

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Feature Based Techniques for Point Sampled Surface Models

Liang Luo

A Thesis

in

The Department

of

Computer Science

**Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada**

March 2004

© Liang Luo, 2004



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-91074-1

Our file Notre référence

ISBN: 0-612-91074-1

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Feature Based Techniques for Point Sampled Surface Models

Liang Luo

Recent advances in 3D acquisition technologies have resulted in a large and growing body of 3D point datasets. Such datasets are typically densely sampled points on the surfaces of the object. They are unstructured, lack connectivity information, and are increasingly becoming very large with upwards of several million of points. Current techniques for handling such large point datasets have evolved from several decades of research in triangle meshes. However, there is increasing interest in techniques that deal with these point sets directly without having to create any underlying mesh or surface information in advance.

The primary objective of the research reported in this thesis is to develop new techniques for dense point sampled surface models that directly work on the point samples. Rather than using a deterministic approach, our techniques use a statistical approach of associating properties with points based on the distribution of the point samples in a local neighborhood. Based on this research, our main thesis can be stated as follows: We can classify points using the statistical techniques of principal component analysis (PCA) into different categories such as flat, corner, crease and border. This classification can then be used to devise efficient techniques for processing such dense point sampled models. Specifically we have developed and tested new techniques for efficient rendering and reverse engineering the boundary representation of such dense point sampled models.

Rendering efficiency is considerably improved by using stochastic sampling that is controlled using various model features and view dependent image space properties. The boundary reconstruction is a rather more complex process and is performed in the five stages – classification, model edge detection, edge refinement, loop detection and half-edge representation of model features, and lastly face level segmentation of point sets using a seed fill like algorithm in 3D. Implementation and testing of these techniques required extensive graphics software development effort (over 30K lines of C

and OpenGL code). While the rendering technique works well for all classes of model shapes, the reconstruction technique is most suited for engineering models which have planar faces and sharp edges.

Acknowledgement

I've been fortunate to be able to get all the help I've needed to complete this work from my supervisor, my colleagues, and a lot of other academic researchers. I thank them all for their direct or indirect valuable help to me in completing this work.

I should like to thank particularly my supervisor Professor S. P. Mudur, who has contributed his precious time to help me, this work would have not been possible without his stimulating suggestions and encouragement.

Very special thanks to Mr. Sushil Bhakar, a doctoral student of my supervisor, who worked together with me on the statistical rendering.

Thanks to Stanford University Computer Graphics Laboratory, the sample 3D scanned data are downloaded from their website. Thanks to Mr. Hao Zhou, a doctoral student of my supervisor, who offered me the Anvil model.

I would also like to thank the more than 40 academic researchers whose preceding research works have stimulated me in this thesis. All their valuable works have been referred in this work.

Finally I thank my wife Yaqin for her love and encouragement, which supported me in this work.

Table Of Contents

1	Introduction.....	1
1.1	Point Sampled Surface Models.....	1
1.2	Point Sampled Surface Techniques	4
1.3	The Problem of Rendering Dense Point Sampled Surfaces.....	4
1.4	The Problem of Recovering a Boundary Representation from Dense Point sampled Surface Models	5
1.4.1	Reverse Engineering	5
1.4.2	Simplification.....	6
1.4.3	Feature Extraction.....	6
1.4.4	Surface Reconstruction	7
1.5	A brief Introduction of Point Sample Surface Data.....	7
1.6	Objectives and Main Contributions of this Research	8
1.7	Organization of this Thesis	10
2	A Survey of Techniques for Processing Point Set Models	12
2.1	Overview of Processing Techniques for Point Sampled Surfaces.....	12
2.2	Some Basic Methods for Dealing with Point Sampled Geometry.....	14
2.2.1	Delaunay Triangulation Algorithm.....	14
2.2.2	Voronoi Diagrams in 3D.....	15
2.2.3	Least Squares Fitting Method	16
2.2.4	Moving Least Squares Fitting Method	16
2.2.5	Principal Components Analysis:.....	17
2.2.6	Eigen-Analysis Algorithm	18
2.3	Boundary Representation.....	19
2.3.1	Winged-Edge Data Structure	19
2.3.2	Half-Edge Representation.....	20
2.4	Previous Work on Processing Techniques for Point Cloud Models.....	21
2.4.1	Simplification.....	21
2.4.2	Surface Reconstruction	24
2.4.3	Modeling.....	25
2.4.4	Segmentation.....	26
2.4.5	Feature Extraction.....	26
2.4.6	Range and Image data segmentation.....	27
2.5	Observations	28
3	Stochastic Technique for Point Rendering	29
3.1	Normal Estimation.....	30
3.2	Stochastic Computing of Visual Cues	31
3.3	Rendering Process.....	34
3.3.1	Construction of oct-tree:	34
3.3.2	Rendering by stochastic sampling:	35
3.3.3	Implementation Heuristics.....	38
3.4	Some Remarks on Stochastic Rendering	41
4	Edge Recovery from 3D Point Cloud Models	43
4.1	Classification of Points	44

4.1.1	Initial Classification Process.....	45
4.1.2	Classification Distribution:	48
4.1.3	Refining Points Classification:	52
4.1.4	Neighborhood size:	56
4.2	Edge Traverse	58
4.2.1	SurfaceCorner Point Adjustment:	58
4.2.2	Direction of Edge Points	59
4.3	Edge Analysis:	61
4.3.1	Edges Analysis Objective:	61
4.3.2	No gap along the edge.....	62
4.3.3	No intersecting edge	64
4.3.4	No parallel edges.....	65
4.3.5	SurfaceCorner Points Merge Operation.....	65
4.3.6	Edge Combination	69
4.4	Remarks	72
5	Face Recovery Techniques for Point Cloud Models	73
5.1	Half Edge Data Structure	73
5.1.1	Vertex/Half Edge Structure:	75
5.1.2	Half Edge/Half Edge Loop/Face Structure:	77
5.1.3	Three types of half edge loops to split surface:	78
5.2	Surface Split Algorithm	83
5.2.1	Types of points used in surface splitting:	83
5.2.2	A 3D Seed Fill Algorithm:.....	85
5.2.3	Close-Boundary Vertex Processing:	87
5.2.4	Close-Boundary Vertex Processing Algorithm:	90
5.3	Remarks	93
6	Implementation Issues and Results.....	94
6.1	Program Structure/Workflow.....	94
6.2	Information of each process.....	97
6.2.1	Cube Establish Process	97
6.2.2	Processing Noise in Data	98
6.2.3	Eigen Matrix Computation Process	101
6.2.4	Using Eigen Values in Rendering Process.....	101
6.2.5	Initial Classification Process.....	103
6.2.6	Classification Refining Process	105
6.2.7	Edge Traversing Process.....	107
6.2.8	Edge Refining	108
6.2.9	Face Splitting Process	110
6.2.10	Time consuming Process	111
7	Conclusions and Potential for Further Work	112
7.1	Implementation Issues	115
7.2	Robustness Issues.....	116
7.3	Potential Extensions.....	116
8	References:.....	118

List Of Figures

Figure 2-1: Diagrammatic Overview of Point Sampled Model process.....	13
Figure 2-2: Voronoi Diagram.	15
Figure 2-3: Winged-edge data structure	20
Figure 2-4: Half Edge Data Structure	21
Figure 3-1: Region classification using different thresholds for classifying flatness.....	32
Figure 3-2: Silhouette Containment.....	33
Figure 3-3: Oct-tree (right) for point sampled surface (left).....	35
Figure 3-4: Rendering at different image sizes.....	41
Figure 4-1: Initial Classification	46
Figure 4-2: Planar Estimation Distributions.	49
Figure 4-3: SurfaceCrease Estimation Distributions.	50
Figure 4-4: SurfaceBorder Estimation Distributions.	51
Figure 4-5: Curvature Estimation	54
Figure 4-6: Maximum Open Angle.....	55
Figure 4-7: The average distance distributions.....	58
Figure 4-8: Vacancy Example.	62
Figure 4-9: Three cases of segment vacancies.....	63
Figure 4-10: The first step of the SurfaceCorner points merge operation.	67
Figure 4-11: Final Result of SurfaceCorner points merge operation.....	69
Figure 4-12: The distance from one point to 3D line.....	70
Figure 5-1: Vertex/Start Vertex/Half Edge Organization.....	76
Figure 5-2: Half Edge/Half Edge Loop/Face Organization.....	78
Figure 5-3: First case of Half Edge Loop Type A.	79
Figure 5-4: Second case of Half Edge Loop Type A.....	80
Figure 5-5: Half Edge Loop Type B.	81
Figure 5-6: Half Edge Loop Type C.....	82
Figure 6-1: The overview of the system.	96
Figure 6-2: The noise analysis of the dragon.....	99
Figure 6-3: The noise analysis of Happy Budda model.....	100
Figure 6-4: Rendering at different image sizes.....	102
Figure 6-5: The initial classification information of each model.	104
Figure 6-6: The refined classification information of each model.	106
Figure 6-7: The result of the edge refining.	109
Figure 6-8: All faces are recovered.....	110

