
property float y
property float z
element face 69451
property list uchar int vertex_indices
end_header
```

A.2.5 Streaming Formats

Streaming mesh formats are useful when the original mesh is too large to fit in the memory of a workstation. A mesh can be streamed from a local disk[45], or over a network[20]. These formats typically rely on loading a simplified mesh, to show the overall structure, then loading small areas at a higher quality and updated the rendered mesh in real-time. The methods employed can allow useful work to be done on a high-quality subsection of the mesh, without sacrificing the context provided by the lower-quality base mesh.

B. SOFTWARE IMPLEMENTATION

In this appendix I describe the software used throughout this thesis, including languages and external libraries.

B.1 Core Algorithms

The core algorithms are implemented in C for maximum portability across a wide range of hardware devices. Code was written to the C90 standard since this is the most widely supported across all compilers. Source code for all presented algorithms is available under a BSD licence.

C's low-level and highly-optimisable nature made the algorithms viable in terms of speed and memory use, even on some embedded systems. This, combined with the fact that some embedded devices do not have C++ compilers available, meant the software could be quickly adapted to even the most constrained environments.

Except as listed in the "Dependencies" section, all software was written from scratch. The software was built using the following compilers:

- Microsoft Visual Studio
- GCC
- Clang
- TCC

and tested on the following operating systems:

- Microsoft Windows
- Ubuntu Linux

- FreeBSD
- OpenBSD

in order to confirm its standards-compliant and cross-platform nature.

Making use of external dependencies would have required them to support all of the above operating systems. Writing portable code from scratch however, allows the code to be taken and used with compilers and systems other than those listed (for instance, Google's Android, Apple's iOS, or some other mobile operating system).

B.1.1 Dependencies

The only external dependency is in the mesh alignment program (and even then only in one implementation). It depends on the Point Cloud Library[91], PCL, which is available for Windows, Linux, and MacOS X. This was for the convenience of using a pre-existing, optimised, and stable implementation of Principal Components Analysis, against which results could be validated.

The GNU Scientific Library (GSL) provides a near drop-in replacement should more portability be required. It would also be simple to implement this algorithm directly in C, and validate this implementation against either GSL or PCL.

B.1.2 Mesh Format

When creating or processing meshes, the file format must be chosen. The format was required to have the following characteristics:

- Simple to parse.
- Usable by a large number of software tools.
- Able to store colour and normal data.

The format was required to be simple to parse in order that a new parser could be written with no external dependencies. Support by a wide variety of software tools was important in order to allow viewing and analysis of the results. The ability to store colour data was important for working on feature detection and mesh comparison. After taking these requirements into consideration, the Object File Format was selected.

B.1.3 Architecture

Code that could be shared between programs was written into separate files and referenced from there. Algorithm-specific code was kept to a single file that gave a high-level overview of the process. For instance, the only code specific to the layered SOM is the function it uses to store node data on the disk, and its main execution function, reproduced in Listing B.1.

Listing B.1: Source code for "main" function from the layered SOM.

```
int main(int argc, char **argv)
{
    /* perform common setup operations */
    initialise(argc, argv);

    printf("Running layered reconstruction.\n");

    /* enter the main loop of reading and processing samples */
    for (curstep = 1; curstep != numsteps; ++curstep)
    {
        if ( 1 == extract_sample(&sample, pt_cloud, PT_CLOUD_HAS_NORMALS) )
            break;

        /* smooth if samples have moved too far in the z direction */
        if ( fabs(coord_ctrs.z - sample.pos.z) > (double)DUMP_LAYERS)
        {
            smooth_SOM(nodes, &sample, STD_SMOOTH);
            coord_ctrs.z = sample.pos.z;
        }

        /* might try to train up to s.p.z+NORM_EXT */
        if ( (z_base + ACTIVE_Z_EXT - sample.pos.z) < (long)NORM_EXT )
        {
            printf("SOM base now at z = \%ld\n", z_base);

            /* store node data on disk */
            dump_node_data(ndump, nodes, DUMP_LAYERS);

            /* record resource usage information */
            dump_resources(mdump, mem_data);

            /* move the SOM up by DUMP_LAYERS */
            z_base += DUMP_LAYERS;
        }
        /* create data from the sample point and its normal */
        create_training_data(training_pts, sample);

