1-1-2002

# Cloud data segmentation and classification for reverse engineering using neural networks

Jiahong Wang
*Ryerson University*

# CLOUD DATA SEGMENTATION AND CLASSIFICATION FOR REVERSE ENGINEERING USING NEURAL NETWORKS

by

Jiahong Wang

MASc. - Mech

Yanshan University, China, 2002

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Mechanical Engineering

Toronto, Ontario, Canada, 2006

# AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institution or individuals for the purpose of scholarly research.

Signature: ______________________ Date: May 12, 2006

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature: Date: May 12, 2006

# Abstract

## Cloud Data Segmentation and Classification for Reverse Engineering using Neural Networks

**Jiahong Wang**
**Master of Applied Science, 2006**
**Mechanical Engineering, Ryerson University**

Automatic segmentation of point data in the past has been mainly applied to single range maps. However, there is a great need for the segmentation of fully digitized objects with multiple viewpoints. This research reports on the automatic segmentation of multiple viewpoint 3D digitized data captured by a laser scanner or a CMM. This is accomplished in two steps. Firstly, the surface normal and principal curvatures are estimated at corresponding point locations. Local Darboux frame and weighted least-square surface fitting are used to calculate the normal values and curvature values of the point data. Secondly, an eight dimensional feature vector (3D coordinate, 3D normal, Gaussian and Mean curvature) is used as an input to a Self-Organized Feature Map (SOFM). A normalized feature vector and a weighted Euclidean distance are adopted in the learning process of the SOFM, which improves the speed and exactness of the segmentation. The segmentation using SOFM is robust to noise and has no limitation to surface type. The algorithm is validated by real and synthetic point data. To improve the quality of surface fitting, segmented subregions of typical surfaces are classified by using a back propagation neural network. The techniques developed play a key role in reducing the length of product development time and the quality of a final surface model.

# ACKNOWLEDGEMENTS

The author would first and foremost like to thank his thesis supervisor, Dr. Vincent Chan, for his guidance, knowledge, and support in this research.

The author would also like to extend thanks Mr. Devin Ostrom for his help with preparing some experimental parts using CNC machine.

Finally, I would like to thank all my family and friends for their support and encouragement through this endeavor.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF APPENDICES

# NOMENCLATURE

| | |
|---|---|
| $x$ | normalized x coordinate of input vector |
| $y$ | normalized y coordinate of input vector |
| z | normalized z coordinate of input vector |
| $n_x$ | normalized x normal component of input vector |
| $n_y$ | normalized y normal component of input vector |
| $n_z$ | normalized z normal component of input vector |
| H | Gaussian curvature of input vector |
| K | mean curvature of input vector |
| $x_i$ | Normalize input vector |
| $w_x$ | x coordinate weight of the representative region |
| $w_y$ | y coordinate weight of the representative region |
| $w_z$ | z coordinate weight of the representative region |
| $w_{nx}$ | x normal component weight of the representative region |
| $w_{ny}$ | y normal component weight of the representative region |
| $w_{nz}$ | z normal component weight of the representative region |
| $w_H$ | Gaussian curvature weight of the representative region |
| $w_k$ | mean curvature weight of the representative region |
| q | the location vector of the winning neuron in output layer |
| m | the number of triangles that share the same vertex |
| $N_q(t)$ | neighborhood size in *t* iteration |
| $\alpha(t)$ | a scalar-valued "learning rate factor" |

| | |
|---|---|
| $\sigma(t)$ | the width of the kernel |
| $h_{qj}$ | a smooth neighborhood kernel of Gaussian function |
| $A$ | parameter- learning rate of connectors between hidden and output layer |
| $B$ | parameter- learning rate of connectors between hidden and input layer |
| $w_j$ | weight vector |
| f(x) | hypertangent function |
| $\frac{dx_{(2,j)(3,i)}}{dt}$ | the adjustment of weights between the ouput and hidden layer |
| $\frac{dx_{(1,k),(2,j)}}{dt}$ | the adjustment of weights between the input layer and the hidden layer |
| $\eta_{(1,i)}$ | input neural value at neural location i |
| $\eta_{(2,j)}$ | hidden neural value at neural location j |
| $\eta_{(3,i)}$ | output neural value at neural location i |
| $w_{(1,i)(2,j)}$ | connector weight between neural at the hidden and input layer |
| $w_{(2,j)(3,i)}$ | connector weight between neural at the hidden and output layer |
| $E_{output,i}$ | Error at the output layer, neuron location i |
| $E_{hidden,j}$ | Error at the hidden layer, neuron location j |
| N | number of iterations for the neural network training |
| $x_{min}$ | minimum value |
| $x_{max}$ | maximum value |
| $R_i$ | reflected vector at the output layer |
| $\sigma$ | population standard deviation |
| $\mu$ | population arithmetic mean |
| $e$ | bias |
| $k_1$ | maximum principal curvature |
| $k_2$ | minimum principal curvature |

| | |
|---|---|
| $P_{i,j}$ | control points |
| $W_{i,j}$ | weights |
| $N_{i,p}(u)$ | B-spline basic functions. |
| LMS | least mean square |
| RMS | root mean square |
| ANN | artificial neural network |

To My Family

# Chapter 1
# Introduction

## 1.1 Reverse Engineering Technology

Reverse engineering (RE) refers to designing and producing of products on the basis of an existing physical part. It has been widely used from spacecraft and automobile design to human organs and copies of artistic sculptures [1-3]. Generally, three steps are considered in RE:

1. Digitize the object's surface using a scanning or measurement device and preprocessing (noise reduction).
2. Segmentation and classification of point data.
3. Surface fitting and CAD model creation.

Ideally, the ultimate goal is to have a fully automatic reverse engineering system which can implement an RE geometric modeling without any user interaction. The emphasis of this work is on the second step of RE: automatic segmentation and classification of point data. Specifically, digitized objects with multiple viewpoints are segmented automatically.

In reverse engineering, segmentation is used to divide point data into subsequent regions according to an object's shape. It plays a key role in reducing the length of product development time and the quality of the final surface model. The point data in RE obtained by non-contact measuring devices, such as a laser scanner or contact measuring devices such as a CMM, consists of thousands of points that only include three-dimensional coordinates on the surface of an object. It is difficult to obtain the geometric information of a part directly from the raw data. Therefore, in most RE systems, the segmentation of digitized points is performed interactively, where the operator defines the approximate locations of part edges and surface boundaries on the digitized data [4, 5].

The manual process by a RE operator is time consuming and prone to error. Automatic segmentation of point data in the past has been mainly applied to single range data maps [6-10]. However, for objects with complex topology or complex geometry, the point data from one viewpoint are not enough to characterize the object shape. To fully digitize an object, multiple viewpoints or orientations need to be taken with multiple scanning passes. Sometimes a multiple sensor integrated approach may be necessary to combine different characteristics of several scanning devices. Unlike a range map, points in multiple viewpoint data are not on a regular grid. Methods that rely on this regular organization of data are not directly applicable for a general mesh. For example, window operators to compute surface normal and curvature values of a range map cannot be used because of the overlapping and disorganized nature of the point data. Methods to segment digitized objects with multiple viewpoints are challenging.

After the surface of an object is segmented into various component surfaces (or multi-patched surfaces), the type of surface that each subset of points belongs to needs to be decided. The classification is a critical component in automated RE and a necessary companion to point data segmentation. A good classification of surfaces complements surface fitting.

## 1.2 Automatic Reverse Engineering System

A Roland Dr. Picza LPX-250 3D laser scanner (Figure 1.1) and a Roland Dr. Picza PIX-30 3D touch scanner (Figure 1.2) are used to gather point data from the surfaces of objects. The LPX-250 is a non-contact scanner. It can scan objects from multiple viewpoints or orientations (up to 6 orientations). The PIX-30 acquires point data using contact methods. Advantages of the contact methods are less noise and higher accuracy.

Point data collected are saved as a STL file. If mesh triangulation is ill-behaved or noise in the cloud data is unacceptable (causing a large error), a reverse engineering software package, Surfacer version 10.6 is used to preprocess point data. Otherwise, the STL file consisting of a list of triangular facet data is used directly to calculate point normal and curvature. In the estimating of principal curvatures using the Darboux frame and the local least-square method, the one-ring neighborhood for accurate 3D cloud data is chosen. If there is noise (causing a small error) in the cloud data, a two-ring or higher-ring neighborhood will be used to estimate the principal curvatures. The calculation of the point normal and curvature are programmed by codes developed in MATLAB.

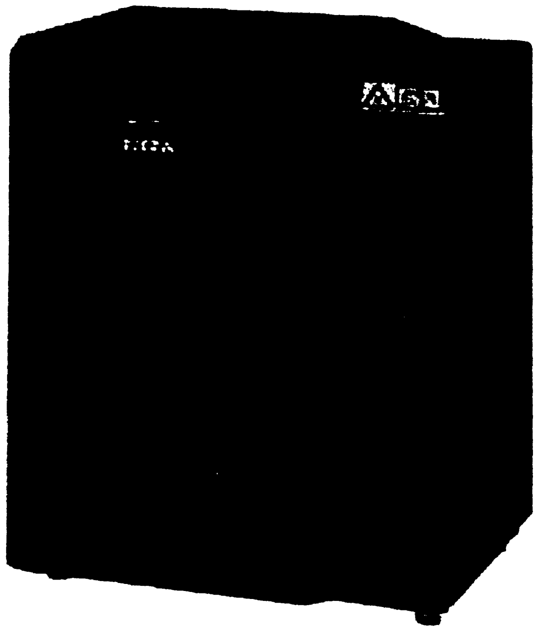


Figure 1.1 Picza LPX-250 3D Laser Scanner

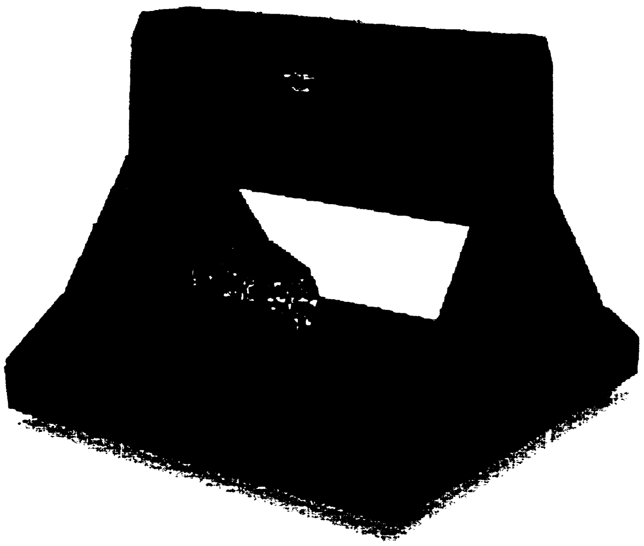


Figure 1.2 Picza PIX-30 3D touch Scanner

After computing the feature vector (3D point coordinate, point normal and point curvatures), measured point data are segmented by an unsupervised neural network-Kohonen's self-organizing feature map. The segmented point subsets are then classified by a supervised feed-forward based neural network. Finally, the segmented point subsets are fitted according to their primitive surfaces. If a segmented patch is in the typical surface library (plane, cylinder, sphere, cone, torus, etc), a least-square method is used to fit the surface. Otherwise, Coons, Bezier, or NURBS surfaces are used to reconstruct the surface. Surface trimming is sometimes needed after the surface fitting using the least-square method. The flow chart of the automatic reverse engineering system is shown in Figure 1.3.

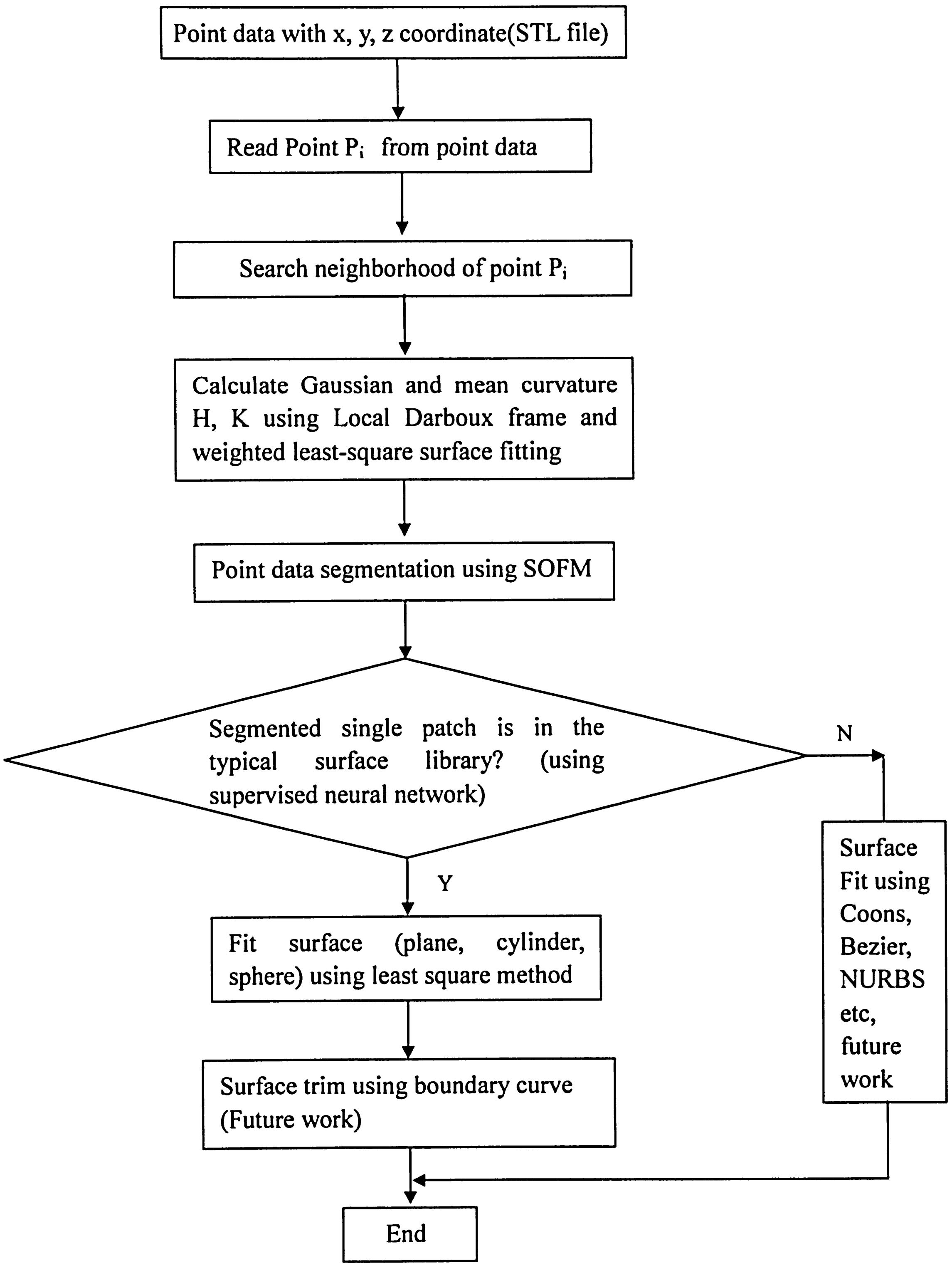

Point data with x, y, z coordinate(STL file)
Read Point $P_i$ from point data
Search neighborhood of point $P_i$
Calculate Gaussian and mean curvature H, K using Local Darboux frame and weighted least-square surface fitting
Point data segmentation using SOFM
Segmented single patch is in the typical surface library? (using supervised neural network)
N
Y
Fit surface (plane, cylinder, sphere) using least square method
Surface trim using boundary curve (Future work)
Surface Fit using Coons, Bezier, NURBS etc, future work
End

Figure 1.3 Automatic Reverse Engineering System

## 1.3 Thesis Objectives

The main objective of this thesis is to set up an automatic reverse engineering geometry modeling system. The system can be implemented through the following objectives:

1. Automatic segmentation of point data with multiple viewpoints.
2. Automatic classification of segmented single patch.
3. Automatic surface fitting.

## 1.4 Thesis Overview

This thesis is organized as follows:

Chapter 1 introduces reverse engineering technology. This chapter describes the problem with current segmentation methods of point data in reverse engineering. To solve this problem, an automatic reverse engineering system is suggested.

Chapter 2 discusses data segmentation methods and the related literature review of data segmentation and curvature estimating techniques.

Chapter 3 presents a SOFM algorithm to segment point data with mutli-viewpoints. The Darboux frame and least-square method are used to calculate feature vector of the SOFM.

Chapter 4 discusses the supervised neural network method applied in this work to classify point data and tests the back propagation algorithm.

Chapter 5 tests SOFM algorithm with synthetic and real reverse engineering data. The experimental results are analyzed.

Chapter 6 discusses the conclusion and future work.

# Chapter 2
# Literature Review

## 2.1 Introduction

Segmentation has been an essential part in the process of surface modelling from a scanned point cloud [11]. A commonly used definition of image segmentation [4, 12] states that if $I$ is the set of all image pixels (or points) and P ( ) is a uniformity predicate defined on groups of connected pixels (or points), a segmentation of $I$ is a partitioning set of connected subsets or image regions$\{R_1,\ldots,R_N\}$ such that

$$\bigcup_{l=1}^{N} R_l = I \text{ where } R_l \cap R_m = \Phi \quad \forall l \neq m \tag{2.1}$$

The uniformity predicate $P(R_l)$=True for all regions, and

$$P\,(R_l \cup R_m) = \text{False} \tag{2.2}$$

whenever $R_l$ is adjacent to $R_m$.

This definition can be applied to all data type including digital image or 3D surface data. To divide the whole measurement point data into subsets according to primitive surface has been a long-standing research problem. Previous work will be reviewed in Section 2.2. To efficiently segment point data, principal curvatures of point data are to be estimated. The different methods of estimating principal curvatures are reviewed in Section 2.3.

## 2.2 Review of Existing Segmentation Methods

The segmentation of 3D digitized data or range data have been studied by various researchers. The currently reported segmentation algorithms are based on the assumption that scanned data exhibits surface coherence. In general, segmentation techniques of 3D point data can be classified into four categories: edge-detection methods, region-growing methods, hybrid methods and neural network methods.

### 2.2.1 Edge-Based Method

Edge-based methods are a two-stage process, edge detection and linking [6, 13]. This works by attempting to find boundaries in the point cloud data representing edges between surfaces. If edges are being sought, an edge-linking process follows, in which disjoint edge points are connected to form a continuous edge. Detected edges usually involve jump edges, crease edges and curvature edges. Jumps edges occur where depth values are discontinuous in a range image. Crease edges correspond to surface pixels where the depth is continuous but the surface normals are discontinuous. Curvature edges are characterized by continuity of the surface normals but discontinuity of the surface curvature.

Woo and Kang [11] used the octree-based 3D-grid method to handle a large amount of unordered point data. The root node in the octree represents the entire point data. The point data is divided into eight child nodes by halving the parent cell (or node) along the x, y and z directions based on the criteria defined by the application (in reference [11], the subdivision criteria of cells is the standard deviation of point normals in each cell). The partitioning is performed iteratively until the leaf nodes are reached. Obviously, the restriction of octree-based method is that each parent node (or cell) must have eight child nodes. Thus regions in the segmented point data have a boxy appearance. This method is usually suitable for the segmentation of boxy appearance point data. At the same time, quadric surfaces such as a cylinder or a cone cannot be segmented accurately because the octree-based segmentation method does not consider Gaussian and mean curvature.

Milory and Bradley [13] used a semi-automatic edge-based method for orthogonal cross-section (OCS) models. In this approach, surface differential properties were estimated at each point of model, and the curvature extremes were identified as possible edge points. Then an energy-minimizing active contour was used interactively to link the edge points. Fan and Medioni [14] used local surface curvature properties to identify significant boundaries in the range data. In order to avoid the loss of localization, scale-space tracking, which convolves the entire data with Gaussian masks having different values of the spread parameter, was employed. Lee and Kim [15] segmented point data by calculating the curvature and angle between two triangles which share an edge, and find an edge with large curvature and curvature deviation regions. Yang and Lee [6] developed a segmentation algorithm using a parametric quadric surface approximation. This method is based on local surface curvature properties. The surface curvature and principal directions are computed from the locally approximated surfaces. Edge points are identified as the curvature extremes and zero crossings, which are found from the estimated surface curvature. After edge points are identified, an edge-neighborhood chain-coding algorithm is used for forming boundary curves. The original point set is then broken down into subsets, which meet along the boundaries. All point data are applied to each boundary loop to partition the points into different regions. But this method is only suitable for single range maps.

Edge-based methods suffer from the following problems [1]. Sensor data, particularly from laser-based scanner, are often unreliable near sharp edges, because of specular reflections there. The number of points used for segmenting the data is small, i.e. only points in the vicinity of the edges are used, which means that information from much of the data is not used to assist in reliable segmentation. On the other hand, region-growing methods work on a larger number of points, in principle using all available data.

## 2.2.2 Region-Based Method

Region-based methods infer connected regions of points that have homogeneity or similar geometrical properties. Hoffiman and Jain [7] segmented the range image into many surface patches and classified these patches as planar, convex or concave shape based on a non-parametric statistical test for trend, curvature values and eigenvalue analysis. Besl and Jain [12] have developed an algorithm that segments a large class of an image into regions of arbitrary shape using a piecewise smooth surface model for image data having surface coherence properties. They approximate image data with bivariate functions so that it is possible to compute a complete noiseless image reconstruction based on the extracted functions and regions. This algorithm is used for segmenting both range data and intensity images. The first stage of the algorithm divides the image into eight different primitive surfaces. The type of primitive surface depends on the sign of the Gaussian and the mean curvature. These primitives are assumed to have invariant characteristics which can break down any given surface. Using the labeled image, a seed region is extracted that is used for region growing in the second region. The second stage is an iterative region growing process. This process is terminated when the region growing has converged or all the polynomials have failed to fit the seed region.

All these region growing techniques depend heavily on initial seed generation which is rather difficult and also critical for the success of regions growing procedures. At the same time most of these fitting methods suffering inherently from computational overhead-segmentation processing time that can be up to several CPU hours depending on image complexity and number of surface primitives used [10].

### 2.2.3 Hybrid Method

The hybrid method refers to the combination of region-based and edge-based methods. A combination of these two approaches is employed to overcome the problems of oversegmetation and undersegmentation, which are encountered in the edge-based and region-based methods. The method proposed by Yokya and Levine [16] divided a three-dimensional measurement data set into surface primitives which are homogeneous in their intrinsic differential geometric properties and do not contain discontinuity in either depth or surface orientation. The method employs a selective surface fit and is based on the computation of first and second partial derivatives determined by locally approximating object surface using biquadratic polynomials. Then by computing the Gaussian and mean curvature and examining their signs, an initial region-based segmentation is obtained in the form of a curvature sign map. Two initial edge-based segmentations are also computed from the partial derivatives and depth values. One detects jump edge by computing differences in range values and crease edge from differences in surface normals. The three initial image maps are then combined to produce the final range image segmentation. Zhao and Zhang [17] employed a method based on triangulation and region grouping that uses edges, critical points and surface normals. In their algorithm, the initial edges and critical points were detected using morphological residues, and the linked pairs of edge points were triangulated. For the final segmentation, smaller surface triangular facets were expanded into large ones and segmented according to the surface normal information. Lee and Park [18] used a hybrid approach to segment point data, a method to the edge detection of 3D points based on a region growing technique. The algorithm consists of two parts. First, polygonal meshes are generated from the scanned point data using the Delaunay triangulation algorithm. Second, the normal vector and the area of a polygonal mesh are checked to find boundary meshes using angle criterion and area criterion based on a region growing technique. The region growing technique aggregates meshes into a region until the area of aggregated meshes reaches an area threshold from a series of seed meshes. Similar to [11], quadric surfaces such as a cylinder can not be segmented accurately because this method does not consider principal curvatures.

### 2.2.4 Neural Network Method

Many researchers have worked on image segmentation using neural networks. Sugata and Mehrotra [10] described a neural network for the segmentation of a noisy image. They used a two-stage connectionist neural net model which extracts local surface features at each image point and group pixel via local interaction among different features. Stage one computes the surface parameters, e.g., surface normals, curvature and discontinuities (crease and jump) by optimally projecting the local range profile onto a set of non-orthogonal basis functions. In stage two, adjacent pixels compete with each other based on surface features associated with them to group themselves into different surface patches. The surface feature extraction is performed using the Daugman's projection neural network and Kohonen's self-organizing neural network is used for competitive region growing.

One of the limitations of the conventional SOFM is that the number of neural units in the competitive layer should be approximately equal to the number of regions desired in the final segmentation. However, it is usually not possible to predetermine the number of regions in the final segmentation. This makes the use of original single layer SOFM for image segmentation rather difficult. When the SOFM has much fewer neural units than the number of visually distinguishable regions in the input image, the result is an undersegmented image in which visually distinguishable regions are incorrectly merged into a single region in the segmented image. On the other hand, when the SOFM has a much larger number of neural units than the number of visually distinguishable regions in the input image, the result is an oversegmented image in which homogeneous regions are incorrectly split into several regions in the segmented image. Thus, there is no control over the number of regions which might result from image segmentation when a single-layer SOFM is used. This is a significant deficiency of the single-layer SOFM since the number of regions in the final segmented image is very much dependent on the input scene content, and is difficult to determine accurately a priori.