List Of Tables

Table 1-1: Sample 3D Point Cloud Models.....	3
Table 3-1: Rendering algorithm parameters.	39
Table 3-2: Number of samples varying with image size.	41
Table 6-1: Processes of the whole system	95
Table 6-2: The cube structure information.	97
Table 6-3: The Neighborhood vertices information.	98
Table 6-4: Noise Analysis Data	98
Table 6-5: Eigen Matrix Calculation Information	101
Table 6-6: Initial Classification Result.	103
Table 6-7: The initial classification and the refined classification comparison.....	105
Table 6-8: The edge data after traversing.	107
Table 6-9: The comparison between the traversing edges and the refined edges....	108
Table 6-10: The face recovering information.	110
Table 6-11: The time costs of each step.....	111

1 Introduction

1.1 *Point Sampled Surface Models*

Point sampled surface models are essentially point sets that are obtained as a result of sampling the surface of any three dimensional object. The sampling process may be the result of the use of 3D scanning devices that are becoming increasingly affordable and accurate these days. Or it could be the result of scientific computations simulating complex physical phenomena, primarily to benefit from some of the advantages that such representations provide further down the processing pipeline. The net impact of this has been that such representations, with a 3D point as the primitive representation are receiving a growing amount of attention as a representation of 3D surface models in computer graphics. A further motivating factor is that highly detailed surfaces require a large number of small primitives, which contribute to less than a pixel when displayed, so that points become an effective display primitive.

Point sets are typically densely sampled points on the surfaces of the object. They are unstructured, lack connectivity information, and are increasingly becoming very large with upwards of several million of points. In order to make such data sets useful for graphics applications, a set of surface attributes could also be associated with each sample point, most importantly: a normal vector, color information, and a conservative estimate for the local sampling density. Historically, it is clear that the use of points as a universal representation was not a difficult concept to devise or justify given the trend towards richer graphics content with extreme geometric

complexity. However, the implementation of effective geometry and graphics processing techniques was never simple, with limited memory and computing power that was available in earlier days. While on current hardware this is much less of a problem, dense point sampled surface models are still much too large for a number of tasks such as real time rendering, shape editing, geometric interactions, etc. Table 1-1 shows some statistics of a few example models that we have used for experimental purposes in our research. It must be noted that these model sizes are not necessarily representative. In practice sizes could be very much larger. The interested reader is referred to the Digital Michelangelo project in Levoy et al. [27].


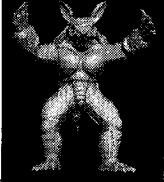






Picture	Model Name	Number of Points	Source
	Bunny	35947	Stanford University Computer Graphics Laboratory
	Armadillo	172,974	Stanford University Computer Graphics Laboratory
	Happy Buddha	543,652	Stanford University Computer Graphics Laboratory
	Dragon	566,098	Stanford University Computer Graphics Laboratory
	Lucy	14,027,872	Stanford University Computer Graphics Laboratory
	Single Box	60,002	Made by myself.
	Dual Box	74,402	Made by myself.
	Anvil-1	162,882	Offered by Hao Zhou

Table 1-1: Sample 3D Point Cloud Models

1.2 Point Sampled Surface Techniques

The point as a graphics rendering primitive was first proposed in Levoy et al. [27]. In recent years the interest has grown beyond rendering and the point is being proposed as the shape representation primitive. In this larger context a number of processing tasks need to be performed on point set based 3D surface models. Major challenges in this type of representation are the handling of large amount of data produced and its interactive rendering. There are basically three broad categories of not necessarily independent techniques that are being researched. These are:

1. Real time rendering at desired resolutions
2. Reverse engineering for obtaining more abstract representations, such as triangle mesh representations, implicit or parametric algebraic representations for the underlying surfaces and boundary representations.
3. Simplification for reducing model data sizes and enabling more efficient model processing.

In this research we have focussed on real time rendering and reverse engineering to obtain boundary representation of the point set.

1.3 The Problem of Rendering Dense Point Sampled Surfaces

Straight forward rendering of the point sampled surface would involve the following for every 3D point in the set:

1. Determine the pixel in the display screen on which this 3D point projects under the current view parameters.
2. Get the normal vector to the underlying surface at this point, the

color/material properties, and texture map values if available.

3. Compute the color and assign to this pixel.

The main difficulties with this problem are the following:

- If the sampling density is very high as compared to the display image size, it may happen that a very large number of pixels map onto the same pixel, resulting in considerable slowing of rendering speeds for virtually no gains in image quality.
- If the sampling density is low, say when zooming into a region, then many pixels in the image may not be covered by any of the 3D points causing holes in the final image. One of the techniques suggested to overcome this problem is to define small regions around each point (or pixel in image space) referred to usually as surface splats, such that a continuous surface can be displayed by accumulating or blending these splats in image space. The other approach is to derive the equation for the underlying surface locally and super sample the local surface equation to generate as many points as needed for a continuous surface image.

1.4 The Problem of Recovering a Boundary Representation from Dense Point sampled Surface Models

1.4.1 Reverse Engineering

Reverse Engineering (RE) usually starts with the acquisition of digital point surface sampled data from the physical/master model and then developing surfaces on this point set data for other engineering purposes. Reverse Engineering involves more

than the mere use of the latest scanning technology and/or the latest surface fitting software. The end result of the RE process must depict the most accurate and usable dataset, for the true representation of the physical product definition.

There are up to three steps in the process of reverse engineering. The first step is to use some input device or technique to collect the raw geometry of the object. This data is usually in the form of (x,y,z) points on the object relative to some local coordinate system. These points may or may not be in any particular order. The second step is to use a computer program to read this raw point data and to convert it into a usable form. This step is not as easy as it might seem. The third step is to transfer the results from the reverse engineering software into some 3D modelling or application software so that you can perform the desired action on the geometry. Sometimes, steps 2 and 3 can be done inside one program.

1.4.2 Simplification

Point sample surface data usually are acquired with a range scanner. It always performs multiple scans; each scan is in its own coordinate system, and combines all scans together by registering the scans in order to get a complete object.

The main function of surface simplification is to reduce the point set data size. The benefits are (1) optimizing rendering performance, (2) the subsequent surface reconstruction becomes significant efficient and faster.

1.4.3 Feature Extraction

Point sample surface is a surface described by a set of sample points without further topological information such as the triangle mesh connectivity or a

parameterization. Furthermore, the point sample data is usually a very large data set. In order to be used for graphic process or rendering system, geometry information is necessary. Geometric feature includes vertices, edges, loops and faces of the original object.

Feature extraction is the process of feature detecting. These processes usually detect line-type feature or surface feature from the point sample data set. The geometric information is very helpful for further graphic processing.

1.4.4 Surface Reconstruction

Surface Reconstruction is also called surface recovery in Hoppe et al. [19], Savadjiev et al [46].

Surface reconstruction is to find an algebraic surface representation that approximates a physical surface by using a set of point coordinates sampled from the surface. These point coordinates may be corrupted with noise, due to imperfections in the acquisition of the data.

1.5 *A brief Introduction of Point Sample Surface Data*

The Point Sample Surface is created by scanning the surface of an object, and it is unorganized point clouds derived from laser scanner data or photogrammetric image measurements. The Point Sample Surface Data are always very big data without any geometry information. It is hard to use Point Sample Surface Data directly by graphics applications.