        /* train nodes with newly-created data */
        learn_distances(nodes, training_pts);
    }

    /* compute the final separation of nodes before dumping to a file */
    calculate_separations(nodes, &sample, FINAL_SMOOTH);

    /* smooth and move the SOM until all layers have been stored on-disk */
    for (int i = 0; i != (ACTIVE_Z_EXT / DUMP_LAYERS); ++i)
    {
        smooth_SOM(nodes, &sample, STD_SMOOTH);
        dump_node_data(ndump, nodes, DUMP_LAYERS);
        sample.pos.z += DUMP_LAYERS;
        z_base += DUMP_LAYERS;
    }

    /* perform common cleanup operations */
    deinitialise();

    return EXIT_SUCCESS;
}
```

B.2 Code Samples

Several global variables are set during initialisation. This was often to fully exploit the regularity of the nodes' positions for optimisation in later stages. The specifics of these variables are in the comments in Listing B.2.

Listing B.2: Setting global variables for easy lookups later.

```c
void set_globals(node ***nodes, vector *coord_ctrs,
    double min_x, double min_y, double min_z,
    double max_x, double max_y, double max_z)
{
    long int ni = 0;

    /* SOM_N_EXT is the full extension of the SOM along that axis.
    * ACTIVE_N_EXT is the SOM extension along that axis that is being trained. */
    SOM_X_EXT = ACTIVE_X_EXT = get_side_length(min_x, max_x);
    SOM_Y_EXT = ACTIVE_Y_EXT = get_side_length(min_y, max_y);
    SOM_Z_EXT = get_side_length(min_z, max_z);

    ACTIVE_Z_EXT = SOM_HEIGHT;

    /* used to speed up calculations later */
    X_HALF_SIDE = (long)ceil(ACTIVE_X_EXT / 2.0);
    Y_HALF_SIDE = (long)ceil(ACTIVE_Y_EXT / 2.0);

    nodes_in_layer = ACTIVE_X_EXT * ACTIVE_Y_EXT;
    NUM_NODES = ACTIVE_Z_EXT * nodes_in_layer;

    /* <name>_test: if the node index modulo this is zero, it's on that SOM face */
    back_test = ACTIVE_X_EXT - 1L;
    right_test = ACTIVE_X_EXT * (ACTIVE_Y_EXT - 1L);
    top_test = (ACTIVE_Z_EXT - 1L) * nodes_in_layer;

    /* used to compensate for negative coordinate when finding node indices */
    lookup_addition = X_HALF_SIDE + (Y_HALF_SIDE * ACTIVE_X_EXT);

    if (NUM_NODES == 0L)
    {
        fprintf(stderr, "Side length equal to 0.\n");
        exit(EXIT_FAILURE);
    }

    (*nodes) = calloc((size_t)NUM_NODES, sizeof(**nodes));
    if ((*nodes) == NULL)
    {
        fprintf(stderr, "Insufficient memory for new nodes\n");
        fprintf(stderr, "NUM_NODES = %ld\n", NUM_NODES);
        fprintf(stderr, "SOM_X_EXT = %ld\n", SOM_X_EXT);
        fprintf(stderr, "SOM_Y_EXT = %ld\n", SOM_Y_EXT);
        fprintf(stderr, "SOM_Z_EXT = %ld\n", SOM_Z_EXT);
        exit(EXIT_FAILURE);
    }

    for (ni = 0; ni != NUM_NODES; ++ni)
        (*nodes)[ni] = NULL;

    /* init_z_base stores the active band's initial z coordinate */
    init_z_base = (long)floor(min_z - BASE_POS);

    /* z_base stores the active band's current z coordinate */
    z_base = init_z_base;
    coord_ctrs->z = (double)init_z_base;
}
```

Listing B.3 shows the simplicity with which a node can be found given a set of coordinates. The coordinates must be rounded to the nearest integer (casting the coordinate does not achieve this result). Consequently, we are able to perform simple integer arithmetic to compute the node's index, a process optimised by the variables set in Listing B.2.

Listing B.3: Looking up a node from integral coordinates.