Thus a hierarchical self-organizing feature map (HSOFM) for range image segmentation was proposed by Jean and Minsoo [8, 9]. HSOFM is an extension of traditional (single-layer) self-organizing feature map (SOFM). It alleviates the shortcomings of SOFM in the context of image segmentation. The problem of image segmentation is formulated as one of vector quantization and mapped onto the HSOFM. The HSOFM combines the ideas of self-organization and topographic mapping with those of multi-scale image segmentation. Chan and Bradley [19] used a multilayer neural network based stereo image processing to locate the object in the CMM work space, and to generate the CMM touch probe path. In their work, the stereo pair of CCD images is first segmented into surface patches using a neural network based algorithm. In this ten layer neural network, layer one is used to input one original image, whereas the layers above each (the remaining nine layers) represent a possible output (winning patch). Each iteration of the algorithm calculates the strength of each neuron by updating the values for excitatory and inhibitory connectors. Iteration is complete when the output converges, i.e. no new or different winning neurons are declared. The location and radius of the holes on each patch is derived from co-ordinate information already calculated for each patch. However, all these methods are not suitable for multiple viewpoint data.

## 2.3 Curvature Estimating

Surface curvature is a concept rooted in differential geometry [20]. Differential geometry states that a local surface shape is uniquely determined by the first and second fundamental forms. Gaussian and mean curvature combine these first and second fundamental forms in two different way to obtain scalar surface features that are invariant to rotation, translations, and changes in parameterization. Therefore, Gaussian and mean curvature can be used to identify surface features such as ridges and valleys, and planar, convex, concave, or saddle shapes. Surfaces are segmented into regions based on these curvature features, and then the segmentations and features are used for object recognition and registration. Curvature estimating can be classified into two categories: range map and triangular mesh.

### 2.3.1 Curvature on Range Image

A number of researchers have worked at curvature estimation from 3D range image for computer vision applications.

#### 2.3.1.1 Window Operator

Range data provides a rectangular array of sample data, usually in the form of pixels. Many of the methods operate on an N×N window centered at a point, where N is an odd integer, typically 5 or 7 [25]. This window provides a natural orthogonal parameterization and well-defined diagonals. Mean and Gaussian curvature can be computed directly from a range image using window operators that yield least squares estimates of first and second partial derivatives with respect to these preferred directions, or directly from the array of sample data.

Gaussian and mean curvature value at any point ( i, j, g(i, j)) in the range image can be evaluated using window operators as follows[7, 12]:

$$E= 1+g_u(i, j) \tag{2.3}$$

$$F= g_u(i, j)\, g_v(i, j) \tag{2.4}$$

$$G=1+ g_v(i, j) \tag{2.5}$$

$$L=\frac{g_{uu}(i,j)}{\sqrt{1+g_u^2(i,j)+g_v^2(i,j)}} \tag{2.6}$$

$$N=\frac{g_{vv}(i,j)}{\sqrt{1+g_u^2(i,j)+g_v^2(i,j)}} \tag{2.7}$$

$$M=\frac{g_{uv}(i,j)}{\sqrt{1+g_u^2(i,j)+g_v^2(i,j)}} \tag{2.8}$$

where $g_u(i, j)$, $g_v(i, j)$, $g_{uv}(i, j)$, $g_{uu}(i, j)$, and $g_{vv}(i, j)$ are the partial derivative estimates, and can be computed via the appropriate 2-D convolutions (denoted*):

$$g_u(i, j)=D_u * g(i, j) \tag{2.9}$$

$$g_v(i, j)=D_v * g(i, j) \tag{2.10}$$

$$g_{uv}(i, j)=D_{uv} * g(i, j) \tag{2.11}$$

$$g_{uu}(i, j)=D_{uu} * g(i, j) \tag{2.12}$$

$$g_{vv}(i, j)=D_{vv} * g(i, j) \tag{2.13}$$

where $D_u$, $D_v$, $D_{uv}$, $D_{uu}$ and $D_{vv}$ are the equally weighted least-square derivative estimation window operators and can be computed as:

$$D_u=d_0d_1^T \tag{2.14}$$

$$D_v=d_1d_0^T \tag{2.15}$$

$$D_{uv}=d_1d_1^T \tag{2.16}$$

$$D_{uu}=d_0d_2^T \tag{2.17}$$

$$D_{vv}=d_2d_0^T \tag{2.18}$$

where $d_0$, $d_1$, and $d_2$ are called the column vectors which depend on the size of window operator. For 7 ×7 windows, these are given by,

$$d_0= \frac{1}{7}[\,1\ 1\ 1\ 1\ 1\ 1\ 1]^T \tag{2.19}$$

$$d_1= \frac{1}{28}[\,-3\ -2\ -1\ 0\ 1\ 2\ 3]^T \tag{2.20}$$

$$d_2= \frac{1}{84}[5\ 0\ -3\ -4\ -3\ 0\ 5]^T \tag{2.21}$$

$$H(i,j)=\frac{(1+g_v^2(i,j))g_{uu}(i,j)+(1+g_u^2(i,j))g_{vv}(i,j)-2g_u(i,j)g_v(i,j)g_{uv}(i,j)}{2\left(\sqrt{1+g_u^2(i,j)+g_v^2(i,j)}\right)^3} \quad (2.22)$$

$$K(i,j)=\frac{g_{uu}(i,j)g_{vv}(i,j)-g_{uv}^2(i,j)}{\left(1+g_u^2(i,j)+g_v^2(i,j)\right)^2} \quad (2.23)$$

### 2.3.1.2 Surface Normal Method

A local technique is used for estimating the surface curvatures by approximating the derivative of the surface normal [7- 9]. A simple estimate of the surface curvature at point p in the direction of point q is given by

$$k(p,q)=\frac{\|n_p-n_q\|}{\|p-q\|}\times s(p,q) \quad (2.24)$$

where $s(p,q)=1$ if $\|p-q\|\le\|n_p-n_q\|$ and $s(p,q)=-1$ otherwise.

Here, $n_p$ and $n_q$ are the unit normal vector at points $p$ and $q$, respectively. The sign factor $s(p,q)$ states that if two surface normals, $n_p$ and $n_q$ at point $p$ and $q$, respectively, approach each other as $q$ approaches $p$ then the surface curvature has a negative value indicating a concave surface, else, the surface curvature has a positive value indicating a convex surface. Using the value $k(p, q)$ one can compute the mean and Gaussian curvature. Let $\Omega(p)$ be the set of points in the neighborhood of point p. The mean curvature $H$ and Gaussian curvature $K$ are given by

$$H=\frac{\left(k^{\min}(p)+k^{\max}(p)\right)}{2} \quad (2.25)$$

$$K=k^{\min}(p)\times k^{\max}(p) \quad (2.26)$$

where $k^{min}(p)=k(p,q_0)=\min_{q=\Omega(p)} k(p,q)$ and $k^{max}(p)=k(p,q_1)=\max_{q=\Omega(p)} k(p,q)$. A $3\times3$ neighborhood is used when computing the mean and Gaussian curvatures.

The mean and Gaussian curvature can be computed by a combination $k^{max}(p)$,

$k^{min}(p)$ and fixed-weighted networks. Each curvature computing unit receives these inputs from a center pixel (or point) and one of its 8 nearest neighbors, and computes a curvature value. The $k^{max}(p)$ and $k^{min}(p)$ collect the curvature values to find the maximum and minimum curvature values respectively, from which Gaussian and mean curvature are calculated.

#### 2.3.1.3 Four-Direction Curvature Method

Fan and Medioni [14] have proved that at every point computing the principal curvature ( $k_1,k_2$) and their orientation $a$ is equivalent to computing curvature in four different directions $45^0$ apart *( $k_0$, $k_{45}$,, $k_{90}$,, $k_{135}$).*

### 2.3.2 Curvature Estimation on Triangular Meshes

Curvature estimation methods have been developed specifically for meshes [21-25]. Meshes have a more general structure than range images. Mesh representations have adjacency information embedded in mesh connectivity, but without any regular organization. Generally speaking, curvature calculation methods on triangular mesh can be classified into three categories: fitting method, discrete estimation of curvature and curvature directions and estimation of a curvature tensor from which curvature and curvature directions. Fitting methods include mainly parameterization method and quadric fitting method. In this work, a simplified quadric fitting method (parabolic fitting) is used.

#### 2.3.2.1 Parameterization Method

Parameterization methods utilize a local 3D coordinate frame with its origin at the vertex [6]. The normal vector at the vertex is frequently chosen as one axis of this frame. The vertex normal can be computed as the average of the face normals for the faces adjacent to the vertex, with various weightings applied, or as the normal to plane that best fits the vertex and some number of nearby vertices. Yang and Lee [6] used a parametric quadric approximation method to calculate Gaussian and mean curvature:

$$r(u,v) = \sum_{j=0}^{2}\sum_{i=0}^{2} Q_{ij}u^i v^j \tag{2.27}$$

Or the matrix form is

$$r(u,v) = [1\ u\ u^2]\ Q \begin{bmatrix} 1 \\ v \\ v^2 \end{bmatrix} \tag{2.28}$$

where Q is a 3x3 matrix with vector-valued elements $Q_{ij}$

$$Q=\begin{bmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{21} & Q_{22} & Q_{23} \\ Q_{31} & Q_{32} & Q_{33} \end{bmatrix} \tag{2.29}$$

$$r(u,v) = (x(u,v), y(u,v), z(u,v)) \tag{2.30}$$

$$Q=\{Q_{ij}\}=(a_{ij}, b_{ij}, c_{ij}) \tag{2.31}$$

Here, N+1 points $P_l$ $(x_l,y_l,z_l)$ are given and their associated parameter values $(u_l, v_l)$, l=0...N. N>8 is required for local surface approximation.

A set of N(=m×n) grid points can be parameterized with the Eq.(2.32)

$$u_l=u_{ij}=\frac{\sum_{k=1}^{i-1}\left|P_{k+1,j}-P_{k,j}\right|}{\sum_{k=1}^{m-1}\left|P_{k+1,j}-P_{k,j}\right|} \tag{2.32}$$

$2<i<m,\ 1<j<n$ with $u_{1j}=0$

$$v_l=v_{ij}=\frac{\sum_{k=1}^{j-1}|P_{i,k+1}-P_{i,k}|}{\sum_{k=1}^{n-1}|P_{i,k+1}-P_{i,k}|} \quad (2.33)$$

$1<i<m,\ 2<j<n$ with $v_{j1}=0$.

where $l=(i-1)\times m+(j-1)$. The vectors are introduced

$W=(u^0v^0,u^0v^1,u^0v^2,u^1v^2,\ldots u^2v^2)^T$ (2.34)

$a=(a_{00},a_{01},a_{02},a_{10},\ldots,a_{22})^T$ (2.35)

$b=(b_{00},b_{01},b_{02},b_{10},\ldots,b_{22})^T$ (2.36)

$c=(c_{00},c_{01},c_{02},c_{10},\ldots,c_{22})^T$ (2.37)

$x=W^Ta, \qquad y=W^Tb, \qquad z=W^Tc$ (2.38)

$Z=(Z_x,Z_y,Z_y)$ (2.39)

$Z_x=[x_0,x_1\ldots x_N]^T, \qquad Z_y=[y_0,y_1,\ldots y_N]^T, \qquad Z_z=[z_0,z_1,\ldots z_N]$ (2.40)

$M=[W_0^T,\ldots W_N^T]^T$ (2.41)

M is a (N+1)x9 matrix, where N>8 is supposed.

Fit equation of surface is MQ.

$E=Z-MQ$ (2.42)

Use least-squared method

$$\|E\|=\sum_{i=0}^{N}e_i^2 \quad (2.43)$$

The principal curvature $k_{n1}$ and $k_{n2}$ can be obtained from

$|G|\,k_n^2-(g_{11}d_{22}+d_{11}g_{22}-2g_{12}d_{12})k_n+|D|=0$ (2.44)

### 2.3.2.2 Quadric Fitting Method

Various forms of the quadratic function have been fitted to the range data and mesh representation. For a general second-order polynomial with six coefficients, applied to a height function:

$$z_i=f(u_i, v_i)=Au_i^2+Bu_iv_i+Cv_i^2+Du_i+Ev_i+F \tag{2.45}$$

where $(u_i, v_i)$ is the parametric location of the ith point in the tangent plane, and $z_i$ is the height of the point above (or below) the tangent plane. N is the number of vertices being fit and i runs from 1 to N. The coefficient A through F are determined by solving a least-square problem. In this work, the augmented Darboux frame proposed by Sander and Zucker [21] is used. This Darboux frame will be described in detail in the next chapter.

# Chapter 3
# Point Data Segmentation

In order to make the surface reconstruction process more efficient, the point data need to be divided into regions. When segmenting the scanned point data using a neural network, the geometric shape of a scanned part should be taken into consideration. In order to extract geometric information, such as point normal and curvature (Gaussian, mean and principal curvature) values, from the point data, additional operations are required. Once the geometric information of a part is obtained, it can be used for data reduction, segmentation and other application.

## 3.1 Kohonen's Self-Organizing Feature Map

Self-organizing approaches attempt to develop a network structure on the basis of given sample data. One popular approach is clustering and mode separation. The objective is to design a network that can discover clusters of similar patterns in the data without supervision, by computing similarity. If the network can encode these types of data, by assigning nodes to clusters in some way, then it is said to undergo both self-organizing and unsupervised learning. Teuvo Kohonen, in the early 1980s, developed an algorithm to mimic the brain's ability to organize itself in response to external stimuli. He called his algorithm a self-organizing feature map. The alternative neural learning structure involving networks that perform dimensionality reduction through conversion of feature space to yield topologically ordered similarity graphs or maps or clustering diagrams with potential statistical interpretation. In addition, the training algorithm implements a form of local competitive learning [26]. A typical SOFM structure is shown in Figure 3.1. It consists of two layers, a layer of the input nodes and a competitive layer consisting of neural units called Kohonen's units. A weight vector is associated with each connection from the input layer to a neural unit in the output layer. The neural units in the competitive and cooperative layer are organized in a regular geometric structure. Complex relationships in problems requiring a higher-dimension of the weight vectors usually require higher-dimensional lattices to sort themselves out. However, biological inspiration and practical processing considerations typically limit the lattice dimension to 2 or 3.

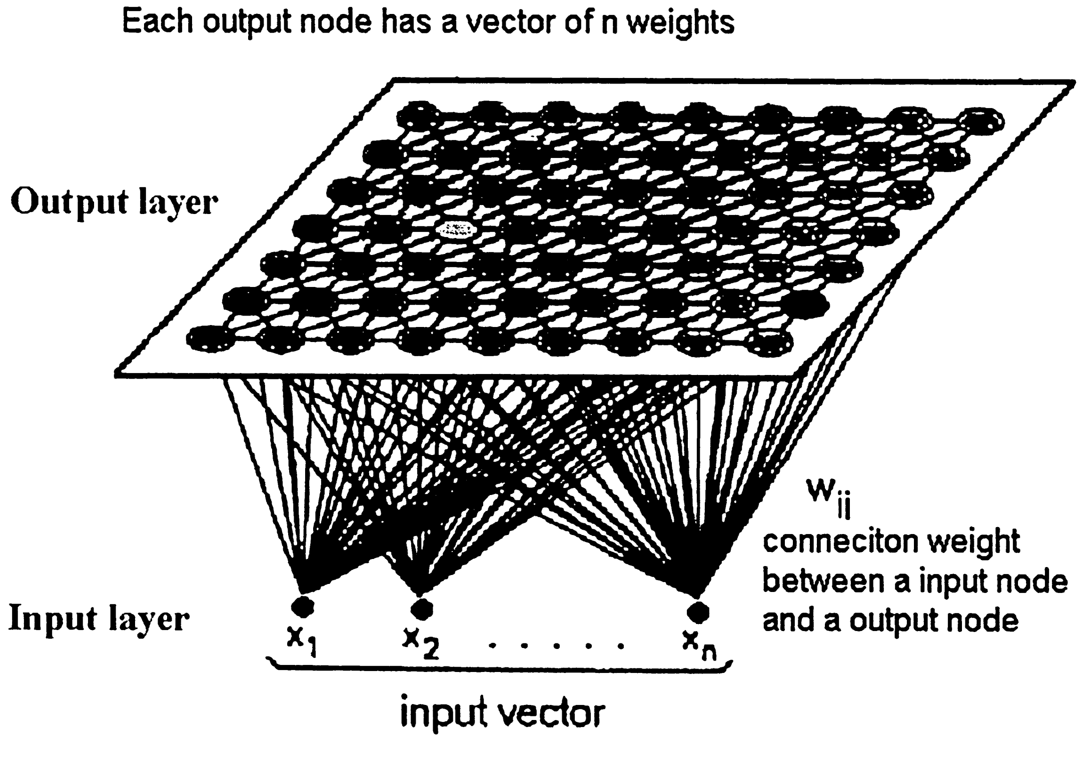

Each output node has a vector of n weights
Output layer
$w_{ij}$
conneciton weight between a input node and a output node
Input layer
$x_1$ $x_2$ . . . . . $x_n$
input vector

Figure 3.1 Kohonen's Self-organization Feature Map [26]

### 3.1.1 Formation Process of SOM

After initialization, Kohonen's SOMs result from the combination of three basic processes:

1. Competition

For a given input pattern, each neuron in the competitive layer receives a sum of weighted inputs from the input layer. All the neurons compute an activation function, and the neuron with the largest activation (i.e. with the smallest Euclidean distance between input vector and weight vector) is declared a winner.

2. Cooperation

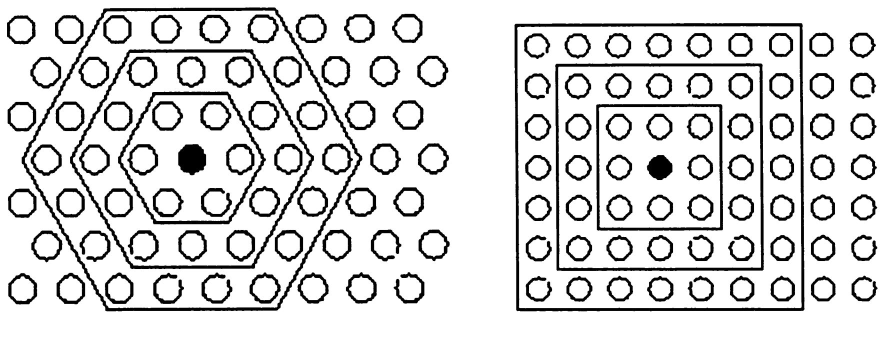
In order to stimulate a topological ordering, every winning neuron in the competitive layer is associated with a collection of other neurons which make up its neighborhood. The winner spreads its activation over a neighborhood of neurons in the map. For a winning neuron, the neurons in its immediate neighborhood excite more than those farther away. Topological neighborhood decays smoothly with lateral distance and size of neighborhood shrinks with time.

3. Adaptation

Adaptation is applied to all neurons inside the neighborhood of a winning neuron. During training, the winner neuron and its topological neighbors are adapted to make their weight vectors more similar to the input vector that caused the activation. Neurons that are closer to the winner will adapt more heavily than neurons that are further away. Neurons inside the neighborhood are moved a bit closer to the input vector. The magnitude of adaptation is controlled with a learning rate, which decay with time to ensure convergence of the SOM.

### 3.1.2 Topological Neighborhood of SOM

The key feature in Kohonen Self-Organizing Maps is to preserve a topological order in the map so that neighboring neurons respond to "similar" input patterns. The topology of Competitive Layer can be organized in 1 dimension, 2 dimensions, or n dimensions. Typical implementations are a one- or two- dimensional lattice for the purpose of visualization and dimensionality reduction. The lattice type of the array can be defined to be rectangular, hexagonal, or even irregular. Hexagonal is effective for visual display (Figure 3.2). The resulting accuracy of the mapping depends upon the choice of the radius of the neighborhood, learning-rate, and the number of iterations. Kohonen cites the use of 10000 to 100000 iterations as typical. Furthermore, learning-rate should start with a value close to 1 and gradually decrease with time. Therefore, it is reasonable to let the radius of the neighborhood be fairly large (Kohonen suggests half the diameter of the map) and shrink the radius of the neighborhood with time.



a) Hexagonal grid b) Rectangular grid

Figure 3.2 Two examples of topological neighborhood and evolution over iteration

## 3.2 Choice of Feature Vectors

In point cloud segmentation via vector quantization, the homogeneity of a segmented region is enforced by the appropriate choice of feature vectors. Significant surface features can be inferred by examining the values of the surface Gaussian and mean curvature. Arbitrary smooth surfaces can be subdivided into simpler regions of constant surface curvature sign based on the signs of the mean and Gaussian curvature at each point. There are only eight possible surface types surrounding any point on a smooth surface based on surface curvature sign: peak, pit, ridge, valley, saddle ridge, saddle valley, flat (planar), and minimal [12]. Surface normals have strong discriminatory power as surface features. For example, adjacent image pixels (or points) belong to the same "class" (or physical surface) if the normal vectors at those points are close to each other. Otherwise, they belong to different classes. The 3D position coordinate x, y and z are also selected as feature attributes in order to preserve the spatial connectivity of subregions in the final segmented image. Similar to [8, 9], an eight-dimensional vector: $x_i = (x, y, z, n_x, n_y, n_z, H, K)$, is chosen as an input feature vector of each point in the point data. Where the $(n_x, n_y, n_z)$ is the unit normal vector at the point (x, y, z) on the three-dimensional surface. H and K are the mean and Gaussian curvature, respectively, at the point (x, y, z) on the three-dimensional surface.

## 3.3 Calculation of Feature Vectors

The three-dimensional point coordinates (x, y, z) on the surface of an object can be obtained directly from digitized point data.

### 3.3.1 Computation of Normal Vector

References [8, 9] use window operators to calculate the surface normal and curvature values. But window operators are not suitable for multiple viewpoint data. To calculate the normal vector of cloud data, a global model can be constructed from various methods. Milory et al [13] constructed a global model from three orthogonal cross-sections (OCS). The model is constructed by interpolating each range map to create a set of cross-sections at specified intervals in a global frame of reference. The individual OCS models are combined by deleting redundant, overlapping segments and merging the individual models to create a non-redundant, single global wire-frame model. Then Darboux frame theory and weighted least-square surface fitting are used to calculate the normal values and curvature values of the point data. However, the triangulated surface model is a best candidate for the global model. Triangles are the simplest polygons for computations and can be created automatically from multiple range maps, and a triangular mesh is a popular method for the global model. When modeling cloud data for prototyping manufacturing, the STL file format has become the *de facto* standard [15]. An STL file can be generated from a triangulation of digitized data. Most commercial CAD/CAM software systems are capable of generating the STL file directly. References [15, 27-29] describe a few triangulation methods in RE.

To facilitate segmentation, a global triangular mesh is used to delete redundant, overlapping points. Soucy and Laurendeau [30] describe a method using a Venn diagram to identify the overlapping data regions, which is followed by a re-parameterization and merging of regions. Turk and Levory [31] devised an incremental algorithm that updates a data reconstruction by eroding redundant geometry and zippering along the remaining boundaries. Sun and Bradley [32] model the large data set by using a unified, non-redundant triangular mesh. This is accomplished from 3D data points in two steps. Firstly, an initial data thinning is performed, to reduce the copious data set size, employing 3D spatial filtering. Secondly, the triangulation commences utilizing a set of heuristic rules, from a user defined seed point. The spatial filtering parameters are extracted from the cloud data set by a series of local surface patches and the required spatial error between the final triangulation and cloud data.

In this work, digitized point data are saved as a STL file. The triangulation net of this STL file is used directly to calculate point normal and curvature if the 3D point data are accurate. If mesh triangulation is ill-behaved or noise in the cloud data is unacceptable (causing a large error), the commercial reverse engineering software package, *Surfacer* vesion 10.6 is used to preprocess the point cloud, and then the algorithm (as shown in Appendix B) is used to calculate the normal values of the point data. Preprocessing can be implemented by the following steps: First the data are sampled according to a specified distance tolerance. This algorithm removes redundant points from the scanned point data so that no points are closer than the specified tolerance to another point. Second, the point data are sorted by coordinate axis direction or by nearest distance tolerance. Finally a triangulation net is created. The calculation of the point normal is programmed by MATLAB codes (Appendix B).