The Point Sample Surface Data has three kinds of model, one is derived from image range data, the second one is multi-view point cloud model and the third one is close surface point cloud model.

The Point Sample Models are separated to engineer structure model and nature model according to the source object. Engineer structure model are composed of many engineer objects, and each of them is composed of the primitives. The nature models are coming from the nature objects, and they are always composed of very complex curve surfaces.

1.6 Objectives and Main Contributions of this Research

Current techniques for handling such large point datasets have evolved from several decades of research in triangle meshes. Creating a triangle mesh representation of a dense point set representing the closest simplicial complex approximation of the underlying surface is a computationally complex task. It is also highly dependent on the sampling density and could be prone to errors if there is wide variation in sampling density. There is a need for efficient techniques that can process and render large point sets directly without having to explicitly digital representations of the underlying surface geometry. This is the primary objective of the research reported in this thesis. It is to develop new techniques for dense point sampled surface models that directly work on the point samples. Rather than using a deterministic approach, our techniques proposed in this thesis use a statistical approach of associating properties with points based on the distribution of the point samples in a local neighborhood. Based on this research, our main thesis can be stated as follows: We can classify points using the statistical techniques of principal component analysis (PCA) into different categories

such as flat, corner, crease and border. This classification can then be used to devise efficient techniques for processing such dense point sampled models. Specifically we have developed and tested new techniques for efficient rendering and reverse engineering the boundary representation of such dense point sampled models. Rendering efficiency is considerably improved by using stochastic sampling that is controlled using various model features and view dependent image space properties. The point set data is first preprocessed into a hierarchical oct-tree representation using PCA results, such that each leaf node of the oct-tree is near planar. Next the PCA results and the proximity information present in the oct-tree are used to estimate correctly oriented normals for any point. Finally depending on the view point and properties such as image size, the oct-tree nodes are traversed to the desired depth, and the nodes for display are selected. These nodes are then rendered using a random sampling of the points in that node.

The boundary reconstruction is a rather more complex process and is performed in the five stages – classification, model edge detection, edge refinement, loop detection and half-edge representation of model features, and lastly face level segmentation of point sets using a seed fill like algorithm in 3D. We have had to develop new methods to enable us to reconstruct boundary representation entities from point sample surface data. During reconstruction of the boundary representation, the first stage is the classification of the vertices into 3 classes: flat, crease and corner vertices. This stage consists of analyzing noise data, the fast neighbor vertices collecting, eigenvalues and eigenvectors calculation, and classifying the vertices. The

second stage is generating the crease edges by recursively traversing the crease vertices and corner vertices along the crease vertices' direction. The third stage is refining the crease edges, and this stage consists of splitting intersecting-edges, culling short edge loops and edge merging at the corner vertices. The fourth stage is generating the crease edge loops. For this we use the half-edge data structure described in chapter 2 later. The fifth stage involves segmenting of points into different faces using the crease edge loops. In this stage we have devised an extended of the seed fill algorithm in 2D to 3D surfaces data. Another difficult task to be addressed here is the disambiguation of points close to the edges that are wrongly classified by the statistical process.

Implementation and testing of these techniques required extensive graphics software development effort (over 30K lines of C and OpenGL code). While the rendering technique works well for all classes of model shapes, the reconstruction technique is most suited for engineering models which have planar faces and sharp edges.

1.7 Organization of this Thesis

The rest of this thesis is organized as follows. Chapter 2 first provides a brief overview of basic techniques such as the principal component analysis and half edge representation data structure. This is followed by a comprehensive survey of the different techniques that have been reported in literature for processing point sampled surface models. Chapter 3 of this thesis describes in detail the statistical rendering

technique. It includes examples of the results of our implementation¹. Chapter 4 addresses the problem and our proposed solution for detecting and refining edge features in the model, and Chapter 5 similarly addresses segmentation of points into faces. Extensive implementation work was involved. All implementation was carried out in C with OpenGL used for 3D graphics. Relevant details of this implementation are presented in Chapter 6. Chapter 7 presents our conclusions and potential for future work.

¹ This implementation was joint effort with Mr. Sushil Bhakar, a doctoral student of my supervisor.

2 A Survey of Techniques for Processing Point Set Models

Point sampled surface data may originate from a number of sources. The data could be from a 3D scanner or the result of scientific simulations. Data from 3D scans could be processed in raw format or could be the processed result of merging raw scan data from different views. In the latter case, the point set data will be unorganized and there may not be any connectivity or spatial proximity information present in the structure of the data. The originating source also plays a significant role when devising a processing technique for such models. In this chapter we first give an overview of processing techniques for point sample data. Then we present some basic methods that are used by these techniques. Lastly we provide a comprehensive survey of techniques devised for irregularly sampled surfaces, the originating source of data that is of interest to our research.

2.1 Overview of Processing Techniques for Point Sampled Surfaces

The point sample surface data could be in two forms. One is image range data from laser range scanners, the second one is a point cloud model, either modeling a closed or open surface. Image range data consists of regularly spaced samples in a two dimensional domain, with a depth value associated with each sample point. Hence these are also referred to sometimes as depth images. Point cloud data is irregularly spaced and sampling density may vary considerably over the complete surface of the model. The output resulting from processing of these point sets can also be different depending on the process and the desired end application. A rendering process will result in an image, usually a color shaded picture of the 3D model. On the other hand,

geometry recovery processes would give us many different kinds of models: 3D triangle mesh model, piecewise implicit or piecewise parametric algebraic surface model, boundary representation of the model in terms of vertices, edges and faces (B-Rep model), etc. Simplification and boundary feature extraction are other processing tasks that could give us different outputs. These processing techniques are not isolated and can be used in sequence. Those are one may simplify, then carry out surface reconstruction, and then do point sample rendering. One may apply point sample rendering directly, or feature extraction and then surface construction etc. Figure 2-1 gives a diagrammatic representation of the data formats, processing techniques and outputs.

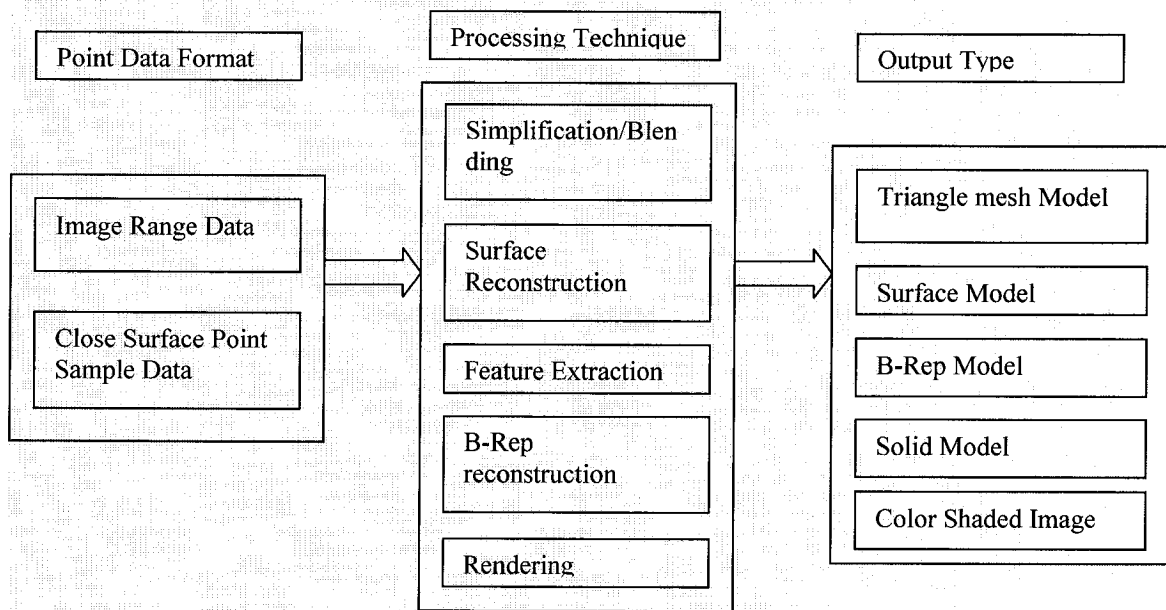


Figure 2-1: Diagrammatic Overview of Point Sampled Model process.