```c
long int node_lookup(short int x, short int y, short int z )
{
    /* node_lookup(...) is sped up by calculating this once and re-using it */
    extern long int lookup_addition;

    if ( x < -X_HALF_SIDE || x >= X_HALF_SIDE ||
         y < -Y_HALF_SIDE || y >= Y_HALF_SIDE ||
         z < 0            || z >= ACTIVE_Z_EXT)
    {
        return -1L;
    }

    return (long)(x + (y * ACTIVE_X_EXT) + (z * nodes_in_layer) + lookup_addition);
}
```

In Listing B.4 another example is shown of how the regular structure of a lattice can be beneficial. In order to find the neighbours of a given node, we can use modular arithmetic to directly determine their indices. One check is performed on the value of the given node's index, and then a simple addition/submission is performed for each neighbour. Again, since some of the values were precomputed (the variable “nodes_in_layer”), we even manage to save performing multiple additions.

Listing B.4: Finding $L_0(1)$ neighbours using only the node index.

```c
void calc_neighbours(long int ni, long int neighbours[])
{
    /* the regular node arrangement allows border detection using only the index */
    neighbours[0] = (ni % ACTIVE_X_EXT == back_test)     ? -1L : ni + 1;
    neighbours[1] = (ni % ACTIVE_X_EXT == 0)             ? -1L : ni - 1;
    neighbours[2] = (ni % nodes_in_layer >= right_test)  ? -1L : ni + ACTIVE_X_EXT;
    neighbours[3] = (ni % nodes_in_layer < ACTIVE_X_EXT) ? -1L : ni - ACTIVE_X_EXT;
    neighbours[4] = (ni >= top_test)                     ? -1L : ni + nodes_in_layer;
    neighbours[5] = (ni < nodes_in_layer)                ? -1L : ni - nodes_in_layer;

