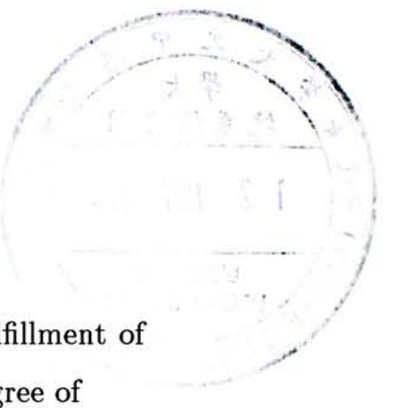
ACTIVE HAPTIC EXPLORATION FOR 3D SHAPE RECONSTRUCTION

By

FUNG WAI KEUNG



A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

MASTER OF PHILOSOPHY

in

the Department of Systems Engineering and Engineering Management

The Chinese University of Hong Kong

June, 1996

Contents

Acknowledgements viii

Abstract 1

1 Overview 3

1.1 Tactile Sensing in Human and Robot 4

1.1.1 Human Hands and Robotic Hands 4

1.1.2 Mechanoreceptors in skin and Tactile Sensor Arrays . 7

1.2 Motivation 12

1.3 Objectives 13

1.4 Related Work 14

1.4.1 Using Vision Alone 15

1.4.2 Integration of Vision and Touch 15

1.4.3 Using Touch Sensing Alone 17

1.4.3.1 Ronald S. Fearing's Work 18

1.4.3.2 Peter K. Allen's Work 22

1.5 Outline 26

2 Geometric Models 27

2.1 Introduction 27

2.2 Superquadrics 27

2.2.1 2D Superquadrics 27

2.2.2 3D Superquadrics 29

2.3 Model Recovery of Superquadric Models 31

2.3.1 Problem Formulation 31

2.3.2 Least Squares Optimization 33

2.4 Free-Form Deformations 34

2.4.1 Bernstein Basis 36

2.4.2 B-Spline Basis 38
2.5 Other Geometric Models 41
2.5.1 Generalized Cylinders 41
2.5.2 Hyperquadrics 42
2.5.3 Polyhedral Models 44
2.5.4 Function Representation 45

3 Sensing Strategy **54**
3.1 Introduction 54
3.2 Sensing Algorithm 55
3.2.1 Assumption of objects 55
3.2.2 Haptic Exploration Procedures 56
3.3 Contour Tracing 58
3.4 Tactile Sensor Data Preprocessing 59
3.4.1 Data Transformation and Sensor Calibration 60
3.4.2 Noise Filtering 61
3.5 Curvature Determination 64
3.6 Step Size Determination 73

4 3D Shape Reconstruction **80**
4.1 Introduction 80
4.2 Correspondence Problem 81
4.2.1 Affine Invariance Property of B-splines 84
4.2.2 Point Inversion Problem 87
4.3 Parameter Triple Interpolation 91
4.4 3D Object Shape Reconstruction 94
4.4.1 Heuristic Approach 94
4.4.2 Closed Contour Recovery 97
4.4.3 Control Lattice Recovery 102

5 Implementation **105**
5.1 Introduction 105
5.2 Implementation Tool – MATLAB 105
5.2.1 Optimization Toolbox 107
5.2.2 Splines Toolbox 108
5.3 Geometric Model Implementation 109
5.3.1 FFD Examples 111

5.4 Shape Reconstruction Implementation 112
5.5 3D Model Reconstruction Examples 120
5.5.1 Example 1 120
5.5.2 Example 2 121

6 Conclusion **128**
6.1 Future Work 129

Appendix **133**

Bibliography **146**

List of Figures

1.1 Ultrasonic tactile sensor array construction and operation mechanism 11

2.1 Superquadric curves with different ϵ 28
2.2 Superquadric model with different ϵ_1 and ϵ_2 49
2.3 Non-uniform and regular parameterizations of the same cube. 50
2.4 Bernstein basis when $n = 5$ 50
2.5 The de Casteljau Algorithm – The cubic case with $t = \frac{1}{3}$ and $t \in [0, 1]$ 51
2.6 The imaginary parallelepiped for FFD. 51
2.7 B-spline basis for different knot vectors. 52
2.7(a) Using clamped uniform knot vector 52
2.7(b) Using unclamped uniform knot vector 52
2.7(c) Using nonuniform knot vector 52
2.7(d) Using knot vector with multiple knots 52
2.8 Several Hyperquadric models 53
2.9 The Winged Edge model (Courtesy from "Geometric and Solid Modeling" by Christoph M. Hoffmann, published by Morgan Kaufmann [1]) 53

3.1 Tactile sensor data preprocessing procedures 60
3.2 Coordinate frame defined in the tactile sensor array 61
3.3 Raw tactile sensor data. 64
3.4 Adaptive Wiener Filter and Median Filter results. 65
3.5 Result of Wiener filter→Median filter. 66
3.6 Result of Median filter→Wiener filter. 67
3.7 Original and approximated s_1 step size component 77
3.8 Original and approximated s_2 step size component 77

3.9 Approximated step size function $s(\kappa, \triangle\kappa)$ 78
3.10 Original and approximated step size function 79

4.1 Contour Match Correspondence 83
4.2 Curve Point Match Correspondence 84
4.3 A typical (s, t, u) parameter space for a model 92

5.1 Original shape 112
5.2 Perspective view of the deformed control lattice 113
5.3 Side view of the deformed control lattice 114
5.4 Original shape and control lattice 115
5.5 Side view of deformed shape 116
5.6 Deformed shape 117
5.7 Deformed shape (shaded) 118
5.8 Example using Bernstein-based FFD 119
5.9 Test object 1 121
5.10 Recovered control lattice of test object 1. 122
5.11 The recovered shape of test object 1. 123
5.12 Top view of the recovered shape of test object 1. 123
5.13 Side view of the recovered shape of test object 1. 124
5.14 Shaded recovered shape of test object 1. 124
5.15 Test object 2 125
5.16 Recovered control lattice of test object 2. 125
5.17 The recovered shape of test object 2. 126
5.18 Top view of the recovered shape of test object 2. 126
5.19 Side view of the recovered shape of test object 2. 127
5.20 Shaded recovered shape of test object 2. 127

List of Tables

1.1 Sufficient feature combinations for the solution of cone orientation. (Extracted from [2]) 20
1.2 Sufficient feature combinations for the solution of cone origin. (Extracted from [2]) 20
1.3 Interpretations of FEE contact combinations (Extracted from [2]) 21
1.4 Interpretations of FFE contact combinations (Extracted from [2]) 21

4.1 Indexing for the control vertices 86
4.2 Four cases of relative positions of the control polygon to the two ellipses 104

Acknowledgements

The research and writing of this dissertation required the guidance and assistance of many people whom I would like to thank. First and foremost, I would like to thank my supervisor, Dr. Y. H. Liu, who has given invaluable advice and guidance to me for research. Moreover, I would like to thank my colleagues for their suggestions and comments to my work. They are Mr. C. M. Chung, Mr. N. T. Fong, Mr. C. T. Ng and Mr. H. S. Ng. They have spent their precious research time for discussion with me about my work. Furthermore, I would like to thank all computing staff of the Department of Systems Engineering and Engineering Management and the Department of Mechanical and Automation Engineering of the Chinese University of Hong Kong for their assistance and maintenance of the computing tools and machines. I also thank the staff of the Department of Systems Engineering and Engineering Management for their support. In addition, I would also like to thank my family for their support in these two years.

Abstract

3D model reconstruction of real world objects is a common problem in computer vision. Several techniques have been developed to tackle this problem, like shape from shading, etc. These methods are, in general, passive in nature. Instead of using vision, 3D object information can also be acquired by active touch sensing or tactile sensing. Touch is another sensing modality of human. However, it has been ignored by researchers and psychologists in favour of other sensing modalities, especially vision. In human, touch sensing can obtain not only geometric information of the objects, but also object properties that are hardly acquired by vision, like surface roughness and object weight. Moreover, human can recover shape information reliably and efficiently by touch sensing alone. Therefore, many researchers try to emulate this human information processing ability in robots. In the future, robotic systems will be needed to handle tasks with great precision and accuracy which require touch sensing very much, like grasping objects or tools. In order to achieve these tasks, geometric information of the objects are needed.

In this project, a geometric model is proposed to represent object models. The geometric model is divided in to parts. The first part is the superquadric model which represents a rough shape of the object and the second part is Free Form Deformation which is used to shape fine tuning. Moreover, an active sensing algorithm (or exploratory procedure) is developed to acquire shape data through tactile sensing. The main idea

is to trace contours of the object surface and record 3D coordinates of the surface points along the contours. The separation between contours is determined from the clues obtained from curvature and change in curvature against adjacent contours. In addition, an algorithm is developed to recover 3D shape information of real world objects from the surface points along contours.

Chapter 1

Overview

3D model reconstruction of real world objects is a common problem in computer vision. Several methods have been developed to tackle this problem, like shape from X (where X represents motion, shading, contour, stereo and so on or combinations of the aforementioned methods), range sensing methods using laser or ultrasonic scanning. These approaches are ,in general, passive. On the other hand, 3D information can also be acquired by active touch sensing. Touch is another important sensing modality of human that is not as common as vision and audition in research topics. Touch sensing can obtain not only shape information, but also object properties that are hardly acquired by vision alone, like elasticity, roughness, texture, temperature, weight and material which the objects are made of. Usually, touch sensing and vision sensing cooperate well in acquisition of shape information. For instance, touch sensing can acquire shape information of the object part that is occluded by itself. The shape information of this occluded part cannot be obtained by vision alone. Human can recover shape information reliably and efficiently by touch sensing alone. The active nature of touch sensing allows the subject hand, which is the sensing organ, to move on the target object and explore it so that the occluded portion (in vision which is a passive approach) of the object can be traced out. In the future,

robotic system will be needed to handle tasks which are highly relied on tactile sensing, like grasping, manipulation, inspection, object recognition assembly and even surgery. Some of the complex tasks, that require high precision, may be performed by using tools that are originally designed for human.

1.1 Tactile Sensing in Human and Robot

Human hand acquires two types of information for 3D shape reconstruction. They are the kinethetic information and the cutaneous information. Kinethetic information includes tactual information acquired from skin receptors that respond to mechanically encoded stimuli from the external world when contact is made with skin [3]. When we press an object against the skin, it deforms the skin surface and we experience the sensation of touch, or pressure. The cutaneous information includes joints information, force and torque information from tendons that drive fingers [4][5]. These kinds of information give the position, orientation, and even movement of our limbs. Using both types of information to derive information about objects is called haptic exploration.

1.1.1 Human Hands and Robotic Hands

As the most sensitive part of touch sensing organ (our skin) concentrates on our fingertips, a dextrous multi-fingered robotic hand, with tactile sensor arrays mounted on fingers, is employed in simulation of the process object shape recovery in human. Kinethetic information is obtained from tactile sensor arrays and cutaneous information is acquired from joint position sensors that gives joints angular data and tendon force sensors that measure tension and torque on each tendon which drives fingers on the hand. Some robotic hands are also equipped with force/torque sensors on

the wrist. There are several ready-made multi-fingered robotic hands in the market, like, the Stanford-JPL Dextrous Hand, the Utah/MIT Dextrous Hand[1] [6][7], the Anthrobot-2 and Anthrobot-3[2], Belgrade/USC Hand[3][2], etc.

In the design process of robotic hands, engineers and scientists tried to reproduce the structure, functionality and dexterity of human hand as much as possible. Therefore, in general, robotic hand and human has a lot of similarities in different aspects. Some of them are listed in the followings.

1. Robotic hands and human hands have multiple fingers. Normally, each person has five fingers in one hand. On the other hand, common multi-fingered robotic hand can have 3 to 5 fingers. For instance, the Stanford-JPL Dextrous Hand has 3 fingers, the Utah/MIT Dextrous Hand has 4 fingers while the Anthrobot-3 Hand has 5 fingers.

2. Every movement of human finger is tracked and controlled by our brain. The brain obtains each finger movement information (the position and the movement) from the array of proprioceptive mechanoreceptors in and around a finger joint and in the muscle tendons [3][5]. On the other hand, general dextrous robotic hands have internal position transducers and multi-axes force sensors in each of their finger joints. Tendon driven robotic hands have force sensors on each tendon to measure tension in each tendon. All these sensor information will be transmitted to the central controller of the robotic hand for analysis and control. Both human and robotic hands can obtain the kinethetic and cutaneous information.

3. The sizes of robotic hand and human hand are similar. The research of dextrous robotic hand is currently subject to lively investigation.

[1] It is developed at the Center for Engineering Design at the University of Utah and the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology.

[2] It is developed by Altas Robotics Inc.

[3] It is developed at the University of Belgrade

Researchers tend to imitate the usage and functionality of human hand through robotic hands, like using tools. The implementation of robotic hand will be simplified if its size is similar to human hand.

Although there are similarities between human hand and multi-fingered robotic hands, they also have several differences between them. Some of them are listed in the followings.

1. The sizes of each finger in a normal human hand are not the same, with the middle finger is the longest one while the little finger is the shortest one and the thumb is the strongest one. On the other hand, the sizes of each finger in common robotic hand are the same, except for the thumb.

2. The number of knuckles (i.e. the number of joints) in a finger of a normal human hand is three[4], except that the thumb has only two knuckles. On the other hand, some robotic hands have the same number of knuckles in each finger, even in the thumb.

3. The distribution of fingers in human hand is that all fingers, excluding the thumb, are located at the top level of human palm. The thumb is located at the middle level of the palm so that it can move to a posture opposite to other fingers. The finger arrangements of robotic hand have several categories. Some robotic hands have arrangements similar to that of human hand, like the Belgrade/USC Hand and the Anthrobot-2 Hand. Some other robotic hands have their fingers evenly distributed around a circle.

4. The tactile sensing area of human hand covers the whole surface of the hand. The human mechanoreceptors (tactile sensing element) are distributed in our skin which encloses our hand, from our phalange

[4]including the metacarpophalangeal joint [2].

to our palm. The details is discussed in section 1.1.2. However, the tactile sensing area of robotic hand is usually limited on fingertips.

1.1.2 Mechanoreceptors in skin and Tactile Sensor Arrays

The human tactile sensing organ is skin which covers the whole human body. It is responsive to both mechanical and thermal stimuli with varying spatial acuity depending on the location on the body. The most responsive region of human skin is at the region of fingertips. There are at least four different types of mechanoreceptors innervated with human skin. They are the Pacinian corpuscle, the Meissner corpuscle, the Merkel cellneurite complexes and the Ruffini endings [5][8]. These mechanoreceptors are connected, by nerve fibers, to neuronal pathways which are connected to the somatosensory cortex and to the brain. Their differences diversify in various aspects, like transduction methods, the types of mechanical stimuli that response maximally, dimensions, depth and location within skin layers, etc.

Moreover, the patterns of activity on the afferent fibers from the hand, which are connected to the mechanoreceptors, can be classified into four groups according to their receptive field sizes and the temporal frequency responses of stimuli. The four groups are SAI, SAII, FAI (or RA) and FAII (or PC) [5][9]. SAI and SAII are slowly adapting and response well to static stimuli while FAI and FAII response to vibratory stimuli. SAI and FAI have small receptive field size so that they response to stimuli over an area of 3 to 4 mm while SAII and FAII have large receptive field size so that they response to stimuli over an area of 10 mm or more. Experiments also showed that SAI responses best to compressive stress (curvature) and FAII responses best to vibration [9]. Moreover, FAI and SAII response best to skin stretches and SAII specifically responses well to directional skin stretches [9]. It is commonly accepted that Pacinian corpuscles feed the

FAII afferent fibers and Meissner corpuscles feed FAI fibers. However, it is still an open issue for whether Merkel endings and Ruffini endings feed SA types fibers.

The technology of tactile sensor arrays has been developing over the past 20 years. During this period, many researchers have developed their sensor array prototypes with different working mechanisms and implementations. They usually consist of two-dimensional array of sensing elements. The main idea in the design of this type of sensor is to record the deformation of the sensor surface (which usually is made of elastic material like rubber) and convert the signal to an usable form, usually in the form of electrical signal. Researchers have tried several ways, including,

Piezoresistive Tactile sensor array surface is made of conductive elastomer which exhibits little change in bulk resistance when it is compressed. The change of electrical signal in the conductive elastomer reflects the deformation of it.

Piezoelectric This type of sensor array is similar to the piezoresistive one. The working principle is based on the Piezoelectric effect. Crystals of quartz produces an electrical charge when pressure is applied to the crystal. Material that exhibits this phenomenon is a polymer called polyvinylidene fluoride (PVF2). Thus, piezoelectric tactile sensor arrays usually are made of this polymer.

Capacitive Capacitance can be used to measure separation between two conductive plates (varied by normal force) and overlapping area between two plates (varied by shear force). With these properties, capacitance can be employed to measure both normal and shear forces respectively. However, sophisticated mechanical design is needed to separate the effects of normal force and shear force.

Magnetoresistive Permalloy[5] magnetoresistive sensors can determine the position and orientation of magnetic dipoles buried in a sheet of silicone elastomer. The elastomer will deform when force is applied on it and the deformation (or normal and tangential forces and torques) affects the magnetic dipoles in their height and orientation.

Electrochemical A polyelectrolytic gel is used to transmit deformation information through the streaming potential to electrode. The application of forces gives migration of different types of mobile ions within the gel. The concentrations of these ions reflect the deformation of the gel.

Optical This type of sensor array involves optical fibers. When force is applied to a transparent elastomer layer, the intensity of light reflected back from the far side of the layer is measured by a computer vision system. The compressed elastomer layer shortens the total distance travelled of the light ray and the reflected light intensity is changed. Then, the pattern picked up by the sensor array reflects the deformation of the elastomer layer.

Detailed surveys of different types of tactile sensor arrays can be found in [5] and [8]

The most common implementation of tactile sensor array is to use ultrasonic technique. Figure 1.1 depicts the internal construction of this type of sensor array and its principle of operation. This sensor type usually has a two-dimensional array (typically a 16×16 array) of ultrasonic transmitters and receivers pairs with a rubber pad placed over the sensor array. The ultrasonic transducers measure the thickness of the overlying rubber pad by the Time of Flight (TOF) technique. When objects contact the sensor pad, the rubber is compressed. The amount of compression depends on the

[5]Permalloy is an alloy of 19% iron and 81% nickel which has strong magnetoresistive effect.

force magnitude applied to the object and the stiffness of the rubber pad. Each sensing element transmits an ultrasonic pulse which travels through the rubber pad, is reflected off the top surface of the rubber and returns to the receiver. The time of flight (TOF) t_1 of this travel is recorded for each sensor element. When a contacted object applies a normal force F to the rubber pad, the rubber pad is compressed according to the shape of the object surface patch touching the pad and the TOF is reduced to t_2. The difference of TOF is proportional to the amount of compression of the pad $(d_1 - d_2)$ (as shown in (1.1)), which is also proportional to the applied force F (as shown in (1.2)). Then, the force pattern applied to the object can be revealed by the sensor array data.

$$d_1 - d_2 = \frac{1}{2}c(t_1 - t_2) \tag{1.1}$$

$$F = k(d_1 - d_2) = \frac{1}{2}kc(t_1 - t_2) \tag{1.2}$$

where c is the speed of ultrasound in rubber pad and k is the rubber stiffness.

There are several differences between human and robot tactile sensing,

1. There are four types of mechanoreceptors in human skin for tactile sensing. They have different responses to different types of mechanical stimuli. On the other hand, there is usually only one type of sensing elements in a tactile sensor array. Each sensing element has similar response to same mechanical stimuli.

2. Practically, the sensing area of tactile sensing system of human is the whole area of skin covering human body. This area is very large. Different regions of the human skin response differently to similar stimuli and the sensitivities also vary. In robots, the sensing area is confined on a small area of the surface of the sensor array.

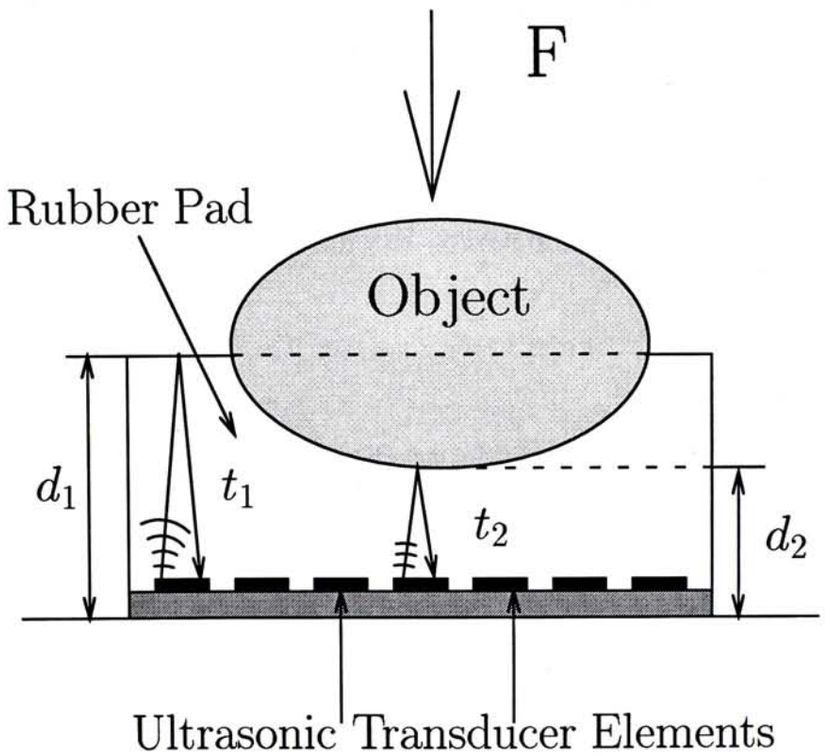


Figure 1.1: Ultrasonic tactile sensor array construction and operation mechanism

3. The sensing process may be forced to stop by involuntary action stimulated by strong responses from other types of receptors, like temperature receptors and pain receptors. When, for example, someone tries to explore haptically a very hot or an electrically charged object, the exploration may be stopped by the involuntary withdrawal of his hand due to the heat and the electrical shock of the object. The involuntary withdrawal is an instinct action to prevent our body from any damage. On the other hand, robot tactile sensing system usually is not equipped with other types of sensor to monitor the external environment. The control unit of tactile sensing system is usually not intelligible to determine when to stop exploration before the whole system is damaged with limited source of input information.

1.2 Motivation

Human is good at identifying objects by haptic exploration. In object recognition, shape information of the object should be first perceived. Some psychologists had conducted experiments on this issue. Experiments proved that human haptic system is extremely effective in object recognition. In one of these studies, which was conducted by Klatzky, Lederman and Metzger [10], subjects were given 100 common manipulable objects (like cups, dishes, etc.) for identification. They found that the rate of successful identification was 96%, and if near misses were accepted, the rate was 99%. Moreover, the modal response latency time was 1 to 2 seconds. It can be concluded that human haptic system can give accurate and fast responses in object recognition. This dissertation tries to develop a robotic system to imitate the shape recovery process of human haptic system in object recognition.

Haptic exploration becomes more and more important in future robotic applications. These applications usually require fine manipulation

of objects and even tools designed for human. In order to achieve these tasks, tactile sensing is an important technique to be managed. Sometimes, object attributes that are difficult to be extracted by vision techniques can be acquired easily by tactile sensing. These object attributes includes, shape information in self-occluded regions, weight, elasticity, etc. One of the branches in recent robotics research is telerobotics. Telerobotics relies on tactile sensing technology heavily, especially in one of its application, telepresence surgery. This application is to perform surgeries by robots with remote supervision of experienced doctors [11].

1.3 Objectives

The objective of this project is to rebuild the 3D shape of an object by active haptic exploration via a multi-fingered, dextrous robotic hand which is equipped with tactile sensors arrays [12][13][14][15]. This project can basically be divided into two parts. The first part is to construct a geometric model to represent real world objects and the second part is to develop exploration strategies for acquisition of tactual information and 3D objects models reconstruction.

The geometric model that I intend to use is the superquadric model with free-form deformations. Superquadrics [16][17] can represent both the curved objects and objects with sharp edges (like polyhedra) successfully. With the (global or local) refinements of the global superquadric model through the technique of free-form deformation [18][19][20][21][22][23][24], a large variety of real world objects (especially those with fine curvature changes on surfaces) can be figured out. However, there is a compromise among the flexibility of this combined geometric model, the computational efficiency and the complexity of the model.

The main idea of the 3D models reconstruction algorithm presented in this dissertation is to trace contours on the object surface. After sufficient

number of contours are traced on the object surface, the 3D object model is reconstructed with the aid of the superquadric model with free form deformation. The process is similar to finding cross section of the object at different slicing altitudes. Each contour is actually a closed plane curve and lies on a slicing plane of the object. The algorithm to determine the altitudes for all contour tracings is also developed. The goal of the algorithm is to find the best fit reconstructed model of the object that minimize the number of contour tracing needed.

1.4 Related Work

Shape reconstruction is a challenging problem in research. In order to reconstruct a best fit model from sensing data, different shape representation and geometric modelling techniques have been developed. Using a good shape representation for best fit model is not enough for the construction of a best fit object model. The sensing technique is also very important. Researchers focus their attention on shape reconstruction from images of object acquired by computer vision system. As there are difficulties in shape reconstruction from vision alone, some researchers began to integrate vision system with other sensing method, like tactile sensing. They found that vision and touch sensing can cooperate well, as in human. Later, some researchers began to investigate the possibility of shape reconstruction by tactile sensing alone. They were motivated by the experimental results, which stated that human is good at identifying objects and acquiring shape information by touch sensing alone. In this chapter, studies of the pioneers in these fields are discussed.

1.4.1 Using Vision Alone

3D shape reconstruction of real world objects is a common problem in computer vision. Different techniques have been developed for this purpose. Given a set of images of an object with different views, its 3D model can be reconstructed. The images set provides a lot of clues about the shape of the object. Reconstruction techniques are characterized by the employment of different clues in the algorithms. These clues include shading of the object in images, motion information of the object extracted from an image sequence, depth information extracted from stereo images pair, contours of the object in images and so on. Some researchers try to combine these clues together to reconstruct a better model. This technique usually called Shape from X.

Although the aforementioned techniques can give rather good model reconstruction, there are some difficulties in these methods. First of all, a large amount of data is needed (like several different views of the same scene for self-occlusion avoidance or a dense image sequence is needed for motion extraction). If only a few views or sparse data are obtained, the self-occluded region of the object cannot be reconstructed well. Secondly, these methods are usually computationally expensive. Thirdly, some object properties are hardly obtained by vision alone, like elasticity, temperature, weight and so on of the object. Therefore, other sources of sensing data are explored with integration of vision. Tactile sensing is one of the alternatives. Multisensor data fusion techniques [25] are employed to integrate data from different sensing systems.

1.4.2 Integration of Vision and Touch

In mid 80's, Peter K. Allen had tried to integrate vision and touch sensing for 3D object structure understanding in object recognition [26]. He used a coffee mug and a pitcher as example objects. The experimental setup

consisted of a stereo pair of CCD cameras and a tactile sensor probe mounted on a PUMA manipulator which has six degree of freedom. The object under investigation was placed on a worktable. The general recognition procedures are simple. Firstly, the vision system takes images of the scene and analyzes all identifiable regions of interest. Then, the tactile system explores each region of interest labelled by the vision system. Surface and feature descriptions can be generated from vision and tactile data. These descriptions are crucial components for matching against the model database. Finally, the hypothesized model is verified by further exploratory sensing.

The construction of model database is a crucial issue in object recognition process. Allen used a hierarchical model database structure. Objects are modelled as collections of surface patches, features and relations. The relational information is especially useful in object recognition. The hierarchical model database is organized into four levels, namely, the object level, the component/feature level, the surface level and the patch level. The relational information among components and features of an object is stored at the object level. The other three lower levels store physical and geometric information of the object components.

The vision system recovers silhouette of the target object from images taken by the CCD cameras through processes of image segmentation, edge detection and edge linking. Moreover, depth information of object surface can derived from stereo image pairs. Detail description of object surface patch can then be determined. On the other hand, the tactile sensing system can be divided into two modules. They are the surface exploration module and the hole/cavity exploration module. Allen had developed different algorithms for surface patch tracing and hole/cavity tracing.

With vision and tactile data in hand, they can be integrated

together to derive 3D descriptions of surface patches and features (like holes and cavities) of the target object for feature matching against the model database. The surface features for matching is modelled by a G^2 continuous Coon's patch [27][28][29] from both the vision and tactile data. A Coon's patch $\vec{C}(u,v)$, which is parametrized by $[u,v] \in [0,1] \times [0,1]$, is defined as

$$\begin{aligned}\vec{C}(u,v) &= [(1-u)u]\begin{bmatrix}\vec{C}(0,0) & \vec{C}(0,1)\\ \vec{C}(1,0) & \vec{C}(1,1)\end{bmatrix}\begin{bmatrix}(1-v)\\ v\end{bmatrix}\\ &= \vec{C}(0,0)(1-u)(1-v)+\vec{C}(0,1)(1-u)v\\ &\quad +\vec{C}(0,1)u(1-v)+\vec{C}(1,1)uv \qquad (1.3)\end{aligned}$$

where $\vec{C}(0,0), \vec{C}(0,1), \vec{C}(1,0)$ and $\vec{C}(1,1)$ are four corner points of the Coon's patch. The other feature is a smoothed boundaries of holes and/or cavities on the object. Using these features, the recognition system tries to match the target object with the models stored in the model database to find the most resemble one. The hypothesized model is then verified by further exploration sensing.

1.4.3 Using Touch Sensing Alone

Few researchers or research groups have worked on touch sensing. Most of the work was concentrated on object recognition with the aid of a object shape database using tactile sensors. Peter K. Allen and his colleagues in the Department of Computer Science in Columbia University were pioneers in this field. They have built a system to implement active touch sensing by multi-fingered dextrous robotic hand. They also have developed three exploratory procedures to recover shape information of unknown objects using this system. In addition to shape information of the objects, we sometimes need to find out the orientation of the object. Ronald S. Fearing has proposed an algorithm to recover global object properties,

such as size, location and orientation of the object with a few from only sparse local geometric information at three fingers. We will briefly describe their work in the following.

1.4.3.1 Ronald S. Fearing's Work

Ronald S. Fearing has investigated the problem of determining global object properties, like size, location and orientation from sparse local geometric information at three finger contacts. The goal of object orientation and origin determination is to reorient this object in hand from an acquisition grasp to task performing grasps like insertion or stable transport grasp when the hand moves through space. The tactile sensors, which are equipped on each finger, provide local shape information, including surface normals, contact location on the finger and principal curvatures and principal directions. Moreover, shape information is also derived from these input data.

Fearing proposed to model objects by Linear Straight Homogeneous Generalized Cylinder (LSHGC) which is a specific class of generalized cylinders (GC). Details about generalized cylinders can be referred to section (2.5.1). These objects are generated by translating a convex cross-section along an orthogonal axis and scaled by a linear sweeping rule. The objects generated actually are cones. The mathematical definition of LSHGC is

$$\vec{p}(z,\theta) = (Az + B)\lambda(\theta)(\cos\theta\hat{x} + \sin\theta\hat{y}) + z\hat{z} \tag{1.4}$$

where A is a scale factor, $\lambda(\theta)$ is the cross-section of the object in polar coordinates. When $A = 0$, the LSHGC is actually a regular cylinder whereas it is a cone with its apex at the origin when $B = 0$. Fearing concentrated on cone in his work.

Moreover, Fearing pointed out that an LSHGC has a limited set of

contact features which are categorized into 4 groups. They are point/vertex group (**V**), plane/face group (**F**), edge (**E**) and parabolic points (**P**)[6]. These four groups of contact features can be furthered classified into four types according to their locations, namely, Top (**T**), Bottom (**B**), Side (**S**) and Meridian (**m**). Fearing also pointed out that a parabolic point can give equivalent information to a meridian. For clear description, a set of notation representing the contact features is employed with the location of the feature specified in the subscript of the abbreviation. For instance, $\mathbf{F_T}$ represents the top face of a cone and $\mathbf{E_m}$ represents a meridian edge.

The orientation and origin of the object can be described by traditional technique, ie. a rotation matrix R and a translation vector $\vec{v} = [v_x, v_y, v_z]^T$.

$$\vec{x}_S = R\vec{x} + \vec{v} \tag{1.5}$$

where $\vec{x}$ is a point in object coordinates and $\vec{x}_S$ is the sensed location in world coordinates. The rotation can be considered as two rotations. The first one is to rotate angle ϕ about the $\hat{z}$ axis and the second one is to rotate angle ψ about the $\hat{y}$ axis. Then, the matrix R can be written as

$$R = \begin{pmatrix} \cos\phi\cos\psi & -\sin\phi & \cos\phi\sin\psi \\ \sin\phi\cos\psi & \cos\phi & \sin\phi\sin\psi \\ -\sin\phi & 0 & \cos\psi \end{pmatrix} \tag{1.6}$$

There are totally six global unknowns for a cone, namely, A, v_x, v_y, v_z, ϕ and ψ. There parameters affect each local measurement. Although every contact feature measurement can introduce equations for solving these unknowns (Details can be referred to [2]), additional local unknown variables will also be introduced to the problem. These local unknowns usually carry shape information. For instance, a parabolic point measurement bring us

[6] A surface point is said to be parabolic if it has zero Gaussian curvature, or one of its principal curvature is zero [30][31][29]. (See section 3.5)

Solve for cone orientation parameters ϕ and ψ	Constraints
$\mathbf{F_T}$ or $\mathbf{F_B}$	
$\mathbf{E_{T1}E_{T2}}$ or $\mathbf{E_{B1}E_{B2}}$	
$\mathbf{E_TE_B}$	Not Parallel
$\mathbf{V_{T1}V_{T2}V_{T3}}$ or $\mathbf{V_{B1}V_{B2}V_{B3}}$	
$\mathbf{E_TV_T}$ or $\mathbf{E_BV_B}$	Not Coincident

Table 1.1: Sufficient feature combinations for the solution of cone orientation. (Extracted from [2])

Solve for cone origin $\vec{v} = [v_x, v_y, v_z]^T$	Constraints
$\mathbf{P_1P_2}$ or $\mathbf{E_{m1}E_{m2}}$ or $\mathbf{PE_m}$	Not Collinear
$\mathbf{E_mF_S}$ or $\mathbf{PF_S}$	Not Coplanar
$\mathbf{F_{S1}F_{S2}F_{S3}}$	None of them is coplanar

Table 1.2: Sufficient feature combinations for the solution of cone origin. (Extracted from [2])

seven independent equations, but it also introduce extra five unknowns to the problem, together with the six global unknowns. The additional unknowns are λ, λ', λ'', θ and z.

In order to determine the orientation and origin of the cone, combinations of contact feature measurement is employed. Fearing had shown that a combination of 2 meridian edges $\mathbf{E_m}$ and a surface normal on an end face $\mathbf{F_T}$ or $\mathbf{F_B}$ will give solution of the cone orientation and origin. Moreover, other combinations are also sufficient to give that solution. Fearing had proposed some of them and they are listed in Table 1.1 and Table 1.2 Fearing also had shown that end face is a necessary contact feature measurement for solution of cone orientation and origin.

In addition, Fearing had examined the interpretations of combinations of one face and two edges contacts (**FEE**), and combinations of two faces and one edge contacts (**FFE**). For **FEE** contact combinations, he found that there are twelve combinations valid for examination. They are listed in Table 1.3 For **FFE** contact combinations, he found that there are eight combinations valid for examination. They are listed in Table 1.4

Type	Combination	Interpretation
(i)	$\mathbf{F_T E_B E_B}$ $\mathbf{F_T E_B E_m}$ $\mathbf{F_B E_T E_T}$ $\mathbf{F_B E_T E_m}$	Meridian edge cannot be be parallel to end face
(ii)	$\mathbf{F_S E_{m1} E_{m2}}$	Apex cannot be on plane $\mathbf{F_T}$
(iii)	$\mathbf{F_S E_m E_T}$	Contact on $\mathbf{E_m}$ must be below the apex
(iv)	$\mathbf{F_S E_{T1} E_{T2}}$ $\mathbf{F_T E_{m1} E_{m2}}$	Sensed face could be side or base (or bottom face if cross section contains origin)
(v)	$\mathbf{F_B E_{m1} E_{m2}}$ $\mathbf{F_S E_B E_m}$ $\mathbf{F_S E_B E_B}$ $\mathbf{F_S E_B P}$	Face could be side or base

Table 1.3: Interpretations of FEE contact combinations (Extracted from [2])

Type	Combination	Interpretation
(i)	$\mathbf{F_T F_S E_B}$ $\mathbf{F_B F_S E_T}$ $\mathbf{F_B E_T E_T}$	$E \cdot \hat{n} \neq 0$ Meridian edge cannot be be parallel to end face
(ii)	$\mathbf{F_S F_S E_m}$	Apex cannot be on plane $\mathbf{F_T}$
(iii)	$\mathbf{F_T F_B E_m}$	End and side planes are not parallel
(iv)	$\mathbf{F_T F_T E_m}$	
(v)	$\mathbf{F_B F_S E_m}$ $\mathbf{F_S F_S E_T}$ $\mathbf{F_S F_S E_B}$	Face could be side or base

Table 1.4: Interpretations of FFE contact combinations (Extracted from [2])

1.4.3.2 Peter K. Allen's Work

Peter K. Allen and his colleagues have proposed a solid background theory on shape information recovery from tactile sensing. They also built an hardware setup for experiments about tactile sensing. This experimental setup demonstrated to other researchers what and how a complete experimental setup for tactile sensing is composed of.

The hardware testbed system they have built composed of a PUMA 560 manipulator with a Utah/MIT dextrous robotic hand mounted on it. The size and shape of the Utah/MIT hand are similar to those of human, but the Utah/MIT hand only has four fingers (including a thumb). Each finger has four degrees of freedom. Each finger joint is controlled by two tendons, namely extensor and flexor. Moreover, the hand has joint position sensors to measure joint angle and tendon force sensors to measure tensions on each of the two tendons at each joint. The position and orientation of each finger tip and contact location can be determined from these information. Each finger is equipped with a tactile sensor array with the dimension of 16×16 at the finger tip. On the other hand, the PUMA 560 manipulator is one of the most commonly used robot arm in the world. It has totally six degrees of freedom, which three dof's[7] govern translational motion and the other three dof's govern rotational motion. Therefore, the whole system has 22 degrees of freedom.

They have implemented three exploratory procedures (EP's) for acquisition of 3D shape information through active haptic sensing. Researchers in human haptic system noticed that human can recognize properties of 3D objects accurately and quickly. These properties include temperature, roughness, weight, size, texture, functions and the most important attribute, the shape information. Human determines these attributes by some special *exploration procedures*. This ability is very

[7] dof is the abbreviation of degree of freedom.

important to future intelligent robotic systems for object manipulation. Peter K. Allen and his colleagues have derived three exploratory procedures based on the EP's in human haptic sensing paradigm. Moreover, the three EP's, which they implemented, can be mapped to three special types of shape representation schemes that have been widely adopted.