Various algorithms with different capabilities and guarantees have been proposed for these different processing techniques. The process of turning a set of

sample points into a computer graphics model generally involves several steps: the reconstruction of an initial piecewise-linear model, simplification, and perhaps fitting with curved surface patches.

Undersampling happens when a surface has small features such as high curvatures that are sampled inadequately. It cannot be avoided when a surface is not smooth. In this case no finite sampling is dense enough for sharp edges or corners. Even the presence of boundaries in the surface can be thought of as being a consequence of undersampling.

Oversampling causes difficulties, particularly in post-processing. A surface is sampled with unnecessarily high density. Surfaces reconstructed from an unnecessarily dense sample contain large numbers of geometric entities and thus become unwieldy for further processing such as graphic rendering or finite element analysis.

2.2 Some Basic Methods for Dealing with Point Sampled Geometry

2.2.1 Delaunay Triangulation Algorithm

A number of point sample processing techniques reported in the literature use the classic Delaunay Triangulation Algorithm to create the 3D triangle mesh connecting the points in the sampled model in [Gumhold 14], [Amenta 1], [Amenta 2]. The triangle mesh imposes topological connectivity and a neighborhood relationship on the sample points. In general, 3D triangulation is a very complex problem as it first needs to create tetrahedra in the 3D space and then to extract the surface triangles. For very large models with millions of points, many degenerate geometric conditions could arise and it is not clear if the reported techniques will take acceptable computation

times or even work correctly. Many papers therefore just assume that the triangulation of the surface of the point cloud model is given and describe further processing based on the connectivity and proximity information provided by such a triangle mesh model.

2.2.2 Voronoi Diagrams in 3D

Voronoi diagrams are a fundamental structure in geometric computing.

Voronoi tessellations are duals of Delaunay triangulations. They provide a polyhedral partitioning of the space so that points in space nearest to any sample point are guaranteed to lie within the convex polyhedron containing the sample point. Like Delaunay triangulation, robust and efficient implementations of Voronoi methods that can work with very large point cloud models are difficult. Fig 2-2a shows the 2D Voronoi tessellation. Figure 2-2b shows the delaunay triangulation of all the points and then the boundary of the original point set. (Figure source: Amenta et al. [1],[2]).

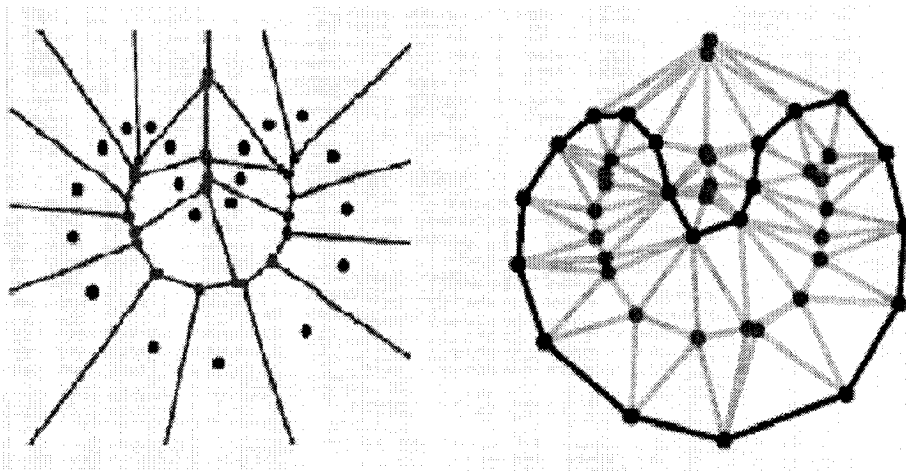


Figure 2-2: Voronoi Diagram.

2.2.3 Least Squares Fitting Method

Least Squares Fitting Method is a popular method used in the 3D point cloud processes. It is usually used for fitting the sample point dataset to primitive surfaces such as planes, quadric surfaces, etc. The least squares fitting method is based on minimize the expression:

$$S = \sum_{i=1}^n [f(x_i, y_i, z_i)]^2$$

$f(x, y, z) = 0$ is the algebraic representation of the geometry primitive.

Feddma et al. [12] describe how to fit large range data sets to geometric primitives such as planes, cylinders, spheres, ellipsoids and other quadric surfaces.

2.2.4 Moving Least Squares Fitting Method

Moving Least Squares Fitting Method is usually used for approximating the local planar surface in the process of refining or simplifying the point cloud data.

Levin et al. [26] describes the moving least squares fitting algorithm. Let $\{x_i\}_{i=1}$ be some data values at these points. The moving least-squares approximation of degree m at a point $x \in \mathbb{R}^d$ is the value $p(x)$ where $p \in \Pi_m^d$ is minimizing, among all $p \in \Pi_m^d$, the weighted least-squares error

$$\sum_{i=1} (\mathbf{p}(x_i) - \mathbf{f}(x_i))^2 \theta(\|x - x_i\|)$$

Throughout, θ is a non-negative weight function, $\|\bullet\|$ is the Euclidean distance in \mathbb{R}^d and Π_m^d is the space of polynomials of total degree m in \mathbb{R}^d . The approximation

is made local if $\theta(s)$ is rapidly decreasing as $s \rightarrow \infty$, or is of finite support, and interpolation is achieved if $\lim_{s \rightarrow 0} \theta(s) = \infty$.

2.2.5 Principal Components Analysis:

Principal components analysis (PCA) was originally introduced as far back as 1901 by Karl Pearson and found its first use or rather misuse in the analysis of intelligence tests.

The basic idea of the method is to describe the variation of a set of multivariate data in terms of uncorrelated (linearly independent) variables each of which is a particular linear combination of the original variables. The new variables are derived in decreasing order of importance so that, for example, the first principal component accounts for as much as possible of the variation in the original data. The objective of this analysis is usually to see whether the first few components account for most of the variation in the data. If so, it is argued that they can be used to summarize the data with little loss of information, thus providing a reduction in the dimensionality of the data, which may be useful in simplifying later analysis.

PCA summarizes the variation in a correlated multi-attribute into a set of uncorrelated components, each of which is a particular linear combination of the original variables.

Principal component analysis (PCA) involves a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called *principal components*. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

The mathematical technique used in PCA is called eigen analysis. The eigenvector associated with the largest eigenvalue has the same direction as the first principal component. The eigenvector associated with the second largest eigenvalue determines the direction of the second principal component. The eigenvector associated with the least eigenvalue determines the direction of the third principal component.

2.2.6 Eigen-Analysis Algorithm

The mean vector of the population is defined as $\mathbf{m} = E\{\mathbf{x}\}$, where $E\{\arg\}$ is the expected value of the argument. Covariance matrix C of the vector population is defined as $C[\mathbf{x}] = E\{(\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^t\}$.

We denote the 3 eigen values of this covariance matrix as $\lambda_0, \lambda_1, \lambda_2$ where

$$\lambda_0 \leq \lambda_1 \leq \lambda_2$$

- **Eigenvalues** measure the amount of the variation described by each principal component (PC) and will be largest for the first PC and smaller for the subsequent PCs.
- **Eigenvectors** provides the vectors for the uncorrelated PC, which are the linear combinations of the centered standardized or centered un-standardized original variables.

Eigenvector and Eigenvalue Calculation:

Jacobi's method and QR iteration are two of the most common algorithms for solving eigenvector and eigenvalue in Sleijpen et al. [50].

2.3 Boundary Representation

A common way to represent a polygon mesh is a shared list of vertices and a list of faces storing pointers for its vertices. This representation is both convenient and efficient for many purposes, however in some domains it proves ineffective.

For a solid model or a close surface model, there are always many queries that need the adjacency relationships between the components of the mesh, for example, the faces, the edges and the vertices. To implement these types of adjacency queries efficiently, the boundary representations (b-reps) have been developed which explicitly model the vertices, edges, and faces of the mesh with additional adjacency information stored inside.

The two common types of boundary representation are Winged-Edge Representation and Half-Edge Representation.

2.3.1 Winged-Edge Data Structure

Winged-edge data structure is developed by Baumgart in Mantyla et al. [30]. Each edge in Winged-edge data structure is represented by pointers to its two vertices, to the two faces sharing the edge, and to four of the additional edges emanating from its vertices. Each vertex has a back ward pointer to one of the edges emanating from it where as each face points to one of its edges.

