3D Modelling Using Partial Differential Equations (PDEs)

Abdusslam Osman

A thesis submitted in partial fulfilment of the requirements of

Sheffield Hallam University

for the degree of Doctor of Philosophy

October 2014

Table of Contents

List of Figures

List of Tables

Acronyms & Abbreviations

API	Application programming interface
ASCII	American Standard Code for Information Interchange
B-spline	Basis spline
CAD	Computer aided design
CAM	Computer Aided Manufacturing
COLLADA	COLLAborative Design Activity
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DE	Differential Equation
DFT	Discreate Fourier Transform
DWT	Discrete Wavelet Transform
DirectX	A collection of application programming interfaces (APIs)
EB	Exabytes
EBCDIC	Extended Binary Coded Decimal Interchange Code
FDM	Finite Difference Method
FEM	Finite Element Method
FFT	Fast Fourier Transform

FWT	Fast Wavelet Transform
GMPR	Geometric Modelling and Pattern Recognition Group
HPF	High pass filtration
HTML	HyperText Markup Language
JPEG	Joint Photographic Expert Group
JSON	Java Script Object Notation
Java3D	3D application programming interface for the Java platform
LPF	Low pass filtration
MLS	The moving least squares
MOL	The method of lines
MPEG-4	Moving Picture Experts Group
Matlab	Matrix laboratory
NURBS	Non-uniform rational basis spline
OBJ	An object file
ODE	Ordinary Differential Equation
OpenGL ARB	The OpenGL Architecture Review Board
PB	Petabytes
PC	Personal Computer
PDE	Partial Differential Equation
QoS	Quality of Service
R^2	Coefficient of determination

RAM	Random access Memory
RMSE	Root Mean Square Error
SSE	Sum of Squared Errors of Prediction
SST	Total Sum of Squares
VRML	Virtual Reality Modeling Language
XML	Extensible Markup Language
iDCT	Inverse Discrete Cosine Transform
iDWT	Inverse Discrete Wavelet Transform
iFFT	Inverse Fast Fourier Transform

Abstract

Partial differential equations (PDEs) are used in a wide variety of contexts in computer science ranging from object geometric modelling to simulation of natural phenomena such as solar flares, and generation of realistic dynamic behaviour in virtual environments including variables such as motion, velocity and acceleration. A major challenge that has occupied many players in geometric modelling and computer graphics is the accurate representation of human facial geometry in 3D. The acquisition, representation and reconstruction of such geometries are crucial for an extensive range of uses, such as in 3D face recognition, virtual realism presentations, facial appearance simulations and computer-based plastic surgery applications among others. The principle aim of this thesis should be to tackle methods for the representation and reconstruction of 3D geometry of human faces depending on the use of partial differential equations and to enable the compression of such 3D data for faster transmission over the Internet. The actual suggested techniques are based on sampling surface points at the intersection of horizontal and vertical mesh cutting planes. The set of sampled points contains the explicit structure of the cutting planes with three important consequences: 1) points in the plane can be defined as a one dimensional signal and are thus, subject to a number of compression techniques; 2) any two mesh cutting planes can be used as PDE boundary conditions in a rectangular domain; and 3) no connectivity information needs to be coded as the explicit structure of the vertices in 3D renders surface triangulation a straightforward task. This dissertation proposes and demonstrates novel algorithms for compression and uncompression of 3D meshes using a variety of techniques namely polynomial interpolation, Discrete Cosine Transform, Discrete Fourier Transform, and Discrete Wavelet Transform in connection with partial differential equations. In particular, the effectiveness of the partial differential equations based method for 3D surface reconstruction is shown to reduce the mesh over 98.2% making it an appropriate technique to represent complex geometries for transmission over the network.

Acknowledgements

The successful completion of this study would not have been possible without the involvement of a number of people and institutions. First, I wish to acknowledge my sincere gratitude to my supervisor, Professor Marcos Rodrigues. His keen enthusiasms for work and constant good humoured spirits have been a source of encouragement to me. His comments and criticisms have been appreciated as much as his learned guidance and advice during the past three years of study and preparation of this thesis.

Second, I would like to sincerely thank my examiners, Dr. Leonardo Bottaci and Dr. David Cooper, for their valuable comments, and for raising interesting points in their report, leading to improvements in this thesis.

Third, I gratefully acknowledge the financial support provided by the Libyan higher education, and Sheffield Hallam University for granting me the opportunity to pursue the Doctor of Philosophy Degree in this institution.

Forth, there is a long list of people who provided help and support. It is impossible to mention all of them, but my special thanks go to my second supervisor Dr Alan Robinson, Prof Ann Macaskill, Tracey Holmes, and all colleagues from the GMPR research group for their guidance and support. To all those who supported me in one way or another but have not been mentioned individually, please accept my gratitude.

Finally, I would like to thank my wife Huda, my son Eecam and daughters Elham, Hala, and all my family for their continuous support and encouragement over the years of my studies, and for always believing in me.

Chapter 1

Introduction

1.1 Scope of the Research

Within three dimensional computer graphics, 3D modelling is the process of creating the numerical rendering from the three-dimensional surface or volumetric/-solid representation of the object via specialised software. Models could be shown like a two-dimensional image via a course of action referred to as 3d render or even used in your personal computer simulation associated with actual phenomena. Such graphical models (a surface or volumetric) are normally designed and constructed using CAD-Computer Aided Design software or acquired through 3D scanners. Once defined in an appropriate format, any 3D model can be printed out using specialised 3D printing devices. This thesis is only concerned with 3D surface data; in particular, surface patches defined on a regular xy-grid where the depth of each point is defined in the z-axis. Such surface patches are typical of data acquired using conventional 3D scanners based on stereo system perspective, structured light or time-of-flight techniques.

This thesis addresses the issue associated with three dimensional data compression as well as uncompressing applied to closed surface patches. Compression means to represent the data with fewer bits than the original representation and can be lossy or lossless. In lossy compression, some information is lost, while lossless means no loss of information. This research only describes lossy compression. Data uncompressing is the process of recovering the original data from the compressed data and normally this is achieved by reversing each step of the process of the compression algorithm. When one refers to compression it normally means both the process of compression and uncompressing. Unless specifically stated otherwise, this thesis uses the word ‘compression’ in this context.

The research approach to 3D compression described in this thesis follows four steps:

1. To investigate a method to define structured geometric information;
2. To investigate polynomial interpolation techniques;
3. To investigate the use of partial differential equations; and
4. To perform comparative analyses with related data compression techniques applied to the 3D case, such as Discrete Fourier, Wavelet, and Cosine transforms.

In the techniques proposed in this dissertation, first, a polygon reduction described in Chapter 4 is applied to the mesh resulting in a set of vertices lying in structured planes of a sparse, regular grid. The data defined on such grids with their practical implementation issues and incorporation into compression techniques are discussed in Chapters 5–7.

The first approach to compression described in Chapter 5 is by using polynomial interpolation. The technique is applied to surface patches and it is shown to be capable of decreasing the mesh by more than 99%. However, there are some limitations such as lack of precision, and for polynomials of higher degree the 3D surface becomes unstable and with smaller compression rates.

The second approach, considered in Chapter 6, is to perform 3D compression based on partial differential equations (PDEs). Compression and 3D surface reconstruction using PDEs have never been solved before in this way. These new methods are tested in various experimental setups and their effectiveness is evaluated and discussed.

In Chapter 7 new methods for 3D data compression and reconstruction are proposed and demonstrated. Upon applying a method of polygon reduction, the vectors describing the data are parametrically defined and a comparative analysis is presented via the Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). The transform coefficients are further processed according to a quality factor, which substantially decreases the amount of data. The file formats are defined with the necessary parameters for a full reconstruction of the sparse mesh. Finally, in order to recover the vertex density of the original mesh, the reconstructed data are represented by elliptic Partial Differential Equations (PDE) and iteratively solved between adjacent planes in connection with the Laplace equation. Experiments demonstrate the effectiveness of the methods allowing compression rates of over 98% compared to the OBJ file format and over 91% compared to a list of vertices in ASCII format.

1.2 Background and Motivation

Current improvements in three dimensional modelling have led to a common number of applications in most areas of science and engineering. Three dimensional objects are now widely used in applications such as games, mechanical and architectural design, archaeology, as well as medical engineering among others. The actual common integration associated with 3D models in different fields motivates the need to be able to store, list, classify, and recover 3D objects automatically and efficiently.

While computer-aided geometric design and computer-aided manufacturing systems tend to be widely used for the design and development of physical objects from digital models, the reverse problem, that of inferring a digital description of an existing physical object, has received much less attention. In addition, in several programs, it will be important to transfer 3D image types over the web to share CAD/CAM models with e-commerce clients, to upgrade material with regard to entertainment applications, or to support collaborative design, research, and show of technological innovation as well as scientific data sets. Bandwidth limitations and storage space restrict the transmission and use of 3D data over the network.

Data compression techniques tend to be centred on representing the actual geometry and connectivity of the vertices in the triangulated mesh. There has been no systematic approach to the geometric parameterization associated with arbitrary 3D objects aiming at efficient representation and compression. As a result, a major concern of this research is to define the possibilities associated with the compression of 3D data for fast transmission over the Internet, without lack of precision and performance.

To achieve this, the thesis involves both theoretical and practical work. The main theoretical work involves the development of mathematical methods for efficient representation and parameterization of PDE-based models as well as geometry optimisation methods for efficient fitting of PDE models to data and efficient encoding of the residuals. This really is achieved by solving a second order, elliptic PDE uses the method of lines to generate a surface from the solution to those equations. Practical work involves the implementation of the methods within the software; what is addressed here is 3D compression by a number of methods and techniques, which is subject to experimentation regarding overall performance and stability.

Within the GMPR Research Group we have developed methods for fast 3D reconstruction using line projection [Robinson et al., 2004; Rodrigues et al., 2007, 2008, 2006]. The method is based on projecting a pattern of lines on the target surface and processing the captured 2D image from a single shot into a point cloud of vertices in 3D space. The reconstructed models are realistic and capture the real Euclidean measurements of the object, and are useful for a large number of applications including, among others, biometric facial recognition, industrial inspection, reverse engineering and multimedia applications. A realistic scenario which is explored in this study involves 3D facial biometric verification at airports. The method is non-intrusive and aims at minimal disruption. It is based on our past experience with 3D biometrics at Heathrow Airport (London, UK) in 2005. In this scenario, an enrolment shot is taken and reconstructed in 3D at an automated check-in desk, where a new database is created for each flight. At the gate before boarding the plane another 3D shot is taken for verification.

The created databases are transmitted to the local Police who perform a search against their records. If the Police find no information to warrant keeping the data for longer, all data must be erased after a time lapse, normally within 24 hours. For international flights and where no mechanisms for sharing information between Police Forces are available, the data can be transmitted to the destination Police authorities *before* the flight actually arrives at the destination. A significant constraint of this scenario is that 3D files are very large; a high definition 3D model of a person's face is around 20MB. For a flight with 400 passengers, this would mean dispatching 8GB of data. If one considers the number of daily flights in a medium sized airport, it can be concluded that this may be unworkable. It is clear that methods to compress 3D data would be beneficial to the scenario considered here but, more importantly, would represent an enabling technology for a large number of other potential applications. For instance, the application of simple texture mapping would lead to the creation of naturally-looking facial images, but on the other hand, conceal the individuality of the subject in the 3D face geometry. Apart from the aspects of privacy, confidentiality, and security concerns, the point being made here is that without data compression it is impossible to make such a scheme work. About 70 million passengers go through London Heathrow per year, almost 200,000 per day. Each high density facial scan takes about 20MB of disk space, so one would be contemplating about 4TB (terabytes) of data per day and 1.4PB (petabytes) per year. To dispatch such a vast amount of data over the network to the local police station and potentially to the origin and destination police authorities is unworkable with current technologies.

Although some standards exist for 3D compression, such as Java 3D and MPEG4, the compression rates are still low for general sharing of files over the

Internet. In general, there are three methods one can use to share 3D data. The first method is based on image compression where each snapshot of a 3D scene is compressed as a 2D image. The second method is based on hierarchical improvement of a 3D structure with regard to transmission, where a coarse mesh is followed by increasing refinements until the original, full 3D model is reconstructed in the other end. The third method is based on mesh compression where algorithms traverse the mesh for a local compression of polygonal relationships.

The principle of compression proposed here is inspired by the GMPR scanning method and its resulting mesh properties. The first step described in Chapter 4 is to cut an arbitrary triangulated mesh with a suitable number of horizontal and vertical cutting planes and detect the intersection point of such planes on the mesh. In order to code the mesh, the (x,y) coordinates are directly given by the distances between the planes, so there is no need to code any (x,y) values explicitly. Only the z-values are subject to compression schemes. The method is not lossless and this research investigates compression techniques for the z-values based on polynomial interpolation and also using PDEs for surface reconstruction. Therefore, for a generic surface path the method involves cutting a number of planes parallel to the Y-axis (or X-axis) of the 3D unconstrained point cloud; then for each plane, finding the points in the structure intercepted by each plane (within a threshold). In this way, an equivalent scan line structure as in the GMPR scanning method is obtained.

This thesis investigates new methodologies on geometric coding of single-value functions where the connectivity is explicitly derived from geometry. Methods for single-value functions are demonstrated in Chapter 5 where connectivity is not coded at all. Once the geometry is coded, compressions over 99% are achieved through the method of re-meshing the structure and representing the (x,y,z), in parametric form using polynomial interpolation.

In Chapter 5 this thesis investigates the use of PDE mesh surfaces for compression and reconstruction of large data files without loss of accuracy, extending the work described in [Rodrigues et al. 2010].The parameterization of PDE-based models are proposed in a way rather different from the previous work on polynomial interpolation highlighted above. Here it is proposed to represent the geometry and connectivity of the mesh by means of solving an elliptic PDE. A perceived advantage of the PDE-based approach is that it defines shapes by means of data distributed around the shape boundaries and feature points only. This approach contrasts with mesh models and spline surfaces, which often require hundreds of control points in order to represent a realistic object. However, it is noted that to date there has been no systematic approach to the geometric parameterization of arbitrary 3D objects aiming at efficient representation and compression.

The main idea is to compress the geometry of each (sparse) cutting plane separately using either Fourier, Discrete Cosine or Discrete Wavelet transforms. And then on the uncompressing stage, use each pair of such planes in turn as boundary conditions for an elliptic PDE and iteratively solve the Laplace equation between the boundaries by the method of lines. The connectivity of the mesh is directly derived from solutions to the Laplace equations and the boundary planes. Therefore, information on the number of vertices as well as a scale of the surface together with the set of points lying in each cutting plane are integral components of the PDE parameters. Cutting planes are used as boundary conditions and there is no dependency on time.

The validation of the proposed method is a demanding task. In general, for each 3D model is not known what is a structure and what is noise in the data. While the PDE method can in theory model the original data set within a prescribed error, it may not be possible to make a strong statement on the validity of the method given the discrete nature of the 3D data, which is in itself an approximation of the real world. It is anticipated that the set of 3D data would be defined parametrically. The thesis describes methods and compression algorithms with experimental results and discusses the suitability of the techniques to a number of applications and general issues in 3D compression and reconstruction. In particular, for each 3D data structure the same tests are performed using the following methods: Fourier, Discrete Cosine and Discrete Wavelet Transforms. The comparative analysis of the techniques is presented by illustrating the Gaussian approximation error distance of different methods.

1.3 Aims and Objectives

The **aims** of research are to demonstrate that PDE based modelling with geometry re-meshing operations can effectively be used for 3D data compression of mesh geometry and connectivity. The approach is different from current methods that are based on coding, connectivity having geometry as a dependent property; the proposed methods are based on geometry coding with connectivity derived from geometry.

The **objectives** are identified as follows for arbitrary surface patches:

- To define a re-meshing method for efficient geometry coding through mesh cutting planes in XY-directions.
- To define the possibilities associated with the compression of 3D data for fast transmission over the Internet. Assuming that effective compression can be achieved, would the proposed scheme yield satisfactory results?
- To collect statistics on the bit rate of such representation and compare with existing polynomial as defined in [Rodrigues et al., 2010], and related work in the literature.
- To investigate and define methods for PDE representation of plane intersections using Laplace and Fourier spaces and alternative representations.
- To define an optimal method for PDE representation from the results of the investigation.

Given an arbitrary surface patch, the proposed method is based on determining the mesh intersection of structured cutting planes in horizontal and vertical directions. Each intersection point is a vertex defined on a regular xy-grid where the z-value is the depth of each vertex. To compress and decompress 3D data, what is first proposed is an interpolation of the z-values by high degree polynomials. Second, a method is proposed for Fourier based data compression and PDE based data uncompression. Finally, a comparative analysis of the PDE method is presented via the DFT, DCT and DWT methods.

1.4 Contributions to Knowledge

This thesis presents a novel approach to accurate, efficient representation and compression of 3D data compression centred on the parameterization of surface patches. The major contributions made by this work are as follows:

- In the first approach using interpolation of polynomials of high degree from 30 to 80 degrees, the result shows a mesh reduction of over 99% compared to the OBJ file format.
- A new approach was taken for 3D compression and reconstruction using the method of lines to solve elliptic PDE, achieving a compression rate of over 98% compared to the OBJ file format. The methods are based on DFT to reconstruct the original data from the vertices lying in each plane. Theoretical results, in addition to numerical illustrations indicate the superiority of this method, compared to the previous approaches used so far.
- The thesis provides a comparative analysis of DFT, DCT, and DWT in connection with PDEs to recover the full vertex density of the original mesh. Results indicate that both DCT and DWT are more robust than DFT for compressing the data mesh.

1.5 Thesis Organisation

The thesis is organised as follows:

1. Chapter 2 presents an overview of related work, with the history of the numerical analysis using different methods of solving the PDEs.

2. Chapter 3 introduces the basic concepts of Partial Differential Equations and their solution. Direct methods and iterative methods are formulated, and their feasibility is considered.

3. Chapter 4 presents the data modelling and the pre-processing to be used in all experiments in the thesis. This is the first step of the compression method.

4. Chapter 5 presents a polynomial interpolation method for efficient 3D data compression through parameterization of free-form surface patches.

5. Chapter 6 introduces Partial Differential Equations for 3D data compression and reconstruction. The focus of this cFhapter is on data interpolation using the Fourier Transform.

6. Chapter 7 describes a comparative analysis of data compression via the Fourier Transform, Discrete Cosine Transform, Discrete Wavelet Transform and Partial Differential Equations.

7. Finally, Chapter 8 discusses the conclusions of the study, and gives some recommendations for possible future work.

Chapter 2

The Related Work

2.1 Introduction

In this chapter, an overview is provided of research work with the relevant background related to the work presented in the thesis. It is beyond the scope of this thesis to give a comprehensive overview of all related work. Thus, this chapter will concentrate mainly on research closely related to the work presented later, categorised in groups according to the method used. The first category is 3D representation, the second is compression and the third is PDE-based approaches.

2.2 3D Representation

There have been many different schemes used to represent the shape of 3D objects, and their associated properties. The particular improvement of the techniques used to represent the 3D models started out of necessity in the computer aided the geometric design community. Since then, many of the techniques have been adopted and extended in the more general computer graphics field. The representation of 3D objects can be separated into two primary categories; surface modelling and solid modelling. Thus, surface modelling deals with the problem of representing 2D surfaces embedded in the 3D space. These types of surfaces might or even may not define a volume degree. Solid modelling extends the actual techniques of surface modelling to deal with the representation as well as manipulation of volumes, totally surrounded by surfaces, say, for example a cube, buildings, and the human body.

There are three well-known methods to represent a model:

1. Polygonal meshes: Points in 3D space, known as vertices, are connected through a line segments to form the polygonal mesh. Most of 3D models today are built as textured polygonal models, as they are flexible and computers can render them so rapidly. Furthermore, polygons are planar and can only estimate rounded surfaces that use many polygons [Foley, 1996; King et al., 2000]. A triangular mesh is a mesh in which all the faces are triangles. Any polygonal mesh can be transformed into a triangular mesh by triangulating each polygonal face. Even though polygonal meshes can precisely approximate any objects with planar surfaces, this approximation can be made arbitrarily close to the curved surface being modelled by using small enough polygons.

2. Curve modelling: Surfaces are defined as a curve blending control point. Curve types include splines, non-uniform rational B-splines (NURBS), patches and geometric primitives. These types can be given either within the implicit or parametric form. The implicit form makes it simple to determine if a point is actually on the surface, and if not, which side it is located. However, the implicit form will not lend itself to computing the points on the surface within a simple way, when sketching for instance and even less to local modifications of the shape. Furthermore, it is very difficult to model free-form objects using the implicit form [Akkouche and Galin, 2001; Bloomenthal, 1988; Witkin and Heckbert, 1994].

3. Subdivision surface: As an alternative to B-spline and NURBS, it starts with a 2-manifold polygonal mesh and iteratively applies a refinement, or subdivision, procedure [Chaikin, 1974; Cohen et al., 1980]. In geometric modelling subdivision, the processes were extended to general topology

[Catmull and Clark, 1978]. The algorithms produce a surface, which is a B-spline surface everywhere, except at a limited number of extraordinary points [Doo and Sabin, 1978].

An algorithm with one refinement step and no corner cutting was proposed in which the refinement step is used to isolate the irregularities of the mesh [Loop, 1994]. In addition, a modified Butterfly subdivision scheme, which is smooth on irregular meshes, is presented in [Zorin et al., 1996]. Subdivision schemes lend themselves to the representation of surfaces of arbitrary topology in addition to surfaces represented by bivariate functions [Dyn and Levin, 2002].

This dissertation is focused on polygonal meshes.

2.3 Compression

The compression schemes for geometric data models have recently been the subject of intensive research. Data compressions are crucial with regard to decreasing space for storage or transfer over the network. There are two types associated with compression, the first lossy data compression, which is not guaranteed to get the same output bit for a bit for example, JPEG. Second is the lossless compression, which is guaranteed to get the same output bit for a bit at decompression example PNG, ZIP and TGZ.

Compression methods for 3D polygonal data are focused on representing the connectivity of the vertices in the triangulated mesh. Examples include the Edgebreaker algorithm [Szymczak et al., 2001] and [Szymczak et al., 2002]. Products also exist in the market that claim 95% lossless file reduction such as from 3D Compression Technologies Inc. [3DCT, 2010] for regular geometric shapes. In addition, the generalisation of the Edgebreaker's formula with regard to data compression as well as decompression would be to divide every quad into triangles based on the guideline that triangles made from every quad tend to be surrounded within Edgebreaker's traversal series (a triangle spanning tree). It leads to an encoding of 30-80% which is smaller than an approach based on randomly splitting quads into triangles [King et al., 2000; Rossignac, 2001],

Other techniques for triangulated models include the work of [Shikhare et al., 2002] and vector quantization based methods [Qian et al., 1998] where rates of over 98.75% have been achieved. However, a significant drawback to this technique in the use of vector quantization, which adds to computation and throws valuable information away. A new compression algorithm that encodes the connectivity of surface meshes directly into their polygonal representation, by improving the triangulated mesh prior to data compression, is able to recover the polygons by marking the edges along with 70% compression rate [Isenburg and Snoeyink, 2000]. Some other local compression and decompression algorithms, which are sufficiently fast for real time applications, accomplished compression rates of more than 60% [Gumhold and Straßer, 1998]. Regarding geometry encoding, recently reported data compression methods for the vertex coordinates (geometry) have used vertex quantization, and geometric predictors, as well as adjustable duration encodings associated with corrective vectors in order to shrink the actual vertex coordinates [Deering, 1995; Kronrod and Gotsman, 2000; Li and Kuo, 1998; Taubin and Rossignac, 1998; Touma and Gotsman, 1998].

The current state of the art in 3D compression is reasonably well developed concerning connectivity representation, but it is in need of improvement concerning geometric coding [Dodgson et al., 2006; Peng and Kuo, 2005]. The Java3D API and MPEG-4 standards, address issues of compression. Because Java3D is a collection of high-level constructs to create and manipulate graphics objects on top of OpenGL or DirectX, it depends on how geometries are defined in underlying environments. Geometric compression in Java3D is possible using a binary format based on the topological surgery algorithm [Taubin and Rossignac, 1998], normally to one order of magnitude [Davidson and Hanson, 2004]. The MPEG-4 multimedia standard also includes 3D mesh coding. MPEG-4 Part 20 contains specifications for scene representation, manipulation and encoding in binary compression format [Smolic et al., 2006] also based on the topological surgery algorithm. While such initiatives provide a reference for research in 3D data compression, current compression rates are still too low for general sharing of 3D geometry files over the internet. The GMPR research group has developed and demonstrated original methods and algorithms for fast 3D scanning for a number of applications with a particular focus on security [Brink et al., 2008; Robinson et al., 2004; Rodrigues and Robinson, 2010, 2011; Rodrigues et al., 2008]. The algorithms can perform 3D reconstruction in 40 milliseconds and recognition in near real-time, but saving such 3D facial models has resulted in a severe bottleneck due to the size of the data files. All data used in this research have been previously acquired using the GMPR scanner.

2.4 PDE-based Approaches

Recently, several approaches for solving PDE-based modelling have been developed. In particular, various methods were discussed in [Bloor and Wilson, 1997, 1989; Jain and Jain, 1978; Malcolm Bloor and Wilson, 1996; Mathews and Fink, 1994]. However, surface modelling techniques tend to be fundamental for many visual processing applications including interactive graphics, CAD/CAM, animation, and digital environments. Frequently-used representation schemes for free-form surface modelling such as spline-based approaches take advantage of simple polynomial functions in collaboration with control points [Böhm et al., 1984; de Boor, 2001; Farin, 1996; Forsey and Bartels, 1988; Piegl, 1991; Piegl and Tiller, 1987; Ugail et al., 1999]. Nevertheless, an over-all way of establishing distinction strategies in order to determine the numerically particular quasi-linear PDE through Levenberg-Marquardt kind algorithms with regard to elliptic as well as parabolic problems may be referred to [Wiegmann and Bube, 1998]. Consequently, the problem of regularisation of the Cauchy problem for Laplace's equation is considered to be close to the exact solution [Ang et al., 1998].

The design and data framework software for solving PDEs is reported in the literature where sequences of finite-element problems could be constructed in the self-adaptive or even quasi-interactive mode. The software includes linear, triangular, finite element areas, a posteriori error estimate, adaptivity of the mesh, conforming mesh-refinement algorithms for triangulations, along with a full multigrid method for resolving linear systems [Bartels et al., 2006; Grebennikov, 2005; Rivara, 1984; Van Schijndel, 2003]. This research favours the method of lines (MOL) which is a convenient method for the numerical integration of PDEs; for example, the Korteweg-de Vries equations have been formulated to model shallow water flow [Saucez et al., 1998; Schiesser, 1994] and are solved by the method of lines.

Point datasets routinely generated via optical and photometric variety finders are usually corrupted through the noise. In order to remove these kind of deficiencies from scanned stage, clouds, a large variety of denoising approaches based on low-pass filtering are used [Linsen, 2001]. Typically, the moving least squares (MLS) surface, used for modelling and also rendering with point clouds fitting [Adamson and Alexa, 2003; Alexa et al., 2003; Amenta and Kil, 2004; Bremer and Hart, 2005; Dey and Sun, 2005] and partial differential equations (PDEs) [Lange and Polthier, 2005; Shu et al., 2003] has been proposed.

The Trefftz method along with the method of particular solutions provides an attractive mesh-free alternative for solving non-linear Poisson equations in two and three dimensions [Balakrishnan and Ramachandran, 1999]. Moreover, for finding the approximate solution of a second order, non-linear PDE by transforming the problem into an optimisation problem and considering it as a distributed parameter control system [Gachpazan et al., 2000; Mai-Duy and Tran-Cong, 2001; Sharan et al., 1997]. Furthermore, the new multi resolution scheme has been proposed based on an image transform by a discretized elliptic partial differential operator and use of a multi grid operator, leading to a pyramidal representation [de Zeeuw, 2005].

Applying PDE-based methodology for image sequences, restoration and motion segmentation by a convergent stable algorithm and approximate a unique solution of the initial minimisation is described in [Kornprobst et al., 1999]. The actual factorisation associated with fourth order PDEs into a set of two nested order problems to generate free surfaces that fulfil artistic requirements that close triangle mesh is described in [Golbabai and Javidi, 2007; Qian et al., 2006; Schneider and Kobbelt, 2001; You et al., 2008]. Solving a fourth order PDEs with three vector valued shape parameters to generate complex free form surfaces has been described in [Zhang and You, 2002]. It has been shown that solving a fourth order PDE with boundary conditions divided into a closed and non-closed form solutions lead to a mixed PDE solution that can be applied to a number of surface modelling types [Du and Qin, 2005; Duan et al., 2004; Zhang and You, 2004*b*]. On the other hand, second order PDEs can be improved by introducing fourth order PDEs for one of the components leading to mixed order PDEs, which have many more degrees of freedom, and hence are able to generate a family of surfaces with sophisticated geometric features [Zhang and You, 2001].

An additional approach for optimisation is based on a PDE formulation enabling efficient shape definition and shape parameterization. It has been showed how the choice of an elliptic PDE enables surfaces to be created that correspond to complex shapes [Ugail, 2003; Ugail and Wilson, 2003]. In particular, an accurate numerical solution of nonlinear PDEs can be obtained by using high order approximation in space and time by solving the fourth order Runge-Kutta method [Kassam and Trefethen, 2005]. The closed form solution associated with PDE has often been either non-existent or not obtainable, depending on the boundary conditions and the coefficients of the PDE; only a small proportion of them result in a closed form solution [Zhang and You, 2004*a*], whereas solving the C^2 continuous surface blending by a sixth-order PDE satisfies the boundary conditions and minimises the overall PDE errors [You et al., 2004]. However, it is possible to solve higher order PDEs and accommodate general boundary conditions in, say, a sixth-order PDE solution. Evaluating higher order PDEs provides a very good capability to make a broader selection of areas whilst sustaining the actual flexibility from the PDE technique through concentrating on areas that are regular, to ensure that topologically they're just like a closed band [Kubiesa et al., 2004]. In addition, solving PDEs with high order boundary continuity conditions produces very fair and desirable solution surfaces [Xu et al., 2006].

The actual formulation associated with 3D surface reconstruction utilizing spectral active surfaces with edge fines could be put in place within spherical geometry. The spectral method uses the dual Fourier sequence being an orthogonal base to resolve the series associated with elliptic PDEs within the unit sphere [Li and Hero, 2004]. Discrete surface patches obtained by solving various geometric PDEs to model geometric shapes can be used to choose suitable PDEs for each problem shape [Qing, 2005]. Accurate modelling results are obtained by solving Laplace's equation for anisotropic 3D magnetic resonance imaging (MRI). A fast and accurate algorithm for generating the thickness map from the potential function is shown to yield better results compared to other methods [Haidar et al., 2005]. Mikhlin's method for solving Laplace's formula in increase linked exterior websites with Dirichlet boundary data obtained highly accurate alternatives in exterior domains [Helsing and Wadbro, 2005].

The reconstruction of the 3D geometry of human faces based on the use of elliptic PDEs using a set of boundary conditions to generate surface patches from the original scanned data is described in [Elyan and Ugail, 2007]. Therefore, the fourth order PDE method is inherently capable of generating smooth facial animations with a complicated face design, by modifying only a relatively small number of boundary curves. The solution of nine various PDEs along with twenty-eight boundary curves was required to generate an entire face model. The continuity within the model is actually assured through prescribing at least one typical boundary condition for surrounding patches [Sheng et al., 2008; Ugail and Sourin, 2008].

In addition, a new technique for quantifying the uncertainty associated with the solution of a PDE involving stochastic parameters is described in [Mathelin and Gallivan, 2010]. The application of the PDE means of designing a parametric representation, and the parameterization and reconstruction of 3D face images have been achieved in [Ahmat et al., 2011; Wang et al., 2012]; their studies show that the simulation may be used to represent the powder compaction process and predict the actual elasticity and plasticity associated with pharmaceutical materials.

Furthermore, a solution to PDE models in 3D provides an ideal platform on which researchers from various fields can communicate with each other. With regard to most cancers modelling, particularly, 3 as well as 4 dimensional visualisation can be handy with regard to doctors in order to localise the actual believed tumour placement inside the site with regard to surgical treatment as well as preparing the remedy. [Enderling et al., 2006].

2.5 Discussion

This Chapter has reviewed various popular schemes for the representation of a complex shape. The most typical techniques tend to be polygonal works, parametric areas as well as subdivision methods, which appear to be better solutions for free form surfaces. The other reviewed techniques Spline and B-spline (NURBS) can only describe a limited set of shapes or are not adequate for modelling purposes. While simple and flexible, polygonal meshes are not capable of accurately representing smooth surfaces. The early compression methods were mainly focused on speeding up the transfer of model data from the CPU to the graphics board, for rendering purposes, across a bus of limited bandwidth. Such methods have to be of low complexity so as to be easily executed by the hardware on the graphics board and therefore they only obtain modest compression ratios.

With regard to compression, most of the recent techniques for "lossless" progressive coding associated with carefully designed meshes use the independent set concept to drive the mesh refinement operations to be organised into a set of patches or along a chain of edges optimised for efficient rendering. Vertex positions are coded using various prediction schemes. Moreover, less work has been done concerning geometry coding than for connectivity coding, since they are lossy and it is difficult to analyse their performance. It is noted from the literature review above that Laplace's equation has not been used for surface reconstruction in connection with PDEs as proposed in this research and demonstrated later on in Chapters 5–7. In this research, each PDE patch is calculated independently using the boundaries defined by the cutting planes and given that the patches are adjacent to one another, they use the same boundaries, so the issue of smoothing between the boundaries will not occur. Laplace's equation has been used in a number of mesh post-processing methods, notably in hole filling, with similar results (that is, no smoothing issues between mesh boundary and inserted vertices) [Rodrigues and Robinson, 2010].

The results presented thus far in the literature are quality deficient for the intended application of 3D data compression, so alternative ways of defining and solving PDEs over surface patches need to be investigated. In the next Chapter, the numerical solution of PDEs is presented and the background is provided on the method that will be used later in this thesis.

Chapter 3

Partial Differential Equations and their Solutions

This chapter features some numerical concepts, that is to be needed during the entire thesis. The partial differential equation (PDE) discretization methods considered here are the method of lines for solving Laplace's equation.

Definition 3.0.1. Any equation involving an unknown function along with some or all of its derivatives is called a differential equation (DE) [Hale and Lunel, 1993; Zill, 2012; Zwillinger, 1998].

Differential equations break down into two major kinds: ordinary differential equations (ODEs) and partial differential equations (PDEs).

3.1 Introduction to Partial Differential Equations

Definition 3.1.1. Partial differential equations (PDEs) are equations containing an unknown function of two or more variables and its partial derivatives with respects to these variables [Evans, 2010; Hadamard, 2003; Jeffrey, 2003; Renardy and Rogers, 2004].

Partial differential equations (PDEs) provide a quantitative description for many primary models in physical, biological, and the social sciences. Typically the description is furnished in terms of unknown functions of two or more independent variables, and the relation between partial derivatives with respect to those variables. A PDE is said to be nonlinear if the relations between the unknown functions and their partial derivatives involved in the equation are nonlinear. Regardless of the apparent simplicity of the fundamental differential relations, nonlinear PDEs governs a vast array of complex phenomena of motion, response, diffusion, equilibrium, conservation, and more. Because of their pivotal role in technology and engineering, PDEs tends to be studied extensively by experts and practitioners. Indeed, these studies have found their method into many entries throughout scientific literature. They reflect a rich development of mathematical theories and analytical techniques to solve PDEs and illuminate the phenomena they govern. Nonetheless analytical theories provides simply a limited account for the selection of complex phenomena governed by simply non-linear PDEs [Babuska, 1995; Griffiths and Schiesser, 2010; Hamdi et al., 2007; Ritger and Rose, 1968; Schiesser, 1991].

The general linear partial differential equations (PDEs) of order two in two independent variables has, the form [Bhamra, 2010; Farlow, 2012; Pinsky, 2011; Sapiro, 2006; Trèves, 1975]

$$A(x,y)U_{xx}+B(x,y)U_{xy}+C(x,y)U_{yy}+D(x,y)U_x+E(x,y)U_y+F(x,y)U=G(x,y) \tag{3.1}$$

where $A,B,C,D,E,F,G,$ may depend on x and y but not on U. U_x is the first partial derivative of U with respect to x, $\partial U/\partial x$, and U_y is the first partial derivative of U with respect to y, $\partial U/\partial y$, U_{xx} is the second partial derivative of U with respect to x, $\partial^2 U/\partial x^2$, and U_{yy} is the second partial derivative of U with respect to y $\partial^2 U/\partial y^2$, and U_{xy} is the second partial derivative of U with respect to y then with respect to x, $\partial^2 U/\partial y\partial x$, and U_{yx} is the second partial derivative of U with respect to x then with respect to y, $\partial^2 U/\partial x\partial y$. A second order equation with independent variables x and y which does not have the form 3.1 is called nonlinear. A linear PDEs is called homogeneous if $G(x,y) = 0$, while if $G(x,y) \neq 0$ it is called non-homogeneous. Equation 3.1 is often classified as:

- if $B^2 - 4AC < 0$ the equation is elliptic (Laplace's equation)
- if $B^2 - 4AC > 0$ the equation is hyperbolic (wave equation)
- if $B^2 - 4AC = 0$ the equation is parabolic (heat or diffusion equation).

This thesis focuses on Laplace's equation, which is a classical Elliptic PDE. There are several ways to solve Laplace's equation, in the experiments of this thesis the focus on two methods, first using a separation of variables which involves the fast Fourier Transform, and second solved by the method of lines on a grid. The method of lines is regarded to be a unique finite difference method, however, is more effective with respect to accuracy as well as computational time than the normal finite difference method. Furthermore, the method of lines is not just a single, straightforward, clearly defined approach to PDE problems, but alternatively, is a general concept that could need a specification of information for each new PDE issue [Schiesser, 1994]. The technique associated with the method of lines has got the subsequent qualities:

- Replace the spatial derivatives in the PDE with algebraic approximations.
- Needs approximately ten times less storage than conventional finite difference methods.
- Mathematical stability: by splitting the difference, it is easy to set up stability and convergence for a variety of problems.
- Decreased programming effort: by a approximating system of ODEs.
- Decreased computational time: since only some discretisation lines are necessary in the calculations, there is no need to fix a large system of equations.

Therefore, the method of lines is a technique where we discretised all the independent variables except one. This leads to a large set of coupled ODEs, this system of ODEs are solved analytically. Any method can be used to discretised the independent variables. This includes Fourier Transform or the finite difference method. The technique being used in this thesis was to replace all the partial derivatives with the central finite difference approximation that gives a system of ODEs. Although this formulation may differ from other approaches, it is clearly advocated by [Liu et al., 2004; Lord et al., 2014; Trefethen, 2000] as an alternative approach, as the fundamental principles are the same. The method of lines which is used in the thesis involves solving the elliptic PDEs over a rectangular domain. The domain is defined by mesh cutting planes yielding vertices on a regular grid that define the top and bottom boundaries of the domain. All vertices on the top boundary can be paired to their corresponding vertices on the opposite bottom boundary. The left and right boundaries are defined by interpolating between the first top and first bottom vertices and last top and last bottom vertices using the finite difference method. The number of interpolated vertices is user defined. All interior vertices to the rectangular domain are initialised to zero and are interpolated by iteratively solving Laplace's equation over the domain. Therefore, we approximate Laplace's equation at each grid point, and the resulting equations are solved by iteration through implementing the Matlab function `'gmprLaplace.m'`. Further description is given in Section 6.2.3.

3.2 Boundary Value Problem

Boundary conditions need to be carefully defined to create a design that performs efficiently and is a good approximation of the phenomenon being modelled. There are three significant kinds of boundary value problems that occur in most applications:

1. Dirichlet boundary condition: "The solution has some value at the endpoint or along the boundary." [Duffy, 2008]

2. Neumann boundary condition: "The derivative of the solution equals a particular value at the endpoint or in the normal direction along a boundary."

[Duffy, 2008]

3. Mixed boundary condition (Robin): “A mixture of the values of the function and its normal derivative is specified on the boundary of the bounded domain.” [Duffy, 2008; Koch and Segev, 1998]

3.3 Classic Fourier Series

Fourier sine and cosine series are consistently known as half range series since only half of a symmetrical period is applied in the integrals interpreting the coefficients. To obtain these series one symbolizes that the function f is an even or an odd function [Edwards, 1979; Grafakos, 2004; Tolstov, 2012; Walker, 1996; Young, 2001].

One can observe that if f is even, then $f(x)\cos(n\pi x/L)$ are also even. The coefficient a_n has an even integrand on $(-L,L)$. We write twice the integral over half the interval and obtain

$$a_n = \frac{2}{L}\int_0^L f(x)\cos\left(\frac{n\pi x}{L}\right)dx \tag{3.2}$$

Since $f(x)\sin(n\pi x/L)$ is odd and b_n has an odd integrand over a symmetric interval, we have

$$b_n = 0$$

With $f(x)$ even, to obtain

$$f(x) \approx \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right), \tag{3.3}$$

where

$$a_n = \frac{2}{L}\int_0^L f(x)\cos\left(\frac{n\pi x}{L}\right)dx \tag{3.4}$$

The interval in this case is $(0,L)$, but the even periodic extension of $f(x)$ presumes a period of $2L$. This series is known as the Fourier cosine series or the half range Fourier cosine series.

If $f(x)$ is an odd function, then $f(x)\sin(n\pi x/L)$ is an even function. Just in case such as this

$$b_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi x}{L}\right)dx \tag{3.5}$$

The product $f(x)\cos(n\pi x/L)$ are odd, and

$$a_n = 0$$

As a result, we may write

$$f(x) \approx \frac{b_0}{2} + \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi x}{L}\right), \tag{3.6}$$

where

$$b_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi x}{L}\right)dx \tag{3.7}$$

Again the interval is $(0,L)$ and a period of $2L$ is assumed when the odd periodic extension of $f(x)$ is considered. This is a Fourier sine series.

Definition 3.3.1. A Fourier series is an infinite series of the form

$$\phi(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}\Big(a_n\cos(\frac{n\pi x}{L}) + b_n\sin(\frac{n\pi x}{L})\Big). \tag{3.8}$$

Assuming the series converges, the function defined by the series is periodic on the interval $[-L,L]$ but it may not be continuous. The coefficients $\{a_n\}_n, \{b_n\}_n$ are generally known as the Fourier coefficients of the function ϕ [Brown and Churchill, 2012*a*; Edwards, 1979; Tolstov, 2012].

Joseph Fourier (1768-1830) applied this particular concept of writing a function as a sum of trigonometric functions within his research from the numerical concept associated with heat conduction [Grattan-Guinness and Ravetz, 2003].

Definition 3.3.2. A function $f(x)$ is said to be periodic with a period L if $f(x+L) = f(x)$ for all x in the domain of f [Brown and Churchill, 2012*a*; Harding, 1985].

3.4 Dirichlet Boundary for Laplace's Equation

In this section solutions to Laplace's equation with Dirichlet boundary problems are discussed. The first solution is through the method of separation of variables which involves the fast Fourier Transform, and the second solution involves the method of lines on a grid.

3.4.1 The solution by separation of variables

To solve the Dirichlet boundary value problem of Laplace's equation in a rectangular domain by separation of variables (see Figure 3.1) [Babuska, 1995; Gakhov, 1990; Haberman, 1983; Ritger and Rose, 1968; Wazwaz, 2002]:

$$U_{xx} + U_{yy} = 0 \tag{3.9}$$

where $u(x,y)$ satisfies the homogeneous boundary conditions, and

$$\begin{aligned} &u(0,y) = u(x,b) = u(x,0) = 0 \\ &u(a,y) = f(y) \end{aligned} \tag{3.10}$$

where $u(x,y)$ satisfies the non-homogeneous boundary condition, and f is a given function.