The first EP is *Grasping by Containment* [12][13][14][15]. The shape representation model that matches this EP is superquadrics with global deformations (linear tapering and bending). The goal of this EP is to "understand the gross contour and volume by effectively molding the hand to the object"([13], p.399). The Grasping by Containment EP is as follows. The object under exploration was fixed on a test platform to prevent displacement of the objects during active exploration. The intention was to gather contact data points from the tactile sensor arrays mounted on fingers[8]. Firstly, the PUMA 560 arm moves the Utah/MIT hand to a position that it can enclose around the object under exploration, with widely stretched fingers. Then, the fingers attempt to close towards the object until a certain tension threshold in the tendons has been reached for each finger. The threshold represents the reference of contact between the object and the finger. Afterwards, the coordinates of contact points can be obtained by converting the joint positions data from the hand and the PUMA arm through the forward kinematic model of the system to the world coordinates system. Next set of contact points will be found by similar procedures when the hand is moved to different positions and orientations. Usually 30-100 contact points are enough to recover the shape of the objects. Using the recovery algorithm developed by Solina [32], the shape of the object can be obtained. Details of Solina's algorithm will be discussed in section 2.3.

The second EP is *Planar Surface Explorer* [12][13][15]. The shape

[8]The Cartesian coordinates of contact points were actually generated from the joint sensors readings and tendons forces data from the hand's sensing systems because the tactile sensors had not been mounted on the finger's links during their initial trials [12][13][14]

representation that matches this EP is the winged-edge type of Face-Edge-Vertex model suggested by Baumgar [33][34]. This model has been widely used in the Boundary Representation (B-rep) for solid modelling. The goal of this EP is "to explore a continuous, homogeneous surface such as a planar face, and to determine its extents." ([15], p.1679). Application of this EP on different planes, details of the planes can be obtained. The edges and vertices of these planes can be easily computed from their intersections. The index finger of hand plays an important role in this EP, which is described as follows,

1. Move the index finger toward the object planar surface until the tactile sensors on it detects a contact with the object. Record the initial contact point coordinates.

2. Lift up the index finger until tactile contact is lost.

3. Move the arm in parallel to the surface.

4. Lower the index finger until tactile contact is detected. Record the contact point coordinates.

5. Repeat steps 2 to 4 until the index finger fails to detect contact with the object in step 4. This is because the finger has moved beyond the surface or the surface is too far away to be detected.

6. Move the index finger back to the initial contact point. Repeat steps 1 to 5, except the searching direction is opposite to that of previous search, until a second edge is detected.

7. Repeat the search (steps 1 to 6) except that the searching direction is perpendicular to that of the previous search.

After the above steps, the edges and extent of the planar surface can be derived from the extreme points at the edges. The plane equation can

be computed by least-square fitting the set of contact points obtained from this EP.

The third EP is *Surface Contour Follower* [12][13][15]. The shape representation that matches this EP is Generalized Cylinders/Cones. There are many different types of generalized cylinders/cones models. Peter K. Allen and his colleagues especially selected RLSHGC (Right Linear Straight Homogeneous Generalized Cones) to model objects. This is equivalent to a surface of revolution and can be completely described by a rotation of a plane curve about an arbitrary axis. The goal of this EP is to trace a contour curve on the object. If two contour curves that are located on either side of the object, the axis of symmetry can be estimated and hence, recover the object shape. This EP uses the thumb and the index finger as primary sensing devices.

1. Move the hand so that it is near one end of the object under exploration. Stretch the thumb and the index finger so that the object can be encompassed by them without any contact.

2. Move the thumb slowly until contact with the object is detected. Then, move the index finger with the same manner. Record the two contact points coordinates. The control system should also check whether the finger has moved the commanded entire distance with no tactile contact detected. If this case happens, this data set will be discarded because there may be something that prevents the finger from moving the commanded distance.

3. Lift off the fingers from the object and move the hand with a small amount along the axis of the object[9].

4. Repeat steps 2 and 3 until one of the fingers fails to make contact with the object when it moves towards the object or the hand has not

[9]The axis of the object is not found by the system, but with human aid

moved the commanded distance along the axis of object.

Each EP gives different shape information of the target object. However, they only give limited knowledge to the robot. They are only suitable for simple object shapes[10] and/or polyhedra[11]. If the object shape is very complex and with rapid changes, like object surface with frequent curvature changes, they can only perceive limited local shape information to the robot. This dissertation will extend Dr. Allen's work to recover shape information of complex objects.

1.5 Outline

This dissertation is organized as follows. Chapter 1, this chapter, gives an introduction of tactile sensing in robotics and the objective of the project. It also introduces some related works of shape reconstruction by vision, tactile sensing and both of them. In chapter 2, geometric models are the main content and some comparisons among different types of geometric models. Chapter 3 and 4 concern about the sensing strategies and shape reconstruction algorithms proposed in this dissertation respectively. Chapter 5 covers issues on implementation. In addition, chapter 6 is the conclusion of the dissertation with future works proposed.

[10] cf The first EP *Grasp by Containment* and the third EP *Surface Contour Follower*.
[11] cf The second EP *Planar surface Explorer*.

Chapter 2

Geometric Models

2.1 Introduction

There are a lot of geometric models proposed by many researchers. Some are simple and some are very sophisticated. Some geometric models are very specific because they have their own properties so that they are suitable to represent object shapes of particular categories while some models can represent a large variety of shapes. In this chapter, different geometric models are discussed and compared. Description of the proposed geometric model is also explained. The proposed geometric model is basically divided into two major parts, one is the global superquadric model and the other is the global or local refinements on the coarse superquadric model by free-form deformations.

2.2 Superquadrics

2.2.1 2D Superquadrics

Superquadric curves [17] can be considered as a superset of traditional quadric curves. The order of the definition of the curve need not be integers. The exponents of each term in the polynomial expression

may be rational numbers. For instance, a superellipse is defined as,

$$\left(\frac{x}{a}\right)^{\frac{2}{\epsilon}} + \left(\frac{y}{b}\right)^{\frac{2}{\epsilon}} = 1$$

,with a and b are the two orthogonal radius in the superellipse. The shape of the curve can be controlled by varying ϵ. When ϵ lies between 0 and 1, the curve is squarish in shape. When ϵ is 1, the curve is actually an ellipse or circle (when $a = b$). When ϵ lies between 1 and 2, the curve will become bevelled diamond in shape. When ϵ is larger than 2, the curve will become pinched. These are illustrated in Figure-2.1.

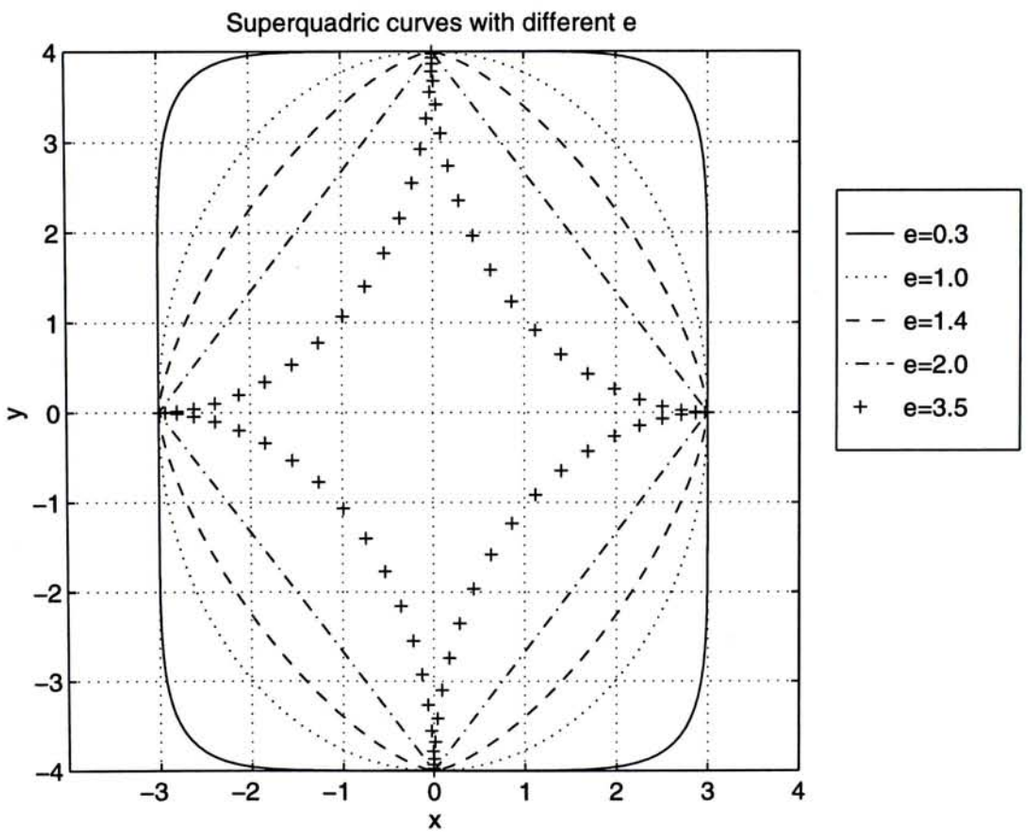


Figure 2.1: Superquadric curves with different ϵ

2.2.2 3D Superquadrics

As the objective of this project is to build a 3D model of an object, superquadric surfaces are investigated. Superquadric solids, in generally, are spherical product of two superquadric curves. They can be represented as 3D vectors in a parametric form, as follows,

$$\vec{x}(\eta,\omega) = \begin{pmatrix} a_1 cos^{\epsilon_1}\eta \, cos^{\epsilon_2}\omega \\ a_2 cos^{\epsilon_1}\eta \, sin^{\epsilon_2}\omega \\ a_3 sin^{\epsilon_1}\eta \end{pmatrix} \quad \text{where} \quad \begin{matrix} -\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2} \\ -\pi \leq \omega \leq \pi \end{matrix} \tag{2.1}$$

where a_1, a_2 and a_3 define the size of the model in x-, y- and z-directions respectively and ϵ_1 and ϵ_2 define the "squareness" of the model in the latitude and longitude planes respectively. The parameters, η and ω, define the spherical coordinates, in which η is the angle between and its projection on the x-y plane and ω is the angle between x-axis and its projection in the x-y plane. ϵ_1 and ϵ_2 have similar behaviour with their superquadric curve counterpart. Figure-2.2 depicts superquadric shapes with different ϵ_1 and ϵ_2 with same superquadric sizes a_1, a_2 and a_3.

In Figure-2.2, we can see that a large variety set of objects can be represented by this model, like, spheres, cylinders, parallelepipeds, diamond bevelled shapes, pinched shapes and so on. As object with pinched shape is rare in real world, the ranges of the two "squareness" parameters ϵ_1 and ϵ_2 are confined the range $0 < \epsilon_1, \epsilon_2 \leq 2$.

In addition, the superquadric model can also be expressed implicitly by eliminating the spherical coordinates parameters, η and ω. The expression is

$$\left(\left(\frac{x}{a_1} \right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2} \right)^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3} \right)^{\frac{2}{\epsilon_1}} = 1 \tag{2.2}$$

With (2.2), an inside-outside function can be easily derived. The function

is defined as follows,

$$IO(x,y,z) = \left(\left(\frac{|x|}{a_1} \right)^{\frac{2}{\epsilon_2}} + \left(\frac{|y|}{a_2} \right)^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{|z|}{a_3} \right)^{\frac{2}{\epsilon_1}} - 1 \qquad (2.3)$$

The absolute values of x, y and z are needed so that 3D points in all the eight quadrants in the Cartesian coordinates system can be considered. If, for any point (x,y,z),

$$IO(x,y,z) \begin{cases} = 0 & \\ > 0 & \text{, then the point is} \\ < 0 & \end{cases} \begin{array}{l} \text{on the surface.} \\ \text{inside the solid.} \\ \text{outside the solid.} \end{array}$$

Superquadric model forms a closed surface and is volumetric in nature. This fits the basic geometric model requirement of this project as real world objects are usually volumetric. One problem of the superquadric model is its ambiguity of shape representation. Same shape can be described by more than 2 sets of shape parameters. Moreover, in Figure-2.2, we can see that the parameterization of superquadric model is non-uniform. The larger the curvature at the portion of the model, the denser the isoparametric curves located on that portion. This can be overcome by reparameterizing the parameter space in a regular grid. By uniformly sampling the spherical coordinates domain (η,ω), $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$, we can have the following [19],

$$\begin{aligned} x &= a_1 \lambda \cos\eta \cos\omega \\ y &= a_2 \lambda \cos\eta \sin\omega \\ z &= a_3 \lambda \sin\eta \end{aligned} \qquad (2.4)$$

where

$$\lambda = \left[\left(|\cos\eta \cdot \cos\omega|^{\frac{2}{\epsilon_2}} + |\cos\eta \cdot \sin\omega|^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + |\sin\eta|^{\frac{2}{\epsilon_1}} \right]^{-\frac{\epsilon_1}{2}} \qquad (2.5)$$

Having this regular parameterization, isoparametric curves will be evenly spaced on object surface and the effect of free-form deformation will be more accurate. Figure-2.3 shows the uniform and non-uniform parameterizations on the same cube.

2.3 Model Recovery of Superquadric Models

2.3.1 Problem Formulation

Solina and Bajcsy developed a method for recovery of superquadric models with global deformations from 3D data points [32]. They defined superquadric inside-outside function as,

$$F(x,y,z) = \left(\left(\left(\frac{x}{a_1} \right)^{\frac{2}{\epsilon_2}} + \left(\frac{y}{a_2} \right)^{\frac{2}{\epsilon_2}} \right)^{\frac{\epsilon_2}{\epsilon_1}} + \left(\frac{z}{a_3} \right)^{\frac{2}{\epsilon_1}} \right)^{\epsilon_1} \tag{2.6}$$

Representing the orientation of the object by Euler angles (ϕ, θ, ψ) and the position vector of the object as $[p_x\, p_y\, p_z]^T$, the inside-outside superquadric function will depend on 11 parameters.

$$F(x,y,z;\Lambda) = F(x,y,z;\, a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z)$$

The model can be further extended to include global deformations so that more complex objects can be represented. The global deformations include linear tapering and bending. Consider linear tapering on a superquadric model along the z-axis (with tapering coefficients K_x and K_y at the x- and y-axes), we have the deformed object points $(x_d, y_d, z_d)^T$,

$$\begin{aligned} x_d &= \left(\frac{K_x}{a_3} z + 1 \right) x \\ y_d &= \left(\frac{K_y}{a_3} z + 1 \right) y \\ z_d &= z \end{aligned} \tag{2.7}$$

For bending, the bending plane is defined by the z-axis and the vector $\vec{r}$ in the x-y plane whose direction is denoted by α. The length of projection of a point on the bending plane is r and the curvature parameter is denoted by k,

$$\begin{aligned} x_d &= x + cos\,\alpha\,(R - r) \\ y_d &= y + sin\,\alpha\,(R - r) \\ z_d &= sin\,\gamma\,(k^{-1} - r) \end{aligned} \tag{2.8}$$

where

$$\begin{aligned} r &= cos\left(\alpha - tan^{-1}\left(\frac{y}{x}\right)\right)\sqrt{x^2 + y^2} \\ \gamma &= zk^{-1} \\ R &= k^{-1} - cos\,\gamma\left(k^{-1} - r\right) \end{aligned} \tag{2.9}$$

Incorporating the two global deformations into the superquadric model (with predefined deformation order), the inside-outside function becomes depending on 15 parameters,

$$F(x, y, z; \Lambda) = F(x, y, z; a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z, K_x, K_y, k, \alpha).$$

Suppose there are N 3D surface points (x_i, y_i, z_i), $\forall i = 1, \ldots, N$, we can find the superquadric model that fits the data points the best by solving the following least-squares minimization problem,

$$\textbf{Min} \sum_{i=1}^{N} [R(x_i, y_i, z_i; \Lambda)]^2 \tag{2.10}$$

where

$$R(x, y, z; \Lambda) = \sqrt{a_1\,a_2\,a_3}\,[F(x, y, z; \Lambda) - 1].$$

The reason of using R as the minimization function is that the optimal

model parameters set with minimal R will give the superquadric model with the smallest volume that the given data points set in the least square sense. The minimization problem (2.10) can be solved by the Levenberg-Marquardt Method. The Levenberg-Marquardt Method can give more robust solution for least square minimization problem than other traditional methods, like the steepest descent method and the Gauss-Newton method.

2.3.2 Least Squares Optimization

In general, a least squares optimization problem [35] is formulated as follows,

$$\mathbf{Min}\,\mathbf{F}(\vec{x}) = \sum_{i=1}^{m} f_i(\vec{x})^2 = \mathbf{f}(\vec{x})^T\mathbf{f}(\vec{x})$$

where $\vec{x}$ is a n-dimensional vector. Define the Jacobian matrix $\mathbf{J}(\vec{x}^k)$ of $\mathbf{f}(\vec{x}^k)$ at the k-th iteration of solution finding,

$$\mathbf{J}(\vec{x}^k) = \begin{bmatrix} \frac{\partial f_1^k}{\partial x_1^k} & \cdots & \frac{\partial f_1^k}{\partial x^k} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m^k}{\partial x_1^k} & \cdots & \frac{\partial f_m^k}{\partial x_n^k} \end{bmatrix} \tag{2.11}$$

Then, the gradient vector can be written as

$$\mathbf{g}(\vec{x}^k) = 2\mathbf{J}^T(\vec{x}^k)\mathbf{f}(\vec{x}^k) \tag{2.12}$$

Differentiating(2.12) with respect to x_p^k will give,

$$G_{pq}^k = 2\sum_{i=1}^{m}\left(\frac{\partial f_i^k}{\partial x_p^k}\frac{\partial f_i^k}{\partial x_q^k} + f_i^k\frac{\partial^2 f_i^k}{\partial x_p^k \partial x_q^k}\right) \tag{2.13}$$

Introducing $\mathbf{H}_i^k = \mathbf{H}_i(\vec{x}^k) = \nabla^2 f_i(\vec{x}^k)$, we have the complete Hessian matrix of $\mathbf{F}(\vec{x}^k)$,

$$\mathbf{G}(\vec{x}^k) = 2\mathbf{J}^T(\vec{x}^k)\mathbf{J}(\vec{x}^k) + 2\sum_{i=1}^{m} f_i(\vec{x}^k)\mathbf{H}_i(\vec{x}^k) \tag{2.14}$$

Equation (2.14) can be simplified by introduction of $\mathbf{Q}(\vec{x}^k) = \sum_{i=1}^{m} f_i(\vec{x}^k)\mathbf{H}_i(\vec{x}^k)$, we have

$$\mathbf{G}(\vec{x}^k) = 2\mathbf{J}^T(\vec{x}^k)\mathbf{J}(\vec{x}^k) + 2\mathbf{Q}(\vec{x}^k) \tag{2.15}$$

By Newton's method [35], we can find the search direction $\vec{d}^k$ and the new estimate of solution at the k-th iteration by solving

$$\begin{aligned} \left(\mathbf{J}^T(\vec{x}^k)\mathbf{J}(\vec{x}^k) + \mathbf{Q}(\vec{x}^k)\right)\vec{d}^k &= -\mathbf{J}^T(\vec{x}^k)f_i(\vec{x}^k) \\ \vec{x}^{k+1} &= \vec{x}^k + \vec{d}^k \end{aligned} \tag{2.16}$$

Equation (2.16) can be simplified and solved by two approaches. The first approach is the Gauss-Newton method [35]. This method sets $\mathbf{Q}^k$ to zero. The second approach is the Levenberg-Marquardt (LM) method [35]. This method sets $\mathbf{Q}^k = \omega^k I$, where I is a identity matrix and ω^k is a scalar. When $\omega^k \to 0$, the LM method reduces to the Gauss-Newton's method. When $\omega^k \to \infty$, the LM method tends to the Steepest Descent method [35]. Actually, the Levenberg-Marquardt method is the most robust algorithms among the three alternatives. During each iteration, ω^k will be updated.

2.4 Free-Form Deformations

Deformation is a powerful tool in geometric modeling and computer animation. In 1984, A. H. Barr [36] presented a series of deformation transformations, like bending, tapering, twisting and composite of them. These techniques are rather global transformation and they do not let the users to deform the object concerned with their idea. The flexibility of these techniques are also limited.

In 1986, Sederberg and Parry [23] proposed method of free-form deformation (FFD). FFD can provide great flexibility in deforming objects

to the users. The concept of FFD is that the object concerned is conceptually made of a lump of clay. Users act as sculptors to build (or deform) the "lump" in a free-form manner.

In general, there are four steps involved in free-form deformation [20].

1. *Construction of parametric solid* — A parametric solid is first constructed by a set of 3D control lattice points and a corresponding set of parametric basis functions. Therefore, for every point in the solid, we can find a parametric coordinates triple for that point. The control lattice is usually parallelepiped in shape, but S. Coquillart [22] proposed that the control lattice can be any arbitrary shape by combining several tricubic Bézier volumes, each represents an individual FFD.

2. *Embedding the object into the parametric solid* — The parameter coordinates set for each point on the object concerned are solved with respect to the constructed solid in step 1.

3. *Deforming the parametric solid* — The process is to change the shape of the parametric solid according to the users' mind by displacing the control vertices of the solid.

4. *Generation of deformed solid* — By using the parametric coordinates triple generated from step 2 and the deformed lattice vertices in step 3, the new shape of the object can then be generated by parametric basis functions set.

One of the advantages of free-form deformation is that the deformed object model is still parametric if the input object is parametric. The deformed object model will match with the original model. FFD can be used with any solid modeling scheme, surfaces or polygonal models. Moreover,

FFD maintains derivative continuity with adjacent, undeformed regions of the model when it is applied locally. FFD can provide true free-form model deformation. Although, FFD has many advantages, it has minor drawbacks. It is computational intensive. Moreover, it is difficult to achieve an exact shape that matches with idea of users because it is difficult to achieve exact placement of object points by carrying locations of control vertices. However, this problem can be solved by the algorithm proposed by Hsu, Hughes and Kaufman [24]. They proposed to allow users to specify the deformed object locations and the corresponding new control lattice is computed. Thus, a deformed model that matches with users' idea can be generated.

There are, in general, three types of parametric basis functions used in Free-Form Deformations. They are the Bernstein basis, the B-spline basis and the NURBS basis, with the Bernstein basis as the most commonly used basis.

2.4.1 Bernstein Basis

Sederberg and Parry [23] first proposed free-form deformation using Bernstein basis [27][30][37]. The Bernstein basis is defined as

$$J_{n,i}(v) = \binom{n}{i} v^i (1-v)^{n-i} \qquad \text{where} \qquad \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

Figure 2.4 depicts a set of Bernstein basis functions with $n = 5$ and $i = 0, 1, \ldots, 5$. This expression is derived from the de Casteljau Algorithm and the Blossoming principle [30][37]. Figure 2.5 shows the algorithm graphically. And, a Bézier curve [27][30] is defined as,

$$P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) \qquad 0 \leq t \leq 1$$

where B_i are the control vertices and n is the number of control vertices.

An imaginary parallelepiped is first defined by choosing a vertex on the box as its origin, $\vec{p}_0$ and the three edges emerging from the origin denote the directions of the axes of box's local coordinate frame, as depicted in Fig-2.6. Let $\hat{s}$, $\hat{t}$ and $\hat{u}$ be the three unit coordinate axes vector of the frame of the box. Any point, $\vec{x}$, in this system has coordinates (s, t, u) such that,

$$\vec{x} = \vec{p}_0 + s \cdot \hat{s} + t \cdot \hat{t} + u \cdot \hat{u} \tag{2.17}$$

And, the coordinates of $\vec{x}$ are (s, t, u) which are given as,

$$\begin{aligned} s &= \frac{(\hat{t} \times \hat{u}) \cdot (\vec{x} - \vec{p}_0)}{(\hat{t} \times \hat{u}) \cdot \hat{s}} \\ t &= \frac{(\hat{s} \times \hat{u}) \cdot (\vec{x} - \vec{p}_0)}{(\hat{s} \times \hat{u}) \cdot \hat{t}} \\ u &= \frac{(\hat{s} \times \hat{t}) \cdot (\vec{x} - \vec{p}_0)}{(\hat{s} \times \hat{t}) \cdot \hat{u}} \end{aligned} \tag{2.18}$$

Then, the imaginary parallelepiped is divided into l, m and n portions in the $\hat{s}$, $\hat{t}$ and $\hat{u}$ directions respectively l, m and n portions in the $\hat{s}$, $\hat{t}$ and $\hat{u}$ directions respectively. Thus, we have a grid of $(l+1)(m+1)(n+1)$ control points, $\vec{B}_{ijk}$, on the parallelepiped such that,

$$\vec{B}_{ijk} = \vec{p}_0 + \frac{i}{l} \cdot \hat{s} + \frac{j}{m} \cdot \hat{t} + \frac{k}{n} \cdot \hat{u} \tag{2.19}$$

where $0 \leq i \leq l$, $0 \leq j \leq m$ and $0 \leq k \leq n$.

Before finding the effect of free-form deformation on the object concerned, the control vertices on the parallelepiped lattice are moved and a new set of control vertices, $\vec{B'}_{ijk}$, is obtained. Based on the new control lattice, the object concerned will become

$$\vec{x}_{FFD} = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} \left[\binom{l}{i} \binom{m}{j} \binom{n}{k} (1-s)^{l-i} s^i (1-t)^{m-j} t^j (1-u)^{n-k} u^k \vec{B'}_{ijk} \right] \tag{2.20}$$

The object concerned is modelled by a tensor product of trivariate Bernstein polynomials [38]. Due to the global nature of Bernstein basis, the new location of each point on the object depends on all displaced control vertices. Thus, FFD using Bernstein basis is a global deformation. Moreover, the orders of polynomials defining the object are limited by the number of control vertices. This lowers the flexibility of FFD.

2.4.2 B-Spline Basis

Griessmair and Purgathofer [21] proposed to use B-spline basis for free-form deformation, instead of using Bernstein basis. This can increase the flexibility of the model as B-spline has a local control property. Moreover, we can control the shape of B-spline curve by

- changing the type of knot vector (clamped or unclamped and uniform or nonuniform) used.
- changing the order of the basis function.
- changing the number and position of control vertices.
- using multiple control vertices.
- using multiple knot values in knot vector.

For the i-th normalized B-spline basis function of order k, the basis functions $N_{i,k}(t)$ are defined by the simple Cox-deBoor recursion formulas [27][30][37],

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } x_i \leq t < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}$$

where $[x_1, x_2, \cdots, x_{n+k+1}]$ is the knot vector. There are $(n+1)$ control vertices to define a B-spline curve. The whole B-spline curve, with B_i as control vertices, is defined as,

$$P(t) = \sum_{i=1}^{n+1} B_i N_{i,k}(t) \qquad t_{min} \leq t < t_{max}, \qquad 2 \leq k \leq n+1$$

Usually, the knot vector used is of clamped uniform type [27][30] and is defined as,

$$x_i = \begin{cases} 0 & 1 \leq i \leq k \\ i-k & k+1 \leq i \leq n+1 \\ n-k+2 & n+2 \leq i \leq n+k+1 \end{cases} \tag{2.21}$$

Fig-2.7 depicts the differences B-spline basis functions using different types of knot vectors.[1] In fig-2.7(b), we can see that the B-spline basis functions are similar except with a constant shifts along the parameter axis. In fig-2.7(d), we can see that there is a cusp at the multiple knots.

Given an imaginary control lattice, $\vec{B}_{ijk}$ which is similar to its Bernstein basis counterpart, the object is modelled by a trivariate B-spline polynomial with orders k_x, k_y and k_z in the x-, y- and z-directions respectively.

$$\vec{x}_{BFFD}(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u) \tag{2.22}$$

The parameter coordinates triple of each object point cannot be found by (2.18) because the parameter ranges in the three directions are governed by the knot vectors in the corresponding axes. The knot vectors

[1] The knot vectors used in fig 2.7 are listed as follows,

1. For fig-2.7(a), knot vector = $[0,0,0,0,0.25,0.5,0.75,1,1,1,1]$;
2. For fig-2.7(b), knot vector = $[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]$;
3. For fig-2.7(c), knot vector = $[0,0,0,0,0.175,0.35,0.625,1,1,1,1]$;
4. For fig-2.7(d), knot vector = $[0,0,0,0,0.575,0.575,0.575,1,1,1,1]$

may not be open uniform and the order of the B-spline may not equal to the number of control vertices in that direction.[2] The parameters coordinates triples can be found by solving the following nonlinear vector equation,

$$\sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1} N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u)\begin{bmatrix} B^x_{ijk} \\ B^y_{ijk} \\ B^z_{ijk} \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \tag{2.23}$$

where $[q_x\, q_y\, q_z]^T$ is a object point. Equation (2.23) can be solved by numerical method. This is called the Point Inversion Problem [37]. However, this is a computational expensive process. An iterative solution for the point inversion problem is proposed in section (4.2.2).

In order to further enhance the flexibility of the model, Lamousin and Waggenspack, Jr. [20] proposed to use NURBS (Non-uniform rational B-spline) basis [27][30][37] in free-form deformation. This is called NFFD. In this case, the knot vector used is non-uniform and the a set of weights is imposed on each control vertex. If a control vertex has a large weight relatively to other vertices, the shape generated will be closer to that vertex. The NURBS based FFD model is defined as,

$$x_{NFFD}(s,t,u) = \sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1} \vec{B}_{ijk}R_{ijk}(s,\, t,\, u) \tag{2.24}$$

where

$$R_{ijk}(s,\, t,\, u) = \frac{h_{ijk}N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u)}{\sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1} h_{ijk}N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u)} \tag{2.25}$$

In this project B-spline-based FFD (BFFD) is employed in fine manipulation of the coarse and global object model represented by superquadrics.

[2] When the number of control vertices equals to the order of the B-spline basis and an open uniform knot vector is used, the B-spline basis reduces to a Bernstein basis of the corresponding order.

2.5 Other Geometric Models

In addition to the above mentioned geometric models, there are many other models in the world. Each type of geometric model can has advantages or drawbacks in the representation certain kinds of objects over the other kinds. Some models represents polyhedral objects well and some models represent objects with curved objects. In the following, a few common geometric models will be discussed.

2.5.1 Generalized Cylinders

Generalized Cylinder [39][40] is one of the most widely adopted representation model for solids bounded by curved surfaces. In general, a generalized cylinder is the solid obtained by sweeping a planar region (called its cross section) along a space curve (called its axis). Moreover, the size of each cross section need not be the same and it is governed by a sweeping rule. Even the definition of each cross section may not be the same. Adjacent cross sections may be different not only in their size, but also in their shapes. In addition, the axis of a generalized cylinder is not necessarily straight or planar. This 3D shape model is flexible.

Although generalized cylinder offers great flexibility to model representation, some limitations are introduced to the model in order to ease the complexity of model recovery problems. One of the common employed generalized cylinder subclass is the straight homogeneous generalized cylinder (SHGC). The definition of SHGC is the solid swept by a planar cross section as it is translated and scaled along a straight axis. To further simplify the model, each cross section is assumed to be orthogonal to the axis. With respect to the Cartesian coordinate system, the mathematical formulation of a SHGC is as follows.

$$SHGC(z,\theta) = \rho(\theta)\, r(z)\, (\cos\theta\, \hat{x} + \sin\theta\, \hat{y}) + z\, \hat{z} \qquad (2.26)$$

where $(z, \theta) \in [a, , b] \times [0, 2\pi]$. $\rho(\theta)$ is the planar cross section of the SHGC. It is parametrized in a polar coordinate system centered on the axis. $r(z)$ is the sweeping rule and it is parametrized along the z-direction (which is parallel to the axis of generalized cylinder) and bounded by the planes $z = a$ and $z = b$. Each cross section is parallel to the $x - y$ plane. Moreover, the axis of the SHGC can be oblique to the z-axis. Let $\vec{v}$ be the axis of the oblique SHGC which has (α, β) as its spherical coordinates, its mathematical expression is,

$$\begin{aligned} SHGC(z,\theta) &= (\rho\, r \cos\theta + z \sin\beta \cos\alpha)\, \hat{x} \\ &\quad + (\rho\, r \sin\theta + z \sin\beta \sin\alpha)\, \hat{y} \\ &\quad + z \cos\beta\, \hat{z} \end{aligned} \tag{2.27}$$

2.5.2 Hyperquadrics

Hyperquadrics [41][42][43] is a new approach to model smoothly deformable shapes with convex polyhedral bounds. The possible shape classes include arbitrary convex polyhedra with or without deformations, like tapering and bending, and complex shapes can be built by applying regularized Boolean operations to different object shapes. This model can be viewed as hyperplanar slices of deformed hyperspheres. This model is a generalization of superquadrics.

A hyperquadric model [42] is defined by the points set (x, y, z) that satisfies,

$$H(x,y,z) = \sum_{i=1}^{N} |H_i(x,y,z)|^{\gamma_i} = 1 \tag{2.28}$$

where

$$H_i(x,y,z) = a_i\, x + b_i\, y + c_i\, z + d_i \tag{2.29}$$

where a_i, b_i, c_i, d_i and γ_i are constants, with $\gamma_i > 0$. This indicates that the

model is constructed by a sum of an arbitrary number of linear terms raised to powers. Each linear term defines a strip bounded by the two hyperplanes $H_i(x, y, z) = 1$ and $H_i(x, y, z)$. Thus, the hyperquadric model is inside the intersection of these strips which is an arbitrary convex polytope and N is the number of these strips. If all the exponents γ_i are greater than 1, the hyperquadric model represent a convex shape. On the other hand, when there are exponents that are smaller than 1, non-convex shapes will be obtained. In order to increase the flexibility of the model, exponential of hyperquadric terms are introduced to the model. These terms give local control and concavities of the shape. The modified model is,

$$H(x, y, z) = \sum_{i=1}^{N} |H_i(x, y, z)|^{\gamma_i} + \sum_{j=1}^{M} a_j e^{-\sum_{k=1}^{L_j} |K_{jk}(x,y,z)|^{\gamma_{jk}}} = 1 \qquad (2.30)$$

where H_i and K_{jk} are the linear form defined as (2.29), N is the number of strips defining the hyperquadric, M is the number of concavities used. In addition, L_j and a_j are the number of strips defining and the size of the jth concavity. Figure 2.8 depicts some hyperquadric models. Models in the lower row involve exponentials of hyperquadric terms (2.30) while models in the upper row involve ordinary hyperquadric terms only (2.28).

One of the disadvantages of the hyperquadric model is its global nature. In the process of model recovery, we cannot fine tune the recovered shape at a particular region by adjusting part of the shape parameters. The whole process of model recovery (usually it is an optimization problem with a lot of parameters to be determined) should be conducted again. Moreover, this model is not quite flexible. Although complex object shapes can be represented by combining several hyperquadric models with regularized Boolean operations (including union, difference and intersection), the computation of model recovery will be further complicated.

2.5.3 Polyhedral Models

Polyhedral model is another common shape representation model. As implied in its name, it is good at representing objects with polyhedral shapes. However, there are few real world objects are in pure polyhedral shapes. Real world objects usually are made of curves (as edges) and surfaces(as faces), instead of straight line segments as edges and plane as faces. Although we can approximate objects with curved surfaces by polyhedron with large number of faces, this will increase the storage of the model dramatically. This will be elaborated in the followings.

We can store up polyhedral shape data (geometric and topological information) by the Boundary Representation scheme (B-rep) [33][34]. B-rep scheme stores the geometric and topological data of vertices, edges and faces and their interrelationship. The most fundamental part of B-rep scheme is the edge model for edge information storage. An efficient edge model can shorten the transversal and searching time for vertices and edges during operations on the model. The most commonly used edge model is the winged-edge model, which was developed by Baumgart [34]. This model makes use of the edge-topological information for representation of the bounding surface of an arbitrary polyhedron. Each face in a polyhedron is bounded by a chain of edges and each vertex is the intersection of two adjacent edges. Using this edge model, each face of the polyhedron can identify all its immediate neighbouring faces directly through its bounding edges. Moreover, directional information of each edge are augmented in this model via pointers to succeeding and preceding edges on each of the two faces it connects[3]. The structure of the winged edge model is depicted

[3]That means each edge is part of boundaries of exactly two immediate adjacent faces in a well-formed polyhedron. This is due to the Möbius' law, which states that a closed surface is topologically consistent oriented if by transversing its triangles (resulting from an arbitrary triangulation) in a clockwise direction, each edge is transversed exactly twice and in opposite directions. Orientability is one of the validity conditions for a well-formed object

in Figure-2.9. In this figure, we can see that the following information is stored, namely, incident vertex, left and right adjacent faces, preceding and succeeding edges both in clockwise and counterclockwise orders.

With the wing edge model, a data structure for representation of a polyhedron can be established. The core part of the representation is linked list. The vertices, edges and faces are stored in corresponding linked lists. The geometric information of these features are stored in the nodes of the lists, while their topological information is represented by the inter-node pointer references in one list. There are also inter-list pointer references among the vertex, edge and face lists to indicate the topological relationship among particular vertices, edges and faces.

Although we can approximate objects with curved surfaces boundary by polyhedron with large number of faces, this will need a large amount of storage space for the whole object model using B-rep scheme. As we can see that the amount of storage space required for a polyhedral model depends on the number of faces, edges and vertices of the model, B-rep is an expensive model in storage view. The number of nodes in the vertex, edge and face lists will be very large, especially for the edge list, which includes the topological information of the polyhedral model.

2.5.4 Function Representation

Function Representation (F-rep) [44] is another type of geometric model which has been developed recently. Objects are defined by the halfspace specified by a multivariate real function. An object is a closed point set in n-dimensional Euclidean space which is defined by the halfspace defining inequality,

$$f(x_1, x_2, x_3, \ldots, x_n) \geq 0 \tag{2.31}$$

and the equation $f(x_1, x_2, \ldots, x_n) = 0$ is the bounding hypersurface, which has dimension of $(n-1)$, of the hypervolume defined by (2.31). The function

f is called the defining function of the object. Usually, researchers are interested in the 3-dimensional case, ie. $f(x_1, x_2, x_3) \geq 0$. It is noticed that F-rep is a generalization of hyperquadric model, which is a generalization of superquadric model[4] . F-rep is an implicit representation of object model. A simple example of F-rep is an unit sphere with radius r, which locates at the coordinate (a, b, c) of the Cartesian coordinate system (x, y, z) and it is defined as

$$r^2 - (x-a)^2 - (y-b)^2 - (z-c)^2 \geq 0$$

Complex objects can be built by extending F-rep to cooperate with Constructive Solid Geometry (CSG) [1]. The primitives of CSG are then represented by F-rep. Complex objects are built on these F-rep primitives by Boolean operations, like union, intersection, complement and difference [1]. Every object can be represented by a CSG tree with each node denotes a primitive or a Boolean operation.