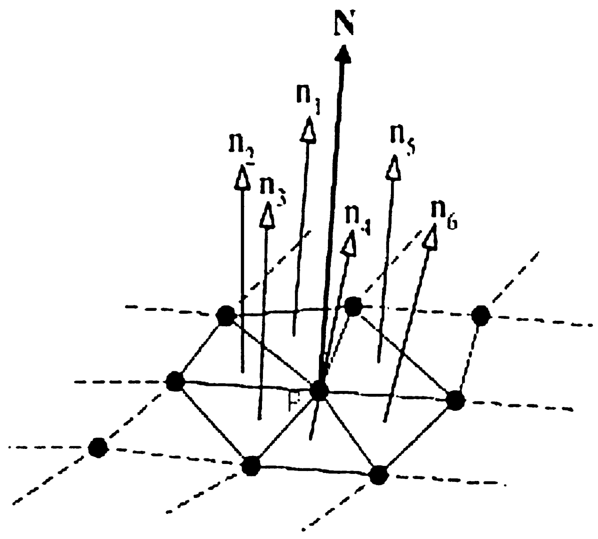


Figure 3.3 Calculation of the normal of a point

After the triangulation, the normal value $N_{Pi}$ of point $P_i$, is estimated from a group of triangles that share a common vertex. Figure 3.3 shows point $P_i$ and its one-ring neighborhood. The normal value of $P_i$ is calculated by Eqn. (3.1).

$$N_{P_i} = \frac{\sum_{i=1}^{m} n_i}{m} \tag{3.1}$$

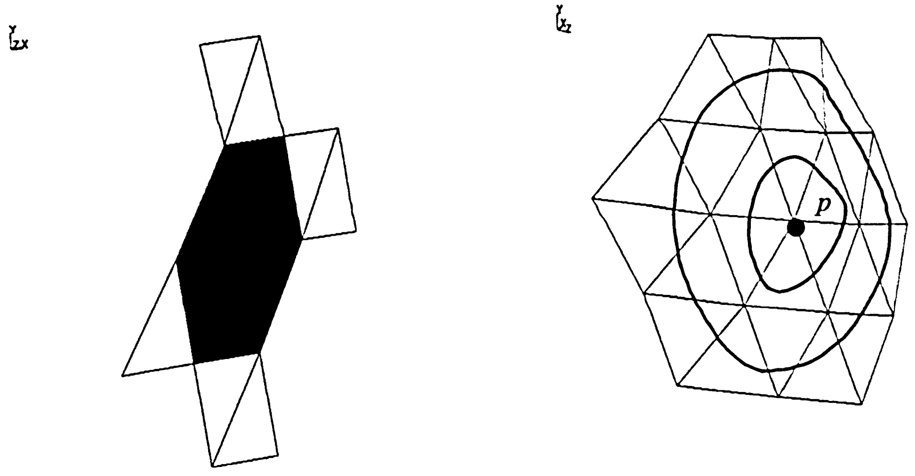
Here m is the number of triangles that share the same vertex. $n_i$ is obtained directly from a list of triangular facet data of the STL file. After calculating the point normals for each point, the normal values are stored into a point data structure, which has x-,y- and z-coordinates and x, y and z normal components.

### 3.3.2 Curvature Calculation

Principal curvatures and local Darboux frame are used during processes which involve extraction of geometric properties from the 3D point data. In order to calculate principal curvatures, polyhedral meshes are used. Curvature estimation is influenced by the ring neighborhood size chosen.

#### 3.3.2.1 Ring Neighborhood

Neighbors are defined as vertices that are part of the same face. All of the vertices that are neighbors to, i.e., share a common face with, a given vertex constitute its one-ring neighborhood. A two-ring neighborhood is obtained by adding all one-ring neighbors of one-ring vertices. The fitting methods based on two or higher rings have better overall performance, albeit at a greater computational cost. Accuracies for three ring neighborhoods did not warrant the increase cost due to the size of the fitting problem [25]. Therefore, in the estimating of the principal curvatures, the highest-ring neighborhood is chosen as a two-ring neighborhood. One-ring and two-ring neighborhood of point *p* are shown in Figure 3.4. Figure 3.4 a) shows a point *p* and its one-ring neighborhood, where the triangles are filled.



a) one-ring neighborhood b) two-ring neighborhood

Figure 3.4 Ring neighborhoods in a triangular mesh

### 3.3.2.2 Coordinate Transformation

Coordinate transformation is necessary in many applications. Such transformations take data or equations connected with a Cartesian coordinate system and convert them into similar information in a shifted and rotated coordinate system (see Figure 3.5). Two popular coordinate transformation methods are Direction Cosine Representation and Euler Angle Representation. The second method for describing a coordinate transformation divides the full transformation into a series of three simple rotations, one after another, around well-defined axes (x, y, z). For the rotations, a positive angle indicates a counterclockwise rotation about the axis of revolution. Because a particular transformation is divided into a series of three simple rotations, the mathematical description of the transformation involves multiplying a combination of the three rotation matrices together. It is important to remember the order in which multiple rotations are performed [33].

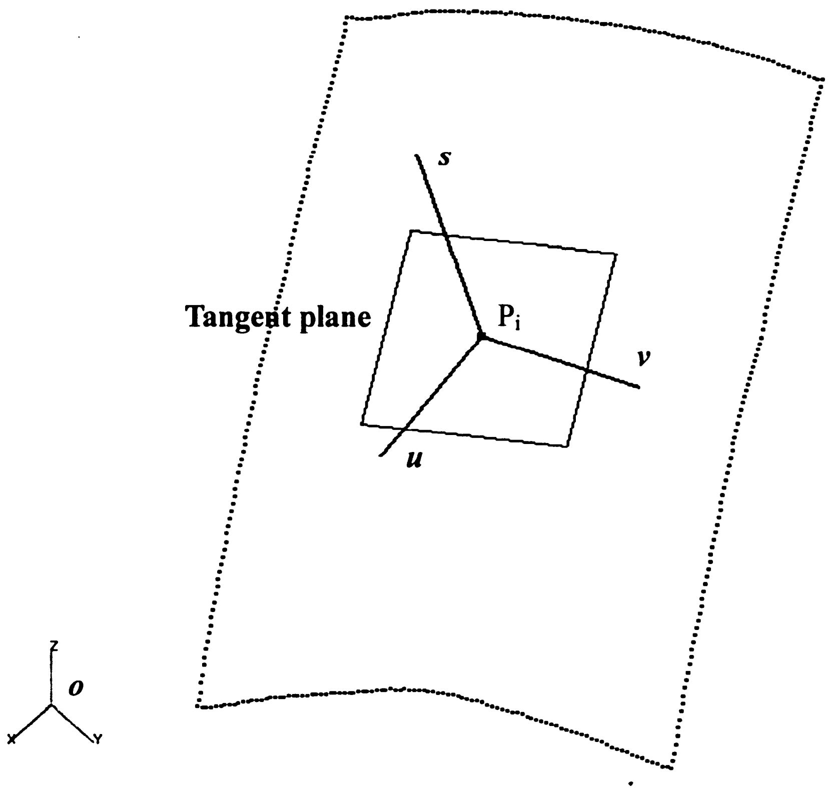


Figure 3.5 Local surface representation-the augmented Darboux frame

To calculate the transformation, Direction Cosine Representation is used. This method relates two coordinate systems (global coordinate system ($O_{xyz}$) and local coordinate system ($P_{suv}$) ) to a set of quantities called the direction cosines. A particular direction $s$ is described by the cosines of the angles that this direction vector makes with the x axis ( $l_{sx}=\cos\theta_{sx}$), y axis ($l_{sy}=\cos\theta_{sy}$), and z axis ($l_{sz}=\cos\theta_{sz}$). The transformation between coordinate systems can be represented in a matrix form as shown in Eqn. (3.2).

$$\begin{pmatrix} u \\ v \\ s \end{pmatrix} = \begin{pmatrix} l_{ux} & l_{uy} & l_{uz} \\ l_{vx} & l_{vy} & l_{vz} \\ l_{sx} & l_{sy} & l_{sz} \end{pmatrix} \bullet \begin{pmatrix} x-x_p \\ y-y_p \\ z-z_p \end{pmatrix} \qquad (3.2)$$

Where the origin of the (x, y, z) system is O=(0, 0, 0) and the origin of the (s, u, v) system is p=($x_p$, $y_p$, $z_p$). Equation 3.2 relates two coordinate systems by first rotating the initial coordinate system and then translating it. The rotation matrix elements are the dot products of the axis relative to each other which are the respective direction cosines, as indicated by the subscripts [34]. Note that there are actually only six distinct quantities required to specify the transformation matrix completely. The remaining three values in the matrix can then be inferred using geometrical relationships among the angles (i.e. the fact that $l_{ux}^2 + l_{vx}^2 + l_{sx}^2 = 1$ and corresponding statements involve the y and z axes) [35].

### 3.3.2.3 Darboux Frame Theory

To calculate the principal curvatures of point data, Darboux frames are used as a local representation for a local surface at each point [13, 21, 32]. Let the local neighborhood of a point $P_i$ on a surface S be described by the parabolic quadric:

$$S(u, v) = au^2 + buv + cv^2 \tag{3.3}$$

Where $P_i$ is at the origin, and the S axis is aligned with the local surface normal N, as shown in Figure 3.5. The *u* and *v* axes lie in the tangent plane. The direction of the *u* and *v* axis can assume an arbitrary direction in the tangent plane. The surface normal is obtained from a simple planar fit to the local neighborhood (the one-ring neighborhood for accurate 3D cloud data, two-ring neighborhood for noise data or ill-behaved mesh triangulation). The local coordinate system ($P_{suv}$) is obtained from the world coordinate system ($O_{xyz}$) by the following 3D transformations:

1. Translate O to $P_i$.
2. Rotate Z axis of the world coordinate system so that it aligned with the S axis.
3. Take x, y axis as *u*,*v* axis.

The local neighborhood of a point $P_i$ in the Darboux frame ($P_{suv}$) is represented as ($u_i$ ,$v_i$ ,$s_i$). $s_i$ is the height of the point above (or below) the tangent plane. Here, i runs from 1 to n, where n is the number of vertices being fit. The coefficients (a, b, c) are calculated by solving the local least-squares problem [22]:

$$AX=B \tag{3.4}$$

$$A=\begin{bmatrix} u_1^2 & u_1v_1 & v_1^2 \\ u_2^2 & u_2v_2 & v_2^2 \\ \vdots & \vdots & \vdots \\ u_n^2 & u_nv_n & v_n^2 \end{bmatrix} \tag{3.5}$$

$$X=\begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.6}$$

$$B=\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix} \tag{3.7}$$

A single unique solution usually does not exist when the number of equations and number of unknowns differ. However, with further constraints, a practical solution can usually be found. In MATLAB, when rank(A)=min(r,c), where r and c are the number of rows and columns in A, respectively, and there are more equations than unknowns(r>c) (i.e., the overdetermined case), a division operator (X=A\B) automatically finds the solution that minimizes the norm of the squared residual error ($e$=A•x-B). This solution is of great practical value and is called the least-square solution.

Following the convention of Ferrie and Lagarde [22], the augmented Darboux frame at each point p is given by $\vartheta$(p)=(p, $m_1$, $m_2$, n, $k_1$, $k_2$) which completely describes the surface at p. The principal directions $m_1$ and $m_2$ are the directions in which the surface normal curvature takes on a minimum and maximum value, denoted by scalars $k_1$ and $k_2$ respectively.

$$m_1=\begin{cases}(c-a+\sqrt{(a-c)^2+b^2},-b) & a<c\\ (b,c-a-\sqrt{(a-c)^2+b^2}) & a\geq c\end{cases} \tag{3.8}$$

$$m_2=\begin{cases}(b,c-a+\sqrt{(a-c)^2+b^2}) & a<c\\ (c-a-\sqrt{(a-c)^2+b^2},-b) & a\geq c\end{cases} \tag{3.9}$$

$$k_1=a+c-\sqrt{(a-c)^2+b^2} \tag{3.10}$$

$$k_2=a+c+\sqrt{(a-c)^2+b^2} \tag{3.11}$$

In terms of the principal curvatures ($k_1$, $k_2$), Gaussian curvature K and mean curvature H can be written

$$K=k_1\,k_2 \tag{3.12}$$

$$H=(k_1+k_2)/2 \tag{3.13}$$

Principal curvatures, Gaussian and mean curvature are calculated by the Darboux frame theory and the local least-square fitting. The task is accomplished by developing MATLAB codes (Appendix B).

## 3.3.3 Modification of Feature Vector

It has sometimes been suggested that the components of the input feature vector be normalized (scaled) to have unit variance before it is used in the SOFM. Normalization is not necessary in principle, but if the feature vectors are normalized, then two schemes ($x_i$=(x, y, z, $n_x$, $n_y$, $n_z$, H, K)) and $w_j$=($w_x$, $w_y$, $w_z$, $w_{nx}$, $w_{ny}$, $w_{nz}$, $w_H$, $w_k$) fall within a specified range (both range from 0 to 1). This assures that for each component, the difference between two samples contribute approximately an equal amount to the summed distance measure between an input sample and reference vector.

Normalization improves the speed and exactness of the segmentation because the similarity measure usually loses identity of component differences via a summation, or treats all components equally, the components must contribute approximately as much to the similarity measure. Otherwise, a component with large variance would shadow components with small variance and thus only the components with large variance would contribute to the distance measure used as a similarity measure. Normalization gives each of the input feature vectors an equal importance and prevents premature saturation of activation functions. Therefore, feature vectors need to be modified before they are inputted into SOFM. An input feature vector is normalized by Eqn. (3.14)-(3.21).

Normalize point coordinate

$$x\_new_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{3.14}$$

$$y\_new_i = \frac{y_i - y_{\min}}{y_{\max} - y_{\min}} \tag{3.15}$$

$$z\_new_i = \frac{z_i - z_{\min}}{z_{\max} - z_{\min}} \tag{3.16}$$

Normalize point normal

$$x\_new_{ni} = \frac{x_{ni}}{\sqrt{x_{ni}^2 + y_{ni}^2 + z_{ni}^2}} \tag{3.17}$$

$$y\_new_{ni} = \frac{y_{ni}}{\sqrt{x_{ni}^2 + y_{ni}^2 + z_{ni}^2}} \tag{3.18}$$

$$z\_new_{ni} = \frac{z_{ni}}{\sqrt{x_{ni}^2 + y_{ni}^2 + z_{ni}^2}} \tag{3.19}$$

Normalize point curvature

$$H\_new_i = \frac{H_i - H_{\min}}{H_{\max} - H_{\min}} \tag{3.20}$$

$$K\_new_i = \frac{K_i - K_{\min}}{K_{\max} - K_{\min}} \tag{3.21}$$

## 3.4 SOFM for Point Cloud Segmentation

Kohonen's algorithm creates a vector quantizer by adjusting weights from common N input node to M output node, arranged in a grid. Output nodes are extensively interconnected with many local connections. Continuous-values (point by point) input vectors are presented sequentially without specifying the desired output. After enough input vectors are presented, weights will specify cluster or vector centers that sample the input space such that the point density function of the vector trends to approximate the probability density function of the input vectors. Moreover, the weights are organized such that topologically close nodes are sensitive to inputs that are physically similar. Output nodes thus become ordered in a natural manner [36-38]. The weight vector $w_j=(w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_k)$ of each unit is the representative vector of a region where $(w_x, w_y, w_z)$ is the coordinates of the center of region in the three-dimensional space and $(w_{nx}, w_{ny}, w_{nz})$ is the representative unit normal vector of the region. The components $w_H$ and $w_k$ are the representative mean and Gaussian curvature of the region.

The Kohonen's training algorithm can be summarized as follows:

1. Initialize network

Initialize weights $w_j=(w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_k)$ with small random value (range from 0 to1). Set the initial radius of the neighborhood around each node to be large in order to avoid a dead neuron which will never fire during the training process. Initialize learning rate, number of iteration rounds and iteration epochs for each round. Usually there are three different types of network initializations.

a. Random initialization: Random initialization means simply that random values are assigned to codebook vectors. This is the case if nothing or little is known about the input data at the time of the initialization.
b. Initialization using initial samples: Initial samples of the input data set can be used for codebook vector initialization. This has the advantage that the points automatically lie in the same part of the input space with the data.
c. Linear initialization: One initialization method takes advantage of the principal component analysis of the input data. The codebook vectors are initialized to lie in the same input space that is spanned by two eigenvectors corresponding to the largest eigenvalues of the input data. This has the effect of stretching the Self-Organizing Map to the same orientation as the data having the most significant amounts of energy.

For this algortithm, random initialization is used.

2. Present input vector

Present input vector $x_i$ at time t ($0<t<n$, where $t=0, 1, 2\ldots$is an integer the discrete-time coordinate. n is the number of iterations defined by the user) to all neuron in the network simultaneously.

3. Calculate winning neuron

Compute the distance $d(x_i, w_j)$ between the input and each output node using the Eqn. (3.10). Winner neuron is the node with the smallest Euclidean distance between input vector and weight vector.

$$d(x_i, w_j)=a\bullet\|x_p-w_p\|+b\bullet\|x_n-w_n\|+c\bullet|H-w_H|+d\bullet|K-w_k| \qquad (3.10)$$

where a is the weight of point coordinate distance, b is the weight of normal distance. c and d are weights of curvature distance. The actual weights are determined experimentally and are given in Chapter 5.

where:

$x_i=(x, y, z, n_x, n_y, n_z, H, K)=(x_p, x_n, H,K)$

$x_p=(x, y, z)$

$x_n=(n_x, n_y, n_z)$

$w_j=(w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_k) = (w_p, w_n, w_H, w_k)$

$w_p=(w_x, w_y, w_z)$

$w_n =( w_{nx}, w_{ny}, w_{nz})$

4. Update the weights

After a winning neuron is determined, the weight vectors connecting the input layer to the winning neuron and its neighboring neurons are updated according to the learning rule.

$$w_j(t+1)=\begin{cases} w_j(t)+h_{qj}\left[x(t)-w_j(t)\right] & if\ j\in N_q(t) \\ w_j(t) & if\ j\notin N_q(t) \end{cases} \tag{3.11}$$

$$h_{qj}=a(t)\bullet\exp\left(-\frac{\left\|r_q-r_j\right\|}{2\sigma^2(t)}\right) \tag{3.12}$$

$$a(t)=a_{initial}\bullet\left(\frac{a_{final}}{a_{initial}}\right)^{t/t_{max}} \tag{3.13}$$

$$\sigma(t)=\sigma_{initial}\bullet\left(\frac{\sigma_{final}}{\sigma_{initial}}\right)^{t/t_{max}} \tag{3.14}$$

where $q$ is the location vector of the winning neuron in the output layer, $N_q(t)$ denotes defined neighborhood size in $t$ iteration, $\left\|r_j-r_q\right\|$ is the Euclidean distance between neuron $j$ and $q$. $h_{qj}$ is a smooth neighborhood kernel of the Gaussian function. $\alpha(t)$ is a scalar-valued "learning rate factor" and the parameter $\sigma(t)$ defines the width of the kernel. Both $\alpha(t)$ and the radius of $N_q(t)$ are decreasing monotonically in time (during the ordering process). $\alpha(t)$ and $\sigma^2(t)$ are used to control the amount of learning rate. Effective choices for these functions and their parameters have so far only been determined experimentally [36]. In this work, the learning rate $\alpha(t)$ is decreased from $a_{initial}$ set to 0.9 to $a_{final}$ set to 0.01 so that Kohonen's learning rule ensures that finial convergence to the asymptotic values. $\sigma_{initial}$ and $\sigma_{final}$ are set to 5 and 0.1 respectively.

5. Go to 2

After a certain number cycles (repeat from 2 to 5) as determined experimentally, decrease the size of the neighborhood $N_q(t)$, until neighboring radius is equal to zero and end iteration.

A summary of the learning algorithm of SOFM network is shown in Figure 3.6.

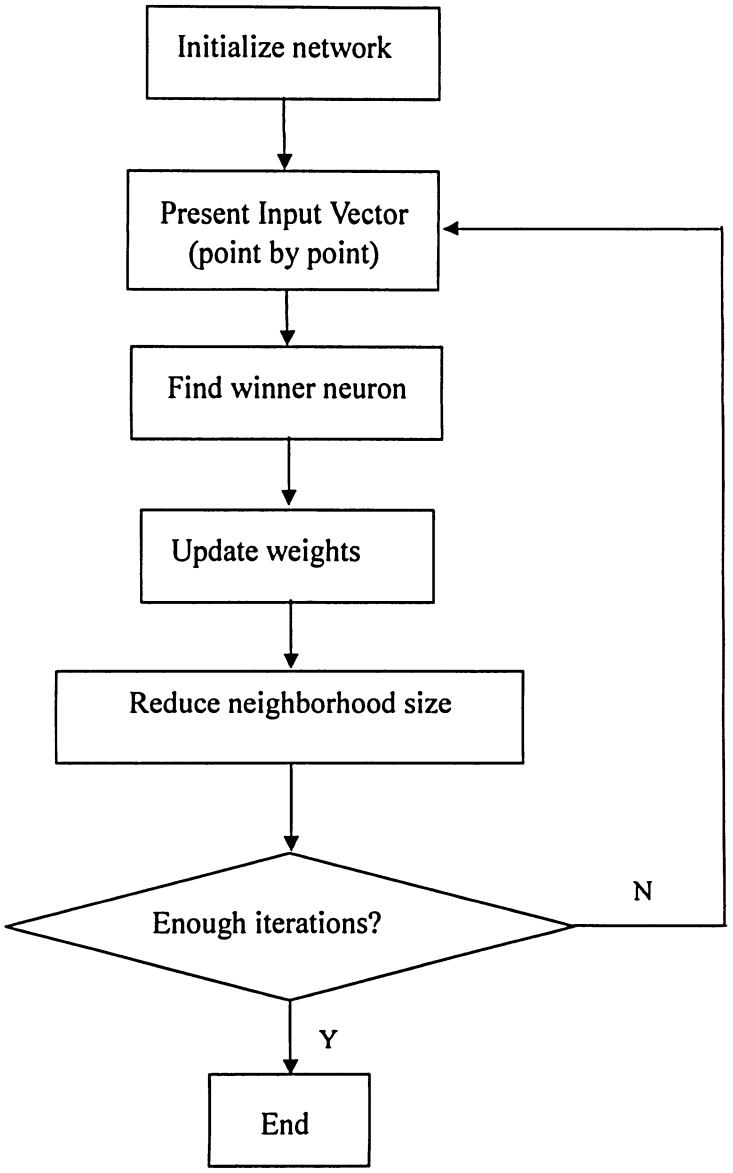


Figure 3.6 SOFM algorithm flowchart

# Chapter 4
# Data Classification

In order to improve the quality of the surface reconstruction, surface geometric primitives commonly used in mechanical parts (plane, cylinder, sphere, cone, torus, etc) are required to be recognized. In this chapter, a back propagation neural network is used to classify data segmented in order to decide to what type of surface each subset of point belongs (e.g., planar, cylindrical).

## 4.1 Introduction

After the point data are segmented into subsets using Kohonen's self-organizing Feature map, each subset corresponds to a unique geometric primitive feature such as plane, cylinder, sphere, cone, torus, or a free-form surface element such as Coons, Bezier, NURBS. Obviously, this sort of information is crucial, since it will determine the quality of the final model to be constructed and have a significant effect on the efficiency of the computations. For example, a set of measured points from a sphere is reconstructed as a free-from surface. This not only decreases the quality of the (sphere) surface but also increases the computational time. Chappuis and Rassineux [39] described a method that surfaces could be represented by the curvature graph shown in Figure 4.1, where $k_1$ and $k_2$ denote the principal curvatures.

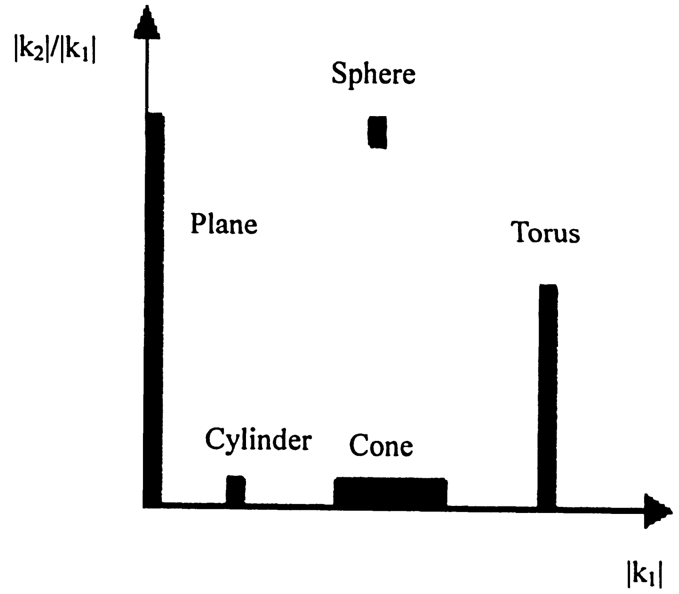
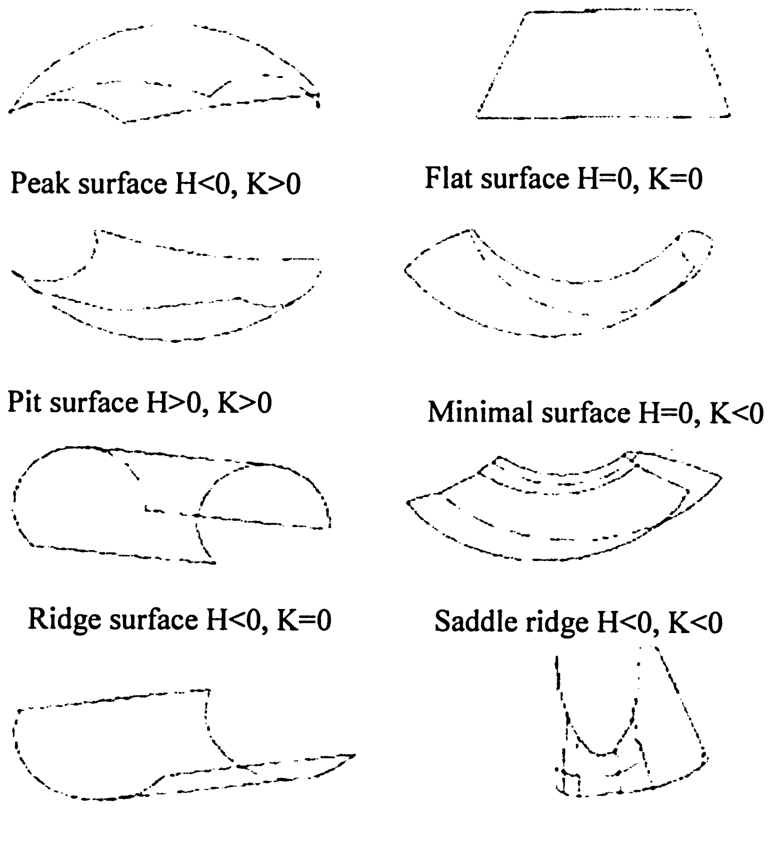


Figure 4.1 Curvature graph classification [39]

Besl and Jain [12] classified point data based on the sign of Gaussian and mean curvature into eight different primitives (Figure 4.2). The classification of the primitives is based on zero values for the Gaussian and mean curvature. However, in practice, exact zeros for curvature can not be obtained due to noise in the data. So, it is a common practice to define a threshold value about the zero for both Gaussian and the mean curvature. Assuming that two threshold values are $K_{zero}$ and $H_{zero}$ respectively, the local surface classification can be performed as follows [4].