By using the technique of separation of variables (a solution can be expressed as a product of unknown functions each of which depends only on one of the independent variables), assume the solution to Laplace's equation is separable form, $u(x,y) = X(x)Y(y)$. To compute the partial derivatives that we require within the equation, we note that

$$u_y(x,y) = \frac{\partial}{\partial y}\big(X(x)Y(y)\big) = X(x)Y'(y) \tag{3.11}$$

$$u_x(x,y) = \frac{\partial}{\partial x}\big(X(x)Y(y)\big) = X'(x)Y(y) \tag{3.12}$$

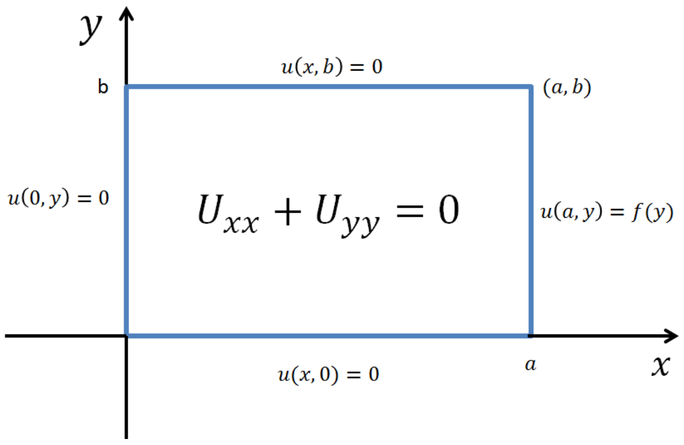

Figure 3.1: Defining Laplace's equation over a rectangular domain.

thus,

$$u_{yy}(x,y) = \frac{\partial^2}{\partial y^2}\big(X(x)Y(y)\big) = X(x)Y''(y) \tag{3.13}$$

$$u_{xx}(x,y) = \frac{\partial^2}{\partial x^2}\big(X(x)Y(y)\big) = X''(x)Y(y) \tag{3.14}$$

Then Laplace's equation 3.9 can be written as:

$$X''(x)Y(y) + X(x)Y''(y) = 0 \tag{3.15}$$

That can be rearranged to form

$$\frac{X''(x)}{X(x)} = -\frac{Y''(y)}{Y(y)} = \lambda, \tag{3.16}$$

where λ is a separation constant. The left hand side depends only on x, while the right hand side depends only on y. Thus Eqs.3.16 is partitioned into two ODEs as

$$X''(x) = \lambda X(x), \tag{3.17}$$

and

$$Y''(y) = -\lambda Y(y). \tag{3.18}$$

Let $Y(y)$ satisfy the Dirichlet boundary condition

$$Y(0) = Y(b) = 0. \tag{3.19}$$

Eqs. 3.17 and 3.18 are ODE and can be solved with basic techniques. There are three different cases, depending on the sign of λ, each will give four different solutions to Laplace's equation. Then, solving for Y in Eq. 3.18 with the boundary condition in Eq.3.19, the nontrivial solution is

$$Y = c\sin\big(\frac{n\pi y}{b}\big) \qquad \text{with} \qquad \lambda = (\frac{n\pi}{b})^2, \tag{3.20}$$

where $n = 1, 2, \ldots$. For the λ in Eq.3.20, it is found that the general solution to Eq. 3.17 is

$$X(x) = Ae^{\frac{n\pi}{b}x} + Be^{-\frac{n\pi}{b}x}. \tag{3.21}$$

or

$$X(x) = c_1\cosh\big[\frac{n\pi}{b}(x-L)\big] + c_2\sinh\big[\frac{n\pi}{b}(x-L)\big]. \tag{3.22}$$

The shift in x by L is selected to satisfy the boundary condition at $x = L$. It is assumed that $X(L) = 0$, which implies $c_1 = 0$. Thus

$$X(x) = c_2\sinh\big[\frac{n\pi}{b}(x-L)\big]. \tag{3.23}$$

Thus, it is found that the nontrivial product solutions to Laplace's equation together with the homogeneous boundary conditions are constant multiples of

$$u(x,y) = c_2\sinh\big[\frac{n\pi}{b}(x-L)\big]\sin(n\pi y/b). \tag{3.24}$$

By the superposition principle theorem, we obtain

$$u(x,y) = \sum_{n=1}^{\infty} c_n\sinh\big[\frac{n\pi}{b}(x-L)\big]\sin(n\pi y/b).. \tag{3.25}$$

The coefficients c_n are identified by the boundary condition

$$u(a,y) = \sum_{n=1}^{\infty} c_n \sin(n\pi a/b)\sinh(n\pi y/b) = f(y). \tag{3.26}$$

Therefore the quantities $c_n \sinh(n\pi a/b)$ must be the coefficients in the Fourier sine series of period $2b$ for f and are given by

$$A_n = \frac{2}{b}\int_0^b f(y)\sin\frac{n\pi y}{b}\mathrm{d}y. \tag{3.27}$$

Thus, it can be written:

$$c_n = \frac{A_n}{\sinh\left[-\frac{n\pi L}{b}\right]} \tag{3.28}$$

Thus the solution to the partial differential Equation 3.9 satisfying the boundary condition 3.10 as given by Eq 3.25 with the coefficients c_n computed from Eq. 3.27. From Eqs.3.25 and 3.27 it can be seen that the solution contains the factor $\sinh(n\pi x/b)/\sinh(n\pi a/b)$.

To estimate this quantity for large n one can use the approximation $\sinh\xi \cong e^{\xi}/2$, and thereby obtain

$$\frac{\sinh(n\pi x/b)}{\sinh(n\pi a/b)} \cong \frac{\frac{1}{2}exp(n\pi x/b)}{\frac{1}{2}exp(n\pi a/b)} = exp[-n\pi(a-x)/b]. \tag{3.29}$$

Thus, this factor has the character of a negative exponential; consequently, the series 3.25 converges quite rapidly unless $a-x$ is very small.

3.4.2 The solution by the method of lines

In the previous section, the solution to Laplace's equation by the method of separation of variables was discussed. However, making use of this technique could be formally complicated given it will involve the particular calculation of the Fourier coefficients. Furthermore, this Fourier series may only converge gradually on the boundary.

Therefore, another alternative method is to solve Laplace's equation on a grid by the method of lines [Lord et al., 2014; Strang and Aarikka, 1986],

$$U_{xx} + U_{yy} = 0 \tag{3.30}$$

In Figure 3.2 divided the interval into N and M sub-intervals with $\Delta x = \frac{a}{(N-1)}$, and $\Delta y = \frac{b}{(M-1)}$ such that $(x_i, y_j) = (i\Delta x, j\Delta y)$ where $i = 0, 1, \ldots, N-1$, and $j = 0, 1, \ldots, M-1$

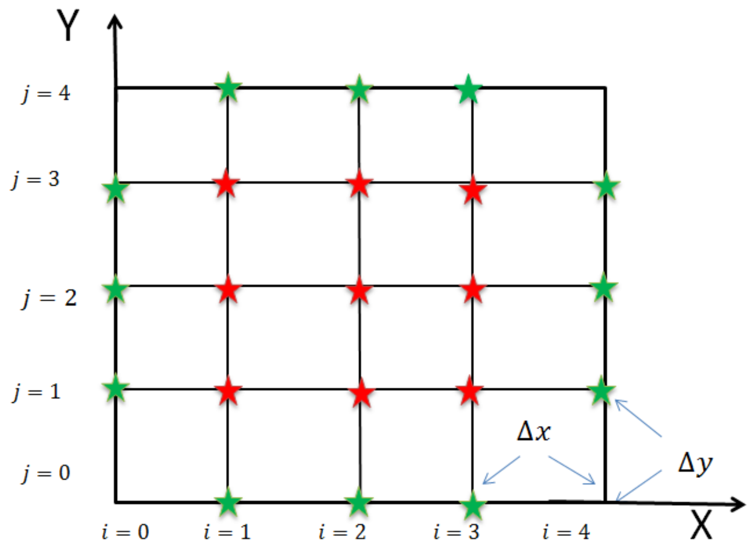

Figure 3.2: The value in green is given by the boundary condition, the only unknowns are $U_{i,j}$ marked in red.

The domain has four boundaries;

$$\begin{aligned} U_{i,0} &\equiv g(x_i, 0), \qquad U_{i,M-1} \equiv g(x_i, b) \qquad i = 0, 1, \ldots, N-1 \\ U_{0,j} &\equiv g(0, y_j), \qquad U_{N-1,j} \equiv g(a, y_j) \qquad j = 0, 1, \ldots, M-1. \end{aligned} \tag{3.31}$$

The boundary conditions are simplified along the boundary (green), and the interior points (red) are unknowns.

Finite difference approximations must now be used to replace U_{xx} and U_{yy} in the

Laplace's equation. Focusing on an interior point (x_i, y_j), the simplest approximations to replace the second derivatives with it is a central finite difference approximation as follow,

$$U_{xx}(x_i, y_j) \approx \frac{U(x_{i-1}, y_j) - 2U(x_i, y_j) + U(x_{i+1}, y_j)}{(\Delta x)^2}, \tag{3.32}$$

and

$$U_{yy}(x_i, y_j) \approx \frac{U(x_i, y_{j-1}) - 2U(x_i, y_j) + U(x_i, y_{j+1})}{(\Delta y)^2} \tag{3.33}$$

Eqs 3.32 and 3.33 tend to be just like individuals for that regular second derivatives, d^2u/dx^2 and d^2u/dy^2, only that in Eqs. 3.32 y is held constant (all terms in Eqs. 3.32 have the same j) and in Eqs. 3.33 x is held constant (all terms have the same i). Eqs. 3.32 and 3.33 are equivalent to

$$U_{xx}(x,y) \approx \frac{U(x-\Delta x, y) - 2U(x,y) + U(x+\Delta x, y)}{(\Delta x)^2}, \tag{3.34}$$

$$U_{yy}(x,y) \approx \frac{U(x, y-\Delta y) - 2U(x,y) + U(x, y+\Delta y)}{(\Delta y)^2} \tag{3.35}$$

Connecting Eqs. 3.32 and 3.33 into the unique Laplace equation and by the used of the method of lines approximation, to obtain a system of ODEs

$$\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{(\Delta x)^2} + \frac{U_{i,j-1} - 2_{i,j} + U_{i,j+1}}{(\Delta y)^2} = 0 \qquad \text{at the grid point}(i, j). \tag{3.36}$$

or

$$U_{i-1,j} + U_{i+1,j} - 4U_{i,j} + U_{i,j-1} + U_{i,j+1} = 0 \tag{3.37}$$

Assuming that $\Delta x = \Delta y$, where $U_{i,j} = U(i\Delta x, j\Delta y)$. That gives

$$U_{i,j} = \frac{U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}}{4}. \tag{3.38}$$

Thus, $U_{i,j}$ should be the average of its nearest neighbours. When the average is higher than some numbers in the neighbouring, those neighbours below the average have the potential to reach higher values after some iterations. Therefore, when U reaches a highest on a few internal levels, then a similar highest can also be achieved by each neighbour within this level. We can continue iterating this technique right up until we cover all the points in the rectangle, and we get the vector $\vec{U}$.

$$\vec{U} = \begin{pmatrix} U_{11} \\ U_{12} \\ U_{13} \\ U_{21} \\ \vdots \\ \vdots \\ U_{MN} \end{pmatrix}, \tag{3.39}$$

Moreover, Eq. 3.38 is an approximation of Laplace's equation, and it is an $M \times N$ matrix that can be solved as Jacobi iteration

$$U_{i,j}^{(N+1)} = \frac{1}{4}\left[U_{i-1,j}^{(N)} + U_{i+1,j}^{(N)} + U_{i,j-1}^{(N)} + U_{i,j+1}^{(N)}\right] \tag{3.40}$$

where the superscript denote the iteration number and $U_{i,j}$ is the solution at the i, j grid point,

$$\lim_{N\to\infty} U_{i,j}^{(N+1)} = U_{i,j}. \tag{3.41}$$

Therefore, $U_{i,j}$ is an excellent approximation to the exact solution to Laplace's equation.

3.5 Signal Representation

Lately considerable effort has been dedicated to finding sparse representations with regard to target signals aiming in enhancing processing speeds upon large-scale data. Sparse representation indicates that a signal can be decomposed into a direct linear combination of a few main signals. In this thesis, we are focusing on one-dimensional signals that can be represented by the following:

1. The Discrete Fourier Transform (DFT).
2. The Discrete Cosine Transform (DCT).
3. The Discrete Wavelet Transform (DWT).
4. The PDE-based Approach.

3.5.1 The Discrete Fourier Transform (DFT)

The discrete Fourier transform is a numerical approximation to the Fourier transform, which is very useful in data compression, because a few coefficients of the Fourier expansion may be sufficient for the reconstructed signal to be close enough to the original function. It has already been found that the use of the FFT techniques has considerably enhanced the strength of digital techniques for a very wide range of problems such as spectral research, sign managing, graphic controls, and also the solution of differential equations [Cooley et al., 1969; Hanna and Rowland, 2008]. The fast Fourier transform is an efficient algorithm for computing the discrete Fourier transform DFT and its inverse, which takes a regularly spaced data value, then returns the value of the Fourier transform for a set of values in frequency space. Moreover, the FFT algorithm can decrease the processing time of a standard Discrete Fourier Transform from several minutes to a few milliseconds, and it is global problem solving technique. The significance of Fourier Transform comes from allowing the evaluation of particular relationships in a problem domain from an entirely different viewpoint. Studying the behaviour of a function and its Fourier Transform is often the key to efficient problem solving [Weinberger, 2012].

The Fourier Transform allows approaching PDE's by modifying them into a simpler differential equation. Once this is done, the facts about the transform must then be used, in order to find its inverse. The continuous Fourier Transform is defined as

$$f(v) = \mathcal{F}_t[f(t)](v) \tag{3.42}$$

$$= \int_{\infty}^{-\infty} f(t) e^{-i2\pi v t} \mathrm{d}t. \tag{3.43}$$

If the integral exists for each value of the parameter f then Eqs 3.43 defines $f(v)$, the Fourier Transform of $f(t_k)$.

Now consider the generalisation to the situation of a discrete function, $f(t) \to f(t_k)$ by letting $f_k \equiv f(t_k)$, where $t_k \equiv k\Delta$, with $k = 0, 1, ...N-1$. Writing this out gives the Discrete Fourier Transform $F_n = \mathcal{F}_k\left[\{f_k\}_{k=0}^{N-1}\right](n)$ as

$$F_n = \sum_{k=0}^{N-1} f_k e^{-i2\pi nk/N}. \tag{3.44}$$

The inverse transform $f_k = \mathcal{F}_n^{-1}\left[\{F_n\}_{n=0}^{N-1}\right](k)$ is then

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{i2\pi kn/N}. \tag{3.45}$$

Discrete Fourier Transforms are useful because they reveal periodicities in data views as well as the relative importance of regularity elements. There are a few details on the interpretation of Discrete Fourier Transforms, however. Typically, the Fourier Transform of an actual series will be a series of an actual and complex variant of the same duration. Particularly, if f_k are real, then F_{N-n} and F_n are approximated by:

$$F_{N-n} = \bar{F}_n, \tag{3.46}$$

for $n = 0, 1, ..., N-1$, where $\bar{z}$ signifies the actual complex conjugate. Exactly what this particular means is that the factor F_0 is always real for real details. Due to the above, a frequency function will contain peaks within not one, but two locations. This happens because the periods become separated into “positive” and “negative” frequency complex components.

3.5.2 The Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) are important to numerous types of lossy compression of audio and image, to solve PDE by spectral techniques where the different version of the DCT matches to slightly different even/odd border circumstances at the two ends of the range. The use of cosine rather than sine features is crucial in these applications: for compression, it can be seen that cosine functions are much more effective.

In particular, the DCT is equivalent to a DFT, but with only real values: the DCT is comparable to a DFT of approximately twice the length since the FFT of a real and even function is real and even, where in some versions the output is shifted by half a sample.

The most common DCT definition applied to 2D image compression is the following [Halpern et al., 2002]:

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{3.47}$$

for $u,v = 0,1,2,\ldots,N-1$. The inverse transform is defined as

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{3.48}$$

for $u,v = 0,1,2,\ldots,N-1$.

3.5.3 The Discrete Wavelet Transform (DWT)

The term ‘wavelet’ is used to describe a spatially localized function. ‘Localized’ means that the wavelet has compact support or it almost has compact support in the sense that outside some interval the amplitude of the wavelet decays exponentially [Jameson, 1993]. Just like the Fourier sequence, wavelets are statistical features that are used to signify information or other features, by analysing the data according to scale. This function has developed mostly over the last 15 years and has generated tremendous interest in many areas of research in mathematics, physics, computer science, as well as architectural. However, most applications of wavelets have focused on analysing data and using wavelets as a tool for data compression.

Wavelet methods combine the advantages of both spectral (Fourier) and finite difference methods and allow both space and time dependent coefficients [Beylkin, 1993; Dahmen et al., 1999; Schneider and Vasilyev, 2009; Vasilyev and Kevlahan, 2005; Vasilyev et al., 1997; Xu and Shann, 1992]. Wavelets allow decomposition of a signal or an image into its components with respect to a whole cascade of levels. This decomposition is done by the fast wavelet transform (FWT) which is of linear complexity as long as the wavelet is compactly supported [Meyer, 1990]. Decomposition and reconstruction allow a signal or an image to be transformed from one representation to a different one; namely, from a single scale to a multi-scale representation. However, successive application of these two operations, gives back the original signal or image as long as the corresponding filters are chosen appropriately. The reason why wavelets are so successful in signal and image processing lies in the fact that the multi-scale representation allows the modification of the signal or image for different purposes.

Firstly, it has been found that reasonable signals or images have a sparse multi-scale representation in the sense that many coefficients in this representation are zero, or at least small. Consequently, it is possible to neglect these small coefficients. This can be the key point of compression. However, just to compress a signal or image is only half of the story. Certainly, one would like to change the original information as little as possible when compressing the data. Since wavelets (no matter whether they are orthogonal or bi-orthogonal) allow the estimation of the error arising in terms of the neglected coefficients, it is quite easy to control the error. The reason for this is that wavelet bases give rise to so-called norm equivalences. Because of this norm of a function (for instance, a signal) is equivalent to the norm of the wavelet coefficients. Finally, such an equivalence not only holds for one single type of norm, but for a whole range [Urban, 2009].

3.5.4 The PDE-based Approach

The PDE-based approach to global sensitivity analysis gives access to a profound theory and broad methodology. Methods of lines are generally simple to implement due to the possibility of using standard ODE solvers. Concerning error control, adaptive ODE solvers straightforwardly allow for temporal adaptivity. However, spatially adaptable methods of lines commonly rely on a posteriori error estimates, that require a complete solution of the system, before the spatial discretization can be adapted [see for instance [Adjerid et al., 1999]]. In that respect, both methods offer a substantial advantage, since the temporal and spatial discretization can be adjusted in each integration step.

The method of lines is a technique that transforms a PDE into a set of ODEs with a single variable. The transformation is done by discretizing the PDE in space, leaving a number of unknowns and their time derivatives. For the space discretization, the techniques referred to previously may be used. For instance, when the finite difference technique can be used, the area discretization results in one unknown and its time derivative at each grid point in the domain, that is a set of ODEs. One advantage of the method of lines is that advanced numerical solution techniques can be found with regard to resolving common ODEs which not necessarily nevertheless are available regarding PDEs. There are, for instance, solvers with automatic step adjustment to find a solution with needed precision. An additional benefit is actually which combined techniques containing both ODE and PDE based models become much easier to solve since the space discretization of the actual PDEs outcomes in ODEs that may be resolved with the already existing ODEs.

Consequently, solve the PDEs by the method of lines [Hamdi et al., 2007; Schiesser, 1991], tend to be of broad interest in science and engineering. The General Ray (Gr) method is applied for the solution of direct boundary value problems, and uses explicit formulas with the fast inversion of the Radon transform. This leads to fast algorithms realised in Matlab [Grebennikov, 2005]. The 2D case has been attempted by works such as those of [Galić et al., 2005; Mainberger and Weickert, 2009; Peloquin, 2009; Stürmer et al., 2008] with promising results in 2D images that can be seen as single-value functions from pixel intensities. However, such methods have not attempted to encode arbitrary 3D geometries. Hence, the method of lines (MOL) given in Section 3.4.2 will be implemented in Section 6.2.3.

3.6 Interpolation and Compression

Definition 3.6.1. Interpolation is the term used for methods that construct new data points from a discrete set of data points. Usually this means to construct a continuous function from a discrete set of function values.

Approximation (a curve fitting in 1D) is similar to interpolation, but it does not necessarily pass through all data points. The advantage of this particular technique is that it frequently leads to a smoother reconstruction. The drawback is generally a reduction in accuracy, image resolution or maybe precision. Moreover, with interpolation a function is sought that allows to approximate $f(x)$ such that functional values between the original data set values may be determined. With the curve fitting, one simply requires a function that is a good fit to the original data points.

Definition 3.6.2. Compression is the process of encoding data by using as few information-bearing units (usually bits) as possible, such that the inverse process, called decoding, will return the original information [Pennebaker and Mitchell, 1993].

The new three dimensional object is a polygonal fine mesh consisting of various entities such as vertices, edges, and faces that are associated to some numerical quantity or attributes such as vertex locations, normal vectors, texture coordinates, in addition to reflectance. Geometric data, specify vertex locations; connectivity data, describe the relationship between vertices, and property data specify the various other attributes that are normally attached to vertices. The real issue of compressing a 3D object is to deal with geometry and connectivity (since properties can be dealt with in the same way as geometry) and a number of methods have been proposed since the early 1990s.

In general, there are three methods one can use to compress 3D data:

1. Image-based compression: where each snapshot of a 3D scene is compressed as a 2D image. This is a palliative solution (for instance, flash animation of the three dimensional picture) and the shortcomings are that this is not fully interactive and not immersive.

2. Single-rate mesh compression: algorithms traverse the mesh searching for areas susceptible to local compression of polygonal relationships.

3. Progressive mesh compression: hierarchical refinement of a 3D structure for transmission, where a coarse mesh is increasingly refined with richer details until a full 3D model is reconstructed at the receiving end.

In this study, the interest is compression methods 2 and 3, which are focused on representing the vertex, geometry and connectivity information in the triangulated mesh.

3.7 3D Geometry Formats

In spite rapid progress in mass-storage density, processor rates of speed, and digital interaction system performance, the demand for data storage space capacity and "data-transmission" data transfer usage continues to outstrip the abilities of available technologies. The recent growth of information intense multimedia based web applications has not only increased the need for finding better ways to represent data, but also made this central storage space and interaction technology.

Here only some preferred open standard formats are highlighted, such as COLLADA, OpenGL and OBJ for file interchange of uncompressed 3D data. COLLADA is an interchanging file format for 3D applications developed by the Khronos

Group [Arnaud and Parisi, 2007; Kessenich et al., 2004]. It uses an XML schema designed to interchange digital assets across software applications. It can store information like vertices, edges, faces, texture maps, and also physical properties like weight, the centre of mass and others. The same COLLADA file can store information about multiple models. To describe the model, it first defines all the vertices in the form of an array of coordinates and then the normal direction for each face. However, the problem of loading COLLADA files directly using WebGL (also defined by the Khronos Group, WebGL is usually an instance of the canvas HTML class that provides a 3D design API implemented in a web browser without the need for plug-ins) will be that programming rapidly will become extremely intricate, as the developer needs to adapt to the file format and to what COLLADA supports. A simpler, more useful and faster the solution is to load data that have been defined in JSON (JavaScript Object Notation) format. JSON is really a textual content document that contains sets associated with ideals inside a specific order.

OBJ (or .OBJ) is a geometry based information framework first developed by Wavefront Technology for its Impressive Visualizer activity package [Kato and Ohno, 2009]. The data structure has been implemented by other 3D design program providers. In most aspects, it is a globally approved structure. The OBJ basic format is straightforward, containing geometry information only, namely the (x,y,z) position of each vertex, the (u,v) texture coordinates of each vertex, and a list of triangulated faces. The list of vertices is defined in a counter-clockwise order negating the need for explicit declaration of face normals.[Min et al., 2003].

Furthermore, object data files can be interchanged with a variety of applications. As an illustration, a 3D model can be defined in OBJ format by specifying the position of vertices in space. An example is shown below containing 8 vertices (lines starting with v) and their faces (lines starting with f specifying which vertices are connected which):

```
%Simple example for OBJ file
% Number of vertices=8
%Number of points=0
```

```
%Number of lines=0
%Number of faces=6
%Number of materials=1
% # Vertex list
v -0.5 -0.5  0.5
v -0.5 -0.5 -0.5
v -0.5 0.5  -0.5
v -0.5 0.5   0.5
v 0.5 -0.5   0.5
v 0.5 -0.5 -0.5
v 0.5  0.5 -0.5
v 0.5  0.5   0.5
%# Point/Line/Face list  use mtl Default
f 4  3  2  1
f 2  6  5  1
f 3  7  6  2
f 8  7  3  4
f 5  8  4  1
f 6  7  8  5
%End of file
```

Since this thesis uses the Matlab program for all experiments, all 3D data can be saved in *.mat format. However, in order to ensure data interchange with other applications and environments, it is proposed to save all original, uncompressed data in .OBJ format. An OBJ exporter has been written and it is included in the Appendix to this thesis that converts 3D data from the Matlab internal representation of a list of vertices and a list of faces to OBJ format. The function `'gmprWriteOBJ'` accepts four arguments: `path,` which is a string representing the file name to be saved; `points3D`, which is an n-by-3 matrix representing a list of vertices; `faces,` which is an m-by- 3 matrixes representing the list of triangular faces in the 3D structure; and `vertexcolour` which is a p-by-3 matrix representing the vertex colour. What it does is to save to the filename provided the list of vertices, faces, and so on as specified by Wavefront's OBJ file format. Please see the Matlab function on the page (170-171).

For compressed 3D data a special representation of the data is required and this is provided in Chapters 5–7. The approach in this thesis is to define the file format as an open standard, and save all data in plain ASCII. In this way, applications can be written to both compressed and uncompressed data following the procedures that will be described in subsequent chapters.

3.8 Discussion

This Chapter discussed elliptic PDEs, the boundary value problem and solutions to boundary conditions. Moreover, the importance of the Fourier sine and cosine series has been emphasized. A note is made here that different and alternative notations will be used in subsequent chapters. The numerical solution of elliptic PDEs can be presented as the Dirichlet boundary for Laplace's equation. Moreover, the Discrete Fourier Transforms, DCT and Wavelet methods are also discussed in this chapter. The general approaches to the solution of a linear system of equations are presented. It has been shown that the Dirichlet problem for Laplace's equation obtains the exact solution in a finite number of operations, but is not suitable for very large sparse matrices, especially 3-dimensional problems. Therefore, iterative methods will be considered in this research, in particular the method of lines (MOL) as it is regarded as a special finite difference method, but are more effective with regard to precision and computational time than the regular finite difference technique. This essentially involves discretizing a given differential equation in one or two dimensions while using the analytical solution in the remaining direction.

The method of lines has got the value associated with both the finite difference method and analytical process; it does not provide spurious modes, nor does it have the problem of "relative convergence". The 3D data file is also discussed and the preferred file format for uncompressed data is OBJ and for compressed data is plain ASCII whose specific information on the 3D data parameters to be saved will be described in subsequent chapters.

Chapter 4

Data Modelling and Pre-Processing

4.1 Introduction

The representation of geometric entities, such as shapes and surfaces, has been a central problem in 3D modelling. In practice, the majority of these entities are represented by triangular meshes specifying both points and connectivity. The digital representation of a real, physical object is described by point clouds, which are sampled on or near the object's surface. The 3D data used in this thesis are acquired by the GMPR scanner, which is a multiple stripe, structured light scanner. On the application of the methods proposed in this thesis it is important to understand and analyse the intrinsic geometry of point clouds in 3D to determine geometric quantities on shapes and surfaces.

The method proposed here was devised from previous research on fast 3D acquisition using structured light methods [Rodrigues et al., 2010], [Rodrigues et al., 2008], [Brink et al., 2008],[Robinson et al., 2004]. The actual 3D scanning method is dependent on splitting the projection pattern into light planes. Every plane hits the target object as a straight line and also the apparent bending of the light due to the position of the digital camera in relation to the projection allows us to calculate the depth associated with any point along the projected light plane. Taking complete advantage of such properties, the proposed method is closer to polygonal mesh compression [Peng et al., 2005; Touma and Gotsman, 1998], but with significant differences, as it does not depend on searching for local relationships that are most susceptible to compression.

In this Chapter we develop new data representation techniques, allowing 3D point cloud data to be defined as single valued functions which are then suitable for compression. We start with a 3D model that is normally represented as a point cloud or polygonal mesh that can be displayed as a smooth surface aided by surface rendering algorithms. The source data model typically uses a connected mesh of vertices with triangular faces. Our proposed technique involves a re-meshing operation over the mesh through structured cutting planes, resulting in a new set of structured vertices. This new set of vertices should not change the geometry of the mesh, providing that the cutting planes are defined as a fine grid. In this way, the sequence of points lying in each cutting plane can be described as points on a curve, and can be parametrically described by a variety of techniques.

This Chapter is organized as follows: Section 4.2 Data representation, Section 4.3 describes the connectivity of the mesh, Section 4.4 the modelling, Section 4.5 presents the method and the data sampling. Finally, a discussion in Section 4.6.

4.2 Data Representation

Without loss of generality, surfaces are described by using certain special curves, and representations for curves generalise to representations of surfaces. Furthermore, shape representation is based on the boundaries of three dimensional objects, which are generally shown as the boundaries (surfaces) of 3D objects. 3D image surfaces can be represented mathematically in various types. The most common types are implicit, explicit, and also parametric. The implicit forms are usually identified with a system regarding algebraic equations, and parametric forms are usually identified through rational polynomials, and they are known as rational curves or surfaces.

A surface can be represented in parametric form as;

$$x = x(u,v), \qquad y = y(u,v), \qquad z = z(u,v), \qquad u_1 \leq u \leq u_2, v_1 \leq v \leq v_2 \quad (4.1)$$

where the coordinates of a point (x,y,z) are expressed as a function of u and v in a closed domain. The function is assumed to be continuous with a sufficient number of continuous derivatives.

The implicit form of a point (x,y,z) satisfies an equation

$$f(x,y,z) = 0. \qquad (4.2)$$

However, the explicit form is a special case of the implicit equation. In fact, all surfaces in the implicit form can be transformed into an explicit form but not vice versa. For a successful geometric modelling in this thesis, both techniques (the implicit and parametric forms) are used.

Each 3D model acquired by the GMPR scanner is reconstructed from equally spaced light planes hitting the surface of the object as illustrated in Figure 4.1 left. On the right, the reconstructed point cloud is visualised. Any point $s = (x,y,z)$ on the point cloud corresponds to a surface point illuminated by plane n. The position of s is given by the scanning function $S(u,v,n) = (x,y,z)$, where (u,v) is the position of the point s' in a plane, and n is the index number of the plane. The index array is one-to-one mapping which takes all the points and labels them with the index of the plane. Not all vertices defined over this grid contain valid data, vertices with data are marked as valid otherwise invalid. In other words, every index array element $[c][r]$ which does not contain valid data will have a value of NULL.

The structure of the data means that the connectivity of the vertices is a derived property, and triangulation of the surface is thus a straightforward task without the need for complex triangulation algorithms. The techniques are described in the following sections.

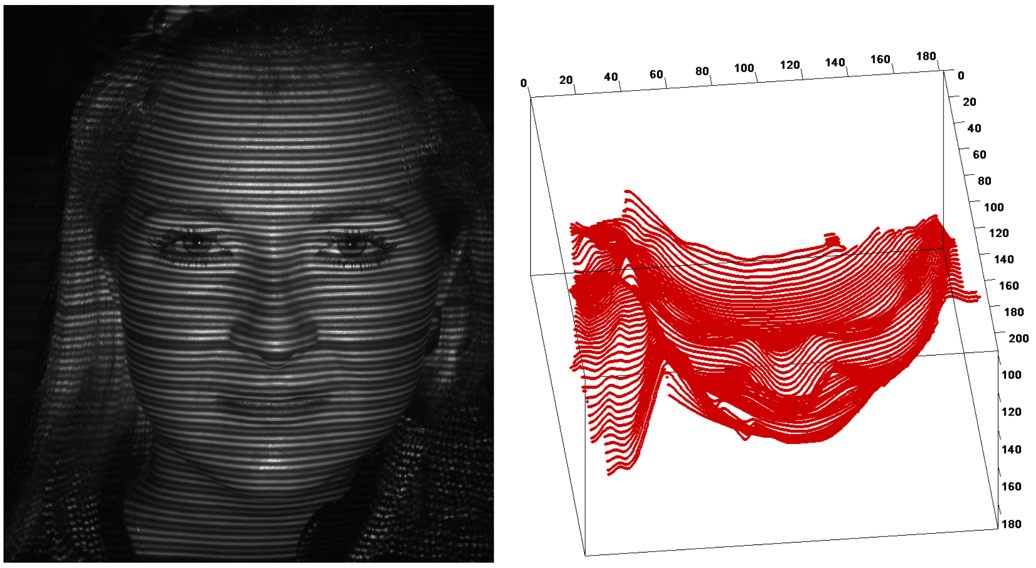

Figure 4.1: The GMPR scanner maps light planes hitting the target to surface points (x, y, z).

4.3 Creating Scattered Interpolation Points

The coding process of polygonal meshes can usually be divided into two components: connectivity and geometry. Connectivity coding works with the topology of the mesh, or quite simply the adjacency relationships between the polygons. On the other hand, geometry coding works with the position within place, or coordinates, of each and every vertex along with optionally the standard, colourings or other model properties. Generally speaking, geometry coding will probably take advantage of the connection details to increase the data compression efficiency. Compression methods are, thus, focused on representing the geometry and connectivity of the vertices in the triangulated mesh. Geometrical approaches aim to reduce the size of the mesh by simplifying its geometry and approaches include geometry coding [Taubin et al., 1998].

A 3D source data model typically uses a connected mesh of vertices with triangular faces, which is the standard data type in many 3D computer generated models, such as Wavefront OBJ, Java 3D, VRML and COLLADA formats [Ames et al., 1997; Arnaud and Barnes, 2006; Chen and Chen, 2008]. In the GMPR 3D scanning system [Brink et al., 2008; Robinson et al., 2004], the model is a constrained version of this mesh, with rows and columns of vertices connected in a rectangular pattern (see Figure 4.1), conforming to the stripes in the original-projected pattern. The Figure clearly suggests that in mapping to 3D space one can simply save the 3-part vector for each vertex, without the need for a separate list of faces and vertex connections, as required in the 3D file formats mentioned above. This explicit arrangement of 3D points makes mesh triangulation a simpler and more reliable process than with an arbitrarily connected mesh, gives a more compact data representation, and allows smaller file sizes when compared with OBJ, VRML and COLLADA formats. Some similarities to the use of triangle strips to encode mesh connectivity can be found in the literature, and a distant resemblance to the method developed by [Auerbach et al., 1997] is acknowledged.

The actual proposed data compression and decompression scheme relies on an adaptive sparsification of the data by means of triangulation coding. In this coding, data are decomposed into a number of triangular regions such that within each region, it can be recovered in sufficient quality by interpolation from the vertices.

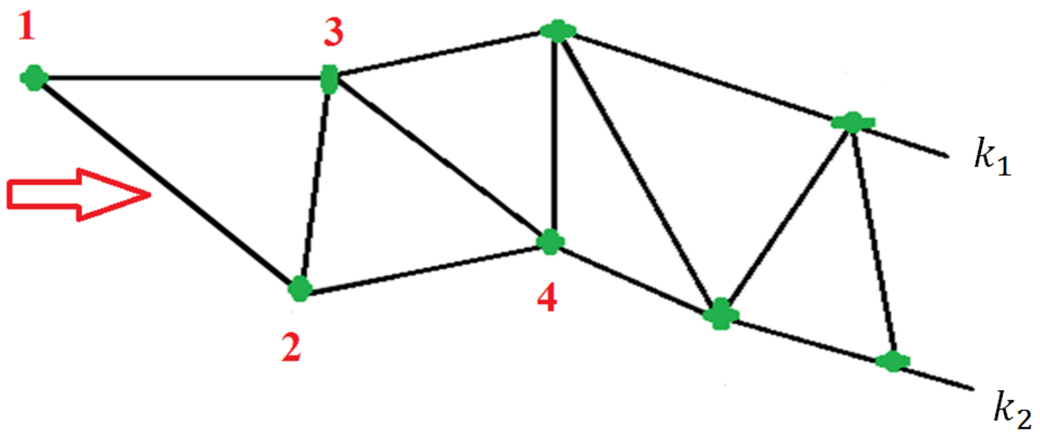

Figure 4.2: The implicit triangulation method between two planes k_1, k_2.

Mesh triangulation is performed between each pair of cutting planes. The idea is that what is required is the sequence of vertices in each plane. Each vertex in one sequence would be paired to their counterpart in the other sequence, within a specified sequence of vertex indices.This is shown in Figure 4.2 for two sequential planes k_1 and k_2 where the neighbouring vertices have been assigned sequential indices. Triangulation then proceeds as follows: using vertices labelled as 1,2,3,4 create the first triangle by connecting vertices 1-2-3 then create the second triangle by connecting vertices 2-3-4 to close the first block. Then move one vertex to the right on planes k_1 and k_2 and relabel them as 1,2,3,4. The same sequence is repeated by moving to the second block taking vertices 1-2-3 then 2-3-4,and then repeat the triangulation process until reaching the end of planes k_1 and k_2. Furthermore, the same will be done in the second and third planes and so on until the whole mesh is triangulated.

In this way, the triangulation (or connectivity of the mesh) is not coded at all, as it can be a derived property of two adjacent planes. The Figure 4.3 shows a number of cutting planes whose triangulation is obtained following the procedure described above.

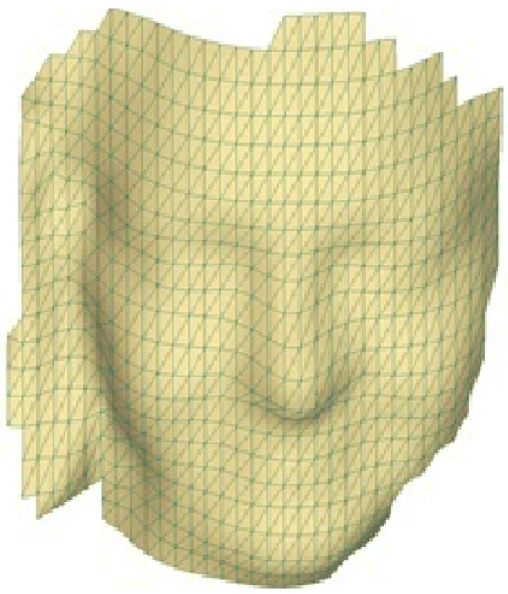
Figure 4.3: Connected path of triangulation mesh.

It is clear from the above discussion that what are subject to compression are the sequences of vertices in each cutting plane, as triangulation for 3D reconstruction and visualisation becomes a straightforward task. To specify reasonable interpolation when interpolating between function values, a function is required that smoothly connects function values. Natural options for differential operators are thus smoothing operators. When used for compression there is also another very important factor that needs consideration: performance. The discrete operator must be as easily computable as possible, otherwise the technique will be very impractical.

In certain applications, it is appropriate to be able to make both three- and four-sided polygons, as most rendering hardware support only three- or four-sided faces. If a renderer supports only three-sided faces, then polygons may be constructed out of triangular strips as illustrated in Figure 4.2. However, many renderers support quads along with larger sided polygons, or are able to turn polygons into triangles on the fly. In any case, following the procedure of defining polygons from the sequence of points in each plane as described above, causes it to become unnecessary to store a new fine mesh in a triangulated form as a sequence of vertices in the plane will suffice for correct triangulation.

4.4 Modelling

Whilst reconstruction specifications may force the decision associated with regardless of whether the data compression plan will be lossy or even lossless, the precise data compression plan utilized is determined by a variety of elements. Probably the most important elements would be the characteristics of the data that need to be compressed. Modelling is a process of setting up an environment to allow the variables of interest to be observed or certain behaviour of a system to be explored. It is a formalisation and extension to the description of a problem. Rules and relationships are often set in mathematical formulae.

Modelling is to extract the information about any redundancy that exists in the data and describe the redundancy in mathematical terms. Typically, the coding is a description of the model and a "description" showing how the data differ from the model. Each tends to be encoded, usually utilizing a binary alphabet. The variation involving the data and the model is often referred to as the residual. Modelling is a critical stage of algorithm design. The model can sometimes define immediately the approaches of the algorithm [Pu, 2005]. In addition, the modelling stage consists of identifying the very best rendering for just about any type of the reconstructed model. Also, there are various ways associated with modelling an object based on the input data, the rendering algorithm and the final uses of the model.

At a high level of description, in this thesis, the proposed model representation and compression techniques used are as follows.

- Source data. The source data model typically uses a connected mesh of vertices with triangular faces.
- Sampling points. The method defines a large number of horizontal and vertical mesh cutting planes and the intersection of all planes on the mesh defines a set of sampling points. The sampling method by structured cutting planes operating on the source data is described in Section 4.5. The connectivity of the mesh is derived directly from the vertices in each cutting plane. The explicit structure of the sampled vertices allows the definition of the x and y coordinates on a regular grid while the z-values will be subject to interpolation and compression by several methods.
- Sampling of data points are compressed using FFT, DCT and DWT, and PDEs are used at the decompression stage to interpolate data between cutting planes after the inverse transformations iFFT, iDCT and iDWT are applied.

4.5 Method

4.5.1 Polygon Reduction by Explicit Structured Vertices

One of the many requirements with regard to geometric design systems is the ability to parameterize the shape of objects. A simple tactic would be to create a general description of an object or class of objects, in which the shape is managed through the values involving a set of design variables or parameters. Moreover, the function of a boundary representation is to describe an object in terms of its boundary surfaces: vertices, edges and faces. In the simplest case, the faces are restricted to planar polygons and the representation is thus a polygonal mesh. The method presented in this Chapter applies to the manner in which re-sampling converts a mesh model or a patch of the model into a regular grid of z-values. An example of such data captured with the GMPR structured light scanning technique [Robinson et al., 2004], is depicted in the Figure 4.4 below.

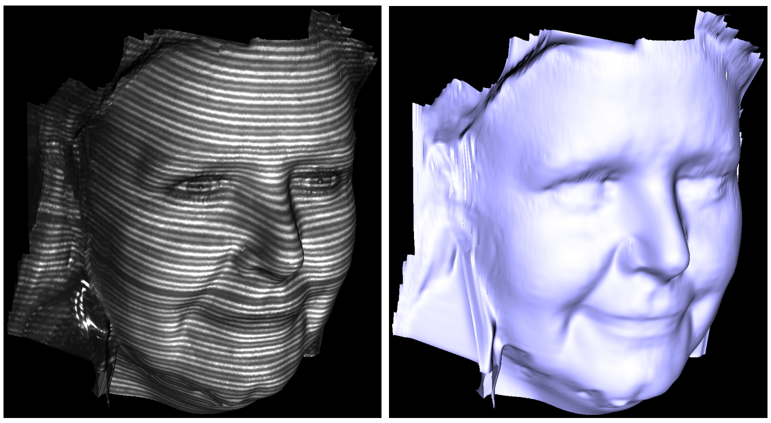

Figure 4.4: Example of a textured and shaded 3D model acquired by the GMPR structured light technique.

The measured surface of the 3D image was represented as a point cloud or polygonal mesh that displayed as a smooth picture aided by the surface rendering algorithms. Figure 4.5 shows a regular grid for sampling data points at the intersection of vertical and horizontal planes. It is important to stress that although the GMPR data have a structure defined by each light plane, the method regarding sampling by cutting planes described here assumes an arbitrary mesh with an arbitrary polygon structure. The purpose of the method is to guarantee that an arbitrary mesh is given the desired structure for compression and reconstruction. In this way, a simpler and the more reliable triangulation process is obtained than with an arbitrarily connected mesh that may require demanding triangulation algorithms such as Delaunay [Weatherill and Hassan, 1994].

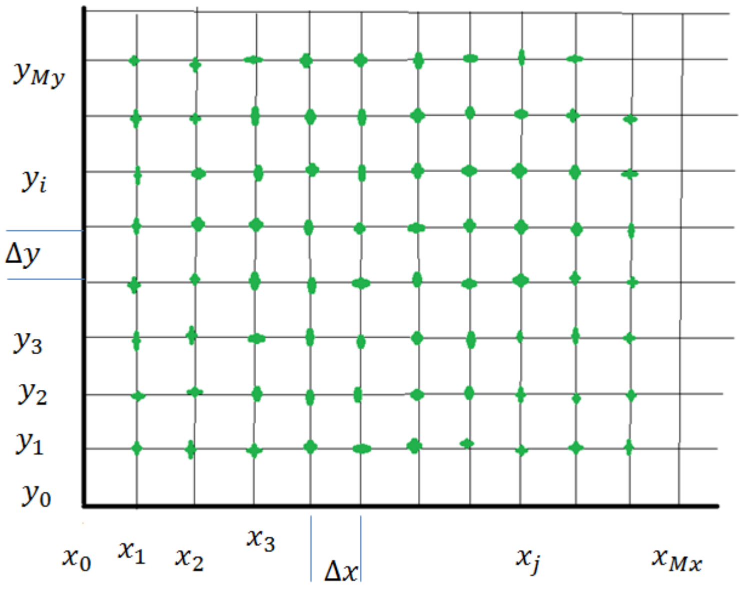

Figure 4.5: Sampling points on a regular grid

In a surface patch, the height of a point is represented by its z-value and it can be stated that the height of a function (x,y) is some function $f(x,y)$.

$$P(u,v) = (u,v,f(u,v)) \tag{4.3}$$

with normal vector $\mathbf{n}(u,v) = (-\delta f/\delta u, -\delta f/\delta v, 1)$. Both u and v are the dependent variables for the function; u-contours lie in planes of constant x, and v-contours lie in planes of constant y. Whenever such a patch is visualized within three dimensional, utilizing quads, for instance, each single edge of the polygon is a trace of the surface cut by a plane with $x = k_1$ and $y = k_2$ for some values associated with k_1 and k_2.

4.5.2 Data Sampling

Sampling will reduce the number of polygons in a mesh. Our method involves polygon reduction through cutting planes defined on a regular grid, resulting in an arrangement of explicit-structured vertices. The mesh to be sampled is a randomly oriented surface patch described in relation to a global coordinate system. The bounding box (or minimum bounding box) of the patch in 3D can be estimated by geometric algorithms (see, for example, [Geng et al., 2013; Lahanas et al., 2000; Lee et al., 2002; Quinn et al., 2007]). The patch must be rotated until its bounding box edges are aligned with the x-, y-, and z-axes of the global coordinate system.