The Winged-Edge Data Structure makes it possible to determine in constant time which vertices or faces are associated with an edge, but it takes longer to query the adjacent relationships.

As show in the Figure 2-3, for the edge E1, its two end vertices, two faces sharing the edge E1 and four edges emanating from E1's end vertices are stored in the E1's data structure.

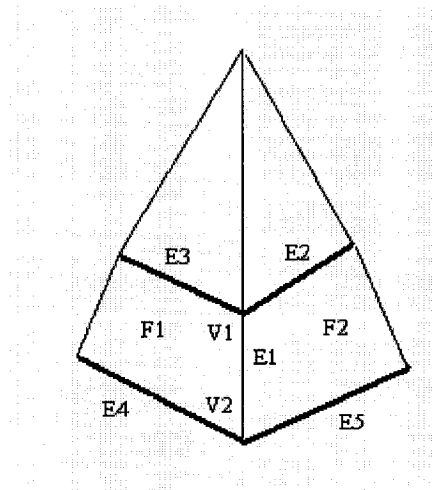


Figure 2-3: Winged-edge data structure

2.3.2 Half-Edge Representation

A halfedge data structure is an edge-centered data structure capable of maintaining incidence information of vertices, edges and faces. Each edge is decomposed into two halfedges with opposite orientations. One incident face and one incident vertex are stored in each halfedge. For each face and each vertex, one incident halfedge is stored. (Figure 2-4 Source: Kettner et al. [25].)

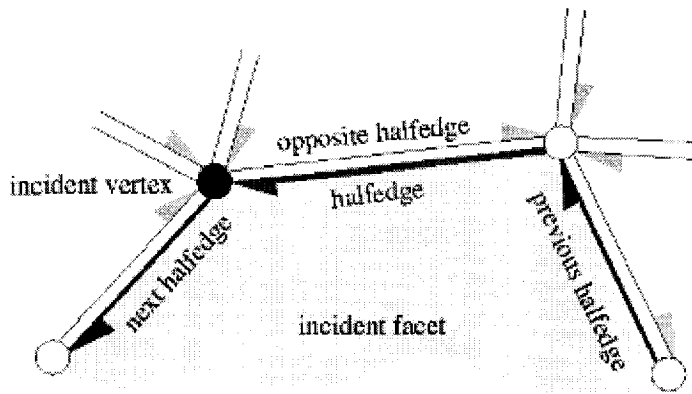


Figure 2-4: Half Edge Data Structure

2.4 Previous Work on Processing Techniques for Point Cloud Models

In recent years, there are a very large number of papers reported in the literature on techniques for processing 3D point cloud data. Below we provide a survey of techniques that are relevant to our research.

2.4.1 Simplification

A major issue that needs to be addressed due to very high sampling density is that, in processing of these models, many points contribute to the same output value. For example, in the rendering process many points will project onto a single pixel. Hence a major pre-processing task that has been the focus of work with such models is that of creating a simplified version of the original sampled set. This can be stated as follows: Given the original sampled set S and its simplified version S' , for any process, say P , we would like to find that S' which minimizes the error $[P(S), P(S')]$, for a given constraint, such as $|S'| \leq \text{a given number}$ or $\text{error } [P(S), P(S')] < \text{a given}$ Cignoni et al. [10]. So, for surface reconstruction, this would translate to saying

$$error[reconstructed_surface(S), reconstructed_surface(S')]$$

should be minimized. All current simplification methods take the approach that most processing tasks are determined by the underlying surface geometry and hence these methods concentrate on minimizing the following error metric:

$$error[surface_geometry(S), surface_geometry(S')].$$

Simplification methods may choose either to ensure that S' is a proper subset of S or may choose to compute an approximate S' that only minimizes the prescribed error metric. These simplification methods can be broadly classified into three categories as follows:

Set Partitioning – S is partitioned into subsets $\{S_1, S_2, \dots, S_n\}$ such that each subset can be represented by a single sample point according to the desired error metric [PfiSter 40, Brodsky 8, Shaffer 47].

Point Pair Collapsing – Point pairs in S are successively considered and if possible collapsed into a single point, according to the desired metric [Alexa 3, Garland 18, Hoppe 20].

Resampling – New sampling positions are computed, say according to local geometric characteristic such as curvature [Kalaih 24], or say by moving particles on the surface of the original set S simulating inter-particle repelling forces [Witkin 55].

Since for all purposes, S' is now the representation of S , a major concern that all these methods try to address is not to lose any significant property present in the original set S . For example, inadequate samples in S' could result in holes in $rendered_picture(S')$, particularly for highly zoomed-in close-up views. This is avoided by storing a disk of influence on the tangent plane [PfiSter 40] or more

elaborate differential geometric information [Kalaih 24] at each point. The point is then rendered either using flat shading optionally followed by screen-space filtering or by choosing a suitably approximating 3D shape or a splat in screen space [Zwicker 56]. This delicate balance between reducing the size of S' and at the same time, not losing any significant information present in the original sampled set S , often results in very complex pre-processing to be carried out on the original point set. A more detailed review can be found in Pauly [34]

Alexa et al. [3] reduce point cloud redundancy by estimating a point's contribution to the moving least squares (MLS) representation of the underlying surface. Those points contributing the least are subsequently removed.

Pauly et al. [34] adapted a technique of the quadric error metric presented for polygonal meshes. The idea in this paper is to approximate the surface locally by a set of tangent planes and to estimate the geometric deviation of a mesh vertex v from the surface by the sum of the squared distances to these planes. In this paper eigenanalysis of the covariance matrix of a local neighbourhood is used to estimate the local surface properties. Pauly et al. [34] adapted two clustering approaches: incremental clustering by region-growing which is computationally efficient, and hierarchical clustering which is feature sensitive and has less approximation error.

Moenning et al. [33] introduced another simplification technique which is called Fast Marching farthest point sampling. FastFPS algorithm computes discrete Voronoi diagrams in the form of weighted distance maps incrementally directly across the input point set.

2.4.2 Surface Reconstruction

Surface reconstruction is to find a surface that approximates a physical surface by using a set of point coordinates. These point coordinates may be the original point set itself and may be corrupted with noise, due to imperfections in the acquisition of the data. Or they could be simplified point set.

2.4.2.1 Voronoi-Based Surface Reconstruction

Amenta et al ([1] [2]) introduced the crust algorithm that is based on the three-dimensional Voronoi Diagram and Delaunay triangulation; it produces a set of triangles that we call the crust of the sample points. All vertices of crust triangles are sample points; in fact, all crust triangles appear in the Delaunay triangulation of the sample points. The algorithm computes the voronoi diagram of the sample points first; and then generates the triangles by Delaunay triangulation. The algorithm removes the triangles which include more than one voronoi vertices, so it only keeps the connectivity of the sample points. The algorithm can not handle surfaces with sharp edges and boundaries; and it fails when the noise level is roughly the same as the sampling density.

Renner et al. [41] introduced an approach to reconstruct the surface from scattered 3D point sampled surface data using α -shape algorithm, Delaunay triangulation and Voronoi diagram.

2.4.2.2 Other methods Surface Reconstruction

Hoppe et al. [19] adapted a combination of K-nearest neighbors, eigen analysis, Signed Distance Function, EMST graph, and contour tracing methods to reconstruct the surface from unorganized point cloud data. The K-nearest neighbors and eigen

analysis method is used to get the local planar properties, and Signed Distance Function and EMST graph is used to determine geometric properties, and the contour tracing method is used to determine the boundary. However, Hoppe et al. [19] assume the point sample data is ρ -dense and δ -noisy.

Savadjiev et al. [46] introduced a curvature consistency algorithm to reconstruct the surface by iteratively minimizing the function to satisfy the local constraints on the curvature.

2.4.3 Modeling

2.4.3.1 Multiresolution modeling

Pauly et al. [37] introduced an approach for modeling multiresolution representation of the point cloud data. The multiresolution representation is each sample p is represented by a point p_0 plus a sequence of normal displacement offsets d_0, \dots, d_{n-1} . The approach includes decomposition and editing of the multiresolution representation.