Boolean operations include union, intersection, complement and difference. Define f_1 and f_2 as defining functions of two F-rep primitives and f_3 as a resultant object after operation, we have

$$\begin{aligned} f_3 &= \sim f_1 && \text{for complement} \\ f_3 &= f_1 \cup f_2 && \text{for union} \\ f_3 &= f_1 \cap f_2 && \text{for intersection} \\ f_3 &= f_1 \setminus f_2 && \text{for difference} \end{aligned} \tag{2.32}$$

These Boolean operations can be implemented by special kind of functions called $\mathcal{R}$-functions[45][44][46]. There are several descriptions for these $\mathcal{R}$-functions. One of them is defined as

$$\sim f_1 = -f_1$$

[4]The hyperquadric and superquadric model actually represent the bounding surfaces of objects. F-rep covers the interior of the objects

$$f_1 \cup f_2 = \frac{1}{\omega+1}\left(f_1 + f_2 + \sqrt{f_1^2 + f_2^2 - 2\omega f_1 f_2}\right) \quad (2.33)$$
$$f_1 \cap f_2 = \frac{1}{\omega+1}\left(f_1 + f_2 - \sqrt{f_1^2 + f_2^2 - 2\omega f_1 f_2}\right)$$

where $\omega(f_1, f_2)$ is an arbitrary continuous function which lies within the range of $(-1, 1]$ and,

$$\omega(f_1, f_2) = \omega(f_2, f_1) = \omega(-f_1, f_2) = \omega(f_1, -f_2)$$

The difference operation can be considered as $f_1 \setminus f_2 = f_1 \cap (\sim f_2)$. Using different ω's will give different shapes. The most common implementation of these Boolean operations is to set $\omega = 0$, ie.,

$$f_1 \cup f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \quad (2.34)$$
$$f_2 \cap f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} \quad (2.35)$$

This implementation, however, has C^1 discontinuity in points where both f_1 and f_2 equal to zero. If C^k continuity is maintained in the operations, the $\mathcal{R}$-functionsbecome,

$$f_1 \cup f_2 = \left(f_1 + f_2 + \sqrt{f_1^2 + f_2^2}\right)(f_1^2 + f_2^2)^{\frac{k}{2}} \quad (2.36)$$
$$f_2 \cap f_2 = \left(f_1 + f_2 - \sqrt{f_1^2 + f_2^2}\right)(f_1^2 + f_2^2)^{\frac{k}{2}} \quad (2.37)$$

This object representation scheme is simple and compact. Only the definition of the defining function need to be stored. Even for complex object, only the extra storage of the CSG tree it is needed. In case, only a certain kind of functions is allowed for the construction of the defining function of a F-rep object, say polynomial of degree n, only $(n+1)$ real coefficients of the defining function is stored for each CSG primitive. However, this also limits the geometric coverage of this modelling scheme. F-

rep also is not enough to exclude representation of nonmanifold solids. Even though all the CSG primitives in F-rep are guaranteed as manifold solids, it cannot ascertain that the resultant objects are manifold solids after Boolean operations.

Moreover, F-rep cannot facilitate the model reconstruction process [47][48]. It is because a set of defining functions types, say an nth order polynomial, should be pre-defined before model recovery. Then, the shape controlling parameters can be recovered by minimizing the square error between the data points and the recovered shape. In order to have good reconstruction of the object model, priori information or intuition about the object shape should be given. For instances, the object needed to be reconstructed can by viewed as an union of two primitives. Usually, it is difficult to assume the defining function types for recovery. In addition, it is usual that no priori information is given to the reconstruction system. Only a large set data points are used as input information. In this case, some more advanced and complicated methods should be employed, like calculus of variations. The variational approach involves heavy computation and usually efficient numerical solution of sophisticated partial differential equations are required.

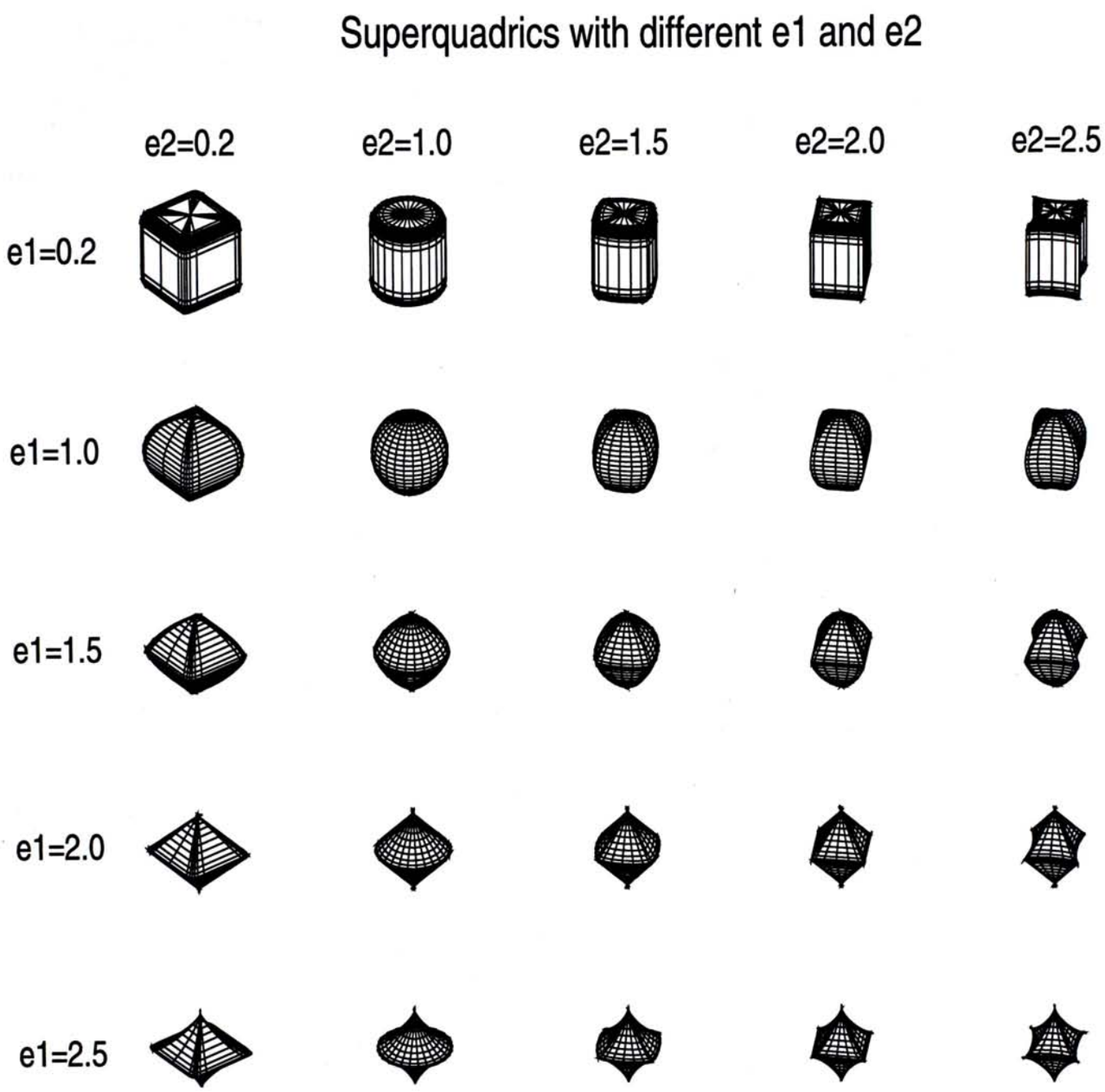


Figure 2.2: Superquadric model with different ϵ_1 and ϵ_2

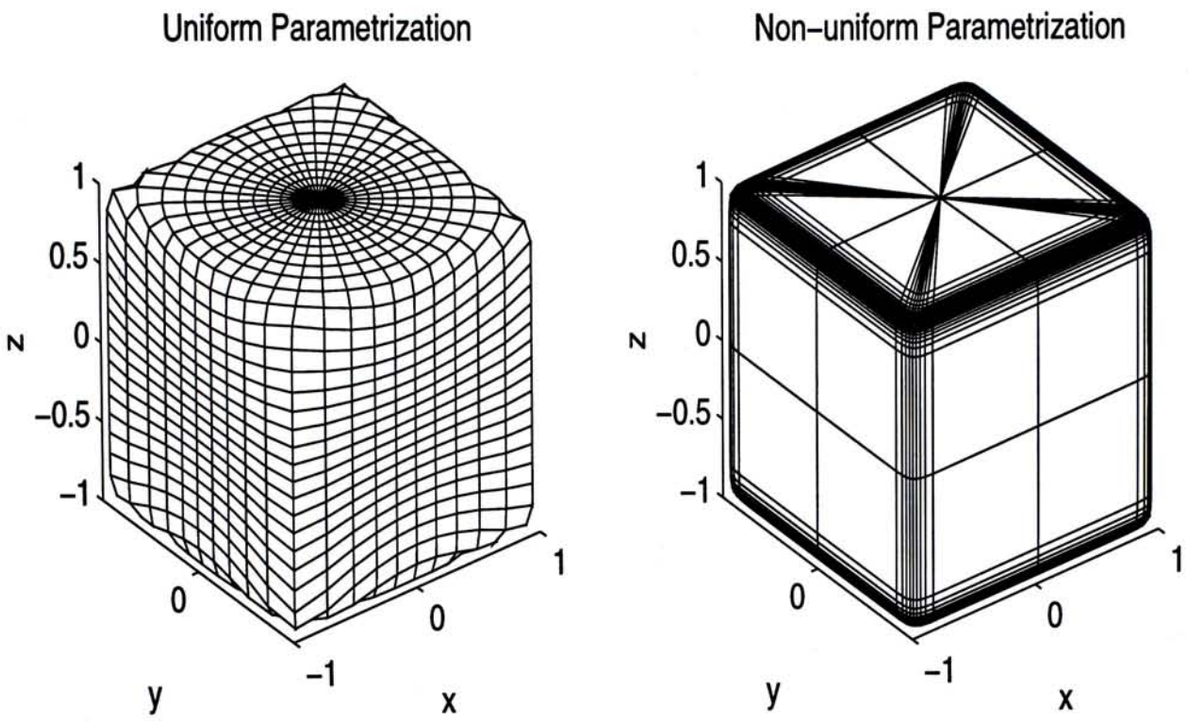


Figure 2.3: Non-uniform and regular parameterizations of the same cube.

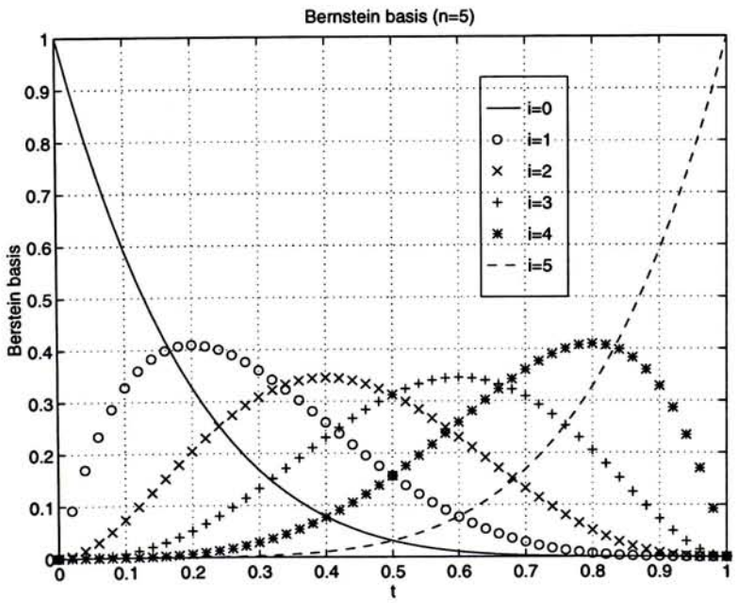


Figure 2.4: Bernstein basis when $n = 5$

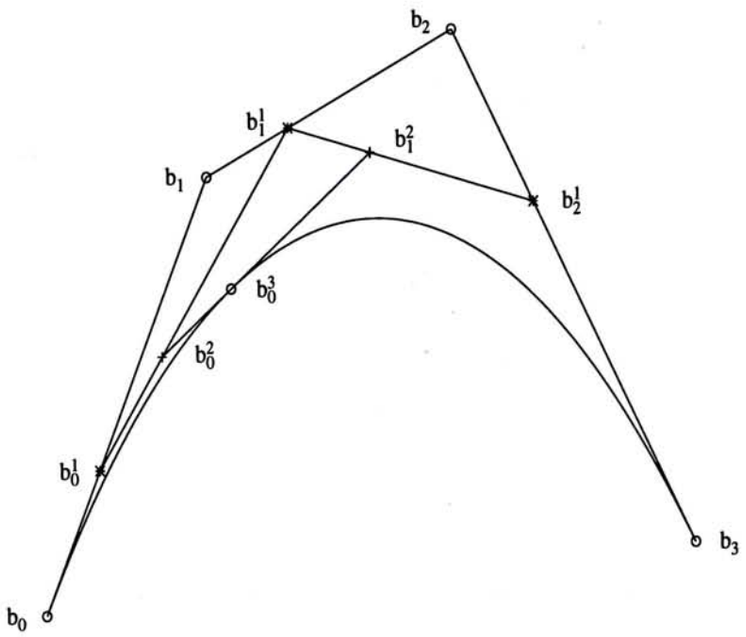


Figure 2.5: The de Casteljau Algorithm – The cubic case with $t = \frac{1}{3}$ and $t \in [0, 1]$

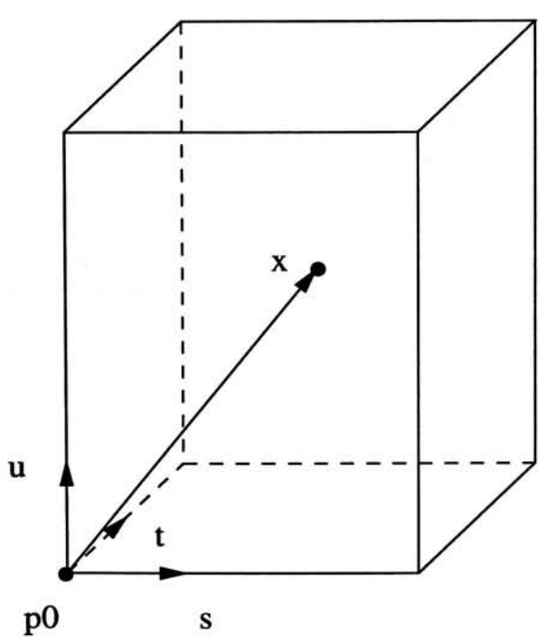
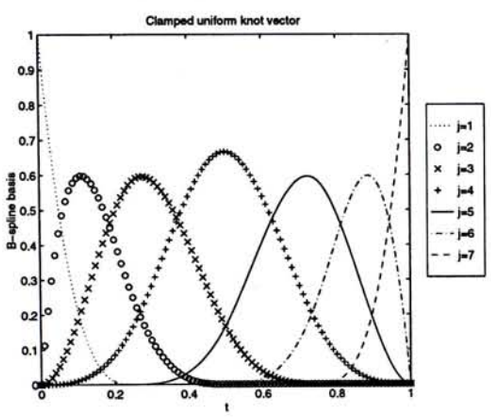
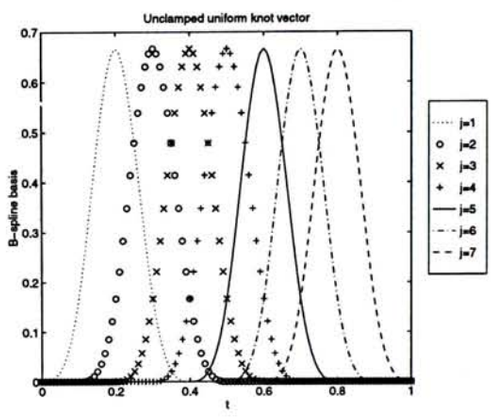


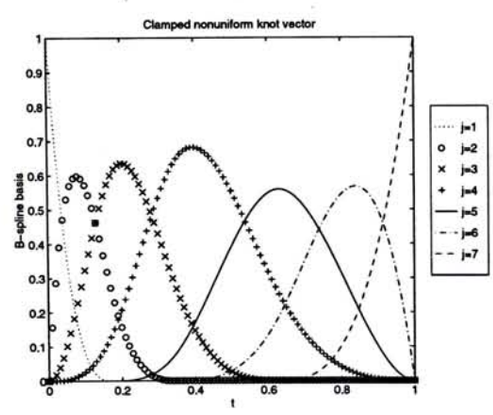
Figure 2.6: The imaginary parallelepiped for FFD.



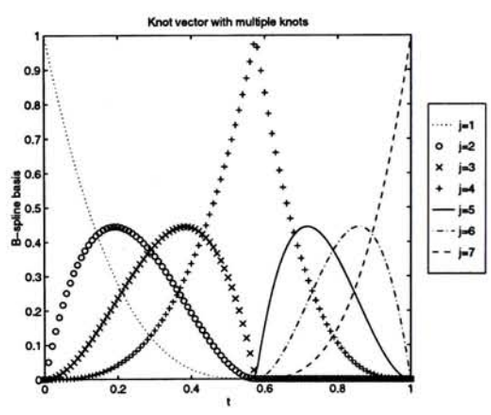
(a) Using clamped uniform knot vector



(b) Using unclamped uniform knot vector



(c) Using nonuniform knot vector



(d) Using knot vector with multiple knots

Figure 2.7: B-spline basis for different knot vectors.

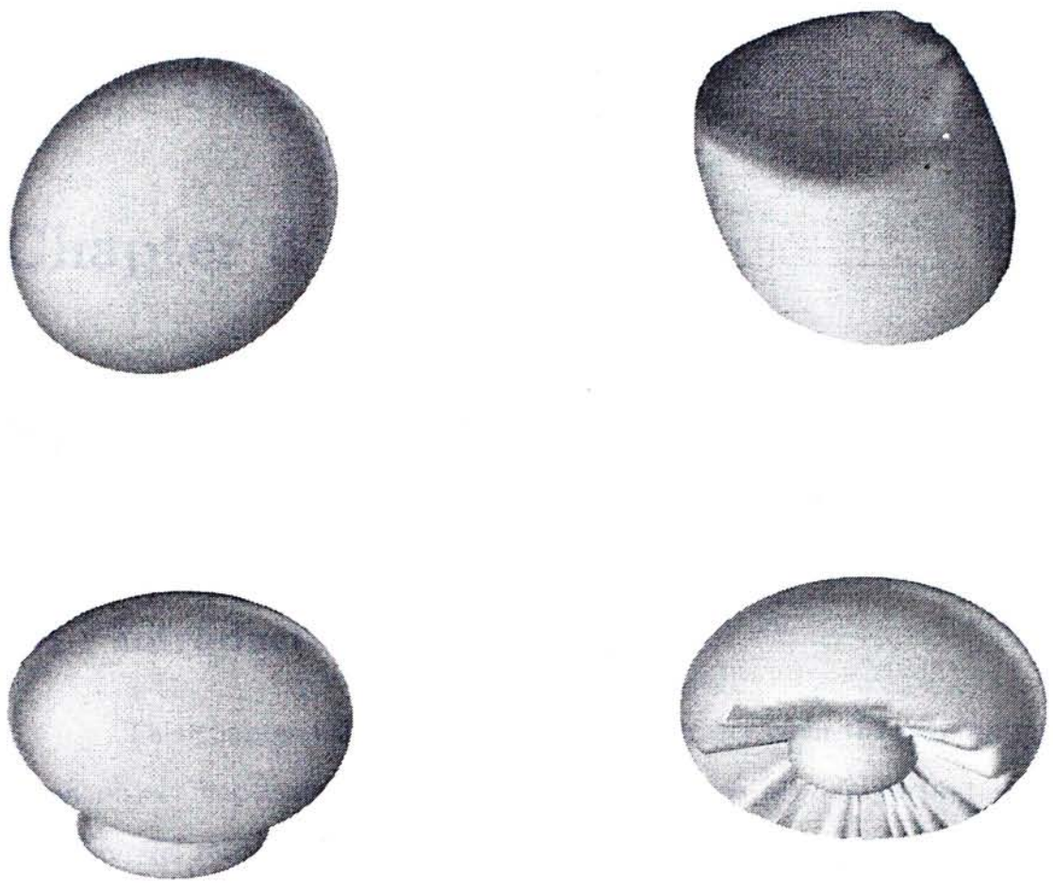


Figure 2.8: Several Hyperquadric models

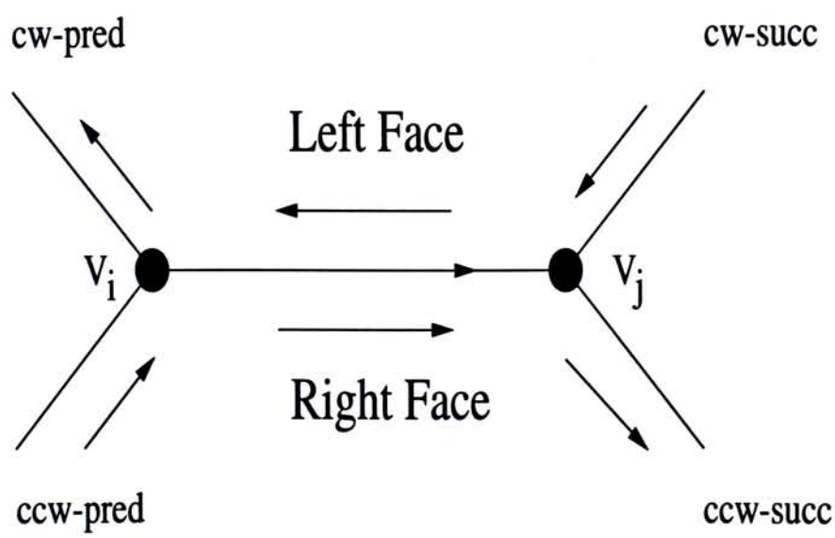


Figure 2.9: The Winged Edge model (Courtesy from "Geometric and Solid Modeling" by Christoph M. Hoffmann, published by Morgan Kaufmann [1])

Chapter 3

Sensing Strategy

3.1 Introduction

This chapter will describe the sensing strategies in the exploration of an object haptically. Efficient sensing strategies play important roles in 3D model reconstruction with all sensing means, including active vision sensing and touch sensing. In the process of 3D model reconstruction, the more the data of object shape is collected, the more accurate and reliable the reconstructed object model. However, the effort in data analysis and the computation time increase drastically with the amount of shape data obtained. There should be a compromise between the amount of data need to be collected and the computation effort involved. Efficient sensing strategies can guide the sensing devices (the multi-fingered dextrous robotic hand with tactile sensor arrays mounted on fingertips in this dissertation) to find data necessary and sufficient for 3D model reconstruction intelligently. Sensing strategies are a set of decision making algorithms, optimization and search algorithms or even heuristic rules for data acceptance, rejection and redundancy checking so that the amount of information required for reconstruction is as little as possible.

3.2 Sensing Algorithm

In this dissertation, a haptic sensing algorithm for object exploration is developed. The main idea of the algorithm is to obtain object surface points coordinates from contours tracing on the object surface. A few questions arise at this stage. The questions include how many contours are needed to be traced on the object surface and where to start each contour tracing. The density of the contours traced is non-uniform. The separation or step size between adjacent contours depends on the curvature and change in curvature (actually their average along a contour). The determination of curvature and other shape information of a surface point from tactile sensor data is described in section 3.5. Before any processing and analysis of the tactile sensor data, noise-suppressing filters are applied to the sensor data in the preprocessing stage. On the other hand, the determination of step size between contours can be referred to section 3.6. Moreover, a haptic exploration procedure is developed to find the optimal distribution of contours traced so that as less data as possible is obtained for the reconstruction of best-fit 3D object models. The EP can be referred to section 3.2.2. In the following sections, details of each part of the sensing algorithm is presented. In addition, there are some assumptions imposed on the object being tested and they are listed in the next section (section 3.2.1).

3.2.1 Assumption of objects

In order to simplify the problem scenario, there are four assumptions imposed on the objects that are under haptic exploration.

- The objects should be rigid bodies. They are undeformable and inelastic.
- The shape of the objects should be convex or with little concavities and topologically homeomorphic to a sphere. This means that the objects

should not have any genus on it. For example, a rectangular box is topologically homeomorphic to a sphere while a cup with a handle is not. A cup has a genus at its handle. This assumption can simplify the 3D model reconstruction process.

- They are placed on the test platform with a proper orientation and position. That means the object is placed on the platform with its major axis or minor axis aligned with the axes of the platform and it locates at the center of the platform. For instance, a rectangular box is placed on the center of a rectangular platform with their sides parallel to each other. This saves the computational effort for recovering the position and orientation of the object. The exploration strategies are only concentrated on acquisition of shape information.

- The weight of the object is small enough so that the robotic hand can lift it up from the test platform and manipulate it.

3.2.2 Haptic Exploration Procedures

The exploration procedure developed in this dissertation is described below. Details of the calculation of normal vector and curvature of a surface point can be referred to section 3.5.

1. Perform a Grasp by Containment EP (described in section 1.4.3.2) to find the general shape of the target object described by a superquadric model and the size of the object. This recovered superquadric model will become the original shape model for Free Form Deformation for fine shape tuning. This EP is simplified by restricted the possible values of superquadric model shape controlling parameters ϵ_1 and ϵ_2 to a few pairs. In this dissertation, three common shape is selected. They are sphere ($\epsilon_1 = 1$ and $\epsilon_2 = 1$), box ($\epsilon_1 = 0.2$ and $\epsilon_2 = 0.2$) and cylinder ($\epsilon_1 = 0.2$ and $\epsilon_2 = 1$). Detailed solution of the recovery of

superquadric model parameters can be referred to section 2.3.

2. Move one of the finger (usually the first finger) to the level which is same as half of the object height. Approach the finger to the object surface. Record this height level as h_{mid}.

3. Trace a contour along the the object surface at the same level. Record the coordinates of all, say m, surface points along the contour. The coordinates obtained are with respect to the fingertip coordinate frame. These 3D point coordinates can be transformed with respect to the world coordinate frame.

4. Determine the vertical search direction for next contour (upwards or downwards). The search direction can be determined in terms of the normal vectors of the surface points along the above traced contour. The normal vectors, $\hat{n}_i$ $i = 1, 2, \ldots, m$, can be calculated by (3.11). Then, the average of these normal vectors, $\hat{n}_{avg} = \frac{\sum_{i=1}^{m} \hat{n}_i}{m}$ is computed. This average normal vector is then compared with the z-axis to determine the search direction. An indicator is introduced and it is denoted as $\phi = \hat{n}_{avg} \cdot \hat{z}$. If $\phi > 0$, the search direction is set to upwards. If $\phi < 0$, the search direction is set to downwards. If $\phi = 0$, either upward or downward direction can be chosen. Records this search direction.

5. Determine the step size for next contour. In general, the step size depends on both the average curvature and average change in curvature. In this step, the first thing to do is to estimate all curvature at the traced surface points. The estimation of curvature of a surface point from tactile sensor data can be found in section 3.5. Then, average the curvature to have κ_{avg}.

(a) If this is the first contour trace, the step size θ^1 is set as

$$\theta^1 = \frac{h \cdot s_1(\kappa_{avg})}{2(1+\eta)} \tag{3.1}$$

where s is obtained from the first step size function component which depends only on the average curvature estimated, h is total height of the object and η is a confidence parameter for step size. It is usually a positive real number and the larger the η, the smaller the step size obtained.

(b) For other contours, find $\triangle\kappa_{avg}$ by taking the difference between the average curvature estimated and the previous average curvature obtained. The step size at the kth contour θ^k is given by,

$$\theta^k = s(\kappa_{avg}, \triangle\kappa_{avg}) \cdot h \tag{3.2}$$

where $s(\kappa_{avg}, \triangle\kappa_{avg})$ is the step size function.

6. Repeat step 5 until the step size determined is smaller than a predefined threshold step size function value for upward searching or the robotic hand touches the testbed platform and it cannot move any further for downward searching. Return the finger to the level at h_{mid} and begin a new search in opposite search direction following step 1 to step 5

3.3 Contour Tracing

There is an assumption about the contour tracing procedure is that an efficient contour tracing is built-in in the haptic exploration system. Using this tracing algorithm, 3D coordinates of a series of surface points on the object along a contour can be obtained for 3D model reconstruction. The distance between adjacent contours depends on the curvature κ and

change in curvature $\triangle\kappa$ determined from tactile sensor data obtained during contour tracing (see section 3.5 below). The reason of contour tracing is to find uniformly arranged data. The data point can be gridded and well organized. This can simplify the process of 3D data interpolation and/or approximation. They are crucial steps in 3D model reconstruction. Gridded data usually facilitate data interpolation and approximation.

3.4 Tactile Sensor Data Preprocessing

In real world, there are several kinds of tactile sensors. They have different working mechanisms and usages. Details can be referred to Section 1.1.2. The tactile sensor array, that I intend to use, consists of an array of 16×16 sensing elements with a rubber pad on top of them. It is a two-dimensional array of ultrasound transmitters and receivers to measure the thickness of an overlaying rubber pad. When objects contact the sensor's pad, the rubber is compressed. The amount of compression depends on the force magnitude applied to the object and the stiffness of the rubber pad. Each sensing element transmits an ultrasonic pulse which travels through the rubber pad, is reflected off the top surfaces of the rubber. Then, the time of flight of the pulse can be measured, which is proportional to the applied force. Then, force pattern applied to the object can be revealed by the sensor array data.

Each sensor data acquired is in form of 16×16 array of force values. Before preprocessing of the raw force data, it should be transformed to the geometric coordinates of the surface points. As the raw data is noisy, appropriate noise-suppressing filters are applied to the data. The filters include Adaptive Wiener filter and Median filter. Then, a B-spline surface is fitted on the filtered data array. The resultant surface can approximate the local shape of the object. Figure 3.1 depicts the tactile sensor data preprocessing procedure.

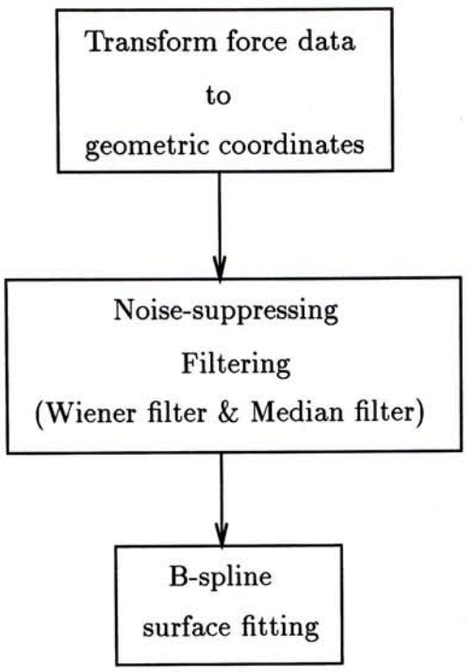


Figure 3.1: Tactile sensor data preprocessing procedures

3.4.1 Data Transformation and Sensor Calibration

As the raw data from tactile sensor array is force values obtained from the 256 sensing elements, it should be transformed to geometric domain. This can be achieved by careful calibration of the tactile sensor array. When an object contacts with the rubber pad of the sensor array, the rubber pad is compressed. The normal force recorded by each sensing element is actually proportional to the amount of compression above this sensing element. The rubber pad deforms to match with the shape of the local object surface patch. A coordinate system is defined as in figure 3.2. We can imagine that the local object surface patch touching the rubber patch surface is parametrized by an uniform grid. This can be constructed by extending a perpendicular line from each sensing element to the rubber pad surface. The coordinates of the intersection points can be found as follows. The array of 3D points can be labelled according to the node of the grid. The Cartesian coordinates of these surface points can be found easily. The x- and y-coordinates of a particular surface point are the same of its

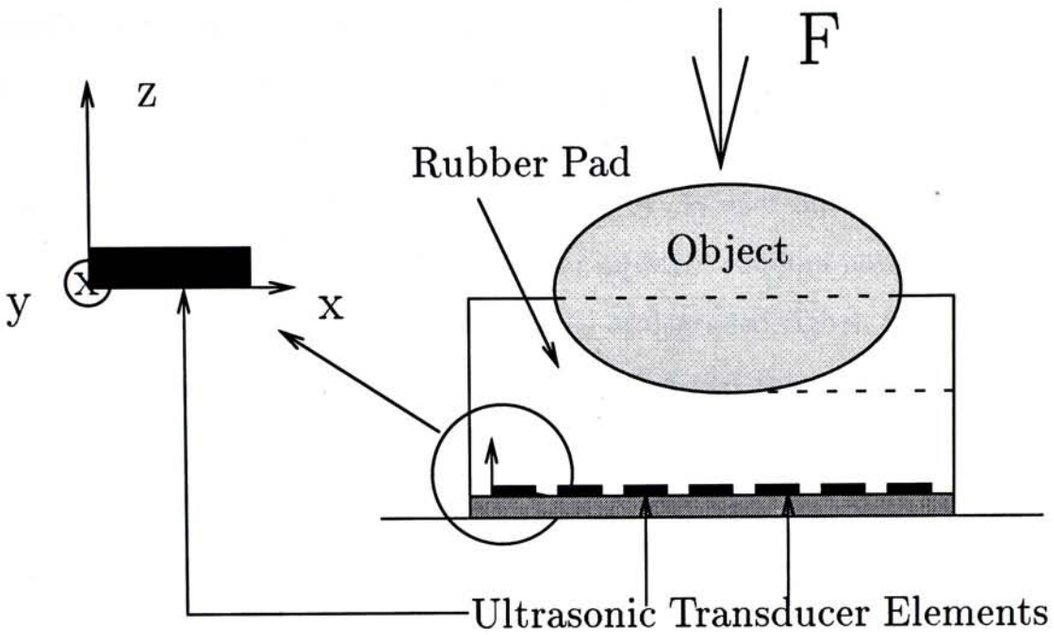


Figure 3.2: Coordinate frame defined in the tactile sensor array

corresponding sensing element. The coordinates of the sensing element are expressed with respect to the working frame of the fingertip. With careful sensor calibration, the output of the sensing element can be scaled to be the same of the z-coordinates of that surface point.

3.4.2 Noise Filtering

Output from all kinds of sensors is noisy, including tactile sensor arrays. The noise should be filtered out or suppressed in the sensor data before performing any operation on and drawing interpretation from the data. This is because sensor data preprocessing is a necessary step for any system involving sensors input for processing. Tactile sensors data preprocessing stage is thus a crucial part of this project.

There are several types of noise filters that can remove or suppress different kinds of noises. Two of them are Adaptive Wiener Filtering and Median Filtering [49]. Adaptive Wiener filtering can reduce additive noise well by estimating local details ils of the noise-free data. On the other hand,

median filtering can reduce salt-and-pepper noise by selecting the median intensity in a sliding window on the data.

In general, Wiener filter can give the optimal linear minimum mean square error estimate of a signal by Wiener filtering its noise contaminated version (signal independent additive random noise) [49]. For two dimensional discrete signal, the model of degraded signal $f(n_1, n_2)$ can be consructed from the original signal $s(n_1, n_2)$ and the noise $v(n_1, n_2)$ as follows,

$$f(n_1, n_2) = s(n_1, n_2) + v(n_1, n_2) \tag{3.3}$$

The frequency response of Wiener filter is given as,

$$H_{Wiener}(\omega_1, \omega_2) = \frac{P_s(\omega_1, \omega_2)}{P_s(\omega_1, \omega_2) + P_v(\omega_1, \omega_2)} \tag{3.4}$$

where $P_s(\omega_1, \omega_2)$ and $P_v(\omega_1, \omega_2)$ are the power spectra of original signal and the noise respectively. These power spectra are usually unknown in priori. Fortunately, they can be estimated locally by adaptive wiener filter algorithm.

Assume the additive white noise $v(n_1, n_2)$ has zero mean and a variance of σ_v^2. Then, we have $P_v(\omega_1, \omega_2) = \sigma_v^2$. For a small local region of the signal, it can be assumed stationary and the signal is modelled as,

$$s(n_1, n_2) = m_s(n_1, n_2) + \sigma_s(n_1, n_2) \cdot w_n(n_1, n_2) \tag{3.5}$$

where m_s and σ_s are the local mean and standard deviation of $s(n_1, n_2)$ and $w_n(n_1, n_2)$ is a white noise with zero mean and a variance of 1. The wiener-filtered signal $r(n_1, n_2)$ is,

$$r(n_1, n_2) = m_s(n_1, n_2) + \frac{\sigma_s(n_1, n_2)}{\sigma_s(n_1, n_2) + \sigma_v} (f(n_1, n_2) - m_s(n_1, n_2)) \tag{3.6}$$

or $r(n_1, n_2) = f(n_1, n_2) \otimes h(n_1, n_2)$ where $h(n_1, n_2)$ is the impulse response

of the adaptive wiener filter and it is expressed as,

$$h(n_1,n_2)=\begin{cases}\frac{\sigma_s^2+\frac{\sigma_v^2}{(2N+1)^2}}{\sigma_s^2+\sigma_v^2} & , n_1=n_2=0\\ \frac{\frac{\sigma_v^2}{(2N+1)^2}}{\sigma_s^2+\sigma_v^2} & , n_1,n_2\neq 0, -N\leq n_1,n_2\leq N\\ 0 & , \text{otherwise}\end{cases} \tag{3.7}$$

where N is the half window size of the filter. We can locally estimate $\hat{\sigma}_s^2$ by $\hat{\sigma}_f^2(n_1,n_2)-\hat{\sigma}_v^2$ if $\hat{\sigma}_f^2>\hat{\sigma}_v^2$, otherwise, it is zero. The power spectrum of the degraded signal is,

$$\hat{\sigma}_f^2(n_1,n_2)=\frac{1}{(2N+1)^2}\sum_{k_1=n_1-N}^{n_1+N}\sum_{k_2=n_2-N}^{n_2+N}(f(k_1,k_2)-\hat{m}_s(n_1,n_2))^2 \tag{3.8}$$

where

$$\hat{m}_s(n_1,n_2)=\frac{1}{(2N+1)^2}\sum_{k_1=n_1-N}^{n_1+N}\sum_{k_2=n_2-N}^{n_2+N}f(k_1,k_2) \tag{3.9}$$

Median filtering is actually a nonlinear operation. It reduces salt-and-pepper noise by setting the intensity of a pixel as the median intensity in a region bounded by a sliding window centered at that pixel. Both the low-pass filter and median filter are good at noise suppression by smoothing. However, median filter outperforms low-pass filter in one aspect. Median filter preserves edges or discontinuities in step function and is capable of smoothing pixels intensities which differ significantly from their surroundings without influences on other pixels. Low-pass filter smooths any discontinuities on image data.

I have taken a set of typical noisy tactile sensor array data to study the effects of these two filters. This tactile image was obtained when a sensor array contacted at the corner of a planar object, like a box, as shown in figure-3.3.