1. $K > K_{zero}$ and $H < -H_{zero}$ peak surface
2. $K > K_{zero}$ and $H > H_{zero}$ pit surface
3. $-K_{zero} > K > K_{zero}$ and $H < -H_{zero}$ ridge surface
4. $-K_{zero} > K > K_{zero}$ and $H > H_{zero}$ valley surface
5. $-K_{zero} > K > K_{zero}$ and $-H_{zero} > H > H_{zero}$ flat surface
6. $K < K_{zero}$ and $-H_{zero} > H > H_{zero}$ minimal surface
7. $K < -K_{zero}$ and $H < -H_{zero}$ saddle ridge surface
8. $K < -K_{zero}$ and $H > H_{zero}$ saddle valley surface

The selection of the threshold value for $K_{zero}$ and $H_{zero}$ is very critical. Besl and Jain [12] have selected threshold based on the maxima of both curvatures. Abdalla and Saeid [4] used a back propagation neural network. In this work, a typical surface library includes plane, cylinder and sphere (this library can expand to cone, torus, even free-from surface). Surface recognition is accomplished by a feed-forward neural network.



Peak surface $H<0$, $K>0$
Flat surface $H=0$, $K=0$
Pit surface $H>0$, $K>0$
Minimal surface $H=0$, $K<0$
Ridge surface $H<0$, $K=0$
Saddle ridge $H<0$, $K<0$
Valley surface $H=0$, $K<0$
Saddle valley $H>0$, $K<0$

Figure 4.2 The set of eight primitive surfaces after Besl and Jain [12]

## 4.2 Artificial Neural Network

The neural network that is used has an input layer, a hidden layer, an output layer, weights, bias and a transfer function (Figure 4.3). The inputs are multiplied by weights, bias is added and the transfer function operates on the total to give the output. Generally, linear transfer functions are best suited to linear problems and non-linear transfer function are best suited non-linear problem. Different transfer functions are tested in order to decide which transfer function is best for this work. By trial and error, the hypertangent function is chosen.

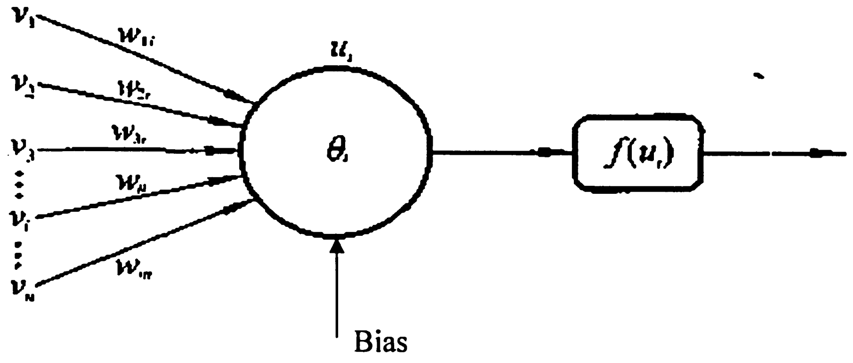


Figure 4.3 Neural network configuration [38]

## 4.3 Back Propagated Neural Network

Back propagation is suitable for non-linearly-separable inputs. Back propagation network is formalized first by Werbos, and later by Parter and by Rummelhart. This network is designed to operate as a multiplayer, feedforward network, using the supervised mode of learning [38].

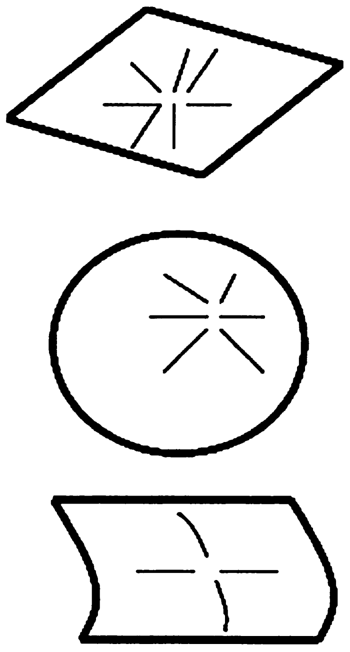
### 4.3.1 Input vector

Both Gaussian, mean curvature (Figure 4.4) and two principal curvatures (Figure 4.5) can define the geometry of the typical surface primitives (i.e., plane, sphere and cylinder). The calculation of these curvatures is the same as in Section 3.3.2. For this algorithm, two principal curvatures are chosen as an input vector. Due to noise and accuracy of measurement devices, there is an error between the curvatures of the ideal surface and the curvatures of the actual surface. Usually there are many points in the subset of a primitive surface in reverse engineering. So, the curvature distribution of point data tends to possess a normal probability distribution according to the central limit theorem [40]. $\mu$ and $\sigma$ are parameters representing the population mean and standard deviation. The mean $\mu$ locates the center of the normal distribution.

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.1}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n} \tag{4.2}$$

where $\chi$ denotes Gaussian, mean , maximum or minimum curvature, n denotes the total number of points in a subset. If the population standard deviation $\sigma$ is large, it shows that there are big errors in this subset. This segmented subset needs to be re-segmented. If the value of $\sigma$ is small, the arithmetic mean of the curvatures is taken as the input vector.



Plane: all directions

$K_1 \; = \; K_2 \; = \; 0$

Sphere: all directions

$K_1 \; = \; K_2 \; < \; 0$

Cylinder

$K_1 \; < \; 0 \quad K_2 \; = \; 0$

Figure 4.4 Principal curvatures of plane, sphere and cylinder

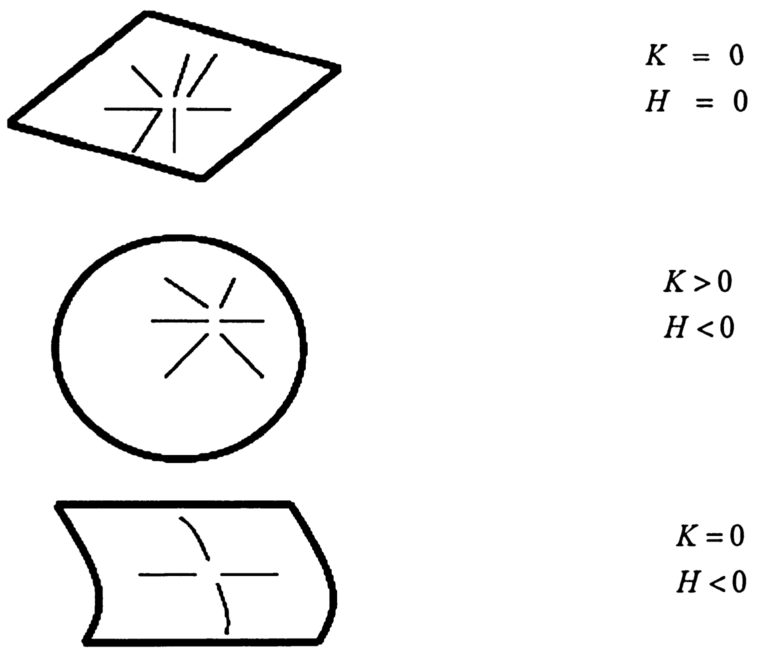


Figure 4.5 Gaussian and mean curvatures of plane, sphere and cylinder

### 4.3.2 Hidden layer

One important issue with respect to a multilayer neural network is how to determine the number of the hidden units required to perform classification. The number of elements in a hidden layer can be determined by experimentation. Increasing the number of elements in the hidden layer resulted in better mapping, however, at the penalty of increasing training time. In this work, for the surface primitive recognition, six neurons are used in the hidden layer (Figure 4.6).

### 4.3.3 Output Layer

This output layer has three output neurons for three kind of surface primitives. The desired output for three kind of surface primitive formed the three dimensional vectors are as follows:

| | |
|---|---|
| plane: | 1 0 0 |
| cylinder: | 0 1 0 |
| sphere: | 0 0 1 |

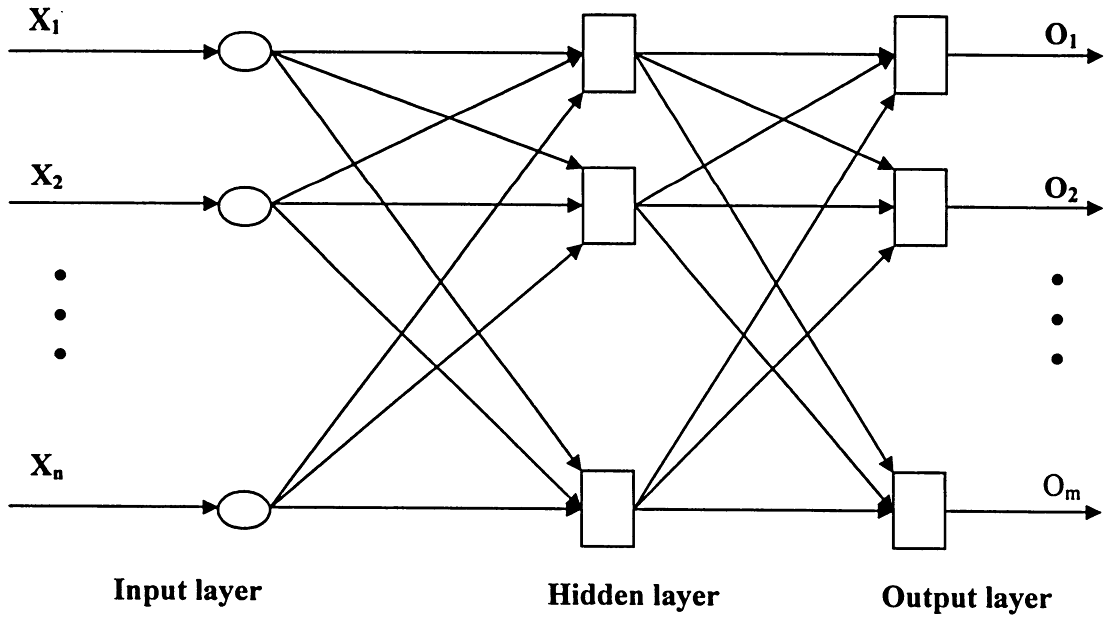


Figure 4.6 Neural network for 3D data classification

### 4.3.4 Back Propagation Training Algorithm

Learning via back propagation involves the presentation of pairs of input and output vectors. The actual output for a given vector is computed with the desired or target output. If there is no difference, no weights are updated; otherwise, the weights are adjusted to reduce the difference. This learning method essentially uses a gradient search technique to minimize the cost function, which is equal to the mean square difference between the desired and the actual outputs. Often, the back propagation network is initialized by setting random weights and thresholds, and the weights are updated with each iteration to minimize the mean-squared error [41]. The back propagation learning algorithm for a three-layer network is as follows [42].

Let:

$\eta_{(1,i)}$=input neural value at neural location i.

$\eta_{(2,j)}$=hidden neural value at neural location j

$\eta_{(3,i)}$=output neural value at neural location i.

$w_{(1,i)(2,j)}$=connector weight between neural at the hidden and input layer.

$w_{(2,j)(3,i)}$=connector weight between neural at the hidden and output layer.

where i, j=1 to n, depend on the respective layer as shown in Figure 4.6, therefore, the neuron value on the hidden layer can be calculated as:

$$\eta_{(2,j)} = \sum_{i=1}^{n} w_{(1,i)(2,j)} \bullet \eta_{(1,i)} \qquad (4.3)$$

The output neuron values are depended on the transfer function. The transfer function performs the task to pass the neuron value from the input layer to the hidden layer or from the hidden layer to the output layer to generate the desired output. Different transfer functions are tested for this data classification to decide which transfer function to use. By trial and error, the hypertangent function was chosen.

$$f(x) = \frac{1 - \exp(-\alpha x)}{1 + \exp(-\alpha x)} \qquad (4.4)$$

where: $\alpha$ is typically between 0.01 and 1.0. In this work, $\alpha$=0.6 is used by experimental determination.

The output error is given as follows.

$$E_{output,i} = (desired\_output_i - \eta_{(3,i)}) \qquad (4.5)$$

where $\eta_{(3,i)}$ is the neuron value at the output

$$\eta_{(n,i)} = Fn(\ldots(F2(F1(XpW(1))W(2))\ldots)W(n)) \qquad (4.6)$$

The reflected vector is the product of the error vector, $E_{output,i}$ and the calculated output vector $\eta_{(3,i)}$, it can be calculated as:

$$R_i = E_{output,i} \bullet \eta_{(3,i)} \bullet (1 - \eta_{(3,i)}) \qquad (4.7)$$

The reflected vector is used to calculate the adjustments to the connectors between the j th neuron in the hidden layer and the i th neuron in the output layer.

The adjustment of weights between the ouput and hidden layer can be calculated by

$$\frac{dx_{(2,j)(3,i)}}{dt} = A \bullet R_i \bullet \eta_{(2,j)} \quad (4.8)$$

The error of the hidden layer is given by

$$E_{hidden,j} = \eta_{(2,j)} \bullet (1 - \eta_{(2,j)}) \bullet \sum_{i=1}^{n} R_i \bullet x_{(2,j),(3,i)} \quad (4.9)$$

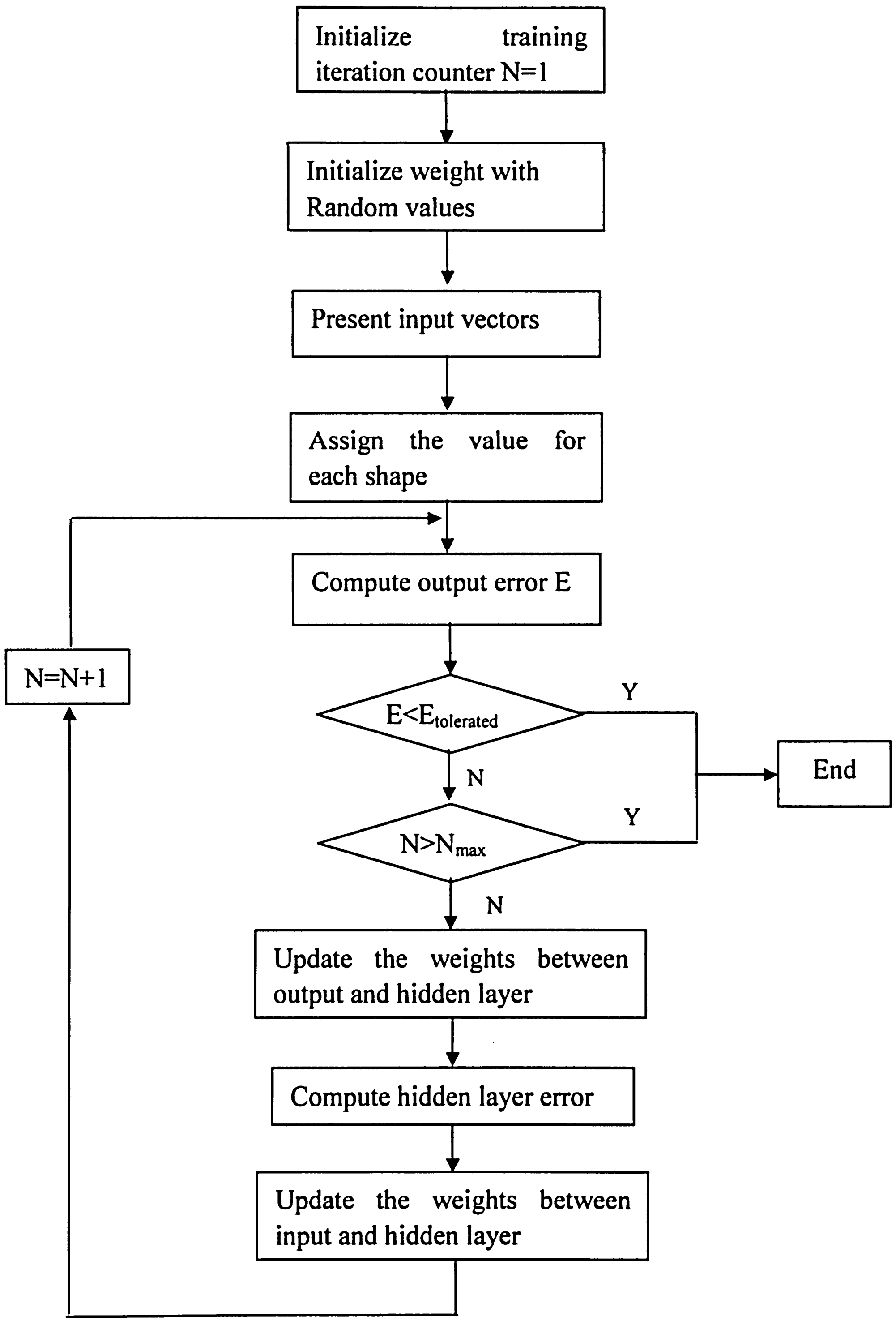
Finally, the adjustment of weights between the input layer and the hidden layer is given by:

$$\frac{dx_{(1,k),(2,j)}}{dt} = B \bullet E_{(hidden,j)} \bullet \eta_{(1,k)} \quad (4.10)$$

Adjusting the two sets of weights between the layers and recalculating the outputs is an iterative process that is repeated for *each training vector* until the errors all fall below a predetermined tolerance level. It flowchart is shown in Figure 4.7.

Large error tolerances will result in a poorly performing neural network, while a very small allowable error will result in excessively long training times. A relatively large error tolerance (0.4) was used at the start, and incrementally lowered to the desired level (0.03) as training is achieved at each succeeding level. This resulted in fewer training iterations than starting out with the desired final error tolerance.

The technique of incrementally lowering the error threshold and learning rates can greatly improve the performance of back propagation training. [43]. The learning rates A and B also need to be determined experimentally. In this work, learning rate starting point 0.4 for A and 0.5 for B. The final learning rate is 0.1 for A and 0.06 for B.



Initialize training iteration counter N=1
Initialize weight with Random values
Present input vectors
Assign the value for each shape
Compute output error E
N=N+1
E<E_tolerated
Y
N
End
N>N_max
Y
N
Update the weights between output and hidden layer
Compute hidden layer error
Update the weights between input and hidden layer

Figure 4.7 Neural network for data classification

## 4.4 Test of Surface Recognition

The neural network was evaluated by testing with different objects, Some results of these tests are shown in Figure 4.8. The back propagation neural network test algorithm of typical surface of machine parts are programmed with MATLAB and are shown in Appendix B.

```
input=[0.002 -0.001];
output =            0.9548     0.3277     0.3277
modify_output =     1          0          0
This is a plane

input= [-10 -0.001];
output =          0.1639     0.8487     0.1681
modify_output =  0          1          0
This is a cylinder

Input = [-20 -20.0009];
output =              0.3326     0.3004     0.9795
modify_output =      0          0          1
This is a sphere

input=[-15 -1];
output =            0.2173     0.2216     0.0544
modify_output =      0          0          0
This is not a typical surface
```

Figure 4.8 Some test result of surface recognition

# Chapter 5
# Experimental Implementation

## 5.1 Introduction

To verify this work, real and synthetic scanned point cloud data were used in the experiments (Fig5.1-5.10). In acquiring the point data, a PICZA 250 3D laser scanner was used. The algorithms (SOFM, normal and curvature calculation) were written using MATLAB. If cloud data are accurate, the algorithms of normal and curvature are directly used. Otherwise, noise need to be removed from the initial scan data before calculating the normal and curvature value of the points. For performing the preprocessing tasks, reverse engineering softeware package, Surfacer version 10.6 was used. In the case of 3D point data, the weights used in the Euclidean distance measure were determined empirically. During the course of computer simulation of SOFM, first, the objects in the input images were observed to have relatively simple surfaces, then complicated surface. The weight ratio a:b:c:d=1:0.7:0.1:0.05 was used for the weighted Euclidean distance measure in all the case of 3D point data.

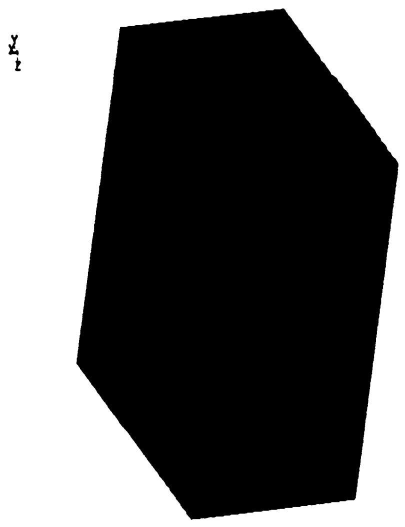
During the scanning, the height-direction pitch and width-direction pitch are both chosen as 0.102 inch (2.59mm).

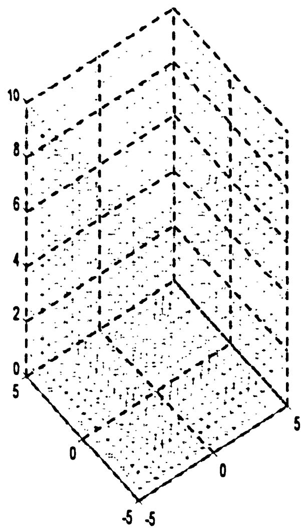
## 5.2 Experiment Results and Analysis

In order to test the performance of the SOFM experimentally, a series of experiments using synthetic and real scanned cloud data are carried out. Each output node denotes if the points belong to the same "class" (physical surface) or different classes. If the surface discontinuity, normal vector difference and curvature difference are high, then the points lie on different surfaces. If the amount of discontinuity is not significant but the normal difference is sufficiently high, then the points belong to different surfaces. The points belong to the same surface, if the normal and curvature difference as well as surface discontinuity are small.