2.4.3.2 Shape modeling

Pauly et al. [36] devised this approach for modeling a hybrid geometry representation by combining unstructured point clouds with the implicit surface definition of the moving least squares approximation. The approach is able to perform the large constrained deformation and Boolean operations on the arbitrary shaped objects.

2.4.3.3 B-Rep modeling

Várady et al. [52], [7] present an algorithm for reverse engineering B-Rep models from multiple point clouds. The algorithm includes four major components:

triangulation/decimation, efficiently segmenting point data into regions, reconstructing translational and rotational surfaces with smooth, constrained profiles, generating the B-Rep topology and adding blends.

2.4.3.4 Mesh modeling

Roth et al. [42] presented the marching cubes algorithm to create triangular mesh from multiple views of point data.

2.4.4 Segmentation

Segmentation is the process of partitioning the point data set into subsets where each subset constitutes a distinct feature of the object represented by this point set. These features could be faces, loops, borders, edges, vertices etc.

Benkö et al. [6] introduced the direct segmentation method that is based on a special sequence of tests. The sequence is the hypotheses in the order of simplicity, which also corresponds to the frequency of occurrence. The first process of the direct segmentation method is to split the point cloud into smaller, distinct point regions by separating the sharing sharp edges and the various types of the smooth edges. For each point region various filters to determine planarity, dimensionality, axis direction, apex angle and rotational axis are applied.

2.4.5 Feature Extraction

Gumhold et al. [14] introduced a two stages method to extract feature lines directly from point sample surface data. The first stage consists of Delaunay triangulation/filtering, eigen analysis, minimizing the penalty weights to the each point

and the edges of the neighbor graph. The second stage is to recover the feature lines and junctions by fitting wedges to the crease lines and corners.

Pauly et al. [35] introduced a multi-scale surface variation method to extract the feature of the point sample surface data. Pauly et al. [35] used principal component analysis/eigenanalysis to estimate local surface properties, and used the automatic scale selection to optimize determining feature weights. And by the minimum spanning tree process and active contour models process, the line-type feature is detected.

Weingarten et al. [54] describe a simple and fast algorithm for generating planar feature of indoor environments with a mobile robot. The emphasis lies on the high performance of the algorithm. The algorithm includes three processes: dividing (the input data set is divided into small cubic neighborhoods), local surface detection (Least Square Planar Fitting) and region growing (merging the neighboring cube cell if they are on the similar plane).

2.4.6 Range and Image data segmentation

Stamos et al. [45] introduced an approach for segmenting range and image data. The range and image data is the combination of range (dense depth estimates) and image sensing (color information) datasets. This algorithm segments the planar regions by clustering the locally planar points that have similar orientation and are close in 3D space. The algorithm detects the intersection lines between the planar regions. The last process is registering the intersection lines onto the range image dataset.

2.5 Observations

From the above survey it is clear that techniques for dealing with point sampled surface data have been researched vigorously and continue to be. Such techniques need to be efficient, correct and robust. The very large volumes of data necessitate the need for efficiency. The unstructured nature and absence of proximity information could result in erroneous outputs. And the variation in sampling densities and the presence of noise contribute to problems in robustness. Each of these problems is individually challenging.

3 Stochastic Technique for Point Rendering

In chapter 2 we have described a number of different techniques that have been developed so far for efficient rendering of large point sampled surface data sets. An important characteristic of all these techniques is that they work on a discrete sampling of the surface of the object and yet are deterministic. They compute local features of the underlying surface or fit a local surface and then use it in a deterministic fashion for rendering the 3D object. In our work we have devised a new rendering technique that is based on statistical estimates of local features as in [4]. We propose a scheme for representing feature based details of a given unstructured point sample geometry using a hierarchical statistical analysis of the original dataset. Hierarchical statistical analysis allows us to trade off accuracy against determinism.

A Principal Component Analysis (PCA) analysis of the original point data set allows us to hierarchically represent the model using an oct-tree data structure, recover model details to the extent needed, as well as use randomized rendering to efficiently display the model. Our motivation for this new representation lies in the following three observations: (1) Features exhibit very high coherence in local point neighborhoods. For example, a point belonging to an edge will be close to other points of that edge, (2) the accuracy with which different parts of the object need to be rendered depends on the type of features to be found in that part and also the view from which the part is being rendered. So, for example, parts which have sharp edge features will need to be rendered with greater accuracy than flat parts. Similarly, if a part consisting of hundreds of point samples projects onto a few pixels, then a simplified rendering of this part may very well suffice. 3) The accuracy required to

generate a visually realistic image from a point cloud model can be achieved using statistical methods on a sparse point representation.

3.1 Normal Estimation

All point-based rendering techniques require a correctly oriented normal at every point that is rendered and often this requires topological connectivity or continuous surface information. With each local region of the point set surface, we first compute and associate a representative normal. We then describe a simple method of orienting these representative normals and then use these to determine the correctly oriented normal at any of the point samples chosen for rendering. Sampling itself is controlled by the use of multiple visual cues, both object based and image based, which include flatness of any region of the model, presence of features such as an edge of the model in the region, pixel coverage or rendered image size and silhouette containment. Fewer points are rendered in flatter regions than in highly curved regions or in regions containing an edge. The number of points itself is proportional to the number of pixels covered in the final rendered image. Along the same lines, more points are rendered closer to the silhouette [Sander 48]. Individually each of the above visual cues has been successfully used in rendering and has been reported earlier in literature. However, together they enable considerable computational speed-ups in the rendering process while at the same time not losing any of the information present in the original point sampled set.

The next section describes the details of computing the different visual cues for densely sampled surfaces using stochastic sampling. This is followed by a brief

description of the Hierarchical oct-tree structure and the rendering algorithm itself. We then show some examples from our implementation.

3.2 *Stochastic Computing of Visual Cues*

- Region Flatness:

Flatness in any region of a surface is a significant cue that can be used to optimize rendering. Clearly flat regions can be rendered with fewer samples, unless we wish to capture special effects like specula highlighting. Given a subset of point samples covering a region of the surface, we use the eigen value analysis of the covariance matrix of points described below to determine the local surface curvature variation [23, 14, 22]. If the number of points in this region is very large, then for increased computational efficiency, a more reasonably sized stochastically sampled subset can be used.

We can construct a population of random vectors of the form $x = [x_1, x_2, x_3]^t$ using x,y,z components of point coordinates.

We decide on the flatness of a region by examining the value of the expression $\lambda_0 / (\lambda_0 + \lambda_1 + \lambda_2)$. Smaller the value of this expression, more stringent is the flatness criterion. Fig. 3.1 shows the regions classified according to two different values for this expression, (0.005 and 0.001).

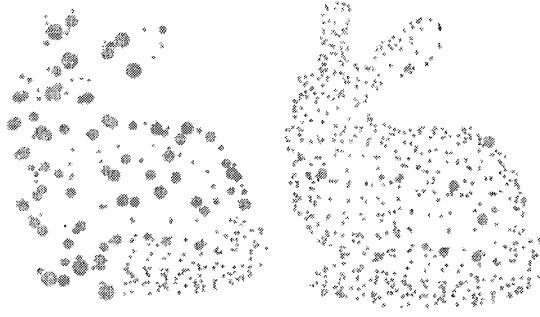


Figure 3-1: Region classification using different thresholds for classifying flatness.

- Edge Containment:

If a region contains an edge of the original surface, then we must choose a larger number of samples to render in that region to avoid aliasing problems. The same method of computing eigen values used for determining flatness can be used to determine the presence of an edge in the region. Point p can be said to be very likely belonging to an edge if $\lambda_0 \approx \lambda_1, \lambda_2 \approx 2\lambda_0$ [14].

In order to estimate the presence of an edge in the region we check a randomly chosen subset of points for being classified as edge points. If none of the chosen points get classified as edge points, then we declare that this region has no edge.

- Pixel Coverage:

The final image size in pixels is another important visual cue that is used to optimize rendering. The number of point samples to be selected for rendering a region of the object surface can be chosen in some proportion to the number of pixels this region will cover in the rendered image. In an oct-tree structure, the cell dimensions and the current viewing transformation are sufficient to give us a usable value for this cue.