Figure-3.4 depicts the effects when individual filter is applied on the sensor array data. The result of median filtering shows a flatter plateau at

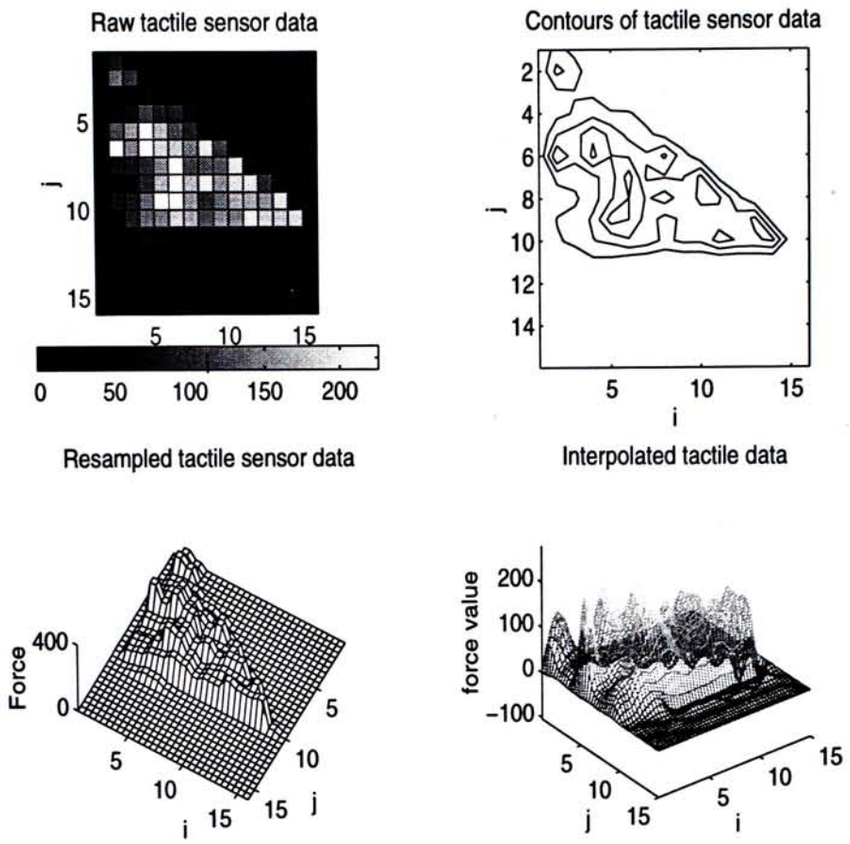


Figure 3.3: Raw tactile sensor data.

top of the image which matches with the real situation. Figure-3.5 depicts the effect of wiener filter and then median filter while figure-3.6 depicts the effect of median filter and then wiener filter. By comparing these two figures, the effect of Wiener filter→Median filter is better as it gives a rather flat top of the tactile image.

3.5 Curvature Determination

After fitting and filtering the tactile sensor data, geometric information about the object that the tactile sensor touches can be derived from the preprocessed tactile sensor data. It is because the deformed rubber pad of the tactile sensor fully touches on the object surface. The shape of the deformed rubber pad surface can thus reflect the actual shape of the

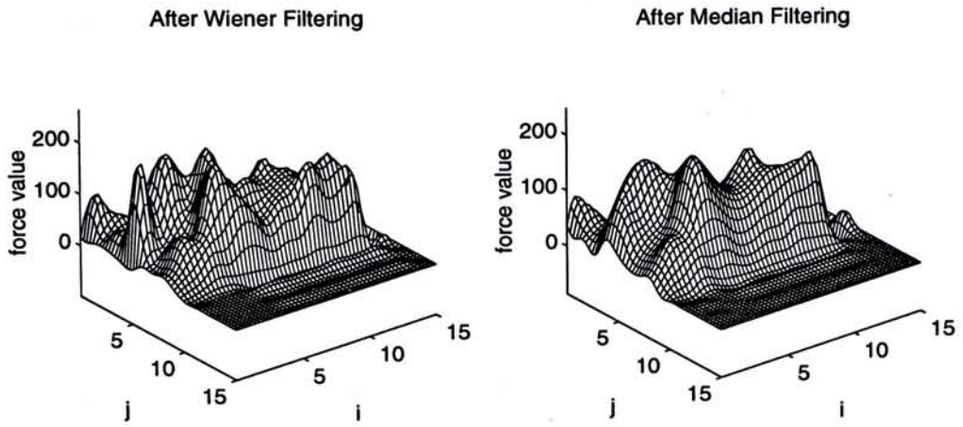


Figure 3.4: Adaptive Wiener Filter and Median Filter results.

object surface in a small local region within some error bound. Therefore, we can estimate geometric properties at local the object surface from our preprocessed sensor data. The geometric property that we need for our reconstruction algorithm is the surface curvature at the direction that is orthogonal to the contour tracing direction.

As a B-spline surface is fitted onto the tactile sensor data, the parametric form of the object surface at local region is obtained. Curvature at any surface point (at a particular direction) of a parametric surface can be explicitly determined based on the theory of fundamental differential geometry [31]. The details is described as follows.

Given a surface patch which is denoted in a parametric form as,

$$\vec{X}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} \qquad where \qquad [u,v] \rightarrow [0,1] \times [0,1]$$

there are two basic vectors for the surface patch defined as the two partial derivatives with respect to u and v of the surface, ie. $\frac{\partial \vec{X}}{\partial u}$ and $\frac{\partial \vec{X}}{\partial v}$ respectively. Actually, $\frac{\partial \vec{X}}{\partial u}$ and $\frac{\partial \vec{X}}{\partial v}$ are the tangent vectors of the surface curves $v = const$ and $u = const$ respectively. For simplicity, they are denoted as $\vec{X}_u$ and $\vec{X}_v$ respectively. They also span the tangent plane of the surface.

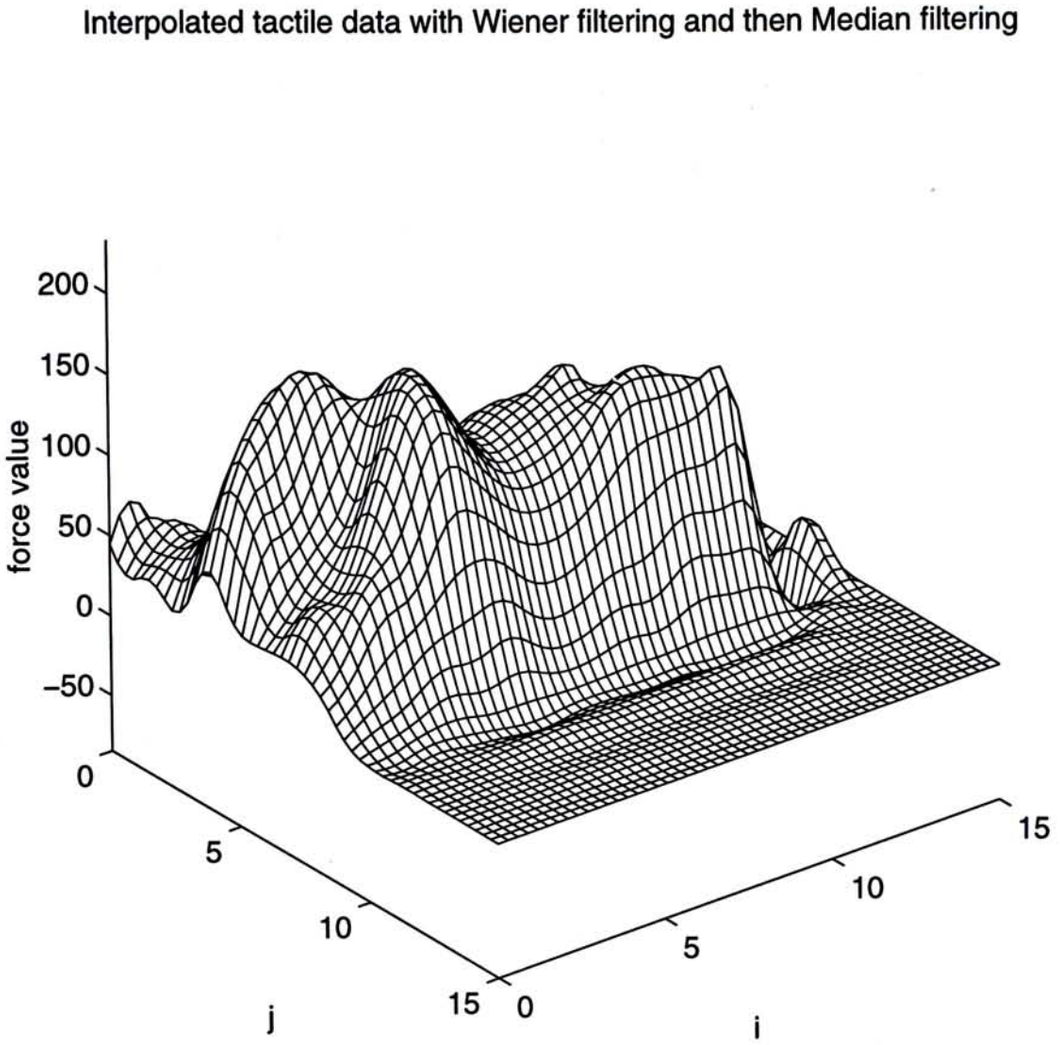


Figure 3.5: Result of Wiener filter→Median filter.

A vector of the surface $d\vec{X}$ that connects two surface points from $\vec{X}(u,v)$ and $\vec{X}(u+du, v+dv)$ can be expressed as,

$$d\vec{X} = \vec{X}_u du + \vec{X}_v dv$$

By taking the square of the magnitude of the surface $\vec{X}$, we have the fundamental magnitudes of the first order of the surface,

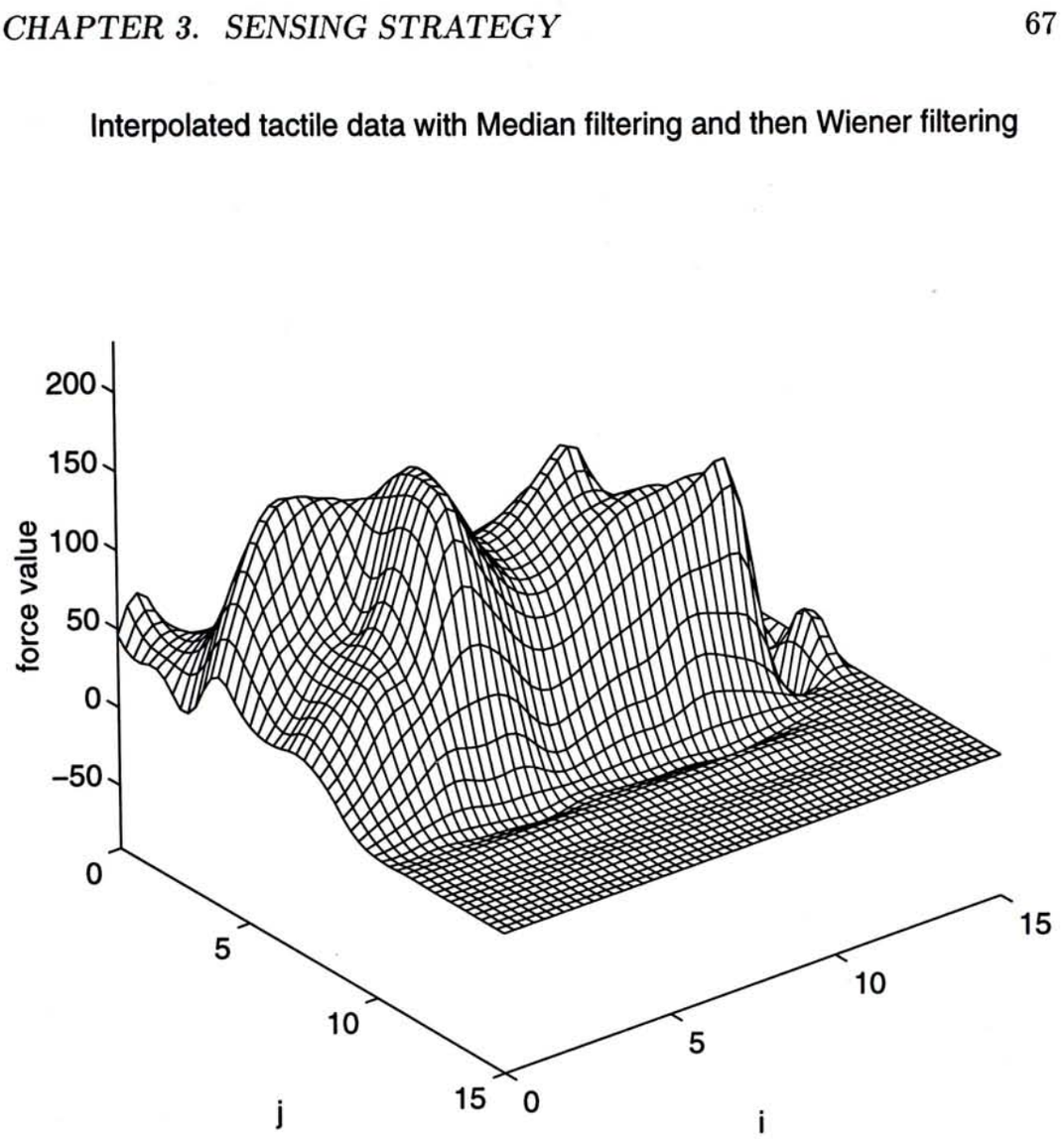


Figure 3.6: Result of Median filter→Wiener filter.

$$\begin{aligned} \left|d\vec{X}\right|^2 &= d\vec{X} \cdot d\vec{X} \\ &= \vec{X}_u \cdot \vec{X}_u (du)^2 + 2\vec{X}_u \cdot \vec{X}_v du dv + \vec{X}_v \cdot \vec{X}_v (dv)^2 \\ &= E(du)^2 + 2F du dv + G(dv)^2 \end{aligned} \tag{3.10}$$

where $E = \vec{X}_u \cdot \vec{X}_u$, $F = \vec{X}_u \cdot \vec{X}_v$ and $G = \vec{X}_v \cdot \vec{X}_v$. The cross product of the two basic vectors defines the normal vector of the surface point and its magnitude is $H = \sqrt{EG - F^2}$. Therefore, the unit normal vector of a

surface point is

$$\hat{n} = \frac{\vec{X}_u \times \vec{X}_v}{H} \tag{3.11}$$

At any regular surface point (which has non-zero normal vector), $\vec{p}$, a pencil of tangent vectors (including the two basic vectors) at $\vec{p}$ spans out a tangent plane. From (3.5), one of the unit tangent vector, $\hat{t}$ at $\vec{p}$ can be expressed as,

$$\vec{t} = \frac{d\vec{X}}{ds} = \vec{X}_u \frac{du}{ds} + \vec{X}_v \frac{dv}{ds} \tag{3.12}$$

where ds is the arc length element of the curve on the surface $\vec{X}(u, v)$ which is determined by the intersection of the normal plane and the surface. The normal plane is defined by both $\hat{t}$ and $\hat{n}$. The surface curve is actually a plane curve lying on the normal plane and it is called a normal section. The curvature, κ, at $\vec{p}$ of the normal curvature at $\vec{p}$. There are infinitely many normal curvature at a regular surface point because we have a pencil of tangent vectors at a surface point which span out a tangent plane. In this project, we are interested on the normal curvature at the direction orthogonal to the contours traced. Before the introduction of the derivation of normal curvature of a surface point, the fundamental magnitudes of the second order of the surface is first derived by differentiating (3.12) with respect to the arc s, as follows.

$$\frac{d\hat{t}}{ds} = \vec{X}_u \frac{d^2u}{ds^2} + \vec{X}_v \frac{d^2v}{ds^2} + \vec{X}_{uu} (\frac{du}{ds})^2 + 2\,\vec{X}_{uv} (\frac{du}{ds})(\frac{dv}{ds}) + \vec{X}_{vv} (\frac{dv}{ds})^2 \tag{3.13}$$

From the Serret-Frenet's formulas of space curves[1][31], (3.13) actually equals

[1]Serret-Frenet's formulas relate the unit tangent $\hat{t}$, normal $\hat{n}$ and binormal $\hat{b}$ vectors of a space curve to their derivatives (with respect to arc length s) with the introduction of curvature κ and torsion τ of the curve, which is expressed as,

$$\frac{d\hat{t}}{ds} = \kappa\,\hat{n}, \qquad \frac{d\hat{n}}{ds} = -\kappa\,\hat{t} + \tau\,\hat{b}, \qquad \frac{d\hat{b}}{ds} = -\tau\,\hat{n}$$

to $\kappa\,\hat{n}$. The first and second terms of (3.13) can be eliminated by forming dot product of the unit normal vector $\hat{n}$ with the both sides of(3.13) because they lie on the tangent plane. Define $L = \hat{n} \cdot \vec{X}_{uu}$, $M = \hat{n} \cdot \vec{X}_{uv}$ and $N = \hat{n} \cdot \vec{X}_{vv}$, we have,

$$\kappa = L\,(\frac{du}{ds})^2 + 2M\,(\frac{du}{ds})(\frac{dv}{ds}) + N(\frac{dv}{ds})^2 \tag{3.14}$$

This is the fundamental magnitudes of the second order of the surface. With the two fundamental magnitudes (3.10)[2] and (3.14), we have,

$$\kappa = \frac{L + 2Mh + Nh^2}{E + 2Fh + Gh^2} \qquad \text{where} \qquad h = \frac{dv}{du} \tag{3.15}$$

The direction of the normal curvature (or the normal section) depends on the ratio $h = \frac{dv}{du}$.

The normal curvature, that is interested, is orthogonal to the contours tracing direction. The contours are usually traced along a plane that is parallel to the testbed. We can arbitrarily embed a local parameter range $[u, v] \rightarrow [0, 1] \times [0, 1]$ on the tactile sensor data surface with the $u-$ and $v-$ directions parallel to the contours tracing direction and its perpendicular. Then, the direction of the normal curvature needed can be defined as $du = 0$, or $h \rightarrow \infty$. The curvature can be simplified as,

$$\kappa = \lim_{h \to \infty} \frac{L + 2Mh + Nh^2}{E + 2Fh + Gh^2} = \frac{N}{G} \tag{3.16}$$

If $L : M : N = E : F : G$, the normal curvature κ is independent of h and the surface point with this property is called umbilical point [30][29][31].

As κ changes with h, we can find its two extreme values, denoted as κ_1 and κ_2 (which are called the principal curvatures), at the directions h_1 and h_2 (which are called the principal directions) respectively. The two

[2] $|d\vec{X}|$ in the equation is actually the arc length of the required normal section, ie. $|d\vec{X}| = ds$

directions satisfy the following equation which has real roots only

$$\begin{vmatrix} h^2 & -h & 1 \\ E & F & G \\ L & M & N \end{vmatrix} = 0 \tag{3.17}$$

or

$$(FN - MG)h^2 + (EN - LG)h + (EM - LF) = 0 \tag{3.18}$$

On the other hand, the two principal curvatures are the roots of the equation,

$$\begin{vmatrix} \kappa E - L & \kappa F - M \\ \kappa F - M & \kappa G - N \end{vmatrix} = 0 \tag{3.19}$$

or

$$(EG - F^2)\kappa^2 - (EN + LG - 2MF)\kappa + (LN - M^2) = 0 \tag{3.20}$$

Two important curvature information of a surface point can be derived from (3.20). The first one is the Gaussian Curvature κ_G which is defined as product of the principal curvatures [30][29][31],

$$\kappa_G = \kappa_1\kappa_2 = \frac{LN - M^2}{EG - F^2} \tag{3.21}$$

The other one is the Mean Curvature κ_M which is defined as the mean of the principal curvatures [30][29][31],

$$\kappa_M = \frac{\kappa_1 + \kappa_2}{2} = \frac{EN + LG - 2MF}{EG - F^2} \tag{3.22}$$

If κ_1 and κ_2 have the same sign or $\kappa_G > 0$, the surface point is called elliptic. If they have different sign or $\kappa_G < 0$, the surface point is called hyperbolic. If one of the principal curvatures is zero or $\kappa_G = 0$, the surface point is called parabolic.

As mentioned above, the tactile sensor data is approximated by

fitting a k-th order B-spline surface $\vec{TS}(u,v)$ on the it as follows,

$$\vec{TS}(u,v) = \begin{bmatrix} TS_x(u,v) \\ TS_y(u,v) \\ TS_z(u,v) \end{bmatrix} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N_{i,k}(u) N_{j,k}(v) \tag{3.23}$$

where $\vec{P}_{ij}^{TS}$ is the $n_1 \times n_2$ control net defining the tactile sensor data and the original tactile sensor data is given with respect to the working coordinate frame of the finger at which the tactile sensor is mounted on (usually at the finger tip). From the above normal curvature derivation[3], the partial derivatives of $\vec{TS}(u,v)$ with respect to its independent variables (ie. u and v) and their mixed derivatives are needed. They are,

$$\begin{aligned}
\vec{TS}_u(u,v) &= \frac{\partial \vec{TS}}{\partial u} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N'_{i,k}(u) N_{j,k}(v) \\
\vec{TS}_v(u,v) &= \frac{\partial \vec{TS}}{\partial v} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N_{i,k}(u) N'_{j,k}(v) \\
\vec{TS}_{uu}(u,v) &= \frac{\partial^2 \vec{TS}}{\partial u^2} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N''_{i,k}(u) N_{j,k}(v) \\
\vec{TS}_{vv}(u,v) &= \frac{\partial^2 \vec{TS}}{\partial v^2} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N_{i,k}(u) N''_{j,k}(v) \\
\vec{TS}_{uv}(u,v) &= \frac{\partial^2 \vec{TS}}{\partial u\, \partial v} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \vec{P}_{ij}^{TS} N'_{i,k}(u) N'_{j,k}(v)
\end{aligned} \tag{3.24}$$

where $(\cdot)'$ is the derivative of $(\cdot)$ with respect to its independent variable. The problem is transformed to the calculation of the derivatives of B-spline basis functions. In general, the derivatives of B-spline basis functions can be computed from two different approaches. For the lth B-spline basis derivative, $N_{i,k}^{(l)}(u)$, the first approach is to calculate it by linear combination

[3]The calculation of $\hat{n}$, E, F, G, H, L, M and N in (3.15)

of basis functions with order $(k-l)$, as follows[37],

$$N_{i,k}^{(l)}(u) = \frac{(k-1)!}{(k-l-1)!} \sum_{j=1}^{l+1} c_{l+1,j} N_{i+j-1,k-l}(u) \tag{3.25}$$

where

$$\begin{cases} c_{1,1} &= 1 \\ c_{p,1} &= \frac{c_{p-1,1}}{x_{i+k-p+2} - x_{i+1}} \\ c_{p,j} &= \frac{c_{p-1,j} - c_{p-1,j-1}}{x_{i+k+j-p+2} - x_{i+j+1}} \qquad j = 2, \ldots, p-1 \\ c_{p,p} &= \frac{-c_{p-1,p-1}}{x_{i+k} - x_{i+p-1}} \end{cases}$$

with $[x_1, x_2, \ldots, x_m]$ is the knot vector. This actually comes out from the following expression,

$$N_{i,k}^{(l)}(u) = p \left(\frac{N_{i,k-1}^{(l-1)}(u)}{x_{i+k} - x_i} - \frac{N_{i+1,k-1}^{(l-1)}(u)}{x_{i+k+1} - x_{i+1}} \right) \tag{3.26}$$

It should be pointed out that $l <= k-1$ and the quotient is defined to be zero when its denominator involving difference of knots is zero. From (3.25), we have, for instances,

$$N'_{i,k}(u) = \frac{p \cdot N_{i,k-1}(u)}{x_{i+k-1} - x_i} - \frac{p \cdot N_{i+1,k-1}(u)}{x_{i+k} - x_{i+1}}$$

One the other hand, the second approach defines the derivatives of B-spline basis function in terms of the same order of derivatives of B-spline basis with previous order[37],

$$N_{i,k}^{(l)}(u) = \frac{k-1}{k-l-1} \left(\frac{u - x_i}{x_{i+k-1} - x_i} N_{i,k-1}^{(l)}(u) + \frac{x_{i+k} - u}{x_{i+k} - x_{i+1}} N_{i+1,k-1}^{(l)} \right) \tag{3.27}$$

At any surface point $\vec{X}(u,v)$, we can first calculate the derivatives of the surface including, $\vec{TS}_u$, $\vec{TS}_v$, $\vec{TS}_{uu}$, $\vec{TS}_{uv}$ and $\vec{TS}_{vv}$. Then, E, F and G in the fundamental magnitudes of first order of the surface in (3.10) can be

computed and so do the unit normal vector at that surface point by forming the cross product of the two basic vectors and computed H. In addition, L, M and N in the fundamental magnitudes of second order of the surface in (3.14) can be formed. With all these values in hand, the normal curvature of a particular direction at that surface point can be computed according to (3.15). Usually, the curvature at the center point of the tactile sensor data surface is needed, ie. $\vec{TS}(0.5, 0.5)$.

3.6 Step Size Determination

We can specify a step size for between neighbouring contours in terms of fraction of total height of the object, denoted as $s \cdot h$, where s is scalar in the range of $0 < s < 0.5$ and h is the total height of the object. The scalar s depends on the average curvature estimated κ_{avg} and change in curvature estimated $\triangle\kappa_{avg}$ and it is denoted as $s(\kappa_{avg}, \triangle\kappa_{avg})$. This called the step size function. For clarity, we omit the subscript notation for the average characterization of them and notations κ and $\triangle\kappa$ carry an average character in the following description unless other meanings are stated. It is limited to 0.5 because we begin exploration from the mid-height of the object and then trace contours in the upward or downward directions. There are some heuristics for the selection of template functions to model the step size behaviours in terms of the curvature κ and change in curvature measured $\triangle\kappa$.

1. $s(\kappa, \triangle\kappa)$ is a strictly decreasing function with respect to both κ and $\triangle\kappa$. This means that

$$\frac{\partial s(\kappa, \triangle\kappa)}{\partial \kappa} < 0 \qquad \frac{\partial s(\kappa, \triangle\kappa)}{\partial \triangle\kappa} < 0$$

2. For a constant κ, $s(\kappa, \triangle\kappa)$ will decrease with increasing $\triangle\kappa$, or $\frac{\partial s(\kappa, \triangle\kappa)}{\partial \triangle\kappa} < 0$.

3. For a constant $\triangle\kappa$, $s(\kappa, \triangle\kappa)$ will decrease with increasing κ, or $\frac{\partial s(\kappa, \triangle\kappa)}{\partial \kappa} < 0$.

4. The effect of 2 is greater than the effect of 3.

5. $s(\kappa, \triangle\kappa)$ depends on the number of step size determinations or number of contours traced on the upper of lower half of the object. The more the number of contours traced, the smaller the step size determined.

For simplicity, the step size function is separated into two components, one depends on curvature κ alone and the other one depends on change in curvature $\triangle\kappa$ alone.

$$s(\kappa, \triangle\kappa) = \lambda \cdot s_1(\kappa) \cdot s_2(\triangle\kappa) + \mu \tag{3.28}$$

Examples that satisfy the above heuristics are the exponential functions, polynomial inverses and logarithmic functions as listed below,

$$\begin{aligned} s_i(\alpha) &= \mathbf{e}^{-a\alpha} \\ s_i(\alpha) &= \frac{a}{\ln(b\alpha + c)} \\ s_i(\alpha) &= \frac{1}{(a\alpha + b)^c} \end{aligned}$$

where a, b, c are constant scalar, α represents κ or $\triangle\kappa$ and $i \in [1, 2]$.

For rapid evaluation of the step size function, it can be approximated by piecewise linear function [50]. It can first be separated into two components as in (3.28) with the first component depends on curvature κ only while the second component depends on change in curvature $\triangle\kappa$. They are called the first and second step size function components respectively. The two components are approximated by piecewise linear

functions independently. The approximation can be divided into two stages. The first stage is to approximate the decreasing rate or shape of the function. A function template is defined for each component which is bounded in the interval $[0, 1]$. Different function template have different characteristics, like decreasing rates, rate of change of decreasing rates and convergence rates. For instance, an exponential function is used to construct one of the step size function component. The function template can be defined as $\bar{s}_t = \mathbf{e}^{-a\alpha}$, where $\alpha = \kappa$ or $\triangle\kappa$. The second stage is to rescale the product of the two step size function components into the the range $[s_{min}, s_{max}]$.

For each step size component s_α, N test pairs are generated, $[\alpha_i, y_i]$, $\forall i \in \{1, \ldots, N\}$. Assume the step size component is approximated by a n-segmented linear spline, a knot sequence is defined as

$$\mathbf{k} = [k_0, k_1, k_2, \ldots, k_{n-1}, k_n]$$

A piecewise continuous linear function is defined as,

$$\bar{s}(\alpha) = \sum_{i=0}^{n} c_i \, |\alpha - k_i| \tag{3.29}$$

In order to simplify the problem, we set $k_0 = \alpha_1$ and $k_n = \alpha_N$ in the knot sequence.

In this approximation problem, there are total of $2n$ unknowns should be solved in the approximation problem. They include $(n + 1)$ unknowns in coefficient sequence $[c_0, c_1, \ldots, c_n]$ and $(n - 1)$ unknowns in knot sequence. A system of nonlinear equations can be set up,

$$
\begin{aligned}
&c_0|\alpha_1-\alpha_1| + c_1|\alpha_1-k_1| +\ldots\ldots+ c_{n-1}|\alpha_1-k_{n-1}| + c_n|\alpha_1-\alpha_N| = y_1\\
&c_0|\alpha_2-\alpha_1| + c_1|\alpha_2-k_1| +\ldots\ldots+ c_{n-1}|\alpha_2-k_{n-1}| + c_n|\alpha_2-\alpha_N| = y_2\\
&\quad\ldots\ldots \quad + \quad \ldots\ldots \quad +\ldots\ldots+ \quad \ldots\ldots \quad + \quad \ldots\ldots \quad = \ldots\\
&c_0|\alpha_{N-1}-\alpha_1|+c_1|\alpha_{N-1}-k_1|+\ldots\ldots+c_{n-1}|\alpha_{N-1}-k_{n-1}|+c_n|\alpha_{N-1}-\alpha_N|=y_{N-1}\\
&c_0|\alpha_N-\alpha_1| + c_1|\alpha_N-k_1| +\ldots\ldots+ c_{n-1}|\alpha_N-k_{n-1}| + c_n|\alpha_N-\alpha_N| = y_N
\end{aligned}
\tag{3.30}
$$

or in matrix form

$$
\begin{bmatrix}
0 & |\alpha_1-k_1| & \ldots & |\alpha_1-k_{n-1}| & |\alpha_1-\alpha_N|\\
|\alpha_2-\alpha_1| & |\alpha_2-k_1| & \ldots & |\alpha_2-k_{n-1}| & |\alpha_2-\alpha_N|\\
\vdots & \vdots & \ddots & \vdots & \vdots\\
|\alpha_{N-1}-\alpha_1| & |\alpha_{N-1}-k_1| & \ldots & |\alpha_{N-1}-k_{n-1}| & |\alpha_{N-1}-\alpha_N|\\
|\alpha_N-\alpha_1| & |\alpha_N-k_1| & \ldots & |\alpha_N-k_{n-1}| & 0
\end{bmatrix}
\begin{bmatrix} c_0\\ c_1\\ \vdots\\ c_{n-1}\\ c_n \end{bmatrix}
=
\begin{bmatrix} y_1\\ y_2\\ \vdots\\ y_{N-1}\\ y_N \end{bmatrix}
\tag{3.31}
$$

or

$$
\mathbf{A}\vec{c} = \vec{y} \tag{3.32}
$$

where $\mathbf{A}$ is a $N \times (n+1)$ matrix.

Firstly, the knot sequence $\mathbf{k}$ is should be solved in equation (3.32). In MATLAB, the function *fsolve* in the Optimization toolbox can be employed to solve system of nonlinear equations. Nonlinear equations are solved by minimizing the squared error of $\vec{e} = \mathbf{A}\vec{c} - \vec{y}$. After optimal $\mathbf{k}$, denoted by $\hat{\mathbf{k}}$, is solved, the coefficient sequence can then be evaluated by

$$
\vec{c} = \left(\mathbf{A}(\hat{\mathbf{k}})^T\mathbf{A}(\hat{\mathbf{k}})\right)^{-1}\mathbf{A}(\hat{\mathbf{k}})^T\vec{y} \tag{3.33}
$$

After both the shape of the step size function components $s_1(\kappa)$ and $s_2(\triangle\kappa)$ are found, the resultant step size function can be mapped into the

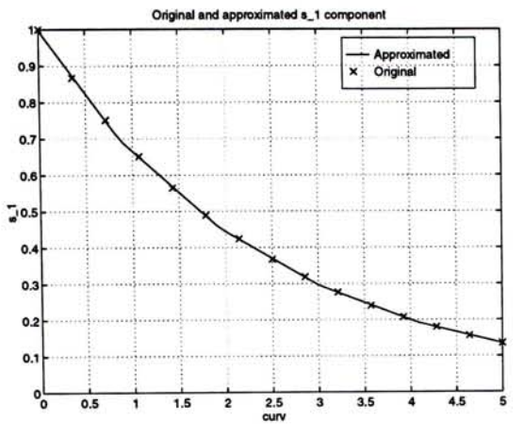


Figure 3.7: Original and approximated s_1 step size component

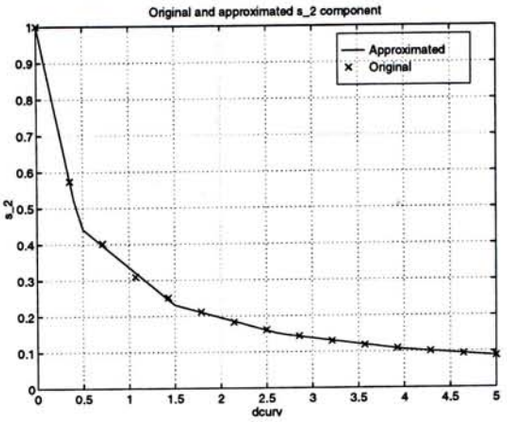


Figure 3.8: Original and approximated s_2 step size component

range $[s_{min}, s_{max}]$ by adjusting the two parameters λ and μ. s is maximum when both s_1 and s_2 equal to 1 and s is minimum when one of the s_1 and s_2 equals to zero.

$$s_{max} = \lambda + \mu \quad \Rightarrow \quad \lambda = s_{max} - s_{min} \tag{3.34}$$

$$s_{min} = \mu \quad \mu = s_{min} \tag{3.35}$$

Figures- 3.7 and 3.8 depict the originl and approximated of the two step size function components. They are approximated by five segmented linear splines. In addition, Figures 3.9 and 3.10 show the original and approximated step size function.

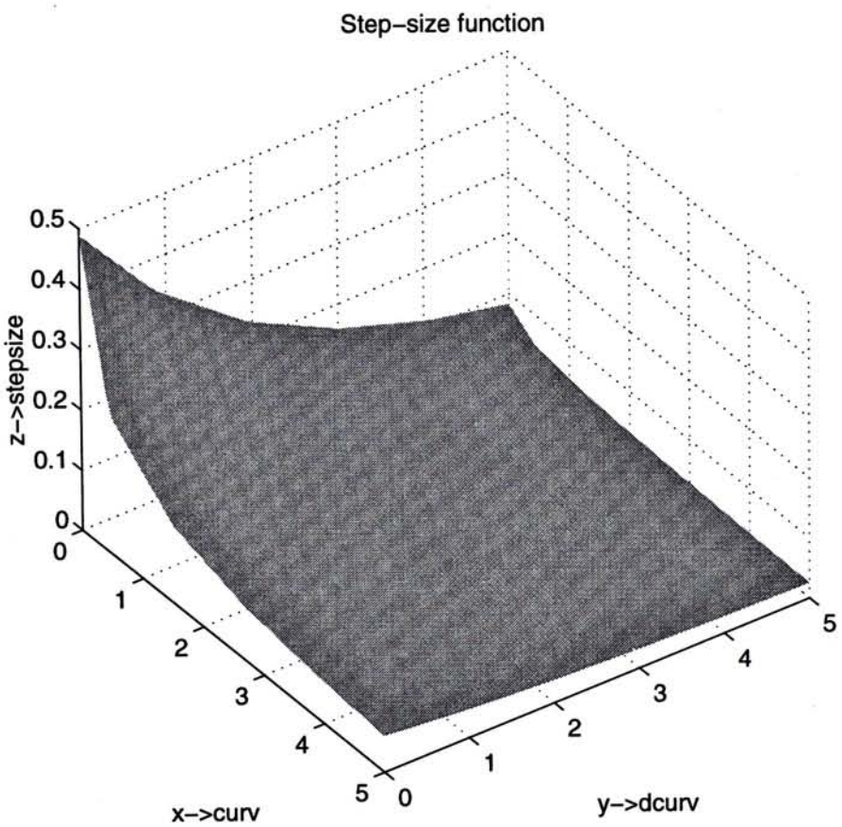


Figure 3.9: Approximated step size function $s(\kappa, \triangle\kappa)$

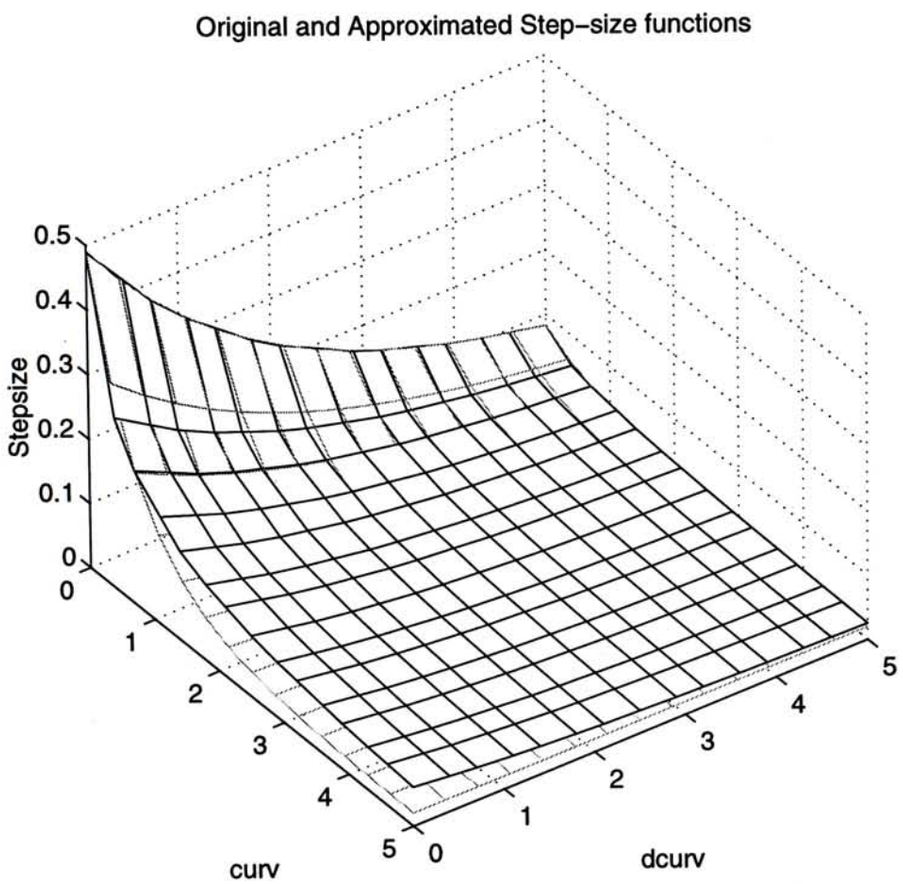


Figure 3.10: Original and approximated step size function

Chapter 4

3D Shape Reconstruction

4.1 Introduction

After acquisition of required 3D points coordinates from contour tracing, a 3D model can be reconstructed from these data points. This chapter mainly deals with the model reconstruction process. The geometric model proposed in this dissertation is superquadric model with free form deformation for shape fine tuning (see Chapter 2). The model reconstruction problem is transformed to the problem of finding a deformed control lattice that can construct an object model which the sensed data points satisfy it with given original object model before FFD (superquadric model) and original control lattice. This is actually an inverse process of FFD. In this inverse process, we solve the correspondence problem between points on the original model and the deformed model (the model that we have known surface points coordinates). After the correspondence problem of all data points are solved, this leads to the point inversion problem for all data points. This means that each data point is assigned a (s, t, u) parameter triple. Original model point and its corresponding data point shares the same (s, t, u) parameter triple. Finally, the recovery of deformed control lattice or the model reconstruction process begins. With only the recovered control

lattice, the geometric model of the object cannot be completely represented. Up to this stage, only object points which have same (s,t,u) parameter triples with the data points can be obtained. Object points on other surface patches region, however, cannot be restored. Thus, an interpolation algorithm is also developed to generate (s,t,u) parameter values for the object surface automatically. In this chapter, the correspondence problem, the point inversion problem and the parameter interpolation algorithm are discussed. After then, the model recovery process is discussed.