    return;
}
```

B.3 Test Data

When creating variant meshes (smoothed, remeshed, noisy, or simplified), MeshLab[21] (an open source tool for rendering and working with 3D models) was used. The filters that modified the files were then stored externally for future use. This ensured reproducible test data creation, a necessity if adding another mesh to a test set, or in case the modified file was lost.

A set of Python scripts were created not only to create the test data in an automatic manner (with no human involvement), but also to run batches of tests. This hands-off approach significantly reduced the possibility of human error influencing the results.

B.4 Analysis of Results

Images of results were created using multiplier (high-resolution) snapshots from MeshLab. 2D and 3D graphs were created using gnuplot, a free, cross-platform, and scriptable plotter. Before each test the mesh/-point cloud variants were created anew.

REFERENCES

[1] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi–based surface reconstruction algorithm. In *SIGGRAPH*, pages 415–422, 1998.

[2] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2-3):127–153, 2001.

[3] Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. In *International Journal of Computational Geometry and Applications*, pages 213–222, 2000.

[4] Mihael Ankerst, Gabi Kastenmller, Hans-Peter Kriegel, and Thomas Seidl. 3d shape histograms for similarity search and classification in spatial databases. In RalfHartmut Gting, Dimitris Papadias, and Fred Lochovsky, editors, *Advances in Spatial Databases*, volume 1651 of *Lecture Notes in Computer Science*, pages 207–226. Springer Berlin Heidelberg, 1999.

[5] Craig Gotsman Gill Barequet and Avishay Sidlesky. Polygon reconstruction from line cross-sections. In *Proceedings of the 18th Canadian Conference on Computational Geometry (CCCG'06)*, pages 81–84, 2006.

[6] J. Barhak and A. Fischer. Adaptive reconstruction of freeform objects with 3D SOM neural network grids. In *Pacific Graphics*, pages 97–105, 2001.

[7] J. Barhak and A. Fischer. Adaptive reconstruction of freeform objects with 3D SOM neural network grids. *Computers and Graphics*, 26(5):745–751, 2002.

[8] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006.

[10] W. Boehler, Bordas M. Vicent, and A. Marbs. Investigating laser scanner accuracy. In *Proceedings of XIXth CIPA International Symposium*, September 2003.

[11] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel poisson surface reconstruction. In *Advances in Visual Computing*, volume 5875 of *Lecture Notes in Computer Science*, pages 678–689. Springer-Verlag, 2009.

[12] Michael Bosse, Robert Zlot, and P. Flick. Zebedee: Design of a spring-mounted 3-d range sensor with application to mobile mapping. *Robotics, IEEE Transactions on*, 28(5):1104–1119, Oct 2012.

[13] Botsch. Mesh data structures. In *Geometric Modelling Based On Polygonal Meshes, Course Notes*. SIGGRAPH, 2007.

[14] Botsch. Model repair. In *Geometric Modelling Based On Polygonal Meshes, Course Notes*. SIGGRAPH, 2007.

[15] Botsch. Surface representations. In *Geometric Modelling Based On Polygonal Meshes, Course Notes*. SIGGRAPH, 2007.

[16] Benedict Brown and Szymon Rusinkiewicz. Global non-rigid alignment of 3-D scans. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26(3), August 2007.

[17] Virginio Cantoni, Alessandro Gaggia, and Luca Lombardi. Extended Gaussian image. In *Encyclopedia of Systems Biology*, pages 724–725. Springer, 2013.

[18] J. C. Carr. Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[19] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth surface reconstruction from noisy range data. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, New York, NY, USA, 2003. ACM.

[20] Bing-Yu Chen and Tomoyuki Nishita. Multiresolution streaming mesh with shape preserving and qoS-like controlling. In *Web3D*, pages 35–42, 2002.

[21] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Sixth Eurographics Italian Chapter Conference*, pages 129–136, 2008.

[22] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum*, 17(2):167–174, 1998.

[23] P. Crossno and E. Angel. Spiraling edge: fast surface reconstruction from partially organized sample points. In *Proc. Visualization '99*, pages 317–538, 1999.

[24] J. Giesen T. K. Dey and J. Hudson. Delaunay based shape reconstruction from large data. In *IEEE Symposium in Parallel and Large Data Visualization and Graphics*, pages 19–27, 2001.

[25] T. Dey and N. Leekha. Surface reconstruction simplified, 1999.

[26] T. K. Dey and J. Giesen. Detecting undersampling in surface reconstruction. In *Proc. 17th ACM Symposium on Computational Geometry*, pages 257–263, 2001.