The characteristic feature of the bounding box is the ratio between the edges. Moreover, the discrete grid coordinates are $gc_i = (u,v)^T$ of all cells c_i belonging to the respective connected component. For the face models used to demonstrate the concepts in this thesis, the smallest dimension of the bounding box is aligned with the z-axis, as this will guarantee that the face models are oriented correctly. The correct orientation of the bounding box depends on the actual characteristics of the data, however, the general principle is that the z-axis is normal to the image sensor in standard 3D scanners.

The bounding box is defined by the minimum and maximum values $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$ of the data. Because of the characteristics of the scanning method, the 3D data are defined as a matrix where the values in it are the z depth of each vertex. Theoretically (because of the characteristics of the surface and possible, missing points at the boundaries, when 3D scanning), it is possible to find a smaller bounding box in some models by rotating the mesh around its x and y-axes in very small steps. However, by simply taking the minimum and maximum (x,y,z) from the scanned model is very near to the optimal minimum bounding box. Therefore, due to the characteristics of the GMPR 3D data, it is not necessary to determine the optimal minimum bounding box in this dissertation and this does not adversely affect compression algorithms. Please see the Matlab function for bounding box `'gmprDrawBox3d.m'` in Appendix B, pages (171-172).

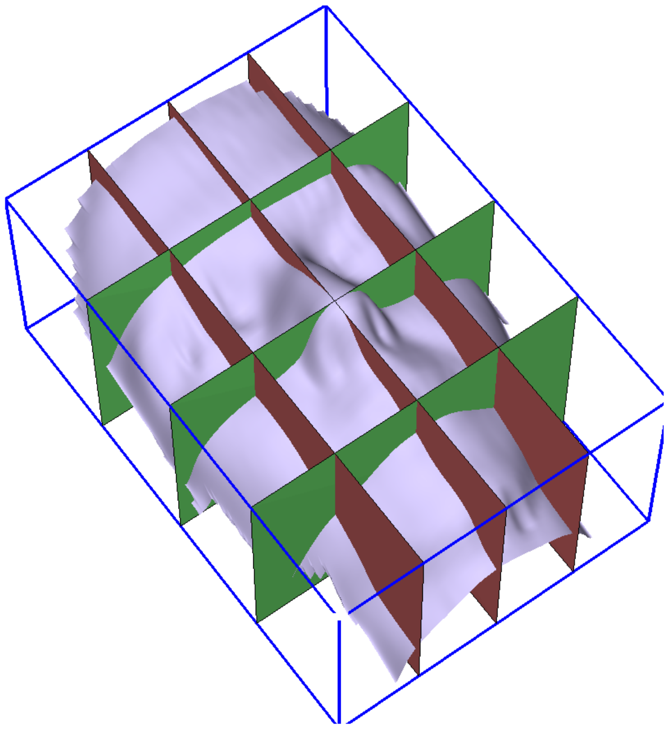
Figure 4.6: The bounding box and structured cutting planes.

A number of structured cutting planes are defined within the boundaries of the bounding box; let us call these 'horizontal' and 'vertical' cutting planes as illustrated in Figure 4.6 where, for clarity, only a few planes are shown. These planes are, thus, defined as parallel to one of the x or y-axes with normal vectors $(1,0,0)$ and $(0,1,0)$ respectively.

Thus the intersection of any two planes defines a line and the point where such line intersects the mesh is defined as a structured vertex. Thus, the number or the density of structured vertices can be controlled by the number of planes in either direction. An issue here is that one cannot guarantee that the intersection of two planes on the mesh will rest on a vertex. More likely, it will intersect somewhere on the face of a polygon. A reasonable approximation would be to move all vertices such that they lie in the cutting planes.

Therefore, in practice it is more convenient to define the cutting planes, then search for each vertex and find the nearest plane to that vertex and move the vertex to the plane. In addition, for convenience, this particular stage is performed at the three dimensional reconstruction stage, such that the data provided already contain vertices within their respective planes. The set of all points belonging to a particular plane is a subset of structured vertices. Based on the characteristics of the surface patch, the set of vertices lying in either horizontal or vertical planes can be selected. If the selected points lie in a plane with a normal vector $(0,1,0)$, the distance between structured vertices in that plane is the distance between planes with a normal $(1,0,0)$ and vice-versa. Calling D_1 as distances between structured vertices in the horizontal plane with normal vector $(0,1,0)$ and D_2 between planes with normal vector $(1,0,0)$, the x and y coordinates of any structured vertex can be recovered for all planes k:

$$\begin{aligned} x_r &= rD_1, \quad \text{where} \quad r = 1,2,\ldots,k_1 && (4.4)\\ y_c &= cD_2, \quad \text{where} \quad c = 1,2,\ldots,k_2 && (4.5)\\ z_{rc} &= z_i, \quad \text{where} \quad i = 1,2,\ldots,k_1k_2 && (4.6) \end{aligned}$$

where (r,c) are the indices of planes. This is significant as, in a stroke, $2/3$ of the 3D data can safely be discarded in the sense that it is not necessary to save the actual values (x,y) of each vertex; instead, only D_1, D_2, k_1 and k_2 are kept for each plane. The number of cutting planes is controlled bearing in mind that the resulting structured vertices should still be representative of the original mesh.

The z-values can be expressed by Eqs 4.3 as a single valued function and mapped to each combination of (x_r, y_c). If one chooses to represent these as the set of structured vertices belonging to planes with normal $(0,1,0)$, this is reduced to a 2D case in which on the horizontal axis there are exactly k_2 points with a constant step of D_2 and on the vertical axis their corresponding z-values. The above operations mean that, starting from a surface patch with a complex polygonal arrangement, one obtains a structured mesh where the number of polygons is reduced and triangulation becomes an unimportant procedure, as it is only necessary to connect vertices from adjacent planes. In other words, the mesh now contains an underlying explicit structure for triangulation.

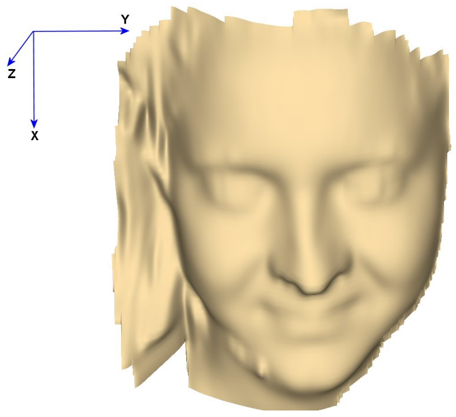

Figure 4.7: Original 3D mesh with 48,672 vertices and 78,043 faces. The size of the file (OBJ format) is 4.83MB with texture mapping and 4.0MB with no texture.

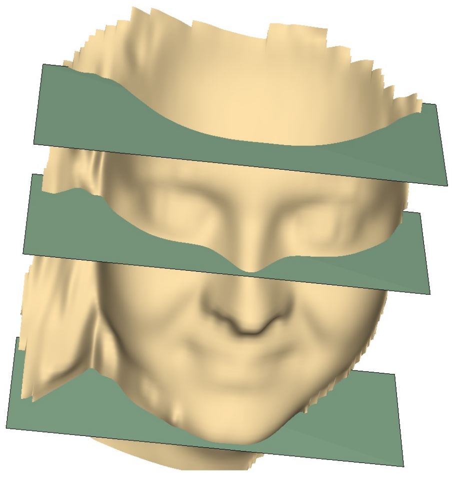

Figure 4.8: Horizontal planes with normal $n = (1,0,0)$ are cut through the mesh, from top to bottom (only 3 planes are shown here).

The proposed steps and parameters for data compression are summarised as follows:

1. A given triangulated surface patch acquired using a structured light scanner is aligned to the global coordinate system where the smallest dimension of its bounding box is aligned with the z-axis (Figure 4.7).

2. A number k_1 of horizontal planes with normal $\mathbf{n} = (1,0,0)^T$ cut the mesh

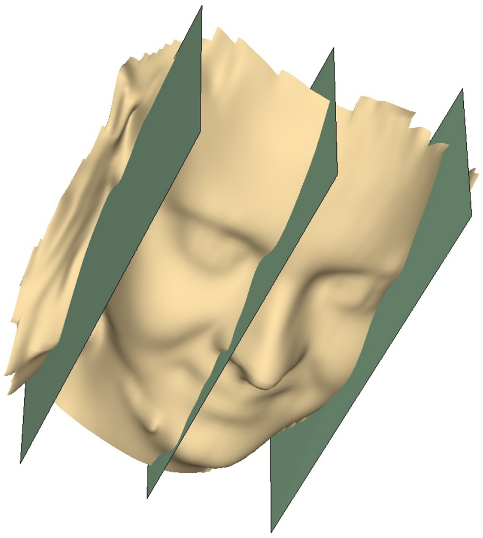
Figure 4.9: Vertical planes with normal $n = (0,1,0)$ are cut through the mesh, from left to right (only 3 planes are shown here).

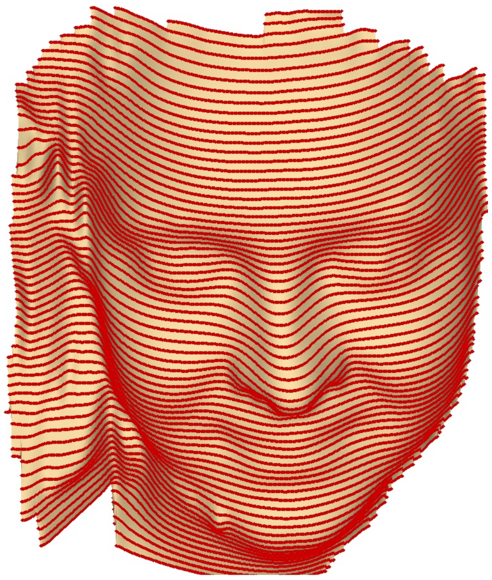
Figure 4.10: The intersection points of each horizontal and vertical planes on the mesh are estimated and marked with a point in red. The model shown has 39,743 valid vertices or intersection points.

as shown in Figure 4.8 (only 3 planes are shown for clarity). These planes are parallel to the $Y-Z$ plane of the coordinate system in Figure 4.7.

3. A number k_2 of vertical planes with normal $\mathbf{n} = (0,1,0)^T$ cut the mesh as shown in Figure 4.9 (only 3 planes are shown for clarity). These planes are parallel to the $X-Z$ plane of the coordinate system.

4. The intersection of each plane k_r with plane k_c defines a line. For each line, determine the point of intersection, by finding the nearest point on the mesh to this line. These are the heights of the single-valued function. In practice the nearest vertex to the intersection line is found as this introduces negligible error.

5. For each plane k_r, k_c make a list of the intersection points.

6. The distance between each horizontal plane is defined as a constant D_1 and the distance between vertical planes is defined as a constant D_2. This is a form of quantization that allows the recovery of (x,y) and the z value is the only variable under compression.

The Matlab methods and data structures developed to load the scanned GMPR 3D data, check for valid and invalid data, visualize, and manipulate and prepare data for compression as required are listed in Appendix B, namely functions

`gmprCalculatePlane.m' page(173-174)
`gmprDrawEdge3d.m', `gmprDrawPlane.m' page(174-177)
`gmprLoadData.m' page(203-205) `gmprSurfaceView.m' page(205-239).

Since the original superfine mesh is a (potentially) dense mesh, and the cutting planes technique will yield a sparse mesh, there are two stages of data compression. First, an initial data compression with re-meshing, then a final data compression with a transformation which is compressed and reconstructed using several methods shown in Chapter 5–7. The polygonal reduction by cutting planes or re-meshing, technique is, in itself, a compressed representation of the original data. In order to demonstrate just how much reduction in data is acquired through re-meshing procedures, Tables 4.1, 4.2 and 4.3 depict reduction rates for all tested data files. It is shown that the re-meshing operation by cutting planes, compared with the OBJ files, for all 86 models yields an average initial compression rate of 52%.

4.6 Discussion

The work presented in this chapter has focused on detailing the proposed method for data modelling and pre-processing, with an example of using the cutting plane technique being discussed.

The original idea is to define a re-meshing operation by the proposed technique of structured cutting planes, resulting in a new set of structured vertices. This new set of vertices should not change the geometry of the mesh, provided that the cutting planes are defined as a fine grid. In this way, the sequence of points lying in each cutting plane can be described as points on a curve, and can be parametrically defined by a number of techniques. In this way, only the parameters of such curves are subject to compression. Therefore, to reconstruct the 3D data it is a matter of reconstructing each curve and recovering the original data points. There are a number of immediate improvements possible, at different stages of processing, such as, for example, computing minimum bounding boxes to improve segmentation of the 3D point cloud.

Since the original 3D data are represented as a point cloud or as a triangulated mesh, to recover the original data after compression, it is necessary to clearly define the structure of the cutting planes. These are normally ‘horizontal’ and ‘vertical’ planes operating within the boundaries of the bounding box. Moreover, the mesh cutting planes can be oriented with a global coordinate system with a constant step. The choice of step size depends on the characteristics of the data and the desired quality of the compression. For the face data used to demonstrate the method in the next few chapters, vertical planes are chosen 8 to 10 times more than horizontal ones, as the former are found to reconstruct the mesh with good quality.

In the following chapters, a polynomial interpolation will be formulated and im-

Table 4.1: Initial compression by re-meshing operation

File number	Initial compress size in KB	Original superfine mesh size in KB	Reduction Rates
1	1,846	3,514	53%
2	1,434	2,711	53%
3	2,138	4,100	52%
4	1,868	3,548	53%
5	1,423	2,972	48%
6	1,587	2,956	54%
7	1,583	2,980	53%
8	1,063	1,963	54%
9	1,077	2,116	51%
10	2,297	4,415	52%
11	1,873	3,674	51%
12	1,703	3,202	53%
13	2,223	4,255	52%
14	1,587	3,059	52%
15	1,912	3,675	52%
16	1,139	2,223	51%
17	1,550	2,933	53%
18	881	1,683	52%
19	1,541	2,915	53%
20	1,332	2,658	50%
21	1,532	2,895	53%
22	975	1,965	50%
23	1,377	2,617	53%
24	1,127	2,100	54%
25	1,044	1,970	53%
26	860	1,717	50%
27	1,169	2,393	49%
28	882	1,770	50%
29	2,285	4,794	48%
30	1,255	2,401	52%
31	1,289	2,630	49%
32	1,688	3,262	52%
33	898	1,799	50%
34	950	1,923	49%
35	1,534	3,063	50%
36	1,611	3,022	53%

Table 4.2: Initial compression by re-meshing operation

File number	Initial compress size in KB	Original superfine mesh size in KB	Reduction Rates
37	1,304	2,459	53%
38	1,276	2,389	53%
39	1,376	2,568	54%
40	1,244	2,360	53%
41	1,256	2,346	54%
42	1,327	2,512	53%
43	975	1,995	49%
44	1,896	3,522	54%
45	2,160	4,311	50%
46	2,549	5,329	48%
47	1,620	3,067	53%
48	1,207	2,419	50%
49	2,153	4,123	52%
50	2,003	3,821	52%
51	1,583	3,049	52%
52	1,849	3,547	52%
53	2,106	3,998	53%
54	1,821	3,421	53%
55	1,356	2,300	59%
56	1,935	3,613	54%
57	1,857	3,652	51%
58	1,937	3,644	53%
59	2,271	4,328	52%
60	1,655	3,214	51%
61	1,863	3,563	52%
62	1,538	3,072	50%
63	3,150	6,311	50%
64	955	1,896	50%
65	1,716	3,427	50%
66	2,020	4,114	49%
67	2,335	4,670	50%
68	1,666	3,274	51%
69	2,054	4,006	51%

Table 4.3: Initial compression by re-meshing operation

File number	Initial compress size in KB	Original superfine mesh size in KB	Reduction Rates
70	1,434	2,838	51%
71	1,464	2,819	52%
72	1,696	3,235	52%
73	1,311	2,545	52%
74	1,527	2,917	52%
75	1,846	3,504	53%
76	1,656	3,278	51%
77	2,276	4,564	50%
78	1,348	2,636	51%
79	1,621	3,117	52%
80	2,209	4,310	51%
81	2,136	4,006	53%
82	1,265	2,395	53%
83	1,825	3,664	50%
84	1,589	3,119	51%
85	1,847	3,523	52%
86	1,865	3,642	51%

plemented for surface patches, and several reconfigurable computing approaches based on the implementation of spectral methods will be presented and evaluated.

Chapter 5

Efficient 3D Data Compression Through Parameterization of Free-Form Surface Patches

5.1 Introduction

This study seeks to present a new technique for 3D data compression centred on the parameterization of surface patches. The data pre-processing has been defined in Chapter 4. A significant feature of this technique is that, it defines the number of cutting planes on the mesh, while the connection or intersections of the planes on the mesh define a set of sampling points. An explicit structure that allows for the parametric definition of both x and y coordinates is contained in these points and the z-values are interpolated using a high degree polynomials. Reconstruction is then achieved by evaluating the polynomials from the saved information, once each plane is recovered by the uncompressing method, and triangulation is achieved given the explicit structure and pairing of the planes and data points as described in Section 4.5. The desired outcome is a polynomial interpolation through most of the control points.

This chapter is structured as follows. Section 5.2 introduces polynomial interpolation, and Section 5.3 describe the instantiation of the method for reconstructing surface patches. Finally, a discussion is given in Section 5.4.

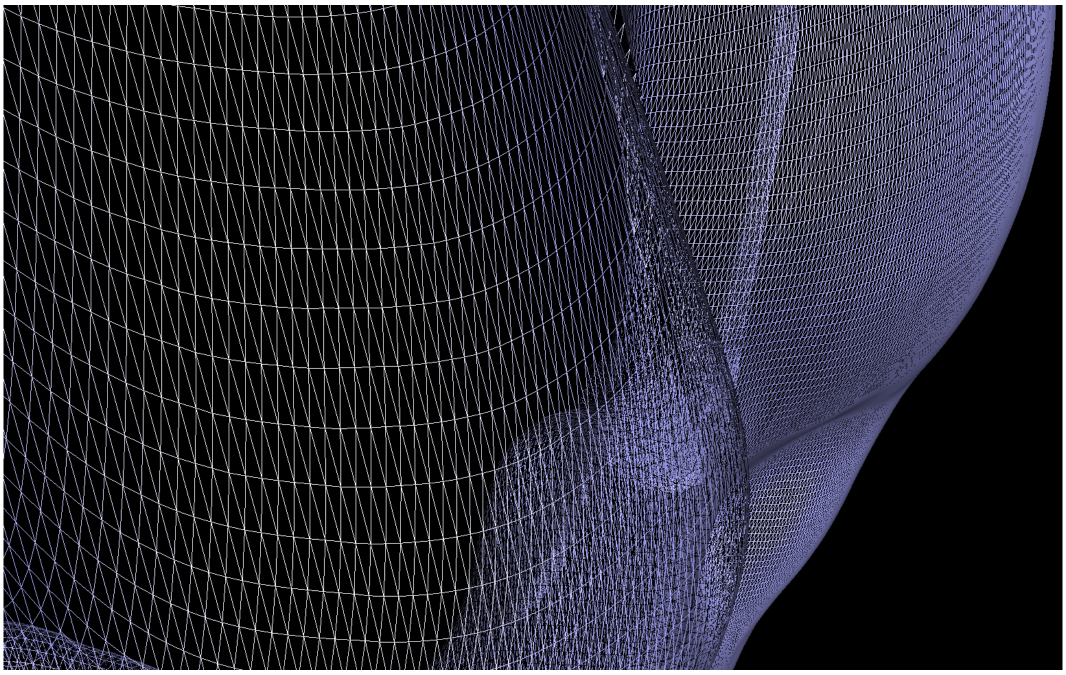

Figure 5.1: Polygonal mesh detail

5.2 Polynomial Interpolation

The process of reconstructing a curve, the surface, or other geometric objects from certain known data can be achieved by interpolation, a word that is derived from the Latin word “interpolate” which means “to refurbish” or “to patch” [Bergh and Löfström, 1976; Davis, 1975; Shepard, 1968; Triebel, 1999]. Portions of curved graph surfaces can be represented as surface patches modelled using polynomials of two variables. For example, a plane can be represented as

$$z = a_0 + a_1 x + a_2 y, \tag{5.1}$$

and curved surface patches can be modelled using higher-order polynomials. In general, if we have two points (x_1, y_1) and (x_2, y_2) on a plane with $x_1 \neq x_2$, the first degree polynomial in x is a straight line. Then given n points in the plane, $(x_k, y_k), k = 1, 2, 3, ..., n$, there is a polynomial in x of degree less that n whose graph passes through or close to the points.

The polynomial of n-th degree in z has the form:

$$P(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \cdots + a_n z^n \tag{5.2}$$

where $a_0, a_1, a_2, \cdots, a_n$ are the coefficients and n is the degree of the polynomial.

Using the data described in Section 4.5, a useful approach to compression would be to represent the data in each plane through fitting a polynomial of a high degree that best fits the data. In this way, only the polynomial coefficients and their boundaries need to be kept allowing reconstruction of the original data within the specified boundaries. To implement the polynomial method on our data, we have to do the following:

1. In the rectangular grid of 3D data (which is defined from a 2D image by the GMPR scanner), valid vertices are defined by the intersection of horizontal and vertical planes with the mesh; any missing vertex is marked as invalid. The polynomial is evaluated over the valid vertices in each plane. The 3D data can then be represented and recovered by the polynomial's vector of coefficients.

2. First, for each plane perform a polynomial fit of a given degree n to get the $n+1$ set of coefficients that best describe the data, by using the Matlab built-in function `polyfit` as:

   $$P1 = \texttt{polyfit}(y, z, n); \tag{5.3}$$

   where $P1$ is a vector of $n+1$ coefficients, y,z are vertex points and n is the polynomial degree.

3. Second, the coefficients of the polynomial are saved for each curve together

with the indices of the k planes for the first and last valid vertices. Then, for each model we get a matrix corresponding to the cutting planes, this matrix is built row by row, where each row of the matrix corresponds to a cutting plane as follows:

$$\begin{aligned} P2 = \big[&\text{coefficients-of-plane-1} \quad \text{bFirst} \quad \text{bLast};\\ &\text{coefficients-of-plane-2} \quad \text{bFirst} \quad \text{bLast};\\ &\text{coefficients-of-plane-3} \quad \text{bFirst} \quad \text{bLast};\\ &\vdots \ldots \big] \end{aligned} \tag{5.4}$$

4. Third, the actual uncompressed mesh is acquired by reconstructing each set associated with z-data curve fitting through the Matlab built-in function `polyval` (a substitute for every value into a polynomial and come up with a corresponding value) as follows:

$$P = \texttt{polyval}(C, Y); \tag{5.5}$$

   where C is a vector of coefficients from the current plane (from $P2$ above) and Y is a vector of equally spaced (by D_2) from bFirst and bLast indices from $P2$ above. Polynomial fitting is performed in each plane using the z-values as "control points" to reproduce the measured data on the known locations. This means that the interpolation will generate data that are close to the source grid. In this case, it will not recreate the actual information at the known location; however, it will be fitting a curve (model) to a known data set on the source grid and estimating the values based on the fitted curve in the destination grid.

In addition, the coefficients of the polynomial are saved for each curve together with the indices of the k planes for the first and last valid vertices. This is so because there may be several plane intersections that do not intersect the mesh and such combination of indices (k_r, k_c) must be marked as invalid vertices – the polynomial is only valid between the specified vertices, it cannot be extrapolated.

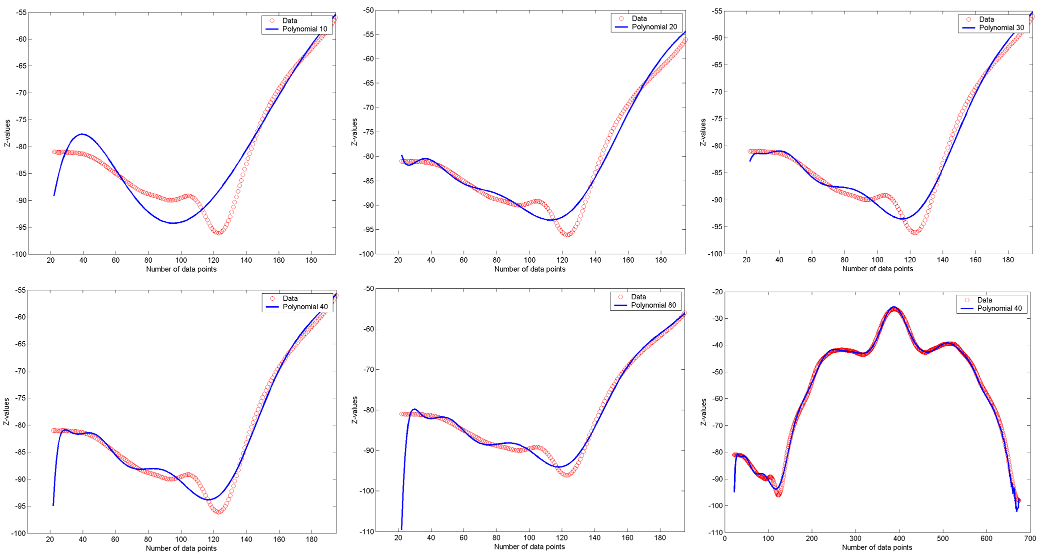

Figure 5.2: Polynomial interpolation for the first few vertices in a cutting plane, not to scale. First row: degrees 10, 20, 30; second row: degrees 40, 80, and a full cutting plane with degree 40. Red: original data, blue: interpolated data.

Figure 5.2 illustrates polynomial interpolation on a first few vertices defined by the cutting plane data marked in red, while data in blue are the results of interpolation by degree 10, 20, 30, 40 and 80. It is clear that none of the results are satisfactory with large errors at the extremities. Analysis over the entire meshes will be described in the next section. Recall that the data are only a sequence of vertex positions in 3D space and also the objective is to replace the sequence of vertices by a parametric definition using high order polynomials. For high-density data as is the case of 3D models, this provides a substantial data reduction. To illustrate the compactness of this representation by using data that have been sampled as defined in Section 4.5.2, assume a mesh with 100,000 vertices. This means 300,000 floating points (one floating point for each of the (x, y, z) values.) Since both (x, y) are defined as a regular grid with spacing defined by the constant distance between cutting planes, it is possible to instantly eliminate 200,000 floating point from representation, replacing these by 4 numbers only: two constant spacings between cutting planes, and the number of rows and columns that make the regular grid.

Therefore, if a mesh is cut with 100 planes, only a set of 100 polynomial coefficients together with the first and last valid vertex indices for each polynomial are required to fully reconstruct the mesh. Assuming a polynomial of degree 25, only 28 numbers are needed for each plane: 26 coefficients plus 2 vertex indices. In the example above, this would be a reduction from 100,000 to 2,800 floating, point numbers. To reconstruct the original mesh, the polynomials used in Eq. 5.2 are evaluated for each plane within their boundaries (first and last valid vertices), and the (x, y) values are evaluated for each combination of (r, c) plane indices through Eqs. 4.4 and 4.5.

For instance, in order to implement the method above for a polynomial of degree 3:

$$P(z) = a_0 + a_1 z_1 + a_2 z_2^2 + a_3 z_3^3 \tag{5.6}$$

First, for each plane a polynomial fit of degree 3 is performed to get the 4 coefficients by using Eq. 5.3 Then save these 4 coefficients of each curve together with the first and last valid point. Furthermore, the exact same technique will apply for the second plane by saving the 4 coefficients with the very first and the last valid point, and repeat the same method to the remaining planes over the model. Therefore, by using Eq. 5.4 we are building a matrix row by row and each row corresponds to a cutting plane with 4 coefficients and 2 vertex indices.

Finally, reconstruction is achieved by evaluating the polynomials from the saved information, by applying Eq. 5.5. Figure 5.3 illustrates that for a polynomial interpolation of degree 3, once each plane is recovered by the uncompressing method, then triangulation is achieved by pairing the planes and data, which produce unsatisfactory interpolation as the actual model appears very poor. This is so because polynomial fitting of degree 3 will not go through most (if any) of the control points, and thus is unable to reconstruct the face model with a reasonable likeness to the original.

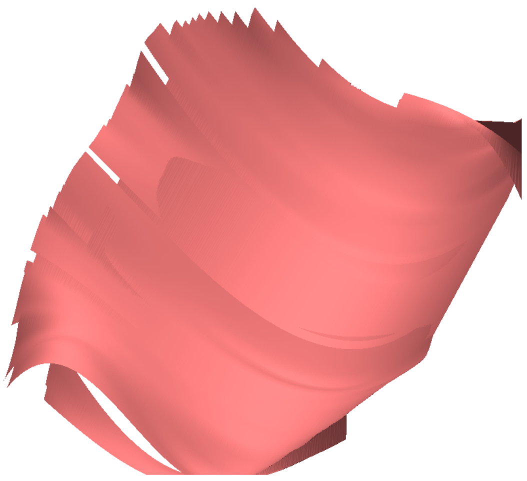

Figure 5.3: Model reconstruction using a polynomial interpolation of degree 3

5.3 Results

In this section, we use the method of mesh sampling described in the Section 4.5 with a comparative analysis of interpolation using various high degree polynomials.

5.3.1 Data Compression by Polynomial

By following the method described in Section 4.5, polynomial interpolation is performed where the coefficients and the plane indices of the first and last valid points are saved. Figure 4.10 depicts the intersection of all horizontal and vertical planes where each intersection is marked with a red point. The structure of the mesh is $k_1 \times k_2$, whose choice depends on the characteristics of the model and the accuracy required. For the models used here, normally 8 to 10 times, more vertical planes than horizontal ones are used due to the characteristics of the GMPR scanner [Robinson et al., 2004]. Ultimately, the number of planes is based on the characteristics of the data; it was found that approximately 50–80 horizontal planes across the face provide for good reconstruction. Thus the number of vertical planes was determined at around 10 times the horizontal scale; this provides a large number of data points for polynomial interpolation and results in a grid 72×676 for the particular face model shown (for different models these dimensions will vary). The horizontal planes are quite noticeable in the Figure 4.10, while the vertical planes are less so due to their proximity.

A polynomial interpolation of high degree is suggested, as the intention is to find a polynomial that goes as closely as possible through most of the control points. In order to be able to reconstruct the set of points later on, the first and last valid points of each list of points are saved, together with each set of coefficients, the size k_1, k_2 of the sampled 3D data structure and the distance between planes D_1 and D_2. This information is organized in the file header:

```
k1 72
k2 676
D1 3.3
D2 0.3
8.0960151e-031 ...  6.7726253e+002 382 482
...
1.8712059e-032 ...  1.0464188e+007 143 437
```

Reconstruction is then achieved by evaluating the polynomials from the saved information. In the file structure above, the 4 lines of header information are followed by 72 lines of polynomial coefficients with their first and last valid points. The degree of the polynomial is inferred from the data. If each line has, say, 23 numbers, the last two numbers are the indices of the first and last valid points, leaving the preceding 21 numbers as polynomial coefficients C. The degree of the polynomial is $C - 1$. Thus, in this case, the data was interpolated with a polynomial of degree 20.

5.3.2 3D Reconstruction

A high-level view of the method is as follows. Given an unstructured mesh, apply the re-meshing technique of cutting planes, which will result in data within a structured regular grid. The values of (x, y) are known from the grid and the only variable to interpolate is the depth value z. Each set of points lying in the plane are thus subject to interpolation. Below are shown the effects of reconstructing a face model using polynomials of various degrees. Figures 5.4, 5.5 and 5.6 show results for polynomials of degrees 3, 10, 15, 20, 30, 40, and 80.

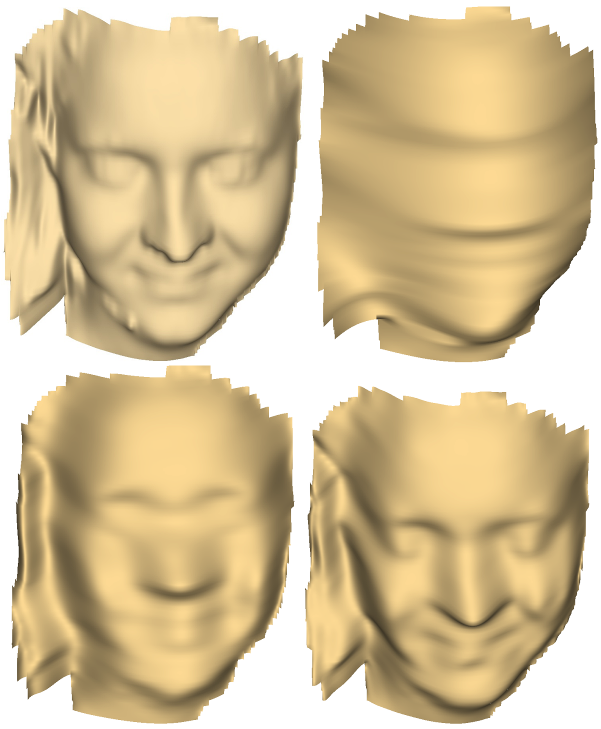

Figure 5.4: Polynomial interpolation degrees 3 to 15. The top row left: original face model with a file size of 4MB; the top right, with polynomial interpolation of degree 3 reducing the file size to 8KB. Bottom row left: polynomial degree 10 reducing the file size to 16KB; bottom right, degree 15 reducing to 25KB.

Table 5.1: Compression rates in percentage.

Degree	20	30	40	50	80
Rate	99.35	99.07	98.79	98.53	97.66

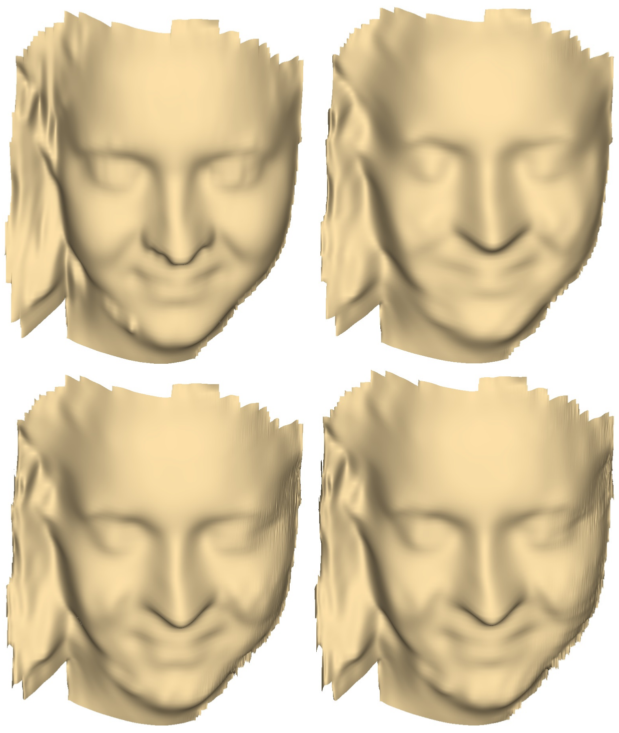

Figure 5.5: Polynomial interpolation degrees 20 to 40. Top row left: the original face model with a file size of 4MB; top right, with polynomial interpolation of degree 20 reducing the file size to 26KB. Bottom row left: polynomial degree 30 reducing the file size to 37.2KB; bottom right, degree 40 reducing to 48.5KB.

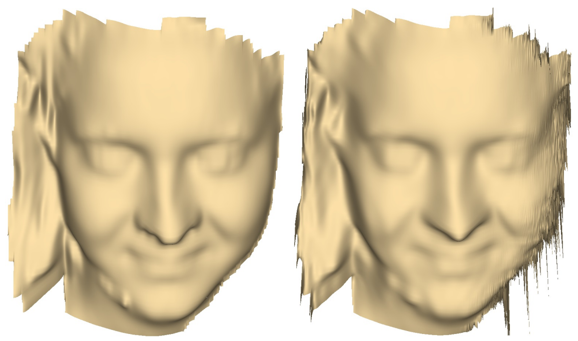

Figure 5.6: Left, the original face model; right, interpolation with polynomial degree 80. It is noted that the model becomes unstable.

Most mesh comparison techniques have been developed to compare a mesh before and after some process, and we wish to know how the process has affected the mesh. The trend observed with polynomial compression is clear. Regarding lower polynomial degrees such as degree 3, the compression rate is very high, but the reconstructed data are useless. As the degree increases, the reconstruction becomes increasingly better, but there is a break point in which the data becomes unstable for very high degrees. This is observed in the Figure 5.6 which shows the original face model on the left together with the reconstructed one with a polynomial of degree 80. It is demonstrated that for the kind of 3D used here, polynomial compression obviously has an optimal point and this seems to be around degree 30. Concerning compression rates, the technique is very efficient as for a polynomial interpolation of degrees 20, for instance, the file size in OBJ format has been reduced from 4MB to 26KB. This is a reduction of 99.35%, and similar reductions were achieved for other polynomials, also; a summary is presented in Table 5.1 showing various compression rates including around the optimal point. To compress data using polynomials of any degree the Matlab function `'gmprCompressPolynomials.m'` has been developed and to uncompress the corresponding function `'gmprUncompressPolynomials.m'` must be used. See Appendix B for page (177-181) details of the functions.

5.3.3 Evaluating the Fit

Determining the quality of a polynomial regression or how well the recovered data points fit the original data can involve a number of tests including statistical summaries. By far the most meaningful way is by plotting the original and regression data, sets, and visually assessing the quality. By visually analysing the models of Figure 5.5, it is suggested that a polynomial interpolation of degrees 20 to 40 describes the data well and can be well suited to most applications.

Another way of assessing quality is to look at the residuals and plot them against predicted values. Figure 5.7 shows the plot for data interpolated with a polynomial of degree 30. For a good fit, the plot should display no patterns and no trends. The scatter plot shows what looks like random noise, which is a good measure of the quality of the fit. Alternatively, if the fit is good, a normal- probability plot of the residuals should display a straight line. The plot depicted in Figure 5.8 shows that

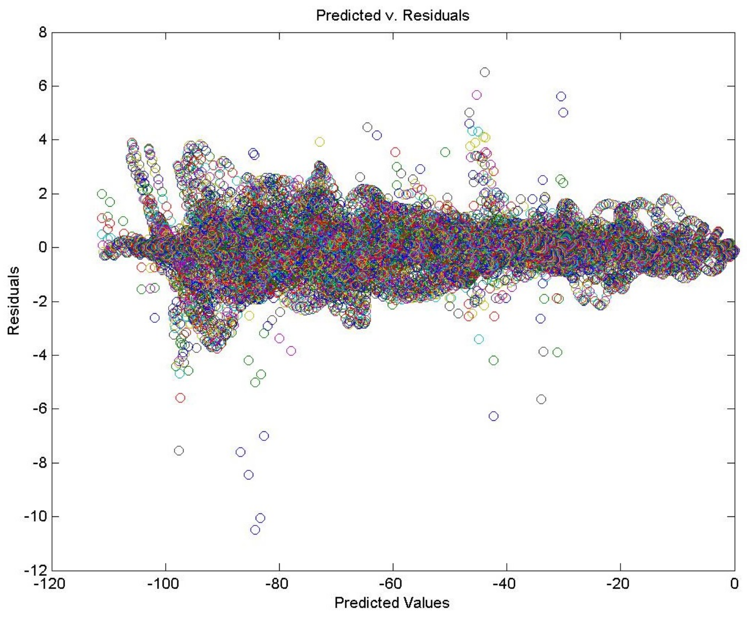

Figure 5.7: Scatter plot of Predicted Values against Residuals. For a good fit, it should show no patterns and no trends. The plot shows what looks like random noise, indicating a good fit.

for most polynomials evaluated at each plane, they do indeed describe a straight line, thereby indicating a good fit.

There are a number of other statistical measures to assess the quality or the appropriateness of a model such as the coefficient of determination, also known as R^2 , that indicates the percentage of the variation in the data that is explained by the model. This can be estimated by first calculating the deviation of the original data set which gives a measure of the spread. While the total variation to be accounted for (SST) is given by the sum of deviation squared, the variation that is *not* accounted for is the sum of the residuals squared (SSE).

$$R^2 = 1 - \frac{SSE}{SST} \tag{5.7}$$

The R^2 values for some interpolated models are described in Table 5.2. The table shows a trend of increasing R^2 as the polynomial degree increases, peaking at around degree 30, which indicates that this is the optimal interpolation point

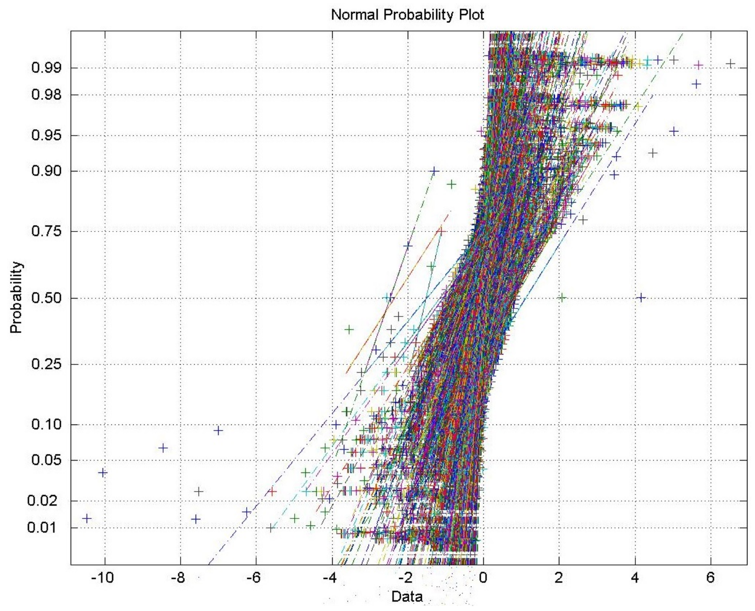

Figure 5.8: The normal-probability plot of the residuals. A good fit should describe a straight line for each polynomial curve, which is verified by the plot, indicating a good fit.

in the data set. For higher degrees, R^2 decreases monotonically, and this is also confirmed by visual inspection of the 3D reconstructed models whose quality deteriorates as they become unstable for high degree polynomials.

Table 5.2: The coefficients of determination R^2 for polynomial fits of degrees 20 to 80 for the given data.

Degree	20	30	40	50	80
R^2	0.9995	0.9996	0.9995	0.9994	0.9909

5.4 Discussion

This chapter has presented and tested a new method for 3D data compression based on polynomial interpolation of various degrees. The new compression method is based on the parameterization of surface patches which was discussed and tested. While the (x,y) values of each vertex are readily determined on a regular grid, the actual z-values are interpolated using a high degree polynomial and also the results show compression rates of over 99%. While the technique has been demonstrated to be a viable method for 3D compression, there are issues of accuracy as shown by the R^2 coefficients. In addition, visual inspection may suggest that such a compression technique may be acceptable for a number of applications, but quality deficient for others, such as for 3D face recognition. Issues to consider when assessing whether or not the technique is appropriate include the required polynomial degree which is dependent on the characteristics of the data, and the fact that for very high degrees the data becomes unstable, as demonstrated here. Therefore, iterative techniques will be considered in this research. In the next Chapter, a method with regard to Fourier-based data compression as well as PDE-based data uncompressing will be introduced.

Chapter 6

Partial Differential Equations for 3D Data Compression and Reconstruction

6.1 Introduction

It has been discussed in previous Chapters that a surface patch can be described as either a point cloud structure or a triangulated mesh. The re-meshing technique described in Chapter 4 will impose a structure on the data from which the connectivity of the mesh can be readily derived. From the structured data, a triangulated surface or an implicit representation, such as a level set function, can be constructed to approximate the point cloud data. Based on these representations, PDE-based techniques and variational techniques provide highly effective tools to draw out implicit geometrical data either locally or globally.

As an alternative to compression, using polynomials, it is now intended to create PDE meshes with high vertex density and to compare the compression efficiency of the resulting data with the original data. In this work face models from the GMPR 3D scanner were used, and a mesh with high vertex density was first constructed: this is called a "superfine mesh". A high-density mesh is necessary for specialised applications such as 3D face recognition in order to measure Euclidean distances on the face more accurately so as to produce the required accuracy and robustness in the face recognition algorithms [Rodrigues and Robinson, 2010, 2011].

It is clearly also important to ensure that when PDE data are reconstructed as the superfine mesh, and used in the face recognition process, there is no loss of accuracy in the reconstructed mesh. Therefore, two questions can be posed:

1. What compression rates can one obtain, using the PDE method?
2. How can one compare the accuracy of the reconstructed PDE, compared with the original superfine mesh?

This Chapter is organized as follows; Section 6.2 describes the compression, and reconstruction method, Section 6.3 presents experimental results, and Section 6.4 assesses the quality of the reconstructed mesh. Finally, a discussion is presented in Section 6.5.

6.2 Method

6.2.1 Data Preparation

The data preparation procedure for the PDE method is the same as for polynomial interpolation and has been described in Chapter 4. Given a (potentially dense) generic surface patch defined as a single-valued function, the first step is to perform a structured re-meshing aiming at reducing the vertex density. It is attained by simply finding the bounding box in 3D [Hill and Kelley, 2007] as described in Chapter 4 and using a number of horizontal and vertical cutting planes for vertex sampling. Each plane intersection defines a line and where this line intercepts the mesh defines a sampled vertex in the plane. All points lying in the plane, either horizontal or vertical, can be treated as a one-dimensional signal and subject to compression. The result of this procedure is that the mesh is redefined as aligned vertices in the horizontal and vertical directions as depicted in Chapter 4, Figure 4.10. It is important to stress here that such a re-meshing operation will yield a sparse mesh as it reduces the number of vertices in the original structure. Upon compression by the Fourier technique described below, it only becomes possible to reconstruct the sparse mesh. However, the objective is to recover the vertex density of the original superfine mesh; that is where the PDE technique comes into play. The number of required horizontal and vertical cutting planes depends on the mesh complexity.