4.2 Correspondence Problem

In the correspondence problem, we have to find a correspondence or linkage between each data point on the required (deformed) object model and a surface point on the original object model. A new technique is developed to solve this problem. The basic concept of the algorithm defines matches between sensed object points and original model points in two stages, namely, the Contour Match Correspondence stage and the Curve Point Match Correspondence stage. Details of the algorithm is presented as follows,

Assume that there are M_1 contours traced by the haptic exploration system and each contour consists of M_2 object surface points,

1. Align the contour search direction parallel to the z-axis of the recovered model space [1].

2. Transform all data points into a normalized domain $\mathcal{N}^3 \stackrel{def}{=} [-1,1] \times [-1,1] \times [-1,1]$. The reason for this step is due to the affine invariance property of B-splines (see section 4.2.1). With this property, the solution of the point inversion problem can be facilitated.

[1] The coordinate frame of this recovered model space becomes the base coordinate frame for other derived spaces during the model reconstruction process. These derived spaces will be defined with respect to this base frame.

3. Construct an original superquadric model according to the shape parameters ϵ_1 and ϵ_2 obtained in the modified Grasp by Containment EP described in section 3.2.2 and an original control lattice which has a base point of $[-1, -1, -1]$ with size of $2 \times 2 \times 2$. This original model can be embedded in the normalized domain $\mathcal{N}^3$.

4. Contour Match Correspondence

   (a) Slice the original model uniformly with M_1 slicing planes parallel to the x-y plane, where M_1 is the number of contours traced. This is equivalent to partition $(M_1 + 1)$ subintervals in the interval $[-1, 1]$ along the z-axis. At each slice, a cross-section of the original model can be obtained.

   (b) Match the boundary curve of each cross-section of the original model with each contour traced in the spatial order along the contour search direction, as shown in Figure 4.1.

5. Curve Point Match Correspondence

   For each matched contour traced by the haptic exploration system,

   (a) Find the centroid of the close contour, $\vec{O}_i = \frac{1}{M_2} \sum_{j=1}^{M_2} \vec{x}_{ij}$ where $\vec{x}_{ij}$ is the j-th point in the i-th contour.

   (b) Fit a closed curve on the traced contour. This actually is a crucial step in model recovery process and it will be discussed in details in section 4.4.2.

   (c) Uniformly sample M_2 points along the fitted contour curve by intersecting M_2 radial lines from the contour centroid O_i to the contour curve.

   (d) Repeat step (5c) for the corresponding cross-section boundary curve in the original model.

(e) Match the object data points and model points with same azimuth angle θ, as shown in Figure 4.2.

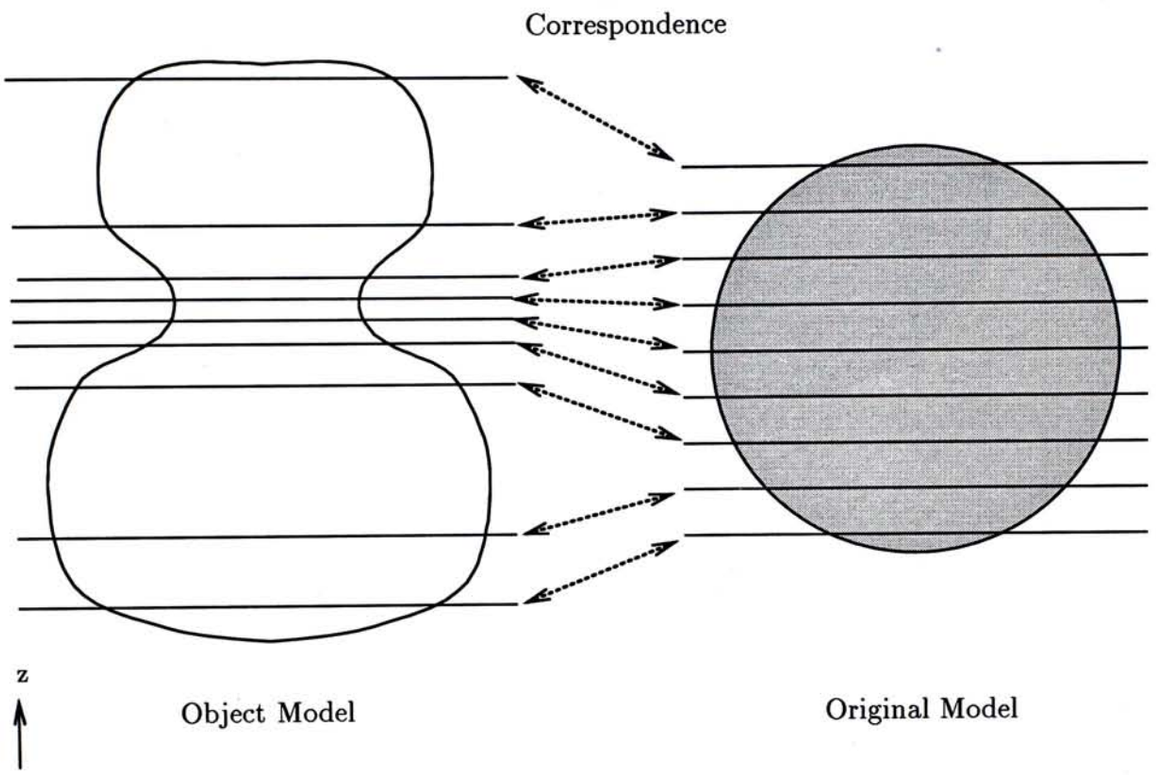


Figure 4.1: Contour Match Correspondence

After all correspondences are assigned for all original model points and data points pairs, the point inversion problems for all data point are then solved. However, it is difficult to find the (s, t, u) of each data point with respect to the deformed control lattice because the deformed control lattice is what we are required to determine. It is an unknown information. Fortunately, as each pair of original model point and data point shares the same (s, t, u) parameter triple, the point inversion problem for data points can be transformed to the point inversion of their original model counterparts. In the original model side, the control lattice is construct in steps of finding correspondences between data points and original points

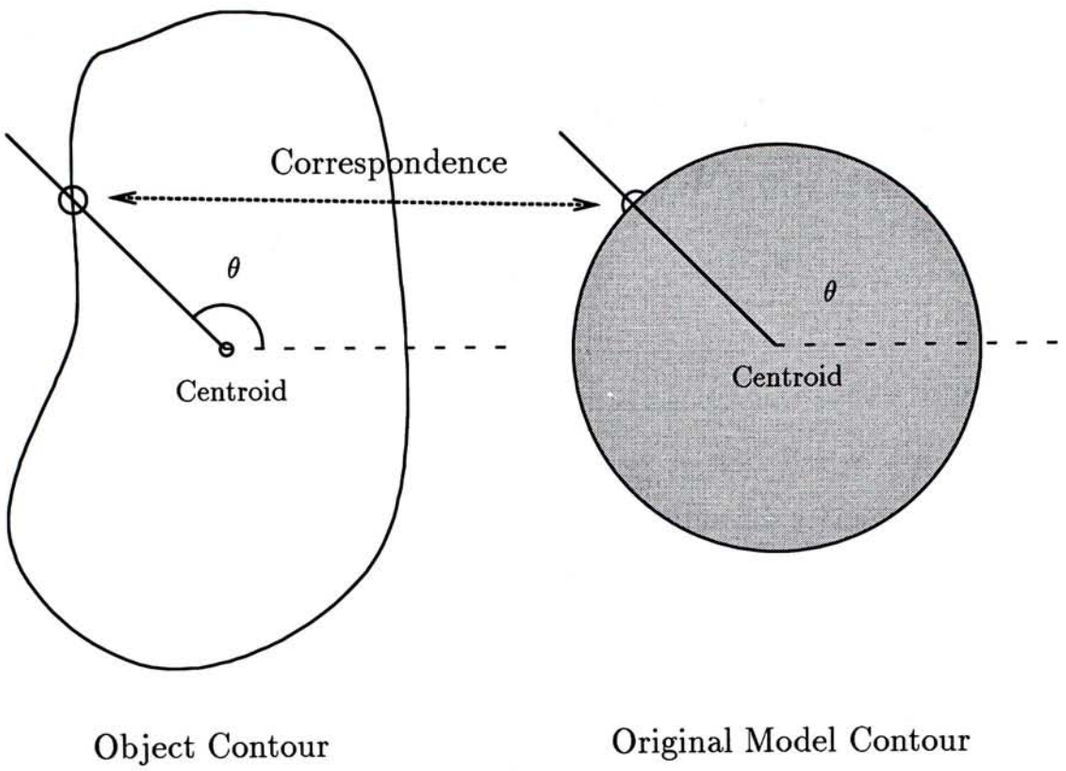


Figure 4.2: Curve Point Match Correspondence

and the point inversion can be solved with an algorithm presented in section 4.2.2. In solving the point inversion problem, an important property of B-splines eases the problem. This is the affine invariance property. With this property, we can solve the point inversion problem for control lattices with different size and base point by transforming the control lattice to a normalized domain where a set of (s, t, u) parameter triple for 3D points embedded in it.

4.2.1 Affine Invariance Property of B-splines

One of the properties of B-spline curves or surfaces is affine invariance. This property can be extended to trivariate B-spline solids of FFD. This property is useful in solving the point inversion problem which should be solved frequently in the process of FFD. The next section will

discuss the Point inversion problem.

Theorem 1 *The (s,t,u) parameter triple are the same on the initial control lattices $i\vec{P}$ and the deformed control lattice $i\bar{P}$ if $i\bar{P} = \Phi(i\vec{P})$, where Φ is an affine transformation.*

Proof:

The initial control lattice $i\vec{P}$, which is partitioned as $(l+1) \times (m+1) \times (n+1)$ sublattices, has a base point at $\vec{p}_0$ and size of $s\vec{i}z$ and the three local unit coordinate axes of the lattice frame are $\hat{s}, \hat{t}$ and $\hat{u}$, or $i\vec{P} \stackrel{def}{=} \{\vec{p}_0, s\vec{i}z, \hat{s}, \hat{t}, \hat{u}\}$. Similarly, we have $i\bar{P} \stackrel{def}{=} \{\bar{p}_0, s\bar{i}z, \bar{s}, \bar{t}, \bar{u}\}$. In general, an affine transformation Φ includes translation, rotations, scaling and shearing. It can be defined as $\Phi : \mathcal{E}^3 \mapsto \mathcal{E}^3$, where $\mathcal{E}^3$ is the three dimensional Euclidean space and in terms of a transformation matrix $\mathbf{R}$ and translation vector $\vec{v}$,

$$\Phi(\vec{r}) = \mathbf{R}\vec{r} + \vec{v} \tag{4.1}$$

For a trivariate B-spline solid,

$$\vec{x}_{BFFD}(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u) \tag{4.2}$$

For clarity, (4.2) can be rewritten to a form with only single index as follow,

$$\vec{x}_{BFFD}(s,t,u) = \sum_{I=1}^{M} \vec{B}_I N_I(s,t,u) = \sum_{I=1}^{N} \beta_I \vec{B}_I \tag{4.3}$$

where $I = (k-1)(l+1)(m+1) + (j-1)(l+1) + i$, $\beta_I = N_I(s,t,u) = N_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u)$ and $M = (l+1)(m+1)(n+1)$. Details of control vertices indexing can be referred to Table 4.1. The partition of unity property of B-spline basis [37] (trivariate extension from cases of curves

k	j	i	I
1	1	1	1
1	1	2	2
⋮	⋮	⋮	⋮
1	1	$l+1$	$l+1$
1	2	1	$l+2$
⋮	⋮	⋮	⋮
2	1	1	$(l+1)(m+1)+1$
⋮	⋮	⋮	⋮
k	j	i	$(k-1)(l+1)(m+1)+(j-1)(l+1)+i$
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
$(n+1)$	$(m+1)$	$(l+1)$	$(l+1)(m+1)(n+1)$

Table 4.1: Indexing for the control vertices

and surfaces) is described as follows, $\forall(s,t,u) \in [0,1]\times[0,1]\times[0,1]$,

$$\sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1} N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u) = 1 \qquad \text{or} \qquad \sum_{I=1}^{M}\beta_I = 1 \tag{4.4}$$

Apply an affine transformation Φ to the B-spline solid,

$$\begin{aligned}
\Phi(\vec{x}_{BFFD}) &= \Phi(\sum_{I=1}^{M}\beta_I\vec{B}_I) \\
&= \mathbf{R}\left(\sum_{I=1}^{M}\beta_I\vec{B}_I\right)+\vec{v} \\
&= \sum_{I=1}^{M}\beta_I\mathbf{R}\vec{B}_I+\sum_{I=1}^{M}\beta_I\vec{v} && \text{[from(4.4)]} \\
&= \sum_{I=1}^{M}\beta_I\left(\mathbf{R}\vec{B}_I+\vec{v}\right) \\
&= \sum_{I=1}^{M}\beta_I\Phi(\vec{B}_I) \\
&= \sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1}\Phi(\vec{B}_{ijk})N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u)
\end{aligned}$$

From the above result, it can be concluded that the parameter space (s,t,u) is invariant under affine transformation. In FFD, the deformed control lattice is usually obtained by translation (which is an affine transformation) of particular control vertices of the original control lattice. This property can speed up the solution of point inversion problem (see sections 4.2.2 and 5.3). The initial control lattice and the object inside it are scaled and translated into a normalized initial control lattice. For the normalized control lattice, the (s,t,u) parameter spaces of a set of orignial shapes (superquadric models with different ϵ_1 and ϵ_2) are pre-computed for generation of initial guesses of solution of the point inversion problem. This initial guess is very efficient.

4.2.2 Point Inversion Problem

Point inversion [37] is a fundamental problem in computer aided geometric design (CAGD). Many complex problems in CAGD involve this problem as a crucial step, like surfaces intersection, finding minimum distance between a point and a curve or surface and so on. For a B-spline curve, the problem is usually stated as[2],

> *Given a point* $\vec{p} = [x,y,z]^T$, *which is assumed to lie on a B-spline curve* $\vec{C}(t)$, *the corresponding parameter* $\bar{t}$ *is found such that* $\vec{C}(\bar{t}) = \vec{p}$.

The above problem statement is not limited to B-spline curve. Any bivariate or trivariate parametric surfaces or solids with other spline basis (rational or irrational) can have analogous problem statement. Even parametric hypersolids with dimension larger than 3 can be applied. In this dissertation, the point inversion problem we have to solve is as follows,

[2]For point projection problem, like finding minimum distance between a point and a curve, the problem statement is similar to the one given above, except that $\vec{p}$ does not lie on the curve.

Given a 3D point $\vec{p} = [x, y, z]^T$, which is assumed to lie on a trivariate B-spline solid $\vec{X}(s,t,u)$, the corresponding parameter triple $(\bar{s}, \bar{t}, \bar{u})$ is found such that $\vec{X}(\bar{s}, \bar{t}, \bar{u}) = \vec{p}$.

where

$$\vec{X}(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u) \tag{4.5}$$

Point inversion problem is usually solved by iterative method, like Newton-Rasphon method. The goal is to minimize the distance between $\vec{p}$ and $\vec{X}(s,t,u)$. Piegl and Tiller [37] proposed iterative algorithm for solving point inversion problems in curves and surfaces. In this dissertation, their algorithms can be extended to cover the case of trivariate parametric solids. Define

$$\vec{R}(s,t,u) = \vec{X}(s,t,u) - \vec{p} \tag{4.6}$$

and we have

$$\begin{aligned}
\vec{R}_s &= \frac{\partial \vec{R}}{\partial s} = \vec{X}_s(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N'_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u) \\
\vec{R}_t &= \frac{\partial \vec{R}}{\partial t} = \vec{X}_t(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N'_{j,k_y}(t) N_{k,k_z}(u) \\
\vec{R}_u &= \frac{\partial \vec{R}}{\partial u} = \vec{X}_u(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N_{j,k_y}(t) N'_{k,k_z}(u)
\end{aligned} \tag{4.7}$$

Define

$$\begin{aligned}
f(s,t,u) &= \vec{R}(s,t,u) \cdot \vec{X}_s(s,t,u) = 0 \\
g(s,t,u) &= \vec{R}(s,t,u) \cdot \vec{X}_t(s,t,u) = 0 \\
h(s,t,u) &= \vec{R}(s,t,u) \cdot \vec{X}_u(s,t,u) = 0
\end{aligned} \tag{4.8}$$

Then, equation (4.8) is solved by an iterative method with an intelligent guess of initial values of $(\bar{s}, \bar{t}, \bar{u})$. At the ith iteration, let

$$J_i = \begin{bmatrix} f_s & f_t & f_u \\ g_s & g_t & g_u \\ h_s & h_t & h_u \end{bmatrix} = \begin{bmatrix} \left|\vec{X}_s\right|^2 + \vec{R}\cdot\vec{X}_{ss} & \vec{X}_t\cdot\vec{X}_s + \vec{R}\cdot\vec{X}_{st} & \vec{X}_u\cdot\vec{X}_s + \vec{R}\cdot\vec{X}_{su} \\ \vec{X}_s\cdot\vec{X}_t + \vec{R}\cdot\vec{X}_{ts} & \left|\vec{X}_t\right|^2 + \vec{R}\cdot\vec{X}_{tt} & \vec{X}_u\cdot\vec{X}_t + \vec{R}\cdot\vec{X}_{tu} \\ \vec{X}_s\cdot\vec{X}_u + \vec{R}\cdot\vec{X}_{us} & \vec{X}_t\cdot\vec{X}_u + \vec{R}\cdot\vec{X}_{ut} & \left|\vec{X}_u\right|^2 + \vec{R}\cdot\vec{X}_{uu} \end{bmatrix} \tag{4.9}$$

$$\vec{\delta}_i = \begin{bmatrix} \triangle s \\ \triangle t \\ \triangle u \end{bmatrix} = \begin{bmatrix} s_{i+1} - s_i \\ t_{i+1} - t_i \\ u_{i+1} - u_i \end{bmatrix} \tag{4.10}$$

$$\vec{\lambda}_i = -\begin{bmatrix} f(s_i, t_i, u_i) \\ g(s_i, t_i, u_i) \\ h(s_i, t_i, u_i) \end{bmatrix} \tag{4.11}$$

where the Jacobian matrix J_i is symmetric and is evaluated at (s_i, t_i, u_i). At each iteration, the following system of equations with the unknown $\vec{\delta}_i$ should be solved.

$$J_i\vec{\delta}_i = \vec{\lambda}_i \qquad \Longrightarrow \qquad \vec{\delta}_i = J_i^{-1}\vec{\lambda}_i \tag{4.12}$$

Then, we have from (4.12),

$$\begin{aligned} s_{i+1} &= s_i + \triangle s \\ t_{i+1} &= t_i + \triangle t \\ u_{i+1} &= u_i + \triangle u \end{aligned} \tag{4.13}$$

As the solution is solved by an iterative process, a set of convergence criteria are introduced to find out when to stop the process. They are listed in the following,

1. Point coincidence

$$\left|\vec{X}(s_i,t_i,u_i)-\vec{p}\right| \leq \epsilon_1 \tag{4.14}$$

2. Zero cosine

$$\begin{aligned}
\frac{\left|\vec{X}_s(s_i,t_i,u_i)\cdot(\vec{X}(s_i,t_i,u_i)-\vec{p})\right|}{\left|\vec{X}_s(s_i,t_i,u_i)\right|\left|\vec{X}(s_i,t_i,u_i)-\vec{p}\right|} &\leq \epsilon_2 \\
\frac{\left|\vec{X}_t(s_i,t_i,u_i)\cdot(\vec{X}(s_i,t_i,u_i)-\vec{p})\right|}{\left|\vec{X}_t(s_i,t_i,u_i)\right|\left|\vec{X}(s_i,t_i,u_i)-\vec{p}\right|} &\leq \epsilon_2 \\
\frac{\left|\vec{X}_u(s_i,t_i,u_i)\cdot(\vec{X}(s_i,t_i,u_i)-\vec{p})\right|}{\left|\vec{X}_u(s_i,t_i,u_i)\right|\left|\vec{X}(s_i,t_i,u_i)-\vec{p}\right|} &\leq \epsilon_2
\end{aligned} \tag{4.15}$$

3. Parameters range

$$\begin{cases}
\text{if} \quad s_{i+1} < 0\,, & s_{i+1} = 0 \\
\text{if} \quad s_{i+1} > 1\,, & s_{i+1} = 1 \\
\text{if} \quad t_{i+1} < 0\,, & t_{i+1} = 0 \\
\text{if} \quad t_{i+1} > 1\,, & t_{i+1} = 1 \\
\text{if} \quad u_{i+1} < 0\,, & u_{i+1} = 0 \\
\text{if} \quad u_{i+1} > 1\,, & u_{i+1} = 1
\end{cases}$$

4. Insignificant changes in parameters

$$\left|\triangle s\cdot\vec{X}_s(s_i,t_i,u_i)+\triangle t\cdot\vec{X}_t(s_i,t_i,u_i)+\triangle u\cdot\vec{X}_u(s_i,t_i,u_i)\right| \leq \epsilon_1 \tag{4.16}$$

where ϵ_1 and ϵ_2 are zero tolerances for measures of Euclidean distance and zero cosine respectively. The iterative process will be stopped if both criteria (1) and (2) are satisfied. Otherwise, a new set of parameter triple $(s_{i+1}, t_{i+1}, u_{i+1})$ is computed from equation (4.13). Then, criteria (3) and (4) are tested. The iterative process is halted if any of criteria (1), (2) and (4) is satisfied. Condition (3) is to guarantee the parameter triple computed

lies inside the parameter range $[0,1] \times [0,1] \times [0,1]$.

4.3 Parameter Triple Interpolation

In previous sections, we have discussed the algorithms for (s,t,u) parameter triple for each data point acquired by the haptic exploration system. With these parameter triples of all data points, we can recover the shape information of model points, which correspond to these parameter triples, with high confidence. For model points with other parameter triples, it is difficult to obtain their shape information. Fortunately, the shape information at these model points can be approximated by interpolation. One method to improve the "likeness" of the model points with the shape of the object under exploration is to develop an algorithm or a function to automatically generate all required (s,t,u) parameter triples under appropriate resolution as follows,

$$\begin{bmatrix} s \\ t \\ u \end{bmatrix} = \mathcal{F}(\eta,\nu), \quad \text{where} \quad (\eta,\nu) \in [0,1] \times [0,1]$$

The model surface is then parametrized by variables (η,ν). With the function $\mathcal{F}(\eta,\nu)$ and the control lattice recovered for model reconstruction, we can completely represent and reconstruction the object model under different resolution by varying the resolution of parameter triples generated. Figure 4.3 depicts a typical (s,t,u) parameter space for an object model. From this figure, we can observe that the (s,t,u)'s for a typical object surface points lie on the four side faces of a round-edged and round-cornered cube.

We can interpolate the parameter triples $(s,t,u) \in \mathcal{P}^3$ for automatic parameters generation stage of model reconstruction with Free Form Deformation by a order k B-spline surface in $\mathcal{P}^3$ [37]. Assume the active

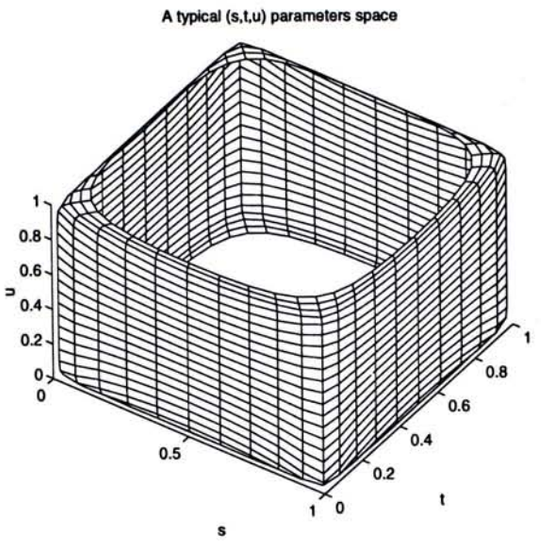


Figure 4.3: A typical (s, t, u) parameter space for a model

haptic exploration system had traced M_1 contours on the test object surface and it acquired M_2 surface points coordinates on each contour, we can construct a B-spline surface interpolating the $\mathrm{T}(\vec{x}_{p,q}) = [s_{p,q}, t_{p,q}, u_{p,q}]^T$ parameter triples computed from the Point Inversion problem algorithm (see section 4.2.2) for these $M_1 \times M_2$ data points set $\vec{x}_{p,q}$, where $p = 1, 2, \ldots, M_1$ and $q = 1, 2, \ldots, M_2$,

$$\mathrm{T}(\vec{x}_{p,q}) = \begin{bmatrix} s \\ t \\ u \end{bmatrix} = \vec{PS}(\bar{\eta}_p, \bar{\nu}_q) = \sum_{i=1}^{M_1} \sum_{j=1}^{M_2} \vec{T}_{ij} N_{i,k}(\bar{\eta}_p) N_{j,k}(\bar{\nu}_q) \qquad (4.17)$$

where $(\bar{\eta}_p, \bar{\nu}_q) \in [0, 1] \times [0, 1]$ are the parameter values for the data point $\vec{x}_{p,q}$.

The computations of $\bar{\eta}_p$ and $\bar{\nu}_q$ are analogous. Only the computation of $\bar{\eta}_p$ is discussed and the computation of $\bar{\nu}_q$ follows. For each q-th parametric curve, the parameters $\bar{\eta}_1^q, \ldots, \bar{\eta}_{M_1}^q$ by 4.18,

$$\bar{\eta}_1^q = 0$$

$$\begin{aligned} \bar{\eta}_l^q &= \bar{\eta}_{l-1}^q + \frac{\sqrt{|\mathrm{T}(\vec{x}_{l,q}) - \mathrm{T}(\vec{x}_{l-1,q})|}}{L} \qquad l = 2, \ldots, M_1 - 1 \quad (4.18) \\ \bar{\eta}_{M_1}^q &= 1 \end{aligned}$$

where $L = \sum_{i=2}^{M_1} \sqrt{|\mathrm{T}(\vec{x}_{i,q}) - \mathrm{T}(\vec{x}_{i-1,q})|}$. This is called the centripetal parametrization [37][28]. This parametrization gives better results than other parameterization methods when the data takes sharp turns. This matches with the distribution of the (s, t, u) parameter triples in $\mathcal{P}^3$ (see figure 4.3). Then, each $\bar{\eta}_i$ can be obtained by taking average across all $\bar{\eta}_i^q$, ie. $\bar{\eta}_i = \frac{1}{M_2} \sum_{l=1}^{M_2} \bar{\eta}_i^l$ with $i = 1, 2, \ldots, M_1$. The corresponding knot vector $[\eta_1, \eta_2, \ldots, \eta_{M_1+k}]$ can be computed by,

$$\begin{aligned} \eta_l &= 0 && l = 1, 2, \ldots, k \\ \eta_{l+k} &= \frac{1}{k-1} \sum_{i=l}^{k-1} \bar{\eta}_{l+i} && l = 1, 2, \ldots, M_1 - k \qquad (4.19) \\ \eta_{M_1+l} &= 1 && l = 1, 2, \ldots, k \end{aligned}$$

Similarly, $\bar{\nu}_i$, $i = 1, \ldots, M_2$ and knot vector $[\nu_1, \ldots, \nu_{M_2+k}]$ can be computed by the aforementioned method.

After computed the parameter values $\bar{\eta}_i$ with $i = 1, \ldots, M_1$, $\bar{\nu}_j$ with $j = 1, \ldots, M_2$ and knot vectors $\hat{\eta} = [\eta_1, \ldots, \eta_{M_1+k}]$ and $\hat{\nu} = [\nu_1, \ldots, \nu_{M_2+k}]$, the control points $\vec{T}_{ij}$ is computed for interpolation. The $\vec{T}_{ij}$ can be recovered efficiently by a sequence of isoparametric curve interpolation. For the q-th isoparametric curve, equation 4.17 can be written as,

$$\mathrm{T}_{p,q} = \mathrm{T}(\vec{x}_{p,q}) = \sum_{i=1}^{M_1} N_{i,k}(\bar{\eta}_p) \left(\sum_{j=1}^{M_2} N_{j,k}(\bar{\nu}_q) \vec{T}_{ij} \right) = \sum_{i=1}^{M_1} N_{i,k} \mathrm{Q}_{i,q} \qquad (4.20)$$

where

$$\mathrm{Q}_{i,q} = \sum_{j=1}^{M_2} N_{j,k}(\bar{\nu}_q) \vec{T}_{ij} \qquad (4.21)$$

The surface interpolation algorithm is as follows,

1. For $j := 1$ to M_2 do

   (a) Use $\hat{\eta}$ as the knot vector and $\bar{\eta}_p$ as parameters for interpolation;

   (b) Perform curve interpolation through $\mathrm{T}_{1,j}, \ldots, \mathrm{T}_{M_1,j}$;

   (c) Get $\mathrm{Q}_{i,j}$, the control points of the isoparametric curve $\vec{TS}(\eta, \bar{\nu}_j)$;

   End

2. For $i := 1$ to M_1 do

   (a) Use $\hat{\nu}$ as the knot vector and $\bar{\nu}_q$ as parameters for interpolation;

   (b) Perform curve interpolation through $\mathrm{Q}_{i,1}, \ldots, \mathrm{Q}_{i,M_2}$;

   (c) Get $\mathrm{T}_{i,j}$, the control points of the isoparametric curve $\vec{TS}(\bar{\eta}_i, \nu)$;

   End

This algorithm is symmetric and same interpolation (or same control points $\mathrm{T}_{i,j}$ can be obtained by performing curve interpolation through $\mathrm{T}_{j,1}, \ldots, \mathrm{T}_{j,M_2}$ first and then $\mathrm{Q}_{1,i}, \ldots, \mathrm{Q}_{M_1,i}$. With $\mathrm{T}_{i,j}$, we can generate recovered shape in any resolution.

4.4 3D Object Shape Reconstruction

4.4.1 Heuristic Approach

After the correspondence problem and the point inversion problem of all data points are solved, we can step to the process of 3D shape reconstruction. The model is expressed as follows.

$$\vec{x}_{BFFD}(s,t,u) = \sum_{i=1}^{l+1} \sum_{j=1}^{m+1} \sum_{k=1}^{n+1} \vec{B}_{ijk} N_{i,k_x}(s) N_{j,k_y}(t) N_{k,k_z}(u) \qquad (4.22)$$

We want to find $\vec{B}_{ijk}$ from a set of data point acquired by the haptic system which correspond to the RHS of (4.22). Given that there are M_1 contours,

which consists of M_2 object points in one contour, had been traced out by the haptic exploration system, the required control lattice $\vec{B}_{ijk}$ can be found by solving a large scale system of linear equation. The system of equations is setup as follows. Firstly, the data points are re-indexed as $\vec{x}^p$ and their corresponding (s,t,u) parameter triples (which are recovered by the algorithm presented in section 4.2.2) are (s^p,t^p,u^p), where $p=1,2,\ldots,M_1\times M_2$.

$$\vec{x}^p_{BFFD}(s^p,t^p,u^p)=\sum_{i=1}^{l+1}\sum_{j=1}^{m+1}\sum_{k=1}^{n+1}\vec{B}_{ijk}N_{i,k_x}(s^p)N_{j,k_y}(t^p)N_{k,k_z}(u^p) \tag{4.23}$$

This equation can also rewritten for the reason of clarity,

$$\vec{x}^p_{BFFD}(s^p,t^p,u^p)=\sum_{I=1}^{H}\vec{B}_I\bar{N}_I(s^p,t^p,u^p) \tag{4.24}$$

where $H=(l+1)\times(m+1)\times(n+1)$, $I=(k-1)(l+1)(m+1)+(j-1)(l+1)+i$[3] and $\bar{N}_I(s,t,u)=N_{i,k_x}(s)N_{j,k_y}(t)N_{k,k_z}(u)$. Then, a set of linear equations can be set up and it is rearranged in matrix form,

$$\begin{bmatrix} \bar{N}_1(s^1,t^1,u^1) & \cdots & \bar{N}_H(s^1,t^1,u^1) \\ \bar{N}_1(s^2,t^2,u^2) & \cdots & \bar{N}_H(s^2,t^2,u^2) \\ \vdots & \ddots & \vdots \\ \bar{N}_1(s^M,t^M,u^M) & \cdots & \bar{N}_H(s^M,t^M,u^M) \end{bmatrix}\begin{bmatrix} \vec{B}_1^T \\ \vec{B}_2^T \\ \vdots \\ \vec{B}_H^T \end{bmatrix}=\begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_M^T \end{bmatrix} \tag{4.25}$$

or

$$\bar{\mathbf{N}}\cdot\vec{\mathbf{B}}=\vec{\mathbf{X}} \tag{4.26}$$

where $M=M_1\times M_2$. The linear equation system (4.25) is difficult to solve because of the numerical instability of the $\bar{\mathbf{N}}$ matrix. Moreover, the condition number of this matrix is very large, usually in the order of magnitude over 10^{15}. Together with the common floating point and

[3]The indexing of control vertices will be discussed in details in Table 4.1

truncation error, the solution of this system is not accurate and reliable.

In this dissertation, a heuristic approach is proposed to find the control lattice $\vec{B}_{ijk}$ in a geometric way. The central concept of the algorithm is to construct the required control lattice in two stages. The control lattice can be considered as consisting of two parts, namely the core and the shell. The shell is actually the outermost shell of the control lattice while the core consists of all the inner shells of the control lattice. They are constructed separately in each stage. The followings describes the algorithm,

1. *Construction of Shell*

   The shell is constructed from stacking all the control polygons recovered from each contour traced by the haptic exploration system in their spatial order along the search direction. The control polygon recovery algorithm will be described in details in section 4.4.2. Links are established between adjacent control polygons on the control vertices along the two polygons (closed chains of control vertices) so that each distance between control vertices pair from the adjacent control polygons is minimized.

2. *Construction of Core*

   For each control polygons computed from traced contour, a rectangular grid is constructed as a inner grid for that layer of control lattice. The inner grid is embedded wholly in its corresponding shell layer (control polygon). After all inner grids are constructed, they are stacked up to form the core of the control lattice. Links are established between node points of two adjacent grids with the same location relative to their grids.

3. Join the Shell and Core parts to form the control lattice of the object model.

The following sections will describe in details about the above algorithms. Section 4.4.2 discusses about the recovery of control polygons of closed contours traced by the haptic exploration system for the construction of shell stage. On the other hand, section 4.4.3 discusses about the construction of the control lattice for the object model from the shell and core.

4.4.2 Closed Contour Recovery

This section concerns with closed curve interpolation. As discussed before, each contour traced by the haptic exploration system consists of a chain of object surface points along it. It is actually a plane closed curve. The close curve is interpolated by a k-th closed B-spline curve. Given that there are M_2 points, D_j along the contour and it is intended to be interpolated by a B-spline curve with shape control by n control vertices. Thus, the required control polygon consists of n control vertices.

$$\vec{X}(\bar{t}_j) = \sum_{i=1}^{n} \vec{P}_i N_{i,k}(\bar{t}_j) \tag{4.27}$$

where $j = 1, 2, \ldots, M_2$. $\bar{t}_j$ is computed by a chord length parameterization [37][27][28],

$$\begin{aligned} \bar{t}_1 &= 0 \\ \bar{t}_l &= \bar{t}_{l-1} + \frac{|D_l - D_{l-1}|}{L} \qquad l = 2, \ldots, M_2 - 1 \\ \bar{t}_{M_2} &= 1 \end{aligned} \tag{4.28}$$

where $L = \sum_{i=2}^{M_2} |D_i - D_{i-1}|$. The knot vector employed is the clamped uniform knot vector described in section 2.4.2. Then, a system of equation

can be set up for solving $\vec{P}_i$,

$$\begin{bmatrix} N_{1,k}(\bar{t}_1) & \cdots & N_{n,k}(\bar{t}_1) \\ \vdots & \ddots & \vdots \\ N_{1,k}(\bar{t}_{M_2}) & \cdots & N_{n,k}(\bar{t}_{M_2}) \end{bmatrix} \begin{bmatrix} \vec{P}_1^T \\ \vdots \\ \vec{P}_n^T \end{bmatrix} = \begin{bmatrix} \vec{X}_1^T \\ \vdots \\ \vec{X}_{M_2}^T \end{bmatrix} \tag{4.29}$$

or

$$\mathbf{N} \cdot \vec{\mathbf{P}} = \vec{\mathbf{X}} \tag{4.30}$$

where $\mathbf{N}$ is a $(M_2 \times n)$ matrix.