[27] Tamal K. Dey and Samrat Goswami. Tight cocone: A water-tight surface reconstructor. *J. Comput. Inf. Sci. Eng.*, 3(4):302–307, 2003.

[28] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. In *SCG '04: Proc. of the 20th annual symposium on Computational geometry*, pages 330–339, 2004.

[29] Tamal K. Dey and Jian Sun. Normal and feature approximations from noisy point clouds. In *Foundations of Software Technology and Theoretical Computer Science*, pages 21–32, 2006.

[30] Diaz-Andreu, Margarita, Hobbs, Richard, Rosser, Nick, Sharpe, Kate, and Trinks. Long meg: Rock art recording using 3D laser scanning. *Past (The Newsletter of the Prehistoric Society)*, 50:2–6, 2005.

[31] Michael Elad, Ayellet Tal, and Sigal Ar. Content based retrieval of vrml objects: an iterative and interactive approach. In *Proceedings of the sixth Eurographics workshop on Multimedia 2001*, pages 107–118. Springer-Verlag, 2002.

[32] Viacheslav Filonenko, Charlie Cullen, and James D. Carswell. Indoor positioning for smartphones using asynchronous ultrasound trilateration. *ISPRS International Journal of Geo-Information*, 2(3):598, 2013.

[33] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. In *SIGGRAPH*, pages 544–552, 2005.

[34] B. Fritzke. Growing Cell Structures - a self organizing network for unsupervised and supervised learning. Technical Report ICSTR-93-026, ICSI, Berkeley, 1993.

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

[36] Bernd Fritzke. Growing self-organizing networks - why? In *In ESANN96: European Symposium on Artificial Neural Networks*, pages 61–72. Publishers, 1996.

[37] Claudia Fuchs and Stephan Heuel. Feature extraction. In *Proc. of Third Course in Digital Photogrammetry*, 1998.

[38] Natasha Gelfand, Szymon Rusinkiewicz, Leslie Ikemoto, and Marc Levoy. Geometrically stable sampling for the icp algorithm. In *3DIM'03*, pages 260–267, 2003.

[39] Stefan Gumhold, Xinlong Wang, and Rob Macleod. Feature extraction from point clouds. In *In Proceedings of the 10 th International Meshing Roundtable*, pages 293–305, 2001.

[40] Norbert Haala, Claus Brenner, and Karl heinrich Anders. 3D urban GIS from laser altimeter and 2D map data. *Int'l Archives Photogrammetry and Remote Sensing*, 32(3):339–346, 1998.

[41] Norbert Haala, Michael Peter, Alessandro Cefalu, and Jens Kremer. Mobile Lidar Mapping For Urban Data Capture. In M. Loannides, A. Addison, A. Georgopoulos, and L. Kalisperis, editors, *14th International Conference on Virtual Systems and Multimedia (VSMM 2008)*, pages 95–100. Archaeolingua, October 2008.

[42] M. Hoffmann and L. Várady. Free-form modelling surfaces for scattered data by neural networks. *Journal for Geometry and Graphics*, 1:1–6, 1998.

[43] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH*, pages 71–78, 1992.

[44] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Wiley-Interscience, 2001.

[45] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *IEEE Visualization*, page 30, 2005.

[46] I. Ivrissimtzis, W.-K. Jeong, and H.-P. Seidel. Using growing cell structures for surface reconstruction. In *SMI*, pages 78–86, 2003.

[47] Ioannis Ivrissimtzis, Won-Ki Jeong, Seungyong Lee, Yunjin Lee, and Hans-Peter Seidel. Surface reconstruction based on neural meshes. In *Proceedings Mathematical Methods for Curves and Surfaces*, pages 223–242, 2007.

[48] Ioannis Ivrissimtzis, Yunjin Lee, Seungyong Lee, Won-Ki Jeong, and Hans-Peter Seidel. Neural mesh ensembles. *3D Data Processing Visualization and Transmission, International Symposium on*, 0:308–315, 2004.

[49] Philipp Jenke, Michael Wand, Martin Bokeloh, Andreas Schilling, and Wolfgang Straßer. Bayesian point cloud reconstruction. *Computer Graphics Forum*, 25(3):379–388, 2006.

[50] Xiangmin Jiao and Michael T. Heath. Feature detection for surface meshes. In *Proceedings of 8th International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 705–714, 2002.

[51] Gunnar Johansson and Hamish Carr. Accelerating marching cubes with graphics hardware. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*. ACM, 2006.

[52] Sakdirat Kaewunruen. Identification and prioritization of rail squat defects in the field using rail magnetisation technology. In *Proc. SPIE 9437*, volume 9437, pages 94371H–94371H–11, 2015.

[53] Thomas Kanzok, Falk S, Lars Linsen, and Paul Rosenthal. Efficient removal of inconsistencies in large multi-scan point clouds. In Vaclav Skala, editor, *Communication Paper Proceedings of WSCG*, pages 120–129, Plzen, Czech Republic, 2013. UNION Agency – Science Press.

[54] David Kaye and Ioannis Ivrissimtzis. Implicit surface reconstruction and feature detection with a learning algorithm. In John Collomosse and Ian Grimstead, editors, *Theory and Practice of Computer Graphics*, pages 127–130, Sheffield, United Kingdom, 2010. Eurographics Association.