An illustration of the steps in the method is shown in Figure 6.1. The original given data are defined as superfine mesh **A** with a high density of vertices. First, the cutting plane technique is used, in order to obtain a sparse mesh **B**. Second, the data are compressed by FFT obtaining a matrix **C**. Third data are uncompressed by the inverse FFT (iFFT) and this step will recover the sparse mesh on **D** which is equivalent to mesh **B**. Finally, to recover the original mesh density, a PDE reconstruction results in mesh **E**. Then one can compare the quality of reconstruction of **D** with **B** and **E** with **A**.

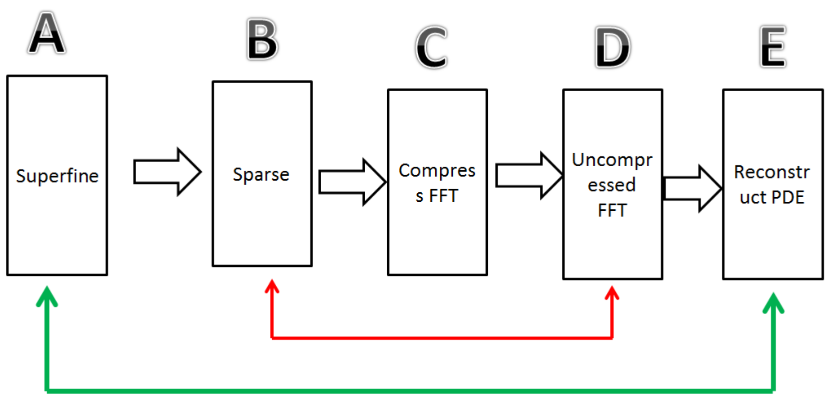

Figure 6.1: The illustration of PDEs compression and reconstruction.

6.2.2 Fourier Series Approximation

The usefulness of the Fourier analysis is that one can break up any arbitrary periodic function into a set of simple terms that can be solved individually and then recombined to reconstruct the original signal with a high degree of accuracy [Bernatz, 2010; Brown and Churchill, 2012*b*; Hanna and Rowland, 2008]. The Fourier transform defined as a limiting case of Fourier series is concerned with analysis of non-periodic phenomena, and are a tool which converts a spatial description of a signal into one in terms of its frequency components, and can be used to transform a periodic and non-periodic signal from time domain to frequency domain. The fast Fourier transform (FFT) is a mathematical tool by taking a time domain signal and turn into frequency domain data, so one can look at the frequency contents of the signal.

In FFT the frequency resolution of the data is inversely proportional to the side of the chunk of time it takes to compute, so the larger the FFT the finer the frequency resolution of the data. The FFT is an efficient algorithm for computing the discrete Fourier transform DFT and its inverse, which takes a regular spaced data values, and returns the value of the Fourier transform for a set of values in frequency space. Moreover, the FFT algorithm can decrease the processing time of a standard discrete Fourier transform from several minutes to a few milliseconds, since the FFT splits the calculation of the DFT into computing two DFT's of half the size.

DFT is, thus, a numerical approximation to the Fourier transforms, which is very useful for data compression, because a few coefficients of the Fourier expansion may be sufficient for the reconstructed signal to be close enough to the original function. Furthermore, DFT applies to uniform spaced data when used as a transform between time and frequency domains. The DFT loses all information considering the time scale, since the input is simply a vector of real or complex-valued samples.

Once the data are in the format specified in Section 6.2.1, typically the vertices lying in each plane can be considered as a one-dimensional signal and subject to compression by a Fourier series. Thus, the discrete Fourier coefficients are evaluated for each set of z-values in the plane; and the continuous functions are generally replaced by discrete functions. Therefore, in this Section we are trying to compress each z-curve in each plane through the use of Discrete Fourier Transform. While using the method of a generalised Fourier series in Section 3.3, often the Fourier series of a function $f(x)$ is given by:

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right)\right) \tag{6.1}$$

where

$$a_0 = \frac{2}{L}\int_0^L f(x)dx \tag{6.2}$$

$$a_n = \frac{2}{L}\int_0^L f(x)\cos\left(\frac{n\pi x}{L}\right)dx \tag{6.3}$$

$$b_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi x}{L}\right)dx \tag{6.4}$$

Thus,

$$\frac{1}{2}a_0^2 + \sum_{n=1}^{\infty}\left(a_n^2 + b_n^2\right) = \frac{1}{L}\int_0^L (f(x))^2 dx. \tag{6.5}$$

Eqs. 6.2, 6.3 and 6.4 are the Fourier coefficients of experimental data. Each signal describes a complex function in each plane with its own set of coefficients. An FFT algorithm requires there to be $n = 2^p$ mesh points in directions to be transformed where p is a non-negative integer. This is to split the sequence into two sequences of length $n/2$. Moreover, with regards to efficiency the DFT requires that the signal length be a power of 2 (Matlab pads with zeros when that is not the case).

The approach adopted in this thesis is to use FFT for estimating polynomial coefficients to interpolate a set of regularly spaced data, an approach that has been described in [Briggs and Henson, 1995]. This approach has been originally described by Gauss [Heideman et al., 1985] and [Cooley and Tukey, 1965]. The Matlab implementation uses a number of built-in functions and, by saving the coefficients (real and imaginary) together with the boundaries of each function and their scale, it is possible to reconstruct faithfully the original data defined by the

Fourier series. From Eqs. 6.1 the reconstructed signal would take the form:

$$y = a_0 + \sum_{n=1}^{N} \Big(a_n \cos(2\pi n x) + b_n \sin(2\pi n x) \Big) \tag{6.6}$$

where a_n is the vector of real coefficients, b_n is the vector of imaginary coefficients, n are the indices $[1, 2, ..., N]$ where N is the length of vectors a_n and b_n. Following the approach described in [Briggs and Henson, 1995], given a signal s of length m, the relevant coefficients in a equation 6.6 can be evaluated (for example, using Matlab built-in functions) as:

$$d = \mathrm{fft}(s) \tag{6.7}$$
$$M = \mathrm{floor}((m+1)/2) \tag{6.8}$$
$$a_0 = d_{(1)} m \tag{6.9}$$
$$a_n = 2 * \mathrm{real}(d_{(2.....M)})/m \tag{6.10}$$
$$a_6 = d_{(M+1)}/m \tag{6.11}$$
$$b_n = -2 * \mathrm{imag}(d_{(2.....M)})/m. \tag{6.12}$$

where d are the coefficients of the fast Fourier Transform for vertices lying in a single plane, m is the length of the signal or the length of the sequence of vertices in a cutting plane, a_0 is the DC component, a_n is the vector of real coefficients, b_n is the vector of imaginary coefficients, and a_6 is the residual error.

Table 6.1: Text file format for 3D compression using DFT

Line number	**ASCII data info**						
1	k_1	k_2	D_1	D_2	Q		
2	v_1	v_2	a_0	a_6	L	a_n	b_n
...	...						
N	v_1	v_2	a_0	a_6	L	a_n	b_n

The complex Fourier series and the sine-cosine series are identical, each representing the spectrum of a signal. The Fourier coefficients, a_n and b_n, express the real and imaginary parts respectively of the spectrum. The set of Fourier coefficients is estimated for each plane are saved in a plain ASCII format into a file with N lines of text where the first line contains a header, information followed by $(N-1)$ lines of data as defined in Table 6.1 where:

1	line 1 contains header info,
$2-N$	lines 2 to N contain data,
k_1, k_2	are the scale factors or distance between two consecutive horizontal and two consecutive vertical planes in mm,
D_1, D_2	are the dimensions of the data in the number of rows and columns,
v_1, v_2	are the first and last valid vertices for each row of data,
Q	the quality of the compression in percentage from 1 to 100,
a_0, a_6	are the scalar Fourier coefficients for each row of data,
L	the vector length of Fourier coefficients,
a_n, b_n	the vector real and imaginary Fourier coefficients for each row of data.

Note that compression of DFT coefficients of the sparse mesh only applies to the set of imaginary coefficients. By discarding a percentage from the end of the vector is a simple operation and such percentage can be attached to a notion of quality of compression which is a user defined.

6.2.3 PDE Modelling

The PDE method to be implemented is to solve Laplace's equation defined in the section 3.4.2 over the boundaries defined through the cutting planes. Since the cutting planes are defined on a regular grid and thus all vertices in one boundary plane could be paired to their corresponding vertices in the opposite boundary plane, the problem is then defined as the interpolation of any desired number of vertices between each pair of vertices. In order to solve Laplace's equation over such domain, the method of lines is an appropriate technique to use, by replacing Laplace's equation with the algebraic approximation of ODEs.

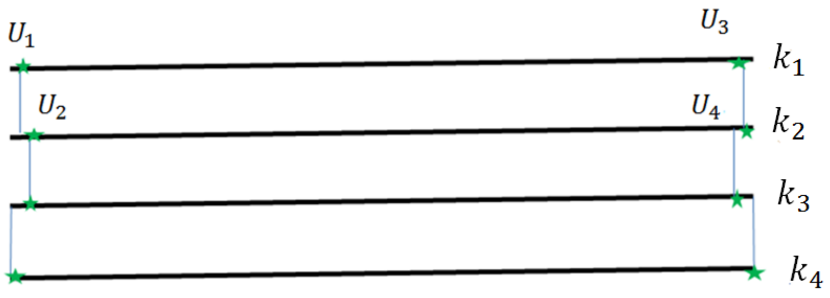

Figure 6.2: Rectangular domain for solving Laplace's equation.

To be able to implement this method in the 3D data model by solving Laplace's equation over a rectangular domain by taking between two consecutive cutting plane as shown in Figure 6.2 (only four planes are shown). The boundaries of the first plane k_1 and the second plane k_2 are defined as follows

- U_1, U_3 are the first and last valid vertices of k_1
- U_2, U_4 are the first and last valid vertices of k_2

Each set of structured vertices lying in the plane can be treated as a one-dimensional signal with a constant step where each value represents the depth z of the data. We assume that the independent variable domain in Figure 6.2 will be divided into an equal sized grid, and taking the four corners as a local Dirichlet boundary condition, applying the method in Section 3.4.2.

That is obtained by linear interpolation; the data in each plane will contain the valid vertex with the length of the plane. Subsequently define the four boundary conditions between two planes and the number of data points in each plane. In addition, each two planes define a rectangular domain with R rows by C columns, by setting the vertical grid spacing between planes and set the horizontal grid spacing along the planes.

To illustrate PDE interpolation between any two planes (these are the top and bottom boundary conditions in the rectangular domain) assume that we wish to interpolate N points between two given planes. First a matrix M of dimension $R \times C$ is defined where $R = N + 2$. The top row of M is initialised with the values of the first plane, and the bottom row of M is initialised with the values of the second plane. The left and right boundaries are solved by the finite difference method by taking the first valid vertex in each plane and, from the number of planes we wish to interpolate, find the discrete step in a straight line between those two vertices. A test is needed to make sure we are pairing the correct vertices by checking their indices: as in Figure 6.2 if the two first valid vertices are U_1 and U_2 in planes k_1 and k_2 respectively, and the two last valid vertices are U_3 and U_4 in planes k_1 and k_2, then the indices to the left and right boundaries of the rectangular domain are defined as:

$$U_{left} = (U_1, U_2)_{\max} \tag{6.13}$$

$$U_{right} = (U_3, U_4)_{\min} \tag{6.14}$$

and any valid indices outside these boundaries are ignored. Once the left and right indices are defined, fill in the missing left and right boundaries by linear interpolation. And all other values internal to M to be approximated by Laplace's equation are initialised to zero. This completes the definition of the rectangular domain. Next, Laplace's equation will be iterated a number of times until convergence is achieved. The number of iterations can be fixed (which is the case in the Matlab code presented in the appendices) or a threshold could be defined. In this case, if changes between two consecutive iterations is less than the set threshold the function would exit. The initialised or starting a point of consecutive domains for $N = 3$ covering the entire model is illustrated on the top left picture of Figure 6.3.

Setting the number of iterations to 10 provides some convergence as illustrated by the picture on the top right, but it is still in need of improvement. The bottom row shows 35 iteration steps (bottom left) and 70 iteration steps (bottom right). It is clear that convergence is good in the latter case. Because a cut off the threshold has not been implemented in the Matlab code in the Appendices, we decided to fix the number of iterations to 100 to guarantee good convergence for all models. The particular Matlab function that has been developed is `'gmprLaplace.m'`, please see Appendix B, page (239-243).

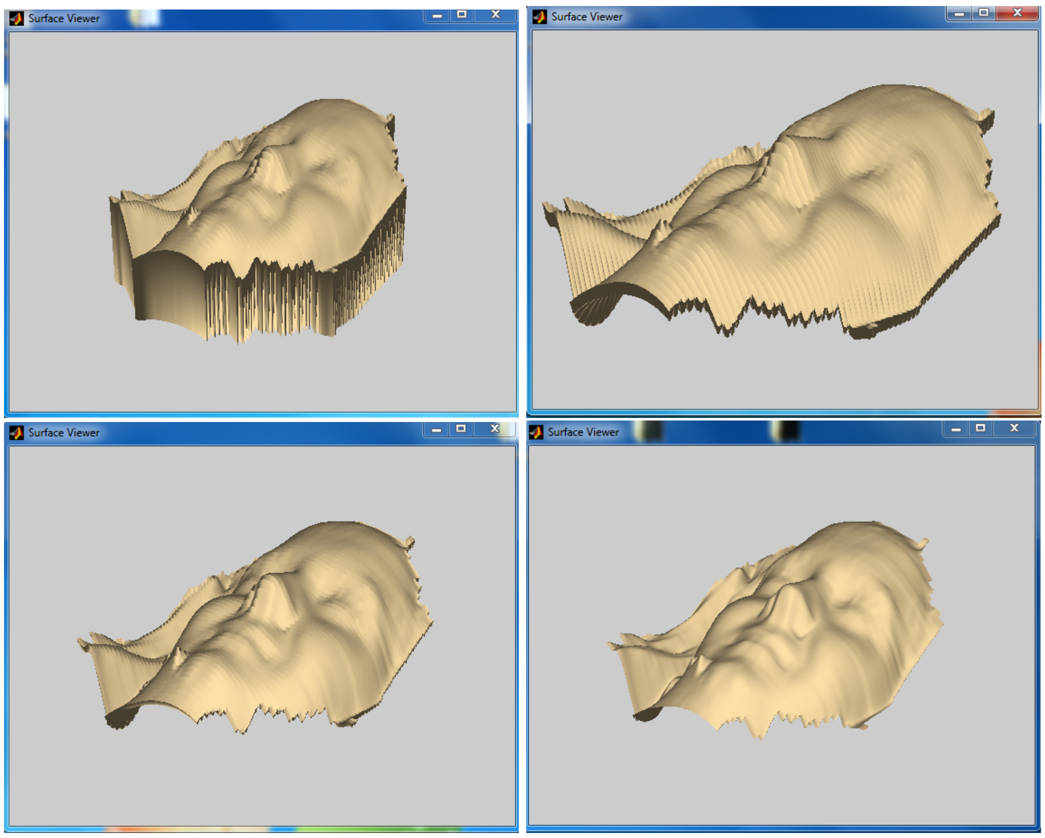

Figure 6.3: The effects of iteration steps on convergence

The particular PDE method generates surfaces from solutions to elliptic partial differential equations where boundary conditions are used to control surface shape. Moreover, the coefficients in the series can be computed by integration or approximate coefficients can be obtained using the FFT as described in Section 6.2.2.

Moreover, the approximation is made at discrete values of the independent variables and the approximation scheme is implemented via Matlab. The method of lines replaces all partial derivatives and other terms in the PDE by approximations. Here one can have Dirichlet, von Neumann or mixed boundary conditions specify the four boundary conditions of the rectangular domain defined in Section 3.2. When the value of the solution is given round the boundary of the region, then the boundary value problem is known as the Dirichlet problem, whereas when the normal derivative of the solution would be around the boundary, the problem is known as a von Neumann problem. In the experimental results described below, it is the Dirichlet boundary conditions that are used, by fixing the value of the vertices in the boundaries of the rectangular domain.

6.3 Experimental Results

All data used in the experiments highlighted in this Section are superfine models. The high-level steps are described in Figure 6.1: impose a structure on the data by the re-meshing technique resulting in a sparse mesh. The sparse mesh is then subject to compression by DFT. On the decompression stage, the inverse DFT is performed and the sparse mesh is recovered. In order to recover the original superfine mesh, Laplace's equation is solved by the PDE method. First, the Fourier coefficients are determined through equations 6.6 for each plane using discrete versions 6.7 -6.12. The sets of Fourier coefficients are saved in a plain text format into a file whose structure is defined in Table 6.1.

The processing of the above data and the 3D reconstruction involves solving the PDE as described in Section 6.2.3 between two consecutive cutting planes $S1$ and $S2$. Concerning specific programming procedures that have been developed to solve the PDE surface in a robust way, the following observations are made. Each plane contains a number of vertices; some are valid while some are invalid. Only the valid vertices from one plane are paired to their valid counterparts with the same index on the other plane (since the cutting planes are defined on a regular grid). The PDE surface is solved for each pair of vertices in turn.

Thus, the PDE boundary conditions are set between the two planes; we experimented interpolation with 1,3,5 and 10 planes by using the finite difference method at the boundary, and all values to be calculated by Laplace's are initialised to zero. As it has been illustrated in Section 6.2.3 the Laplace's equation requires a good number of iterations for good convergence, and this has been set to 100 to guarantee good results for all models used. This can be optimised by changing the Matlab code and defining a threshold to exit the iteration loop.

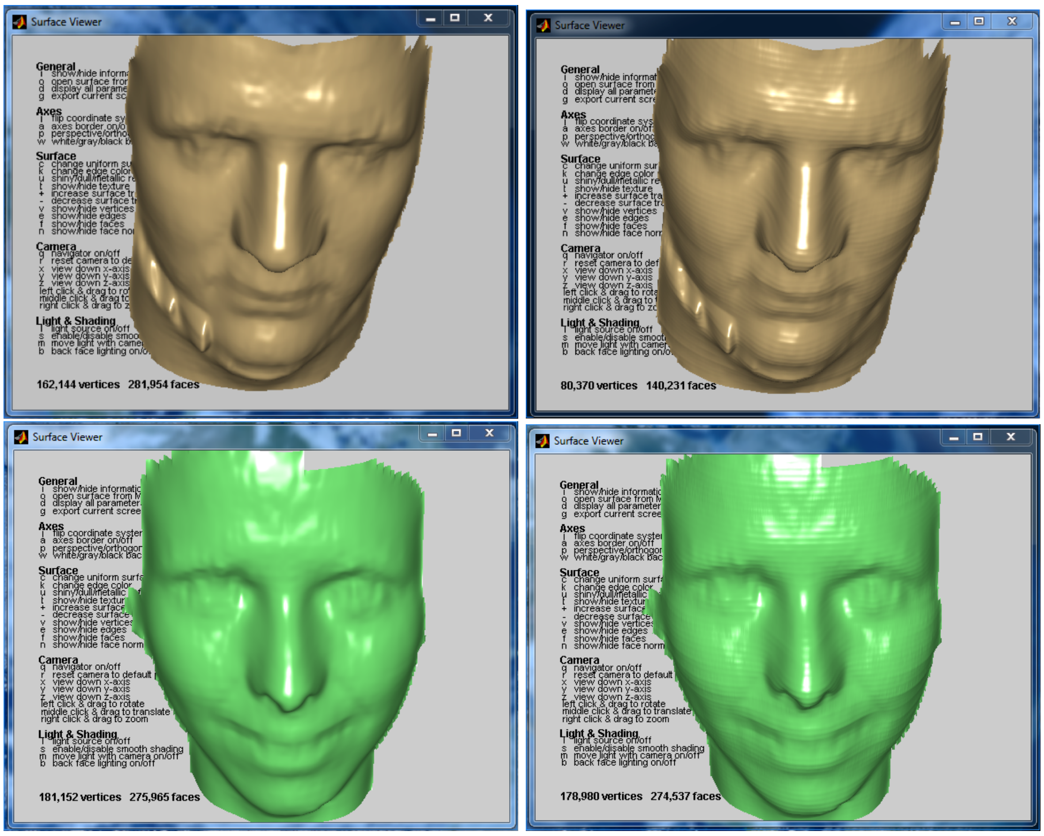
Surface Viewer
162,144 vertices 281,954 faces
80,370 vertices 140,231 faces
181,152 vertices 275,965 faces
178,980 vertices 274,537 faces

Figure 6.4: Left: original superfine meshes; right: PDE reconstructed

Typical results from the approach highlighted in this thesis are illustrated in Section 6.2.3. Further results from the technique being used in the Chapter is illustrated in Figure 6.4 where the left column depicts the original superfine meshes with 162K and 181K vertices. Each of these files saved as a standard OBJ file format takes around 20MB of disk space. Both meshes were subject to the same re-meshing operation, compression via DFT coefficients, load and reconstruction procedures. Here a detailed account is given of the top mesh: first the mesh was cut up into horizontal planes 3.3 mm apart and vertical planes 0.5mm apart; this resulted in 72 horizontal planes on each mesh and 563 vertical planes. Fourier coefficients were estimated from the z-values of each of the 72 planes and saved in the prescribed format. These operations, reduced the file size from 20M down to 668KB, a reduction of over 96.6% (if the file were zipped then the final size would only be 111KB, a reduction of over 99.4%). The pictures on the right column show the reconstructed meshes using the PDE method as described above. Due to the Nyquist sampling theorem, the reconstructed meshes are half the size of the original mesh, that is, the number of vertices along each cutting plane are half their original numbers; this is shown on the mesh on the top right of Figure 6.4. On the bottom right, the number of interpolated planes were adjusted to recover the original mesh density. It can be clearly seen that PDE reconstruction of compressed files as defined in this thesis does preserve the quality of the mesh.

For compression by FFT, `'gmprCompressFFT.m'` was used see page(181-184), to uncompress FFT `'gmprUnCompressFFT.m'` was used see page(185-188), and for Laplace's equation the Matlab code `'gmprLaplace.m'` page (239-243) was used (see Appendix B).

6.4 Assessing the Quality of 3D Reconstruction

The computing time to both compress and uncompress 3D data might be critical to some applications. A comparison of processing times is deferred until the next chapter, which provides a comparative analysis of DFT with Discrete Cosine, and Discrete Wavelet Transforms in connection with PDE reconstruction. Here, quality assessment focuses on measures to determine the accuracy and the goodness of fit or how well the 3D reconstructed data points fit the original data. Furthermore, most 3D data comparison techniques have been developed to compare a mesh before and after the process, and the aim is to know how the process has affected the 3D data. For example, will the compression techniques change the 3D data? Techniques implicitly assume that the two meshes are the same size (the same number of pixels for images and, for the case presented here, the same number of vertices) and that they are perfectly aligned. Thus, if we subtract the original mesh from the other and the result is everywhere zero, they are identical and the process preserved the 3D data perfectly. In general, though, the difference is not zero. There is therefore a desire to know how much difference there is between the two meshes in this case, and 3D data comparison methods provide different ways to answer that question. The assessment described below is the same as the method described in Chapter 5:

1. Visual assessment of the data and residuals.
2. Residuals plotted against predicted values.
3. A normal-probability plot of the residuals.
4. The coefficient of determination R^2.

The visual assessment of the quality can be inferred from examples in Figure 6.4. Visual inspection suggests that there is a perceived good fit between the PDE reconstructed data and the original data sets. However, extracting quantitative data allows a more objective comparison of the goodness of fit to be made. By subtracting the PDE reconstructed from the original mesh, one would expect that, if the two meshes were exactly the same, then the difference would describe a zero-plane at origin with normal $(0,0,1)^T$, as all vertex differences would be zero. Figure 6.5 left shows such a difference surface with vertex values oscillating around zero. Although there are small errors across the surface, especially around the nose area and on the boundaries of the mesh, such errors may not be significant enough to impair recognition algorithms. On the right of Figure 6.5 is shown a quantification of the error surface – essentially a view of the residuals across the *yz*-plane. Note that the nose region is at the center of the plot while the left and right regions of the plot correspond to the oscillations observed in the error surface. The majority of errors are within a range of ± 1mm with the largest error approaching 2.5mm at the boundaries.

Another way of assessing the quality of the reconstructed mesh is to look at the residuals and plot them against their predicted values. Figure 6.6 left depicts a scatter plot reconstructed against the original data. For a good fit, the plot should display no patterns and no trends, and this is verified in the plot, indicating a good measure of fit. Similarly, a normal-probability plot of the residuals should display

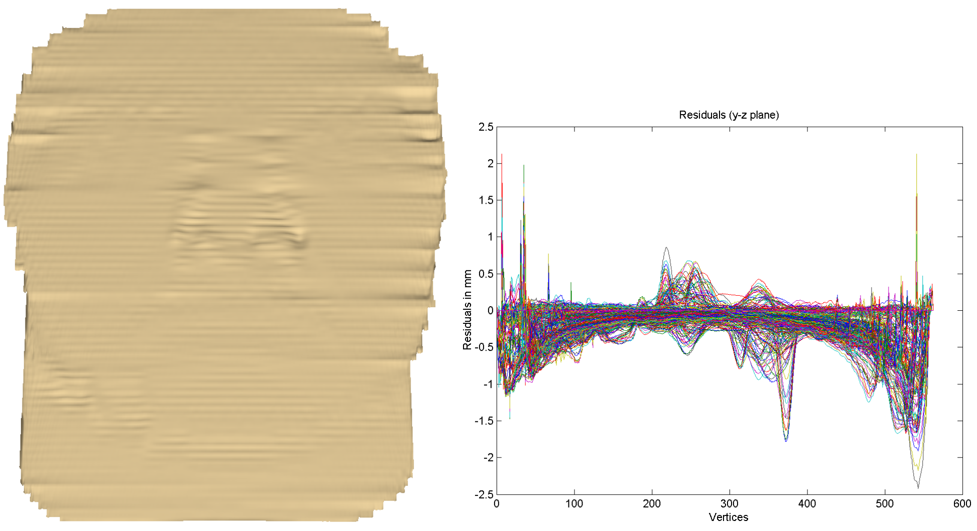

Figure 6.5: On the left, a visualisation of the error surface and, on the right the quantification of such errors in mm.

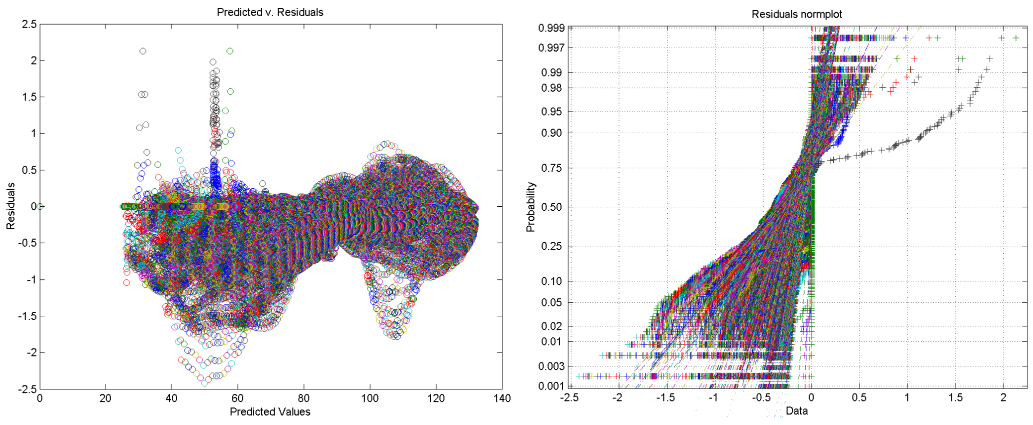

Figure 6.6: Left: Scatter plot of Predicted Values against Residuals (a good fit is indicated by no patterns and no trends). Right, The normal-probability plot of the residuals (a good fit is indicated by a straight line for each set of data)

a straight line for a good fit. On the right of Figure 6.6, it can be verified that most data sets evaluated at each plane are in straight lines, indicating a good fit.

The R^2 values for the PDE interpolated data are above 0.98 for all data sets described in this thesis (more details on the datasets used are described in Chapter 7). Again, this indicates a good measure of fit and suggests that the technique is appropriate for a wide range of applications.

6.5 Discussion

In this chapter the PDE method is exploited aiming at recovering the original density of unstructured superfine meshes. Initially, the original surface data are sparsely re-meshed by a number of cutting planes whose intersection points on the mesh tend to be represented by Fourier coefficients in each plane. The Fourier compressed data are then reconstructed to the sparse density and Laplace's equation is solved by the PDE method using each cutting plane as boundary conditions, thus recovering the superfine mesh density. Solving this system of ODEs yields a discrete solution along lines, which is why the method of lines is an appropriate technique. The derivations of such ODEs with a finite difference approximation of the spatial derivatives of the PDE are demonstrated. Additionally, the distinct approximation of a differential structure on the manifold symbolized by point clouds is based only on the neighbourhood approximation (by solving Laplace's equation over paired vertices by the method of lines) which is easy, effective and precise. This allows the extraction and recovery of the complete neighbourhood geometry. The method is highly efficient and allows high quality mesh compression over 96%. Comparing with the polynomial method of the previous Chapter in which compression rates are of the order of 99%, this is a somewhat less efficient compression. This is so because here it is necessary to keep both Fourier coefficients and the Fourier error vector (the imaginary components) while in the polynomial method only one set of coefficients are kept. The advantage, however, is that unlike the polynomial method, superfine meshes are recovered with good accuracy and there are no stability problems in the solution. In addition, in this thesis all patches being used are closed patches, and the method implemented within a closed patches, therefore the issue associated with smoothing in between boundaries does not arise. If two distinct surface patches are to be joined together (i.e. registered) then the issue would arise. However, this is not the case in this dissertation as we only deal with one patch at a time. It follows that there is no smoothing associated with the boundaries of the regions modelled by PDEs within any closed patch as it can be verified by the reconstructed models discussed in this

Chapter. Even though it is confirmed in this dissertation that Laplace's equation can certainly be used in this context, additionally it is also accepted that it is an efficient but rather a blunt tool.

In the next Chapter the use of DFT, DCT and DWT in connection with PDE reconstruction will be investigated and contrasted.

Chapter 7

3D Data Compression with Comparative Analysis via the Fourier Transform, Discrete Cosine Transform, Discrete Wavelet Transform and Partial Differential Equations

7.1 Introduction

This Chapter investigates alternative compression methods and the use of PDE surfaces for reconstruction of large data files. This is an extension of the work described in Chapters 5 and 6. The source data models are the same as previously described, which typically are surface patches defined as either a point cloud or a connected mesh of vertices with triangular faces. These are equivalent to standard data types in many 3D computer generated models, such as Wavefront OBJ and Java3D, VRML, and COLLADA formats [Drath et al., 2008; Hase, 1997; Rule, 1996]. As before, the methods proposed here rely on structured re-meshing of the surface by a polygon reduction resulting in an explicit structure of vertices.

The method of polygon reduction by such vertices is described in Section 4.5.1. Each set of vertices lying in the plane is subject to DFT, DCT, and DWT transform whose coefficients are then compressed using a quality factor as described in Sections 7.2.1, 7.2.2 and 7.2.3. The sets of coefficients are saved to ASCII files with specific structures that contain the necessary information to allow reconstruction using the inverse transforms of DFT, DCT and DWT padded with zeros where required. The issue of recovering the original mesh density (before polygon reduction) is addressed by defining the set of structured vertices as boundary conditions to elliptic PDEs described in Section 6.2.3. The PDEs are then iteratively solved through Laplace's equation.

The experimental data will be presented in Section 7.3, then the results using 86 high-density facial models are described in Section 7.4. Visualization of original and reconstructed models is provided under various quality parameters allowing a qualitative assessment of compression and reconstruction. In addition, error surfaces are estimated with corresponding root mean square errors (RMSE) for a more objective assessment of quality. Statistics are also presented for average compression rates for all models for quality parameters varying from 5 to 100. Finally, a discussion will be presented in Section 7.5.

7.2 Method

7.2.1 The DFT Method

Once experimental data are represented by the z-values of each structured plane as specified in Section 4.5.1, the vertices lying in each plane are treated as a Fourier series. The usefulness of the Fourier analysis is that any arbitrary periodic function can be divided up into a set of easy conditions that can be set individually and then recombined to reconstruct the original signal to a high degree of accuracy. The continuous Fourier Transform is defined as specified in Section 6.2.2.

The file format for compressed DFT data has been defined in Table 6.1. The parameter Q is defined as the quality of the compression and is expressed as a percentage. It refers to the percentage of coefficients to keep and it is applied slightly differently for DFT, DCT and DWT. A compression of DFT coefficients only applies to the set of imaginary coefficients. Normally, the most imaginary coefficients for high frequency signals are zero or close to zero and the most significant ones are the first few. Therefore, the options faced here are either to force any value below a certain threshold to zero or simply discard a percentage from the end of the vector, which is the chosen option for its simplicity of operation. In this way, it is guaranteed to keep the most relevant ones even for low values of quality. A quality $Q = 100$ means do not discard any coefficient while $Q = 30$ means discard 70% of them from back to the front.

7.2.2 The DCT Method

The DCT transform and its variants have been used in a variety of contexts, most notably in image and video compression (for example: [Belkasim, 2011; Gharge and Krishnan, 2007; Kim and Shin, 2003]). DCT is a close relative to the DFT transform as it defines a sequence of data in terms of the sum of the cosine functions at different frequencies. It can be seen as the ‘real’ version of the DFT in which the basic vectors contain only co-sinusoidal patterns. While a DFT contains real and imaginary components, the DCT operates on data with even symmetry, which means that a DCT is equivalent to a DFT with about twice the length of the data. In practice, it would be equivalent to a DFT by doubling the sampling data and shifting the added data to the end of the signal. There are many variants of the DCT and the one that is used here is the unitary Discrete Cosine Transform as defined in Matlab [Briggs et al., 1995]. The DCT transform of one dimensional signal z representing the depths on each structured plane is expressed as:

$$y(k) = w(k) \sum_{n=1}^{N} z(n) \cos(\frac{\pi(2n-1)(k-1)}{2N}) \tag{7.1}$$

for $k = 1, 2, \ldots N$ where N is the length of the signal z, $y(k)$ are the DCT coefficients of z and $w(k)$ is a scale factor. For making DCT values orthogonal, we multiply the terms simply by scale factors

$$w(k) = \begin{cases} 1/\sqrt{N} & \text{for } k = 1, \\ \sqrt{2/N} & \text{for } 2 \leq k \leq N. \end{cases}$$

The built-in Matlab function *dct* used as $y = dct(z, n)$ truncates z to a length n before transforming. The length of the coefficients y is the same as the original signal z. The advantage here is that only a few coefficients are necessary in order to reconstruct the signal.

The majority of signals can be reconstructed with more than 99% accuracy by using just a handful of coefficients. The inverse cosine transform recovers the initial signal from the set of coefficients $y(k)$:

$$z(n) = \sum_{k=1}^{N} w(k) y(k) \cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right) \tag{7.2}$$

for $n = 1, 2, \ldots N$ where N is the length of the coefficients in Eq. 7.1.

Table 7.1: Text file format for 3D compression using DCT

Line number	**ASCII data info**				
1	k_1	k_2	D_1	D_2	Q
2	v_1	v_2	B		
...	...				
N	v_1	v_2	B		

where k_1 is the number of horizontal planes, k_2 is the number of vertical planes, D_1 is the distance between each horizontal plane, D_2 is the distance between each vertical plane, v_1, v_2 are the first and last valid vertices for each row of data, and B are the DCT coefficients of each row of data (from each cutting plane). The DCT thus, is applied to each row of data and the first and last valid vertices together with coefficients B are appended to file from line 2. The parameters depicted in Table 7.1 are saved in plain ASCII format. Note that B is the set of DCT coefficients estimated by Equation 7.1 and shortened by parameter Q. In other words, the number of coefficients (the length of vector B) to be saved is defined by the floor of $(kQ/100)$. For compression by DCT the function $'$`gmprCompressDCT.m`$'$ is used, see page(188-190), and to uncompress DCT the function $'$`gmprUnCompressDCT.m`$'$ is used page(191-193). (See Appendix B for details of the functions).

7.2.3 The DWT Method

The DWT transform [Nicholl et al., 2010; Talukder and Harada, 2011; Vonesch et al., 2007; Wali et al., 2012] is a time-scale representation of a signal obtained using digital filtering techniques where the signal to become analysed is authorized via filtration along with various cut-off wavelengths at different gadgets. The technique is realised by iteration and the resolution of the signal, which usually decides the amount of details from the signal, can be controlled by sub-sampling (up and down) operations. For a given signal, two sets of coefficients are computed referred to as the *approximation coefficients A* and *detail coefficients D*. The A coefficients are obtained by convolving the signal with a low-pass filter and the D ones are obtained by convolving with a high-pass filter.

As the signal is decomposed by the half band filters, these results in signals spanning only half the frequency bands. This doubling of frequency resolution reduces uncertainty in frequency by half. Following the Nyquist's rule, the signal can now be down-sampled by removing 50 percent the examples with no lack of information. The outcome is that while the 50 percent group, low pass filtering removes half the frequencies, thus halving the resolution, a decimation by 2 halves the time resolution and thus doubles the scale.

Convolving the signal $z(n)$ with a half band digital low pass filter with reaction response $h(n)$ can be defined in discrete time as:

$$x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{7.3}$$

Applying the Nyquist rule by sub-sampling the signal by 2 can be represented as

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(2n-k) \tag{7.4}$$

Eqs. 7.4 is used for both high pass and low pass filtering operations. That one degree decomposition is usually indicated seeing that:

$$y_{high} = \sum_{n} x(n)g(2k-n) \tag{7.5}$$

$$y_{low} = \sum_{n} x(n)h(2k-n) \tag{7.6}$$

where y_{high} and y_{low} are the outputs of high and low pass filters after decimation by 2. In order to reconstruct the original signal, the procedure is straightforward given that half-band filters form orthonormal bases. At every level of decomposition the signal is up-sampled by two, filtered through a high pass and low pass synthesis filters $g'(n)$ and $h'(n)$ and then summed over. Thus, for every level of decomposition the recovered signal is represented as:

$$x(n) = \sum_{k=-\infty}^{\infty} \left(y_{high}(k).g(-n+2k)\right)\left(y_{low}(k).h(-n+2k)\right) \tag{7.7}$$

It is important to note that if the filters are not an ideal half band, then perfect reconstruction is not possible. While it is clear that ideal filters are not possible to realise, some filters under some conditions can provide perfect reconstruction. The most used and most accurate ones are the Daubechie's filters, also known as Daubechies wavelets [Vonesch et al., 2007] and these are the ones used in the experimental results described in the next section. Furthermore, in order to save the DWT coefficients to a text file for subsequent reconstruction it is necessary to decide on the number of levels of decomposition. This thesis is set for 3 levels as no significant gain is achieved with further levels for tested facial data.

The parameters $k_1, k_2, v_1, v_2, D_1, D_2$ and Q in Table 7.2 are defined in Section 7.2.1 and the introduced parameters are related to the approximation and detail coeffi-

Table 7.2: Text file format for 3D compression using DWT

Line number	**ASCII data info**							
1	k_1	k_2	D_1	D_2	Q			
2	v_1	v_2	L_1	L_2	L_3	L_4	L_5	C
...	...							
N	v_1	v_2	L_1	L_2	L_3	L_4	L_5	C

cients for a 3-level decomposition as follows:

L_1	is the length of approximation coefficients level 3 ($A3$),
L_2	is the length of detail coefficient level 3 ($D3$),
L_3	is the length of detail coefficient level 2 ($D2$),
L_4	is the length of detail coefficient level 1 ($D1$),
L_5	is the length of the original signal,
C	is the vector of coefficients to save.

Note that the number n of coefficients to save depends on the quality factor and it is defined as the floor of (length(C) $Q/100$) as before. For any value of quality $Q < 100$ implies discarding some of the detail coefficients $D1$, $D2$, and $D3$ in that order. Upon reconstruction, these are padded with zeros. The algorithm for discarding coefficients is as follows:

1. Estimate ($d = (\text{length}\,C) - n$) as the number of detail coefficients to discard,
2. if $d > L_4 + L_3$ discards all from $D1$ and $D2$ plus some or all from $D3$,
3. if $d > L_4$ discards all from $D1$ plus some or all from $D2$,
4. if $d \leq L_4$ discard some or all from $D1$.

Note that the approximation coefficient level, 3 are not subject to compression. If they were, the quality of the reconstructed data is largely deteriorated. The method proposed here uses a 3-level decomposition; if further levels are required, then the saved data and the algorithm above the need to be adjusted accordingly. For compression by DWT, the function `'gmprCompressDWT.m'` is used, see page(193-199), and to uncompress `'gmprUnCompressDWT.m'` is used see page(199-203). (See Appendix B for details of the functions).

7.3 Experimental Data

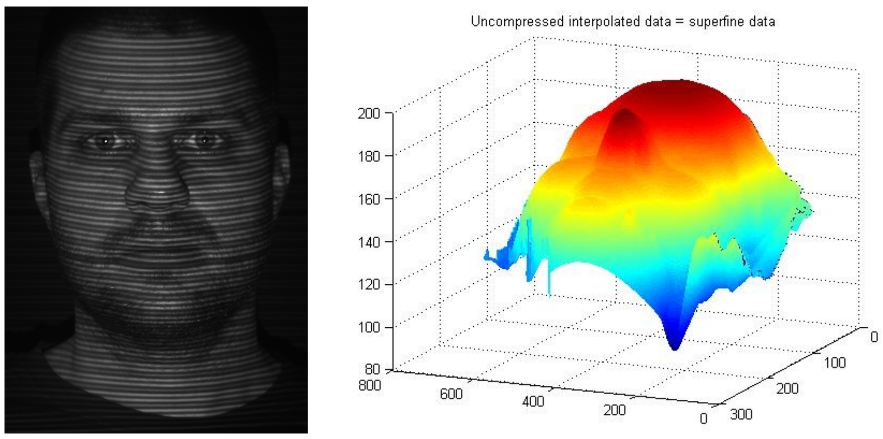

Figure 7.1: On the left is the original data image, on the right a perspective colour map of the data. The colours merely the threshold the data, helping to visualize areas with similar values. In this image, signal values range between 0–200mm.

In Figure 7.1, a representative image of a 3D model is shown as a function of two independent spatial aspects. The colour map used (JET colour map in Matlab) represents the depth values or z-coordinates of each vertex in space. A set of 86 data files are used in this dissertation, and some visualisation examples are depicted in Tables 7.3–7.5. Specific information on all files are summarised in Tables 7.6–7.8. It is noted that the size of the files varies widely from 24,450 to 103,680 vertices, and from 37,685 to 111,926 faces. Upon re-meshing by structured Planes, the number of such planes, both horizontal and vertical also varies widely; the largest file contains 1,296 vertical planes while the smallest has 431. The images depicted in Table 7.3, 7.4 and 7.5 were generated by the function `'gmprLoadData.m'` see page(203-205). See Appendix B for details of the function.