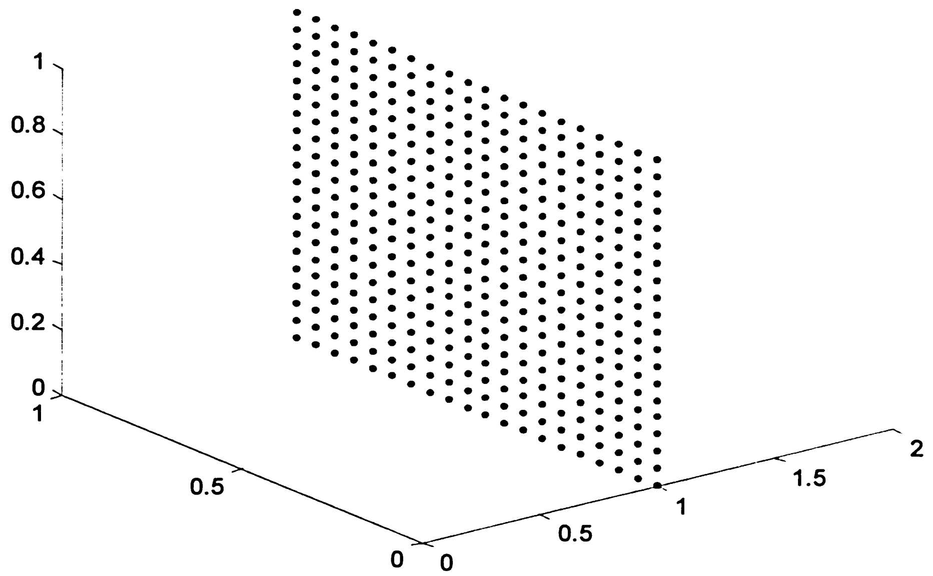
Simple synthetic images were used (Figure 5.1). The use of synthetic images was convenient during the development phase of the SOFM because the ideal output was known. The slopes of test block in Figure 5.2 are 45 degree. The different slopes (30 degree and 70 degree) were also tested.

i) block left top face and j) block right top face in Figure 5.3 have same width, but i) show wider than j) that is because the axis in i) is from 0 to 0.5 but the axis in j) is from 0 to 1 so the scale in i) is twice than j), same reason can explain why h) and i) in Figure 5.4 shows different even though they are in same dimension. Figure 5.4 is different from Figure 5.3. The cloud data in Figure 5.4 are synthetic. During scanning, the groove left face and the groove right face of the block in Figure 5.3 can not be scanned (jump edges without point data). The two faces are synthetic in Figure 5.4 and then the point data are segmented by SOFM.


a) Test block

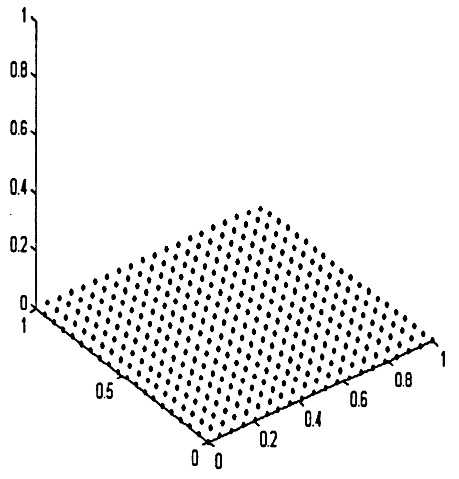


b) Cloud data

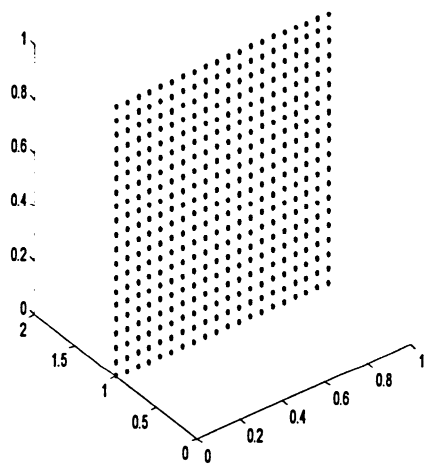


c) Right face

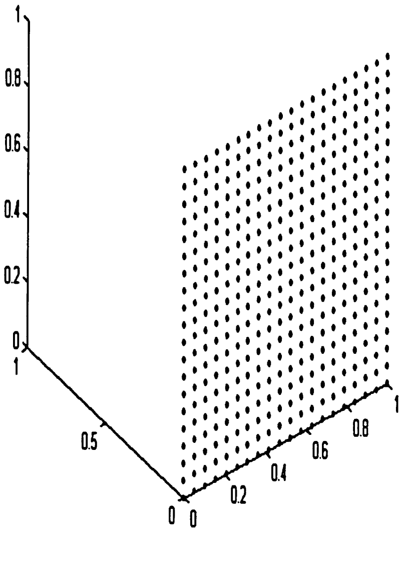
Figure 5.1 Test bock and segmented results



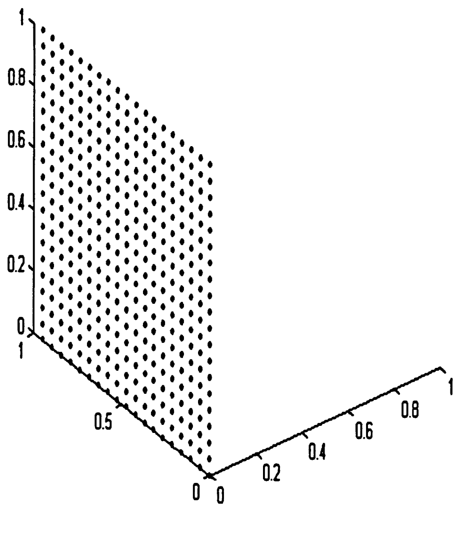
d) Bottom face



e) Back face

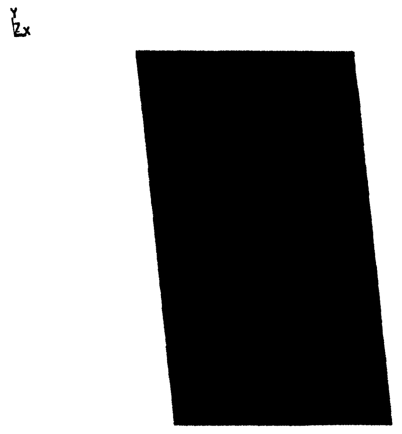


f) Front face

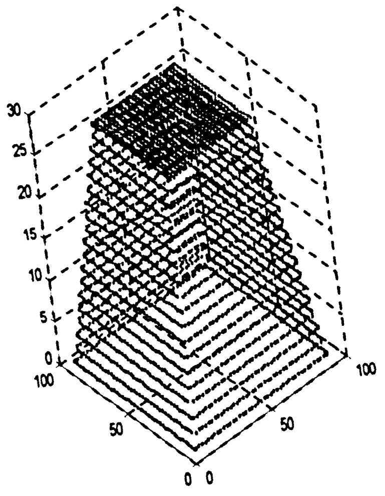


g) Left face

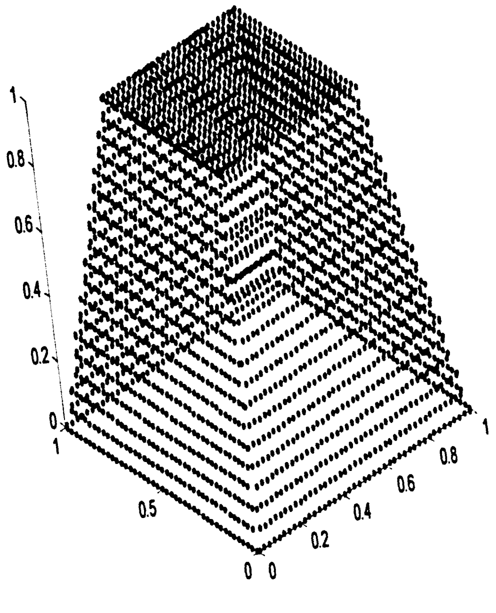
Figure 5.1 (continued)



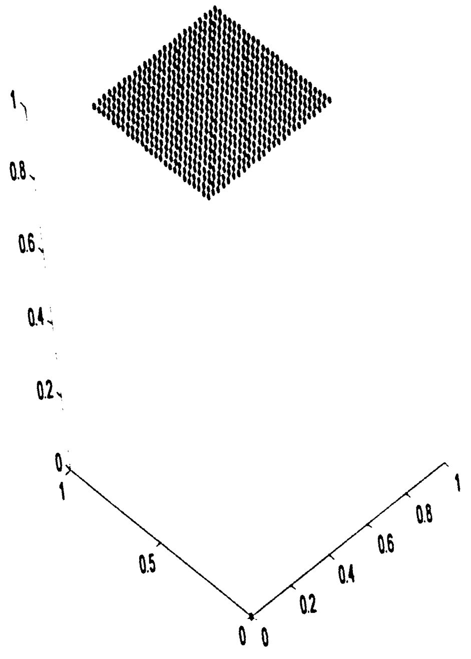
a) Test block



b) Cloud data

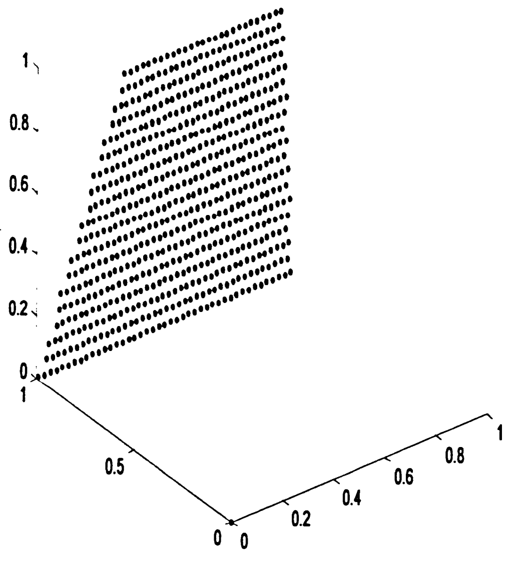


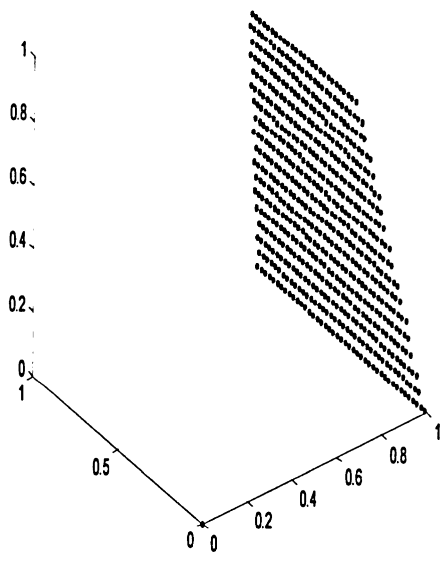
c) Normalized cloud data

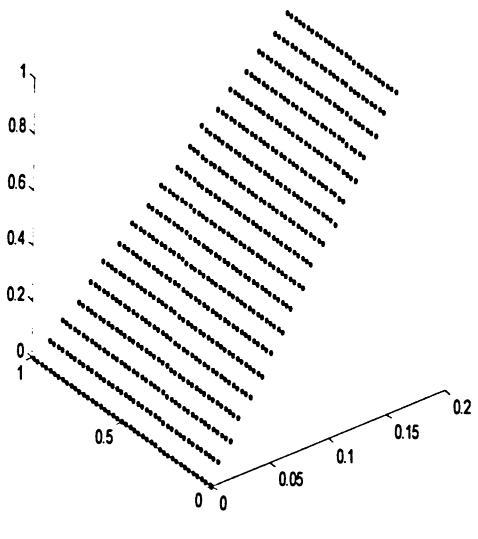


d) Top face

Figure 5.2 Test bock and segmented results


e) Back face


f) Right face


d) Left face

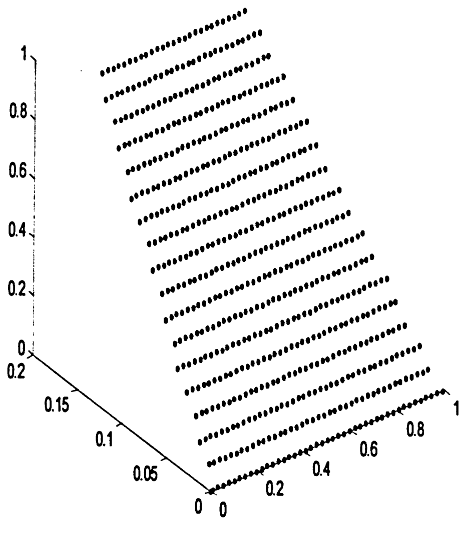
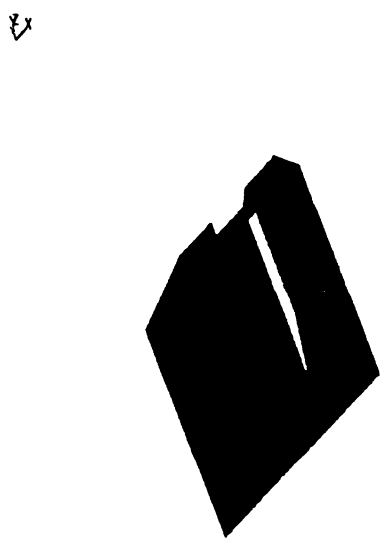
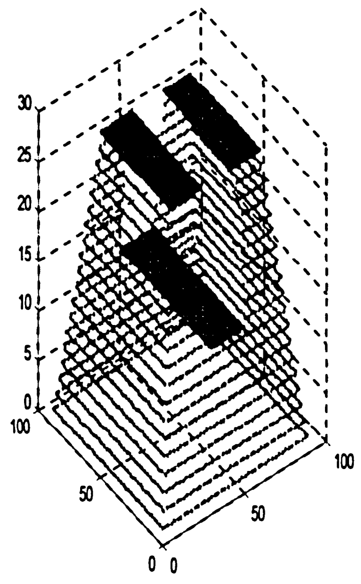

d) Front face

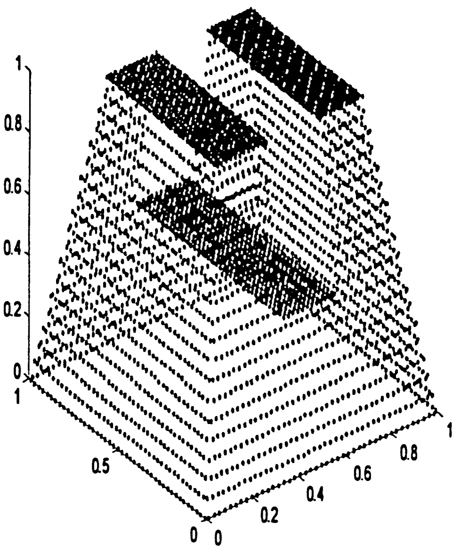
Figure 5.2 (continued)

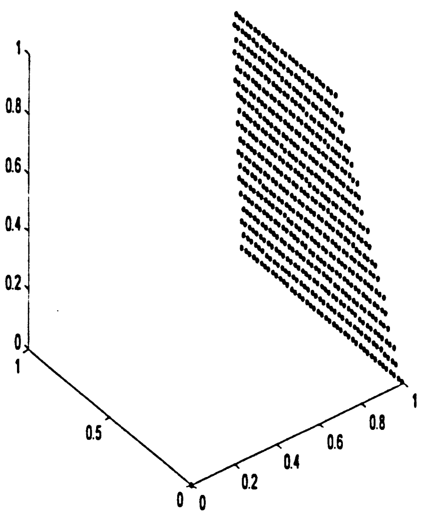


a) Test block (7 faces)



b) Cloud data

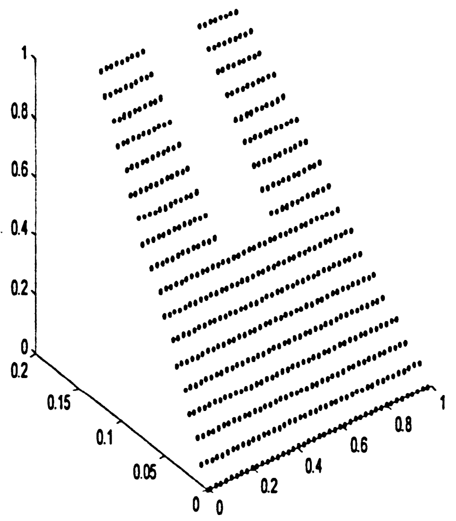




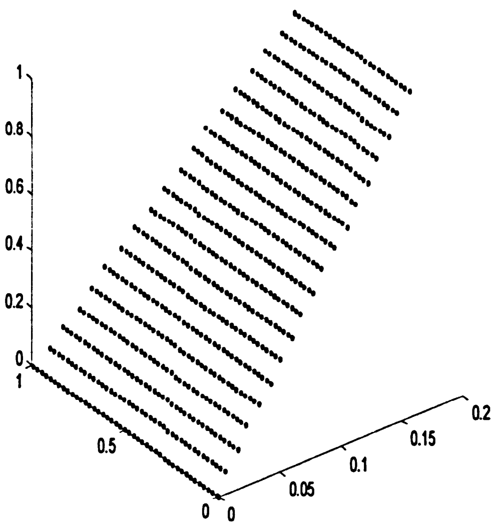
c) Normalized cloud data

d) Right face

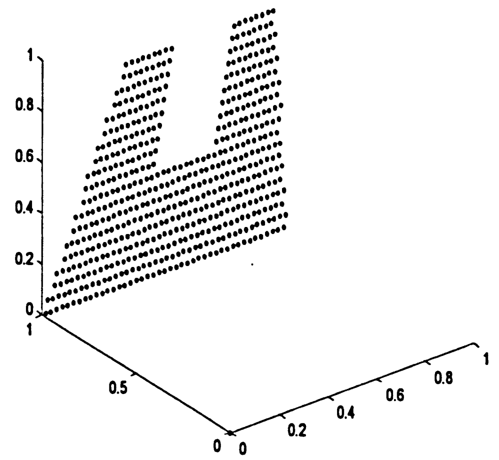
Figure 5.3 Test bock and segmented results



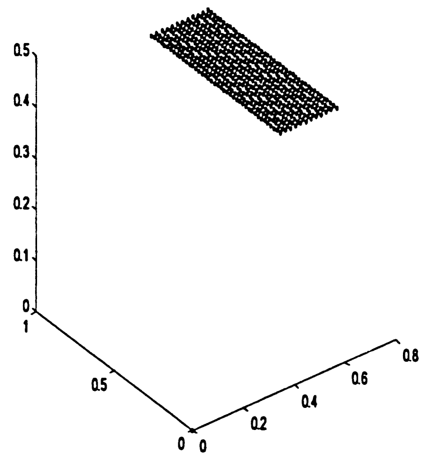
e) Front face



f) Left face

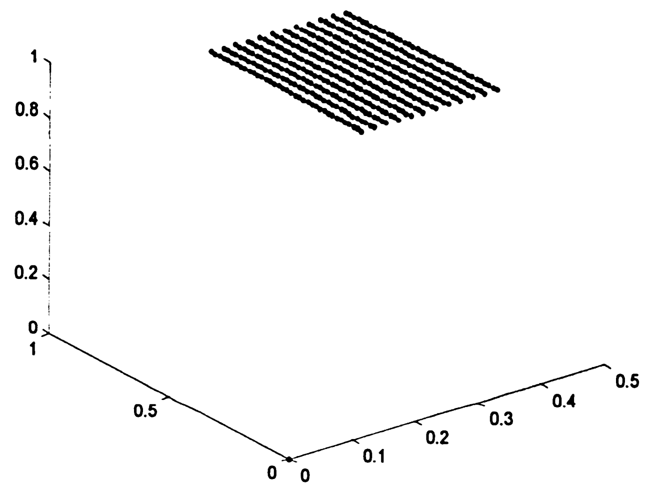


g) Back face

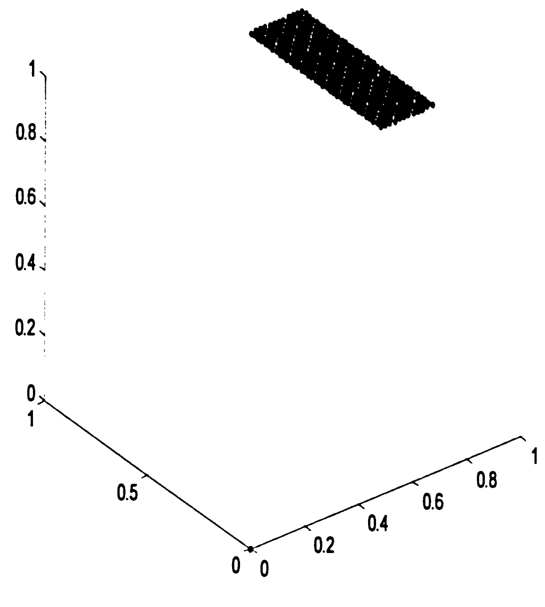


h) Groove top face

Figure 5.3 (continued)

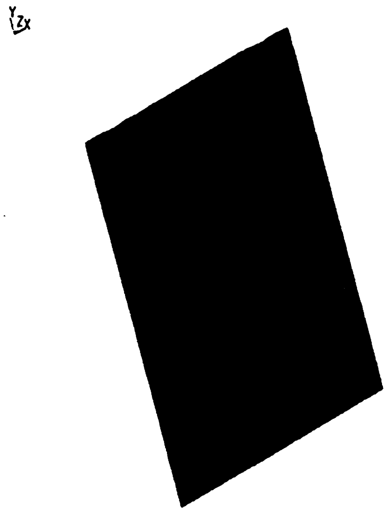


i) Block left top face

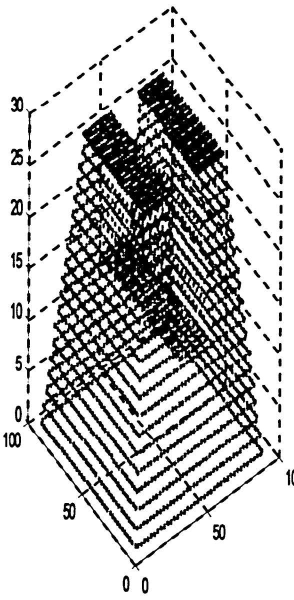


j) Block right top face

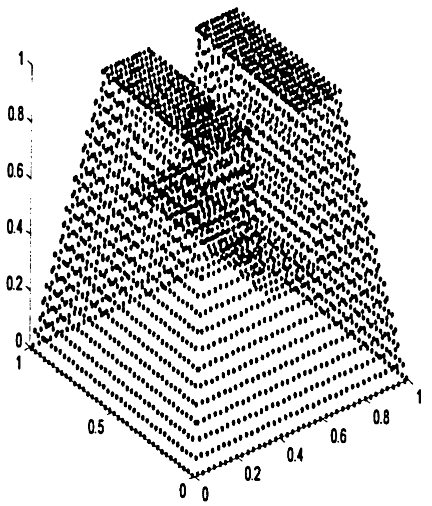
Figure 5.3 (continued)



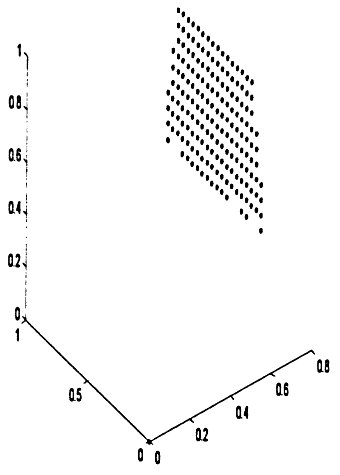
a) Test block (9 faces)



b) Cloud data

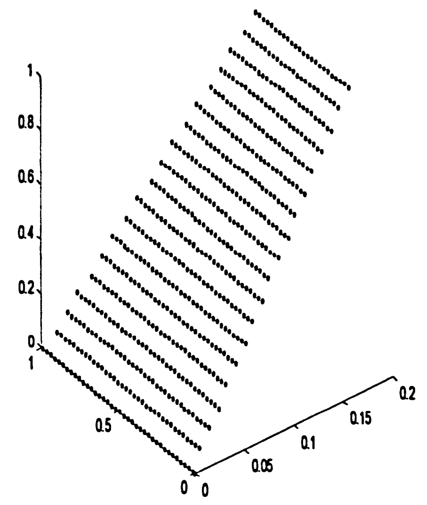


c) Normalized cloud data

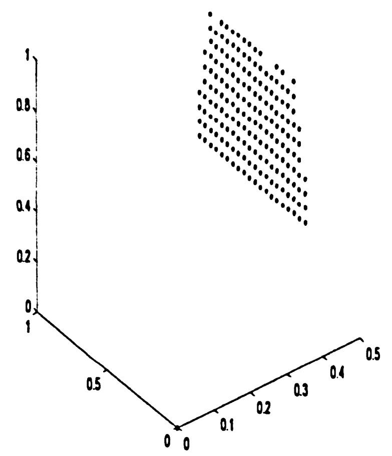


d) Groove right face

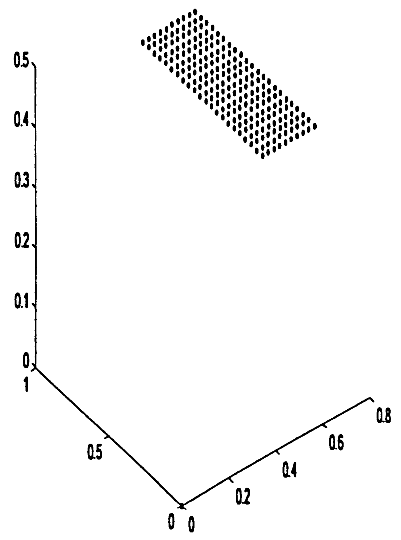
Figure 5.4 Test bock and segmented results



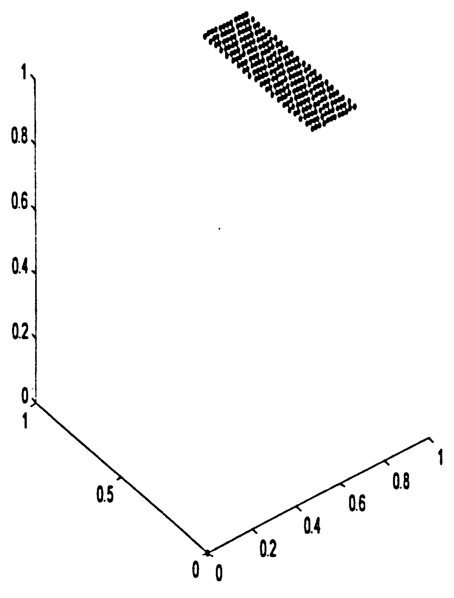
e) Block left face



f) Groove left face

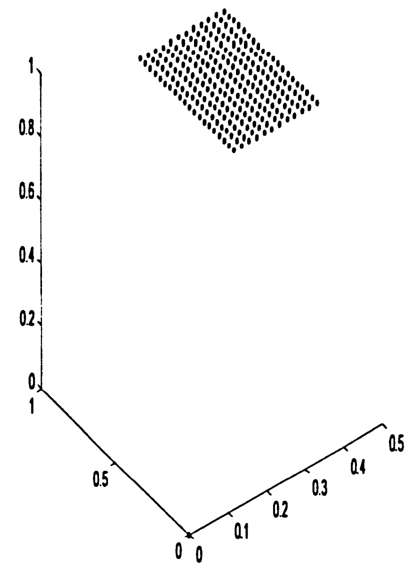


g) Groove top face

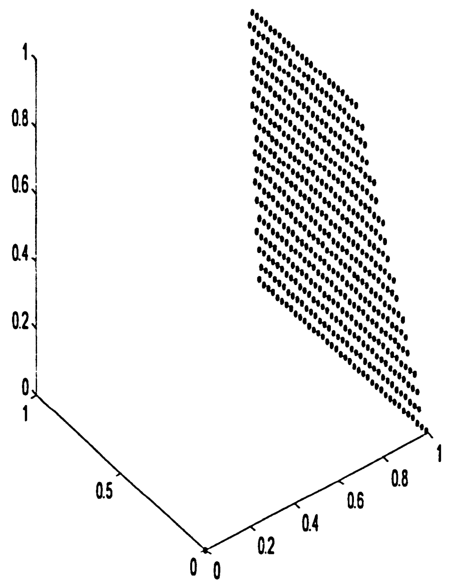


h) Block right top face

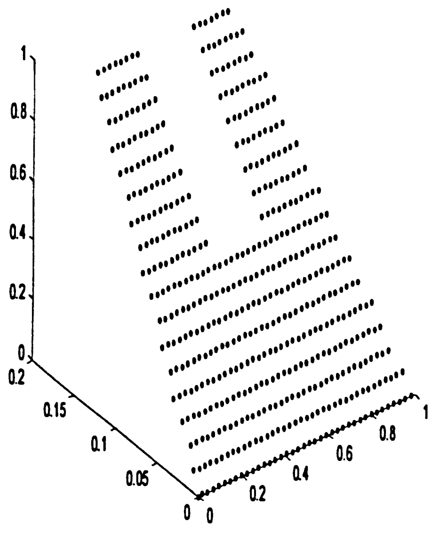
Figure 5.4 (continued)



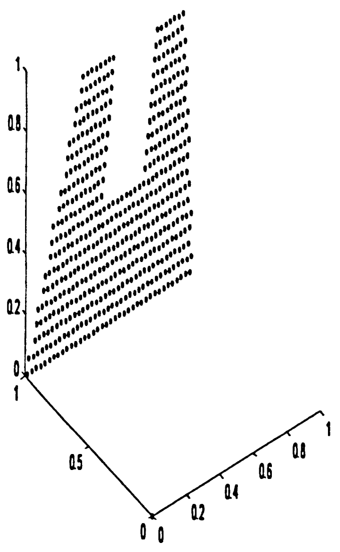
i) Block left top face



j) Block right face



k) Block front face



l) Block back face

Figure 5.4 (continued)

To avoid an oversegmented or an undersegmented surface, the number of neural units in the competitive layer needs to be chosen close to the number of regions desired in the final segmentation. In most case, the number of neural unit is not strict. For example, in Figure 5.4 the competitive size can be $3\times3$, $3\times4$, $4\times4$, $4\times5$ (20 units are over two times actual faces (9 faces) and an oversegmented occurs if the size is bigger than $6\times6$. For a simple object (Figure 5.1), the competitive size can be up to $7\times7$(49 units are almost ten times actual faces (5 faces)). In the following experiment (Figure 5.6), the final segmentation result shows the successful segmentation of two planar surfaces (top surface and bottom surface). But it fails to segment the cone surface as a whole surface. An oversegmentation occurs even though every point on the cone surface has the same nonzero magnitude Gaussian curvature. It segments the cone surface into several longitudinal strips (patches). The reason for the segmentation of the cone surfaces into longitudinal strips is that surface points on a single longitudinal strip have identical or similar unit normal vector values. And the weight of Gaussian curvature is much smaller than the weight of the normal. To solve this problem, three methods can be used.