[55] David Kaye and Ioannis Ivrissimtzis. Mesh alignment using grid based pca. In *GRAPP*, Berlin, Germany, 2015. Springer.

[56] David Paul Kaye and Ioannis Ivrissimtzis. Memory efficient surface reconstruction based on self organising maps. In Ian Grimstead and Hamish Carr, editors, *Theory and Practice of Computer Graphics*, pages 25–32, Warwick, United Kingdom, 2011. Eurographics Association.

[57] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, pages 61–70, 2006.

[58] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *SGP '03*, 2003.

[59] Y. Kil, B. Mederos, and N. Amenta. Laser scanner super-resolution. In *SoPBG*, pages 9–16, 2006.

[60] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM, 2001.

[61] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[62] Ravikrishna Kolluri. Provably good moving least squares. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1008–1017, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[63] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, and P. Grussenmeyer. First Experiences with Kinect v2 Sensor for Close Range 3d Modelling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 93–100, February 2015.

[64] Thiago F. Leal, Aruquia B. M. Peixoto, Cassia I. G. Silva, Marcelo de A. Dreux, and Carlos A. de Moura. Local changes in marching cubes to generate less degenerated triangles. In *Proceedings of the 10th International Conference on Computer Graphics Theory and Applications (VISIGRAPP 2015)*, pages 143–149, 2015.

[65] Y. Lee, M. Yoon, S. Lee, I. Ivrissimtzis, and H.-P. Seidel. Ensembles for surface reconstruction. In *Proc. of Pacific Graphics*, pages 1–2, 2005.

[66] David Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531, 1998.

[67] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[68] 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.

[69] Martin M. Lipschutz. *Differential Geometry.* The McGraw-Hill Companies, 1969.

[70] Shengjun Liu and Charlie C.L. Wang. Orienting unorganized points for surface reconstruction. *Computers & Graphics*, 34(3):209 – 218, 2010. Shape Modelling International (SMI) Conference 2010.

[71] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algoritm. *Computer Graphics*, 21(4):163–169, 1987.

[72] Lena Maier-Hein, Thiago R. dos Santos, Alfred M. Franz, and Hans-Peter Meinzer. Iterative closest point algorithm in the presence of anisotropic noise. In *Bildverarbeitung fr die Medizin'10*, pages 231–235, 2010.

[73] Thomas Martinetz and Klaus Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[74] Boris Mederos, Luiz Velho, Luiz Henrique, and De Figueiredo. H.: Moving least squares multiresolution surface approximation. In *In Proceedings of SIBGRAPI*, 2003.

[75] Niloy Mitra, An Nguyen, and Leonidas Guibas. Estimating surface normals in noisy point cloud data. *International J. of Computational Geometry and Applications*, 4:261–276, 2004.

[76] Yukie Nagai, Yutaka Ohtake, and Hiromasa Suzuki. Smoothing of partition of unity implicit surfaces for noise robust surface reconstruction. *Computer Graphics Forum*, 28(5):1339–1348, 2009.

[77] Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. Technical report, NIST, 1992.

[78] D. Nehab, S. Rusinkiewicz, J. Davis, and R. Ramamoorthi. Efficiently combining positions and normals for precise 3D geometry. In *SIGGRAPH*, pages 536–543, 2005.

[79] Y. Ohtake, A. Belyaev, and H.-P. Seidel. 3D scattered data approximation with adaptive compactly supported radial basis functions. In *SMI*, pages 31–39, 2004.

[80] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. In *SIGGRAPH*, pages 463–470, 2003.

[81] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Shape Modeling International 2003*. IEEE Computer Society, 2003.

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

[83] Panagiotis Papadakis, Ioannis Pratikakis, Stavros Perantonis, and Theoharis Theoharis. Efficient 3D shape matching and retrieval using a concrete radialized spherical projection representation. *Pattern Recogn.*, 40(9):2437–2452, September 2007.

[84] Jason R. Parent, John C. Volin, and Daniel L. Civco. A fully-automated approach to land cover mapping with airborne lidar and

high resolution multispectral imagery in a forested suburban landscape. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 104:18 – 29, 2015.

[85] M. Pauly, R. Keiser, and M. Gross. Estimation of planar curves. *Comp. Graph. Forum*, 22(3):281–289, 2003.

[86] Jianbo Peng, Vasily Strela, and Denis Zorin. A simple algorithm for surface denoising. In *Proceedings of the Conference on Visualization '01*, VIS '01, pages 107–112, Washington, DC, USA, 2001. IEEE Computer Society.