Table 7.3: Examples of data files used in this thesis

Table 7.4: Examples of data files used in this thesis

Table 7.5: Examples of data files used in this thesis

Table 7.6: Load and display PDE data 1

File number	Sampling cutting planes	Vertices per cutting plane	Number of vertices	Number of faces
1	71	709	50,339	76,922
2	65	588	38,220	60,655
3	77	764	58,828	88,552
4	65	781	50,765	77,332
5	62	818	50,716	50,382
6	71	563	39,973	69,330
7	71	586	41,606	67,458
8	50	554	27,700	45,048
9	56	547	30,632	47,287
10	77	827	63,679	94,490
11	74	827	58,312	69,578
12	74	599	44,326	73,029
13	77	803	61,831	91,114
14	65	758	49,270	58,743
15	71	759	53,889	77,547
16	50	693	43,650	44,668
17	65	643	41,795	64,750
18	50	489	24,450	37,685
19	71	572	40,612	65,864
20	65	570	37,050	60,997
21	74	550	40,700	64,915
22	53	535	28,355	44,802
23	71	431	30,601	46,886
24	53	516	27,348	45,759
25	50	499	24,950	39,179
26	65	541	35,165	52,243
27	53	481	25,493	40,724
28	71	1148	81,508	81,065
29	65	553	35,945	50,272
30	67	589	39,463	55,972
31	71	651	46,221	72,240
32	56	468	26,208	40,619
33	56	501	28,056	43,213
34	71	601	42,671	70,234
35	65	651	42,315	68,253

Table 7.7: Load and display PDE data 2

File number	Sampling cutting planes	Vertices per cutting plane	Number of vertices	Number of faces
36	62	556	34,472	55,480
37	59	569	33,571	54,187
38	62	577	35,774	58,644
39	65	524	34,060	51,644
40	65	489	31,785	55,384
41	65	547	35,555	56,250
42	53	535	28,355	44,802
43	53	535	28,355	44,802
44	73	697	50,881	76,986
45	71	991	70,361	78,476
46	74	1220	90,248	90,153
47	68	636	43,248	68,570
48	65	525	43,125	55,745
49	68	821	58,618	90,735
50	68	821	55,828	82,058
51	68	768	41,736	70,325
52	74	753	52,224	74,443
53	74	753	55,722	90,165
54	65	729	47,385	77,878
55	68	455	30,940	54,901
56	68	713	48,484	84,723
57	71	775	55,025	75,017
58	70	777	54,390	75,724
59	74	825	61,050	95,378
60	71	648	46,008	70,276
61	68	744	50,592	78,423
62	65	686	44,590	66,543
63	80	1296	103,680	111,926
64	53	518	27,454	43,270
65	71	722	51,262	72,172
66	71	905	64,255	79,747
67	73	1019	74,387	88,541
68	68	675	45,900	73,377
69	77	732	56,364	89,036

Table 7.8: Load and display PDE data 3

File number	Sampling cutting planes	Vertices per cutting plane	Number of vertices	Number of faces
70	68	586	39,848	64,052
71	65	595	38,675	64,979
72	74	582	43,068	76,752
73	62	591	36,642	55,931
74	65	652	42,380	62,866
75	74	642	47,508	81,360
76	68	729	49,572	67,473
77	71	1052	74,692	82,469
78	68	546	37,128	60,045
79	74	581	42,994	71,233
80	71	934	66,314	84,918
81	77	750	57,750	89,105
82	56	604	33,824	53,931
83	68	888	60,384	65,732
84	71	635	45,085	68,089
85	71	721	51,191	75,735
86	74	711	52,614	78,624

7.4 Results

Three sets of experiments were carried out: 1) DFT, DCT and DWT applied to points lying in a single plane; 2) DFT, DCT and DWT on multiple planes; 3) DFT, DCT and DWT on multiple planes in connection with PDE reconstruction. The model depicted in Figure 7.2 is used for illustration purposes in all experiments, although the full set of 86 models were used whose statistics and error analysis are presented in the following sections.

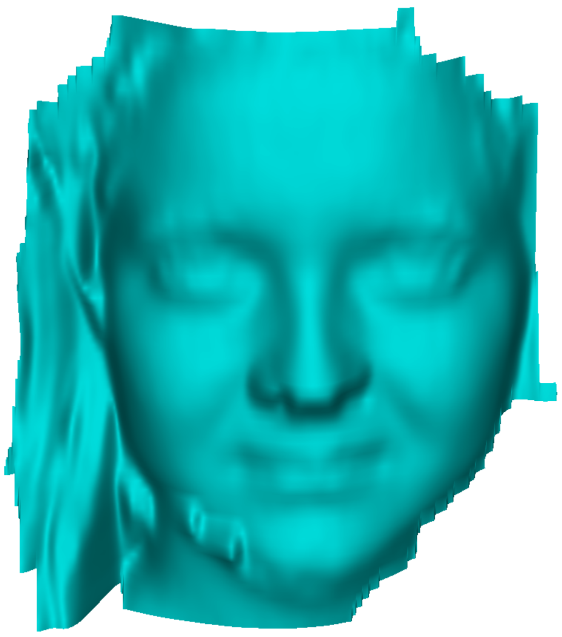

Figure 7.2: The 3D model used for illustration of compression techniques.

7.4.1 DFT, DCT and DWT Applied to Vertices Lying in a Single Plane

This first experiment is aimed at validating the approach and computer programs were applied to a set of vertices lying in a single plane. A plane across the model depicted in Figure 7.2 was selected that includes the tip of the nose, as this is a typical complex curve representative of the data set. Each of the techniques was

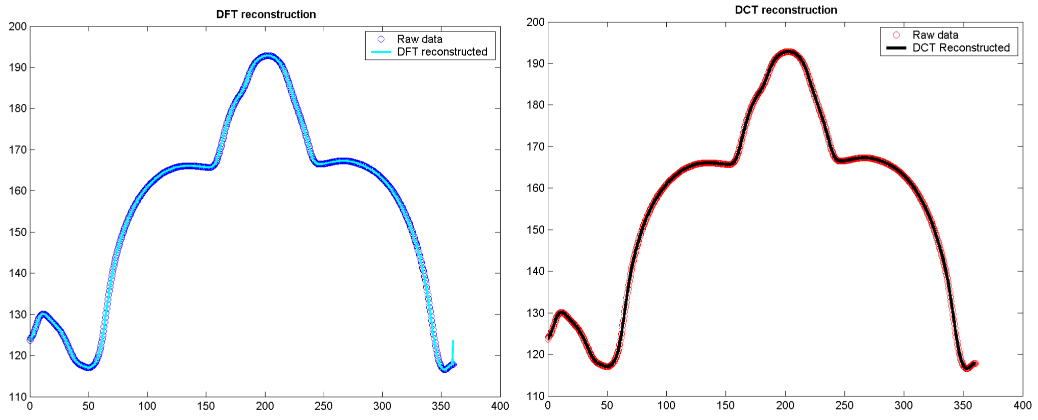

Figure 7.3: DFT (left) and DCT (right) reconstruction of vertices lying in a single plane.

applied in turn and results are presented in Figures 7.3 and 7.4 for DFT, DCT and DWT. No compression of coefficients was applied at this stage, which means the quality parameter was set to $Q = 100$. The first thing to note is that there seems to be no significant difference between the techniques, they are all capable of faithfully reconstructing the data and, in principle, any would be appropriate for the entire set of planes. The only aspect to take into consideration here is that the last point of the DFT transform is spurious as, due to periodicity, reconstruction is performed within the range $0.0,\ldots,2\pi$ forcing the last point to join up to the first data point, and it thus needs to be deleted from every reconstructed plane.

In order to explain exactly how compression would be applied to each of these curves in slightly different ways consider a quality parameter $Q = 50$. For the DFT curve, it would mean that the bottom half of the imaginary components would be discarded. Upon reconstruction with inverse DFT, the missing coefficients are padded with zeros. For DCT, it would simply mean that the bottom half of the coefficients in Equation 7.1 would be discarded and then padded with zeros upon reconstruction with the inverse DCT. For the DWT method, only the detail coefficients $D1$, $D2$ and $D3$ from the high pass filter are the ones subject to compression (Equation 7.5). In this case, 50% of all detail coefficients would be discarded starting from $D1$, then $D2$ then $D3$. Note that before discarding any of

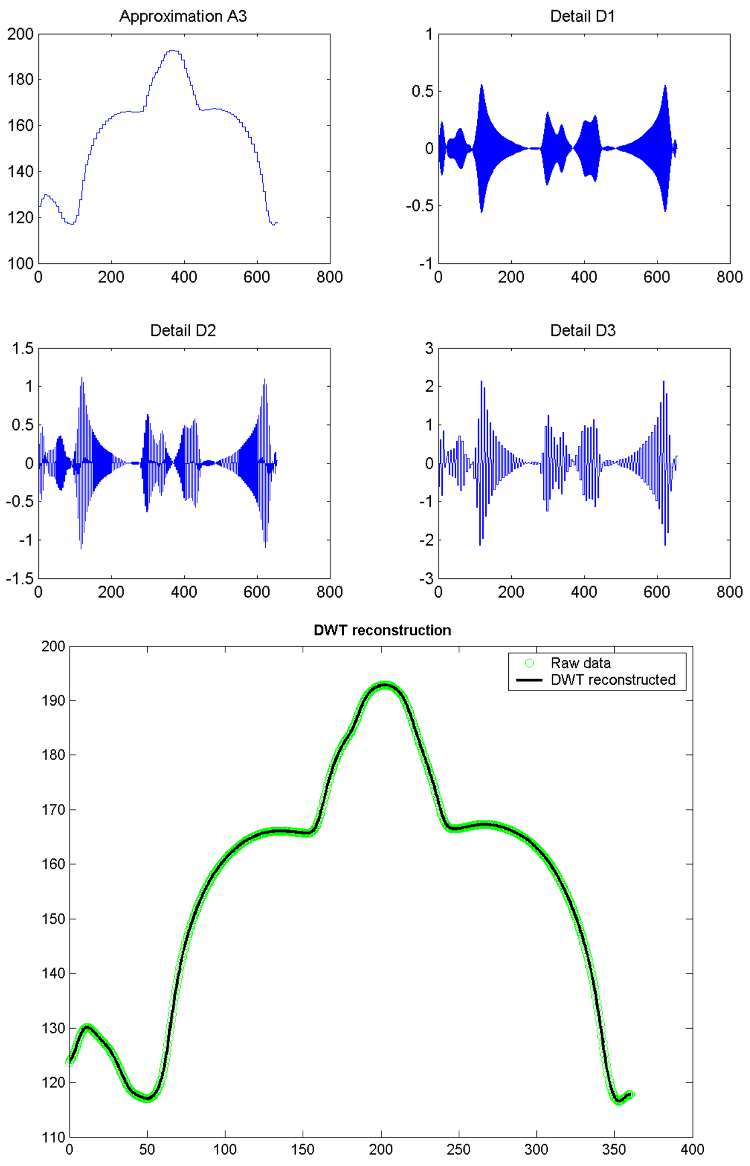

Figure 7.4: DWT 3-level decomposition and reconstruction of vertices in a single plane.

$D2$ it is necessary to discard *all* from $D1$. The same for $D3$ with regards to $D2$. On reconstruction using the inverse DWT all missing coefficients are padded with zeros. Observe that the approximation coefficients $A3$ are not subject to compression.

7.4.2 Extending DFT, DCT and DWT to Multiple Planes

In this section, the techniques are applied to a set of multiple planes describing the entire 3D model and illustrate the quality of perceived reconstructed models and error surfaces. These techniques use quality $Q = 100$ (no compression of coefficients) and $Q = 50$ (50% compression rate) for both qualitative and quantitative assessments of the resulting model. Two sets of experiments were performed, the first to test the effectiveness of the techniques applied to a sparse mesh with simple compression and decompression. The second experiment was identical, although with the objective of recovering the high density meshes using the PDE method. Note that in both cases the compressed model is the sparse mesh, except that the PDE method allows the recovery of the original mesh density; without PDE, only the sparse mesh could be recovered.

To apply the techniques to multiple planes, the first step is to perform a polygon reduction as defined in Section 4.5.1. Then the number of structured planes will define the density of the sparse mesh and this is a function of the distance between the planes, both horizontally and vertically (the distances D_1 and D_2 of Eqs. 4.4 and 4.5). For face models, it is sufficient to place each horizontal plane spaced around 3mm apart, and along each plane, as many data points as possible are desired in order to capture all the nuances of the face. The set density chosen for facial models is around 0.25mm between each data point, or 4 points per millimetre. These choices will reduce the number of vertices by a factor of around 4 in the original dense mesh of Figure 5.1. For each model tested, similar distances between planes were used. The actual number of planes in each model depends on the original extension of its bounding box.

Following compression, Figure 7.5 depicts the uncompressed models with quality $Q = 100$ and respective error surfaces (a numerical quantification of error surfaces is presented below). Assessing the overall appearance of the 3D models, it can be stated that any of the three techniques can successfully be used to compress and decompress 3D data. The bottom row of Figure 7.5 shows the error surfaces which were estimated by subtracting the reconstructed model from the sparse mesh.

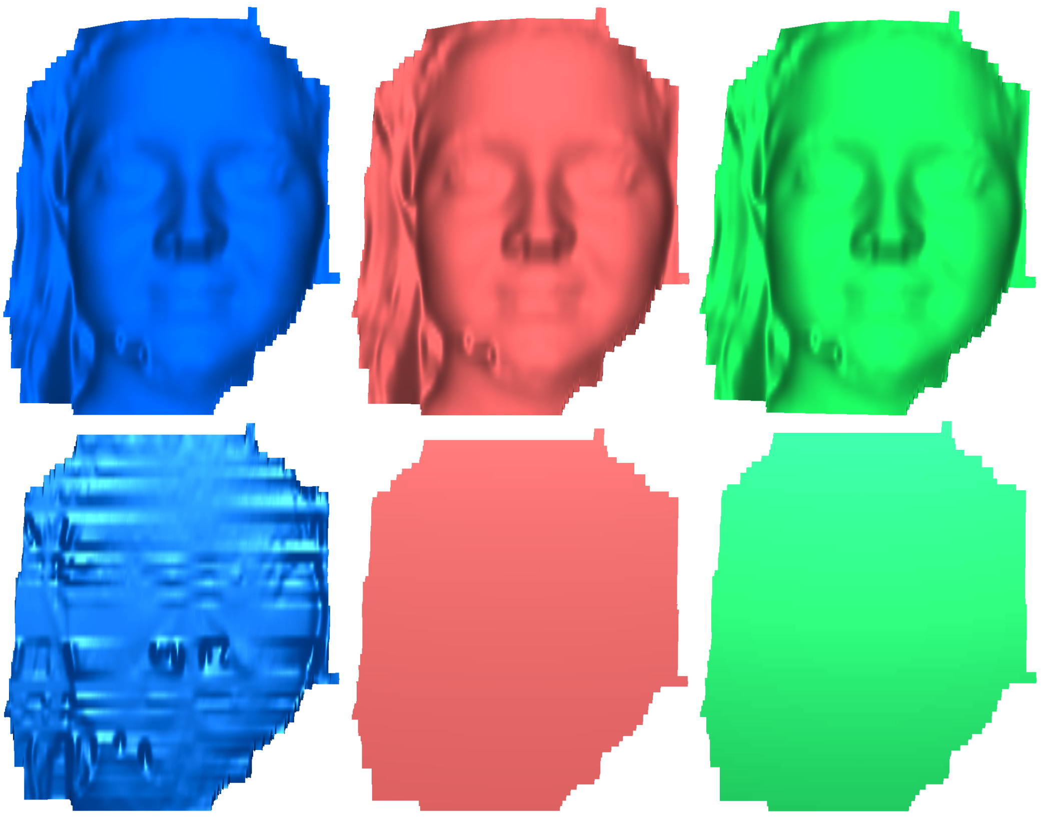

Figure 7.5: Quality $Q = 100$. Top row: reconstructed models DFT (blue) DCT (red) and DWT (green). Bottom row: respective error surfaces.

A perfect match would mean that the error surface would lie in the xy plane with all coordinates $z = 0$. The error surface of the DFT shows relatively large errors located on the more complex areas of the face such as around the nose and at the boundaries. In contrast, both DCT and DWT show the desired flat surface with errors at or near zero. On this basis, it is clear that DCT and DWT are better techniques and apparently equivalent for compression and decompression with a quality parameter of 100.

Figure 7.6 shows the results for compression using the quality parameter set to 50. Again, the DFT technique shows a relatively large error surface, pointing to the superiority of both the DCT and DWT techniques. The DCT transform shows small errors, mostly at the boundary of the model while DWT techniques have the error distributed along the surface, and it is noticeable that high frequency ripples

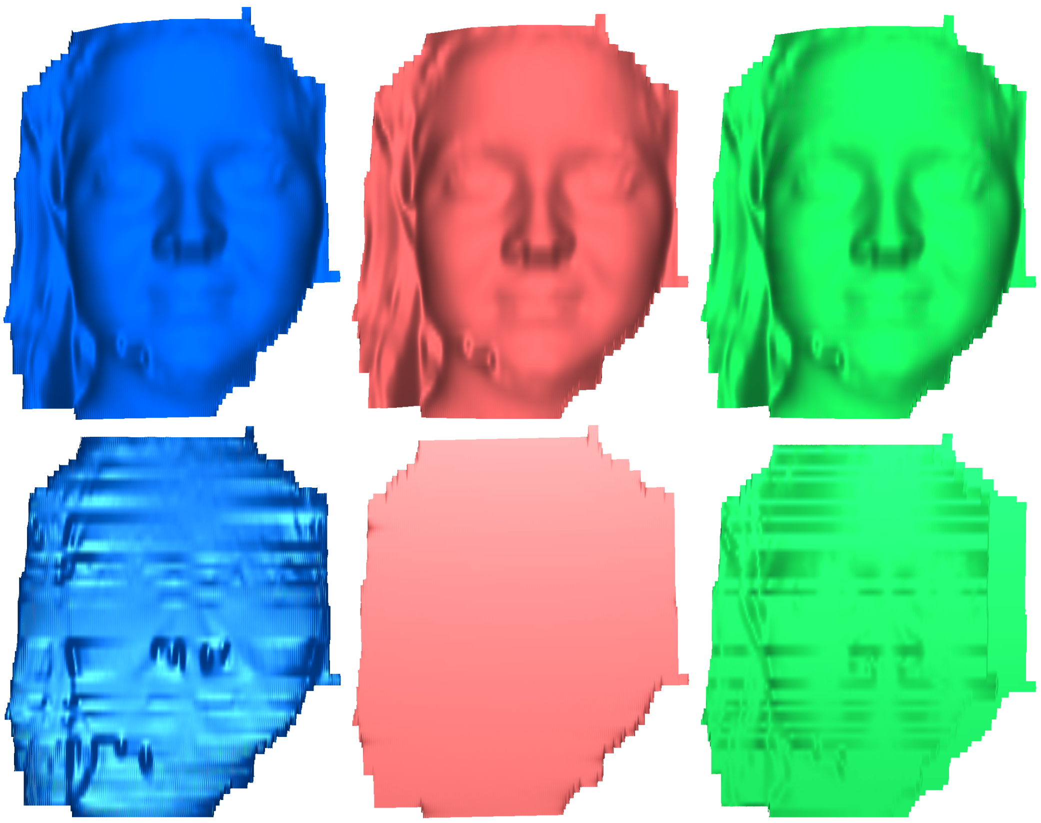

Figure 7.6: Quality $Q = 50$. Top row: reconstructed models DFT (blue) DCT (red) and DWT (green). Bottom row: respective error surfaces.

start to appear in the green model.

The issue now is to recover the original mesh density using the PDE method as described in Section 6.2.3. Each pair of structured planes was used as boundary conditions for an elliptic PDE and the distance D_1 was used to estimate the discrete step Δx between any two planes. Since the distance between the planes is 3mm, it is necessary to solve the Laplace equation using 5 steps (2 at the boundaries and 3 internal steps) resulting in a mesh density with an average quad face area of exactly 0.75×0.25mm – this is comparable to the original high density mesh where the average area of each quad face is 0.75×0.26mm.

Results using the PDE method are illustrated in Figures 7.7 and 7.8 for $Q = 100$ and $Q = 50$ respectively. The first point to note is that solving the Laplace equation

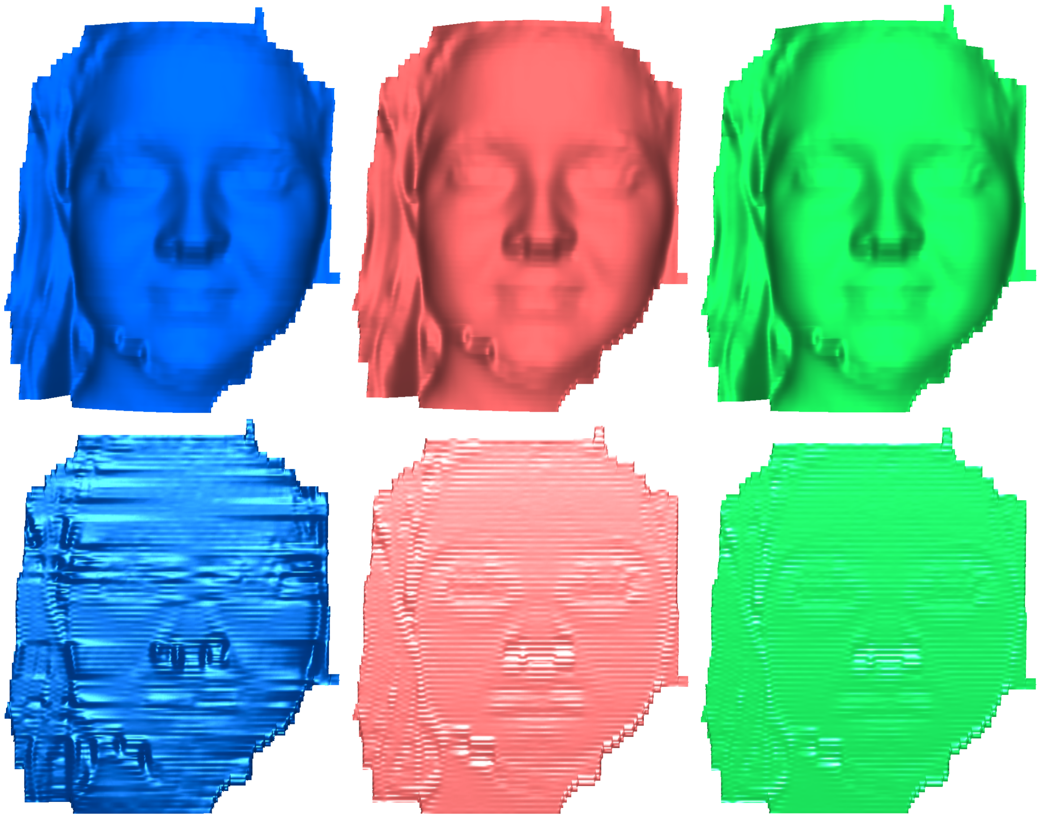
Figure 7.7: Quality $Q = 100$ with PDE based reconstruction. Top row: DFT (blue) DCT (red) and DWT (green). Bottom row: respective error surfaces.

over the mesh creates a higher level of noise than for the sparse mesh shown earlier. While on the 3D models this is not readily apparent, the error surfaces nevertheless point to the introduction of higher levels of noise. The DFT is clearly the most affected, but now the DCT and DWT also show ripples across the face caused by the bluntness of the Laplace solution. For quality 50 the effects are similar, except that error surfaces are larger than expected. The price paid for such an introduction of noise is that the PDE method has the advantage that while the compressed file sizes are the same as for the sparse mesh, the uncompressed mesh has a high density compared to the original model.

The advantages are, therefore, smaller file sizes and the discovery that high density meshes are amenable to compression and recovery using relatively few structured planes. Error surfaces as a function of quality were quantified by running

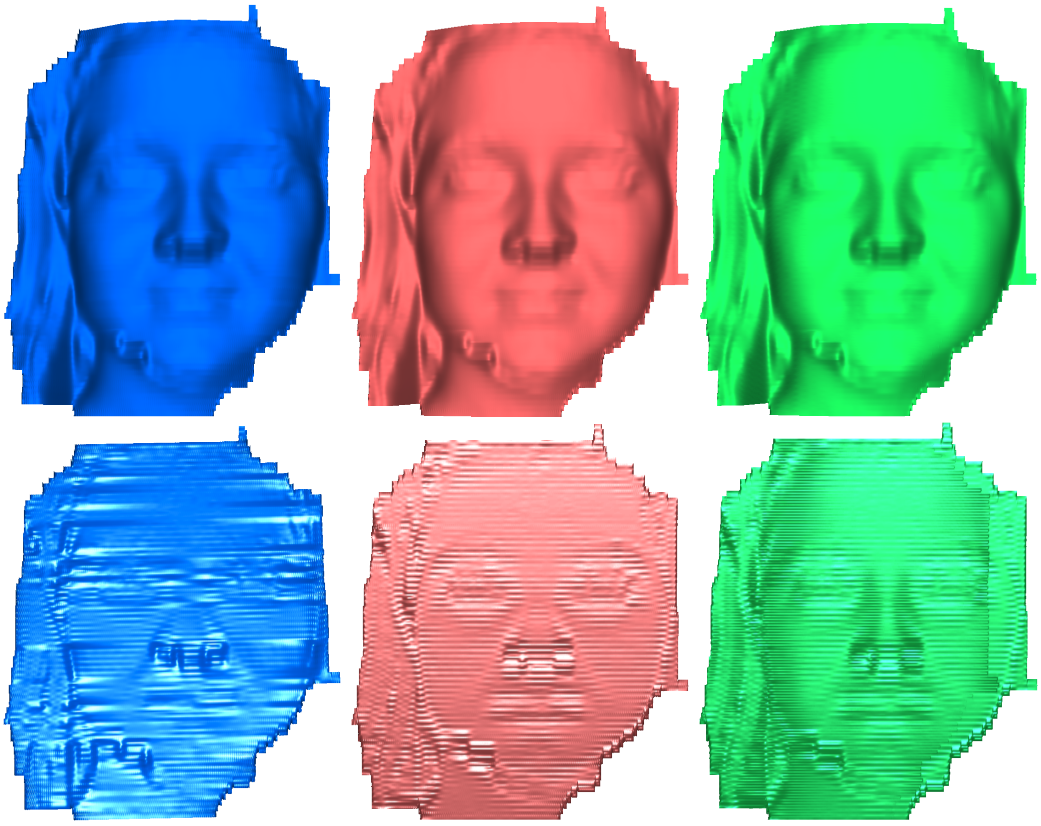

Figure 7.8: Quality $Q = 50$ with PDE based reconstruction. Top row: DFT (blue) DCT (red) and DWT (green). Bottom row: respective error surfaces.

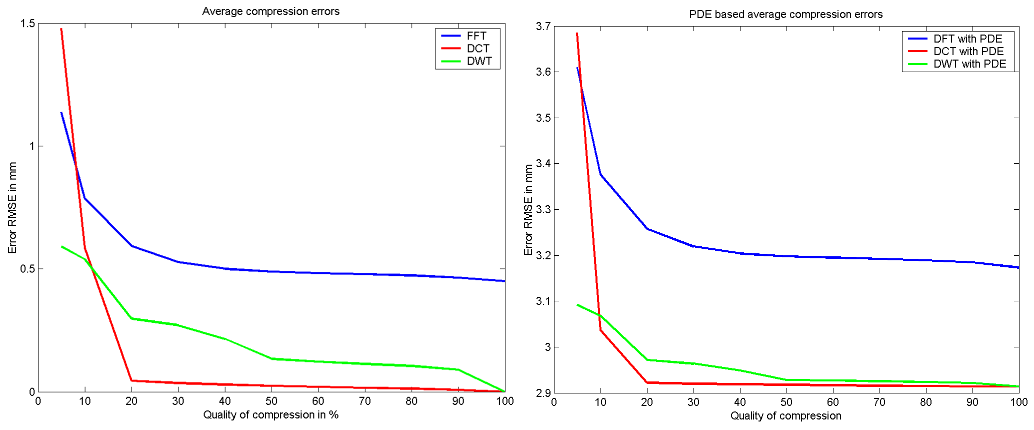

Figure 7.9: Average RMSE errors of uncompressed data for quality parameter $5 \leq Q \leq 100$. Left, standard DFT, DCT and DWT. Right, PDE based reconstruction.

experiments where the quality parameter was set to $Q =$ [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. Each quality parameter was applied in turn to the 86 models and summary statistics were computed for both with and without PDE based reconstruction. The RMSE of each error surface was estimated and the averages over 86 models are shown in Figure 7.9. The picture on the left was estimated by simply compressing the data from the multiple planes followed by reconstruction and it is directly comparable between these two data sets. Because this refers to the sparse mesh, it provides a straight comparison of the effectiveness of the techniques, although the original mesh density is not recovered. It is clear that DCT is the most appropriate technique as errors are very small up to a compression rate of 80% ($Q = 20$).

For any larger compression rate, surface errors grow exponentially. For aggressive compression rates of 90% and larger, the DWT technique is the most stable with relatively small error surfaces. The DFT is the worst performer showing consistently larger errors, but a point to note is that the RMSE of all three techniques stay near 0.5mm over a long range between $0-80\%$ compression. The picture on the right of Figure 7.9 shows the results for reconstruction using the PDE method to recover the original mesh density, with similar behaviour but with larger RMSEs. This is expected as the initial errors are compounded by uncertainties of data points estimated by PDE. Consequently, a comparison of the original dense mesh with the PDE mesh will show errors introduced by the Laplace approximation added to the underlying errors of the previous sparse mesh reconstruction. The Matlab code for error estimation can be found in functions `'gmprEstimateErrors.m'` see page(243-260) and `'gmprRMSE.m'` page (260) of Appendix B.

Finally a comparative analysis concerning file sizes was performed as specified in Tables 6.1 (DFT file format), 7.1 (DCT file format) and 7.2 (DWT file format). All compressed data are saved in plain ASCII format and the comparison is made with the Wavefront OBJ file format and a simple triplet of (x,y,z) floating points capable of holding equivalent 3D data in ASCII format.

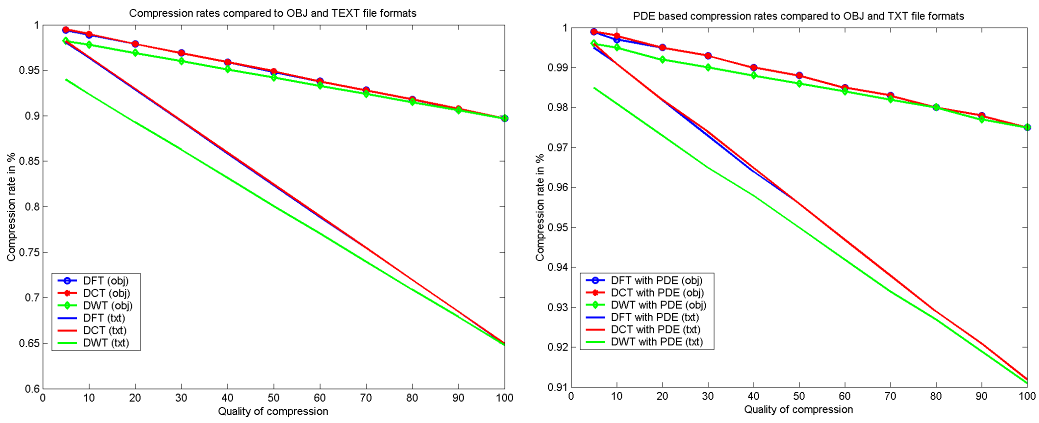

Figure 7.10: Average compression rates for quality parameter $5 \leq Q \leq 100$. Left, sparse mesh; right, sparse mesh with PDE reconstruction

Again the quality parameter was set to $Q = [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$ and compressed each of the 86 data files in turn. Figure 7.10 depicts the results for both sparse mesh and with PDE reconstruction. While both images show the same behaviour, the difference lies in the compression rates achieved. Regarding sparse meshes (left hand picture) where the density of triangular faces is similar in the sparse and uncompressed meshes, compression rates from 90–99% were achieved compared to OBJ files for all three techniques. When compared to the equivalent text file, 68–98% for DFT and DCT and 68–94% data compression rates are generally achieved for DWT. Using the PDE method yields higher compression rates (picture on the right) ranging from 97.5–99% compared to OBJ files for all three techniques. Compared to the equivalent text file, compression rates using the PDE method range from 91–99.5% for DFT and DCT and from 91–98.5% for DWT.

Concerning computation time, the performance of the process of compression and decompression of data has been measured and it is depicted in Tables 7.9–7.11 with an average compression rate of 98.2% for all 86 models. An aspect to note is that the proposed methods, although code and computationally efficient, have limitations concerning real time performance – for instance, the current implemented programs in Matlab cannot be applied to demanding applications such as real time 3D face recognition. For such applications, a re-implementation in more

Table 7.9: Compressed data files and CPU time

File number	Uncompress size in MB	Compress size in MB	Reduction Rates	Compress Time in Seconds
1	3.54	0.102	97.1%	76
2	2.711	0.034	98.7%	42
3	4.1	0.037	99.1%	138
4	3.46	0.035	99%	71
5	2.9	0.031	98.9%	31
6	2.88	0.055	98.1%	55
7	2.90	0.053	98.2%	52
8	1.91	0.061	96.8%	21
9	2.06	0.062	97.1%	24
10	4.31	0.072	98.3%	148
11	3.58	0.053	98.5%	59
12	3.12	0.057	98.2%	60
13	4.15	0.056	98.6%	124
14	2.98	0.050	98.3%	40
15	3.58	0.052	98.5%	70
16	2.16	0.050	97.7%	22
17	2.86	0.061	97.9%	47
18	1.64	0.060	96.3%	14
19	2.84	0.062	97.8%	48
20	2.59	0.057	97.8%	40
21	2.82	0.052	98.1%	47
22	2.55	0.064	97.9%	38
23	2.04	0.054	97.3%	23
24	1.92	0.054	97.2%	21
25	1.67	0.062	96.3%	15
26	2.33	0.051	97.8%	30
27	1.72	0.053	96.9%	16
28	4.68	0.059	98.9%	144
29	2.34	0.052	97.8%	28
30	2.56	0.060	97.6%	34
31	3.18	0.056	98.2%	59
32	1.75	0.057	96.7%	16
33	1.87	0.052	97.2%	19
34	2.99	0.057	98.1%	55
35	2.95	0.055	98.1%	52

Table 7.10: Compressed data files and CPU time

File number	Uncompress size in MB	Compress size in MB	Reduction Rates	Compress Time in Seconds
36	2.4	0.052	97.8%	33
37	2.33	0.052	97.7%	31
38	2.50	0.054	97.8%	37
39	2.30	0.052	97.7%	29
40	2.29	0.062	97.3%	33
41	2.45	0.051	97.9%	34
42	1.91	0.052	97.2%	20
43	1.91	0.408	97.9%	21
44	3.44	0.404	98.8%	68
45	4.21	0.036	99.1%	85
46	5.2	0.041	99.2%	140
47	2.99	0.041	98.6%	53
48	2.36	0.042	98.2%	33
49	4.02	0.039	99.1%	101
50	3.73	0.050	98.6%	81
51	2.97	0.040	98.6%	55
52	3.46	0.047	98.6%	65
53	3.90	0.045	98.8%	95
54	3.33	0.052	98.4%	70
55	2.24	0.032	98.6%	32
56	3.52	0.045	98.7%	82
57	3.56	0.037	99.0%	66
58	3.55	0.037	98.9%	68
59	4.22	0.051	98.9%	110
60	3.13	0.042	98.6%	57
61	3.47	0.043	98.8%	72
62	2.99	0.038	98.7%	51
63	6.16	0.042	99.3%	220
64	1.85	0.041	97.7%	19
65	3.34	0.042	98.72%	62

Table 7.11: Compressed data files and CPU time

File number	Uncompress size in MB	Compress size in MB	Reduction Rates	Compress Time in Seconds
66	4.01	0.044	98.9%	82
67	4.55	0.045	99.1%	131
68	3.19	0.047	98.5%	67
69	3.91	0.041	98.9%	100
70	2.77	0.044	98.4%	46
71	2.75	0.038	98.6%	47
72	3.15	0.048	98.4%	67
73	2.48	0.037	98.5%	34
74	2.84	0.043	98.5%	44
75	3.42	0.043	98.7%	75
76	3.20	0.035	98.9%	53
77	4.45	0.037	99.2%	97
78	2.57	0.036	98.6%	40
79	3.04	0.041	98.6%	57
80	4.20	0.041	99.0%	90
81	3.96	0.045	98.9%	94
82	2.33	0.044	98.1%	31
83	3.57	0.038	98.9%	56
84	3.04	0.033	98.9%	52
85	3.43	0.046	98.6%	66
86	3.55	0.049	98.6%	73

efficient programming language and environments will be necessary. To measure CPU processing time in all experiments, we used the built-in Matlab function `cputime`.

The overall observed pattern is that for sparse meshes (where the density of the structured planes is similar to the original density of the mesh with no significant polygon reduction applied) the error surfaces are generally very small with good compression rates. For high-density meshes errors tend to increase as polygon reduction introduces errors, which are made worse by the Laplace approximation. The advantage is that the PDE method is able to achieve higher compression rates for high-density meshes while the perceived quality of the data does not deteriorate significantly.

7.5 Discussion

In this chapter a comparative analysis of 3D data compression using the Discrete Fourier Transform, Discrete Cosine Transform, and Discrete Wavelet Transform is presented. It is shown that for the compression of 3D surface patches, both DCT and DWT based methods are superior to DFT in terms of error measures. The DCT has fewer coefficients compared to DFT because the implicit periodicity of DFT gives rise to boundary discontinuities that result in significant high frequency coefficients. Furthermore, most data (for instance 2D images) do not have much energy in the high frequency coefficients. In addition, studies have shown that DCT provides better energy compaction than DFT for most natural images. Furthermore, DCT approximates a linear interpolation between any two endpoints by using two lowest frequency coefficients and no discontinuity is observed, (see [Blinn, 1993] for more details).

In order to test the limits of compression while preserving the quality of the mesh, we carried out sensitivity analysis on the coefficients. Since the coefficients of DFT, DCT and DWT are the main parameters that define a compression, a discrete quality parameter Q was used ranging from 5–100. In practice, Q means the percentage of coefficients to keep, so $Q = 100$ means zero compression of coefficients. Q is applied to the vector of imaginary coefficients of DFT, to the DCT coefficients, and to the detail coefficients of DWT starting from the highest level.

A set of experiments using 86 high density meshes were used to compress and recover the data. Results demonstrate that both DCT and DWT are more robust than DFT to parametrically define the set of vertices on a mesh and reconstruct within a wide range of quality parameters. In particular, if we desire to compress coefficients very aggressively to over 90% then it is recommended to use DWT.

The PDE method has proved useful to recover the original mesh density, but it increases the RMSE of the reconstructed model as compared to a sparse mesh. The Laplace's equation solved by the method of lines (MOL) has also proved to be useful, but at the same time a blunt tool for this problem. Despite a higher RMSE the PDE method has been demonstrated to be an effective technique to recover high-density meshes and can be exploited in conjunction with other data compression techniques as demonstrated in this thesis.

Chapter 8

Conclusions and Further Work

8.1 Summary

The main focus of this thesis has been to investigate novel methods for 3D data compression with particular emphasis on surface patches. All data have been acquired by the GMPR scanner, whose point cloud characteristics are typical of standard 3D scanners. The research approach involved theoretical and experimental work. From a theoretical point of view, this thesis has developed new models for 3D data representation and compression from techniques such as polynomial interpolation, Fourier Transforms, Discrete Cosine Transforms, Wavelets Transforms, and Partial Differential Equations. Experimental work involved the development and testing of algorithms concerning the stability and accuracy of the solutions.

It is important to stress that the approach in this thesis has been to code the geometry of the data having connectivity of the mesh as a derived property. This is novel and contrasts with current approaches in the literature focused on coding the connectivity of the mesh. In the methods demonstrated here, connectivity is inferred directly from the vertex structure and thus, at reconstruction stage, no complex triangulation algorithms such as Delauney's are required.

First, given a set of unstructured 3D data from a surface patch (represented either as a point cloud or as a triangulated surface), this thesis has proposed a new method for imposing structure on the data. The method is based on calculating the minimum bounding box and orienting the surface patch with a global coordinate system where the z-axis of the data is oriented with the global Z-axis. Next, a number of cutting planes oriented with the global X and Y axes are defined over the bounding box. The number of such planes is arbitrary and depends on the characteristics of the data but, in principle, the higher the number of planes the more precise the representation. Each plane intersection defines a line and the point where this line intersects the surface defines a structured vertex. Each set of structured vertices lying in the plane can be treated as a one-dimensional signal with a constant step where each value represents the depth z of the data.

Second, a new method of polynomial interpolation was demonstrated for data compression. While polynomial interpolation is a well-known technique, specific problems were solved concerning missing data and required information to allow full reconstruction after compression. Since the cutting planes define a regular grid and not all vertices defined over this grid contain data, vertices need to be marked as valid or invalid. A specific representation was designed with information on the step between planes, the range of valid points, and the polynomial coefficients. This information is saved in plain ASCII allowing high rates of compression together with robust reconstruction of the data. Efficient compression rates of over 99% were achieved compared to the standard OBJ file format. The major issues with the method are concerned with the stability of the solution. While high degree polynomials (around degree 30 of the tested data) can reconstruct the data to acceptable accuracies with lowest RMSE they also introduce artefacts into the solution for higher degrees. Therefore, there seem to be intrinsic limitations to using high degree polynomials to approximate complex real world surface patches and new approaches are needed.

Third, novel spectral methods based on the Fourier Transform were proposed and demonstrated. The method follows on from polynomial interpolation and is based on coding the information in each plane by saving the scalar and vector (real and imaginary) Fourier coefficients. A quality parameter was introduced applied to the imaginary coefficients. A quality of 50 means discarding half of the coefficients from back to front, a quality of 100 means discard none. Note that the quality parameter is not applied to the real coefficients otherwise, it would adversely affect the quality of the reconstructed mesh. Upon reconstruction, all discarded coefficients are padded with zeros. The technique proved efficient and accurate, with RMSE at around 0.5mm for quality ranging from 10–100 with compression rates from 98–90%. The largest errors were observed at the boundaries of the model and on complex areas of the surface.

Fourth, the concepts of Partial Differential Equations were introduced and a novel method of reconstructing the mesh was demonstrated. The problem PDEs are addressing is that, the technique of cutting planes to structure the mesh normally results in a sparse mesh with fewer polygons than the original unstructured data. This can be significant, as experiments have shown that the cutting planes can reduce the original mesh to a quarter of its original size with no significant deterioration in quality. For some applications (such as 3D face recognition), it is desired to recover the original mesh density so not to impair recognition algorithms while not compromising compression ratios. The PDE method was applied over pairs of cutting planes used as boundary conditions and Laplace's equations were solved by the method of lines over this domain, thus recovering the original mesh density. Results have shown an increase in RMSE by a factor of 6 with compression rates from 99.8–97.5% for quality 5–100. Laplace's equation proved to be a blunt tool but still, the perceived quality of the reconstructed models makes the DFT with the PDE method a strong, valid solution.

Fifth, a novel method based on the Discrete Cosine Transform was proposed and demonstrated. Because the DCT only has real coefficients, the ASCII representation is more compact when compared to DFT. The quality parameter here applies to such coefficients and, similarly to DFT, all discarded coefficients are padded with zeros upon reconstruction. Experiments were carried out with and without the PDE method. Without PDE, results show very low RMSE, below 0.2mm for quality ranging 20–100. Compression rates in the order of 98–90% were achieved for quality 5–100. When PDE is included, the behaviour is the same, but with larger RMSE and higher compression rates ranging from 99.8–97.5% for the same quality parameters. With or without PDE, the DCT technique proved to be more accurate than DFT.

Sixth, a novel method for data compression based on Wavelet Transforms, was introduced and demonstrated. The DWT requires a more complex representation of the compressed data when compared with DFT and DCT. Here, it is necessary to keep all approximation and detail coefficients of the transformation. A 3-level Daubechie's DWT was used, as experiments have shown that little information is gained by increasing the level of decomposition. In this case, no compression is applied to the approximation coefficients, only the high frequency detail coefficients $D1, D2, D3$ are subject to compression. A quality parameter is applied similarly to DFT and DCT, starting to discard from $D3$, $D2$ then $D1$ in that order. Upon reconstruction, all discarded coefficients are padded with zeros. For most of the quality range from 5–100 DWT compression proved equivalent to DCT in terms of RMSE and compression rates. However, it proved superior for very low levels of quality, which means if one wished to compress very aggressively with quality at or below 10, the DWT is the preferred technique.

8.2 Conclusions

Evaluating the success of these methods will depend on the applications in which they are used; the assumption made here is that 3D surfaces are fairly complex with concave and convex local features. The human face is such a case, and it has been shown here that some errors show up around the nostril area and at the boundaries of the model. The observed errors may not impair applications such as in face recognition, because the nostrils are a known cause of errors and normally measurements in this area are avoided. Thus, it is expected that compression and reconstruction by the techniques demonstrated in this thesis would not cause significant problems for most applications.

The method of cutting planes applied to an unstructured point cloud or triangulated surface imposes a regular grid structure on the vertices and has proved a necessary step in applying the techniques described in this thesis. It is, in itself, a compression technique as vertices can now be saved for each plane without connectivity information as this can be inferred from the structure of the planes.

The source of some errors with the DFT method can be traced to the rounding off of the Fourier coefficients. Rounding increases the efficiency of data compression: for instance, for each entry in those vectors if $\text{abs}(a_n, b_n) < 0.001$ the values are rounded to zero. However, there are applications (outside the face recognition domain) where these very complex and detailed surfaces must be accurately modelled and the rounding operation may be skipped altogether. The result would be a less efficient compression but a more accurate data reconstruction. Another issue is to control the quality parameter for the intended application. High quality means larger files but this may not be a limiting factor for most uses. The main aspect about the technique is that all data are saved into relatively small ASCII files making them amenable to fast and secure encryption algorithms that can be efficiently and securely transmitted over a network.

The PDE method applied at the reconstruction stage proved very successful in recovering the original mesh density, which may be crucial to some applications. Laplace's equation is deemed an appropriate tool to solve the problem but, at the same time, it introduced errors as it is evaluated between pairs of cutting planes with obvious disregard to data lying in other adjacent planes. It is observed that the method of lines tends to evaluate Laplace's equation in almost straight lines between the boundaries and perhaps a more specialised function needs to be devised.

The comparative analysis of DFT, DCT and DWT in connection with PDE demonstrate that both DCT and DWT are more robust than DFT to parametrically define the set of vertices on a mesh and reconstruct within a wide range of quality parameters. In particular, if a very aggressive compression to over 90% is required then it is recommended to use the DWT technique. All techniques have been tested with an adjustable quality parameter, and yield low and relatively low RMSE either with or without PDE interpolation.