The above formulation is complete for open curve only. On the other hand, extra $(k-1)$ control vertices should be introduced at the end of the control vertices chain in the closed curve interpolation process. These control vertices are called *pseudo-vertices* and they should be equal to the first corresponding $(k-1)$ control vertices in the vertices chain individually. That means there will be $(n+k-1)$ control vertices recovered, namely, $\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_{n-1}, \vec{P}_n, \vec{P}_{n+1}, \ldots, \vec{P}_{n+k-1}$ and the matrix $\mathbf{N}$ in (4.30) becomes a $M_2 \times (n+k-1)$ matrix. The conditions of the pseudo-vertices $\vec{P}_{n+1}, \ldots, \vec{P}_{n+k-1}$ are listed,

$$\begin{aligned} \vec{P}_{n+1} &= \vec{P}_1 \\ \vec{P}_{n+2} &= \vec{P}_2 \\ &\vdots \\ \vec{P}_{n+k-2} &= \vec{P}_{k-2} \\ \vec{P}_{n+k-1} &= \vec{P}_{k-1} \end{aligned} \tag{4.31}$$

Moreover, in order to maintain the continuity between the junctions of the closed contour[4], two derivatives continuities constraints are imposed on the junction. The first and the second derivatives should be continuous

[4] A closed curve can be considered as an open curve with the same endpoints

at the junction of the curve,

$$\begin{aligned} \vec{X}'(\bar{t}_1) &= \vec{X}'(\bar{t}_{M_2}) \\ \vec{X}''(\bar{t}_1) &= \vec{X}''(\bar{t}_{M_2}) \end{aligned}$$

or

$$\sum_{i=1}^{n+k-1} \vec{P}_i \left[N'_{i,k}(\bar{t}_1) - N'_{i,k}(\bar{t}_{M_2}) \right] = 0 \tag{4.32}$$

$$\sum_{i=1}^{n+k-1} \vec{P}_i \left[N''_{i,k}(\bar{t}_1) - N''_{i,k}(\bar{t}_{M_2}) \right] = 0 \tag{4.33}$$

By imposing the pseudo-vertices and derivatives continuities constraints onto the linear equations system (4.30), we have,

$$\hat{\mathbf{N}} \cdot \hat{\mathbf{P}} = \hat{\mathbf{X}} \tag{4.34}$$

where,

$$\hat{\mathbf{X}} = \begin{bmatrix} \vec{X}^T(\bar{t}_1) \\ \vec{X}^T(\bar{t}_2) \\ \vdots \\ \vec{X}^T(\bar{t}_{M_2}) \\ \hline 0 \\ \vdots \\ 0 \end{bmatrix} \quad (4.35) \qquad \hat{\mathbf{P}} = \begin{bmatrix} \vec{P}_1^T \\ \vec{P}_2^T \\ \vdots \\ \vec{P}_{n+k-1}^T \\ \hline 0 \\ \vdots \\ 0 \end{bmatrix} \quad (4.36)$$

$$\hat{\mathbf{N}} = \begin{bmatrix} N_{1,k}(\bar{t}_1) & \cdots & N_{n+k-1,k}(\bar{t}_1) \\ \vdots & \ddots & \vdots \\ N_{1,k}(\bar{t}_{M_2}) & \cdots & N_{n+k-1,k}(\bar{t}_{M_2}) \\ \hline 1 & 0 \ \cdots \ -1 \ 0 & \cdots \\ 0 & \ddots \ \cdots \ \ddots & 0 \\ 0 & \cdots \ 1 \ 0 \ \cdots & -1 \\ \hline N'_{1,k}(\bar{t}_1)- & \cdots & N'_{n+k-1,k}(\bar{t}_1)- \\ N'_{1,k}(\bar{t}_{M_2}) & & N'_{n+k-1,k}(\bar{t}_{M_2}) \\ N''_{1,k}(\bar{t}_1)- & \cdots & N''_{n+k-1,k}(\bar{t}_1)- \\ N''_{1,k}(\bar{t}_{M_2}) & & N''_{n+k-1}(\bar{t}_{M_2}) \end{bmatrix} \quad (4.37)$$

The matrix $\hat{\mathbf{X}}$ has dimension of $(M_2 + k + 1) \times 3$ with the last $(k+1)$ rows are zeros and $\hat{\mathbf{P}}$ is a $(n + k - 1) \times 3$ matrix. The matrix $\hat{\mathbf{N}}$ has dimension of $(M_2 + k + 1) \times (n + k - 1)$, with the first M_2 rows express the usual interpolation constraints. The middle $(k - 1)$ rows and the last 2 rows of $\hat{\mathbf{N}}$ express the pseudo-vertices constraints and the derivatives continuities constraints respectively. The pseudo-vertices constraints submatrix consists

of a diagonal of 1 from the first column to the $(k-1)$-th column and a diagonal of -1 from the $(n+1)$ column to the last column. The solution of the equation system can be obtained by taking pseudo inverse of the matrix $\hat{\mathbf{N}}$,

$$\hat{\mathbf{P}} = \hat{\mathbf{N}}^{\dagger}\hat{\mathbf{X}} \tag{4.38}$$

The solution of (4.38) is not the best because there are some errors between the pseudo-vertices and their corresponding control vertices counterparts. This can be remedied by an iterative parameter optimization procedure. The objective of the optimization is to minimize the sum of squares of distance between the pseudo-vertices and their control vertices counterparts, $\mathcal{F}$,

$$\mathcal{F} = \sum_{i=1}^{k-1} \left| \vec{P}_i - \vec{P}_{n+i} \right|^2 \tag{4.39}$$

The iterative process begins with the chord length parameterization as the initial parameterization. At each iteration, the value of $\mathcal{F}$ is checked. If it is greater than a predefined tolerance, the parameterization will be updated as $\hat{t}_i$,

$$\hat{t}_i = \bar{t}_i + \frac{\left(\vec{D}_i - \vec{X}(\bar{t}_i)\right) \cdot \left.\frac{\partial \vec{X}}{\partial t}\right|_{t=\bar{t}_i}}{\left| \left.\frac{\partial \vec{X}}{\partial t}\right|_{t=\bar{t}_i} \right|^2}, \qquad i = 1, 2, \ldots, M_2 \tag{4.40}$$

This parameter optimization technique was proposed by D. F. Rogers and N. G. Fog [28][51][52]. The predefined tolerance can be assigned as a small percentage of the mean of the largest and smallest diameters of the closed contour. After the parameterization has been updated, a new $\hat{\mathbf{N}}$ is generated, a new set of control vertices is computed from (4.38) and the new $\mathcal{F}$ is tested. This process continues until $\mathcal{F}$ is smaller than the predefined tolerance. The required control polygon for the closed contour is then the first n control vertices obtained, excluding all pseudo-vertices.

4.4.3 Control Lattice Recovery

This section mainly concerns with the construction of control lattice for the object model. The construction algorithm is described in details in the following,

1. Set the size of the control lattice for the object model. Define the dimension of each grid layer of the control lattice as $m_1 \times m_2$. The dimension of the lattice in the z-axis should be equal to the number of contours traced, M_1. Then, the dimension of the control lattice is $m_1 \times m_2 \times M_1$. Typical dimensions of each grid layer are 5×5 and 7×7.

2. For each contour traced,

   (a) Recover the control polygon for that contour using the algorithm described in section 4.4.2. The number of control vertices in the control polygon should be $2(m_1 + m_2 - 2)$. This recovered control polygon form a boundary of its corresponding control layer grid.

   (b) Form an inner grid for each control layer by an iterative approach.

      i. Form the bounding box of the control polygon by finding the maximum and minimum x- and y- coordinates of all control vertices. The bounding box can be represented by four number showing the coordinates of the its four corners. The center and the size of the bounding box can also be obtained.

      ii. Generate an ellipse that is wholly inscribed in the bounding box with its major and minor diameters equals to the length and width of the bounding box.

      iii. Use Fibonacci Search to find the maximum scaling factor of the diameters of the ellipse so that it is wholly inscribed in

the control polygon. This step will be described in details in Note (1).

iv. Construct a bounding box for the ellipse with its major and minor diameters as the size of the bounding box and they share the same center. This will be the boundary of the control lattice core at that layer.

v. Construct the inner node points to complete the inner grid at that layer for the lattice core.

(c) Reorder the control vertices chain for the closed contour so that the first control vertices has the minimum phase angle difference with the northwest corner of its bounding box generated in step (2(b)i).

(d) Join the boundary (the control polygon found in step (2a)) and the inner grid (generated from step (2(b)v) and a control lattice is formed.

3. Join all control layers formed in step (2) together according to their spatial order in the contours searching direction.

Note (1)

Use the Fibonacci Search to find the maximum scaling factor $\bar{\lambda}$ of the major and minor radii of the ellipse so that it is wholly inscribed in the control polygon. The scaling factor lies in the range of $[\lambda_{min}, 1]$ where λ_{min} is a positive number smaller than 1. This is an initial range of the scaling factor. At each iteration of the search, a new range is updated with two bounds candidates, λ_1 and λ_2, $\lambda_1 < \lambda_2$, generated from the Fibonacci sequence and the bounds generated in the previous iteration. Two ellipses, C_1 and C_2, are generated with their radii scaled by λ_1 and λ_2 respectively. The size of C_1 is smaller than C_2. The two ellipses are then tested whether they are inside the control polygon or not. This can be tested by checking

intersections among all edge lines of the control polygon and the ellipse. If there is/are intersections among edge lines and the ellipse, the control polygon is either wholly inside or crossing the ellipse. Otherwise, the ellipse is wholly inscribed in the control polygon. There are several cases for selecting the new scaling factor range according to the relative position of the control polygon to the two ellipses and they are listed in Table 4.2, The

Case	Edge lines cross with		Relation with		New Range	Remark
	C_1	C_2	C_1	C_2		
1	√	√	C/O	C/O	$[\lambda_{min}, \lambda_1]$	
2	×	√	I	C/O	$[\lambda_1, \lambda_2]$	
3	×	×	I	I	$[\lambda_2, 1]$	
4	√	×	C/O	I	NIL	Contradiction since $\mathbf{size}(C_1) < \mathbf{size}(C_2)$
C=Cross, O=Outside and I=Inside						

Table 4.2: Four cases of relative positions of the control polygon to the two ellipses

iterative process stops when the difference between λ_1 and λ_2 is smaller than a predefined tolerance.

Chapter 5

Implementation

5.1 Introduction

All the computation tasks in this project are implemented in a software called MATLAB v4.2c [53]. MATLAB is good at numeric computation and data visualization. As the operations in 3D shape reconstruction are computational expensive, MATLAB is chosen as implementation tools in this project. Moreover, MATLAB has a series of auxiliary toolboxes for users to solve problems in a large variety of fields of study, for example, the Splines Toolbox.

5.2 Implementation Tool – MATLAB

MATLAB [53] is a technical computing environment for high performance numeric computation and data visualization. It integrates general numerical analysis, matrix computation and graphics in an environment that is easy to use and user friendly. MATLAB has become the standard computing tools for study and research in both the academic and industrial uses. That is why MATLAB has various versions that can run on different computing platforms, including PC's with Windows NT/95, Sun

SPARCstations, HP 9000, DEC RISC and so on.

The name MATLAB is the abbreviation for Matrix Laboratory. As indicated from its name, MATLAB is strong in matrix numeric computation and all data elements defined in it is in a form of matrix. Moreover, MATLAB is a powerful data visualization tool. It supports 2D plots, 3D mesh and/or surface plots, contours plots and so on. MATLAB also supports rendering on surfaces. One of the advantages of MATLAB is its extensibility. MATLAB has a script language so that users can create their own application routines that suit their needs. All the computation tasks in this project is implemented in MATLAB script language. Moreover, MATLAB has external interfaces linking to various applications and computer languages in order to further extend its ability, including MAPLE, Mathematica, Microsoft Word, C/C++, Fortran, etc. For instance, the data and program control can be exchanged between C/C++, Fortran and MATLAB by the uses of MAT and MEX files respectively [54].

Another advantage of MATLAB is that it has various auxiliary toolboxes concerning different applications. Areas in which toolboxes are available include control systems, signal processing, statistics, fuzzy logic, neural networks, finance, image processing and so on. These toolboxes are developed mainly based on MATLAB script language and some are written in MEX files (for decreases in execution time for computational expensive routines). The functions in various MATLAB auxiliary toolboxes can extend the MATLAB environment so as to solve particular categories of problems. There is also an auxiliary package in use with MATLAB called Simulink. Simulink allows users to run simulations on various kinds of systems. Simulink also allows users to design different systems by connecting pre-defined or user-defined different building blocks and to observe the behaviour of the system through "artificial CRO". It even can generate animation about the behaviour of the systems. This is very useful for system

designers.

5.2.1 Optimization Toolbox

The Optimization toolbox [55] for MATLAB is a powerful toolbox for minimization or maximization on general nonlinear functions. It can tackle a large variety of optimization problems. The functions can be scalar functions or vector functions. It can also handle unconstrained and constrained optimization problems. For constrained problems, it can handle both the equality and inequality constraints. Moreover, the toolbox has efficient algorithm implementations for solution in some typical optimization problem classes, like the minimax problem, the nonlinear least squares problems, quadratic programming and multi-objective optimization problems.

For each optimization problem, different parameters set governing the optimization procedure can be assigned in order to find better approximation to the optimal solution numerically. Users can change the termination criteria (error tolerance) for the independent variables and the objective function of the problem, the maximum number of iteration passed, the minimum and maximum perturbation in variables for finite difference gradient calculations and so on. Users are even allowed to change the algorithms used in the optimization process, like, the main optimization algorithm and the search direction algorithm. On the other hand, information about the optimization process is returned to users for analysis. The information includes, number of iteration passed, number of function and gradient evaluations and so on.

In this project, solution of system of nonlinear equations is needed. This is a computationally expensive process. The Optimization Toolbox can handle it with no effort by the function called *fsolve*. This function solves system of nonlinear equations in a least squares sense. It is built in

with two common algorithms, namely, the Gauss-Newton method and the Levenberg-Marquardt method. This function is very robust and efficient.

5.2.2 Splines Toolbox

The Splines Toolbox [56] for MATLAB is written by Carl de Boor. The core of this toolbox is a MATLAB implementation of algorithms presented in the book, *A Practical Guide to Splines* by the same author [57]. The main usage of this toolbox is for construction and manipulation of piecewise polynomial functions. Splines usually are a chain of curve segments or surface patches joining together with certain level of continuity condition maintained at the junctions. This toolbox provides functions for spline interpolation, least squares approximation to functions, splines smoothing, evaluation of derivatives and integral of splines functions and so on.

The Splines Toolbox supports two representations of spline functions, namely the pp-form and the B-form. The pp-form of a spline describes a spline by its breakpoints or knots $\zeta_1, \ldots, \zeta_{l+1}$ and the local polynomial coefficients α_{ij} of its segments

$$g_j(x) = \sum_{i=1}^{k} \alpha_{ij} \frac{(x - \zeta_j)^{k-i}}{(k-i)!}$$

The pp-form is convenient for evaluation of splines, their derivatives and their integrals. On the other hand, the B-form describes a spline as a linear combination of B-spline basis functions as explained in section (2.4.2). One requirement for the knot vector of B-spline basis is that the knot vector is a non-negative monotonic increasing sequence. The B-form is convenient for construction of splines. Most functions in the Splines Toolbox support both representations and functions for conversion between the two forms are also provided.

In spline construction, interpolation and approximation, users

always need to evaluate spline basis function values and derivatives and integrals with a lot of input parameters. The Splines Toolbox provides a function called *spcol* which can satisfy the needs for these purposes. This function can generate a collocation matrix, in which each row contains the r-th derivative at t of the j-th B-spline basis, $\frac{d^r N_{j,k}(t)}{dt^r}$. The r can be specified by repeating the input parameter t by $(r+1)$ times in the input parameter vector [56]. This function is widely used in this project for calculation of B-spline basis function values and its derivatives.

5.3 Geometric Model Implementation

In implementing B-spline based FFD or NFFD, the B-spline basis function for different parameters should be computed frequently. In the Spline toolbox [56] of MATLAB, the function, *spcol*, generates the B-spline collocation matrix. The spline collocation matrix contains B-spline basis function and its derivatives at particular parameter values. This function can save computation time because it can generate a B-spline basis function and derivative values corresponding to a lot of parameter values by one function call.

The implementation process basically follows the four main step of FFD. A regularly parametrized superquadrics is first generated. Then, a control lattice is generated with all control vertices coordinates computed. The control lattice is parallelepiped in shape. After that, the corresponding parametric coordinates with respect to the control lattice frame of each object point are computed. For FFD using Bernstein basis, the parameters can be calculated easily by (2.18)[1]. For BFFD or NFFD, the point inversion problem (see section 4.2.2) should be solved for each data point of the original shape. It is cumbersome to calculate the (s,t,u) parameter triples

[1]This is actually a point inversion problem. For trivariate Bernstein solid with a regular control lattice, the point inversion problem can be simplified to (2.18). A regular control lattice is one that has scale same as the Euclidean space on which it is defined.

for all data points every time FFD is performed. As we have proved that B-spline basis is invariant under affine transformation (see 4.2), the point inversion problem can be speeded up greatly. We first map the original object points $\vec{p} = [x, y, z]^T$ into $\hat{p} = [\hat{x}, \hat{y}, \hat{z}]^T$ in a normalized domain, which is defined as $\mathcal{N}^3 \stackrel{def}{=} [-1,1] \times [-1,1] \times [-1,1]$, by the mapping, $\Phi : \mathcal{E}^3 \mapsto \mathcal{N}^3$,

$$\hat{p} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = \Phi(\vec{p}) = \begin{bmatrix} \frac{2}{sx}(x - p_0^x) - 1 \\ \frac{2}{sy}(y - p_0^y) - 1 \\ \frac{2}{sz}(z - p_0^z) - 1 \end{bmatrix} \tag{5.1}$$

and its inverse is

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \Phi^{-1}(\hat{p}) = \begin{bmatrix} \frac{sx}{2}(\hat{x} + 1) + p_0^x \\ \frac{sy}{2}(\hat{y} + 1) + p_0^y \\ \frac{sz}{2}(\hat{z} + 1) + p_0^z \end{bmatrix} \tag{5.2}$$

where $\vec{p_0} = [p_0^x, p_0^y, p_0^z]^T \in \mathcal{E}^3$ is the base point of the original control lattice and $\vec{siz} = [sx, sy, sz]^T$ are the sizes of the original shape at the x-, y- and the z-axes respectively. The initial control lattice of original shape is mapped to $\mathcal{N}^3$. As the original shape is in the superquadric model (see section 2.2.2), the shape of the original object can be controlled by two parameters, namely, ϵ_1 and ϵ_2 and its size.In the normalized domain $\mathcal{N}^3$, we can pre-computed a set of parameter triples (s, t, u) with typical original shapes, like sphere ($\epsilon_1 = 1$ and $\epsilon_2 = 1$), cylinder ($\epsilon_1 = 0.2$ and $\epsilon_2 = 1$) and cube ($\epsilon_1 = 0.2$ and $\epsilon_2 = 0.2$). These pre-computed parameter triples can be applied to the point inverse problem of each original data point, generated from superquadric model with different ϵ_1 and ϵ_2, nearest to the 3D point of typical superquadric model as initial guess of the iterative point inversion process. With this initial guess, the number of iteration can be lowered to 2 to 3 for each object point which is greatly smaller than the number of

iterations for convergence with other initial guesses, say [0.5, 0.5, 0.5] which the center of the parametric space.

After finding all parameters coordinates triple corresponding to each object point, the deformation can then be performed according to the newly deformed control lattice, according to (2.20) (for Bernstein-based FFD) or (2.22) (for B-spline based FFD) or (2.24) (for NURBS-based FFD). From the algorithm of FFD, we can see that the computational complexity mainly depends on how small the step of the grid of the spherical coordinates (η, ω) are discretized. It is independent of the complexity of the deformed shape of the model.

5.3.1 FFD Examples

Figures-5.1 to 5.7 show, as an example, the NURBS-based free-form deformation process. Figure-5.1 depicts that the original shape is an ellipsoid. Figures-5.2 and 5.3 show the different views of the deformed lattice. The lattice looks like a zig-zag shape with a linear tapered extension in the bottom. Figure-5.4 depicts that the original shape is embedded to the control lattice. Figures-5.6 and 5.5 shows the different view of the deformed object model. The deformed model generally follows the shape of the control lattice. Figure-5.7 shows the shaded version of the deformed object model.

The deformed shapes using Bernstein-based FFD and NURBS-based FFD are different. The shape from Bernstein-based FFD does not follow the control lattice very much, compared to the case of using NFFD. Figure-5.8 shows a the deformed object using Bernstein-based FFD, under the same parameters settings of the above-mentioned case. This figure illustrates that Bernstein-based FFD usually cannot follow the portions with high curvature (e.g. zig-zag shape) of control lattice. However, the computation time of Bernstein-based FFD is much lower than that of NFFD. It is because the process of finding the parameter coordinates triple (s, t, u) for each

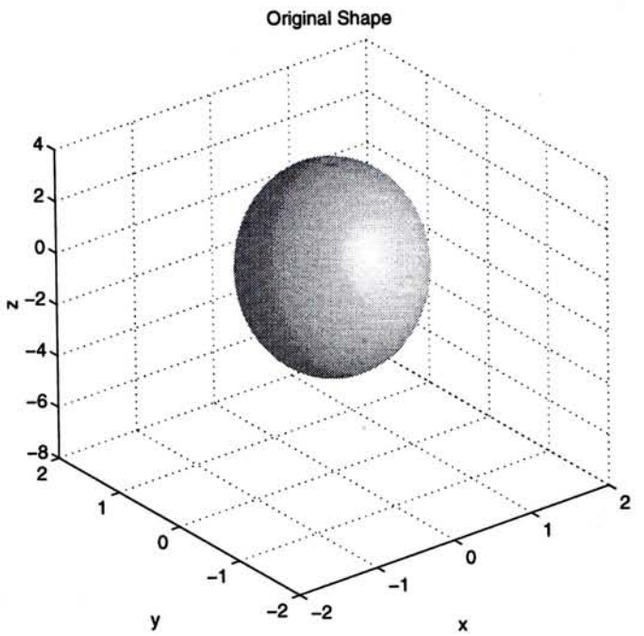


Figure 5.1: Original shape

object point of NFFD needs to solve the point inversion problem while that of Bernstein-based FFD needs only to compute the values from (2.18), which is an efficient computation compared to solving nonlinear multivariate equations by numerical search.

5.4 Shape Reconstruction Implementation

In chapter 4, an algorithm of model recovery is developed. The implementation steps are listed below,

1. From the modified Grasp by Containment EP, we get a rough shape in superquadric model and the dimension $\vec{siz}$ of the object. This will be input information of the original control lattice.

2. Compute the centroid of the data points $\vec{p_0}$ acquired by the tactile sensor array. We can construct an original control lattice of the size measured of the target object and a base point at $(\vec{p_0} - \vec{siz})$.

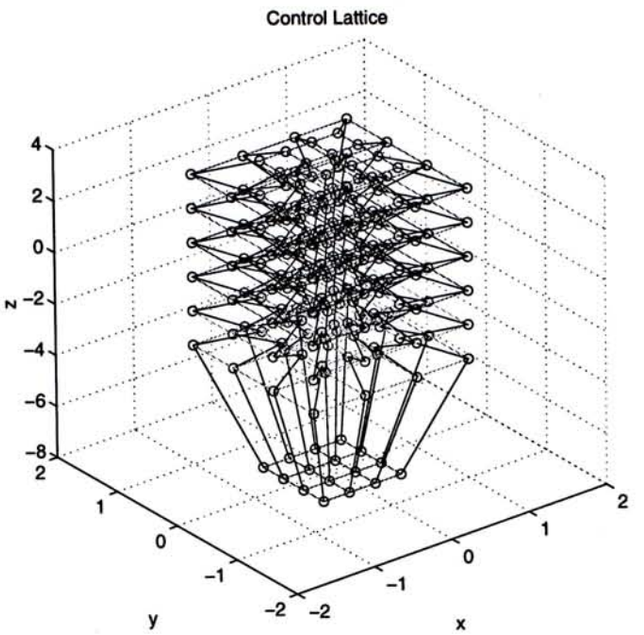


Figure 5.2: Perspective view of the deformed control lattice

3. Transform the data points into the normalized domain $\mathcal{N}^3$ and find the spherical coordinates of the data points with respect to the normalized space.

4. Establish correspondence between data points and model points on the original shape according to the criteria described in section 4.2.

5. Solve the point inversion problem for each data point. The corresponding (s, t, u) parameter triple for each data point is then obtained.

6. Perform parameter triple interpolation (see section 4.3) for model reconstruction.

7. Compute the control polygon for each contour traced by the haptic exploration system by the algorithm developed in section 4.4.

8. Generate the inner grid that can be wholly inscribed in the control polygon obtained in step (7) for each contour. This inner grid becomes

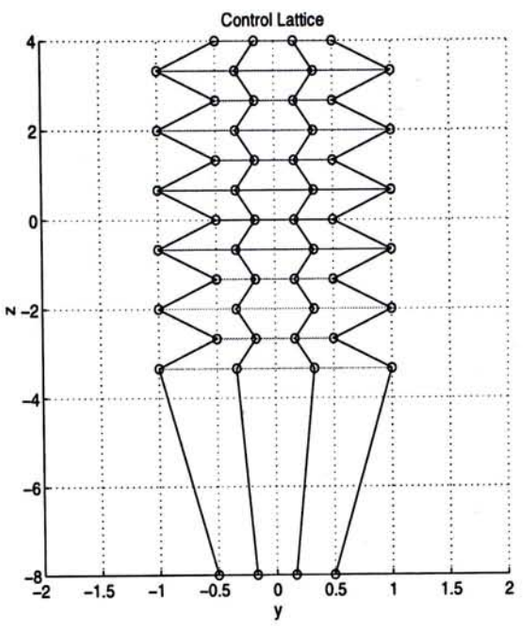


Figure 5.3: Side view of the deformed control lattice

one layer of the resultant control lattice for model reconstruction. Then, the control polygon obtained in step (7) and its corresponding inner grid for each contour are linked together by the algorithm described in section 4.4.2. Then, one control layer of the control lattice for each contour is constructed.

9. Stack up all control layers constructed in step (8) according to their spatial order along the contour search direction. A set of linkage is developed between adjacent control layers. The control lattice for model reconstruction is constructed.

10. Rebuild the object model from the control lattice obtained in step (9) and the parameter triple implementation developed in step (6) for automatic parameter triple generation by Free Form Deformation (FFD).

A set of functions is developed for MATLAB to implement the above algorithm. Some of them are essential to the whole implementation. They

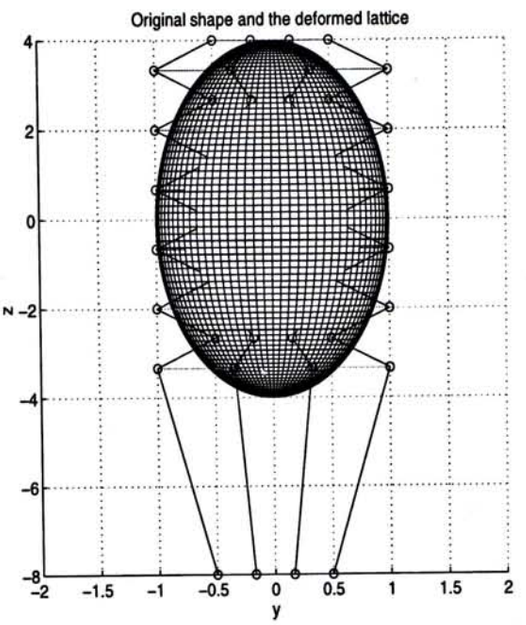


Figure 5.4: Original shape and control lattice

can be found attached in the Appendix on p. 133. They are listed and described in a hierarchical structure as follows,

1. formlattice.m This is the main control program for the model reconstruction implementation. It governs the flow of the programs according to the description above, from finding correspondence between data points and original model points, building control layers from all contours, forming control lattice by control layers stacking, parameter triple interpolation to object model reconstruction.

1.1 newfparam.m This function solves the Point Inversion Problem (see section 4.2.2) for all data points

1.1.1 findp.m It implements the iterative algorithm for solving the Point Inversion Problem for one data point.

1.2 buildgrid.m It constructs control layer for each contour. This includes building of the boundary and the core of a control layer.

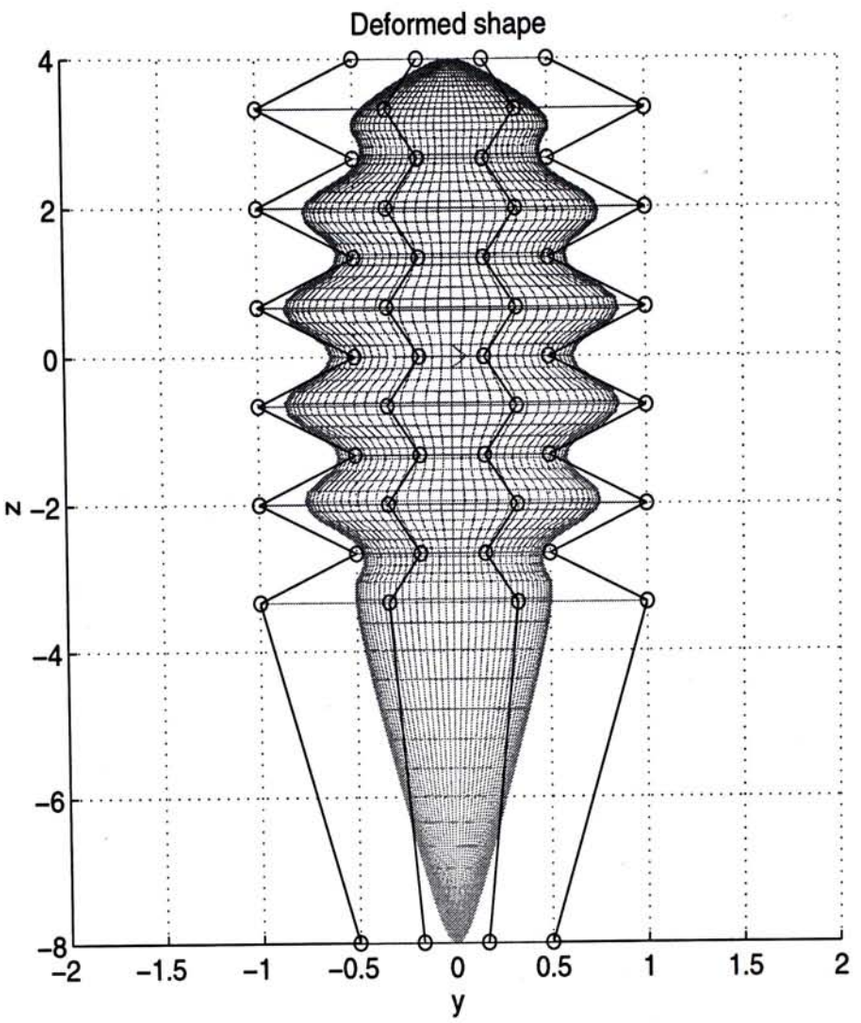


Figure 5.5: Side view of deformed shape

1.2.1 getpol.m This function recovers the control polygon of a closed contour. The control polygon will become the boundary of a control layer.

1.2.2 isint.m This function checks whether there are intersections between ellipses and the control polygon in the stage of determining the size of the inner grid, which will become the core of a control layer. Details can be referred to section 4.4.3.

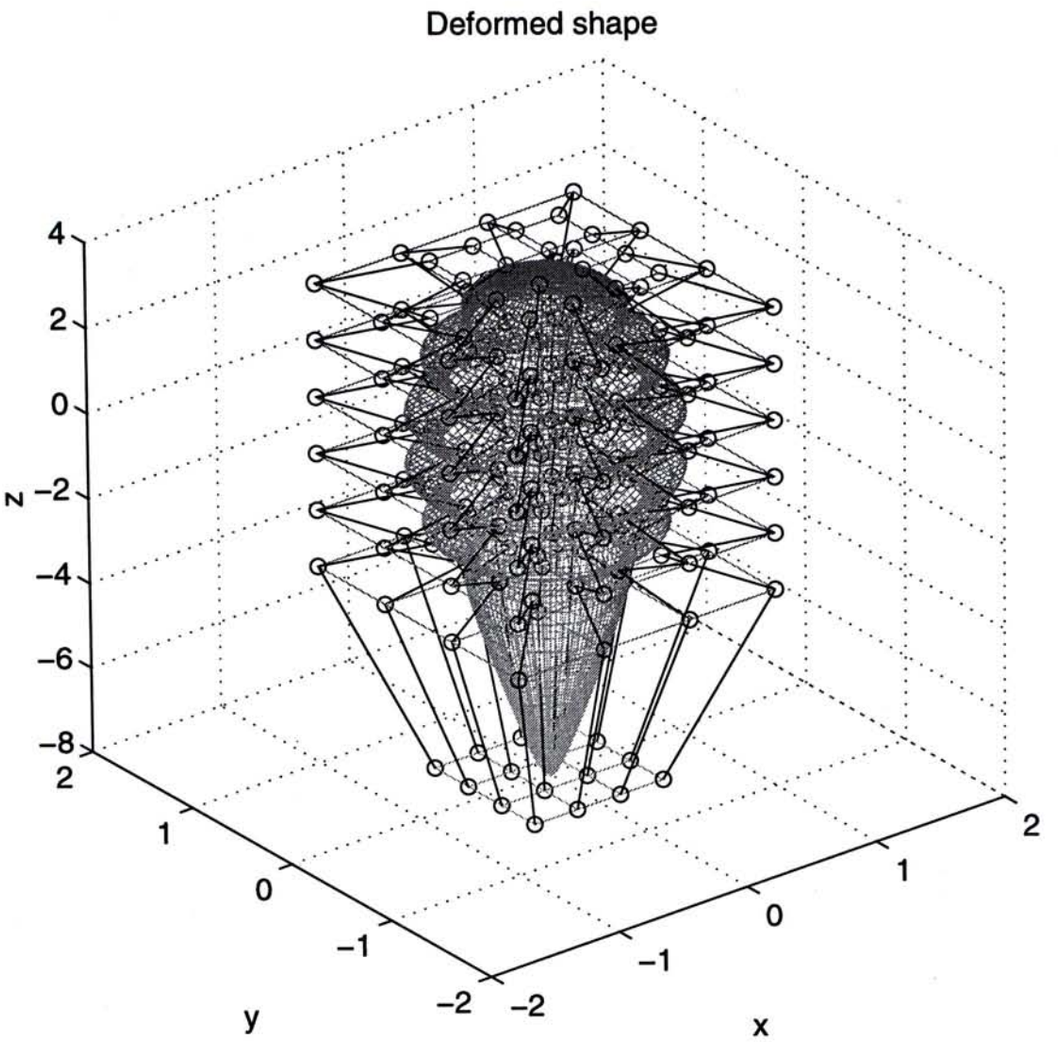


Figure 5.6: Deformed shape

1.2.3 phasediff.m It determines the phase angles of control vertices in one layer relative to the northwest corner of the corresponding bounding box. This information is important in rearrangement the control vertices for control layer boundary.

1.3 surffit.m This function tackles the automatic (s, t, u) parameter triple generation by parameter triple interpolation, discussed in section 4.3.

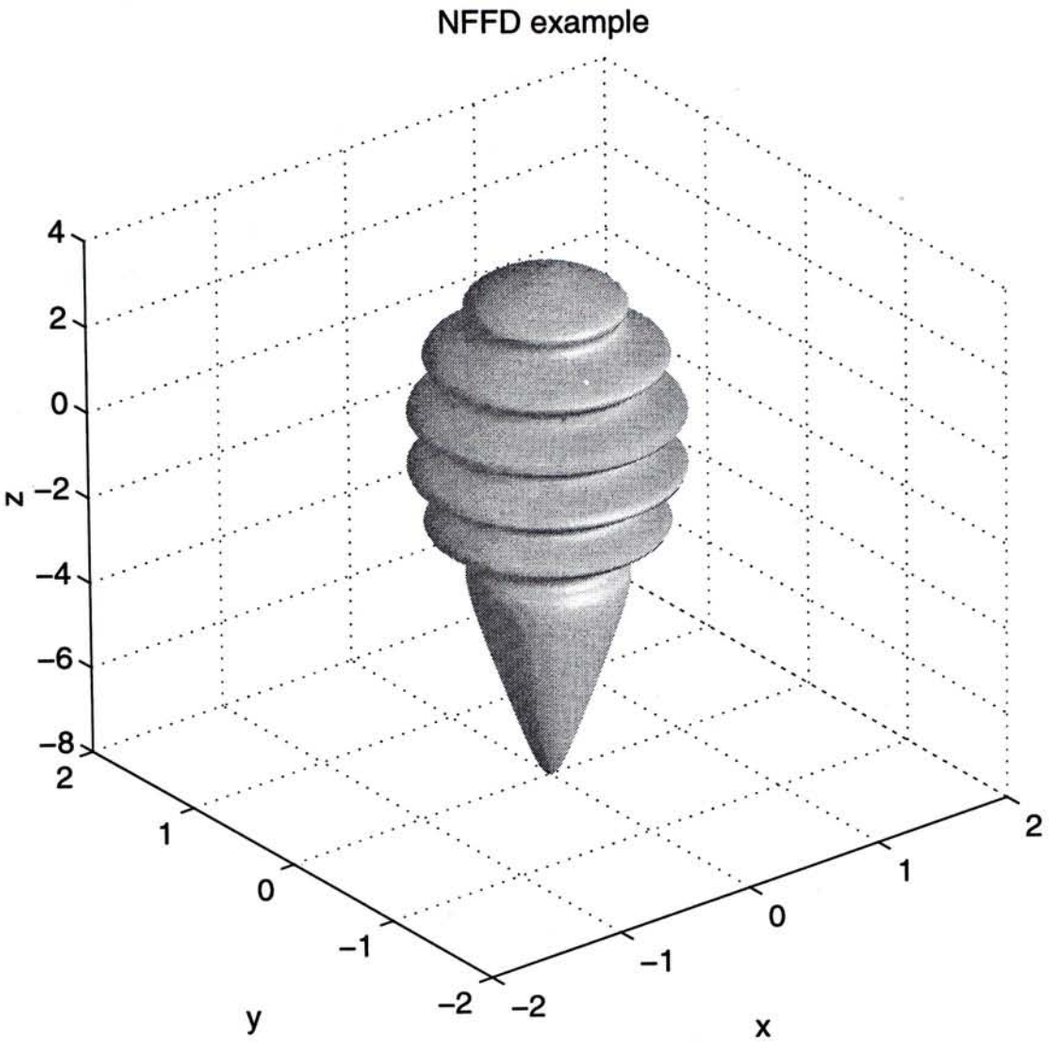


Figure 5.7: Deformed shape (shaded)

1.3.1 surfparam.m This function computes all parameters pair $(\bar{\eta}, \bar{\nu})$ corresponding to all (s, t, u) parameter triples for B-spline surface interpolation.

1.3.1.1 curparam.m This function computes parameter values $\bar{\eta}$ and $\bar{\nu}$ of each iso-parametric curves.

1.4 fnffd1.m This function performs the Free Form Deformation (FFD) process for object model reconstruction. This function is also employed for testing data generation.