- Silhouette Containment:

In regions that include the silhouette, we must choose a larger number of samples. A region contains a silhouette if some of the points in the region have normals facing the eye point and other points have normals facing away. Once again we select a subset of points in the region. Normal computation is done again using the eigen value analysis described above. The correct orientation of the normal is computed by using the representative normal for that region. This is simply done by ensuring the normal orientation is such that its dot product with the representative normal is positive. Using the chosen subset of points we obtain a probabilistic estimate for whether the region contains a silhouette or not. If all normals are either facing towards the eye point or are all facing away from the eye point, we say that this node does not contain any silhouette.

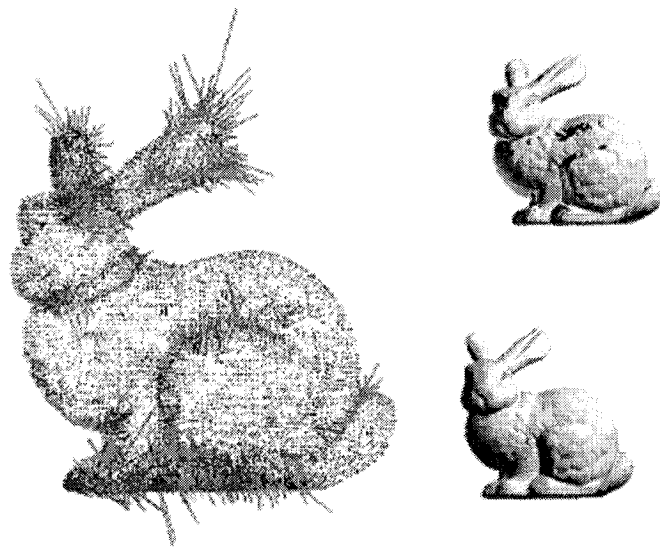


Figure 3-2: Silhouette Containment

Figure 3-2) (left) Un-oriented normals (red facing inwards). B) (right) Results after normal orientation. (top images show picture shaded using normals as computed using eigen value analysis). Bottom shows images rendered after orientation correction.

3.3 Rendering Process

3.3.1 Construction of oct-tree:

Given a group of points S , the first step we do is to organize the set into an oct-tree. The oct-tree construction process is well known and straightforward. The bounding box for the entire set S is first computed as the root and then subdivision proceeds until the following criteria are met:

- The number of points in a node is less than a pre-set number, say, `max_point_budget`.
- The points in that node satisfy a given flatness criterion.
- With each leaf node of this oct-tree we associate the following information:
- Pointers to the set of points belonging to this node.
- Count of total number of points in this node.
- A marker indicating presence/absence of an edge; this is done by carrying out the edge containment computation described earlier.
- A correctly oriented normal; the method for computing the correctly oriented normal for the region of the object's surface covered in this node is described below.

Figure 3-2 shows an oct-tree visualization using cubes for a point-sampled surface

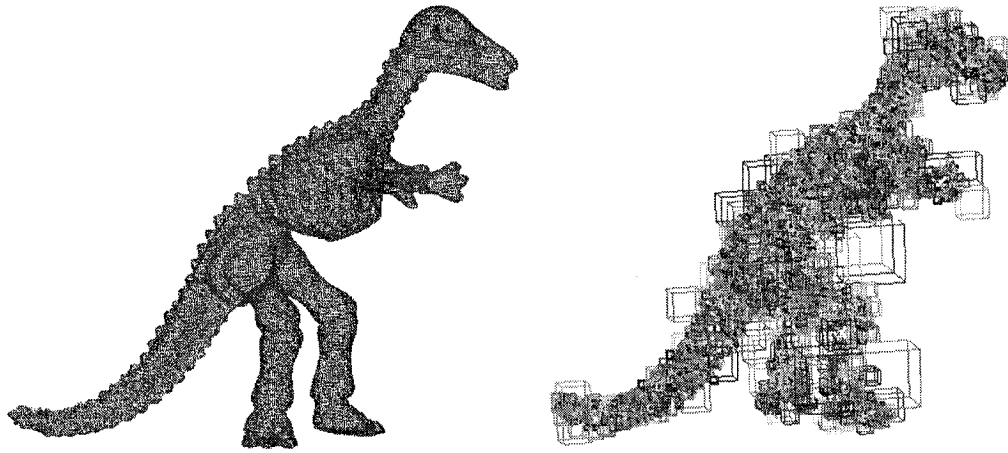


Figure 3-3: Oct-tree (right) for point sampled surface (left).

3.3.2 Rendering by stochastic sampling:

During rendering every leaf node is traversed and then the number of samples to be selected for rendering from the region represented by the leaf node is determined based on the values for the different visual cues. The basic structure of this algorithm is given below. We first describe the algorithm without giving values for a number of the factors used in the algorithm. For example, we have just said that if the silhouette is present then suitably increase the sample size. However, later in Table 3.1 we give the values that we have used in our experiments for the different factors that appear in this algorithm.

```
void render_leaf_node(node) {
//find number of points needed to render
int Ns = find_Ns(this_node);

for (i=0;i<Ns;i++) {
//select a point (random) in this node and find its neighborhood
```

```

current_point = node_points[random(Ns)];
points[] = find_neighbouring_points();

//find eigenvalues and eigenvectors.
//eigen[0] is smallest eigenvalue. Eigenvectors contains corresponding
eigenvectors.
eigens[3], eigenVectors[3] = eigen_computations(points);
normal = eigenVector[0].normalize();

//check if normal properly oriented.
//otherwise reverse direction
if( dot_product(normal, rep_normal ) < 1 ) {
normal = -normal;
}

// find and render an ellipse in tangent plane based on principle curvatures [14]
curvatures[2] = compute_principle_curvatures(points);
draw_ellipse(curvatures);
}
}

//Ns is number of points needed for rendering leaf node
int find_Ns(leaf_node) {
//estimate initial size
int Ns = projected_screen_area() * pixel_density;

// do eigen computations for this node.
// eigenvalues are stored in increasing magnitude.
eigens[3] = perform_eigen_analysis(leaf_node);
eigens = normalize(eigens);

```



```

//adjust Ns based on min_eigenvalue for flatness.
// K=constant to adjust rendering speed vs quality
double flatness = K* eigen[0];
Ns = Ns* (flatness);

//check for edge and update Ns
if(eigen[0] = eigen[1] && eigen[2] = 2* eigen[0]) {
//edge present;
Ns = Ns * edge_factor();
}

//check for silhouette and update Ns
double silhouette_factor = perform_silhouette_analysis();
Ns = Ns*silhouette_factor;
}

Return Ns;
}

void find_rep_normals(leaf_nodes[]) {
//first find a cell for which we always know the orientation.
// our first cell is the one with max z coordinate
start_node = find_leaf_max_z();

normal_dir = {0,0,1};
// call recursive function to correct nodes starting with this //node
correct_neighboring_nodes(startnode, normal_dir);
}

//recursive function to correct orientations.

```

```

void correct_neighboring_nodes(node, prev_rep_normal){
// if no more neighbors return.
if(node == null)
return;

//see if rep_normal correctly oriented, otherwise reverse //direction
rep_normal = unoriented_normal(node);
If(dot_product(rep_normal, prev_rep_normal) < 1) {
rep_normal = -rep_normal;
}
//recursively correct neighbors of this node
correct_neighboring_nodes(, rep_normal);
}

```

3.3.3 Implementation Heuristics

The implementation of the preprocessing task and the rendering algorithm as described above is rather straightforward. There are a number of factors that have to be heuristically determined. These include the various ratios and factors mentioned earlier that decide on whether a node is flat or curved, whether a point can be classified as edge or not, the factor for the nominal number of points to be rendered, etc. In our present implementation we have experimented with different values. Table 1 contains the values, which seem to give us good results in all of the cases we have experimented with.

- Efficiency Improvements

Our rendering process depends very heavily on computing eigen values and eigen vectors of a point set. We have come up with an efficient method to carry out these eigen value computations. There are 2 key observations:

In our case we need to perform eigen value analysis on 3x3 matrix only. Since cubic equations can be solved explicitly, this makes this calculation linear in time with respect to the number of points.