The range of 3D applications is continuously expanding: examples include medical visualisation, games, entertainment, engineering, CAD/CAM collaborative design, e-learning, and security, to mention just a few. Moreover, facilities for delivering 3D interactive models using a standard web browser are becoming available from Google and Mozilla; the number of such applications will also substantially increase in the near future. Internet bandwidth imposes hard limits and, although bandwidth is increasing slowly, current infrastructure constraints mean that the availability of efficient 3D compression technologies would benefit a wide range of industries.

8.3 Future Work

The work in this thesis presented a complete and coherent picture of employing new methods for 3D compression and reconstruction of surface patches. Nevertheless, the possible extensions to this work are both broad and numerous. Some possible extensions to be investigated are outlined as follows.

- The use of splines will be investigated, as it is possible to get more accurate results than with polynomials, and without stability problems. The price to be paid is that this will generate larger files, as the coefficients of all polynomials between control points need to be recorded.
- Future work will be focused on defining optimal PDE parameters aiming at reducing error surfaces. In particular, alternatives to Laplace's equation and to the method of lines that would be more appropriate to model complex 3D data with convex and concave regions. Furthermore, it will involve performing sensitivity analyses concerning levels of noise, smoothness of the surface, rounding errors and the complexity of data within each cutting plane.

- As an alternative to PDE and Laplace's equation, work is in progress on developing a moving 4th order polynomial interpolation applied to 4 adjacent vertices on 4 consecutive planes and recursively estimating the missing vertices.

- Equally, as an alternative to the method of lines, work is in progress on developing an approach for solving elliptic PDEs by spectral methods.

- While the human eye can smooth out the reconstructed meshes and perceive little difference between various compression rates, further work includes carrying out sensitivity analysis of face recognition algorithms operating on compressed meshes (as, for instance in [Rodrigues and Robinson, 2011]).

- Finally, and this is perhaps the most demanding aspects of future work, it will involve investigating how the methods demonstrated here would scale up to full 3D models as opposed to surface patches.

References

3DCT [2010], *3D Compression Technologies Inc.*, www.3dcompress.com/ web/default.asp.

Adamson, A. and Alexa, M. [2003], Approximating and intersecting surfaces from points, *in* 'Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing', Eurographics Association, pp. 230–239.

Adjerid, S., Flaherty, J. E. and Babuška, I. [1999], 'A posteriori error estimation for the finite element method-of-lines solution of parabolic problems', *Mathematical Models and Methods in Applied Sciences* **9**(02), 261–286.

Ahmat, N., Ugail, H. and Castro, G. G. [2011], 'Method of modelling the compaction behaviour of cylindrical pharmaceutical tablets', *International journal of pharmaceutics* **405**(1), 113–121.

Akkouche, S. and Galin, E. [2001], Adaptive implicit surface polygonization using marching triangles, *in* 'Computer Graphics Forum', Vol. 20, Wiley Online Library, pp. 67–80.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D. and Silva, C. T. [2003], 'Computing and rendering point set surfaces', *Visualization and Computer Graphics, IEEE Transactions on* **9**(1), 3–15.

Amenta, N. and Kil, Y. J. [2004], 'Defining point-set surfaces', **23**(3), 264–270.

Ames, A. L., Nadeau, D. R. and Moreland, J. [1997], *VRML 2.0 sourcebook*, Wiley.

Ang, D., Nghia, N. and Tam, N. [1998], ‘Regularized solutions of a cauchy problem for the laplace equation in an irregular layer: a three-dimensional model’, *Acta Math. Vietnam* **23**(1), 65–74.

Arnaud, R. and Barnes, M. C. [2006], *COLLADA: sailing the gulf of 3D digital content creation*, AK Peters Wellesley.

Arnaud, R. and Parisi, T. [2007], ‘Developing web applications with collada and x3d’, *A Whitepaper. March* **25**.

Auerbach, J. S., Chow, C.-S., Crigler, J. C. and Kaplan, M. A. [1997], ‘Creation and distribution of cryptographic envelope’. US Patent 5,673,316.

Babuska, I. [1995], *Modeling, mesh generation, and adaptive numerical methods for partial differential equations*, Vol. 75, Springer.

Balakrishnan, K. and Ramachandran, P. A. [1999], ‘A particular solution trefftz method for non-linear poisson problems in heat and mass transfer’, *Journal of Computational Physics* **150**(1), 239–267.

Bartels, S., Carstensen, C. and Hecht, A. [2006], ‘P2q2iso2d= 2d isoparametric fem in matlab’, *Journal of computational and applied mathematics* **192**(2), 219–250.

Belkasim, S. [2011], Multi-resolution analysis using symmetrized odd and even dct transforms, *in* ‘Data Compression Conference (DCC), 2011’, IEEE, pp. 447–447.

Bergh, J. and Löfström, J. [1976], *Interpolation spaces. An introduction*, Berlin.

Bernatz, R. [2010], *Fourier Series and Numerical Methods for Partial Differential Equations*, Wiley.

Beylkin, G. [1993], Wavelets and fast numerical algorithms, *in* ‘Proceedings of symposia in applied mathematics’, Vol. 47, pp. 89–117.

Bhamra, K. S. [2010], *Partial Differential Equations*, Prentice-Hall Of India Pvt. Limited.

Blinn, J. F. [1993], 'What's that deal with the dct?', *Computer Graphics and Applications, IEEE* **13**(4), 78–83.

Bloomenthal, J. [1988], 'Polygonization of implicit surfaces', *Computer Aided Geometric Design* **5**(4), 341–355.

Bloor, M. I. and Wilson, M. J. [1997], 'Generating parametrizations of wing geometries using partial differential equations', *Computer methods in applied mechanics and engineering* **148**(1), 125–138.

Bloor, M. and Wilson, M. [1989], 'Generating blend surfaces using partial differential equations', *Computer-Aided Design* **21**(3), 165–171.

Böhm, W., Farin, G. and Kahmann, J. [1984], 'A survey of curve and surface methods in cagd', *Computer Aided Geometric Design* **1**(1), 1–60.

Bremer, P.-T. and Hart, J. C. [2005], A sampling theorem for mls surfaces, *in* 'Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings', IEEE, pp. 47–54.

Briggs, W. L. et al. [1995], *The DFT: An Owners' Manual for the Discrete Fourier Transform*, Siam.

Brink, W., Robinson, A. and Rodrigues, M. A. [2008], Indexing uncoded stripe patterns in structured light systems by maximum spanning trees., *in* 'BMVC', pp. 1–10.

Brown, J. and Churchill, R. [2012*a*], *Fourier Series and Boundary Value Problems*, Brown and Churchill series, McGraw-Hill.

Brown, J. W. and Churchill, R. V. [2012*b*], 'Fourier series and boundary value problems', *AMC* **10**, 12.

Catmull, E. and Clark, J. [1978], ‘Recursively generated b-spline surfaces on arbitrary topological meshes’, *Computer-aided design* **10**(6), 350–355.

Chaikin, G. M. [1974], ‘An algorithm for high-speed curve generation’, *Computer graphics and image processing* **3**(4), 346–349.

Chen, J. and Chen, C. [2008], *Foundations of 3D Graphics Programming: Using JOGL and Java3D*, Foundations of 3D Graphics Programming: Using JOGL and Java3D, Springer.

Cohen, E., Lyche, T. and Riesenfeld, R. [1980], ‘Discrete **B**-splines and subdivision techniques in computer-aided geometric design and computer graphics’, *Computer graphics and image processing* **14**(2), 87–111.

Cooley, J. W., Lewis, P. A. and Welch, P. D. [1969], ‘The fast fourier transform and its applications’, *Education, IEEE Transactions on* **12**(1), 27–34.

Cooley, J. W. and Tukey, J. W. [1965], ‘An algorithm for the machine calculation of complex fourier series’, *Mathematics of computation* **19**(90), 297–301.

Dahmen, W., Müller, S. and Schlinkmann, T. [1999], ‘On a robust adaptive multigrid solver for convection-dominated problems’.

Davidson, D. and Hanson, R. [2004], ‘Interpreting shock tube ignition data’, *International journal of chemical kinetics* **36**(9), 510–523.

Davis, P. J. [1975], *Interpolation and approximation*, Courier Dover Publications.

de Boor, C. [2001], *A Practical Guide to Splines*, Applied Mathematical Sciences, Springer New York.

de Zeeuw, P. M. [2005], ‘A multigrid approach to image processing’, pp. 396–407.

Deering, M. [1995], Geometry compression, *in* ‘Proceedings of the 22nd annual conference on Computer graphics and interactive techniques’, ACM, pp. 13–20.

Dey, T. K. and Sun, J. [2005], . an adaptive mls surface for reconstruction with guarantees., *in* ‘Symposium on Geometry Processing’, pp. 43–52.

Dodgson, N., Floater, M. and Sabin, M. [2006], *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, Springer.

Doo, D. and Sabin, M. [1978], 'Behaviour of recursive division surfaces near extraordinary points', *Computer-Aided Design* **10**(6), 356–360.

Drath, R., Luder, A., Peschke, J. and Hundt, L. [2008], Automationml-the glue for seamless automation engineering, *in* 'Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on', IEEE, pp. 616–623.

Du, H. and Qin, H. [2005], 'Dynamic pde-based surface design using geometric and physical constraints', *Graphical Models* **67**(1), 43–71.

Duan, Y., Yang, L., Qin, H. and Samaras, D. [2004], Shape reconstruction from 3d and 2d data using pde-based deformable surfaces, *in* 'Computer Vision-ECCV 2004', Springer, pp. 238–251.

Duffy, D. G. [2008], *Mixed boundary value problems*, CRC Press.

Dyn, N. and Levin, D. [2002], 'Subdivision schemes in geometric modelling', *Acta Numerica* **11**, 73–144.

Edwards, R. E. [1979], *Fourier series*, Springer.

Elyan, E. and Ugail, H. [2007], 'Reconstruction of 3d human facial images using partial differential equations', *Journal of computers* **2**(8), 1–8.

Enderling, H., Anderson, A., Chaplain, M., Rowe, G. et al. [2006], 'Visualisation of the numerical solution of partial differential equation systems in three space dimensions and its importance for mathematical models in biology', *Mathematical Biosciences and Engineering* **3**(4), 571.

Evans, L. [2010], *Partial Differential Equations*, Vol. 19 of *Graduate studies in mathematics*, American Mathematical Society.
URL: *http://www.ams.org/bookstore-getitem/item=GSM-19-R*

Farin, G. E. [1996], *Curves and surfaces for computer-aided geometric design: a practical code*, Academic Press, Inc.

Farlow, S. J. [2012], *Partial differential equations for scientists and engineers*, Courier Dover Publications.

Foley, J. [1996], *Computer Graphics: Principles and Practice*, Addison-Wesley systems programming series, Addison-Wesley.

Forsey, D. R. and Bartels, R. H. [1988], Hierarchical b-spline refinement, *in* 'ACM SIGGRAPH Computer Graphics', Vol. 22, ACM, pp. 205–212.

Gachpazan, M., Kerayechian, A. and Kamyad, A. [2000], 'A new method for solving nonlinear second order partial differential equations', *Korean Journal of Computational & Applied Mathematics* **7**(2), 333–345.

Gakhov, F. D. [1990], *Boundary value problems*, Courier Dover Publications.

Galić, I., Weickert, J., Welk, M., Bruhn, A., Belyaev, A. and Seidel, H.-P. [2005], Towards pde-based image compression, *in* 'Variational, Geometric, and Level Set Methods in Computer Vision', Springer, pp. 37–48.

Geng, B., Zhang, H., Wang, H. and Wang, G. [2013], 'Approximate poisson disk sampling on mesh', *Science China Information Sciences* **56**(9), 1–12.

Gharge, S. and Krishnan, S. [2007], Simulation and implementation of discrete cosine transform for mpeg-4, *in* 'Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on', Vol. 4, IEEE, pp. 137–141.

Golbabai, A. and Javidi, M. [2007], 'A variational iteration method for solving parabolic partial differential equations', *Computers & Mathematics with Applications* **54**(7), 987–992.

Grafakos, L. [2004], *Classical and Modern Fourier Analysis*, Pearson/Prentice Hall.
URL: *http://books.google.co.uk/books?id=MToZAQAAIAAJ*

Grattan-Guinness, I. and Ravetz, J. [2003], *Joseph Fourier, 1768-1830: A Survey of His Life and Work*, MIT Press.
URL: *http://books.google.co.uk/books?id=DNmKHAAACAAJ*

Grebennikov, A. [2005], Solution of direct and inverse problems for laplace type equations by gr-method, *in* 'Proceedings of the WSEAS International Conferences MATH05, Cancun', pp. 1–6.

Griffiths, G. and Schiesser, W. E. [2010], *Traveling wave analysis of partial differential equations: numerical and analytical methods with MATLAB and Maple*, Academic Press.

Gumhold, S. and Straßer, W. [1998], Real time compression of triangle mesh connectivity, *in* 'Proceedings of the 25th annual conference on Computer graphics and interactive techniques', ACM, pp. 133–140.

Haberman, R. [1983], *Elementary applied partial differential equations*, Prentice Hall Englewood Cliffs, NJ.

Hadamard, J. [2003], *Lectures on Cauchy's problem in linear partial differential equations*, Courier Dover Publications.

Haidar, H., Egorova, S. and Soul, J. S. [2005], 'New numerical solution of the laplace equation for tissue thickness measurement in three-dimensional mri', *Journal of Mathematical Modelling and Algorithms* **4**(1), 83–97.

Hale, J. and Lunel, S. [1993], *Introduction to Functional Differential Equations*, number v. 99 *in* 'Applied Mathematical Sciences', Springer.
URL: *http://books.google.co.uk/books?id=ZNLjAJQMhqwC*

Halpern, D., Wilson, H. B. and Turcotte, L. H. [2002], *Advanced mathematics and mechanics applications using MATLAB*, CRC press.

Hamdi, S., Schiesser, W. and Griffiths, G. [2007], 'Method of lines', *Scholarpedia* **2**(7), 2859.

Hanna, J. R. and Rowland, J. H. [2008], *Fourier series, transforms, and boundary value problems*, Courier Dover Publications.

Harding, R. [1985], *Fourier Series and Transforms*, A computer illustrated text, Taylor & Francis.

Hase, H.-L. [1997], *Dynamische virtuelle Welten mit VRML 2.0*, Dpunkt, Verlag für digitale Technologie.

Heideman, M. T., Johnson, D. H. and Burrus, C. S. [1985], 'Gauss and the history of the fast fourier transform', *Archive for history of exact sciences* **34**(3), 265–277.

Helsing, J. and Wadbro, E. [2005], 'Laplaces equation and the dirichlet–neumann map: a new mode for mikhlins method', *Journal of Computational Physics* **202**(2), 391–410.

Hill, F. and Kelley, S. [2007], *Computer Graphics Using OpenGL, 3/E*, Pearson.

Isenburg, M. and Snoeyink, J. [2000], Face fixer: Compressing polygon meshes with properties, *in* 'Proceedings of the 27th annual conference on Computer graphics and interactive techniques', ACM Press/Addison-Wesley Publishing Co., pp. 263–270.

Jain, A. and Jain, J. [1978], 'Partial differential equations and finite difference methods in image processing–part ii: Image restoration', *Automatic Control, IEEE Transactions on* **23**(5), 817–834.

Jameson, L. [1993], On the daubechies-based wavelet differentiation matrix, Technical report, DTIC Document.

Jeffrey, A. [2003], *Applied Partial Differential Equations: An Introduction*, Academic Press.
URL: *http://books.google.co.uk/books?id=xHfgL0xF-p8C*

Kassam, A.-K. and Trefethen, L. N. [2005], 'Fourth-order time-stepping for stiff pdes', *SIAM Journal on Scientific Computing* **26**(4), 1214–1233.

Kato, A. and Ohno, N. [2009], 'Construction of three-dimensional tooth model by micro-computed tomography and application for data sharing', *Clinical oral investigations* **13**(1), 43–46.

Kessenich, J., Baldwin, D. and Rost, R. [2004], 'The opengl shading language', *Language version* **1**.

Kim, D. and Shin, D. [2003], Energy-based adaptive dct/idct for video coding, *in* 'Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on', Vol. 1, IEEE, pp. I–557.

King, D., Rossignac, J. and Szymczak, A. [2000], 'Connectivity compression for irregular quadrilateral meshes', *arXiv preprint cs/0005005* .

Koch, C. and Segev, I. [1998], *Methods in neuronal modeling: from ions to networks*, MIT press.

Kornprobst, P., Deriche, R. and Aubert, G. [1999], 'Image sequence analysis via partial differential equations', *Journal of Mathematical Imaging and Vision* **11**(1), 5–26.

Kronrod, B. and Gotsman, C. [2000], Efficient coding of non-triangular mesh connectivity, *in* 'Computer Graphics and Applications, 2000. Proceedings. The Eighth Pacific Conference on', IEEE, pp. 235–242.

Kubiesa, S., Ugail, H. and Wilson, M. [2004], 'Interactive design using higher order pdes', *The Visual Computer* **20**(10), 682–693.

Lahanas, M., Kemmerer, T., Milickovic, N., Karouzakis, K., Baltas, D. and Zamboglou, N. [2000], 'Optimized bounding boxes for three-dimensional treatment planning in brachytherapy', *Medical Physics* **27**(10), 2333–2342.

Lange, C. and Polthier, K. [2005], 'Anisotropic smoothing of point sets', *Computer Aided Geometric Design* **22**(7), 680–692.

Lee, H., Alliez, P. and Desbrun, M. [2002], Angle-analyzer: A triangle-quad mesh codec, *in* ‘Computer Graphics Forum’, Vol. 21, Wiley Online Library, pp. 383–392.

Li, J. and Hero, A. O. [2004], ‘A fast spectral method for active 3d shape reconstruction’, *Journal of Mathematical Imaging and Vision* **20**(1-2), 73–87.

Li, J. and Kuo, C.-C. [1998], A dual graph approach to 3d triangular mesh compression, *in* ‘Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on’, Vol. 2, IEEE, pp. 891–894.

Linsen, L. [2001], *Point cloud representation*, Univ., Fak. für Informatik, Bibliothek.

Liu, F., Anh, V. and Turner, I. [2004], ‘Numerical solution of the space fractional fokker–planck equation’, *Journal of Computational and Applied Mathematics* **166**(1), 209–219.

Loop, C. [1994], Smooth spline surfaces over irregular meshes, *in* ‘Proceedings of the 21st annual conference on Computer graphics and interactive techniques’, ACM, pp. 303–310.

Lord, G., Powell, C. and Shardlow, T. [2014], *An Introduction to Computational Stochastic PDEs*, Cambridge Texts in Applied Mathematics, Cambridge University Press.

Mai-Duy, N. and Tran-Cong, T. [2001], ‘Numerical solution of differential equations using multiquadric radial basis function networks’, *Neural Networks* **14**(2), 185–199.

Mainberger, M. and Weickert, J. [2009], Edge-based image compression with homogeneous diffusion, *in* ‘Computer Analysis of Images and Patterns’, Springer, pp. 476–483.

Malcolm Bloor, I. and Wilson, M. J. [1996], ‘Spectral approximations to pde surfaces’, *Computer-Aided Design* **28**(2), 145–152.

Mathelin, L. and Gallivan, K. [2010], 'A compressed sensing approach for partial differential equations with random input data', *Comput. Methods Appl. Mech. Eng.(2010, submitted)* .

Mathews, J. H. and Fink, K. D. [1994], 'Using matlab as a programming language for numerical analysis', *International Journal of Mathematical Education in Science and Technology* **25**(4), 481–490.

Meyer, Y. [1990], 'Ondelettes, vol. i: Ondelettes et op erateurs', *Hermann, Paris* .

Min, P., Halderman, J. A., Kazhdan, M. and Funkhouser, T. A. [2003], Early experiences with a 3d model search engine, *in* 'Proceedings of the eighth international conference on 3D Web technology', ACM, pp. 7–ff.

Nicholl, P., Ahmad, A. and Amira, A. [2010], Optimal discrete wavelet transform (dwt) features for face recognition, *in* 'Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on', IEEE, pp. 132–135.

Peloquin, C. E. [2009], 'Determination of critical factors for fast and accurate 2d medical image deformation'.

Peng, J., Kim, C.-S. and Jay Kuo, C.-C. [2005], 'Technologies for 3d mesh compression: A survey', *Journal of Visual Communication and Image Representation* **16**(6), 688–733.

Peng, J. and Kuo, C.-C. J. [2005], Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition, *in* 'ACM Transactions on Graphics (TOG)', Vol. 24, ACM, pp. 609–616.

Pennebaker, W. B. and Mitchell, J. L. [1993], *JPEG: Still image data compression standard*, Springer.

Piegl, L. [1991], 'On nurbs: a survey', *IEEE Computer Graphics and Applications* **11**(1), 55–71.

Piegl, L. and Tiller, W. [1987], 'Curve and surface constructions using rational b-splines', *Computer-Aided Design* **19**(9), 485–498.

Pinsky, M. [2011], *Partial Differential Equations and Boundary-value Problems with Applications*, Pure and applied undergraduate texts, American Mathematical Society.
URL: *http://books.google.co.uk/books?id=vi1HOeTwV5YC*

Pu, I. M. [2005], *Fundamental data compression*, Butterworth-Heinemann.

Qian, S.-E., Hollinger, A. B., Williams, D. and Manak, D. [1998], 3d data compression of hyperspectral imagery using vector quantization with ndvi-based multiple codebooks, *in* 'Geoscience and Remote Sensing Symposium Proceedings, 1998. IGARSS'98. 1998 IEEE International', Vol. 5, IEEE, pp. 2680–2684.

Qian, Z., Fu, C.-L. and Xiong, X.-T. [2006], 'Fourth-order modified method for the cauchy problem for the laplace equation', *Journal of Computational and Applied Mathematics* **192**(2), 205–218.

Qing, X. G. P. [2005], 'Geometric modelling by discrete surface patches based on geometric partial differential equations [j]', *Journal of Computer Aided Design & Computer Graphics* **12**, 002.

Quinn, J. A., Langbein, F. C. and Martin, R. R. [2007], Low-discrepancy point sampling of meshes for rendering., *in* 'SPBG', pp. 19–28.

Renardy, M. and Rogers, R. C. [2004], *An introduction to partial differential equations*, Vol. 4, Springer.

Ritger, P. and Rose, N. [1968], *Differential Equations with Applications*, Dover Books on Mathematics Series, Dover Publications.
URL: *http://books.google.co.uk/books?id=Eoaxq73utboC*

Rivara, M.-C. [1984], 'Design and data structure of fully adaptive, multigrid, finite-element software', *ACM Transactions on Mathematical Software (TOMS)* **10**(3), 242–264.

Robinson, A., Alboul, L. and Rodrigues, M. [2004], ‘Methods for indexing stripes in uncoded structured light scanning systems’, *Journal of WSCG* **12**(3), 371–378.

Rodrigues, M. A. and Robinson, A. [2010], ‘Novel methods for real-time 3d facial recognition’, *Strategic Advantage of Computing Information Systems in Enterprise Management, Majid Sarrafzadeh and Panagiotis Petratos (Eds) ISBN* pp. 978–960.

Rodrigues, M. A. and Robinson, A. [2011], Real-time 3d face recognition using line projection and mesh sampling, *in* ‘Proceedings of the 4th Eurographics conference on 3D Object Retrieval’, Eurographics Association, pp. 9–16.

Rodrigues, M. A., Robinson, A. and Brink, W. [2007], Issues in fast 3d reconstruction from video sequences, *in* ‘Proceedings of the 9th WSEAS international conference on Mathematical and computational methods in science and engineering’, World Scientific and Engineering Academy and Society (WSEAS), pp. 312–317.

Rodrigues, M. A., Robinson, A. and Brink, W. [2008], ‘Fast 3d reconstruction and recognition’, *New Aspects of Signal Processing, Computational Geometry and Artificial Vision, 8th WSEAS ISCGAV, Rhodes* pp. p15–21.

Rodrigues, M. A., Robinson, A. and Osman, A. [2010], Efficient 3d data compression through parameterization of free-form surface patches, *in* ‘Signal Processing and Multimedia Applications (SIGMAP), Proceedings of the 2010 International Conference on’, IEEE, pp. 130–135.

Rodrigues, M., Robinson, A., Alboul, L. and Brink, W. [2006], ‘3d modelling and recognition’, *WSEAS Transactions on Information Science and Applications* **3**(11), 2118–2122.

Rossignac, J. [2001], 3d compression made simple: Edgebreaker with zipandwrap on a corner-table, *in* ‘Shape Modeling and Applications, SMI 2001 International Conference on.’, IEEE, pp. 278–283.

Rule, K. [1996], *3D graphics file formats: a programmer's reference*, Addison Wesley Longman Publishing Co., Inc.

Sapiro, G. [2006], *Geometric partial differential equations and image analysis*, Cambridge university press.

Saucez, P., Wouwer, A. V. and Schiesser, W. [1998], 'An adaptive method of lines solution of the korteweg-de vries equation', *Computers & Mathematics with Applications* **35**(12), 13–25.

Schiesser, W. [1991], *The Numerical Method of Lines: Integration of Partial Differential Equations*, Academic Press.
URL: *http://books.google.co.uk/books?id=1vLFQgAACAAJ*

Schiesser, W. [1994], 'Method of lines solution of the korteweg-de vries equation', *Computers & Mathematics with Applications* **28**(10), 147–154.

Schneider, K. and Vasilyev, O. V. [2009], 'Wavelet methods in computational fluid dynamics*', *Annual Review of Fluid Mechanics* **42**(1), 473.

Schneider, R. and Kobbelt, L. [2001], 'Geometric fairing of irregular meshes for free-form surface design', *Computer aided geometric design* **18**(4), 359–379.

Sharan, M., Kansa, E. and Gupta, S. [1997], 'Application of the multiquadric method for numerical solution of elliptic partial differential equations', *Applied Mathematics and Computation* **84**(2), 275–302.

Sheng, Y., Willis, P., Castro, G. G. and Ugail, H. [2008], Pde-based facial animation: making the complex simple, *in* 'Advances in Visual Computing', Springer, pp. 723–732.

Shepard, D. [1968], A two-dimensional interpolation function for irregularly-spaced data, *in* 'Proceedings of the 1968 23rd ACM national conference', ACM, pp. 517–524.

Shikhare, D., Babji, S. V. and Mudur, S. [2002], Compression techniques for distributed use of 3d data–an emerging media type on the internet, *in* 'Proceedings of the International Conference on Computer Communication', Vol. 15, p. 676.

Shu, C., Ding, H. and Yeo, K. [2003], 'Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible navier–stokes equations', *Computer Methods in Applied Mechanics and Engineering* **192**(7), 941–954.

Smolic, A., Mueller, K., Merkle, P., Fehn, C., Kauff, P., Eisert, P. and Wiegand, T. [2006], 3d video and free viewpoint video-technologies, applications and mpeg standards, *in* 'Multimedia and Expo, 2006 IEEE International Conference on', IEEE, pp. 2161–2164.

Strang, G. and Aarikka, K. [1986], *Introduction to applied mathematics*, Vol. 16, Wellesley-Cambridge Press Wellesley, MA.

Stürmer, M., Köstler, H. and Rüde, U. [2008], 'A fast full multigrid solver for applications in image processing', *Numerical linear algebra with applications* **15**(2-3), 187–200.

Szymczak, A., King, D. and Rossignac, J. [2001], 'An edgebreaker-based efficient compression scheme for regular meshes', *Computational Geometry* **20**(1), 53–68.

Szymczak, A., Rossignac, J. and King, D. [2002], 'Piecewise regular meshes: Construction and compression', *Graphical Models* **64**(3), 183–198.

Talukder, K. H. and Harada, K. [2011], Enhancement of discrete wavelet transform (dwt) for image transmission over internet, *in* 'Information Technology: New Generations (ITNG), 2011 Eighth International Conference on', IEEE, pp. 1054–1055.

Taubin, G., Horn, W. P., Lazarus, F. and Rossignac, J. [1998], 'Geometry coding and vrml', *Proceedings of the IEEE* **86**(6), 1228–1243.

Taubin, G. and Rossignac, J. [1998], 'Geometric compression through topological surgery', *ACM Transactions on Graphics (TOG)* **17**(2), 84–115.

Tolstov, G. P. [2012], *Fourier series*, Courier Dover Publications.

Touma, C. and Gotsman, C. [1998], 'Triangle mesh compression', *PROC GRAPHICS INTERFACE. pp. 26-34. 1998* .

Trefethen, L. [2000], *Spectral Methods in MATLAB*, Software, Environments, and Tools, Society for Industrial and Applied Mathematics.
URL: *http://books.google.co.uk/books?id=pB4xiZKZ4ecC*

Trèves, F. [1975], *Basic linear partial differential equations*, Vol. 62, Academic press.

Triebel, H. [1999], *Interpolation Theory - Function Spaces - Differential Operators*, Wiley.
URL: *http://books.google.co.uk/books?id=BuWbGQAACAAJ*

Ugail, H. [2003], 'Parametric design and optimisation of thin-walled structures for food packaging', *Optimization and Engineering* **4**(4), 291–307.

Ugail, H., Bloor, M. I. and Wilson, M. J. [1999], 'Techniques for interactive design using the pde method', *ACM Transactions on Graphics (TOG)* **18**(2), 195–212.

Ugail, H. and Sourin, A. [2008], Partial differential equations for function based geometry modelling within visual cyberworlds, *in* 'Cyberworlds, 2008 International Conference on', IEEE, pp. 224–231.

Ugail, H. and Wilson, M. [2003], 'Efficient shape parametrisation for automatic design optimisation using a partial differential equation formulation', *Computers & structures* **81**(28), 2601–2609.

Urban, K. [2009], *Wavelet methods for elliptic partial differential equations*, Oxford University Press Oxford.

Van Schijndel, A. [2003], ‘Modeling and solving building physics problems with femlab’, *Building and Environment* **38**(2), 319–327.

Vasilyev, O. V. and Kevlahan, N. K.-R. [2005], ‘An adaptive multilevel wavelet collocation method for elliptic problems’, *Journal of Computational Physics* **206**(2), 412–431.

Vasilyev, O. V., Yuen, D. A., Paolucci, S. et al. [1997], ‘Wavelets: an alternative approach to solving pdes’, *UMSI research report/University of Minnesota (Minneapolis, Mn). Supercomputer institute* **97**, 97.

Vonesch, C., Blu, T. and Unser, M. [2007], ‘Generalized daubechies wavelet families’, *Signal Processing, IEEE Transactions on* **55**(9), 4415–4429.

Wali, M. K., Murugappan, M., Ahmad, R. B. and Zheng, B. S. [2012], Development of discrete wavelet transform (dwt) toolbox for signal processing applications, *in* ‘Biomedical Engineering (ICoBE), 2012 International Conference on’, IEEE, pp. 211–216.

Walker, J. [1996], *Fast Fourier Transforms, Second Edition*, Studies in Advanced Mathematics, Taylor & Francis.
URL: *http://books.google.co.uk/books?id=cOA-vwKIffkC*

Wang, C., Shi, Z., Li, L. and Niu, X. [2012], ‘Adaptive parameterization and reconstruction of 3d face images using partial differential equations’, *IJACT: International Journal of Advancements in Computing Technology* **4**(5), 214–221.

Wazwaz, A. [2002], *Partial Differential Equations*, Taylor & Francis.

Weatherill, N. P. and Hassan, O. [1994], ‘Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints’, *International Journal for Numerical Methods in Engineering* **37**(12), 2005–2039.

Weinberger, H. F. [2012], *A first course in partial differential equations: with complex variables and transform methods*, Courier Dover Publications.

Wiegmann, A. and Bube, K. P. [1998], 'The immersed interface method for nonlinear differential equations with discontinuous coefficients and singular sources', *SIAM Journal on Numerical Analysis* **35**(1), 177–200.

Witkin, A. P. and Heckbert, P. S. [1994], Using particles to sample and control implicit surfaces, *in* 'Proceedings of the 21st annual conference on Computer graphics and interactive techniques', ACM, pp. 269–277.

Xu, G., Pan, Q. and Bajaj, C. L. [2006], 'Discrete surface modelling using partial differential equations', *Computer Aided Geometric Design* **23**(2), 125–145.

Xu, J.-C. and Shann, W.-C. [1992], 'Galerkin-wavelet methods for two-point boundary value problems', *Numerische Mathematik* **63**(1), 123–144.

You, L., Comninos, P. and Zhang, J. J. [2004], 'Pde blending surfaces with¡ i¿ c¡/i¿¡ sup¿ 2¡/sup¿ continuity', *Computers & Graphics* **28**(6), 895–906.

You, L., Comninos, P., Zhang, M., Mikhael, W., Caballero, A., Abatzoglou, N., Tabrizi, M., Leandre, R., Garcia-Planas, M. and Choras, R. [2008], Analytical pde solid modelling, *in* 'WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering', WSEAS.

Young, R. M. [2001], *An Introduction to Non-Harmonic Fourier Series, Revised Edition, 93*, Academic Press.

Zhang, J. J. and You, L. [2001], Surface representation using second, fourth and mixed order partial differential equations, *in* 'Shape Modeling and Applications, SMI 2001 International Conference on.', IEEE, pp. 250–256.

Zhang, J. J. and You, L. [2002], 'Pde based surface representationvase design', *Computers & Graphics* **26**(1), 89–98.

Zhang, J. J. and You, L. [2004*a*], ‘Surface blending using a power series solution to fourth order partial differential equations’, *International Journal of Shape Modeling* **10**(02), 155–185.

Zhang, J.-J. and You, L.-H. [2004*b*], ‘Pde surface generation with combined closed and non-closed form solutions’, *Journal of Computer Science and Technology* **19**(5), 650–656.

Zill, D. [2012], *A First Course in Differential Equations with Modeling Applications*, Cengage Learning.
URL: *http://books.google.co.uk/books?id=pasKAAAAQBAJ*

Zorin, D., Schröder, P. and Sweldens, W. [1996], Interpolating subdivision for meshes with arbitrary topology, *in* ‘Proceedings of the 23rd annual conference on Computer graphics and interactive techniques’, ACM, pp. 189–192.

Zwillinger, D. [1998], *Handbook of differential equations*, Vol. 1, Gulf Professional Publishing.

Appendix A

Published papers

width=!,height=!,pages=2-4,7 width=!,height=!,pages=1-13

Appendix B

Source code

```
function gmprWriteOBJ(VL,FL,filename,flipfaces);
% GMPRWRITEOBJ( VL, FL, FILENAME, FLIPFACES )
% creates a Wavefront OBJ file from vertex and face
    lists and save to disk

if nargin == 3, flipfaces = 0; end
if flipfaces, FL = FL(:,[3 2 1]); end

fid = fopen(filename,'w');


nv = size(VL,1);
nf = size(FL,1);

n = nv + nf;
h = waitbar(0,'Writing obj file...','Position',[376
    400 272 53]);

for j = 1:nv,
    fprintf(fid,['v ', num2str(VL(j,:)), '\n']);
    waitbar(j/n);
end
fprintf(fid,'\n');

for j = 1:nf,
    fprintf(fid,['f  ', num2str(FL(j,:)), '\n']);
    waitbar((j+nv)/n);
end

fclose(fid);

close(h);
```

==

```
function gmprDrawBox3d(box, varargin)
% GMPRDRAWBOX3D Draw a 3D box defined by box
    coordinates
% BOX = [XMIN XMAX YMIN YMAX ZMIN ZMAX].
% The function draws only the outline edges of the box
    .

xmin = box(:,1);
xmax = box(:,2);
ymin = box(:,3);
ymax = box(:,4);
zmin = box(:,5);
zmax = box(:,6);

nBoxes = size(box, 1);

for i=1:length(nBoxes)
% lower face (z=zmin)
```

```matlab
gmprDrawEdge3d([xmin(i) ymin(i) zmin(i)    xmax(i) ymin(i) zmin(i)], varargin{:});
gmprDrawEdge3d([xmin(i) ymin(i) zmin(i)    xmin(i) ymax(i) zmin(i)], varargin{:});
gmprDrawEdge3d([xmax(i) ymin(i) zmin(i)    xmax(i) ymax(i) zmin(i)], varargin{:});
gmprDrawEdge3d([xmin(i) ymax(i) zmin(i)    xmax(i) ymax(i) zmin(i)], varargin{:});

% front face (y=ymin)
gmprDrawEdge3d([xmin(i) ymin(i) zmin(i)    xmin(i) ymin(i) zmax(i)], varargin{:});
gmprDrawEdge3d([xmax(i) ymin(i) zmin(i)    xmax(i) ymin(i) zmax(i)], varargin{:});
gmprDrawEdge3d([xmin(i) ymin(i) zmax(i)    xmax(i) ymin(i) zmax(i)], varargin{:});

% left face (x=xmin)
gmprDrawEdge3d([xmin(i) ymax(i) zmin(i)    xmin(i) ymax(i) zmax(i)], varargin{:});
gmprDrawEdge3d([xmin(i) ymin(i) zmax(i)    xmin(i) ymax(i) zmax(i)], varargin{:});

% the last 3 remaining edges
gmprDrawEdge3d([xmin(i) ymax(i) zmax(i)    xmax(i) ymax(i) zmax(i)], varargin{:});
gmprDrawEdge3d([xmax(i) ymax(i) zmin(i)    xmax(i) ymax(i) zmax(i)], varargin{:});
gmprDrawEdge3d([xmax(i) ymin(i) zmax(i)    xmax(i) ymax(i) zmax(i)], varargin{:});
end
```

```
=============================================================================
function VL = gmprCalculatePlane(n,p,minX,maxX,minY,maxY,minZ,maxZ);
% VL = GMPRCALCULATEPLANE(N,P,MINX,MAXX,MINY,MAXY,MINZ,MAXZ);
% Returns a list of the four corner vertices of a plane with normal N, passing through point P.

VL = zeros(4,3);
if abs(n(1)) > max(abs(n(2:3))),
    yy = maxY; zz = maxZ; xx = (n(2)*(p(2) - yy) + n(3)*(p(3) - zz))/n(1) + p(1); VL(1,:) = [xx yy zz];
    yy = maxY; zz = minZ; xx = (n(2)*(p(2) - yy) + n(3)*(p(3) - zz))/n(1) + p(1); VL(2,:) = [xx yy zz];
    yy = minY; zz = minZ; xx = (n(2)*(p(2) - yy) + n(3)*(p(3) - zz))/n(1) + p(1); VL(3,:) = [xx yy zz];
    yy = minY; zz = maxZ; xx = (n(2)*(p(2) - yy) + n(3)*(p(3) - zz))/n(1) + p(1); VL(4,:) = [xx yy zz];
elseif abs(n(2)) > max(abs(n([1,3]))),
    xx = maxX; zz = maxZ; yy = (n(1)*(p(1) - xx) + n(3)*(p(3) - zz))/n(2) + p(2); VL(1,:) = [xx yy zz];
    xx = maxX; zz = minZ; yy = (n(1)*(p(1) - xx) + n(3)*(p(3) - zz))/n(2) + p(2); VL(2,:) = [xx yy zz];
    xx = minX; zz = minZ; yy = (n(1)*(p(1) - xx) + n(3)*(p(3) - zz))/n(2) + p(2); VL(3,:) = [xx yy
```

```
        zz];
    xx = minX; zz = maxZ; yy = (n(1)*(p(1) - xx) + n
        (3)*(p(3) - zz))/n(2) + p(2); VL(4,:) = [xx yy
        zz];
else
    xx = maxX; yy = maxY; zz = (n(1)*(p(1) - xx) + n
        (2)*(p(2) - yy))/n(3) + p(3); VL(1,:) = [xx yy
        zz];
    xx = maxX; yy = minY; zz = (n(1)*(p(1) - xx) + n
        (2)*(p(2) - yy))/n(3) + p(3); VL(2,:) = [xx yy
        zz];
    xx = minX; yy = minY; zz = (n(1)*(p(1) - xx) + n
        (2)*(p(2) - yy))/n(3) + p(3); VL(3,:) = [xx yy
        zz];
    xx = minX; yy = maxY; zz = (n(1)*(p(1) - xx) + n
        (2)*(p(2) - yy))/n(3) + p(3); VL(4,:) = [xx yy
        zz];
end
```

==

```
function varargout = gmprDrawEdge3d(varargin)
% GMPRDRAWEDGE3D Draw 3D edge
% Draw the edge EDGE on the current axis. EDGE has the
    form:[x1 y1 z1 x2 y2 z2].
% No clipping is performed.

nCol = size(varargin{1}, 2);
if nCol==6
edges = varargin{1};
options = varargin(2:end);
elseif nCol==3
edges = [varargin{1} varargin{2}];
```

```
options = varargin(3:end);
elseif nCol==6
edges = [varargin{1} varargin{2} varargin{3} varargin
    {4} varargin{5} varargin{6}];
options = varargin(7:end);
end

h = line(   [edges(:, 1) edges(:, 4)]', ...
                [edges(:, 2) edges(:, 5)]', ...
                [edges(:, 3) edges(:, 6)]', 'color',
                   'b', 'Linewidth', 2);

                if ~isempty(options)
                set(h, options{:});
                end

        if nargout>0
                varargout{1}=h;
                end
```

==

```
function gmprDrawPlane()
% Draw planes at the vertex list VL
% VL = DRAWPLANE(N,P,MINX,MAXX,MINY,MAXY,MINZ,MAXZ);
% Returns a list of the four corner vertices of a
    plane with normal N, passing through point P.



addpath = [pwd '\Data'];
path(path, addpath);
gmprLoadData( 'Data\Data07.txt', 'Data\Data07Scale.txt
    ');
data  = load( 'Data\Data07.txt');
scale = load( 'Data\Data07Scale.txt');

depthScale1 = scale(1);
depthScale2 = scale(2);

maxZ =      max(max(data));
minZ =      min(min(data));

maxX = size(data,1);
maxY = size(data,2);

X = round( [1 2 3 ].*[ maxX/4 maxX/4 maxX/4]);
Y = round( [1 2 3 ].*[ maxY/4 maxY/4 maxY/4]);


Red   = [255/255 125/255 125/255];
Green = [125/255 255/255 125/255];

for i=1:3
    V=gmprCalculatePlane( [0 1 0], [X(i)*depthScale1 Y
        (i)*depthScale2 data(X(i)+1,Y(i))], 0, maxX*
        depthScale1, Y(i)*depthScale2, Y(i)*depthScale2
        , maxZ,minZ);
    patch(V(:,1),V(:,2),V(:,3),Red);
end

for i=1:3
    V=gmprCalculatePlane( [1 0 0], [X(i)*depthScale1 Y
        (i)*depthScale2 data(X(i)+1,Y(i))], maxY*
        depthScale1 , maxY*depthScale1 , 0, maxY*
        depthScale2 , maxZ,minZ);
    patch(V(:,1),V(:,2),V(:,3),Green);
end
```

==

```
function [P, First , Last] = gmprCompressPolynomials()
%Compress data by polynomial interpolation
%Before using this function adjust to the desired
   degree and filename

load face.txt
curves=face;

depthScale1 = 3.332910;
depthScale2 = 0.289025;

X=[];Y=[];Z=[]; P=[];
    bFirst = 0;
    bLast  = 0;
    bStarted = 0;
    bDone = 0;
    D = 10; %polynomial degree
    Filename = [ 'coefficients', num2str(D) ];

for x=1:size(curves,1)
    bFirst=0; bLast=0; bStarted=0; bDone=0;
    Y=[];Z=[];
    for y=1:size(curves,2)
        if ( isnan(curves(x,y)) & bStarted == 0 )
            %do nothing
        elseif ( isnan(curves(x,y)) & bStarted == 1 )
```

```
                %bLast = y-1;
                bDone=1;
            elseif ( bStarted == 0 )
                bStarted = 1;
                bFirst = y;
                Y = [Y; y*depthScale2];
                Z = [Z; curves(x,y)];
            elseif ( bStarted == 1 )
                Y = [Y; y*depthScale2];
                Z = [Z; curves(x,y)];
                if ( bDone == 0 )
                    bLast = y;
                end
            end
    end
    P = [P; polyfit(Y,Z,D) bFirst bLast];
    Y=[];Z=[];
end

%//////////////////// fix next line with the desired
    polynomial degree in the filename
save coefficients10.txt P -ASCII %save all
    coefficietns in ASCII
save(Filename,'P') %save all coefficients in .mat
    where the filename is "coefficients" + "D"

%now reconstruct the polynomials
pcurves = [];
for i=1:size(P,1)
    First = D+2; %first valid point
    Last  = D+3; %last valid point
    p = polyval( P(i,1:D+1), P(i,First)*depthScale2:depthScale2:P(i,Last)*depthScale2 );
    if P(i,First)> 1
        for j=1:P(i,First)-1
            p = [NaN p];
        end
    end
    if P(i,Last)<size(curves,2)
        for k=P(i,Last)+1:size(curves,2)
            p = [p NaN];
        end
    end
    pcurves = [pcurves; p];
end


%////////////////// fix next line with the desired polynomial degree in the filename
save facepoly10.txt pcurves -ASCII %save in ASCII with a generic name
Filename = [ 'facepoly', num2str(D) ];
save( Filename, 'pcurves' ); %save in .mat

% To load and visualize reconstructed 3D run from the command prompt
% gmprUncompressPolynomials( Filename );
% where Filename is 'facepoly3.txt', 'facepoly10.txt', 'facepoly15.txt', etc...

% The size of the compressed files can be checked by looking at the compessed files
% coefficients3.txt, coefficients10.txt, coefficients15.txt, etc...
=======================================================================
function [VL, FL]= gmprUncompressPolynomials(filename);
%Load polynomial compressed data, uncompress and display
%Parameter filename is 'facepoly3.txt', 'facepoly10.txt', 'facepoly15.txt', etc...


%3 Oct 2013
curves = load(filename); %this will load the correct plynomial file

depthScale1 = 3.332910; %this is for the test file, adjust if using a different file
depthScale2 = 0.289025;

depthZDim1 = size(curves,1);
depthZDim2 = size(curves,2);

X=[];Y=[];Z=[]; count=0;
for x=1:size(curves,1)
    for y=1:size(curves,2)
        X = [X; (x-1)*depthScale1];
        Y = [Y; (y-1)*depthScale2];
        Z = [Z; curves(x,y)];
        if ( not(isnan(curves(x,y))) )
            count=count+1;
        end
        end
end
VL = [X Y Z];

FL=[];
for x=1:depthZDim1-1
        for y=1:depthZDim2-1
                if ( not(isnan(curves(x,y))) & not(isnan(
                    curves(x+1,y))) & not(isnan(curves(x,y+1)))
                    ) %first triangle numbered 3-2-1
                    FL=[ FL; ( x*depthZDim2 + y ) ( (x-1)*
                        depthZDim2 + y +1 )  ((x-1)*depthZDim2
                        + y ) ];
                end
                if ( not(isnan(curves(x,y+1))) & not(isnan(
                    curves(x+1,y))) & not(isnan(curves(x+1,y+1)
                    )) ) %second triangle numbered 2-3-4
                    FL=[ FL; ((x-1)*depthZDim2 + y + 1) ( x*
                        depthZDim2 + y )  ( x*depthZDim2 + y +
                        1 )];
                end
        end
end

'valid vertices ='
count
gmprSurfaceView(VL,FL);
```