[87] Gheorghe Postelnicu, Lilla ZÃ¶llei, and Bruce Fischl. Combined volumetric and surface registration. *IEEE Trans Med Imaging*, 28(4):508–22, April 2009.

[88] Jeroen De Reu, Gertjan Plets, Geert Verhoeven, Philippe De Smedt, Machteld Bats, Bart Cherrett, Wouter De Maeyer, Jasper Deconynck, Davy Herremans, Pieter Laloo, Marc Van Meirvenne, and Wim De Clercq. Towards a three-dimensional cost-effective registration of the archaeological heritage. *Journal of Archaeological Science*, 40(2):1108 – 1121, 2013.

[89] Paul A. Rosen, Scott Hensley, Ian R. Joughin, Fuk K. Li, Sren N. Madsen, Senior Member, Ernesto Rodrguez, and Richard M. Goldstein. Synthetic aperture radar interferometry. In *Proceedings of the IEEE*, pages 333–382, 2000.

[90] Michaël Roy, Sebti Foufou, and Frédéric Truchetet. Mesh comparison using attribute deviation metric. *Journal of Image and Graphics*, 4:1–14, 2004.

[91] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (PCl). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[92] H. S. Sahambi and K. Khorasani. A neural-network appearance-based 3-D object recognition using independent component analysis. *IEEE Trans. Neur. Netw.*, 14(1):138–149, 2003.

[93] S. Schaefer and J. Warren. Dual marching cubes: primal contouring of dual grids. In *Proc. 12th Pacific Conference on Computer Graphics and Applications PG 2004*, pages 70–76, 2004.

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

[95] M. Schmitt. Three-dimensional reconstruction of urban areas by multi-aspect tomosar data fusion. In *Urban Remote Sensing Event (JURSE), 2015 Joint*, pages 1–4, March 2015.

[96] Konstantinos Sfikas, Theoharis Theoharis, and Ioannis Pratikakis. Pose normalization of 3d models via reflective symmetry on panoramic views. *The Visual Computer*, 30(11):1261–1274, 2014.

[97] Hoi Sheung and Charlie C. L. Wang. Robust mesh reconstruction from unoriented noisy points. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 13–24, New York, NY, USA, 2009. ACM.

[98] P. N. Suganthan. Shape indexing using self-organizing maps. *IEEE Transactions on Neural Networks*, 13(4):835–840, July 2002.

[99] Compton J. Tucker. Red and photographic infrared linear combinations for monitoring vegetation. *Remote Sensing of Environment*, 8(2):127 – 150, 1979.

[100] G. Turk and J. O'Brien. Modelling with implicit surfaces that interpolate. *ACM ToG*, 21(4):885–873, 2002.

[101] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[102] L. Várady, M. Hoffmann, and E. Kovács. Improved free-form modelling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics*, 3:177–181, 1999.

[103] DV Vranic, D Saupe, and J Richter. Tools for 3d-object retrieval: Karhunen-loeve transform and spherical harmonics. In *Workshop on Multimedia Signal Processing*, 2001.

[104] Jun Wang, Kai Xu, Ligang Liu, Junjie Cao, Shengjun Liu, Zeyun Yu, and Xianfeng David Gu. Consolidation of low-quality point clouds from outdoor scenes. *Computer Graphics Forum*, 32(5):207–216, 2013.

[105] Zoë Wood, Hugues Hoppe, Mathieu Desbrun, and Peter Schröder. Removing excess topology from isosurfaces. *ACM Trans. Graph.*, 23(2):190–208, April 2004.

[106] Micheol Yoon, Ioannis Ivrissimtzis, and Seungyong Lee. Self-organising maps for implicit surface reconstruction. In *UK Theory and Practice of Computer Graphics*. Eurographics Association, 2008.

[107] Mincheol Yoon, Ioannis P. Ivrissimtzis, and Seungyong Lee. Variational bayesian noise estimation of point sets. *Computers & Graphics*, 33(3):226–234, 2009.

[108] Mincheol Yoon, Yunjin Lee, Seungyong Lee, Ioannis Ivrissimtzis, and Hans-Peter Seidel. Surface and normal ensembles for surface reconstruction. *Computer-Aided Design*, 39(5):408–420, 2007.

[109] S. Yoshizawa, A. Belyaev, and H.-P. Seidel. Fast and robust detection of crest lines on meshes. In *Symposium on Solid and Physical Modeling*, pages 227–232. ACM Press, 2005.

[110] Y. Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization*, pages 61–64, 1999.

[111] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti. Performance evaluation of the 1st and 2nd generation kinect for multimedia applications. In *Multimedia and Expo (ICME), 2015 IEEE International Conference on*, pages 1–6, June 2015.

[112] Juanjuan Zhu, Richard P. Collins, Joby B. Boxall, Robin S. Mills, and Rob Dwyer-joyce. Non-destructive in-situ condition assessment of plastic pipe using ultrasound. *Procedia Engineering*, 119:148 – 157, 2015. Computing and Control for the Water Industry (CCWI2015) Sharing the best practice in water management.

[113] Lingli Zhu, Matti Lehtomki, Juha Hyypp, Eetu Puttonen, Anssi Krooks, and Hannu Hyypp. Automated 3d scene reconstruction from open geospatial data sources: Airborne laser scanning and a 2d topographic database. *Remote Sensing*, 7(6):6710, 2015.