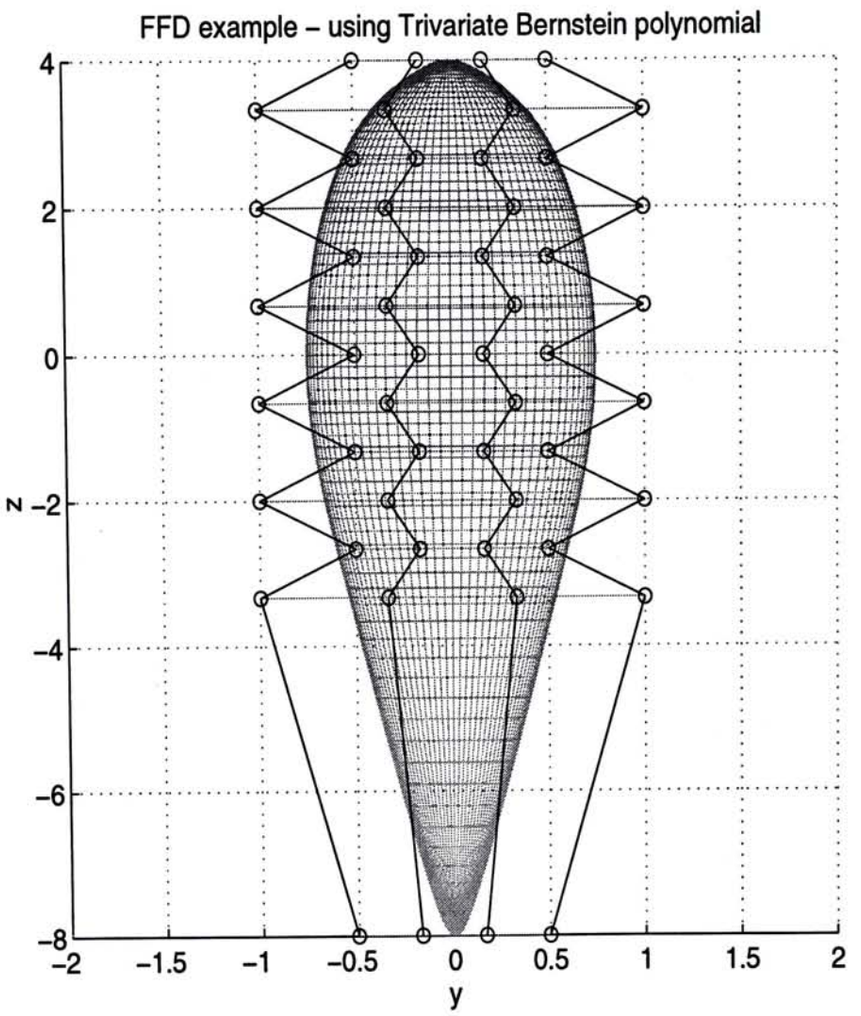


Figure 5.8: Example using Bernstein-based FFD

2. Auxiliary functions These are several auxiliary functions that are also very crucial in the implementation. For instance, *spcol* computes the B-spline basis and its derivatives values for a list of parameters for a given order and knot vector and *pinv* calculates the pseudo-inverse of a matrix and it has been widely employed in control polygons recovery steps and surface interpolation.

5.5 3D Model Reconstruction Examples

In this section, two examples for 3D models reconstruction will be illustrated. The first example illustrates the 3D model reconstruction from a set of uniformly sampled contours along the search direction. On the other hand, the second example illustrates model reconstruction from a set of contours points traced according to the exploratory procedure (EP) proposed in section 3.2.2.

5.5.1 Example 1

The example illustrates the 3D model reconstruction of a complex object with frequent curvature changes along the search direction on the object surface, as shown in Figure- 5.9. In order to rebuild the object model, 27 contours had been traced uniformly along the search direction on the object surface. Each contour consisted of 81 object surface points along it. Using the algorithm proposed in section 4, the object model is reconstructed in MATLAB running on a Sun Ultra 1 machine and the time elapsed for the model recovery is 1042.2s or about 17.37 minutes. The control lattice for that object model is depicted in Figure- 5.10. Figure- 5.11 shows the recovered shape of test object 1. The crosses located in the figure locate all data points used in 3D model reconstruction. Figure- 5.12 and Figure- 5.13 depict the top and side views of the recovered object model. From these

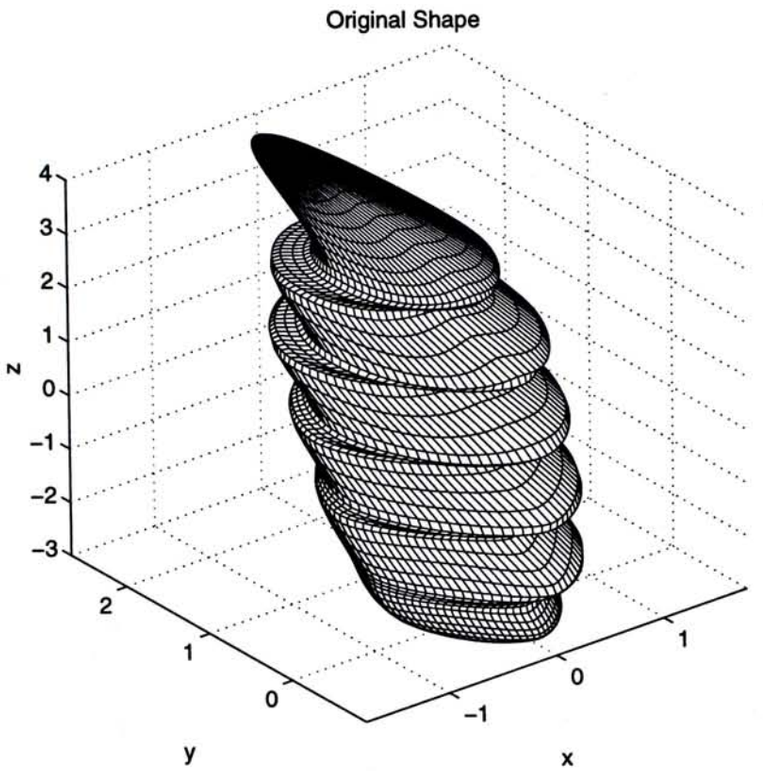


Figure 5.9: Test object 1

two figures, the recovered shape generally follows the shape changing of test object 1. However, there is little shape discrepancy between the sensed data points and the recovered shape. In addition, the shaded object model is also shown in Figure- 5.14.

5.5.2 Example 2

The example illustrates the 3D model reconstruction of the test object shown in Figure- 5.15. The separation between neighbouring contours is non-uniform. The contour separations are set according to the criteria for step size determination discussed in section In order to rebuild the object model, 24 contours had been traced uniformly along the search direction on the object surface. Each contour consisted of 41 object surface points along it. Using the algorithm proposed in section 4, the object model is reconstructed in MATLAB running on a Sun Ultra 1 machine and the time

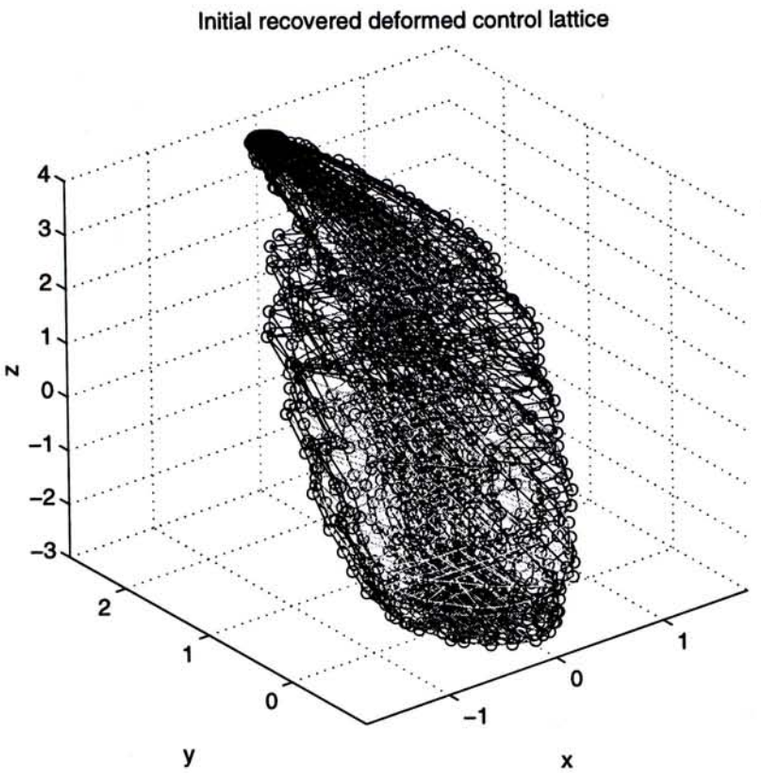


Figure 5.10: Recovered control lattice of test object 1.

elapsed for the model recovery is 594.4s or about 9.9 minutes. The control lattice for that object model is depicted in Figure- 5.16. Figure- 5.17 shows the recovered shape of test object 2. The crosses located in the figure locate all data points used in 3D model reconstruction. Figure- 5.18 and Figure- 5.19 depict the top and side views of the recovered object model. From these two figures, the recovered shape matches with the shape of test object 2 quite well. However, there is little shape discrepancy between the sensed data points and the recovered shape. In addition, the shaded object model is also shown in Figure- 5.20.

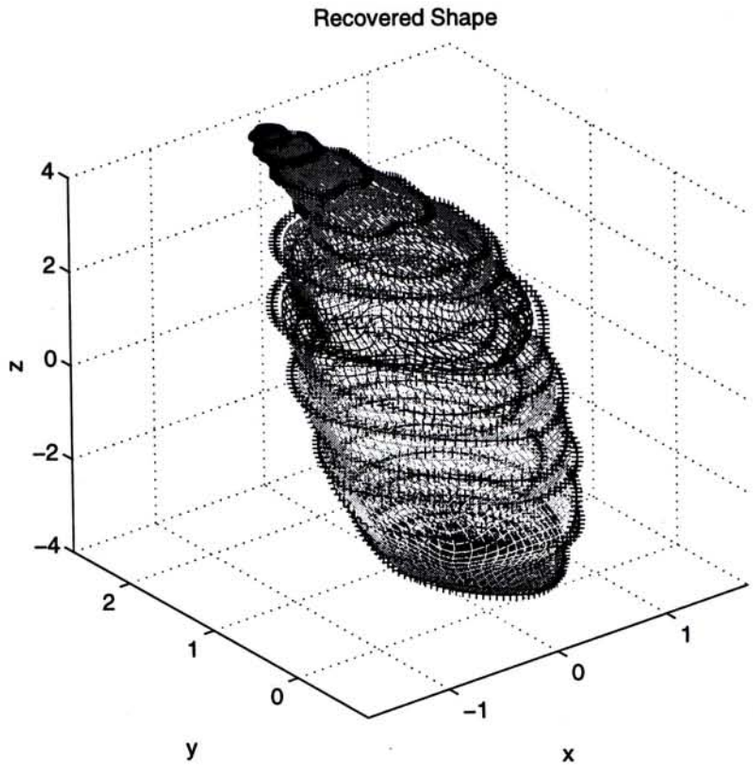


Figure 5.11: The recovered shape of test object 1.

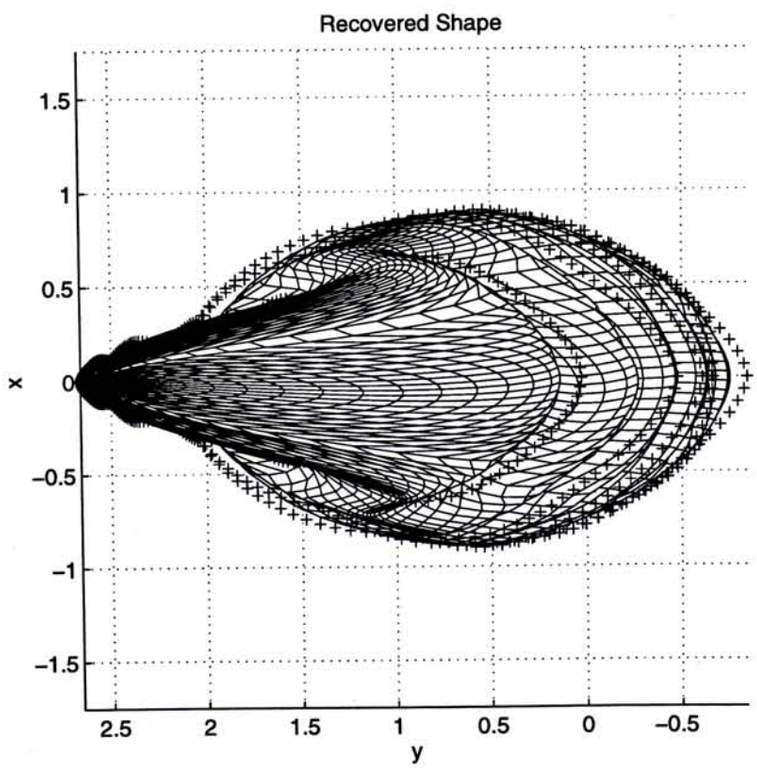


Figure 5.12: Top view of the recovered shape of test object 1.

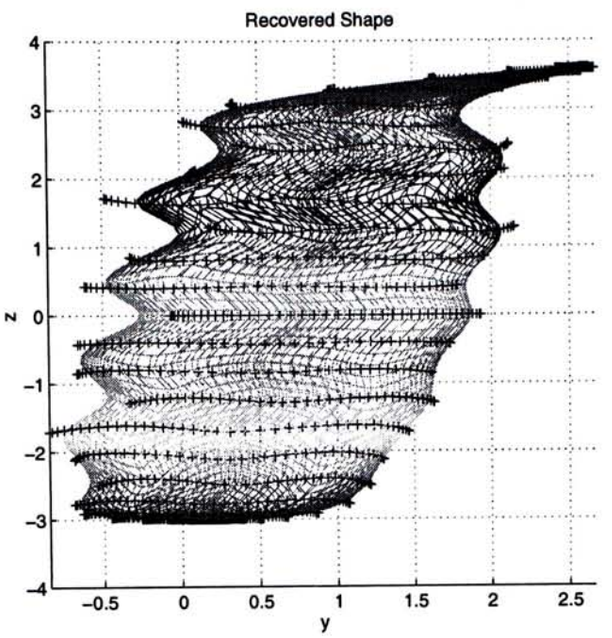


Figure 5.13: Side view of the recovered shape of test object 1.

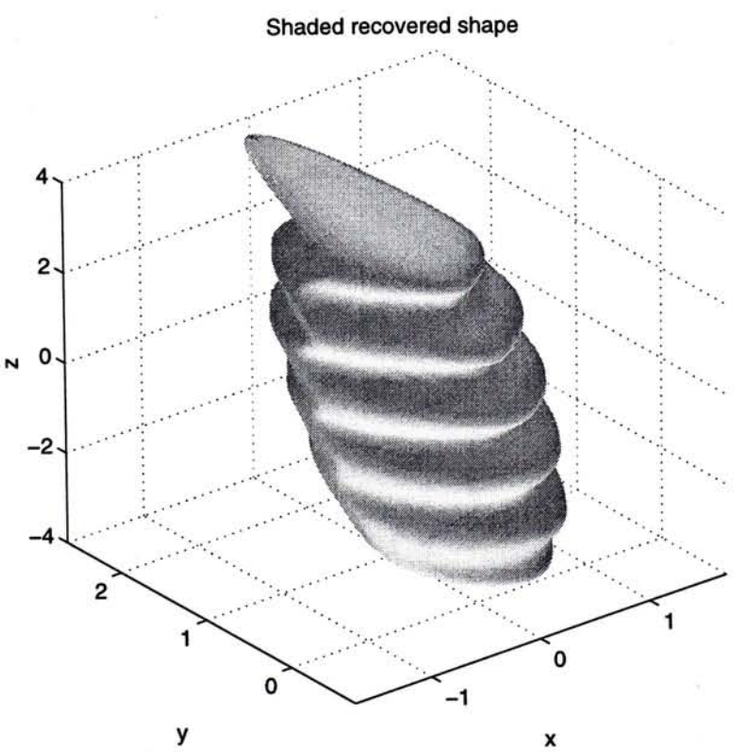


Figure 5.14: Shaded recovered shape of test object 1.

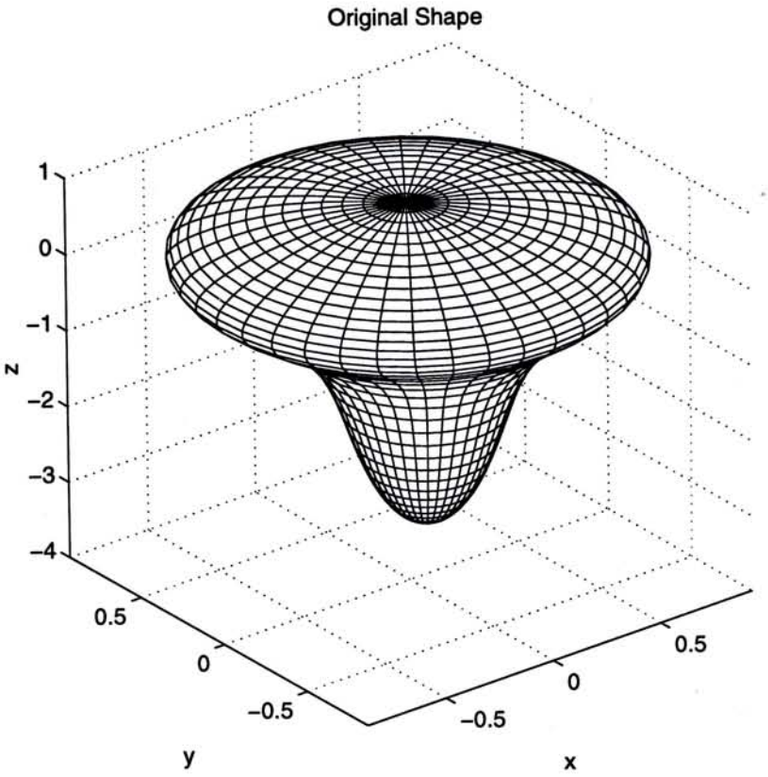


Figure 5.15: Test object 2

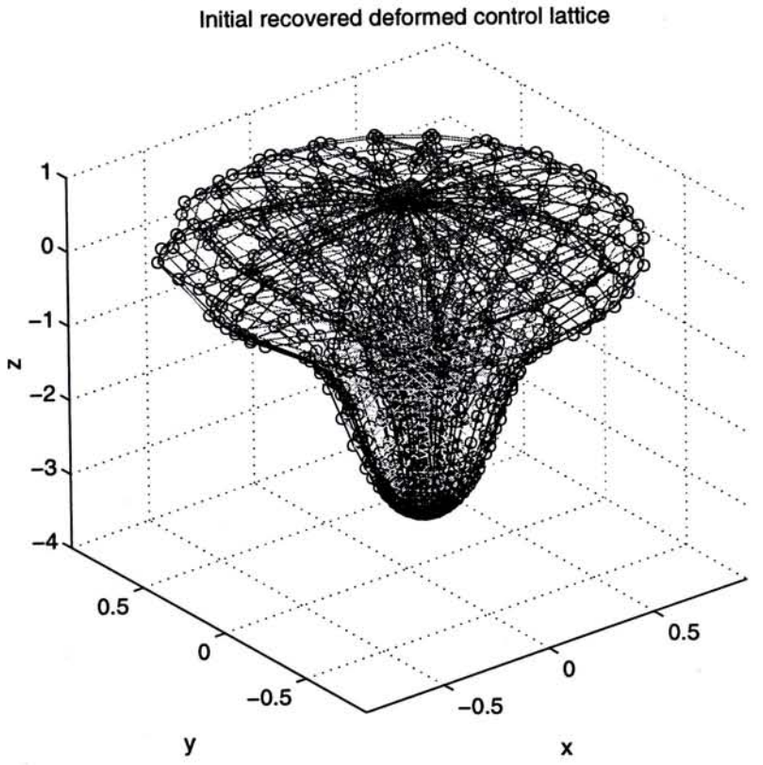


Figure 5.16: Recovered control lattice of test object 2.

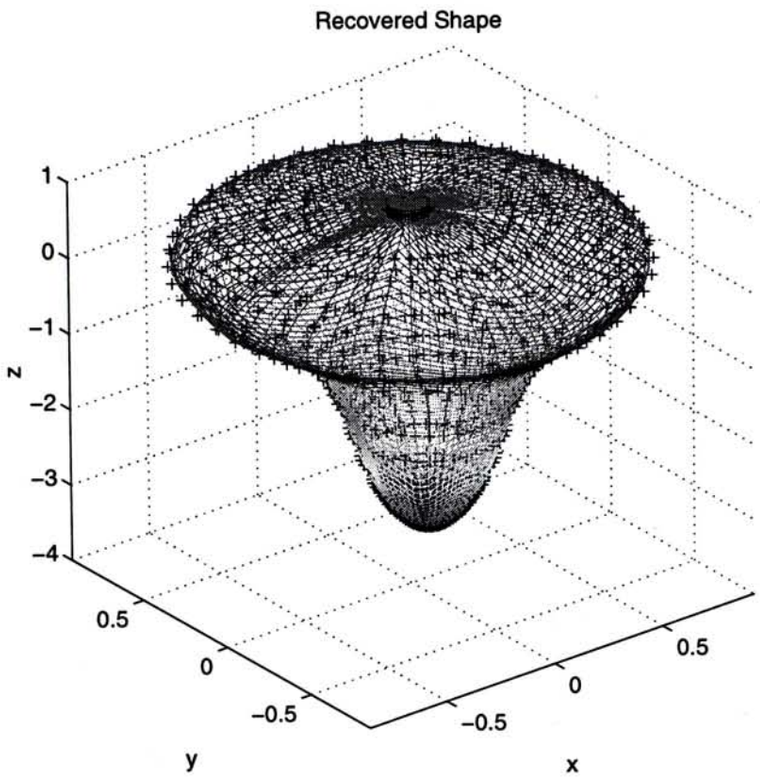


Figure 5.17: The recovered shape of test object 2.

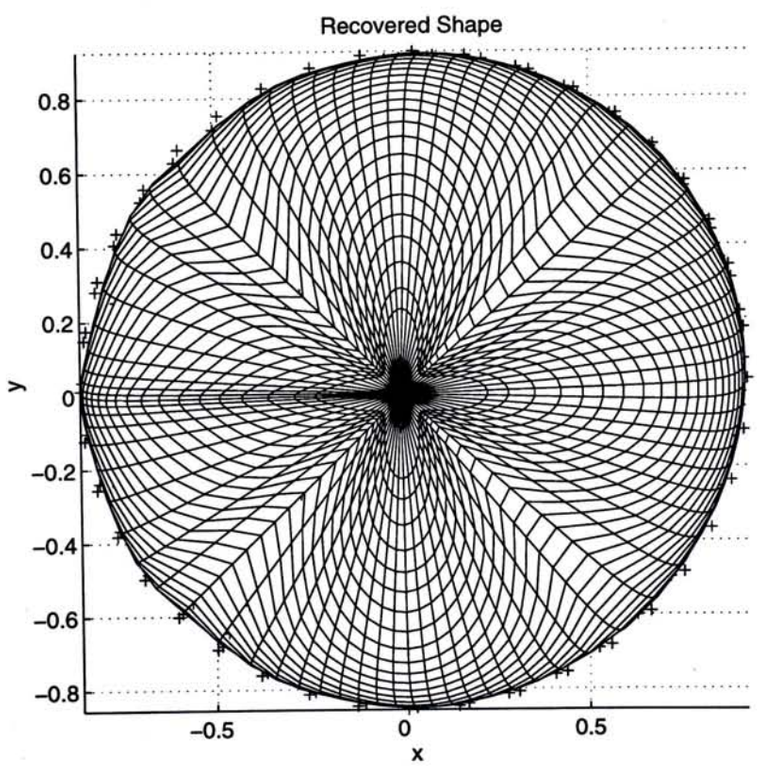


Figure 5.18: Top view of the recovered shape of test object 2.

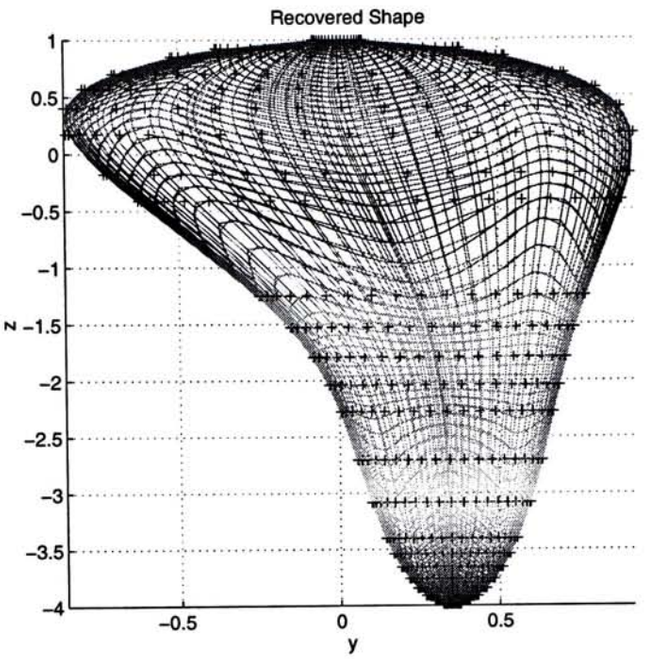


Figure 5.19: Side view of the recovered shape of test object 2.

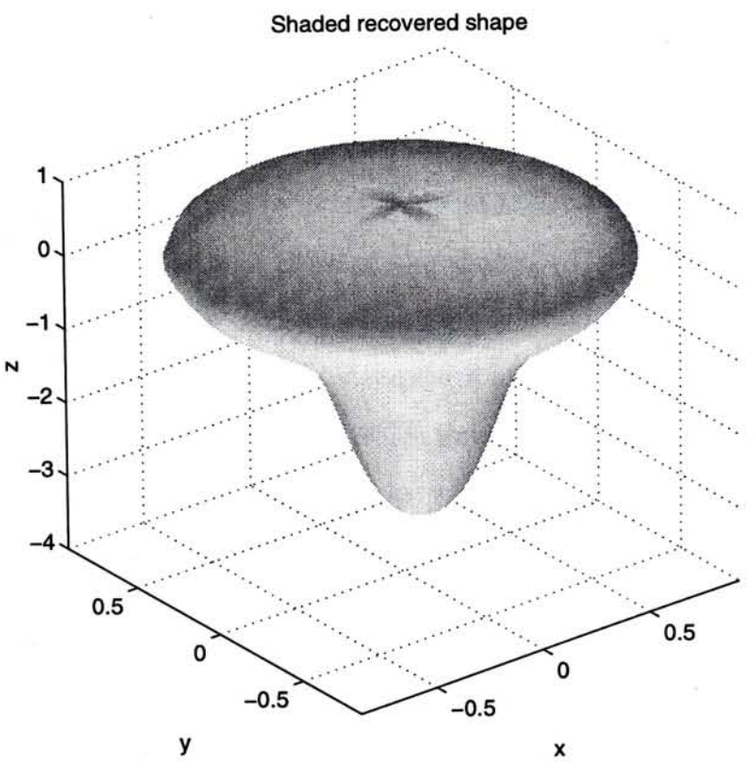


Figure 5.20: Shaded recovered shape of test object 2.

Chapter 6

Conclusion

In this dissertation, the problem of 3D model reconstruction by active haptic exploration alone is investigated.

A geometric model for model representation is proposed in this dissertation. The geometric model includes two components. The first component is a superquadric model for general shape description. The second component enhances the superquadric model by fine tuning its shape by a technique called Free Form Deformation or FFD. This is a very flexible model. Using B-spline basis in description of trivariate parametric solids in FFD can even improve the control of the shape.

An active sensing strategy of contour tracing is proposed for 3D coordinates of object surface points acquisition. This includes a Exploratory Procedure, proposed in this thesis, which guide the robotic hand with tactile sensor array mounted on its fingertip where to begin a new tracing on the object surface. The separation between adjacent contours depends on curvature κ and change in curvature $\triangle\kappa$ along the search direction. Moreover, an algorithm is also proposed for 3D model reconstruction from data acquired by tactile sensor data.

6.1 Future Work

This dissertation can be further extended in the following aspects,

- Develop and implement an efficient contour following algorithm for a multi-fingered dextrous robotic hand with tactile sensor arrays mounted on fingertips. In this dissertation, I assume that a contour following algorithm is developed and implemented in the haptic exploration system. With the aid of an efficient contour tracing strategy, the shape reconstruction process will be facilitated.

- Speed up the computation of B-spline basis and its derivatives evaluation by developing parallel algorithm for the evaluation. The B-spline basis and its derivatives need to be computed frequently in B-spline based free form deformation (BFFD), surface fitting in tactile sensor data, shape recovery by least square error method, etc. If this frequent evaluation is speeded up, the whole shape reconstruction process can be much efficient. One possible remedy is to construct pipelined architecture for B-spline basis and its derivatives computation implementation [58].

- Setup an active haptic exploration testbed for testing algorithms and experiments. Due to the incomplete of experimental setup, experimental results are not presented in this dissertation. The planned setup includes a dextrous hand with five fingers. Each finger will be equipped with a 16×16 ultrasonic tactile sensor array on fingertip. The hand system is mounted on a redundant robotic arm with 7 degree of freedom. There are two techniques for acquisition of object model data standards for comparison. The first one is to generate each test object by a Rapid Prototyping System (RPS) which generate 3D model prototypes rapidly and accurately according to the model data fed into the system. We can compare the original shape

data fed into the RPS with the reconstructed shape recovered from the proposed algorithm with active haptic exploration. The other technique is to gather object surface information from the Coordinates Measuring Machine (CMM) first. Then, we can reconstruct the object shape by active haptic exploration and compare the recovered shape with the object shape measured by the CMM.

- Improve the level of robustness of derivatives estimation of tactile sensor data for normal vector and curvature evaluation. As the higher order information, like surface curvature is highly susceptible to noise, robust estimation of derivatives should be achieved. One possible improvement is to apply Extended Kalman filter [59] to optimally estimate noise-suppressed sensor data points (by Adaptive Wiener Filter and Median Filter discussed in section 3.4.2 on the tactile sensor data surface. Then, the derivatives of the sensor data surface can be derived by some numerical differentiation techniques.

- In this dissertation, the test object is assumed to be fixed in a proper orientation and position on a platform. This unrealistic assumption can be given up by incorporating the position and orientation determination function into the original superquadric model determination stage of the haptic exploration procedure (see section 3.2.2) using the Solina's algorithm described in section 2.3. After the Euler angles and the object position vector are found, the transformation matrix $\mathbf{T}$ describing the test object position and orientation with respect to the base frame can be obtained. With the affine invariance property of B-spline, FFD and inverse FFD can be performed in any space under affine transformation, like $\mathbf{T}$. The data points are first transformed into the base coordinate frame with $\mathbf{T}$ before operations. After operations, they are then transformed back

into their original frame.

- Improve the step size determination module so that it can handle a large variety of shape. This can be implemented by a multi-layered neural network with curvature κ and change in curvature $\triangle\kappa$ as inputs and contour step size s as output. Moreover, it can also be implemented by fuzzy logic techniques. A set of membership functions that describe the different ranges of fuzzy inputs, curvature and change in curvature, with a set of linguistic variables, like SMALL, MEDIUM, LARGE and so. In order to have reasonable results, a large set of training data and testing data should be generated beforehand in the two approaches.

- In this dissertation a separable step size function model was considered. This means that the dependence of κ and $\triangle\kappa$ in the step size function are independently. For instances,

$$\begin{aligned} s(\kappa, \triangle\kappa) &= \mathbf{e}^{\lambda\kappa+\mu\triangle\kappa} \\ s(\kappa, \triangle\kappa) &= \frac{\lambda}{\ln(\alpha\kappa + \beta\triangle\kappa)} \\ s(\kappa, \triangle\kappa) &= \frac{\lambda}{(\alpha\kappa + \beta\triangle\kappa)^{\mu}} \end{aligned}$$

  where λ, μ, α and β are constants. In order to extend the model, non-separable step size function model will be investigated. The effects of non-separable step size function are also investigated.

- Analysis of the effect of the function templates of the curvature and change in curvature employed in step size determination for contour tracing in model reconstruction may be conducted. This analysis includes finding the function template(s) that best fit the model reconstruction of different kinds of objects, like objects with sharp changes on their surfaces or objects with gradual changes or even on

changes.

- A shape refinement algorithm would be developed to further improve the likeness between the recovered shape from the heuristic model reconstruction algorithm proposed in section 4.4 and the object shape sensed by the haptic exploration system. This can be achieved by advanced techniques of shape control of splines on the control lattice recovered, including control points repositioning and weight modification for NURBS based FFD in model construction [37][60][61].

- The recovered shape may not be the same if different contour search directions, including directions parallel to or different with the same search axis, are employed in the haptic exploration procedures for sensor data acquisition. This can be remedied by the recovery of the major axis of the test object in the modified Grasp by Containment stage (see section 3.2.2) using Principal Component Analysis techniques [62] in the sparse data points acquired. The contour search direction is then set to be parallel to the recovered major axis. The recovered shapes should be consistent when it is haptically explored by the system.

Appendix

All source codes written for this project are attached in this section.

formlattice.m

```
function [P,ngrids1,px,py,pz,ax1,ay1,az1]=form_lattice(dx,dy,dz,k,m,n,sm,tol)
% P=form_lattice(dx,dy,dz,k,m,n,sm,tol)
%
% Function to form the deformed control lattice from data pts [dx,dy,dz]
% [dx,dy,dz] are matrices with each row represents a contour of data pts.
% [m,n]=dimension of the control layer.
% sm indicates the Search Method.
% sm=1 for Fibonacci search, sm=0 for Golden Section Search
% tol is the error tolerance for the search
% k is the order of B-spline in closed curving fitting
%
% call buildgrid.m

[mm,nn]=size(dx);
if (size(dx)~=size(dy)) | (size(dy)~=size(dz)) | (size(dz)~=size(dx)),
  error('Dimensions of data points array NOT match!');
end;
P=[];
disp('Finding control polygon for each contour ...');
for i=1:mm,
  dzz=mean(dz(i,:)).*ones(size(dz(i,:)));
    [cpgx,cpgy,cpgz,cpx,cpy,cpz,knots,t,count]= ...
    buildgrid(dx(i,:),dy(i,:),dzz,k,m,n,sm,tol,1,0);
  P=[P; [cpgx(:) cpgy(:) cpgz(:)]];
  disp(sprintf('Contour #%d has been processed.',i));
end;
ngrids1=[m-1,n-1,mm-1];

disp('Reconstruction begins ...');
kk=[4 4 4];
knotx=openknot(kk(1),ngrids1(1));
knoty=openknot(kk(2),ngrids1(2));
knotz=openknot(kk(3),ngrids1(3));

[px,py,pz]=newfparam(dx,dy,dz,[0.15 0.15],kk,ngrids1,mm,nn);

disp('Fitting the (s,t,u) parameter space ...');
[pfitx,pfity,pfitz,ku,kv,tu,tv]=surffit(px,py,pz,4);
save pfit1 pfitx pfity pfitz ku kv tu tv;

num=50;
vv=linspace(0,1,num);  uu=vv;
Nv=spcol(kv,4,vv);  Nu=spcol(ku,4,uu);
ppx=Nv*pfitx*Nu';
ppy=Nv*pfity*Nu';
ppz=Nv*pfitz*Nu';
% To make sure the parameter computed in inside the range of [0,1]
```

```
% i1=find(ppx>1);  i2=find(ppx<0);
% ppx=ppx(:);  ppx(i1)=ones(length(i1),1);  ppx(i2)=zeros(length(i2),1);
% ppx=reshape(ppx,num,num); i1=find(ppy>1);  i2=find(ppy<0);
% ppy=ppy(:);  ppy(i1)=ones(length(i1),1);  ppy(i2)=zeros(length(i2),1);
% ppy=reshape(ppy,num,num); i1=find(ppz>1);  i2=find(ppz<0);
% ppz=ppz(:);  ppz(i1)=ones(length(i1),1);  ppz(i2)=zeros(length(i2),1);
% ppz=reshape(ppz,num,num);

ppx=min(ppx,1);  ppx=max(ppx,0);
ppy=min(ppy,1);  ppy=max(ppy,0);
ppz=min(ppz,1);  ppz=max(ppz,0);

disp('Rebuilding the object model ...');
for i=1:size(ppx,1),
  [axt,ayt,azt]=fnffd1(ppx(i,:),ppy(i,:),ppz(i,:), ...
                       ngrids1,P,knotx,knoty,knotz,kk);
  ax1(i,:)=axt;  ay1(i,:)=ayt;  az1(i,:)=azt;
end;

top=mean([ax1(1,:)' ay1(1,:)']);
bot=mean([ax1(size(ax1,1),:)' ay1(size(ay1,1),:)']);
minss=0.02; % Min. step size
ax1=[top(1).*ones(1,size(ax1,2)); ax1; bot(1).*ones(1,size(ax1,2))];
ay1=[top(2).*ones(1,size(ay1,2)); ay1; bot(2).*ones(1,size(ay1,2))];
az1=[mean(az1(1,:)).*ones(1,size(az1,2))-1/2*minss; az1; ...
     mean(az1(size(az1,1),:)).*ones(1,size(az1,2))+1/2*minss];

figure;
plot_lattice(P,ngrids1);
axis square equal;
xlabel('x');  ylabel('y');  zlabel('z');
title('Initial recovered deformed control lattice');

figure;
mesh(ax1,ay1,az1);
axis square equal;
xlabel('x');  ylabel('y');  zlabel('z');
title('Recovered Shape');
```