==

```
function compressedfilename = gmprCompressFFT(
    datafilename ,  scalefilename , quality )
% COMPRESSEDFILENAME = GMPRCOMPRESSFFT( DATAFILENAME,
    SCALEFILENAME, QUALITY )
% Compress 3D data using Fast Fourier Transform

addpath = [pwd '\Data'];
path(path, addpath);

stripes      = load(datafilename);
scale        = load(scalefilename);
depthScale1  = scale(1);
depthScale2  = scale(2);
depthZDim1   = size(stripes,1);
depthZDim2   = size(stripes,2);

compressedfile = [ 'cFFT' num2str(quality)
    datafilename ];
fid = fopen( compressedfile, 'w');

fprintf(fid,'%12.4f\t%12.4f\t%i\t%i\t%i\n',
    depthScale1, 2*depthScale2, depthZDim1, depthZDim2,
     quality );

started=0; start=0; finish=0;
a0=0; a6=0; an=[]; bn=[];
for r=1:depthZDim1
    started=0;
    for c=1:depthZDim2
        if not(isnan(stripes(r,c))) & not(started)
            started=1;
            start = c;
        elseif not(isnan(stripes(r,c))) & started
            finish = c;
        end
    end
    signal=[]; a0=0; a6=0; an=[]; bn=[];
    signal = stripes( r, start:finish );
    if length(signal) > 2
        [a0, a6, an, bn] = getfouriercoeff( stripes( r
            , start:finish ) );
    end

    an = clean(an,quality);
     bn = clean(bn,quality);
    fprintf(fid, '%i\t%i\t%12.4f\t%12.4f\t%i\t', start
        , finish, a0, a6, length(an) );
     L = length(an);
     Q = floor( (quality/100)*L );

    for k=1:Q
        fprintf(fid, '%12.4f\t', an(k) );
    end
    for k=1:Q
        fprintf(fid, '%12.4f\t', bn(k) );
    end
    fprintf(fid, '\n');
end
fclose(fid);

status = copyfile( compressedfile, ['Data\'
    compressedfile ]);
delete(compressedfile);
compressedfilename = compressedfile;

```

```

function c = clean( a, quality )
if length(a)>1
     L= length( a );
     Q = floor( (quality/100)*L );

     if (Q+1)<=L
          a(Q+1:end) = 0;
     end
end
c = a;

function [a0, a6, an, bn] = getfouriercoeff(
    singlestripe )
L=length(singlestripe)-1;
x=0:360/L:360;
d = fft(singlestripe);
m = length(singlestripe);
M = floor((m+1)/2);

a0 = d(1)/m;
an = 2*real(d(2:M))/m;
a6 = d(M+1)/m;
bn = -2*imag(d(2:M))/m;

n = 1:length(an);
y = a0 + an*cos(2*pi*n'*x/360) ...
        + bn*sin(2*pi*n'*x/360) ...
        + a6*cos(2*pi*6*x/360);
plot(x,y,'Linewidth',2)
legend('Raw data','FFT Interpolated')
```

```
===================================================================
function [ data, datafilename, scalefilename ] =
    gmprUncompressFFT( filename, N )
% [DATA, DATAFILENAME, SCALEFILENAME] =
    GMPRUNCOMPRESSFFT( FILENAME, N )
% uncompress data from fourier coefficientS

addpath = [pwd '\Data'];
path(path, addpath);

fid = fopen( filename, 'r');
depthScale1 = fscanf( fid, '%f', 1);
depthScale2 = fscanf( fid, '%f', 1);
depthZDim1  = fscanf( fid, '%i', 1);
depthZDim2 =  fscanf( fid, '%i', 1);
quality    =  fscanf( fid, '%i\n',1);

data = zeros( depthZDim1, round(depthZDim2/2) );
data(:,:)=NaN;

for r=1:depthZDim1
     start  = fscanf( fid, '%i', 1);
     finish = fscanf( fid, '%i', 1);
     a0 = fscanf( fid, '%f',1);
     a6 = fscanf( fid, '%f',1);
     L  = fscanf( fid, '%i',1);
     an=[];if L>1 an=zeros(1,L); end
     bn=[];if L>1 bn=zeros(1,L); end

     if L >= 2
          Q = floor( (quality/100)*L );
```

```
            for k=1:Q
                an(k) = fscanf( fid , '%f' ,1);
            end
            for k=1:Q
                bn(k) = fscanf( fid , '%f' ,1);
            end

            y = reconstructstripe( a0, a6, an, bn );
            data( r, round(start/2):round(start/2)+length(
               y)-1 ) = y;
        end
end
fclose(fid);

newstripes = []; depthScale2 = depthScale2/2;
for r=1:size(data,1)
    stripe=[];
    for c=1:size(data,2)-1
        stripe=[ stripe data(r,c) (data(r,c) + data(r,
           c+1))/2 ];
    end
    newstripes = [ newstripes; stripe ];
end
data = newstripes;
[R, C] = size(data);

uu=[];
if N>0
        for r=1:R-1
        S1 = data(r,:);
        S2 = data(r+1,:);
```

```
            u = gmprLaplace( S1, S2, N );
            if r==1
                uu = [uu; u];
            else
                uu = [uu; u(2:end,:)];
            end
        end
else
    uu=data;
end
depthScale1 = depthScale1/(N+1);
scale = [ depthScale1 depthScale2 ];
data = uu;

save pdeData.txt data -ASCII
save pdeScale.txt scale -ASCII
scalename = filename(1:end-4);
scalename = ['Data\pde' num2str(N) scalename 'Scale.
   txt'];
filename  = ['Data\pde' num2str(N) filename];
copyfile( 'pdeData.txt', filename);
copyfile( 'pdeScale.txt', scalename);
delete('pdeData.txt');
delete('pdeScale.txt');

datafilename  = filename;
scalefilename = scalename;

function y = reconstructstripe( a0, a6, an, bn )
n = 1:length(an);
L = length(an);
```

```
x = 0:360/L:360;
y = a0 + an*cos(2*pi*n'*x/360) ...
        + bn*sin(2*pi*n'*x/360) ...
        + a6*cos(2*pi*6*x/360);
y(end) = NaN;
```

==

```
function compressedfilename = gmprCompressDCT(
    datafilename, scalefilename, quality )
%COMPRESSEDFILENAME = GMPRCOMPRESSDCT( DATAFILENAME,
    SCALEFILENAME, QUALITY )
%Compress 3D data using Discrete Cosine Transform

if nargin == 0 | nargin == 1,
     disp('Not enough user-defined input parameters.
        Program aborted.');
     return
end

if nargin == 2,
     if isstr( datafilename ) & isstr( scalefilename )
          disp('Quality set to default.');
          quality = 100;
     else
       disp('Error.');
       disp('Data and scale filenames must be strings.'
          );
       disp('Program aborted.');
       return
     end
end

```

```
if nargin == 3,
    if isstr( datafilename ) & isstr( scalefilename )
      & quality >=1 & quality <=100
    else
      disp('Error.');
      disp('Data and scale filenames must be strings
         and 1<=quality <=100.');
      disp('Program aborted.');
      return
    end
end

addpath = [pwd '\Data'];
path(path, addpath);

stripes      = load(datafilename);
scale        = load(scalefilename);
depthScale1  = scale(1);
depthScale2  = scale(2);
depthZDim1   = size(stripes,1);
depthZDim2   = size(stripes,2);

compressedfile = [ 'cDCT' num2str(quality)
   datafilename ];
fid = fopen( compressedfile, 'w');

fprintf(fid,'%12.4f\t%12.4f\t%i\t%i\t%i\n',
   depthScale1, depthScale2, depthZDim1, depthZDim2,
   quality );

started=0; start=0; finish=0;
```

```
a0=0; a6=0; an=[]; bn=[];
for r=1:depthZDim1
    started=0;
    start=1; finish=1;
    for c=1:depthZDim2
        if not(isnan(stripes(r,c))) & not(started)
            started=1;
            start = c;
        elseif not(isnan(stripes(r,c))) & started
            finish = c;
        end
    end
    signal=[];
    signal = stripes( r, start:finish );
    B = getdctcoeff( signal );

    L= length( signal ); Q = floor( (quality/100)*L );
    fprintf(fid, '%i\t%i\t', start, finish );
    for k=1:Q
        fprintf(fid, '%12.4f\t', B(k) );
    end
    fprintf(fid, '\n');
end
fclose(fid);

copyfile( compressedfile, ['Data\' compressedfile ]);
delete(compressedfile);
compressedfilename = compressedfile;

function B = getdctcoeff( singlestripe )
B = dct( singlestripe );
```

```
================================================================
function  [ data, datafilename, scalefilename ]  =
    gmprUncompressDCT( filename, N )
% [DATA, DATAFILENAME, SCALEFILENAME] =
    GMPRUNCOMPRESSDCT( FILENAME, N )
% uncompress data from DCT coefficients

if nargin == 2,
    if isstr( filename )
    else
      disp('Error.');
      disp('Filename must be string. Program aborted.'
         );
      return
    end
end

addpath = [pwd '\Data'];
path(path, addpath);

fid = fopen( filename, 'r');
depthScale1 = fscanf( fid, '%f', 1);
depthScale2 = fscanf( fid, '%f', 1);
depthZDim1  = fscanf( fid, '%i', 1);
depthZDim2 =  fscanf( fid, '%i',1);
quality    =  fscanf( fid, '%i\n',1);

data = zeros( depthZDim1, depthZDim2 );
data(:,:)=NaN;

for r=1:depthZDim1
```

```
     start  = fscanf( fid , '%i', 1);
     finish = fscanf( fid , '%i', 1);

     L = finish-start+1;
     Q = floor( (quality/100)*L );
     dn=[];
     for k=1:Q
         dn(k) = fscanf( fid , '%f',1);
     end

     B=zeros(1,L);
     B(1:Q) = dn;

     y = idct( B );
     data( r, start:finish ) = y;
end
fclose(fid);

[R,C] = size(data);
uu=[];
if N>0
         for r=1:R-1
         S1 = data(r,:);
         S2 = data(r+1,:);
         u = gmprLaplace( S1, S2, N );
         if r==1
             uu = [uu; u];
         else
             uu = [uu; u(2:end,:)];
         end
         end
```

```
else
    uu=data;
end
depthScale1 = depthScale1/(N+1);
scale = [ depthScale1 depthScale2 ];
data = uu;

save pdeData.txt data -ASCII
save pdeScale.txt scale -ASCII

scalename = filename(1:end-4);
scalename = ['Data\pde' num2str(N) scalename 'Scale.
    txt' ];
filename  = ['Data\pde' num2str(N) filename];

copyfile( 'pdeData.txt', filename);
copyfile( 'pdeScale.txt', scalename);
delete('pdeData.txt');
delete('pdeScale.txt');

datafilename  = filename;
scalefilename = scalename;
```

==

```
function compressedfilename = gmprCompressDWT(
    datafilename, scalefilename, quality )
% COMPRESSEDFILENMAE = GMPRCOMPRESSDWT( DATAFILENAME,
    SCALEFILENAME, QUALITY )
% Compress 3D data using Discrete Wavelet Transform

addpath = [pwd '\Data'];
path(path, addpath);
```

```

stripes       = load(datafilename);
scale         = load(scalefilename);
depthScale1  = scale(1);
depthScale2  = scale(2);
depthZDim1   = size(stripes,1);
depthZDim2   = size(stripes,2);

compressedfile = [ 'cDWT' num2str(quality)
    datafilename ];
fid = fopen( compressedfile, 'w');

fprintf(fid,'%12.4f\t%12.4f\t%i\t%i\t%i\n',
    depthScale1, depthScale2, depthZDim1, depthZDim2,
    quality );

started=0; Start=0; Finish=0;
a0=0; a6=0; an=[]; bn=[];
for r=1:depthZDim1
    started=0;
    Start=1; Finish=1;
    for c=1:depthZDim2
        if not(isnan(stripes(r,c))) & not(started)
            started=1;
            Start  = c;
            Finish = c;
        elseif not(isnan(stripes(r,c))) & started
            Finish = c;
        end
    end
    signal=[];
    signal = stripes( r,  Start:Finish  );
    [C,L] = wavedec(signal,3,'db1');
   D = length( C( L(1) + 1  :  end  )  );
   Q = floor(  quality*D/100  );

    Q2Delete  = D-Q;
        if  Q2Delete > L(4)  + L(3)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+L(4);
       C(start:finish)=NaN;
        start  = L(1)+L(2)+1;
        finish  = L(1)+L(2)+L(3);
       C(start:finish)=NaN;
        start= L(1)+1;
        finish= L(1)+Q2Delete-(L(4)+L(3));
       C(start:finish)  = NaN;

        elseif  Q2Delete > L(4)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+L(4);
       C(start:finish)=NaN;
        start  = L(1)+L(2)+1;
        finish  = L(1)+L(2)+Q2Delete-L(4);
       C(start:finish)=NaN;

        elseif  Q2Delete <= L(4)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+Q2Delete;
       C(start:finish)=NaN;
       end

    fprintf(fid, '%i\t', Start );
    fprintf(fid, '%i\t', Finish );
    for k=1:length(L)
        fprintf(fid, '%i\t', L(k) );
    end
    fprintf(fid, '%i\t', length(C) );
    for k=1:length(C)
        if ~isnan( C(k) )
            fprintf(fid, '%12.4f\t', C(k) );
        else
            C(k)=0;
        end
    end
    fprintf(fid, '\n');
end
fclose(fid);

copyfile( compressedfile, ['Data\' compressedfile ]);
delete(compressedfile);
compressedfilename = compressedfile;

figure,mesh(stripes),title('Original data')
A3 = wrcoef('a',C,L,'db1',3);
D1 = wrcoef('d',C,L,'db1',1);
D2 = wrcoef('d',C,L,'db1',2);
D3 = wrcoef('d',C,L,'db1',3);
figure
subplot(2,2,1); plot(A3); title('Approximation A3')
subplot(2,2,2); plot(D1); title('Detail D1 compressed'
    )
subplot(2,2,3); plot(D2); title('Detail D2 compressed'
    )
subplot(2,2,4); plot(D3); title('Detail D3 compressed'
    )

A0 = waverec(C,L,'db1');
error3 = gmprRMSE( signal , A0 )

figure ,plot(A0)
figure ,plot(C)

quality = 1;
load Data01.txt
s=Data01( floor(size(Data01,1)/2), :);
s=s(29:683);
l_s=length(s);

[C,L] = wavedec(s,3,'db1');
cA3 = appcoef(C,L,'db1',3);
cD3 = detcoef(C,L,3);
cD2 = detcoef(C,L,2);
cD1 = detcoef(C,L,1);
[cD1,cD2,cD3] = detcoef(C,L,[1,2,3]); cD1(:)=0;

A3 = wrcoef('a',C,L,'db1',3);
D1 = wrcoef('d',C,L,'db1',1);
D2 = wrcoef('d',C,L,'db1',2);
D3 = wrcoef('d',C,L,'db1',3);
figure ,title( 'DWT')
subplot(2,2,1); plot(A3); title('Approximation A3')
subplot(2,2,2); plot(D1); title('Detail D1')
subplot(2,2,3); plot(D2); title('Detail D2')
subplot(2,2,4); plot(D3); title('Detail D3')

A0 = waverec(C,L,'db1');
error3 = max(abs(s-A0))

D = length( C( length(cA3)+1 : end ) );
Q = floor( quality*D/100 );

Q2Delete = D-Q;
if Q2Delete > L(4) + L(3)
     start = L(1)+L(2)+L(3)+1;
     finish = start+L(4)-1;
    C(start:finish)=0;
     start = L(1)+L(2)+1;
     finish = start + L(3)-1;
    C(start:finish)=NaN;
     start= L(1)+1;
     finish= start+Q2Delete-L(4)-L(3);
    C(start:finish) = NaN;

elseif Q2Delete > L(4)
     start = L(1)+L(2)+L(3)+1;
     finish = start+L(4)-1;
    C(start:finish)=NaN;
     start = L(1)+L(2)+1;
     finish = start + Q2Delete-L(4);
    C(start:finish)=NaN;

elseif Q2Delete <= L(4)
     start = L(1)+L(2)+L(3)+1;
     finish = start+Q2Delete;
    C(start:finish)=NaN;
end

A3 = wrcoef('a',C,L,'db1',3);
D1 = wrcoef('d',C,L,'db1',1);
D2 = wrcoef('d',C,L,'db1',2);
D3 = wrcoef('d',C,L,'db1',3);
figure
subplot(2,2,1); plot(A3); title('Approximation A3')
subplot(2,2,2); plot(D1); title('Detail D1 compressed'
    )
subplot(2,2,3); plot(D2); title('Detail D2 compressed'
    )
subplot(2,2,4); plot(D3); title('Detail D3 compressed'
    )

A0 = waverec(C,L,'db1');
error3 = gmprRMSE( s, A0 )
return
```

==

```
function [ data, datafilename, scalefilename ] =
    gmprUncompressDWT( filename, N )
% [DATA, DATAFILENAME, SCALEFILENAME] =
    GMPRUNCOMPRESSDCT( FILENAME, N )
% uncompress data from DWT coefficientS

if nargin == 2,
    if isstr( filename )
    else
      disp('Error.');
      disp('Filename must be string. Program aborted.'
            );
        return
    end
end

addpath = [pwd '\Data'];
path(path, addpath);

fid = fopen( filename, 'r');
depthScale1 = fscanf( fid, '%f', 1);
depthScale2 = fscanf( fid, '%f', 1);
depthZDim1  = fscanf( fid, '%i', 1);
depthZDim2 =  fscanf( fid, '%i',1);
quality     =  fscanf( fid, '%i\n',1);

data = zeros( depthZDim1, depthZDim2 );
data(:,:)=NaN;

for r=1:depthZDim1
    Start  = fscanf( fid, '%i', 1);
    Finish = fscanf( fid, '%i', 1);
    for k=1:5
        L(k) = fscanf( fid, '%i', 1);
    end
    Lc = fscanf( fid, '%i', 1);
    C  = zeros(1,Lc);
    D = length( C( L(1) + 1 : end ) );
    Q = floor( quality*D/100 );

    Q2Delete = D-Q;
        if Q2Delete > L(4) + L(3)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+L(4);
        C(start:finish)=NaN;
        start  = L(1)+L(2)+1;
        finish  = L(1)+L(2) + L(3);
        C(start:finish)=NaN;
        start= L(1)+1;
        finish= L(1)+Q2Delete-(L(4)+L(3));
        C(start:finish) = NaN;

        elseif Q2Delete > L(4)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+L(4);
        C(start:finish)=NaN;
        start  = L(1)+L(2)+1;
        finish  = L(1)+L(2) + Q2Delete - L(4);
        C(start:finish)=NaN;


        elseif Q2Delete <= L(4)
        start  = L(1)+L(2)+L(3)+1;
        finish  = L(1)+L(2)+L(3)+Q2Delete;
        C(start:finish)=NaN;
        end

    for k=1:Lc
        if ~isnan( C(k) )
            C(k) = fscanf(fid, '%f', 1 );
        else
            C(k)=0;
        end
    end

    A0 = waverec(C,L,'db1');
    data( r, Start:Finish ) = A0;
end
fclose(fid);


[R,C] = size(data);
uu=[];
if N>0
        for r=1:R-1
        S1 = data(r,:);
        S2 = data(r+1,:);
        u = gmprLaplace( S1, S2, N );
        if r==1
            uu = [uu; u];
        else
            uu = [uu; u(2:end,:)];
        end
        end
else
    uu=data;
end
depthScale1 = depthScale1/(N+1);
scale = [ depthScale1 depthScale2 ];
data = uu;

save pdeData.txt data -ASCII
save pdeScale.txt scale -ASCII

```

```
scalename = filename(1:end-4);
scalename = ['Data\pde' num2str(N) scalename 'Scale.
    txt' ];
filename  = ['Data\pde' num2str(N) filename];

copyfile( 'pdeData.txt', filename);
copyfile( 'pdeScale.txt', scalename);
delete('pdeData.txt');
delete('pdeScale.txt');

datafilename  = filename;
scalefilename = scalename;
```

==

```
function [VL, FL]= gmprLoadData( datafilename,
    scalefilename)
% [VL, FL] = GMPRLOADDATA( DATAFILENAME SCALEFILENAME
    ) loads a file
% and returns the Vertex List (VL) and Face List (FL)
    of the mesh structure.

addpath = [pwd '\Data'];
path(path, addpath);

stripes      = load(datafilename);
scale        = load(scalefilename);
depthScale1  = scale(1);
depthScale2  = scale(2);

depthZDim1  = size(stripes,1);
depthZDim2  = size(stripes,2);

```

```
disp(['Loading file ' datafilename '...']);
disp(['Number of stripes: ' num2str(depthZDim1) ]);
disp(['Vertices per stripe: ' num2str(depthZDim2)]);
disp('When it loads, toggle "i" for info, then "e" for edges' );
disp('Visualization can take a while, please wait...');


X=[];Y=[];Z=[]; count=0;
for x=1:size(stripes,1)
    for y=1:size(stripes,2)
        X = [X; (x-1)*depthScale1];
        Y = [Y; (y-1)*depthScale2];
        Z = [Z; stripes(x,y)];
        if ( not(isnan(stripes(x,y))) )
            count=count+1;
        end
    end
end
VL = [X Y Z];

FL=[];
xstep=1;
ystep=1;
for x=xstep+1:xstep:depthZDim1-xstep
    for y=1:ystep:depthZDim2-ystep
        if ( not(isnan(stripes(x,y))) & not(isnan(stripes(x+xstep,y))) & not(isnan(stripes(x,y+ystep))) )
            FL=[ FL; ( x*depthZDim2 + y ) ( (x-xstep)*
depthZDim2 + y + ystep )  ((x-xstep)*depthZDim2 + y ) ];
        end
        if ( not(isnan(stripes(x,y+ystep))) & not(isnan(stripes(x+xstep,y))) & not(isnan(stripes(x+xstep,y+ystep))) )
            FL=[ FL; ((x-xstep)*depthZDim2 + y + ystep ) ( x*depthZDim2 + y )  ( x*depthZDim2 + y + ystep )];
        end
    end
end

gmprWriteOBJ( VL, FL, [ datafilename(1:end-3) 'obj' ] );
gmprSurfaceView(VL,FL);
```

===

```
function gmprSurfaceView(VL,FL,CL,i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15,i16,i17,i18,i19,i20,i21,i22);
% SURFACEVIEW   Visualize a 3D surface defined by vertex list and face list.
%
% surfaceview(VL,FL)
%   Draws the surface defined by vertex list VL and face list FL.
%
% surfaceview(VL,FL,CL)
%   Draws the surface defined by VL and FL, with added vertex colors given by CL.
%
```

```
% surfaceview(VL,FL,CL,info,axesbox,projection,
  background, ...
%               texture,facecolor,edgecolor,material,
  facealpha,vertices,edges,faces,normals, ...
%               viewpoint,cameraviewangle,cameratarget,
  ...
%               light,lightposition,smoothshading,
  movelight,backfacelight)
%
%  Possible values for these arguments (default
  values shown in {}):
%     CL = n-by-3 matrix  {[]}
%     info = {0} | 1
%     navigator = {0} | 1
%     axesbox = {0} | 1
%     projection = 0 | {1}
%     background = 1 | {2} | 3
%     texture = {0} | 1
%     facecolor = 3-by-1 vector  {[.9 .8 .6]}
%     edgecolor = 3-by-1 vector  {[0 .4 .4]}
%     material = 1 | {2} | 3
%     facealpha = scalar >= 0 and <= 1  {1}
%     vertices = {0} | 1
%     edges = 0 | {1}
%     faces = 0 | {1}
%     normals = {0} | 1
%     viewpoint = 2-by-1 vector  {[135 24]}
%     cameraviewangle = scalar > 0 and <= 180  {[]}
%     cameratarget = 3-by-1 vector  {[]}
%     light = 0 | {1}
%     lightposition = 3-by-1 vector  {[]}
%       smoothshading = 0 | {1}
%       movelight = 0 | {1}
%       backfacelight = {0} | 1

global handlefig handleaxes handlenavi handlesurf
    handlevert handlenorm handleligt info infotext navi
     navigator flippinview axesbox proj background
    texture canmap facecolor edgecolor materiaal
    facealpha vertices edges faces normals ligt
    smoothshading movelwcam bfrlight boxrange


if nargin == 0,
    VL = [0 0 0];
    FL = [];
    CL = [];
end

if nargin < 25,
    if nargin > 3, disp('Not enough user-defined input
        parameters, defaulting all values.'); end
    if nargin == 2, canmap = 0; else canmap = ~isempty
        (CL); end
    info = 0;
    navigator = 0;
    axesbox = 0;
    proj = 1;
    background = 2;
    texture = 0;
    facecolor = [.9 .8 .6];
    edgecolor = [0 .4 .4];
    materiaal = 2;
    facealpha = 1;
    vertices = 0;
    edges = 1;
    faces = 1;
    normals = 0;
    viewpoint = [135  24];
    viewangle = [];
    camtarg = [];
    ligt = 1;
    lightpos = [];
    smoothshading = 1;
    movelwcam = 1;
    bfrlight = 0;
else
    canmap = ~isempty(CL);
    info = i1;
    navigator = i2;
    axesbox =i3;
    proj = i4;
    background = i5;
    texture = i6;
    facecolor = i7;
    edgecolor = i8;
    materiaal = i9;
    facealpha = i10;
    vertices = i11;
    edges = i12;
    faces = i13;
    normals = i14;
    viewpoint = i15;
```

```
     viewangle = i16;
     camtarg = i17;
     ligt = i18;
     lightpos = i19;
     smoothshading = i20;
     movelwcam = i21;
     bfrlight = i22;
end

if isempty(info), info = 0; end
if isempty(navigator), navigator = 0; end
if isempty(axesbox), axesbox = 0; end
if isempty(proj), proj = 1; end
if isempty(background), background = 2; end
if isempty(texture), texture = 0; end
if isempty(facecolor), facecolor = [.9 .8 .6]; end
if isempty(edgecolor), edgecolor = [0 .4 .4]; end
if isempty(materiaal), materiaal = 2; end
if isempty(facealpha), facealpha = 1; end
if isempty(vertices), vertices = 0; end
if isempty(edges), edges = 1; end
if isempty(faces), faces = 1; end
if isempty(normals), normals = 0; end
if isempty(viewpoint), viewpoint = [135 24]; end
if isempty(ligt), ligt = 1; end
if isempty(smoothshading), smoothshading = 1; end
if isempty(movelwcam), movelwcam = 1; end
if isempty(bfrlight), bfrlight = 0; end

%fname = '(no file loaded)';
if isempty(FL),
    sNF = '';
    sNV = '';
else
    N = size(VL,1); sN = num2str(N);
    ekspvorm = 0; for j = 1:length(sN), if sN(j) == 'e
        ', ekspvorm = 1; end; end
    if ~ekspvorm,
        if abs(round(log10(N)) - log10(N)) < 10*eps,
            aantalkommas = floor(floor(log10(N + 1))/3)
            ; else aantalkommas = floor(floor(log10(N))
            /3); end
        for j = 1:aantalkommas, sN = strcat(sN(1:(end
            -4*j+1)),',',sN((end-4*j+2):end)); end
    end
    sNV = sN;
    N = size(FL,1); sN = num2str(N);
    ekspvorm = 0; for j = 1:length(sN), if sN(j) == 'e
        ', ekspvorm = 1; end; end
    if ~ekspvorm,
        if abs(round(log10(N)) - log10(N)) < 10*eps,
            aantalkommas = floor(floor(log10(N + 1))/3)
            ; else aantalkommas = floor(floor(log10(N))
            /3); end
        for j = 1:aantalkommas, sN = strcat(sN(1:(end
            -4*j+1)),',',sN((end-4*j+2):end)); end
    end
    sNF = sN;
end

warning off % turn "Unrecognized OpenGL version,
    defaulting to 1.0." warnings off
```

```
rand('state',sum(100*clock)); % reset state of random
    number generator

X = VL(:,1);
Y = VL(:,2);
Z = VL(:,3);

handlefig = figure;
set(handlefig,'CloseRequestFcn','closereq, warning on'
    ); %turn warnings back on after killing figure
set(handlefig,'Name','Surface Viewer','NumberTitle','
    off','MenuBar','none');

a1 = axes('Position',[0 0 1 1],'Visible','off');
handlenavi = axes('Position',[.89 .05 .06 .06]);
handleaxes = axes('Position',[0.13 0.11 0.775 0.815]);
     axis off

set(gcf,'CurrentAxes',handlenavi), hold on
navi = {};
navi{1} = plot3(0,0,0,'.','Color',[0 0 0.85]);
navi{2} = patch('Vertices',0.08*[1 0 0; 0 0 0; 0 1 0;
    1 1 0; 1 0 1; 0 0 1; 0 1 1; 1 1 1],'Faces',[1 2 3
    4; 5 8 7 6; 1 5 6 2; 2 6 7 3; 3 7 8 4; 4 8 5 1],'
    FaceColor',[.6 .3 .3],'EdgeColor','k','LineWidth'
    ,1,'FaceAlpha',0.98);
navi{3} = plot3([0 0.15],[0 0],[0 0],'Color',[0 0
    0.85],'LineWidth',2);
navi{4} = plot3([0 0],[0 0.15],[0 0],'Color',[0 0
    0.85],'LineWidth',2);
```

```
navi{5} = plot3([0 0],[0 0],[0 0.15],'Color',[0 0
    0.85],'LineWidth',2);
navi{6} = text(0.18,0,0,'x','FontName','Arial','
    FontSize',9,'FontWeight','bold');
navi{7} = text(0,0.18,0,'y','FontName','Arial','
    FontSize',9,'FontWeight','bold');
navi{8} = text(0,0,0.18,'z','FontName','Arial','
    FontSize',9,'FontWeight','bold');
axis image, view([135,24]); axis vis3d, axis off

if ~navigator,
    for j = 1:length(navi), set(navi{j},'Visible','off
        '); end
end

set(gcf,'CurrentAxes',a1), hold on

axis([0 1 0 1]);
infotext = {};
infotext{1} = text(.05,.92,'General','FontName','Arial
    ','Fontsize',9,'FontWeight','bold','Color',[0 0 0])
    ;
infotext{2} = text(.05,.80,'Axes','FontName','Arial','
    Fontsize',9,'FontWeight','bold','Color',[0 0 0]);
infotext{3} = text(.05,.68,'Surface','FontName','Arial
    ','Fontsize',9,'FontWeight','bold','Color',[0 0 0])
    ;
infotext{4} = text(.05,.44,'Camera','FontName','Arial'
    ,'Fontsize',9,'FontWeight','bold','Color',[0 0 0]);
infotext{5} = text(.05,.24,'Light & Shading','FontName
    ','Arial','Fontsize',9,'FontWeight','bold','Color'
    ,[0 0 0]);

infotext{6} = text(.06,.90,'i','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{7} =  text(.08,.90,'show/hide information'
    ,'FontName','Arial','Fontsize',8);
infotext{8} = text(.06,.88,'o','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{9} =  text(.08,.88,'open surface from MAT-
    file','FontName','Arial','Fontsize',8);
infotext{10} = text(.06,.86,'d','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{11} =  text(.08,.86,'display all
    parameters in command window','FontName','Arial','
    Fontsize',8);
infotext{12} = text(.06,.84,'g','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{13} =  text(.08,.84,'export current screen
     as TIFF','FontName','Arial','Fontsize',8);

infotext{14} = text(.06,.78,'j','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{15} = text(.08,.78,'flip coordinate system
    ','FontName','Arial','Fontsize',8);
infotext{16} = text(.06,.76,'a','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{17} = text(.08,.76,'axes border on/off','
    FontName','Arial','Fontsize',8);
infotext{18} = text(.06,.74,'p','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{19} = text(.08,.74,'perspective/orthogonal
     projection','FontName','Arial','Fontsize',8);
infotext{20} = text(.06,.72,'w','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{21} = text(.08,.72,'white/gray/black
    background','FontName','Arial','Fontsize',8);

infotext{22} = text(.06,.66,'c','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{23} = text(.08,.66,'change uniform surface
     color','FontName','Arial','Fontsize',8);
infotext{24} = text(.06,.64,'k','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{25} = text(.08,.64,'change edge color','
    FontName','Arial','Fontsize',8);
infotext{26} = text(.06,.62,'u','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{27} = text(.08,.62,'shiny/dull/metallic
    reflectance','FontName','Arial','Fontsize',8);
infotext{28} = text(.06,.60,'t','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{29} = text(.08,.60,'show/hide texture','
    FontName','Arial','Fontsize',8);
infotext{30} = text(.06,.58,'+','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{31} = text(.08,.58,'increase surface
    transparency','FontName','Arial','Fontsize',8);
infotext{32} = text(.06,.56,'-','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{33} = text(.08,.56,'decrease surface
    transparency','FontName','Arial','Fontsize',8);
infotext{34} = text(.06,.54,'v','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{35} = text(.08,.54,'show/hide vertices','
    FontName','Arial','Fontsize',8);
infotext{36} = text(.06,.52,'e','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{37} = text(.08,.52,'show/hide edges','
    FontName','Arial','Fontsize',8);
infotext{38} = text(.06,.50,'f','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{39} = text(.08,.50,'show/hide faces','
    FontName','Arial','Fontsize',8);
infotext{40} = text(.06,.48,'n','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{41} = text(.08,.48,'show/hide face normals
    ','FontName','Arial','Fontsize',8);

infotext{42} = text(.06,.42,'q','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{43} = text(.08,.42,'navigator on/off','
    FontName','Arial','Fontsize',8);
infotext{44} = text(.06,.40,'r','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{45} = text(.08,.40,'reset camera to
    default position','FontName','Arial','Fontsize',8);
infotext{46} = text(.06,.38,'x','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{47} = text(.08,.38,'view down x-axis','
    FontName','Arial','Fontsize',8);
infotext{48} = text(.06,.36,'y','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{49} = text(.08,.36,'view down y-axis','
    FontName','Arial','Fontsize',8);
infotext{50} = text(.06,.34,'z','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{51} = text(.08,.34,'view down z-axis','
    FontName','Arial','Fontsize',8);
infotext{52} = text(.057,.32,'left click & drag to
    rotate','FontName','Arial','Fontsize',8);
infotext{53} = text(.057,.30,'middle click & drag to
    translate','FontName','Arial','Fontsize',8);
infotext{54} = text(.057,.28,'right click & drag to
    zoom','FontName','Arial','Fontsize',8);

infotext{55} = text(.06,.22,'l','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{56} = text(.08,.22,'light source on/off','
    FontName','Arial','Fontsize',8);
infotext{57} = text(.06,.20,'s','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{58} = text(.08,.20,'enable/disable smooth
    shading','FontName','Arial','Fontsize',8);
infotext{59} = text(.06,.18,'m','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{60} = text(.08,.18,'move light with camera
     on/off','FontName','Arial','Fontsize',8);
infotext{61} = text(.06,.16,'b','FontName','Arial','
    Fontsize',8,'HorizontalAlignment','center');
    infotext{62} = text(.08,.16,'back face lighting on/
    off','FontName','Arial','Fontsize',8);

infotext{63} = text(.05,.07,strcat([sNV,' vertices   '
    , sNF, ' faces']),'FontName','Arial','Fontsize',9,'
    FontWeight','bold','HorizontalAlignment','left');
if isempty(FL), set(infotext{63},'String','no file
    loaded'); end

if ~info,
     for j = 1:length(infotext), set(infotext{j},'
        Visible','off'); end;
end

set(gcf,'CurrentAxes',handleaxes), hold on


if isempty(FL), handlesurf = trisurf(FL,X,Y,Z);
else handlesurf = trisurf(FL(:,[3 2 1]),X,Y,Z); end
handlevert = plot3(X,Y,Z,'.','Color',[.8 0 0]);

if canmap, set(handlesurf,'FaceVertexCData',CL); end

set(handlesurf,'FaceColor',facecolor,'FaceLighting','
    gouraud','FaceAlpha',facealpha,'BackFaceLighting','
    lit','EdgeColor',edgecolor);

if bfrlight, set(handlesurf,'BackFaceLighting','
    reverselit'); end

if canmap & texture, set(handlesurf,'FaceColor','
    interp'); end
if ~canmap, texture = 0; end

if materiaal == 1, material shiny;
elseif materiaal == 2, material dull;
```

```
else material metal; end

handleligt = light('Position',[1 0 1],'Style','
    infinite');

view(viewpoint);

if axesbox, axis on, else axis off, end
xlabel(''), ylabel(''), zlabel(''),

if isempty(viewangle),
    set(handleaxes,'CameraViewAngleMode','auto');
else
    set(handleaxes,'CameraViewAngle',viewangle);
end
axis image, axis vis3d

if proj,
    set(gca,'Projection','perspective');
else
    set(gca,'Projection','orthographic');
end


set(gca,'Box','on','Color','none');


set(gca,'XColor','k','YColor','k','ZColor','k');
set(gca,'FontName','Arial','Fontsize',9.0,'FontWeight'
    ,'Demi');