1. The number of neural units chosen is close to the number of regions desired in the final segmentation. For example, an oversegmentation can be avoided if the number of neural units is chosen as 3, or 4 (Figure 5.7) and 5, or 6 (Figure 5.10).

2. The weights of Gaussian and mean curvature are increased. For example, the number of neural units can be up to 9 (9 is 3 times the number of final segmented regions) if the weight of Gaussian curvature is bigger than 0.6 (Figure 5.7) and the number of neural units can be up to 16 (16 is over 3 times the number of final segmented regions) if the weights of Gaussian and mean curvature are both bigger than 0.7.

3. A hierarchical self-organizing feature map (HSOFM) can be used. Jean and Minsoo [8, 9] found that an HSOFM can alleviate the oversegmentation shortcomings of SOFM in the context of image segmentation. But even in HSOFM, the weights of a, b, c, d are still variational.

After study, it is suggested that the first method be used, as:

1. The weight ratio a:b:c:d=1:0.7:0.1:0.05 can be used for the weighted Euclidean distance measure in all the case of 3D point data.

2. Computational times are very short. Trying different size of the neural net is possible. For example, a training step is chosen as 50000 (Figure 5.9). The resulting time is as little as 2. 3 seconds.

3. The oversegmentation will not occur if the central angle of a cylinder surface is less than $90^0$ (Figure 5.8).

4. Due to the manufacturing error, actual geometric features are not perfect. Flatness evaluates the largest vertical distance between the highest and lowest points on a surface. The flatness tolerance defines the distance between two perfect planes within which all point on a surface must lie (Figure 5.5). Flatness could be considered straightness on a surface, applied in all directions. Therefore, an oversegmentation will occur if the weights of Gaussian and mean curvature are chosen greater. For example, the test block (Figure 5.9) will be segmented into many small patches because of the poor flatness if the weights of Gaussian and mean are both taken as 0.7.

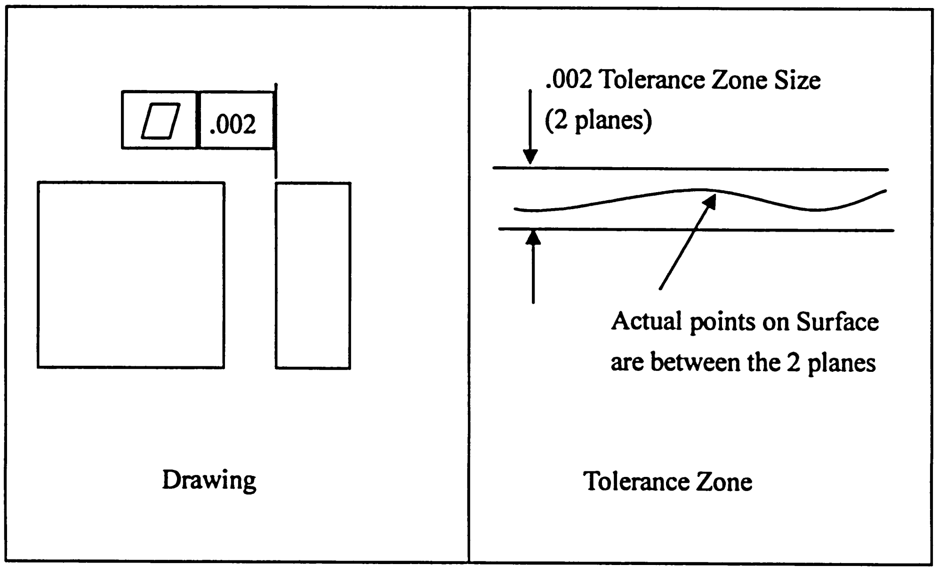


Figure 5.5 Flatness

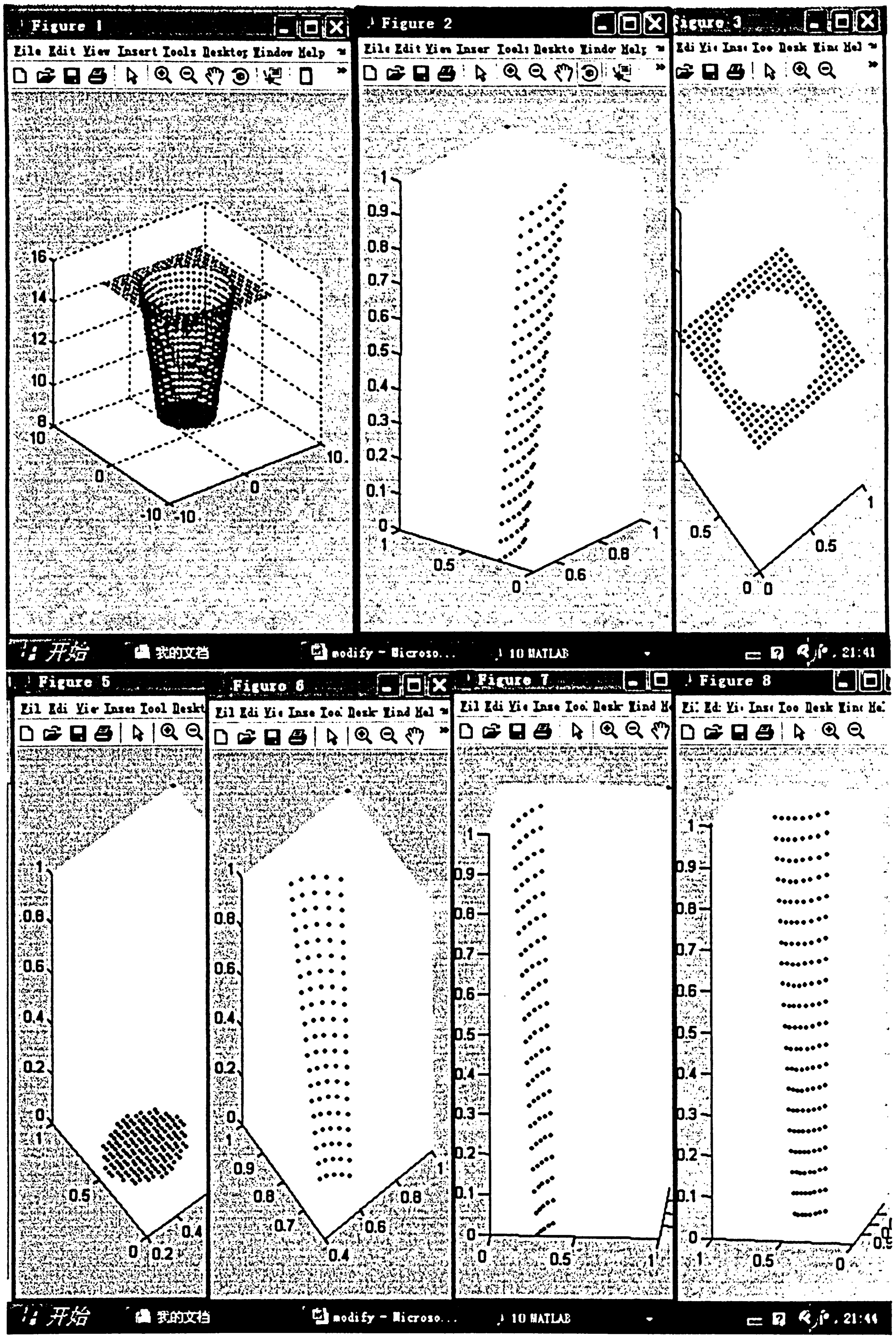
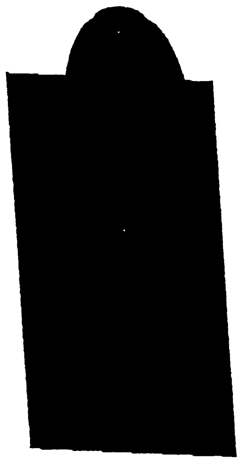
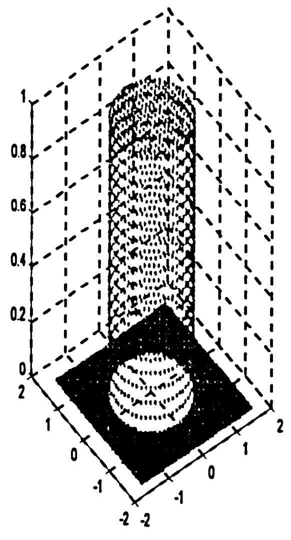
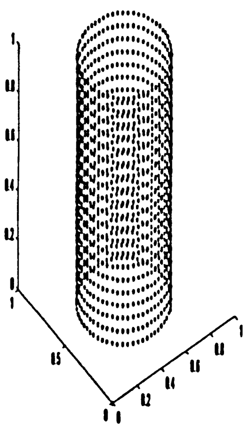
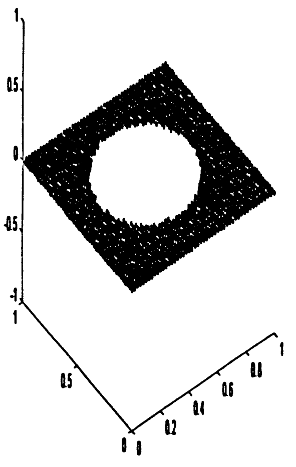


Figure 5.6 An example of oversegmentation


a) Test block (3 faces)


b) Cloud data


c) Cylinder face


d) Bottom face

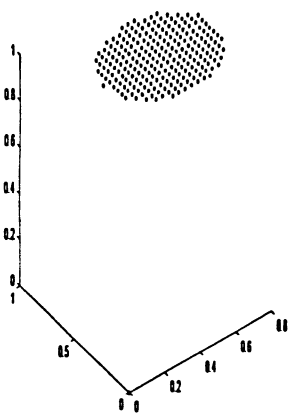
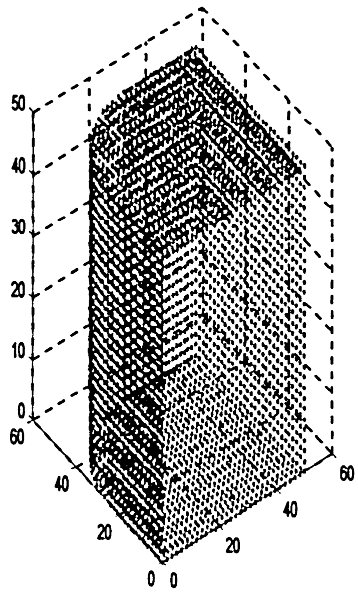
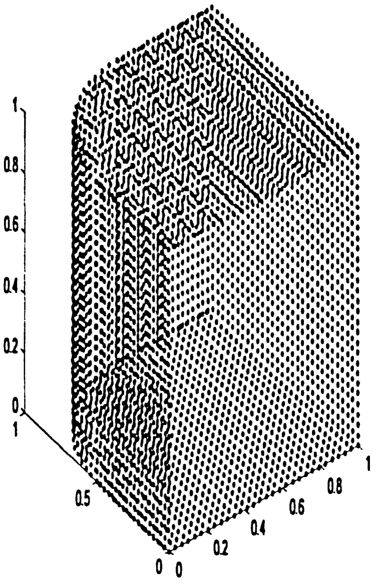

e) Top face

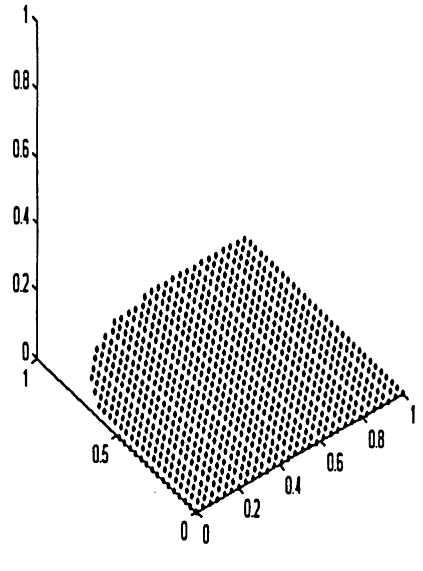
Figure 5.7 Test object and segmented results



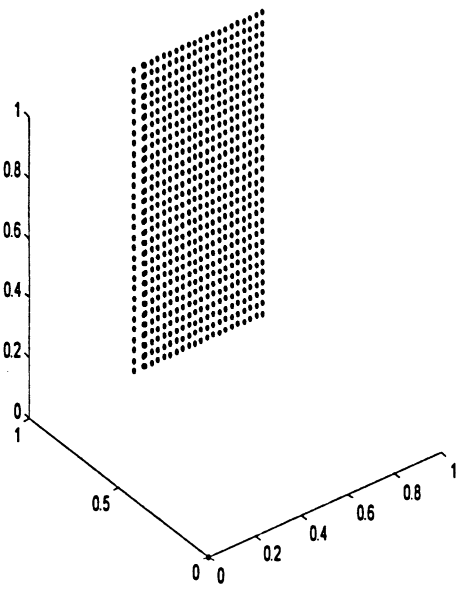
a) Cloud data (6 faces)



b) Normalized cloud data

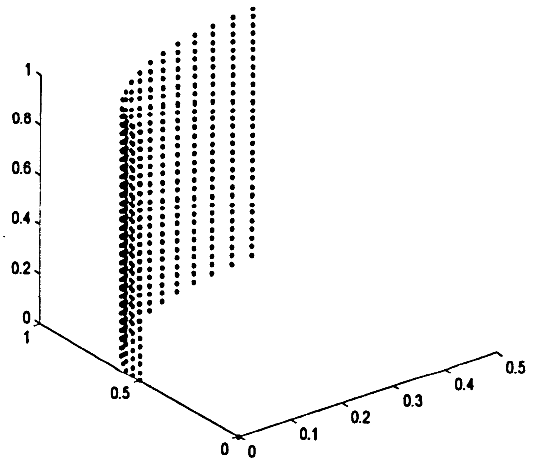


c) Bottom face

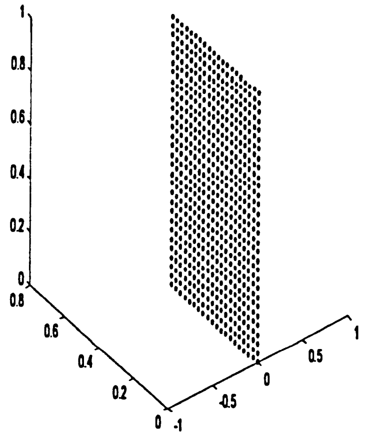


d) Back face

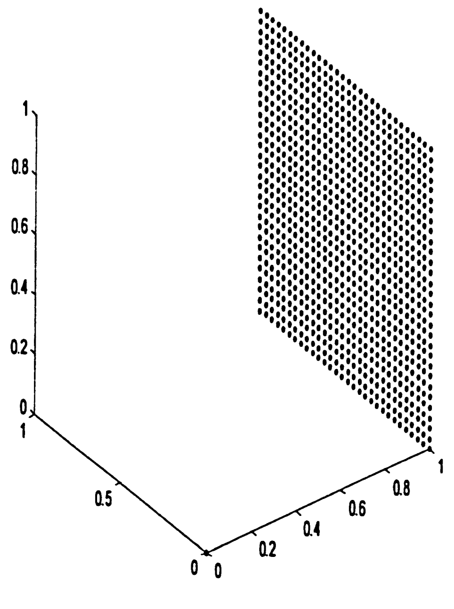
Figure 5.8 Test object and segmented results



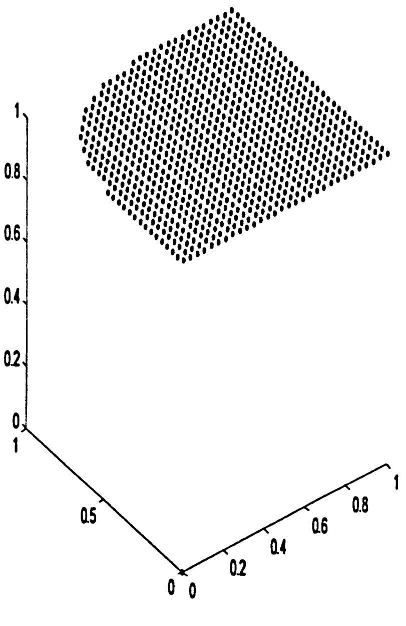
e) Cylinder face (fillet face)



f) Left face



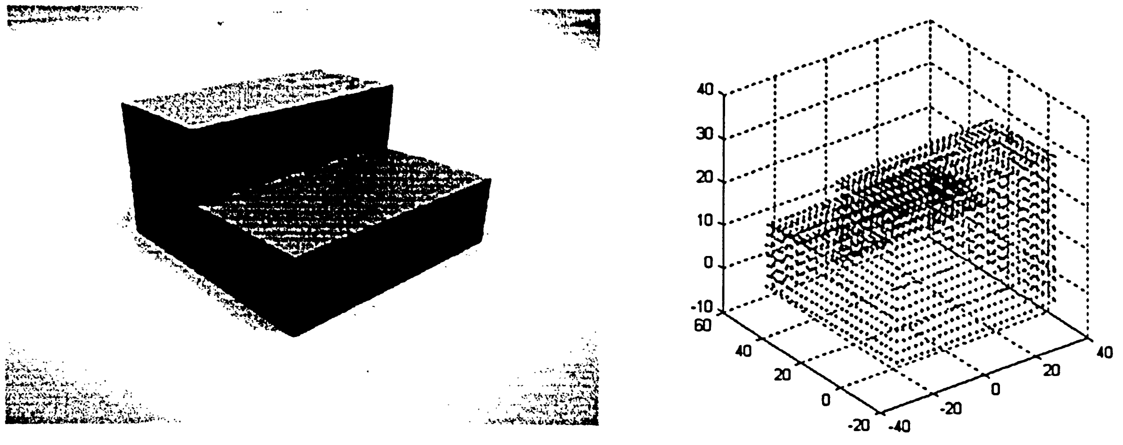
g) Right face



h) Top face

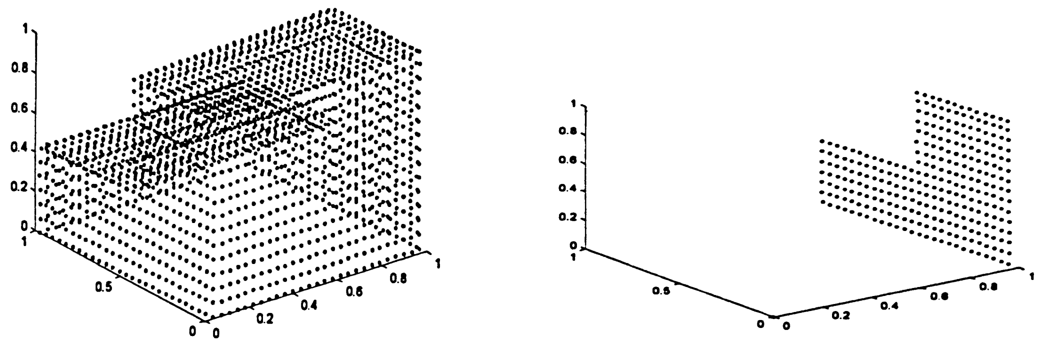
Figure 5.8 (continued)

Due to manufacturing errors, the 3D dimensions of the test blocks are inaccurate, For example, the actual biggest width dimension of the block in Figure 5.9 is 62.15 mm and the smallest dimension is 61.92 mm, so the left face in Figure 5.9 e) has a slight angle with axis (0.04, not exactly 0). Similarly, the length dimension of this block is from 61.92 mm to 62.33 mm can explain why j) back face is not exactly in line with the axis.



a) Test block (7 faces)