getpo1.m

```
function [cpx,cpy,cpz,NN1,knots,t,err,count]=getpo1(cx,cy,cz,n,k,tol,g)
%
% Get the closed control polygon from the data pts
% obtained from a closed contour (not self-intersecting).
%
% Usage: [cpx,cpy,cpz,NN,knots,t,err,count]=getpo1(cx,cy,cz,n,k,tol)
% where [cpx,cpy,cpz] are the coordinates of the control polygon.
%       [cx,cy,cz] are the coordinates of sample pts. Each is a row vector
%       n is the number of control vertices needed.
%       (Only define non-repeated vertices sequence, like B1, ...., Bn.
%       k is the order of the B-spline basis function used (3 or 4)
%       NN is the basis function matrix.
% knots is the knot vector used
% err is the err. % repeating vertices & their corr. pseudo-vertices
% t is the parameter values for [cx cy cz]
% count is the no. of iteration used.
% if g==1, graphic output, else no graphics.
%
% Use open uniform knot vector
%
% n < N for reasonable speed and shape of control polygon.
%
% Reference: D. F. Rogers and J. A. Adams, Mathematical Elements for Computer
%            Graphics, 2nd edition, McGraw Hill, pp.332-336, 346-351
%
```

```
% In addition, 1st and 2nd derivatives continuity constraints at the junction
% between the first segment and the last segment of the closed curve are also
% added to the system.  The minimum order is 3.
% Parameters optimization by "Constrained B-spline curve and surface fitting"
% by Rogers, D.F. and Fog, N.G., Computer Aided Design 21, pp. 641-648.

N=length(cx);
n1=n+k-1;
maxiterations=300;
if (nargin<6),  tol=0.05;  end;
knots=augknt(linspace(0,1,n1-k+2),k);

% chord length parametrization
dx=diff(cx);  dy=diff(cy);  dz=diff(cz);
dl=sqrt(dx.*dx+dy.*dy+dz.*dz);
sumdl=sum(dl);
t=zeros(1,length(dl)+1);
for j=2:length(dl)+1
  t(j)=dl(j-1)+t(j-1);
end
t=t./sumdl;

%%% knot vector -- interior knot by averaging parameter values
%%% Bad result for closed curve
% knots=zeros(1,n1+k);
% knots(n1+1:n1+k)=ones(1,k);
%%% Sample the data pt parameter value for averaging for interior knots
% ind=floor(linspace(1,N,n1));
% ik=t(ind);
% for i=1:n1-k,
%   knots(k+i)=mean(ik(i:i+k-1));
% end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ratio=t;
err=100;
dt=zeros(size(t));
count=0;
NN=[];
while ((err > tol) & (count < maxiterations))
  t1=t+dt;
  index=find(t1<0);
  t1(index)=t(index);
  index=find(t1>1);
  t1(index)=t(index);
  index=find(diff(t1)<0); % avoid decreasing parameter
  % t1(index)=t(index);
  if (any(index)~=0),
    if (index(length(index))<length(t)-1),
      t1(index+1)=mean([t1(index); t1(index+2)]);
    else
      ind=index(1:length(index)-1);
      t1(ind+1)=mean([t1(ind)' t1(ind+2)']);
      t1(index(length(index)))=mean([t1(index(length(index))+1); 1]);
    end;
  end;
  t=t1;
  tt=repeat(t,3);
  NN=spcol(knots,k,tt);
  appendNN=zeros(k-1+2,size(NN,2));
  for i=1:k-1
    appendNN(i,i)=1;  appendNN(i,n+i)=-1;
  end;
  appendNN(k:k+1,:)=[NN(2,:)-NN(3*N-1,:); NN(3,:)-NN(3*N,:)];

  index=[1:3:3*N];
  NN1=NN(index,:);
  index=[2:3:3*N];
  NNt=NN(index,:);
  index=[3:3:3*N];
  NNtt=NN(index,:);
  tempNN=NN1;
```

```
  NN1=[NN1;appendNN];
  appenddata=zeros(k-1+2,3);
  datapt=[cx' cy' cz'; appenddata];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  cpolygon=pinv(NN1,1e-10)*datapt;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  cpx=cpolygon(:,1)';
  cpy=cpolygon(:,2)';
  cpz=cpolygon(:,3)';

  cpolygon=[[cpx(1:n)'; cpx(1:k-1)'] ...
  [cpy(1:n)'; cpy(1:k-1)'] [cpz(1:n)';   cpz(1:k-1)']];
  P=tempNN*cpolygon;
  px=P(:,1)';  py=P(:,2)';  pz=P(:,3)';

  % Calculating delta t
  Data=[cx;cy;cz;];
  Pt=NNt*cpolygon;
  Ptt=NNtt*cpolygon;
  ERR=Data-P';


  % Parameter Optimization by Josef Hoschek
  nPt=sqrt(dot(Pt',Pt'));
  nPt=nPt([1 1 1],:);

  % tempcp1=cpolygon';
  % tempcp1=tempcp1(:,1:n);
  % tempcp2=[tempcp1(:,2:n) tempcp1(:,1)];
  % dtempcp=tempcp2-tempcp1;
  % const=1/sum(dot(dtempcp,dtempcp));
  % dt=dot(ERR,Pt'./nPt).*const;
  % dt=dot(ERR,Pt')./sqrt(sum(Pt'.*Pt')).*const;

  % Parameter Optimization by Roger and Fog.
  dt=dot(ERR,Pt')./sum(Pt'.*Pt')./2;

  % Another Parameter Optimization algorithm by Josef Hoschek
  % May be wrong
  % dt=dot(ERR,Pt')./(dot(ERR,Ptt')+dot(Pt',Pt'));

  ex=cpx(1:k-1)-cpx(n+1:n+k-1);
  ey=cpy(1:k-1)-cpy(n+1:n+k-1);
  ez=cpz(1:k-1)-cpz(n+1:n+k-1);
  err=sqrt(sum(ex.*ex)+sum(ey.*ey)+sum(ez.*ez));
  count=count+1;
end

if (g==1)
figure;
plot3([cpx(1:n) cpx(1)],[cpy(1:n) cpy(1)],[cpz(1:n) cpz(1)]);  hold on;
plot3([cpx(1:n) cpx(1)],[cpy(1:n) cpy(1)],[cpz(1:n) cpz(1)],'rx');
plot3(cpx(1:3),cpy(1:3),cpz(1:3),'ro');
plot3(cpx(n),cpy(n),cpz(n),'g*');
plot3(cx,cy,cz,'b');
axis square equal
view(0,90);
hold off;
title(sprintf('n=%d, k=%d', n, k));
end;

cpolygon=[[cpx(1:n)'; cpx(1:k-1)'] [cpy(1:n)'; cpy(1:k-1)'] [cpz(1:n)'; cpz(1:k-1)']];
P=tempNN*cpolygon;
px=P(:,1)';  py=P(:,2)';  pz=P(:,3)';

if (g==1),
figure; hold on;
```

```
plot3([px px(1)],[py py(1)],[pz pz(1)],'g');
% plot3([cx cx(1)],[cy cy(1)],[cz cz(1)],'r');
% plot3(cx(1:10),cy(1:10),cz(1:10),'b');
plot3(cx(1),cy(1),cz(1),'yo');
axis equal square;
view(0,90);
hold off;
title(sprintf('Recovered curve, n=%d, k=%d',n,k));
end;

ex=cpx(1:k-1)-cpx(n+1:n+k-1);
ey=cpy(1:k-1)-cpy(n+1:n+k-1);
ez=cpz(1:k-1)-cpz(n+1:n+k-1);
err=sqrt(sum(ex.*ex)+sum(ey.*ey)+sum(ez.*ez));

disp(sprintf('The error=%f',err));
disp(sprintf('no. of iterations=%d',count));

ex=ex./cpx(1:k-1)*100;
ey=ey./cpy(1:k-1)*100;
ez=ez./cpz(1:k-1)*100;
err=[err ex ey ez];

disp(['The percentage error of x-coordinates of pseudo-vertices are ',mat2str(ex,5)]);
disp(['The percentage error of y-coordinates of pseudo-vertices are ',mat2str(ey,5)]);
disp(['The percentage error of z-coordinates of pseudo-vertices are ',mat2str(ez,5)]);

cpx=cpx(1:n);  cpy=cpy(1:n);  cpz=cpz(1:n);

if (g==1),
figure(gcf-1);
hold on;
scpx=sort(cpx);  scpy=sort(cpy);
xleft=scpx(1);  xright=scpx(n);
ytop=scpy(n);  ybottom=scpy(1);
xx=[xleft xright xright xleft xleft];
yy=[ytop ytop ybottom ybottom ytop];
zz=cpz(1:5);
plot3(xx,yy,zz,'y');
end;
```

buildgrid.m

```
function [cpgx,cpgy,cpgz,cpx,cpy,cpz,knots,t,count]=buildgrid(cx,cy,cz,k,m,n,sm,tol,layer,g)

% [cpgx,cpgy,cpgz]=buildgrid(cx,cy,cz,k,m,n,sm,tol)
%
% Function to compute 1 control layer from data pts [cx cy cz]
% This includes the boundary of the grid (the control polygon recovered
% from [cx cy cz]) and the inner grid (calculated by meshgrid the vertices
% of the largest rectangle that wholly inscribed inside the control polygon.
%
% [cpgx,cpgy,cpgz] is the coordinates of 1 control layer grid.
% layer is a complex parameter.  It has size of 1 or 3.
% if size(layer)=[1 1],  first layer and layer=1, the layer index
% if size(layer)=[1 3],  layer=[layer_index pivot],where pivot=[px,py] is
% the coordinates of the pivot vertex
% [m,n] is the dimension of the control layer grid.
% sm indicates the Search Method.
% sm=1 Fibonacci Search, sm=0 Golden Section Search
% tol is the tolerance for the search
% if g==1, graphics output, else no graphics.
%
% call getpo1.m, isint.m, phasediff.m

if (nargin<9),  layer=1;  end; % Default is for the first layer
if (nargin<8),  tol=1e-6;  end;
```

```
if (nargin<7),  sm=1;  end;
if (nargin<6),  n=m;  end;
if (nargin<5),  error('Too few input parameters !!!');  end;

tol1=mean([abs(max(cx)-min(cx)),abs(max(cy)-min(cy))])*0.02;
nn=2*(m+n-2);
[cpx,cpy,cpz,NN,knots,t,err,count]=getpo1(cx,cy,cz,nn,k,tol1,0);

scpx=sort(cpx);  scpy=sort(cpy);
xleft=scpx(1);  xright=scpx(nn);
ytop=scpy(nn);  ybottom=scpy(1);

% Coordinates of the bounding box
xx=[xleft xright xright xleft xleft];
yy=[ytop ytop ybottom ybottom ytop];

% Center of the bounding box
center=1/2*[(xleft+xright) (ytop+ybottom)];

% size of the bounding box
l=xright-xleft;  w=ytop-ybottom;

maxit=50; % Max iterations
if (sm==1),
  % Generate the Fibnacci sequence
  fib=[1 1];
  for i=3:maxit,
    fib=[fib fib(i-1)+fib(i-2)];
  end;
end;

% Test whether the conic is inside the control polygon
% if the conic is wholly inside the control polygon, all line segemnts formed
% by consecutive control vertices lie outside the conic.
% ie. All the straight lines of the line segments do not have any intersection
% with the conic.

dcpx=diff([cpx cpx(1)]);  dcpy=diff([cpy cpy(1)]);
a=tol; b=1;
count=1;
while(~((count>maxit-1) | (abs(b-a)<=tol)))
  temp=[a b];
  y1=[];  y2=[];
  for i=1:n,
    y1=[y1 isint([dcpy(i) dcpx(i) dot([cpy(i) -cpx(i)]', ...
                  [dcpx(i) dcpy(i)]')],1/2*a*[l w],center)];
    y2=[y2 isint([dcpy(i) dcpx(i) dot([cpy(i) -cpx(i)]', ...
                  [dcpx(i) dcpy(i)]')],1/2*b*[l w],center)];
  end;
  y1=sum(y1);  y2=sum(y2);
  if (sm==1),% Fibonacci Search
    a1=a+fib(maxit-count-1)/fib(maxit-count+1)*(b-a);
    b1=a+fib(maxit-count)/fib(maxit-count+1)*(b-a);
  else % Golden Section Search
    a1=b-0.618*(b-a);
    b1=a+0.618*(b-a);
  end;

  % Case(i): if both ellipse cross the control polygon
  % C1 and C2 are outside or crossof the control polygon
  if ((y1>0) & (y2>0)),  a=temp(1);  b=a1;  end;

  % Case(ii): if C2 cross the control polygon while C1 does not
  % C1 is inside the control polygon while C2 crosses or is outside it
  if ((y1==0) & (y2>0)), a=a1;  b=b1;  end;

  % Case(iii): if both C1 & C2 do not cross the control polygon
  % C1 and C2 are inside the control polygon
  if ((y1==0) & (y2==0)),  a=b1;  b=temp(2);  end;

  % Case(iv): if C1 crosses the control polygon, while C2 does not
```

```
  % C1 crosses or outside  while C2 is inside the control polygon
  % Contradiction since size(C1)<size(C2)
  %if ((y1>0) & (y2==0)),  a=a1;  b=b1;  end;

  count=count+1;
end;
lam=0.5*(a+b);

% The new dimension of the bounding box becomes
l=lam*l*0.95;  w=lam*w*0.95;

% The 4 corners of the inner grid
xleft1=center(1)-l/2;  xright1=center(1)+l/2;
ytop1=center(2)+w/2;  ybottom1=center(2)-w/2;

% Build inner grid
xx=linspace(xleft1,xright1,n-2);
yy=linspace(ybottom1,ytop1,m-2);
[xx,yy]=meshgrid(xx,yy);
zz=cpz(1)*ones(size(xx));

% Reorder the control vertices of the control polygon
if (size(layer==1)==[1 1])
  pivot=[xleft ytop];
else
  pivot=layer(2:3); % other layer
end
cen=center(ones(1,length(cpx)),:)';
[mdphase,index]=min(abs(phasediff([cpx;cpy]-cen,...
pivot(ones(1,length(cpx)),:)'-cen)));
tcpx=cpx(1);  tcpy=cpy(1);
if (index>1),% if index==1, no need for reordering
  cpx=cpx([index:nn,1:index-1]);
  cpy=cpy([index:nn,1:index-1]);
  cpz=cpz([index:nn,1:index-1]);
end

% Reorder the boundary vertices and inner grid vertices
% into a control layer grid
% boundary vertices index
indexb=[1:m:(n-1)*m+1,(n-1)*m+2:m*n,(n-1)*m:-m:m,(m-1):-1:2];
cpgx(indexb)=cpx;  cpgy(indexb)=cpy;  cpgz(indexb)=cpz;

% inner grid index
indexg=[1:m*n];
indexg=reshape(indexg,m,n)';
if (abs(atan2(tcpy,tcpx))>pi/2),
  indexg=rot90(indexg(2:n-1,2:m-1)',1); % <-------------
else
  indexg=rot90(indexg(2:n-1,2:m-1),1);
end;
indexg=indexg(:);
cpgx(indexg)=xx(:);  cpgy(indexg)=yy(:);  cpgz(indexg)=zz(:);
cpgx=reshape(cpgx,m,n);
cpgy=reshape(cpgy,m,n);
cpgz=reshape(cpgz,m,n);

if (g==1),
% Plot the resultant control layer
figure;
mesh(cpgx,cpgy,cpgz);  axis square equal;  grid on;
hold on;
xlabel('x');  ylabel('y');  zlabel('z');
title('One recovered control layer');
hidden off;
plot3(cpx,cpy,cpz,'ro');
plot3(cpx(1:3),cpy(1:3),cpz(1:3),'g*');
plot3(cpx(nn),cpy(nn),cpz(nn),'bo',cpx(nn),cpy(nn),cpz(nn),'yx');
% set(gca,'ZLim',[500 800],'YTick',[-20:10:20],'XTick',[480:10:520]);
% view(0,90);
```

```
end;
```

isint.m

```
function y=isint(l,d,center)

% y=isint(l,d,center)
% Test whether a straight line defined by l intersects with
% the conic defined by c with centre at centre
%
% l=[a b c], d=[p q], center=[xc yc];
% Straight line equation : ax+by+c=0,
% Conic equation : p^2(x-xc)^2+q^2(y-yc)^2-(pq)^2=0;
% y=1 if they have intersection else y=0

a=l(1);  b=l(2);  c=l(3);
p=d(1);  q=d(2);
xc=center(1);  yc=center(2);

if (a==0),
  k=-c/b;
  if (q < abs(k-yc))
    y=0
  else
    y=1;
  end
elseif (b==0),
  k=-c/a;
  if (p < abs(k-xc))
    y=0;
  else
    y=1;
  end
else
  c=(a*xc+b*yc+c)^2;
  if ((q^2*b^2+p^2*a^2) < c)
    y=0;
  else
    y=1;
  end
end
```

phasediff.m

```
function dtheta=phasediff(v1,v2)

% Calculate the phase angle difference between 2D vectors v1 and v2.
% Use rotation to model phase difference
%
% The mathematical model is as follows,
%
% [v2_x v2_y]=[v1_x v1_y]*[cos(a) sin(a);-sin(a) cos(a)];
%
% =>[v2_x v2_y]^T=[v1_x -v1_y;v1_y v1_x]*[cos(a) sin(a)]^T
% =>v2   =       v1_n     *      A
% =>A=inv(v1_n)*v2
%
% If v1 and v2 contains several sets of vectors and are arranged as
% v1=[v11 | v12 | v13 | ......],
% First concatenate each vectors in v2 as a column vector.
% Arrage each v1i_n at the diagonal of the resultant matrix v1_n
% then, A=[cos(a1) sin(a1) cos(a2) sin(a2) ......]^T

N=size(v1,2);
if (size(v1,2)~=size(v2,2))
  N=min(size(v1,2),size(v2,2));
  v1=v1(:,1:N);  v2=v2(:,1:N);
end;
```

```
% Make v2 as column vector
v2=v2(:);
v1n=zeros(2*N,2*N);
for i=1:N,
  temp=[v1(1,i) -v1(2,i);v1(2,i) v1(1,i)];
  v1n(2*(i-1)+1:2*i,2*(i-1)+1:2*i)=temp;
end;
A=inv(v1n)*v2;
B=A(2:2:2*N)';
A=A(1:2:2*N-1)';
dtheta=atan2(B,A);
```

fnffd1.m

```
function [x,y,z]=fnffd1(px,py,pz,ngrids,pnew,knotx,knoty,knotz,k)
[m,n]=size(px);
x=zeros(m,n);
y=x;  z=x;
nx=ngrids(1);  ny=ngrids(2);  nz=ngrids(3);
% t0=cputime;

for iy=1:n
  bx=spcol1(knotx,k(1),px(:,iy)');
  by=spcol1(knoty,k(2),py(:,iy)');
  bz=spcol1(knotz,k(3),pz(:,iy)');
%for iy=1:n
  for ix=1:m
    %index=ix*(n-1)+iy;
    tx=repeatp(bx(ix,:),(ny+1)*(nz+1));
    ty=repeatp(repeat(by(ix,:),nx+1),nz+1);
    tz=repeat(bz(ix,:),(nx+1)*(ny+1));
    % scale=h'.*tx.*ty.*tz;
    % den=sum(scale);
    x(ix,iy)=sum(tx.*ty.*tz.*pnew(:,1)');
    y(ix,iy)=sum(tx.*ty.*tz.*pnew(:,2)');
    z(ix,iy)=sum(tx.*ty.*tz.*pnew(:,3)');
  end
end
% disp(sprintf('CPU Time for fnffd.m=%f',cputime-t0));
```

findp.m

```
function [px,py,pz,r,count]=findp(p,knotx,knoty,knotz,k,ngrids,CL,seed,tol)
% Point Inversion Problem
% Solve (s,t,u) for the point p.
%
% [px,py,pz]=findp(p,knotx,knoty,knotz,k,ngrids,CL,seed)
% CL = control lattice
% seed = initial guess
% parameter range [0,1] => seed=[0.5 0.5 0.5]

maxit=50;
count=0;
flag=0;
pp=seed;
nx=ngrids(1);  ny=ngrids(2);  nz=ngrids(3);
if (nargin<9),
  e1=1e-6;  e2=1e-6;
else
  e1=tol(1); e2=tol(2);
end;
while ((count < maxit) & (flag==0))
  tx=spcol(knotx,k(1),pp(1)*ones(1,3));
  ty=spcol(knoty,k(2),pp(2)*ones(1,3));
  tz=spcol(knotz,k(3),pp(3)*ones(1,3));
```

```
s=repeatp(tx(1,:),prod([ny nz]+1));
t=repeatp(repeat(ty(1,:),nx+1),nz+1);
u=repeat(tz(1,:),prod([nx ny]+1));

ds=repeatp(tx(2,:),prod([ny nz]+1));
dt=repeatp(repeat(ty(2,:),nx+1),nz+1);
du=repeat(tz(2,:),prod([nx ny]+1));

dss=repeatp(tx(3,:),prod([ny nz]+1));
dtt=repeatp(repeat(ty(3,:),nx+1),nz+1);
duu=repeat(tz(3,:),prod([nx ny]+1));

% X
tm=s.*t.*u;
x=sum(tm.*CL(:,1)');  y=sum(tm.*CL(:,2)'); z=sum(tm.*CL(:,3)');

% Xs
tm=ds.*t.*u;
dsx=sum(tm.*CL(:,1)');  dsy=sum(tm.*CL(:,2)');  dsz=sum(tm.*CL(:,3)');

% Xt
tm=s.*dt.*u;
dtx=sum(tm.*CL(:,1)');  dty=sum(tm.*CL(:,2)');  dtz=sum(tm.*CL(:,3)');

% Xu
tm=s.*t.*du;
dux=sum(tm.*CL(:,1)');  duy=sum(tm.*CL(:,2)');  duz=sum(tm.*CL(:,3)');

% Xss
tm=dss.*t.*u;
dssx=sum(tm.*CL(:,1)');  dssy=sum(tm.*CL(:,2)');  dssz=sum(tm.*CL(:,3)');

% Xtt
tm=s.*dtt.*u;
dttx=sum(tm.*CL(:,1)');  dtty=sum(tm.*CL(:,2)');  dttz=sum(tm.*CL(:,3)');

% Xuu
tm=s.*t.*duu;
duux=sum(tm.*CL(:,1)');  duuy=sum(tm.*CL(:,2)');  duuz=sum(tm.*CL(:,3)');

% Xst
tm=ds.*dt.*u;
dstx=sum(tm.*CL(:,1)');  dsty=sum(tm.*CL(:,2)');  dstz=sum(tm.*CL(:,3)');

% Xsu
tm=ds.*t.*du;
dsux=sum(tm.*CL(:,1)');  dsuy=sum(tm.*CL(:,2)');  dsuz=sum(tm.*CL(:,3)');

% Xtu
tm=s.*dt.*du;
dtux=sum(tm.*CL(:,1)');  dtuy=sum(tm.*CL(:,2)');  dtuz=sum(tm.*CL(:,3)');

% R
r=[x y z]-p;

J=zeros(3,3);  % J is the Jacobian matrix which is symmetric
J(1,1)=dot([dsx dsy dsz]',[dsx dsy dsz]')+dot(r',[dssx dssy dssz]');
J(2,2)=dot([dtx dty dtz]',[dtx dty dtz]')+dot(r',[dttx dtty dttz]');
J(3,3)=dot([dux duy duz]',[dux duy duz]')+dot(r',[duux duuy duuz]');
J(1,2)=dot([dtx dty dtz]',[dsx dsy dsz]')+dot(r',[dstx dsty dstz]');
J(1,3)=dot([dux duy duz]',[dsx dsy dsz]')+dot(r',[dsux dsuy dsuz]');
J(2,3)=dot([dux duy duz]',[dtx dty dtz]')+dot(r',[dtux dtuy dtuz]');
J(2,1)=J(1,2);  J(3,1)=J(1,3);  J(3,2)=J(2,3);

kk=-[dot(r',[dsx dsy dsz]') dot(r',[dtx dty dtz]') dot(r',[dux duy duz]')]';

dp=inv(J)*kk;
pp=pp+dp';

%if (pp(1)<0),  pp(1)=0;  end;
%if (pp(1)>1),  pp(1)=1;  end;
%if (pp(2)<0),  pp(2)=0;  end;
```

```
  %if (pp(2)>1),  pp(2)=1;  end;
  %if (pp(3)<0),  pp(3)=0;  end;
  %if (pp(3)>1),  pp(3)=1;  end;

  pp=max(pp,0);  pp=min(pp,1);

  % Tests
  % Point coincidence test
  lr=norm(r);

  % Zero cosine tests
  fs=abs(dot([dsx dsy dsz]',r'))/norm([dsx dsy dsz])/lr;
  ft=abs(dot([dtx dty dtz]',r'))/norm([dtx dty dtz])/lr;
  fu=abs(dot([dux duy duz]',r'))/norm([dux duy duz])/lr;

  % Parameter not change significantly
  f=norm(dp(1)*[dsx dsy dsz]+dp(2)*[dtx dty dtz]+dp(3)*[dux duy duz]);

  flag=(lr<=e1) | ((fs<=e2) & (ft<=e2) & (fu<=e2)) | (f<=e1);
  count=count+1;
end;

% For closed solid, make sure the parameters lie in the range [0,1]

px=pp(1);  py=pp(2);  pz=pp(3);
```

newfparam.m

```
function [px,py,pz]=newfparam(x,y,z,orgparam,k,ngrids,m,n)
% [px,py,pz]=newfparam(x,y,z,orgparam,k,ngrids,p0,siz)
%
% Find correspondence between data point and model point on the original shape

p0=[-1 -1 -1];  siz=[2 2 2];
nth=n-1;
% [m,n]=size(x);
% x=x(:);  y=y(:);  z=z(:);
e1=orgparam(1);  e2=orgparam(2);

knotx=openknot(k(1),ngrids(1));
knoty=openknot(k(2),ngrids(2));
knotz=openknot(k(3),ngrids(3));

%%%%%%%%%%%%%%% Correspondence Problem %%%%%%%%%%%%%%%
maxx=max(x(:));  minx=min(x(:));
maxy=max(y(:));  miny=min(y(:));
maxz=max(z(:));  minz=min(z(:));

nsiz=1.0*abs([maxx-minx,maxy-miny,maxz-minz]);
cen=mean([[maxx minx]' [maxy miny]' [maxz minz]']);
np0=cen-nsiz/2;

% Normalize the data point into the domain [-1,1]x[-1,1]x[-1,1]
%nx=(x-np0(1)-1/2*nsiz(1))*2/nsiz(1);
%ny=(y-np0(2)-1/2*nsiz(2))*2/nsiz(2);
nz=(z-np0(3)-1/2*nsiz(3))*2/nsiz(3);

phi=atan2(linspace(-1,1,size(nz,1))',ones(size(nz,1),1))';
% phi=atan2(nz(:,1),ones(size(nz,1),1))';
nphi=length(phi);
theta=[-nth:2:nth]/nth*pi;

ta=siz/2;  a1=ta(1);  a2=ta(2);  a3=ta(3);
cosphi=cos(phi);  sinphi=sin(phi);
costh=cos(theta);  sinth=sin(theta);
for i=1:nphi,
  rho=((abs(costh.*cosphi(i)).^(2/e2)+abs(sinth.*cosphi(i)).^(2/e2)).^(e2/e1)+ ...
  abs(sinphi(i)).^(2/e1)).^(-e1/2);
  xx(i,:)=a1.*rho.*costh.*cosphi(i);
```

```
  yy(i,:)=a2.*rho.*sinth.*cosphi(i);
  zz(i,:)=a3.*rho.*sinphi(i).*ones(1,nth+1);
end;
xx=xx(:);  yy=yy(:);  zz=zz(:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Porg=lattice(p0,siz,ngrids);

options=foptions;
options(1)=-1;
options(2)=1e-6;
options(3)=1e-6;
options(5)=0;

load standard_stu;
total=length(xx);
inc=round(total/10);

for i=1:total,
  ox=[xx(i) yy(i) zz(i)];
  ix=find(xx(i)>=xs);  ix=ix(length(ix));
  iy=find(yy(i)>=ys);  iy=iy(length(iy));
  iz=find(zz(i)>=zs);  iz=iz(length(iz));
  seed=[ss(ix) tt(iy) uu(iz)];
  [pxt,pyt,pzt]=findp(ox,knotx,knoty,knotz,k,ngrids,Porg,seed);
  px(i)=pxt;  py(i)=pyt;  pz(i)=pzt;
  if (rem(i,inc)==0),
    disp([num2str(round(i/total*100)),'% completed.']);
  end;
end;
px=reshape(px,nphi,nth+1);
py=reshape(py,nphi,nth+1);
pz=reshape(pz,nphi,nth+1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Begin to find an interpolation of [s,t,u]=f(a,b)
% where f(a,b) is a bivariate B-spline function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% [pfitx,pfity,pfitz,ku,kv,tu,tv]=surffit(px,py,pz,4);

% save pfit pfitx pfity pfitz ku kv tu tv;
```

surffit.m

```
function [pfitx,pfity,pfitz,ku,kv,tu,tv]=surffit(px,py,pz,k0)

% Interpolation of (s,t,u) parameter triple
% u is in the row direction
% v is in the column direction
[mm,nn]=size(px);
[tu,tv]=surfparam(px,py,pz);

% compute the 2 knot vectors, ku and kv resp.
ku=meanknot(k0,tu);
kv=meanknot(k0,tv);

% Begin interpolation
% In the row direction
NN=spcol(ku,k0,tu,1);              % NN is in the almost block diagonal form
for i=1:mm,
  CP=slvblk(NN,[px(i,:)' py(i,:)' pz(i,:)']);
  Rx(i,:)=CP(:,1)';  Ry(i,:)=CP(:,2)';  Rz(i,:)=CP(:,3)';
end;
NN=spcol(kv,k0,tv,1);
for i=1:nn,
  CP=slvblk(NN,[Rx(:,i) Ry(:,i) Rz(:,i)]);
  pfitx(:,i)=CP(:,1);  pfity(:,i)=CP(:,2);  pfitz(:,i)=CP(:,3);
end;
```

surfparam.m

```
function [tu,tv]=surfparam(px,py,pz)
% Compute the parameter for data point for surface interpolation
% tu is in the row direction
% tv is in column direction

[m,n]=size(px);

% For each row
tu=[];
for i=1:m,
  tmp=curparam([px(i,:)' py(i,:)' pz(i,:)']);
  tu=[tu; tmp];
end;
tu=mean(tu);

% For each column
tv=[];
for i=1:n,
  tmp=curparam([px(:,i) py(:,i) pz(:,i)]);
  tv=[tv; tmp];
end;
tv=mean(tv);
```

curparam.m

```
function t=curparam(data)
% Find the parameter for each data point for curve interpolation
% Use Centripetal Parametrization
% This parameterization gives good result for data points with sharp turns.

N=length(data);

% Assign paramterization of data points
dx=diff(data(:,1)');  dy=diff(data(:,2)');  dz=diff(data(:,3)');
dl=sqrt(dx.*dx+dy.*dy+dz.*dz);
sumdl=sum(sqrt(dl));
t=zeros(1,length(data));
for i=2:length(data),          % Centripetal Parametrization
  t(i)=t(i-1)+sqrt(dl(i-1))/sumdl;
end;
```

Bibliography

[1] Christoph M. Hoffmann. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann, 1989.

[2] S. T. Venkataraman and T. Iberall, editors. *Dextrous Robot Hands.* Spring-Verlag, 1990.

[3] Stanley Corden, Lawrence M. Ward, and James T. Enns. *Sensation and Perception.* Harcourt Brace and Company, fourth edition, 1994.

[4] William Schiff and Emerson Foulke, editors. *Tactual Perception: A Sourcebook.* Cambridge University Press, 1982.

[5] Howard R. Nicholls, editor. *Advanced Tactile Sensing for Robotics*, volume 5 of *World Scientific Series in Robotics and Automation Systems.* World Scientific, 1992.

[6] S. C. Jacobsen, J. E. Wood, D. F. Knutti, and K. B. Biggers. The Utah/MIT Dextrous Hand: Work in Progress. In M. Brady and R. P. Paul, editors, *The 1st International Conference on Robotics Research*, pages 601–53. The MIT Press, 1984.

[7] S. C. Jacobsen, E. K. Iversen, D. F. Knutti, R. T. Johnson, and K. B. Biggers. Design of the Utah/MIT Dextrous Hand. In *Proceedings 1986 IEEE International Conference on Robotics and Automation*, pages 1520–33, 1986.

[8] R. Andrew Russell. *Robot Tactile Sensing.* Prentice Hall, 1990.

[9] Robert D. Howe. Tactile Sensing and Control of Robotic Manipulation. *Advanced Robotics*, 8(3):245–61, 1994.

[10] R. L. Klatzky, S. J. Lederman, and V. A. Metzer. Identifying Objects by Touch: An Expert System. *Perception and Psychophysics*, 37(4):299–302, 1985.

[11] Richard M. Satava. The Modern Medical Battlefield: Sequitur on Advanced Medical Technology. *TEEE Robotics and Automation Magazine*, 1(3):21–25, September 1994.

[12] Peter K. Allen. Active Sensing with a Dextrous Robotic Hand. In T. C. Henderson, editor, *Traditional and non-Traditional Robotic Sensors*, volume F63 of *NATO ASI*, pages 223–239. Spring-VerlagBerlin Heidelberg, 1990.

[13] Peter K. Allen. Acquisition and Interpretation of 3-D Sensor Data from Touch. *IEEE Transactions on Robotics and Automation*, 6(4):397–404, August 1990.

[14] Peter K. Allen and Kenneth S. Roberts. Haptic Object Recognition using a Multi-Fingered Dextrous Hand. In *Proceedings of IEEE International Conference on Robotics and Automation* [15], pages 342–7.

[15] Peter K. Allen. Mapping Haptic Exploratory Procedures to Multiple Shape Representations. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 1679–84, 1990.

[16] A. H. Barr. Superquadrics and Angle-Preserving Transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, January 1981.

[17] Wm. Randolph Franklin and Alan H. Barr. Faster Calculation of Superquadric Shapes. *IEEE Computer Graphics and Applications*, pages 41–47, July 1981.

[18] Alan H. Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* ACM Press: New York/Addison-Wesley Pub., 1992.

[19] E. Bardinet, N. Ayache, and L. D. Cohen. Fitting of iso-surfaces using superquadrics and free-form deformations. In *Proc. of IEEE Workshop on Biomedical Image Analysis*, pages 184–193, June 1994.

[20] Henry J. Lamousin and Warren N. Waggenspack Jr. NURBS-based Free-Form Deformations. *IEEE Computer Graphics and Applications*, pages 59–65, November 1994.

[21] J. Greiessmair and W. Purgathofer. Deformation of solids with trivariate b-splines. In *Proc. Eurographics 89*, pages 137–148. Elsevier Science PublishersNorth-Holland, 1989.

[22] Sabine Coquillart. Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modelling. *Computer Graphics*, 24(4):187–193, August 1990.

[23] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Solid Geometric Models. *Computer Graphics*, 20(4):151–160, August 1986.

[24] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct Manipulation of Free-Form Deformation. *Computer Graphics*, 26(2):177–184, July 1992.

[25] Mongi A. Abidi and Rafael C. Gonzalez, editors. *Data Fusion in Robotics and Machine Intelligence.* Academic Press, 1992.

[26] Peter K. Allen. *Robotic Object Recognition Using Vision and Touch.* Kluwer Academic Publishers, 1987.

[27] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics.* McGraw Hill Publishing Company, second edition, 1990.

[28] Josef Hoschek and Dieter Lasser. *Fundamentals of Computer Aided Geometric Design.* A K Peters, Ltd., 1993. Translated byLarry L. Schumaker.

[29] Mamoru Hosaka. *Modeling of Curves and Surfaces in CAD/CAM.* Spring-VerlagBerlin Heidelberg, 1992.

[30] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide.* Academic Press Inc., third edition, 1993.

[31] Alfred Gary. *Modern Differential Geometry of Curves and Surfaces.* CRC Press, Inc., 1993.

[32] Franc Solina and Ruzena Bajcsy. Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):131–47, February 1990.

[33] Martti Mantyla. *An Introduction to Solid Modeling.* Computer Science, 1988.

[34] Yehuda E. Kalay. *Modeling Objects and Enivronments.* John Wiley and Sons, 1989.

[35] L. E. Scales. *Introduction to Non-linear Optimization.* Macmillan Publishers LTD., 1985.

[36] A. H. Barr. Global and Local Deformations of Solid Primitives. *Computer Graphics*, 18(3):21–30, July 1984.

[37] Les Piegl and Wayne Tiller. *The NURBS Book.* Spring-Verlag, 1995.

[38] Dieter Lasser. Visualization of Free Form Volume. In *Proceedings of the First IEEE Conference on Visualization, Visualization'90*, pages 379–86, 1990.

[39] Jean Ponce and David Chelberg. Invariant Properties of Straight Homogeneous Generalized Cylinders and their Contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(9):951–66, September 1989.

[40] F. Ulupinar and R. Nevatia. Shape from Contour: Straight Homogeneous Generalized Cylinders and Constant Cross Section Generalized Cylinders. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):120–35, February 1995.

[41] Andrew J. Hanson. Hyperquadrics: Smoothly Deformable Shapes with Convex Polyhedral Bounds. In *Computer Vision, Graphics, and Image Processing*, volume 44, pages 191–210. Academic Press, Inc., 1988.

[42] Isaac Cohen and Laurent D. Cohen. A Hybrid Hyperquadric Model for 2-D and 3-D Data Fitting. *Computer Vision, Graphics and Image Processing: Image Understanding*, 1996.

[43] Senthil Kumar and Dmitry Goldgof. Model based Part Segmentation of Range Data – Hyperquadrics and Dividing Planes. In *Proceedings of the Workshop on Physics-Based Modeling in Computer Vision*, pages 17–23. IEEE Computer Society Press, June 1995.

[44] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer*, 11(8):429–46, 1995.

[45] V. Shapiro. Real Functions for Representation of Rigid Solids. *Computer Aided Geometric Design*, 11(2):153–175, 1994.

[46] K. T. Miura, A. A. Pasko, and V. V. Savchenko. Parametric Patches and Volumes in Function Representation of Geometric Solids. *CSG 96, Set-theoretic Solid Modelling Techniques and Applications*, pages 217–31, 1996.

[47] V. Savchenko and A. Pasko. Recomstruction from Contour Data and Sculpting 3D Objects. *Journal of Computer Aided Surgery*, 1:56–7, October 1995.

[48] Vladimir V. Savchenko, Alexander A. Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function Representation of Solids Reconstructed from Scattered Surface Points and Contours. *Computer Graphics Forum*, 14(4):181–8, 1995.

[49] Jae S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, 1990.

[50] Peter Lancaster and Kęstutis Šalkauskas. *Curve and Surface Fitting, An Introduction.* Academic Press Inc., third edition, 1990.

[51] J. Hoschek. Intrinsic Parametrization for Approximation. *Computer Aided Geometric Design*, 5:27–31, 1988.

[52] Biplab Sarkar and Chia-Hsiang Menq. Parameter Optimization in Approximating Curves and Surfaces to Measurement Data. *Computer Aided Geometric Design*, 8:267–290, 1991.

[53] The MathWorks Inc. *MATLAB Reference Guide*, August 1992.

[54] The MathWorks Inc. *MATLAB External Interface Guide*, January 1993.

[55] Andrew Grace. *Optimization Toolbox - For Use with MATLAB.* The MathWorks Inc., November 1992.

[56] Carl de Boor. *Spline Toolbox - For Use with MATLAB.* The MathWorks Inc., November 1992.

[57] Carl de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciences.* Spring-Verlag, 1978.

[58] John Pearson Refling. *Piplined Implementation of B-splines and Beta-splines for Computer Graphics and other Discrete Applications.* PhD thesis, University of California, Irvine, 1993.

[59] Donald E. Catlin. *Estimation, Control, and the Discrete Kalman Filter.* Spring-Verlag, 1989.

[60] Les Piegl. Modifying the Shape of Rational B-splines. Part 1: Curves. *Computer Aided Design*, 21(8):509–18, 1989.

[61] Les Piegl. Modifying the Shape of Rational B-splines. Part 2: Surfaces. *Computer Aided Design*, 21(9):538–46, 1989.

[62] I. T. Jolliffe. *Principal Component Analysis.* Spring-Verlag, 1986.