```

```
if background == 1,
    set(handlefig,'Color',[1 1 1]);
    set(gca,'Color','none','XColor','k','YColor','k','
        ZColor','k');
    for j = 1:length(infotext), set(infotext{j},'Color
        ','k'); end;
elseif background == 2,
    set(handlefig,'Color',[0.8 0.8 0.8]);
    set(gca,'Color','none','XColor','k','YColor','k','
        ZColor','k');
    for j = 1:length(infotext), set(infotext{j},'Color
        ','k'); end;
else
    set(handlefig,'Color',[0 0 0]);
    set(gca,'Color','none','XColor','w','YColor','w','
        ZColor','w');
    for j = 1:length(infotext), set(infotext{j},'Color
        ','w'); end;
end;


if ~vertices, set(handlevert,'Visible','off'); end
if ~edges, set(handlesurf,'EdgeColor','none'); end
if ~faces, set(handlesurf,'FaceColor','none'); end

if ~smoothshading, set(handlesurf,'FaceLighting','flat
    '); end
if ~ligt, set(handlesurf,'FaceLighting','none'); end

if isempty(lightpos),
    set(handleligt,'Position',get(handleaxes,'
        CameraPosition'));
else
    set(handleligt,'Position',lightpos);
end


if ~isempty(camtarg), set(gca,'CameraTarget',camtarg);
    end


if ~isempty(FL),
    NL = cross(VL(FL(:,2),:)-VL(FL(:,1),:),VL(FL(:,3)
        ,:)-VL(FL(:,1),:));
    normNL = sqrt(sum(NL.^2,2));
    NL = [NL(:,1)./normNL, NL(:,2)./normNL, NL(:,3)./
        normNL];
    FMP = [mean(reshape(VL(FL,1),size(FL,1),3),2),
        mean(reshape(VL(FL,2),size(FL,1),3),2), mean(
        reshape(VL(FL,3),size(FL,1),3),2)];

    handlenorm = quiver3(FMP(:,1),FMP(:,2),FMP(:,3),NL
        (:,1),NL(:,2),NL(:,3));
    set(handlenorm(1),'Color',[0.8 0 0]);

    if ~normals,
        set(handlenorm(1),'Visible','off');
        set(handlenorm(2),'Visible','off');
    end
else
    handlenorm = [];
end
```

```

mbutton = [0 0 0];
prevmousex = 0;
prevmousey = 0;

mymousedown = [...
'global handlefig mbutton prevmousex prevmousey, '...
'prevmousex = get(handlefig,''CurrentPoint'');
    prevmousey = prevmousex(2); prevmousex = prevmousex
    (1); '...
'button = get(handlefig,''SelectionType''); '...
'if button(1) == ''n'', mbutton = [1 0 0]; '...
'elseif button(1) == ''a'', mbutton = [0 0 1]; '...
'elseif button(1) == ''o'', '...
'else mbutton = [0 1 0]; end; '...
'clear button handlefig mbutton prevmousex prevmousey;
     '...
];

mymouseup = [...
'global mbutton, '...
'mbutton = [0 0 0]; '...
'clear mbutton; '...
];

boxrange = max([(max(VL(:,1))-min(VL(:,1))), (max(VL
    (:,2))-min(VL(:,2))), (max(VL(:,3))-min(VL(:,3)))])
    ;

mymousemove = [...
'global handlefig handleligt handleaxes handlenavi
```

```
movelwcam mbutton prevmousex prevmousey boxrange, '
...
'if any(mbutton), '...
'   x = get(handlefig,''CurrentPoint''); y = x(2); x =
    x(1); '...
'   p = get(handlefig,''Position''); '...
'   if mbutton(1) == 1, '...
'       set(gcf,''CurrentAxes'',handlenavi); '...
'       camorbit((prevmousex-x)/p(3)*360*2,(prevmousey
-y)/p(4)*360*2); '...
'       set(gcf,''CurrentAxes'',handleaxes); '...
'       camorbit((prevmousex-x)/p(3)*360*2,(prevmousey
-y)/p(4)*360*2); '...
'       if movelwcam, camlight(handleligt,''headlight'
'); end; '...
'   elseif mbutton(2) == 1, '...
'       set(gcf,''CurrentAxes'',handleaxes); '...
'       vax = get(gca,''CameraUpVector''); vax = vax/
norm(vax); '...
'       hax = cross(get(gca,''CameraUpVector''),get(
gca,''CameraPosition'')); hax = hax/norm(hax); '...
'       d = (prevmousex-x)*hax/(p(3)/boxrange)*1.5; '
...
'       set(gca,''CameraPosition'',get(gca,''
CameraPosition'')+d); set(gca,''CameraTarget'',get(
gca,''CameraTarget'')+d); '...
'       d = (prevmousey-y)*vax/(p(3)/boxrange)*1.5; '
...
'       set(gca,''CameraPosition'',get(gca,''
CameraPosition'')+d); set(gca,''CameraTarget'',get(
gca,''CameraTarget'')+d); '...
```

```
'    else '...
'        set(gcf,''CurrentAxes'',handleaxes); '...
'        if (y-prevmousey) > 0, camzoom(1+(y-prevmousey
   )/50); '...
'        elseif ((y-prevmousey) < 0) & ((prevmousey-y)
   < 50), camzoom(1-(prevmousey-y)/50); end; '...
'    end; '...
'    prevmousex = x; '...
'    prevmousey = y; '...
'end; '...
'clear p x y vax hax d handlefig handleligt handleaxes
    handlenavi movelwcam mbutton prevmousex prevmousey
    boxrange; '...
];

set(handlefig,'WindowButtonDownFcn',mymousedown);
set(handlefig,'WindowButtonUpFcn',mymouseup);

set(handlefig,'WindowButtonMotionFcn',mymousemove);

flippinview = 1;

mykeyboard = [...
'global handlefig handleaxes handlenavi handlesurf
   handlevert handlenorm handleligt boxrange info
   infotext navi navigator flippinview axesbox
   bfrlight proj background texture canmap facecolor
   edgecolor materiaal facealpha vertices edges faces
   normals ligt smoothshading movelwcam, '...
'key = get(handlefig,''CurrentCharacter''); '...
'if key == ''i'', '...
```

```
'    info = ~info; '...
'    if info, '...
'        for j = 1:length(infotext), set(infotext{j},''
    Visible'',''on''); end; '...
'    else '...
'        for j = 1:length(infotext), set(infotext{j},''
    Visible'',''off''); end; '...
'    end; '...
'end; '...
'if key == ''d'', '...
'    disp('' ''); disp('' ''); '...
'    if info, disp(''info = on''); else disp(''info =
    off''); end; '...
'    if navigator, disp(''navigator = on''); else disp(
    ''navigator = off''); end; '...
'    if axesbox, disp(''axesborder = on''); else disp('
    'axesbox = off''); end; '... '    if axeslab, disp('
    'axeslabels = on''); else disp(''axeslabels = off''
    ); end; '...
'    if proj, disp(''projection = perspective''); else
    disp(''projection = orthographic''); end; '...
'    if background==1, disp(''background = white'');
    elseif background==2, disp(''background = gray'');
    else disp(''background = black''); end; '...
'    if texture, disp(''texture = on''); else disp(''
    texture = off''); end; '...
'    disp([''facecolor = ['',num2str(facecolor(1)),'' '
    ',num2str(facecolor(2)),'' '',num2str(facecolor(3))
    ,'']'']); '...
'    disp([''edgecolor = ['',num2str(edgecolor(1)),'' '
    ',num2str(edgecolor(2)),'' '',num2str(edgecolor(3))
    ,'']'']); '...
'   disp(['facealpha = '',num2str(facealpha)]); '...
'   if vertices, disp(''vertices = on''); else disp(''
    vertices = off''); end; '...
'   if edges, disp(''edges = on''); else disp(''edges
    = off''); end; '...
'   if faces, disp(''faces = on''); else disp(''faces
    = off''); end; '...
'   if normals, disp(''normals = on''); else disp(''
    normals = off''); end; '...
'   [i,j] = view; disp([''viewpoint = ['',num2str(i),'
    ' '',num2str(j),'']'']); '...
'   disp([''cameraviewangle = '',num2str(get(
    handleaxes,''CameraViewAngle''))]); '...
'   j = get(handleaxes,''CameraTarget''); disp([''
    cameratarget = ['',num2str(j(1)),'' '',num2str(j(2)
    ),'' '',num2str(j(3)),'']'']); '...
'   if ligt, disp(''light = on''); else disp(''light =
     off''); end; '...
'   j = get(handleligt,''Position''); disp([''
    lightposition = ['',num2str(j(1)),'' '',num2str(j
    (2)),'' '',num2str(j(3)),'']'']); '...
'   if smoothshading, disp(''smoothshading = on'');
    else disp(''smoothshading = off''); end; '...
'   if movelwcam, disp(''movelight = on''); else disp(
    ''movelight = off''); end; '...
'   if bfrlight, disp(''backfacelighting = on''); else
     disp(''backfacelighting = off''); end; '...
'   disp('' ''); disp('' ''); '...
'end; '...
'if key == ''o'', '...
```

```
'    [fname,pname] = uigetfile(''*.mat'',''Open''); '
...
'    if pname == 0, i = {'' ''}; else i = who(''-file''
,[pname,fname]); end; '...
'    f_ex = 0; v_ex = 0; c_ex = 0; '...
'    for j = 1:length(i), '...
'         if length(i{j}) == 2 & i{j} == ''FL'', f_ex =
1; end; '...
'         if length(i{j}) == 2 & i{j} == ''VL'', v_ex =
1; end; '...
'         if length(i{j}) == 2 & i{j} == ''CL'', c_ex =
1; end; '...
'    end; '...
'    if f_ex & v_ex, '...
'         if c_ex, canmap = 1; else canmap = 0; end; '
...
'         eval([''load '''''',pname,fname,'''''''']); '
...
'         N = size(VL,1); sN = num2str(N); '...
'         ekspvorm = 0; for j = 1:length(sN), if sN(j)
== ''e'', ekspvorm = 1; end; end; '...
'         if ~ekspvorm, '...
'              if abs(round(log10(N)) - log10(N)) < 10*
eps, aantalkommas = floor(floor(log10(N + 1))/3);
else aantalkommas = floor(floor(log10(N))/3); end;
'...
'              for j = 1:aantalkommas, sN = strcat(sN(1:(
end-4*j+1)),'','',sN((end-4*j+2):end)); end; '...
'         end; '...
'         sNV = sN; '...
'         N = size(FL,1); sN = num2str(N); '...
'         ekspvorm = 0; for j = 1:length(sN), if sN(j)
    == ''e'', ekspvorm = 1; end; end; '...
'         if ~ekspvorm, '...
'              if abs(round(log10(N)) - log10(N)) < 10*
    eps, aantalkommas = floor(floor(log10(N + 1))/3);
    else aantalkommas = floor(floor(log10(N))/3); end;
    '...
'              for j = 1:aantalkommas, sN = strcat(sN(1:(
    end-4*j+1)),'','',sN((end-4*j+2):end)); end; '...
'         end; '...
'         sNF = sN; '...
'         set(infotext{63},''String'',strcat([sNV,''
    vertices   '', sNF, '' faces''])); '...
'         NL = cross(VL(FL(:,2),:)-VL(FL(:,1),:),VL(FL
    (:,3),:)-VL(FL(:,1),:)); '...
'         normNL = sqrt(sum(NL.^2,2)); '...
'         NL = [NL(:,1)./normNL, NL(:,2)./normNL, NL
    (:,3)./normNL]; '...
'         FMP = [mean(reshape(VL(FL,1),size(FL,1),3),2),
     mean(reshape(VL(FL,2),size(FL,1),3),2), mean(
    reshape(VL(FL,3),size(FL,1),3),2)]; '...
'         if ~isempty(handlenorm), delete(handlenorm);
    end; '...
'         handlenorm = quiver3(FMP(:,1),FMP(:,2),FMP
    (:,3),NL(:,1),NL(:,2),NL(:,3)); '...
'         handlenorm(2) = plot3(FMP(:,1),FMP(:,2),FMP
    (:,3),''.'',''Color'',[0.5 0 0]); '...
'         set(handlenorm(1),''Color'',[0.8 0 0]); '...
'         if ~normals, '...
'              set(handlenorm(1),''Visible'',''off''); '
    ...
```

```
'                set(handlenorm(2),''Visible'',''off''); '
    ...
'           end; '...
'           set(handlevert,''XData'',VL(:,1),''YData'',VL
    (:,2),''ZData'',VL(:,3)); '...
'           set(handlesurf,''Faces'',[],''Vertices'',VL,''
    Faces'',FL(:,[3 2 1])); '...
'           if canmap, set(handlesurf,''FaceVertexCData'',
    CL); end; '...
'           if canmap & texture, set(handlesurf,''
    FaceColor'',''interp''); end; '...
'           if ~canmap, '...
'                texture = 0; '...
'                set(handlesurf,''FaceColor'',facecolor); '
    ...
'           end; '...
'           if ~faces, set(handlesurf,''FaceColor'',''none
    ''); end; '...
'           axis image, view([135,24]); set(handleaxes,''
    CameraViewAngleMode'',''auto''); '...
'           axis image, axis vis3d, '...
'           if movelwcam, set(handleligt,''Position'',get(
    handleaxes,''CameraPosition'')); end; '...
'           boxrange = max([(max(VL(:,1))-min(VL(:,1))), (
    max(VL(:,2))-min(VL(:,2))), (max(VL(:,3))-min(VL
    (:,3)))]); '...
'           set(gcf,''CurrentAxes'',handlenavi); view
    (135,24); '...
'           set(gcf,''CurrentAxes'',handleaxes); '...
'     end; '...
'end; '...
```

```
'if key == ''g'', '...
'    for j = 1:length(navi), set(navi{j},''Visible'',''
   off''); end; '...
'    for j = 1:length(infotext), set(infotext{j},''
   Visible'',''off''); end; '...
'    print -dtiff screen, '...
'    if navigator, for j = 1:length(navi), set(navi{j},
   ''Visible'',''on''); end; end; '...
'    if info, for j = 1:length(infotext), set(infotext{
   j},''Visible'',''on''); end; end; '...
'end; '...
'if key == ''q'', '...
'    navigator = ~navigator; '...
'    if navigator, '...
'        for j = 1:length(navi), set(navi{j},''Visible'
   ',''on''); end; '...
'    else '...
'        for j = 1:length(navi), set(navi{j},''Visible'
   ',''off''); end; '...
'    end; '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'end; '...
'if key == ''j'', '...
'    flippinview = mod(flippinview,3) + 1; '...
'   N = get(handlesurf,''Vertices''); '...
'    set(handlesurf,''Vertices'',N(:,[3 1 2])); '...
'    set(handlevert,''XData'',N(:,3),''YData'',N(:,1),'
   'ZData'',N(:,2)); '...
'   N = [get(handlenorm(1),''XData''), get(handlenorm
   (1),''YData''), get(handlenorm(1),''ZData'')]; '...
'    set(handlenorm(1),''XData'',N(:,3),''YData'',N
(:,1),''ZData'',N(:,2)); '...
'    N = [get(handlenorm(2),''XData''); get(handlenorm
(2),''YData''); get(handlenorm(2),''ZData'')]; '...
'    set(handlenorm(2),''XData'',N(3,:),''YData'',N
(1,:),''ZData'',N(2,:)); '...
'    axis image; '...
'    if flippinview == 1, set(navi{6},''String'',''x'')
; set(navi{7},''String'',''y''); set(navi{8},''
String'',''z''); '...
'    elseif flippinview == 2, set(navi{6},''String'',''
z''); set(navi{7},''String'',''x''); set(navi{8},''
String'',''y''); '...
'    else set(navi{6},''String'',''y''); set(navi{7},''
String'',''z''); set(navi{8},''String'',''x''); end
; '... '    if axeslab, '...'        if flippinview
== 1, xlabel(''x''); ylabel(''y''); zlabel(''z'');
set(navi{6},''String'',''x''); set(navi{7},''String
'',''y''); set(navi{8},''String'',''z''); '...'
        elseif flippinview == 2, xlabel(''z'');
ylabel(''x''); zlabel(''y''); set(navi{6},''String'
',''z''); set(navi{7},''String'',''x''); set(navi
{8},''String'',''y''); '...'        else xlabel(''y'
'); ylabel(''z''); zlabel(''x''); set(navi{6},''
String'',''y''); set(navi{7},''String'',''z''); set
(navi{8},''String'',''x''); end; '...'    end; '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'end; '...
'if key == ''a'', '...
'    axesbox = ~axesbox; '...
'    if axesbox, axis on; '...
'    else axis off; end; '...
```

```
'end; '...
'if key == ''b'', '...
'    bfrlight = ~bfrlight; '...
'    if bfrlight, set(handlesurf,''BackFaceLighting'',''reverselit''); '...
'    else set(handlesurf,''BackFaceLighting'',''lit''); end; '...
'end; '...
'if key == ''p'', '...
'    proj = ~proj; '...
'    if proj, set(gca,''Projection'',''perspective'');'...
'    else set(gca,''Projection'',''orthographic''); end; ',...
'end; '...
'if key == ''w'', '...
'    background = background + 1; '...
'    if background > 3, background = 1; end; '...
'    if background == 1, '...
'         set(handlefig,''Color'',[1 1 1]); '...
'         set(gca,''Color'',''none'',''XColor'',''k'',''YColor'',''k'',''ZColor'',''k''); '...
'         set(navi{2},''EdgeColor'',''k''); '...
'         for j = 6:8, set(navi{j},''Color'',''k''); end; '...
'         for j = 1:length(infotext), set(infotext{j},''Color'',''k''); end; '...
'    elseif background == 2, '...
'         set(handlefig,''Color'',[0.8 0.8 0.8]); '...
'         set(gca,''Color'',''none'',''XColor'',''k'',''YColor'',''k'',''ZColor'',''k''); '...
'        set(navi{2},''EdgeColor'',''k''); '...
'        for j = 6:8, set(navi{j},''Color'',''k''); end
    ; '...
'        for j = 1:length(infotext), set(infotext{j},''
    Color'',''k''); end; '...
'   else '...
'        set(handlefig,''Color'',[0 0 0]); '...
'        set(gca,''Color'',''none'',''XColor'',''w'',''
    YColor'',''w'',''ZColor'',''w''); '...
'        set(navi{2},''EdgeColor'',''w''); '...
'        for j = 6:8, set(navi{j},''Color'',''w''); end
    ; '...
'        for j = 1:length(infotext), set(infotext{j},''
    Color'',''w''); end; '...
'   end; '...
'end; '...
'if key == ''t'', '...
'   if canmap, '...
'        texture = ~texture; '...
'        if texture, set(handlesurf,''FaceColor'',''
    interp''); '...
'        else set(handlesurf,''FaceColor'',facecolor);
    end; '...
'        if ~faces, set(handlesurf,''FaceColor'',''none
    ''); end; '...
'   end; '...
'end; '...
'if key == ''c'', '...
'   j = uisetcolor(facecolor,''Change surface color'')
    ; '...
'   if length(j) == 3, '...
'        facecolor = j; '...
'        if ~texture & faces, set(handlesurf,''
   FaceColor'',facecolor); end; '...
'    end; '...
'end; '...
'if key == ''k'', '...
'    j = uisetcolor(edgecolor,''Change edge color''); '
   ...
'    if length(j) == 3, '...
'        edgecolor = j; '...
'        if edges, set(handlesurf,''EdgeColor'',
   edgecolor); end; '...
'    end; '...
'end; '...
'if key == ''u'', '...
'    materiaal = mod(materiaal,3) + 1; '...
'    set(gcf,''CurrentObject'',handlesurf); '...
'    if materiaal == 1, material shiny; '...
'    elseif materiaal == 2, material dull; '...
'    else material metal; end; '...
'end; '...
'if key == ''+'', '...
'    if facealpha == 1, facealpha = 0.98; elseif
   facealpha == 0.98, facealpha = 0.95; elseif
   facealpha >= 0.05, facealpha = facealpha - 0.05;
   end; '...
'    set(handlesurf,''FaceAlpha'',facealpha); '...
'end; '...
'if key == ''-'', '...
'    if facealpha == 0.95, facealpha = 0.98; elseif
   facealpha == 0.98, facealpha = 1; elseif facealpha
```

```
    <= 0.9, facealpha = facealpha + 0.05; end; '...
'   set(handlesurf,''FaceAlpha'',facealpha); '...
'end; '...
'if key == ''v'', '...
'   vertices = ~vertices; '...
'   if vertices, set(handlevert,''Visible'',''on''); '
    ...
'   else set(handlevert,''Visible'',''off''); end; '
    ...
'end; '...
'if key == ''e'', '...
'   edges = ~edges; '...
'   if edges, set(handlesurf,''EdgeColor'',edgecolor);
     '...
'   else set(handlesurf,''EdgeColor'',''none''); end;
    '...
'end; '...
'if key == ''f'', '...
'   faces = ~faces; '...
'   if faces, '...
'       if texture, set(handlesurf,''FaceColor'',''
    interp''); '...
'       else set(handlesurf,''FaceColor'',facecolor);
    end; '...
'   else '...
'       set(handlesurf,''FaceColor'',''none''); '...
'   end; '...
'end; '...
'if key == ''n'', '...
'   normals = ~normals; '...
'   if normals, '...
'        set(handlenorm(1),''Visible'',''on''); '...
'        set(handlenorm(2),''Visible'',''on''); '...
'    else '...
'        set(handlenorm(1),''Visible'',''off''); '...
'        set(handlenorm(2),''Visible'',''off''); '...
'    end; '...
'end; '...
'if key == ''r'', '...
'    set(gcf,''CurrentAxes'',handlenavi); view(135,24);
     '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'    view(135,24); '...
'    set(handleaxes,''CameraViewAngleMode'',''auto'');
   '...
'    axis image, axis vis3d, '...
'    if movelwcam, set(handleligt,''Position'',get(
   handleaxes,''CameraPosition'')); end; '...
'end; '...
'if key == ''x'', '...
'    if flippinview == 1, N = [90 0]; elseif
   flippinview == 2, N = [0 0]; else N = [0 90]; end;
   '...
'    set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'    view(135,24); '...
'    set(handleaxes,''CameraViewAngleMode'',''auto'');
   '...
'    axis image, axis vis3d, '...
'    view(N); '...
'    if movelwcam, set(handleligt,''Position'',get(
   handleaxes,''CameraPosition'')); end; '...
```

```
'end; '...
'if key == ''X'', '...
'   if flippinview == 1, N = [-90 0]; elseif
    flippinview == 2, N = [-180 0]; else N = [-180
    -90]; end; '...
'   set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'   set(gcf,''CurrentAxes'',handleaxes); '...
'   view(135,24); '...
'   set(handleaxes,''CameraViewAngleMode'',''auto'');
    '...
'   axis image, axis vis3d, '...
'   view(N); '...
'   if movelwcam, set(handleligt,''Position'',get(
    handleaxes,''CameraPosition'')); end; '...
'end; '...
'if key == ''y'', '...
'   if flippinview == 1, N = [0 0]; elseif flippinview
     == 2, N = [0 90]; else N = [90 0]; end; '...
'   set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'   set(gcf,''CurrentAxes'',handleaxes); '...
'   view(135,24); '...
'   set(handleaxes,''CameraViewAngleMode'',''auto'');
    '...
'   axis image, axis vis3d, '...
'   view(N); '...
'   if movelwcam, set(handleligt,''Position'',get(
    handleaxes,''CameraPosition'')); end; '...
'end; '...
'if key == ''Y'', '...
'   if flippinview == 1, N = [-180 0]; elseif
    flippinview == 2, N = [-180 -90]; else N = [-90 0];
    end; '...
'    set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'    view(135,24); '...
'    set(handleaxes,''CameraViewAngleMode'',''auto'');
   '...
'    axis image, axis vis3d, '...
'    view(N); '...
'    if movelwcam, set(handleligt,''Position'',get(
   handleaxes,''CameraPosition'')); end; '...
'end; '...
'if key == ''z'', '...
'    if flippinview == 1, N = [0 90]; elseif
   flippinview == 2, N = [90 0]; else N = [0 0]; end;
   '...
'    set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'    view(135,24); '...
'    set(handleaxes,''CameraViewAngleMode'',''auto'');
   '...
'    axis image, axis vis3d, '...
'    view(N); '...
'    if movelwcam, set(handleligt,''Position'',get(
   handleaxes,''CameraPosition'')); end; '...
'end; '...
'if key == ''Z'', '...
'    if flippinview == 1, N = [-180 -90]; elseif
   flippinview == 2, N = [-90 0]; else N = [-180 0];
   end; '...
'    set(gcf,''CurrentAxes'',handlenavi); view(N); '...
'    set(gcf,''CurrentAxes'',handleaxes); '...
'     view(135,24);  '...
'     set(handleaxes,''CameraViewAngleMode'',''auto'');
    '...
'     axis image,  axis vis3d,  '...
'     view(N);  '...
'     if movelwcam,  set(handleligt,''Position'',get(
    handleaxes,''CameraPosition''));  end;  '...
'end;  '...
'if key == ''l'',  '...
'     ligt = ~ligt;  '...
'     if ligt,  '...
'          if smoothshading,  '...
'               set(handlesurf,''FaceLighting'',''gouraud'
    ');  '...
'          else  '...
'               set(handlesurf,''FaceLighting'',''flat'');
     '...
'          end;  '...
'     else  '...
'          set(handlesurf,''FaceLighting'',''none'');  '
    ...
'     end;  '...
'end;  '...
'if key == ''s'',  '...
'     smoothshading = ~smoothshading;  '...
'     if ligt,  '...
'          if smoothshading,  '...
'               set(handlesurf,''FaceLighting'',''gouraud'
    ');  '...
'          else  '...
'               set(handlesurf,''FaceLighting'',''flat'');
```

```
    '...
'        end; '...
'    end; '...
'end; '...
'if key == ''m'', '...
'    movelwcam = ~movelwcam; '...
'    if movelwcam, set(handleligt,''Position'',get(
   handleaxes,''CameraPosition'')); end; '...
'end; '...
'clear key f_ex v_ex c_ex aantalkommas ekspvorm fname
   pname sN sNF sNV N FMP NL normNL i j handlefig
   handleaxes handlenavi handlesurf handlevert
   handlenorm handleligt boxrange info infotext navi
   navigator flippinview axesbox bfrlight proj
   background texture canmap facecolor edgecolor
   materiaal facealpha vertices edges faces normals
   ligt smoothshading movelwcam, '...
];


set(handlefig,'KeyPressFcn',mykeyboard);
```

==

```
function uu = gmprLaplace( S1,S2, N )
% FUNCTION GMPRLAPLACE
% Iteratively solves the Laplace equation over a
   rectangular domain

if isempty(S1) | isempty(S2)
    error('Input vectdors cannot be empty');
    return
end
```

```

if size(S1,1) == 1
else
    S1 = transpose(S1);
end
if size(S2,1) == 1
else
    S2 = transpose(S2);
end

if length(S1) ~= length(S2)
    error('ERROR: Input dimensions must agree.');
    return
end
if N<1
    uu = [S1; S2];
    return
end

N = N+2;
L = length(S1);
data = zeros(N, L);
data(:,:)=NaN;
data(1,:)=S1;
data(N,:)=S2;

U1=L; U2=1;
for i=1:L
    if isnan(S1(i))
    else
        if i<U1 U1=i; end
```

```
            if i>U2 U2=i; end
        end
end

U3=L; U4=1;
for i=1:L
        if isnan(S2(i))
        else
            if i<U3 U3=i; end
            if i>U4 U4=i; end
        end
end

if (U2-U1+1)<2
        uu=[S1;S2];
        return
end
if (U4-U3+1)<2
        uu=[S1;S2];
        return
end

if U1<U3 U1=U3; else U3=U1; end
if U2>U4 U2=U4; else U4=U2; end

C = U2-U1+1;
UU = zeros(N,C);

v = 1/(N-1);
h = 1/(C-1);
B12 = S1(U1:U2);
```

```
B34 = S2(U3:U4);
B13 = U1:(U3-U1)/(N-1):U3;
B24 = U2:(U4-U2)/(N-1):U4;

if isempty(B12)
      B12 = S1(U1)*(ones(1,size(S1,1)));
end
if isempty(B34)
      B34 = S2(U3)*(ones(1,size(S1,1)));
end
if isempty(B13)
      B13 = S1(U1)*(ones(1,N));
end
if isempty(B24)
      B24 = S1(U2)*(ones(1,N));
end

UU(1,:)     = B12;
UU(end,:)   = B34;
UU(:,1)     = transpose(B13);
UU(:,end)   = transpose(B24);

for k=1:100
      tmp = UU;
      for r=2:N-1
            for c=2:C-1
                  tmp(r,c)=...
                        (1/4)*(UU(r-1,c)+UU(r+1,c)+UU(r,c-1)+
                            UU(r,c+1));
            end
      end
```

```
        UU=tmp;
end

data(2:end-1,U1:U2) = UU(2:end-1,:);
uu = data;

if U1==1
        leadingNaNs = [];
else
        leadingNaNs = ones(N, U1-1);
        leadingNaNs(:,:) = NaN;
end
if U2 == length(S1)
        trailingNaNs = [];
else
        trailingNaNs = ones(N, length(S1)-U2);
        trailingNaNs(:,:) = NaN;
end

uu = [ leadingNaNs uu trailingNaNs ];
```

==

```
function gmprEstimateErrors()

addpath = [pwd '\Data'];
path(path, addpath);

N=0;
pdefftfile   = ['pde' num2str(N) 'fftdata'];
pdedctfile   = ['pde' num2str(N) 'dctdata'];
pdedwtfile   = ['pde' num2str(N) 'dwtdata'];

```

```
datafile  ='Data01.txt';
scalefile ='Data01Scale.txt';

F=5;
for i=1:F
    tic
    datafile(6)  = num2str(i);
    scalefile(6) = num2str(i);

    for quality = [ 100 90 80 70 60 50 40 30 20 10 5 ]

        disp(['FFT compression quality=' num2str(
           quality) ', PDE interpolation=' num2str(N)
           ]);
        if N==0
            compressedfile = gmprCompressFFT( datafile
               , scalefile, quality );
            [fftdata{i}{quality}, f1, f2 ] =
               gmprUncompressFFT( compressedfile, N );
        else
            compressedfile = ['cFFT' num2str(quality)
               datafile ];
            [fftdata{i}{quality}, f1, f2 ] =
               gmprUncompressFFT( compressedfile, N );
        end

        disp(['DCT compression quality=' num2str(
           quality) ', PDE interpolation=' num2str(N)
           ]);
        if N==0
                    compressedfile = gmprCompressDCT(
                                    datafile , scalefile , quality );
                [dctdata{i}{quality}, f3, f4] =
                    gmprUncompressDCT( compressedfile , N );
            else
                compressedfile = ['cDCT' num2str(quality)
                    datafile ];
                [dctdata{i}{quality}, f3, f4 ] =
                    gmprUncompressDCT( compressedfile , N );
            end

            disp(['DWT compression quality=' num2str(
                quality) ', PDE interpolation=' num2str(N)
                ]);
            if N==0
                                compressedfile = gmprCompressDWT(
                                    datafile , scalefile , quality );
                [dwtdata{i}{quality}, f5, f6] =
                    gmprUncompressDWT( compressedfile , N );
            else
                compressedfile = ['cDWT' num2str(quality)
                    datafile ];
                [dwtdata{i}{quality}, f5, f6 ] =
                    gmprUncompressDWT( compressedfile , N );
            end
        end
        dt = toc;
        fHours   = ( ( F - i) *dt )/3600;
        nHours   = floor( fHours );
        fMinutes = (fHours - nHours ) * 60;
        nMinutes = floor( fMinutes );
        nSeconds = floor( (fMinutes - nMinutes ) * 60);
```

```
    if fHours > 24
        days  = floor( fHours/24 );
        hours = floor( (fHours/24 - days)*24 );
    else
        days  = 0;
        hours = floor( fHours );
    end
    disp( [ 'End of file ' num2str(i) '. Remaning time
        is ' num2str(days) ' days ' num2str(floor(
       hours)) ' hours ' num2str(nMinutes) ' minutes '
        num2str(nSeconds) ' seconds'] );
end


save pdefftfile fftdata
save pdedctfile dctdata
save pdedwtfile dwtdata

copyfile( 'pdefftfile.mat', ['Data\' pdefftfile '.mat'
    ]);
copyfile( 'pdedctfile.mat', ['Data\' pdedctfile '.mat'
    ]);
copyfile( 'pdedwtfile.mat', ['Data\' pdedwtfile '.mat'
    ]);
delete('pdefftfile.mat');
delete('pdedctfile.mat');
delete('pdedwtfile.mat');

if N==0
    load Data\pde0fftdata.mat
    load Data\pde0dctdata.mat
    load Data\pde0dwtdata.mat
else
    load Data\pde3fftdata.mat
    load Data\pde3dctdata.mat
    load Data\pde3dwtdata.mat
end

for i=1:F
    if N==0
        datafile  ='Data01.txt';
        scalefile ='Data01Scale.txt';
    else
        datafile  ='Data01sf.txt';
        scalefile ='Data01sfScale.txt';
    end

    datafile(6)  = num2str(i);
    scalefile(6) = num2str(i);
    data = load( datafile );

    for quality = [ 100 90 80 70 60 50 40 30 20 10 5 ]
        [R C] = size(data);
        [r c] = size(fftdata{i}{quality});
        if r<R R=r; end
        if c<C C=c; end

        data1 = data(1:R, 1:C);
        data2 = fftdata{i}{quality}(1:R, 1:C);

        Efft{i}{quality}    = data2-data1;
        RMSEfft{i}{quality} = gmprRMSE( data1 , data2 )
```

```
            ;

        [R C] = size(data);
        [r c]  = size(dctdata{i}{quality});
        if r<R R=r; end
        if c<C C=c; end

        data1 = data(1:R, 1:C);
        data2 = dctdata{i}{quality}(1:R, 1:C);

        Edct{i}{quality}    = data2-data1;
        RMSEdct{i}{quality} = gmprRMSE( data1 , data2 )
            ;

        [R C] = size(data);
        [r c]  = size(dwtdata{i}{quality});
        if r<R R=r; end
        if c<C C=c; end

        data1 = data(1:R, 1:C);
        data2 = dwtdata{i}{quality}(1:R, 1:C);

        Edwt{i}{quality}    = data2-data1;
        RMSEdwt{i}{quality} = gmprRMSE( data1 , data2 )
            ;

    end
end

if N==0
    Efft0 = Efft;
```

```
        save Data\Efft0 Efft0
        Edct0 = Edct;
        save Data\Edct0 Edct0
        Edwt0 = Edwt;
        save Data\Edwt0 Edwt0
        RMSEfft0 = RMSEfft;
        save Data\RMSEfft0 RMSEfft0
        RMSEdct0 = RMSEdct;
        save Data\RMSEdct0 RMSEdct0
        RMSEdwt0 = RMSEdwt;
        save Data\RMSEdwt0 RMSEdwt0
end
if N==3
        Efft3 = Efft;
        save Data\Efft3 Efft3
        Edct3 = Edct;
        save Data\Edct3 Edct3
        Edwt3 = Edwt;
        save Data\Edwt3 Edwt3
        RMSEfft3 = RMSEfft;
        save Data\RMSEfft3 RMSEfft3
        RMSEdct3 = RMSEdct;
        save Data\RMSEdct3 RMSEdct3
        RMSEdwt3 = RMSEdwt;
        save Data\RMSEdwt3 RMSEdwt3
end

addpath = [pwd '\Data'];
path(path, addpath);

N=0;
```

```
if N==0
            load RMSEfft0
            load RMSEdct0
            load RMSEdwt0
            RMSEfft=RMSEfft0;
            RMSEdct=RMSEdct0;
            RMSEdwt=RMSEdwt0;
end
if N==3
            load RMSEfft3
            load RMSEdct3
            load RMSEdwt3
            RMSEfft=RMSEfft3;
            RMSEdct=RMSEdct3;
            RMSEdwt=RMSEdwt3;
end

quality=[ 100 90 80 70 60 50 40 30 20 10 5 ];
RR1=[]; RR2=[]; RR3=[];
SS1=[]; SS2=[]; SS3=[];
for i=1:5
      S1=[]; S2=[]; S3=[]; R1=[]; R2=[]; R3=[];
      for k=1:11
            S1=[S1 num2str( RMSEfft{i}{quality(k)}) ' '];
            R1=[R1 RMSEfft{i}{quality(k)}];
            S2=[S2 num2str( RMSEdct{i}{quality(k)}) ' '];
            R2=[R2 RMSEdct{i}{quality(k)}];
            S3=[S3 num2str( RMSEdwt{i}{quality(k)}) ' '];
            R3=[R3 RMSEdwt{i}{quality(k)}];
      end
      RR1=[RR1; R1]; RR2=[RR2;R2]; RR3=[RR3;R3];
```

```
    SS1=[SS1 S1];   SS2=[SS2 S2]; SS3=[SS3 S3];
end

compression=[ '100';' 90';' 80';' 70';' 60';' 50';' 40
    ';' 30';' 20';' 10';'  0'];
figure,plot(quality,mean(RR1,1),'b-','Linewidth',2);
hold on
plot(quality,mean(RR2,1),'r-', 'Linewidth',2);
plot(quality,mean(RR3,1),'g-', 'Linewidth',2);

xlabel(['Quality of compression']);
ylabel(['Error RMSE in mm']);
if N==0
    legend('DFT', 'DCT', 'DWT' );
    title('Average compression errors');
else
    legend('DFT with PDE','DCT with PDE', 'DWT with
       PDE');
    title('PDE based average compression errors');
end
hold off
disp('Done!!!');

return

%  Compression rates as calculated, see file "
    filesizes.xlsx":
%  AVERAGE SIZE                                    4589
        19125    1348     5349     473      425      378
         331      284      237      190      143      96
          49       25
```

```
% COMPRESSION RATE OBJ SPARSE
                          0.897    0.907    0.918    0.928
  0.938    0.948    0.959    0.969    0.979    0.989
  0.994
% COMPRESSION RATE OBJ SF
                               0.975             0.978    0.980
  0.983    0.985    0.988    0.990    0.993    0.995
  0.997    0.999
% COMPRESSION RATE TEXT SPARSE
                          0.649    0.685    0.720    0.755
  0.789    0.824    0.859    0.894    0.929    0.964
  0.981
% COMPRESSION RATE TEXT SF
                               0.912             0.921    0.929
  0.938    0.947    0.956    0.964    0.973    0.982
  0.991    0.995
%
% AVERAGE SIZE                                                4589
       19125     1348      5349      472       424       377
        330       283       236       189       142        95
         47        24
% COMPRESSION RATE OBJ SPARSE
                          0.897    0.908    0.918    0.928
  0.938    0.949    0.959    0.969    0.979    0.990
  0.995
% COMPRESSION RATE OBJ SF
                               0.975             0.978    0.980
  0.983    0.985    0.988    0.990    0.993    0.995
  0.998    0.999
% COMPRESSION RATE TEXT SPARSE
                          0.650    0.685    0.720    0.755
```

```
    0.790     0.825     0.860     0.895     0.930     0.965
    0.982
%   COMPRESSION RATE TEXT SF
                             0.912          0.921     0.929
    0.938     0.947     0.956     0.965     0.974     0.982
    0.991     0.996
%
%   AVERAGE SIZE                                            4589
        19125     1348      5349      475       433       392
    351       309       268       226       185       144       102
          81
%   COMPRESSION RATE OBJ SPARSE
                        0.897     0.906     0.915     0.924
    0.933     0.942     0.951     0.960     0.969     0.978
    0.982
%   COMPRESSION RATE OBJ SF
                             0.975          0.977     0.980
    0.982     0.984     0.986     0.988     0.990     0.992
    0.995     0.996
%   COMPRESSION RATE TEXT SPARSE
                        0.648     0.679     0.709     0.740
    0.771     0.801     0.832     0.863     0.893     0.924
    0.940
%   COMPRESSION RATE TEXT SF
                             0.911          0.919     0.927
    0.934     0.942     0.950     0.958     0.965     0.973
    0.981     0.985

%from file "filesizes.xlsx":
cmpFFT=[
 0.897     0.907     0.918     0.928     0.938     0.948     0.959
```

```
         0.969    0.979    0.989    0.994;
0.975    0.978    0.980    0.983    0.985    0.988    0.990
         0.993    0.995    0.997    0.999;
0.649    0.685    0.720    0.755    0.789    0.824    0.859
         0.894    0.929    0.964    0.981;
0.912    0.921    0.929    0.938    0.947    0.956    0.964
         0.973    0.982    0.991    0.995];

cmpDCT=[
0.897    0.908    0.918    0.928    0.938    0.949    0.959
         0.969    0.979    0.990    0.995;
0.975    0.978    0.980    0.983    0.985    0.988    0.990
         0.993    0.995    0.998    0.999;
0.650    0.685    0.720    0.755    0.790    0.825    0.860
         0.895    0.930    0.965    0.982;
0.912    0.921    0.929    0.938    0.947    0.956    0.965
         0.974    0.982    0.991    0.996];

cmpDWT=[
0.897    0.906    0.915    0.924    0.933    0.942    0.951
         0.960    0.969    0.978    0.982;
0.975    0.977    0.980    0.982    0.984    0.986    0.988
         0.990    0.992    0.995    0.996;
0.648    0.679    0.709    0.740    0.771    0.801    0.832
         0.863    0.893    0.924    0.940;
0.911    0.919    0.927    0.934    0.942    0.950    0.958
         0.965    0.973    0.981    0.985];

quality = [ 100 90 80 70 60 50 40 30 20 10 5 ];

figure ,plot(quality ,cmpFFT(1,:),'bo-','Linewidth',2);
```

```
hold on
plot(quality,cmpDCT(1,:),'r*-', 'Linewidth',2);
plot(quality,cmpDWT(1,:),'gd-', 'Linewidth',2);
xlabel(['Quality of compression']);
ylabel(['Compression rate in %']);

plot(quality,cmpFFT(3,:),'b-','Linewidth',2);
plot(quality,cmpDCT(3,:),'r-', 'Linewidth',2);
plot(quality,cmpDWT(3,:),'g-', 'Linewidth',2);
xlabel(['Quality of compression']);
ylabel(['Compression rate in %']);

legend('FFT (obj)', 'DCT (obj)', 'DWT (obj)' ,'FFT (
    txt)', 'DCT (txt)', 'DWT (txt)' );
title('Compression rates compared to OBJ and TEXT file
    formats');
hold off

figure,plot(quality,cmpFFT(2,:),'bo-','Linewidth',2);
hold on
plot(quality,cmpDCT(2,:),'r*-', 'Linewidth',2);
plot(quality,cmpDWT(2,:),'gd-', 'Linewidth',2);
xlabel(['Quality of compression in %']);
ylabel(['Compression rate in %']);

plot(quality,cmpFFT(4,:),'b-','Linewidth',2);
plot(quality,cmpDCT(4,:),'r-', 'Linewidth',2);
plot(quality,cmpDWT(4,:),'g-', 'Linewidth',2);
xlabel(['Quality of compression']);
ylabel(['Compression rate in %']);
legend('FFT with PDE (obj)', 'DCT with PDE (obj)', '
    DWT with PDE (obj)', 'FFT with PDE (txt)', 'DCT
    with PDE (txt)', 'DWT with PDE (txt)' );
title('PDE based compression rates compared to OBJ and
    TXT file formats');
hold off

load Data\Efft0
errorsurface = Efft0{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01Scale.txt');

load Data\Edct0
errorsurface = Edct0{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01Scale.txt');

load Data\Edwt0
errorsurface = Edwt0{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01Scale.txt');

load Data\Efft3
errorsurface = Efft3{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01sfScale.txt')
    ;

load Data\Edct3
errorsurface = Edct3{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01sfScale.txt')
```

```
    ;

load Data\Edwt3
errorsurface = Edwt3{1}{50};
save errorsurface.txt errorsurface -ASCII
gmprLoadData( 'errorsurface.txt', 'Data01sfScale.txt')
    ;

gmprLoadData('pde3cFFT50Data01.txt','
    pde3cFFT50Data01Scale.txt');
gmprLoadData('pde3cDCT50Data01.txt','
    pde3cDCT50Data01Scale.txt');
gmprLoadData('pde3cDWT50Data01.txt','
    pde3cDWT50Data01Scale.txt');

gmprDrawPlane;
data=load('Data\Data07.txt');
scale = load( 'Data\Data07Scale.txt');
depthScale1 = scale(1);
depthScale2 = scale(2);
maxZ =      max(max(data));
minZ =      min(min(data));
X = size(data,1);
Y = size(data,2);
maxX = X*depthScale1;
maxY = Y*depthScale2;
minX = 0;
minY = 0;

box = [ minX maxX minY maxY minZ maxZ ];
gmprDrawBox3d( box );
```

```

data=load('Data01.txt');
size(data);
singlestripe=data(41,:);
singlestripe=singlestripe(28:683);

function [a0, a6, an, bn] = getfouriercoeff(
    singlestripe )
L=length(singlestripe)-1;
x=0:360/L:360;

d = fft(singlestripe);
m = length(singlestripe);
M = floor((m+1)/2);

a0 = d(1)/m;
an = 2*real(d(2:M))/m;
a6 = d(M+1)/m;
bn = -2*imag(d(2:M))/m;

n = 1:length(an);
y = a0 + an*cos(2*pi*n'*x/360) ...
        + bn*sin(2*pi*n'*x/360) ...
        + a6*cos(2*pi*6*x/360);

figure, plot(x,singlestripe,'bo'),
title('{\bf DFT reconstruction}')
hold on
plot(x,y,'c-','Linewidth',2)
legend('Raw data','DFT reconstructed')

```

```
function B = getdctcoeff( singlestripe )
L=length(singlestripe)-1;
x=0:360/L:360;
figure, plot(x,singlestripe,'ro')
title('{\bf DCT reconstruction}')
hold on

B = dct( singlestripe );
y = idct( B );

plot(x,y,'k-','Linewidth',3)
legend('Raw data','DCT Reconstructed')
hold off


[C,L] = wavedec(singlestripe,3,'db1');
cA3 = appcoef(C,L,'db1',3);
cD3 = detcoef(C,L,3);
cD2 = detcoef(C,L,2);
cD1 = detcoef(C,L,1);
[cD1,cD2,cD3] = detcoef(C,L,[1,2,3]);

A3 = wrcoef('a',C,L,'db1',3);
D1 = wrcoef('d',C,L,'db1',1);
D2 = wrcoef('d',C,L,'db1',2);
D3 = wrcoef('d',C,L,'db1',3);

figure,title('DWT')
subplot(2,2,1); plot(A3); title('Approximation A3')
subplot(2,2,2); plot(D1); title('Detail D1')
subplot(2,2,3); plot(D2); title('Detail D2')
subplot(2,2,4); plot(D3); title('Detail D3')

A0 = waverec(C,L,'db1');
figure,plot(x,singlestripe,'go')
title('{\bf DWT reconstruction}')
hold on
plot(x,A0,'k-','Linewidth',2)
legend('Raw data','DWT reconstructed')
hold off
```

==

```
function r=gmprRMSE(data,estimate)
% Function to calculate root mean square error from a
    data vector or matrix
% and the corresponding estimates.

I = ~isnan(data) & ~isnan(estimate);
data = data(I); estimate = estimate(I);

r=sqrt(sum((data(:)-estimate(:)).^2)/numel(data));
```

==

3D data compresssion file sizes

original file	OBJ sparse	OBJ SF	TEXT	TEXT SF
Data01.txt	4992	20879	1428	5667
Data02.txt	3903	16160	1110	4404
Data03.txt	5761	24224	1653	6567
Data04.txt	5063	21184	1446	5739
Data05.txt	3228	13180	1104	4368
AVERAGE SIZE	**4589**	**19125**	**1348**	**5349**
COMPRESSION RATE OBJ SPARSE				
COMPRESSION RATE OBJ SF				
COMPRESSION RATE TEXT SPARSE				
COMPRESSION RATE TEXT SF				

original file	OBJ sparse	OBJ SF	TEXT	TEXT SF
Data01.txt	4992	20879	1428	5667
Data02.txt	3903	16160	1110	4404
Data03.txt	5761	24224	1653	6567
Data04.txt	5063	21184	1446	5739
Data05.txt	3228	13180	1104	4368
AVERAGE SIZE	**4589**	**19125**	**1348**	**5349**
COMPRESSION RATE OBJ SPARSE				
COMPRESSION RATE OBJ SF				
COMPRESSION RATE TEXT SPARSE				
COMPRESSION RATE TEXT SF				

original file	OBJ sparse	OBJ SF	TEXT	TEXT SF
Data01.txt	4992	20879	1428	5667
Data02.txt	3903	16160	1110	4404
Data03.txt	5761	24224	1653	6567
Data04.txt	5063	21184	1446	5739
Data05.txt	3228	13180	1104	4368
AVERAGE SIZE	**4589**	**19125**	**1348**	**5349**
COMPRESSION RATE OBJ SPARSE				
COMPRESSION RATE OBJ SF				
COMPRESSION RATE TEXT SPARSE				
COMPRESSION RATE TEXT SF				

FFT 100	FFT 90	FFT 80	FFT 70	FFT 60	FFT 50	FFT 40
511	460	409	358	307	257	205
404	363	323	283	243	203	163
588	529	470	411	353	295	236
520	467	415	364	312	261	209
340	306	272	238	204	171	137
473	**425**	**378**	**331**	**284**	**237**	**190**
0.897	0.907	0.918	0.928	0.938	0.948	0.959
0.975	0.978	0.980	0.983	0.985	0.988	0.990
0.649	0.685	0.720	0.755	0.789	0.824	0.859
0.912	0.921	0.929	0.938	0.947	0.956	0.964

DCT 100	DCT 90	DCT 80	DCT 70	DCT 60	DCT 50	DCT 40
511	459	408	357	306	255	204
403	363	323	282	242	202	161
587	528	469	411	352	293	235
519	467	415	363	311	260	208
340	305	271	237	204	170	136
472	**424**	**377**	**330**	**283**	**236**	**189**
0.897	0.908	0.918	0.928	0.938	0.949	0.959
0.975	0.978	0.980	0.983	0.985	0.988	0.990
0.650	0.685	0.720	0.755	0.790	0.825	0.860
0.912	0.921	0.929	0.938	0.947	0.956	0.965

DWT 100	DWT 90	DWT 80	DWT 70	DWT 60	DWT 50	DWT 40
514	468	424	379	334	290	245
406	371	335	300	266	229	194
590	539	487	436	384	333	281
521	476	430	385	339	294	248
342	312	282	253	223	193	163
475	**433**	**392**	**351**	**309**	**268**	**226**
0.897	0.906	0.915	0.924	0.933	0.942	0.951
0.975	0.977	0.980	0.982	0.984	0.986	0.988
0.648	0.679	0.709	0.740	0.771	0.801	0.832
0.911	0.919	0.927	0.934	0.942	0.950	0.958

FFT 30	FFT 20	FFT 10	FFT 5
155	104	53	27
122	82	42	22
178	119	61	31
157	105	53	28
103	69	35	19
143	**96**	**49**	**25**
0.969	0.979	0.989	0.994
0.993	0.995	0.997	0.999
0.894	0.929	0.964	0.981
0.973	0.982	0.991	0.995

DCT 30	DCT 20	DCT 10	DCT 5
153	102	51	26
121	81	40	20
176	118	59	30
156	104	52	26
102	68	34	17
142	**95**	**47**	**24**
0.969	0.979	0.990	0.995
0.993	0.995	0.998	0.999
0.895	0.930	0.965	0.982
0.974	0.982	0.991	0.996

DWT 30	DWT 20	DWT 10	DWT 5
200	155	111	88
158	123	88	70
230	179	127	101
203	158	112	89
134	104	74	59
185	**144**	**102**	**81**
0.960	0.969	0.978	0.982
0.990	0.992	0.995	0.996
0.863	0.893	0.924	0.940
0.965	0.973	0.981	0.985