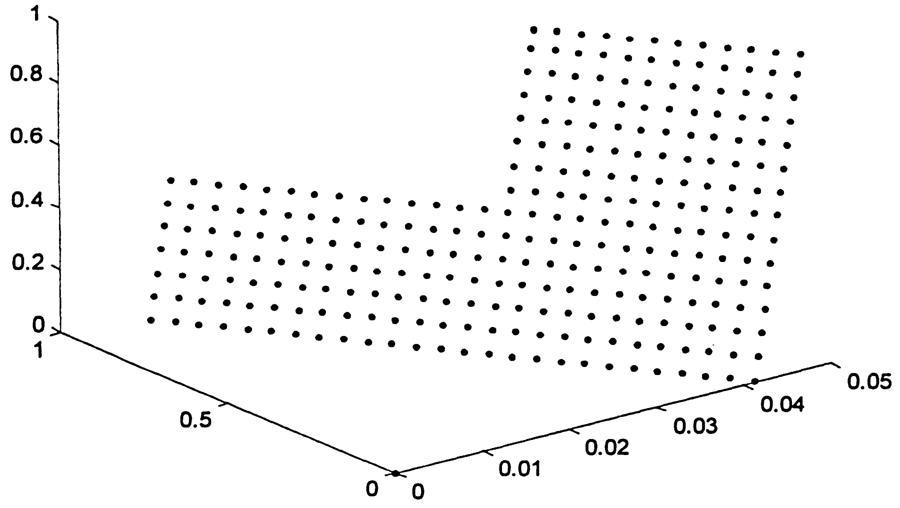
b) Cloud data



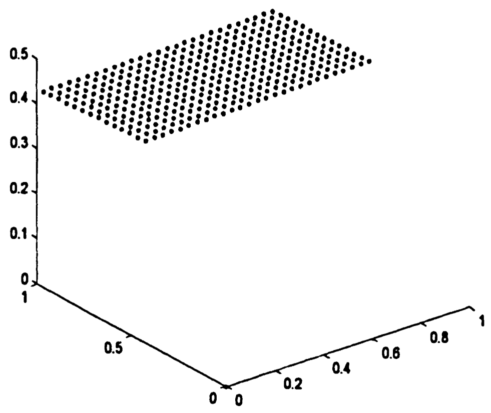
c) Normalized cloud data

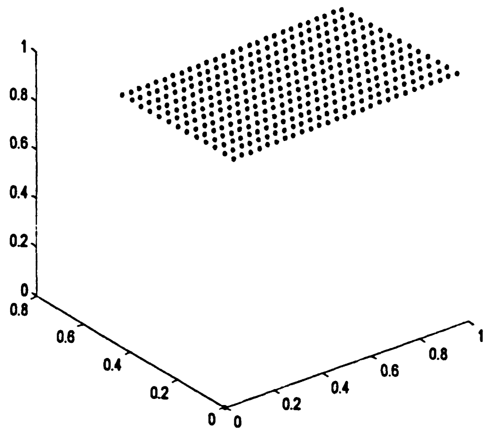
d) Right face

Figure 5.9 Test bock and segmented results



e) Left face

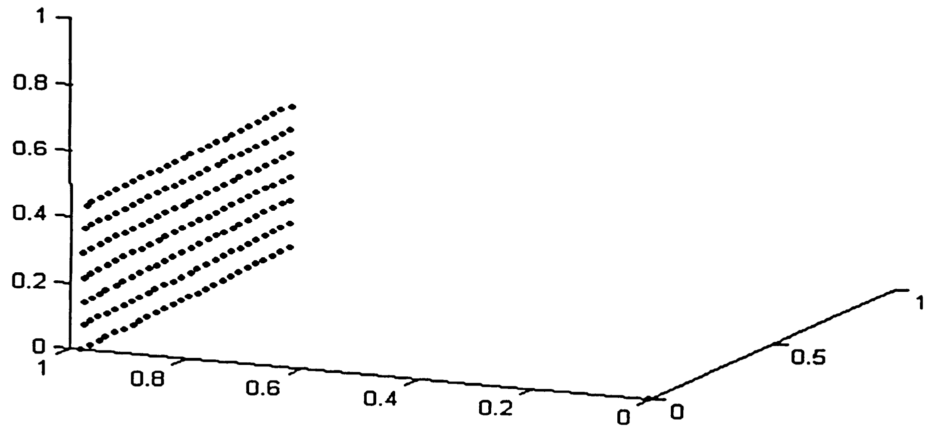




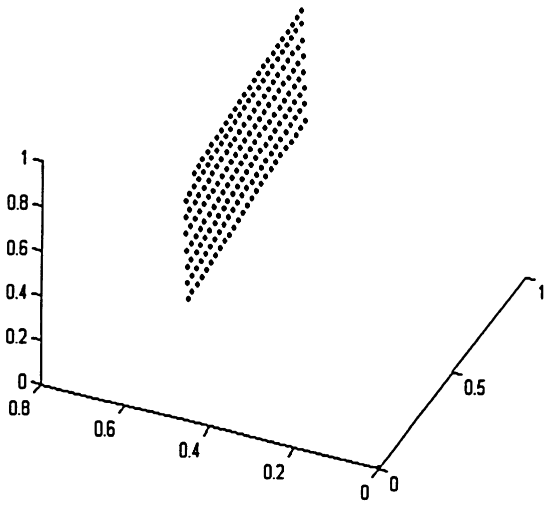
f) Step top face

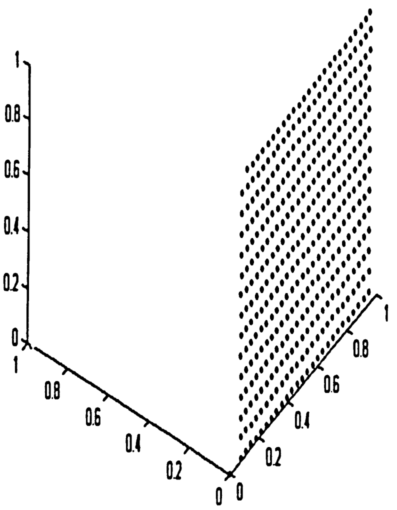
g) Block top face

Figure 5.9 (continued)



h) Fornt face

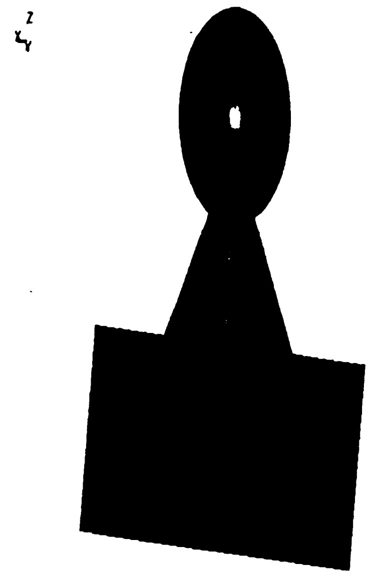




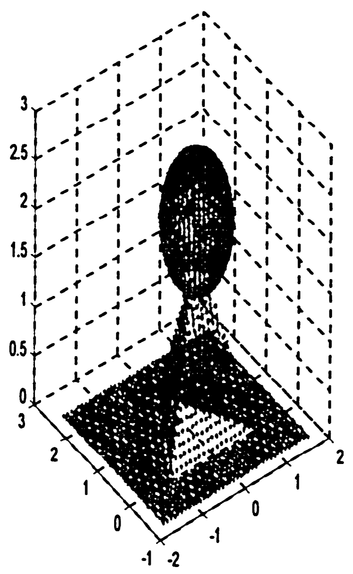
i) Middle face

j) Back face

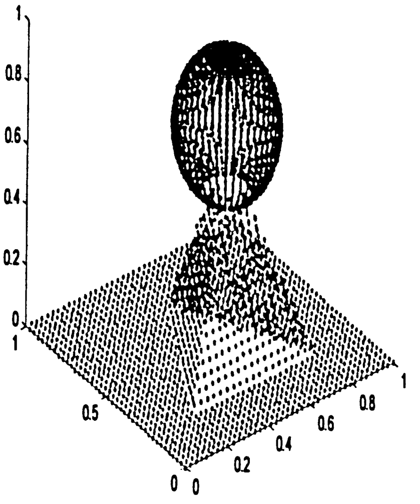
Figure 5.9 (continued)



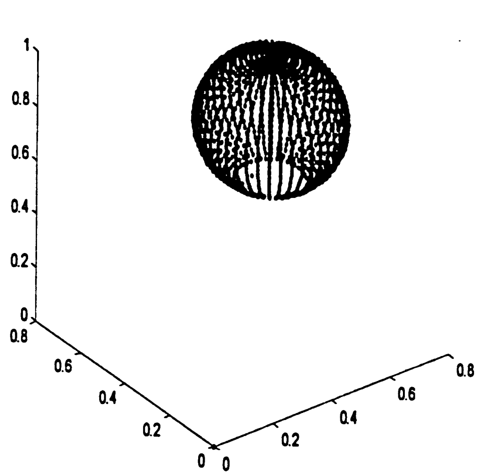
a) Test block (5 faces)



b) Cloud data

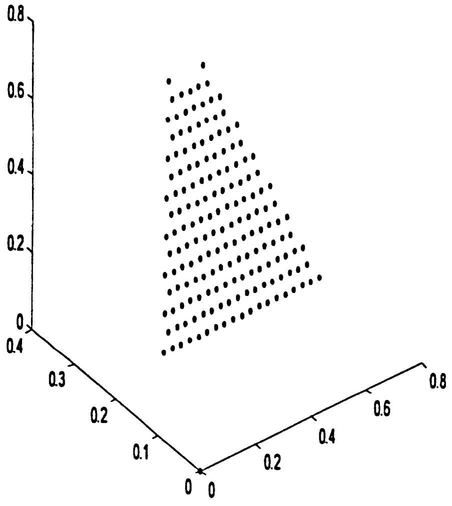


c) Normalized cloud data

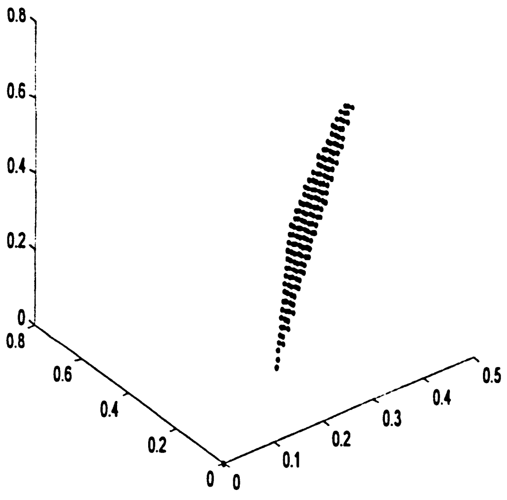


d) Sphere face

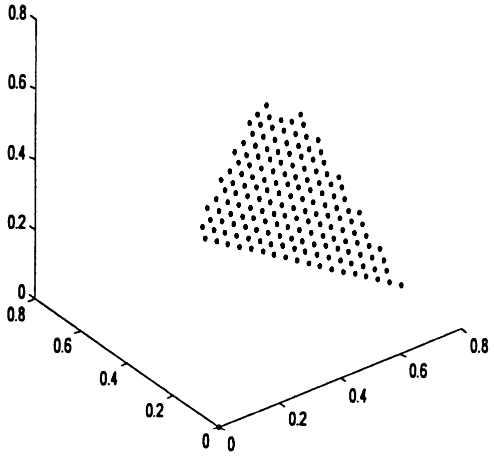
Figure 5.10 Test object and segmented results



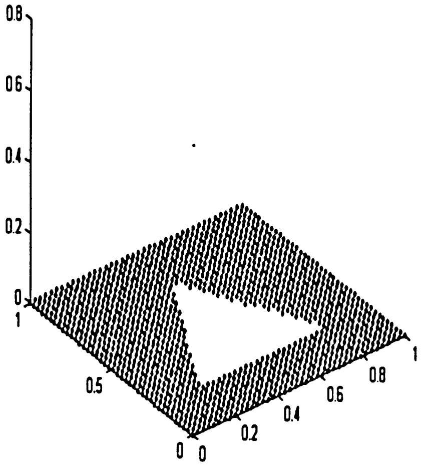
e) Support face 1



f) Support face 2



g) Support face 3



h) Bottom face

Figure 5.10 (continued)

# Chapter 6
# Conclusions and Future work

## 6.1 Conclusions

In most RE systems, the segmentation of digitized points is performed interactively. This manual process by a RE operator is time consuming and prone to error. This research reports on the automatic segmentation and classification of point data. The principal contributions of this work and conclusions are as follows.

1. An automatic reverse engineering system is proposed. This system is simple and practical.
2. After existing segmentation methods are reviewed, an automatic segmentation algorithm of multiple viewpoints 3D digitized point data captured by a laser scanner or a CMM is proposed and implemented. Some of the building blocks (SOFM, Darboux frame and the weighted least-square method) are well known, but it is the way in which these are tied together. Experimental results prove that this method is effective and correct.
3. Normalized feature vector and weighted Euclidean distance adopted in the learning process of the SOFM can improve the speed and exactness of the segmentation.
4. The segmentation using the SOFM is robust to noise (the accuracy of Dr. Picza-250 is 0.2 mm) and has no limitation to surface type.

## 6.2 Future Work

To construct an automatic reverse engineering modeling system, surface fitting and surface trim will be programmed by MATLAB after segmentation and classification and cloud data.

For most mechanical parts, CAD models can be represented by a set of parametric patches such as plane, cylinder, sphere, cone and torus. Lukaxs and Marshall [44] describe Geometric least-squares fitting of spheres, cylinders, cones and tori. Reference [45] gave MATLAB source codes of the Least Squares Geometric Elements library. This library consists of MATLAB functions to find the least-squares fit of geometric shapes to data, implementing a number of the geometric fitting routines key functions. It is based on a general purpose non-linear least-squares solver that takes as input function-and-gradient routines, and these routines are implementations of the geometric evaluation key functions.

Surface trimming is sometimes needed after surface fitting using the least-square methods. For example, in Appendix C, a whole face is obtained when the segmented bottom face is fitted using the least-square method. Arshad [46] used a back propagation neural network to recognize these boundary curves. The final surface can be created by trimming the whole face with boundary curves using Boolean operators.

Free-form surfaces are needed to reconstruct if the segmented point subset is a free-form surface. The NURBS method seems to be the most effective and is introduced in Appendix D.

# APPENDICES

## Appendix A: Glossary of Terms

| Term | Definition |
|---|---|
| ANN | Artificial Neural Network- a computer algorithm based on the architecture of a biological brain. |
| SOM | Self-Organizing Maps-networks in which units exhibit a competitive form of behavior and presents neural-based examples of unsupervised learning. |
| Back-Propagation | Method to update connector weights in multi-layer neural networks based on error. |
| CAD | Computer Aided Design |
| CAM | Computer Aided Manufacturing |
| Cloud data | Term used to describe the collected points in RE |
| Neuron | A node in neural network |
| Topology | The spatial relationship of different patches to each other |
| Free-form surface | A surface not belong to any geometric primitive |
| NURBS | Non-uniform rational B-spline surface |
| One-Ring | One group triangles which share a same vertex |
| Segmentation | Divide the cloud data into subregions |

## Appendix B: MATLAB Program codes

```
% MATLAB code for normal calculation of point data

clear all
close all
tic                                     % Start a stopwatch timer
load faceplane1.txt;                    % Loading faceplane1 text file
x=faceplane1(:,1);
y=faceplane1(:,2);
z=faceplane1(:,3);
n=length(x);
for i=1:n
    if rem(i,4)~=1                      % exclude non-point coordinate
     k=1;
% assign the first normal value of point data
        if rem(i,4)==2
            normal_x=x(i-1);
            normal_y=y(i-1);
            normal_z=z(i-1);
        elseif rem(i,4)==3
            normal_x=x(i-2);
            normal_y=y(i-2);
            normal_z=z(i-2);
        else normal_x=x(i-3);
            normal_y=y(i-3);
            normal_z=z(i-3);
        end
```

```
% Find the points with same vertex
            for j=1:n
                  if rem(j,4)~=1
                        if (x(i)==x(j))&&(y(i)==y(j))&&(z(i)==z(j))&&(i~=j)
                              k=k+1;
                         if rem(j,4)==2
                               normal_x=normal_x+x(j-1);
                               normal_y=normal_y+y(j-1);
                               normal_z=normal_z+z(j-1);
                         elseif rem(j,4)==3
                               normal_x=normal_x+x(j-2);
                               normal_y=normal_y+y(j-2);
                               normal_z=normal_z+z(j-2);
                         else normal_x=normal_x+x(j-3);
                               normal_y=normal_y+y(j-3);
                               normal_z=normal_z+z(j-3);
                         end
                        end
                  end
            end

% calculate normal value of cloud data
            pointnormal_x(i)=normal_x/k;       % x normal components
            pointnormal_y(i)=normal_y/k;       % y normal components
            pointnormal_z(i)=normal_z/k;       % z normal components
      end
end
```

```
for i2=1:n
      if rem(i2,4)==1                              % normal values
         x(i2)=nan;
         y(i2)=nan;
         z(i2)=nan;
         pointnormal_x(i2)=nan;
         pointnormal_y(i2)=nan;
         pointnormal_z(i2)=nan;
       end
end

delete=find(isnan(x));                            % find duplicate points
x(delete)=[];
y(delete)=[];
z(delete)=[];
pointnormal_x(delete)=[];
pointnormal_y(delete)=[];
pointnormal_z(delete)=[];
nn=length(x);
for i4=1:nn
      for j4=(i4+1):nn
            if (x(i4)==x(j4))&&(y(i4)==y(j4))&&(z(i4)==z(j4))
                  x(j4)=nan;
                  y(j4)=nan;
                  z(j4)=nan;
            end
      end
```

```
end
delete1=find(isnan(x));
x(delete1)=[];
y(delete1)=[];
z(delete1)=[];

% Elimination of duplicate point
pointnormal_x(delete1)=[];
pointnormal_y(delete1)=[];
pointnormal_z(delete1)=[];
toc
```

```
% MATLAB codes for curvature calculation of poit data
% using local Darboux frame and least-square method


clear all
close all
tic
load stlfile.txt;           %load stlfile text of point data
X=stlfile(:,1);
Y=stlfile(:,2);
Z=stlfile(:,3);
m=length(X);
k=0;
for ii=1:m
    if rem(ii,4)~=1         % exclude non point coordinate
        k=k+1;
    x(k)=X(ii);
    y(k)=Y(ii);
    z(k)=Z(ii);
    end
end
n=length(x);
for i=1:n
    l=2;
   % assign the neighthood point of first triangle
    if rem(i,3)==1
        neighborhood_x(1)=x(i+1);
        neighborhood_y(1)=y(i+1);
        neighborhood_z(1)=z(i+1);
        neighborhood_x(2)=x(i+2);
        neighborhood_y(2)=y(i+2);
        neighborhood_z(2)=z(i+2);
    elseif rem(i,3)==2
        neighborhood_x(1)=x(i-1);
        neighborhood_y(1)=y(i-1);
        neighborhood_z(1)=z(i-1);
        neighborhood_x(2)=x(i+1);
        neighborhood_y(2)=y(i+1);
        neighborhood_z(2)=z(i+1);
    else neighborhood_x(1)=x(i-2);
        neighborhood_y(1)=y(i-2);
        neighborhood_z(1)=z(i-2);
        neighborhood_x(2)=x(i-1);
        neighborhood_y(2)=y(i-1);
        neighborhood_z(2)=z(i-1);
    end
% find the points share with a common vertex
    for j=1:n
        if (x(i)==x(j))&&(y(i)==y(j))&&(z(i)==z(j))&&(i~=j)
            l=l+2;
            if rem(j,3)==1
                neighborhood_x(l-1)=x(j+1);
                neighborhood_y(l-1)=y(j+1);
                neighborhood_z(l-1)=z(j+1);
```

```
                    neighborhood_x(l)=x(j+2);
                    neighborhood_y(l)=y(j+2);
                    neighborhood_z(l)=z(j+2);
               elseif rem(j,3)==2
                    neighborhood_x(l-1)=x(j-1);
                    neighborhood_y(l-1)=y(j-1);
                    neighborhood_z(l-1)=z(j-1);
                    neighborhood_x(l)=x(j+1);
                    neighborhood_y(l)=y(j+1);
                    neighborhood_z(l)=z(j+1);
               else neighborhood_x(l-1)=x(j-2);
                    neighborhood_y(l-1)=y(j-2);
                    neighborhood_z(l-1)=z(j-2);
                    neighborhood_x(l)=x(j-1);
                    neighborhood_y(l)=y(j-1);
                    neighborhood_z(l)=z(j-1);
               end
          end
     end
ll=length(neighborhood_x);
for i1=1:(ll-1)
     for j1=(i1+1):ll
          % find duplicate points
          if (neighborhood_x(i1)==neighborhood_x(j1))...
&&(neighborhood_y(i1)==neighborhood_y(j1))...
&&(neighborhood_z(i1)==neighborhood_z(j1))
               neighborhood_x(j1)=nan;
```

```
                neighborhood_y(j1)=nan;
                neighborhood_z(j1)=nan;
            end
        end
    end
    % Eleminating the duplicate points
    delete=find(isnan(neighborhood_x));
    neighborhood_x(delete)=[];
    neighborhood_y(delete)=[];
    neighborhood_z(delete)=[];
    % check number of data points
      m1 = length(neighborhood_x);
      if m1 < 2
        sprintf('error:At least 3 data points required ');
      end


      XX1=[(neighborhood_x)' (neighborhood_y)' (neighborhood_z)'];
      XX2=[x(i) y(i) z(i)];
      XX=[XX1;XX2];
    % calculate centroid
      x0 = mean(XX)';
    % form matrix A of translated points
      A = [(XX(:, 1) - x0(1)) (XX(:, 2) - x0(2)) (XX(:, 3) - x0(3))];
    % calculate the SVD of A
      [U, S, V] = svd(A, 0);
    % find the smallest singular value in S and extract from V the
    % corresponding right singular vector
```

```
 [s, iii] = min(diag(S));
 cosangle = V(:, iii);
 llineydirection1=neighborhood_x(1)-x(i);
 llineydirection2=neighborhood_y(1)-y(i);
 if (llineydirection1==0)&&(llineydirection2==0)
       llineydirection3=1;
 elseif cosangle(3)==0
       llineydirection3=0;
 else
 llineydirection3=(cosangle(1)*llineydirection1+cosangle(2)*llineydirection2)...
/(-cosangle(3));
 end
normal=((llineydirection1)^2+(llineydirection2)^2+(llineydirection3)^2)^0.5;
lineydirection1=llineydirection1/normal;
lineydirection2=llineydirection2/normal;
lineydirection3=llineydirection3/normal;
linexdirection1=(1-(lineydirection1)^2-(cosangle(1))^2)^.5;
linexdirection2=(1-(lineydirection2)^2-(cosangle(2))^2)^.5;
linexdirection3=(1-(lineydirection3)^2-(cosangle(3))^2)^.5;
nn=length(neighborhood_x);
for i2=1:nn
        BB.x(i2)=neighborhood_x(i2)-x(i);
        BB.y(i2)=neighborhood_y(i2)-y(i);
        BB.z(i2)=neighborhood_z(i2)-z(i);
  end
        BB1=[BB.x;BB.y;BB.z];
```

```
                newcoordinate=[linexdirection1 linexdirection2 linexdirection3;...
lineydirection1 lineydirection2 lineydirection3;...
cosangle(1) cosangle(2) cosangle(3)]*BB1;
        newcoordinate_x=newcoordinate(1,:);
        newcoordinate_y=newcoordinate(2,:);
        newcoordinate_z=newcoordinate(3,:);

        newcoordinate_xsquare=(newcoordinate_x).^2;
        newcoordinate_xy=(newcoordinate_x).*(newcoordinate_y);
        newcoordinate_ysquare=(newcoordinate_y).^2;
coordinate_matrix=[newcoordinate_xsquare; newcoordinate_xy;
newcoordinate_ysquare]';
% least-square method to slove the coefficent
coefficent=coordinate_matrix\(newcoordinate_z)';
% calcualte principal curvature
k1=coefficent(1)+coefficent(3)-((coefficent(1)-coefficent(3))^2+coefficent(2)^2)^0.5;
k2=coefficent(1)+coefficent(3)+((coefficent(1)-coefficent(3))^2+coefficent(2)^2)^0.5;
% calculate gaussian and mean curvature
K(i)=k1*k2;
H(i)=(k1+k2)/2;
end
save hkcurvature H K
toc
```

```
% MATLAB code for segmentation of cloud data using SOFM

clear all
close all
tic
load step2.txt;                    % load step2 text file
X=step2(:,1);
Y=step2(:,2);
Z=step2(:,3);
Normalx=step2(:,4);
Normaly=step2(:,5);
Normalz=step2(:,6);
load hkstep2 H K                   % load Gaussian and mean curvature file
[m n]=size(step2);
%classes=input('Please input the number to classify:');
POINTCLASS=zeros(m,2);   %remember the winning neuron for input point
%build SOM neural network
SOMNEURONX=input('Please input the number of rows of SOM:');
SOMNEURONY=input('Please input the number of columns of SOM:');
% assign random weight
weight=rand(SOMNEURONX,SOMNEURONY,3);
disp('Initialized coordinate weights')
weight;              %showing initialized weights matrix
weightnormal=rand(SOMNEURONX,SOMNEURONY,3);
disp('Initialized normal weights')
weightnormal; %showing initialized weights matrix
```

```
weightH=rand(SOMNEURONX,SOMNEURONY);
weightK=rand(SOMNEURONX,SOMNEURONY);
% assign coefficients
a=1;
b=0.7;
c=0.1;
d=0.005;
SOMloop=5;
epochs=50000;
alpha=0.9;
sigma=2.5;
figure;
plot3(X,Y,Z,'g.');          %showing the distribution of point
axis square;
grid on;
wx=weight(:,:,1);
wy=weight(:,:,2);
wz=weight(:,:,3);
grid on;
wnx=weightnormal(:,:,1);
wny=weightnormal(:,:,2);
wnz=weightnormal(:,:,3);
xmin=min(X);
xmax=max(X);
ymin=min(Y);
ymax=max(Y);
zmin=min(Z);
```

```
zmax=max(Z);
Hmin=min(H);
Hmax=max(H);
Kmin=min(K);
Kmax=max(K);
%Normalized feature vector
for ii=1:m
      X(ii)=(X(ii)-xmin)/(xmax-xmin);
      Y(ii)=(Y(ii)-ymin)/(ymax-ymin);
      Z(ii)=(Z(ii)-zmin)/(zmax-zmin);
Normalx(ii)=Normalx(ii)/((Normalx(ii)^2+Normaly(ii)^2+Normalz(ii)^2)^0.5+2*eps);
Normaly(ii)=Normaly(ii)/((Normalx(ii)^2+Normaly(ii)^2+Normalz(ii)^2)^0.5+2*eps);
Normalz(ii)=Normalz(ii)/((Normalx(ii)^2+Normaly(ii)^2+Normalz(ii)^2)^0.5+2*eps);
      H(ii)=(H(ii)-Hmin)/(Hmax-Hmin);
      K(ii)=(K(ii)-Kmin)/(Kmax-Kmin);
end
step2=[X,Y,Z,Normalx,Normaly,Normalz,H,K];
figure;
plot3(X,Y,Z,'r.');
      for epoch1=1:epochs
      alpha=0.98*(0.001/0.98)^(epoch1/epochs);
      sigma=1.5*(0.01/1.5)^(epoch1/epochs);
            if rem(epoch1,10000)==0
                  SOMloop=SOMloop-1;


i=1;
                        while i<m %i is the point position
```

```
                    distanceinput=zeros(SOMNEURONX,SOMNEURONY);
                     %find the winner neuron
                     for i1=1:SOMNEURONX %i1,j1 are the SOM neurons position
                          for j1=1:SOMNEURONY
                               distanceinput(i1,j1)=a.*(((X(i)-weight(i1,j1,1))^2+...
                               (Y(i)-weight(i1,j1,2))^2+(Z(i)-weight(i1,j1,3))^2)^0.5)+...
b.*(((Normalx(i)-weightnormal(i1,j1,1))^2+(Normaly(i)-weightnormal(i1,j1,2))^2+...
(Normalz(i)-weightnormal(i1,j1,3))^2)^0.5)+c.*(H(i)-weightH(i1,j1))+d.*(K(i)-weightK
(i1,j1));
                               %neighborhood=abs(i1-i)+abs(j1-j);
                          end
                     end
                     minimumd=min(min(distanceinput));
                       %i1,j1 are the SOM neurons position
  for i2=1:SOMNEURONX
                          for j2=1:SOMNEURONY
                               if minimumd==distanceinput(i2,j2)
                               POINTCLASS(i,1)=i2;
                               POINTCLASS(i,2)=j2;
                               end
                          end
                          end
                      %determine if computing deltaw
                     for i1=1:SOMNEURONX %i1,j1 are the SOM neurons position
                          for j1=1:SOMNEURONY
neighborhood=((i1-POINTCLASS(i,1))^2+(j1-POINTCLASS(i,2))^2)^.5;
                               if neighborhood <= SOMloop
```

```
distance=(i1-POINTCLASS(i,1))^2+(j1-POINTCLASS(i,2))^2;
deltawx=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,1)-weight(i1,j1,1));
deltawy=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,2)-weight(i1,j1,2));
deltawz=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,3)-weight(i1,j1,3));
deltawnx=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,4)-weightnormal(i1,j1,1));
deltawny=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,5)-weightnormal(i1,j1,2));
deltawnz=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,6)-weightnormal(i1,j1,3));
deltawH=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,7)-weightH(i1,j1));
deltawK=alpha.*exp(-distance/(2*sigma^2)).*(step2(i,8)-weightK(i1,j1));
weight(i1,j1,1)=weight(i1,j1,1)+deltawx;
weight(i1,j1,2)=weight(i1,j1,3)+deltawy;
weight(i1,j1,3)=weight(i1,j1,3)+deltawz;
weightnormal(i1,j1,1)=weightnormal(i1,j1,1)+deltawnx;
weightnormal(i1,j1,2)=weightnormal(i1,j1,2)+deltawny;
weightnormal(i1,j1,3)=weightnormal(i1,j1,3)+deltawnz;
weightH(i1,j1)=weightH(i1,j1)+deltawH;
weightK(i1,j1)=weightK(i1,j1)+deltawK;
                                    end
                           end
                      end
                                             %POINTCLASS(i,j,1)=i1;
                                        %POINTCLASS(i,j,2)=j1;
                 i=i+1;
end
    end
end
```

```
% 'The weights after training';
weight;
weightnormal; %showing weights matrix after training
weightH;
weightK;
x1=weight(:,:,1);
y1=weight(:,:,2);
z1=weight(:,:,3);
figure;
plot3(x1,y1,z1,'r+');
grid on;
Nx=weightnormal(:,:,1);
Ny=weightnormal(:,:,2);
Nz=weightnormal(:,:,3);
figure;
plot3(Nx,Ny,Nz,'r+');
grid on;
%to classify each pixel
POINTCLASS=zeros(m,2); %remember the winning neuron for input point
%classify each point
POINTCLASSNO=zeros(SOMNEURONX,SOMNEURONY);
for i=1:m
        for i1=1:SOMNEURONX %i1,j1 are the SOM neurons position
            for j1=1:SOMNEURONY
                distanceinput(i1,j1)=a.*(((X(i)-weight(i1,j1,1))^2+...
                        (Y(i)-weight(i1,j1,2))^2+(Z(i)-weight(i1,j1,3))^2)^0.5)+...
```

```
b.*(((Normalx(i)-weightnormal(i1,j1,1))^2+(Normaly(i)-weightnormal(i1,j1,2))^2+...
(Normalz(i)-weightnormal(i1,j1,3))^2)^0.5)+c.*(H(i)-weightH(i1,j1))+d.*(K(i)-weightK
(i1,j1));
            end
        end
        minimumd=min(min(distanceinput(:,:)));
      for i1=1:SOMNEURONX %i1,j1 are the SOM neurons position
                      for j1=1:SOMNEURONY
                          if minimumd==distanceinput(i1,j1)
                          POINTCLASS(i,1)=i1;
                          POINTCLASS(i,2)=j1;
                       POINTCLASSNO(i1,j1)=POINTCLASSNO(i1,j1)+1;
                          end
                      end
      end
end
'The number of points belongs to the each output neuron'
POINTCLASSNO
%ttest=input('please press any key to continue...');
%set each point with same color
%ttest=input('please press any key to continue...');
%set each point with same color
%display classified image for each type
for i1=1:SOMNEURONX
    for j1=1:SOMNEURONY
        if POINTCLASSNO(i1,j1) >3
            IOUT = zeros(m,3);
```

```
                for i=1:m
                        if (POINTCLASS(i,1)==i1) && (POINTCLASS(i,2)==j1)
                            for kk=1:3
                                IOUT(i,kk)=step2(i,kk);
                            end
                        end
                end
                figure;
                plot3(IOUT(:,1),IOUT(:,2),IOUT(:,3),'.');
            end
        end
    end
    toc
```

```
% MATLAB codes for back propogation algorithm

clear all
close all
% predetermined the weight
for i=1:2
      for j=1:6
            weight1(i,j)=0.01;
      end
end
weight2=[0.2000      0.2000      0.2000
              0.2000      0.2000      0.2000
              0.2000      0.2000      0.2000
              0.2000      0.2000      0.2000
            -0.6069     -3.2974     -3.2974
            30.4156     3.0235     3.0235];
input=[-0.002 -0.001];
input=[-10 -0.001];
input=[-20 -20.001];
input=[-15   -1];
%desired_output
desired_output=[1 0 0];
for k=1:800
      for i=6
            hidden(i)=0.18;
            for j=1:2
```

```
        hidden(i)=hidden(i)+(weight1(j,i)*input(j));
    end
end
for i=1:3
    output(i)=0.8;
    for j=1:6
        output(i)=output(i)+(weight2(j,i)*hidden(j));
    end
    output(i)=(1-exp(-0.6*(output(i))))/(1+exp(-0.6*(output(i))));
end
total_error=0;
for i=1:3
    total_error=total_error+abs(desired_output(i)-output(i));
end
if total_error<0.03;
else
    for i=1:3
        R(i)=(desired_output(i)-output(i))*output(i)*(1-output(i));
    end
    a=0.1;
    for i=1:6
        for j=1:3
            weight2(i,j)=weight2(i,j)+(a*R(j)*hidden(i));
        end
    end
    for i=1:6
        E_hidden(i)=0;
```

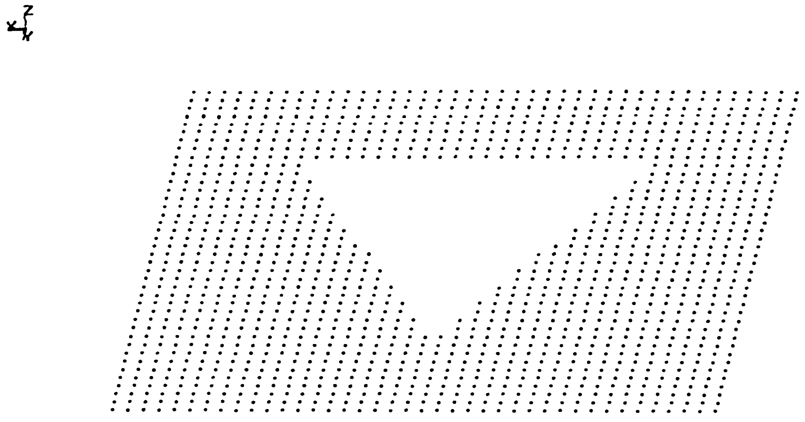
```
                for j=1:3
                      E_hidden(i)=E_hidden(i)*(R(j)*weight2(i,j));
                end
                E_hidden(i)=hidden(i)*(1-hidden(i))*E_hidden(i);
          end
          b=0.06;
          for i=1:2
                for j=1:6
                      weight1(i,j)=weight1(i,j)+(b*E_hidden(j)*input(i));
                end
          end
    end
end
disp('the weights between input and the hidden layer');
weight1;
disp('the weights between the hidden and the output layer');
weight2;
disp('the output')
output
modify_output=( 0.8 < output)
if (modify_output==desired_output)
    disp('this is a plane' )
else
% predetermined the weight
for i=1:2
    for j=1:6
          weight1(i,j)=0.01;
    end
end
weight2=[0.2000      0.2000      0.2000
    0.2000      0.2000      0.2000
    0.2000      0.2000      0.2000
    0.2000      0.2000      0.2000
   -0.6069     -3.2974     -3.2974
   -2.9622     42.0219     -2.7749];
%desired_output
desired_output=[0 1 0];
for k=1:800
    for i=6
        hidden(i)=0.18;
        for j=1:2
            hidden(i)=hidden(i)+(weight1(j,i)*input(j));
        end
    end
    for i=1:3
        output(i)=0.8;
        for j=1:6
            output(i)=output(i)+(weight2(j,i)*hidden(j));
        end
        output(i)=(1-exp(-0.6*(output(i))))/(1+exp(-0.6*(output(i))));
    end
    total_error=0;
    for i=1:3
        total_error=total_error+abs(desired_output(i)-output(i));
        end
        if total_error<0.03;
        else
            for i=1:3
                R(i)=(desired_output(i)-output(i))*output(i)*(1-output(i));
            end
            a=0.1;
            for i=1:6
                for j=1:3
                    weight2(i,j)=weight2(i,j)+(a*R(j)*hidden(i));
                end
            end
            for i=1:6
                E_hidden(i)=0;
                for j=1:3
                    E_hidden(i)=E_hidden(i)*(R(j)*weight2(i,j));
                end
                E_hidden(i)=hidden(i)*(1-hidden(i))*E_hidden(i);
            end
            b=0.06;
            for i=1:2
                for j=1:6
                    weight1(i,j)=weight1(i,j)+(b*E_hidden(j)*input(i));
                end
            end
        end
    end
```

```
disp('the weights between input and the hidden layer');
weight1;
disp('the weights between the hidden and the output layer');
weight2;
disp('the output')
output
modify_output=( 0.8 < output)
if (modify_output==desired_output)
disp('this is an cylinder' )
else
% predetermined the weight
for i=1:2
      for j=1:6
            weight1(i,j)=0.01;
      end
end
weight2=[0.2000      0.2000      0.2000
      0.2000      0.2000      0.2000
      0.2000      0.2000      0.2000
      0.2000      0.2000      0.2000
     -0.6069     -3.2974     -3.2974
     -3.1418     -2.3864     -30.9949];
desired_output=[0 0 1];
for k=1:800
      for i=6
            hidden(i)=0.18;
            for j=1:2
```

```
                hidden(i)=hidden(i)+(weight1(j,i)*input(j));
        end
end
for i=1:3
        output(i)=0.8;
        for j=1:6
                output(i)=output(i)+(weight2(j,i)*hidden(j));
        end
        output(i)=(1-exp(-0.6*(output(i))))/(1+exp(-0.6*(output(i))));
end
total_error=0;
for i=1:3
        total_error=total_error+abs(desired_output(i)-output(i));
end
if total_error<0.03;
else
        for i=1:3
                R(i)=(desired_output(i)-output(i))*output(i)*(1-output(i));
        end
        a=0.1;
        for i=1:6
                for j=1:3
                        weight2(i,j)=weight2(i,j)+(a*R(j)*hidden(i));
                end
        end
        for i=1:6
                E_hidden(i)=0;
```

```
                    for j=1:3
                         E_hidden(i)=E_hidden(i)*(R(j)*weight2(i,j));
                    end
                    E_hidden(i)=hidden(i)*(1-hidden(i))*E_hidden(i);
               end
               b=0.06;
               for i=1:2
                    for j=1:6
                         weight1(i,j)=weight1(i,j)+(b*E_hidden(j)*input(i));
                    end
               end
          end
     end
     disp('the weights between input and the hidden layer');
     weight1;
     disp('the weights between the hidden and the output layer');
     weight2;
     disp('the output')
     output
     modify_output=( 0.8 < output)
     if (modify_output==desired_output)
          disp('this is a sphere' )
     else
          disp('this is not a typical surface')
     end
     end
     end
```

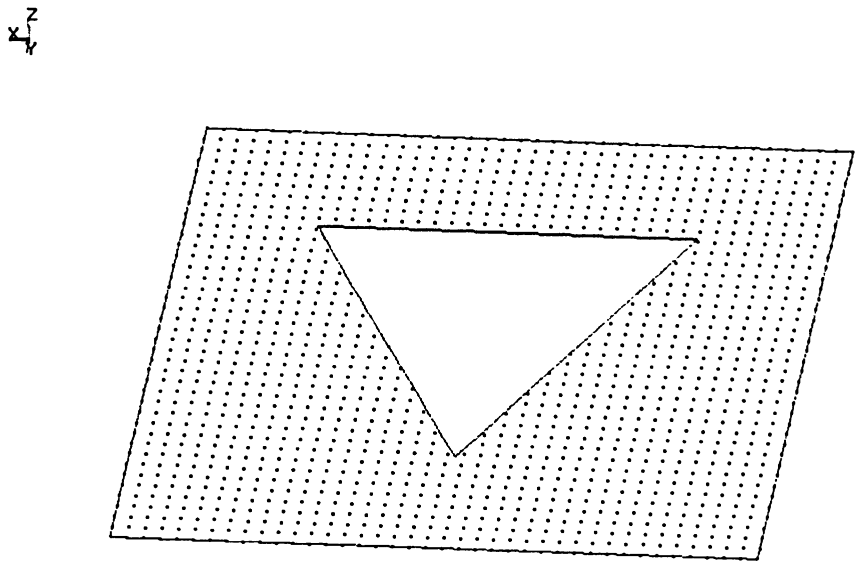
# Appendix C: Surface Trim using Boundary curves



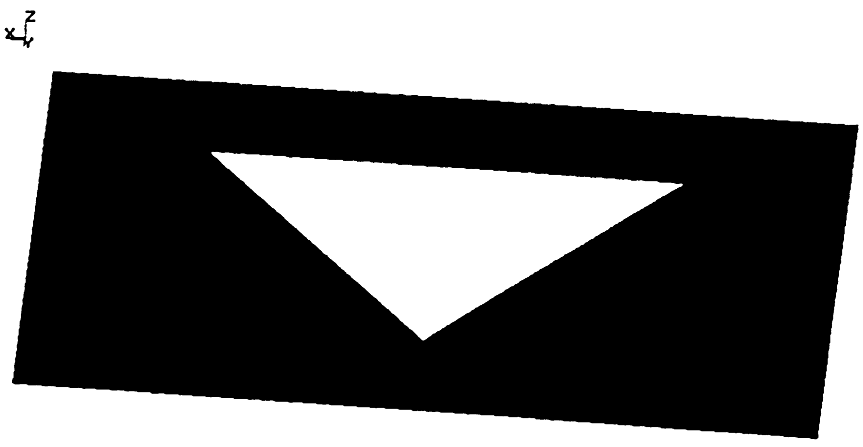
a) segmented patch using SOFM

b) Surface fitting using least-square method

Figure A.1 Surface trim using boundary curves



c) Boundary feature recognition



d) Surface trim using Boolean operations

Figure A.1 (continued)

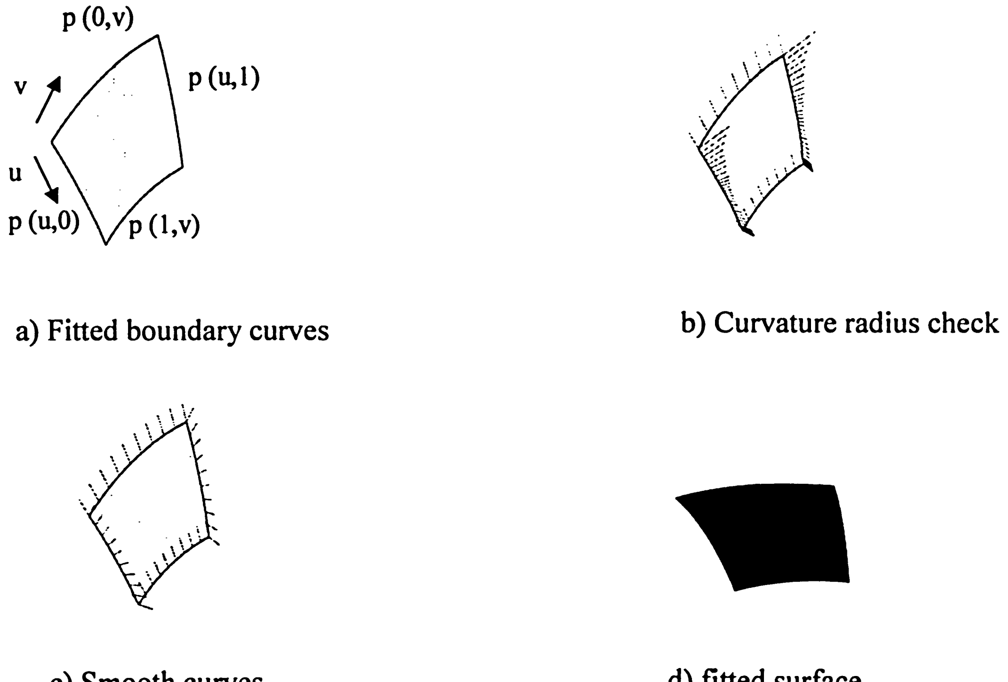
# Appendix D: Free-form Surface Reconstruction

When free-form surfaces are reconstructed, the NURBS method seems to be the most effective [47]. The mathematical formula of the two parametric (u,v) function can be expressed [48-50]:

$$S(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}W_{ij}P_{ij}N_{i,p}(u)N_{i,p}(v)/\sum_{i=0}^{m}\sum_{j=0}^{n}W_{ij}N_{i,p}(u)N_{j,q}(v)\text{, } u,v\in[0,1]; \qquad (6.1)$$

where $P_{i,j}$ are control points, $W_{i,j}$ are weights, $N_{i,p}(u)$ and $N_{j,q}(v)$ are B-spline basic functions.

A example of a free-form reconstruction is shown in Figure A.2.



a) Fitted boundary curves

b) Curvature radius check

c) Smooth curves

d) fitted surface

FigureA.2 Surface reconstruction by boundary curves

# REFERENCES

1. T. Varady, RR Martin, J. Cox, "Reverse engineering of geometric models-an introduction", Computer-Aided Design, 29(4), pp. 255–268, 1997.
2. Y. H. Chen and Y. Z. Wang, "Genetic algorithms for optimized retriangulation in the context of reverse engineering", Computer-Aided Design, 31(4), pp. 261–271, 1999.
3. KH Qin, WP Wang, ML Gong, "A genetic algorithm for the minimum weight triangulation", Proceedings of the IEEE Conference on Evolutionary Computation, 13–16 April, Indianapolis, IN, USA, 1997, pp. 541–546.
4. A. Alrashdan, M. Saeid, "Automatic segmentation of digitized data for reverse engineering applications", IIE Transitions (2000) 32, pp. 59-69.
5. Y. Jun, V. Raja, "Geometric feature recognition for reverse engineering using neural network", Advanced Manufacturing Technology (2001) 17, pp. 462-470.
6. M. Yang, E. Lee, "Segmentation of measured point data using a parametric quadric surface approximation", Computer-Aided Design 31(1999), pp. 449-457.
7. R. Hoffman and Jain, "Segmentation and classification of range images", IEEE Transactions on Pattern Analysis and Machine Intelligence 9 (1987), pp. 608-620.
8. J. Koh, M. Suk, "A multilayer self-organizing feature map for range image segmentation", Neural Networks 8(1) (1995): 67-86.
9. M. B. Suchendra, J. Koh, "A hierarchical neural network and its application to image segmentation", Mathematics and Computers in Simulation 41(1996): 337-355.
10. G. Sugata, M. Rajiv, "Range surface characterization and segmentation using neural networks", Pattern recognition 28(1995): 711-727.
11. H. Woo, E. Kang, "A new segmentation method for point cloud data", Machine Tools& Manufacture 42 (2002): 167-178.

12. P.J. Besl, R.C. Jain, "Segmentation through variable-order surface fitting", IEEE Transactions on Pattern Analysis and Machine Intelligence 10 (2) (1988): 167-192.
13. M.J. Milory, C. Bradley, G.W. Vickers, "Segmentation of a wrap-around model using an active contour", Computer-Aided Design 29(4) (1997): 299-320.
14. T. Fan, G. Medioni, R. Nevatia, "Segmentation description of 3-D surfaces", IEEE Transactions on Robotics and Automation RA-3(6) (1987):527-538.
15. SH Lee, HC Kim, "STL file generation from measured point data by segmentation and Delaunay triangulation", Computer-Aided Design 34 (2002): 691-704.
16. N. Yokoya, M.D. Levine, "Range image segmentation based on differential geometry: a hybrid approach", IEEE Transactions on Pattern Analysis and Machine Intelligence 11(6) (1997): 643-649.
17. D. Zhao, X. Zhang, " Range-data-based object surface segmentation via edges and critical points", IEEE Transactions in Image Processing 6(6) (1997): 826-830.
18. Y. Lee, S. Park, "A robust approach to edge detection of scanned point data", Advanced Manufacture Technology (2004) 23:263-271.
19. V.H. Chan, C. Bradley, G.W. Vickers, "a multi-sensor approach to automating co-ordinate measuring machine-based reverse engineering", Computers in Industry 44 (2001): 105-115.
20. M P. do Carmo, "Differential Geometry of Curves and Surfaces", Prentice-Hall, Inc., 1976.
21. P. Sander and S. Zucker, "Inferring differential structure from 3-D image", IEEE Trans. Patt. Anal. Machine Intell., 1990; 12(9):833-854.
22. FP Ferrie, J Lagarde, P. Whaite, "Darboux frames, snakes, and super-quadrics: geometry from the bottom up", IEEE Transactions on Pattern Analysis and Machine intelligence 1993;15(8):771-784.
23. T Surazhsky, E Magid, O Soldea, " A Comparison of Gaussian and Mean Curvatures Estimation Methods on Triangular Meshes", IEEE Sept. 2003 vol. 1:1021-1026.

24. C. S. Dong, G.Z. Wang., " Curvatures estimation on triangular mesh", JZUS 2005(6):128-136.
25. TD. Gatzke, CM. Grimm, "Estimating curvature on triangular meshes", International Journal of Shape Modeling 2005 1-23.
26. S. T. Welstead, "Neural Network and fuzzy logic Applications in C/C++", John Wiley&sons, Inc, Toronto, 1994.
27. KH Qin, WP Wang, ML Gong, "A genetic algorithm for the minimum weight triangulation", Proceedings of the IEEE Conference on Evolutionary Computation, 13–16 April, Indianapolis, IN, USA, 1997, pp. 541–546.
28. LL. Schumaker, "Triangulations in GAGD", IEEE Computer Graphics& Applications. 1993:1.
29. S.-M. Hur1, H.-C. Kim1 and S.-H. Lee, "STL File Generation with Data Reduction by the Delaunay Triangulation Method in Reverse Engineering", Advanced Manufacturing Technology 2002(19):669-678.
30. M Soucy, D Laurendeau, "A general surface approach to the integration of a set of range views", IEEE Pattern Analysis and Machine Intelligence 1995; 17(4):344-58.
31. G Turk, M Levoy, "Zipped polygon meshes from range images", Proceedings of Siggraph, 1994, p351-358.
32. W. Sun, C. Bradley, "Cloud data modeling employing a unified, non-redundant triangular mesh", Computer-Aided Design 33(2001) 183-193.
33. W. G. Heinrich, " Differential Geometry", New York, 1977.
34. R. Murray and Spiegel, " Mathematical Handbook of Formulas and Tables", McGraw-Hill, New York 2001.
35. HJ Bartsch, "Handbook of Mathematical Formulas", Academic Press, Inc. New York, 1974.
36. T Kohonen, "Self-organizing Maps", 2nd Edition Springer-Verlag Berlin Heidelberg New York, 1997.

37. J A. Freeman, DM. Skapure, " Neural Networks Algorithms, Applications, and Programming Techniques", Addison-Wesley publishing Company, Inc. 1992.
38. K Gurney, "AN Introduction to neural networks", University of Sheffield, 1997.
39. C. Chappuis, A. Rassineux, P. Breitkopf, " Improving surface meshing from discrete data by feature recognition", Engineering with Computers (2004) 20:202-209.
40. M Beaver, " Introduction to probability and statistics", 9th edition, 1994.
41. AD. Kulkarni, "Computer vision and fuzzy-neural systems", Prentice Hall 2001.
42. V.H. Chan, "Artificial Intelligence for Mechanical Engineers", Class Lecture notes, March 2005, Ryerson University.
43. RJ. Schalkoff, "Artificial neural networks" 1997: 157-162.
44. http://citeseer.ist.psu.edu/482177.html
45. http://www.eurometros.org/gen_report.php?category=distributions&pkey=14&subfo rm=yes.
46. M Arshad, "Feature recognition in geometric reverse engineering", MASc thesis, Department of Mechanical Engineering. Ryerson University, 2004.
47. A. Werner, K. Skalski, "Reverse engineering of free-form surfaces", Materials Processing Technology 76(1998):128-132.
48. L. Piegl and W. Tiller, "Algorithm for approximate NURBS skinning", Computer-Aided Design, 28(9), pp. 699–706, 1996.
49. L. pigel, T. Wayne, "Curve and surface constructions using rational B-spline. Computer-Aided Design", 19(9) (1987) 485-498.
50. G. Farin, "Curves and surfaces for computer aided geometric design", A practical guide, 2nd ed., Academic Press, New York 1990.