The other key observation is that this 3x3 matrix is symmetric in nature. Hence the complexity of cubic equation is less than the full general form of cubic equation. Pauly et al have used the Newton-Rapson method to solve this cubic equation in Pauly et al. [35]. They have said that it needs on the average less than 3 or 4 iterations. In our case, we have taken advantage of the special structure of cubic equation, which guarantees us that roots are always real.

Property	Criterion/formulae
Flatness	$f = \lambda_0 / (\lambda_0 + \lambda_1 + \lambda_2) \leq 0.005$
Edge point classification	$\lambda_0 \approx \lambda_1, \lambda_2 \approx 2\lambda_0$
Nominal number of samples in a flat region – Ns	Let Np be the point count of the points in that leaf node; Let W be the estimate of the number of pixels covered by this leaf node in screen space taking into account current viewing parameters. Then $Ns = \min(W/4, Np)$.
Flatness adjustment factor	$Ns = \min((1 + f/0.005) * Ns, Np)$ At most we will choose double the number of nominal points.
Silhouette/Edge containment factor	If silhouette is present in this leaf node, then $Ns = \min(4.0 * Ns, Np)$
Splat dimensions	If the ratios of the two principal curvatures is 1.0, then a circle is in the tangent plane is chosen with radius R such that R maps to $\text{ceil}[\sqrt{W/Ns}]$ number of pixels. If the ratio is less than 1.0, then the minor axis size is suitably scaled. The major axis is aligned with the direction of maximum principal curvature.

Table 3-1: Rendering algorithm parameters.

1) Most of the time we are only interested in finding whether to subdivide the cell further depending on whether it is nearly flat or not and then find the corresponding eigen vector that is used as the normal direction. This can be done very efficiently as follows:

- a) We know that sum of reciprocals of roots of cubic equation = (sum of products taking 2 roots at a time) / product of all 3 roots*
- b) Since smallest root must have a very small value for the flat regions, its reciprocal is very large. Hence reciprocal of smallest root approximately equals the sum of reciprocals of roots*
- c) This allows us to get the smallest root without solving the cubic equation but by evaluating an expression in terms of coefficients of cubic equation.*
- d) We verify the correctness of this as the root by substituting it back in the cubic equation. If this is not a root, then it also implies that the region under consideration is not flat.*

- **Results**

We carried out some experiments to check the performance of our method. Fig.3.5 shows a model rendered at different image sizes. The larger images have been cropped from the right to fit into the column. In Table 3.2 we give the image size and the actual number of samples selected and rendered. The variation in the number of samples required for each case is as expected.

We see that number of sample points rendered does not increase linearly with image size in pixels. This is due to the fact that visual cues (such as flatness criteria) help us in reducing the number of points needed to render.

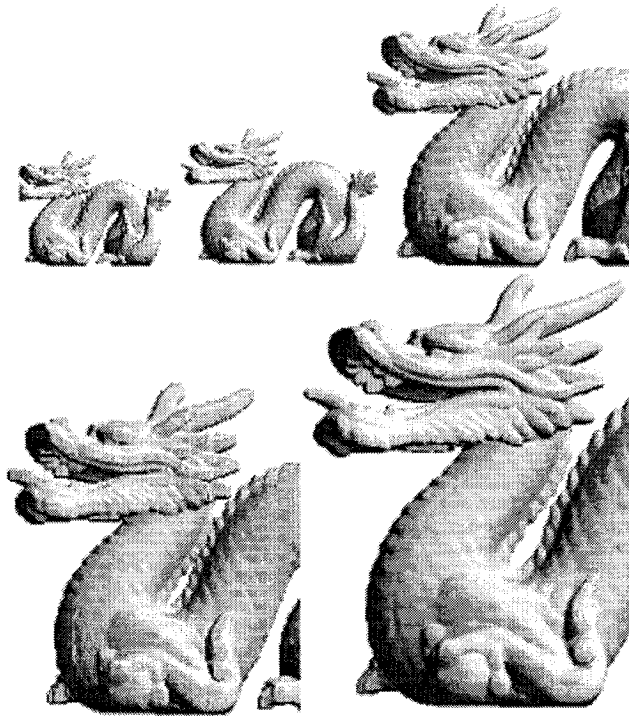


Figure 3-4: Rendering at different image sizes.

Image # in Fig. 3-4.	Image size in pixels	# of Sample points rendered	performance improvement factor
(i)	96 x 96	10247	55
(ii)	160 x 160	21882	25
(iii)	250 x 250	44235	12.5
(iv)	380 x 380	76789	7.3
(v)	500 x 500	109412	5.2

Table 3-2: Number of samples varying with image size.

3.4 Some Remarks on Stochastic Rendering

We have presented a novel way to render a large point cloud model using a non-deterministic approach. The method works by exploiting local coherence with a PCA analysis. It is fairly general in that it can handle other local attributes such as

color within this framework. The oct-tree representation efficiently approximates the original geometry with hierarchy. The real application of this is in situations where we need fast rendering of large models without the need for extremely accurate images.

We have observed a few problems. The treatment of the PCA subdivision is done with no connectivity information and this can sometimes cause two non-adjacent surface areas to be merged into one PCA node. Similarly, in case of an animated view of the object, the fact that different sample points are displayed in a view dependent manner could sometimes result in temporal discontinuities in the displayed image. These are major research problems and we have not addressed them as part of our research.

4 Edge Recovery from 3D Point Cloud Models

While surface fitting techniques applied to point cloud models can give us a mathematical representation of the best surface approximating the 3D object's surface, these techniques are applicable only to 3D objects or parts of 3D objects that can be considered as a continuous surface. Similarly, techniques that fit a triangle mesh to the point cloud model give back the surface as a connected mesh of small triangles. This too is more suited for natural or sculptured objects. For most engineering objects however, these techniques result in a far too low level and verbose representation. More importantly they miss out on the representation of key features such as vertices, edges, loops and faces that make up the object. The B-Rep technique of representing 3D objects essentially describes a 3D object using these features. In our research we have attempted to devise new algorithms for detecting such features. In this chapter we shall describe our technique for detecting vertices and edges. And in the next chapter we describe our techniques for detecting edge loops and for segmenting the point set into different faces bounded by these edge loops.

Edge recovery techniques for 3D point cloud models is done in two major stages. The first stage involves classification of points in the point set. And in the second stage we collect all the points belonging to an edge. This is done first by traversing and collecting points that belong to the same edge and then further refining the points collected for an edge.

The entire point set is very large. For increased efficiency, we need to restrict our traversal to a subset of points that are most likely to belong to the edge being traversed. It would be best if we can classify points as definitely not belonging to any

edge. This is done in the first stage of point classification. As in Gumhold et al. [14], we classify the complete point set into four types according to the geometric properties of the point with respect to its neighbourhood. These categories are: FaceInterior, SurfaceBorder, SurfaceCrease and SurfaceCorner. The FaceInterior points are interior to some face of the object's surface. The SurfaceCorner, SurfaceBorder and SurfaceCrease points are the subset of points that need to be traversed for belonging to an edge of the 3D object. SurfaceCorner points are points of the 3D object, where multiple faces and multiple edges meet.

Every edge is bounded by two SurfaceCorner points. Once the point classification stage is completed, we recursively traverse the SurfaceCrease points starting from one SurfaceCorner point until we reach another SurfaceCorner point. This traversal is along the potential direction determined for that edge of the 3D object. Once we have collected and identified the subset of points that make up each edge, we analyze all edges in order to use them subsequently in the face recovery process. The edge analysis task includes erasing of duplicate edges, splitting of edges that intersect other edges, and merging the two or more edges if their edge end points are very close.

4.1 Classification of Points

For classification of points we need to arrive at exclusive criteria that can be evaluated. Such evaluations depend on different parameters computed on the point set. The parameters to be computed and their use in evaluation criteria vary according to the type of object surface sampled and also the sampling attributes. For a model with sharp corners and edges, it is easy to define the parameter values to classify the points because the parameter values are not continuous, but for a smooth model, it is hard to

determine the best parameter values for classification. Furthermore, there are many other factors that affect the parameters, for example, noise, sampling density (dense/sparse) and the variation in sampling density, etc. Hence, there is no unique method to define parameter values that can work for all models. Some other researchers just ask the user to define the parameter values to be fed in as user input interactively. However in our work we have proposed a statistical method. The statistical method has two stages in the classification process: initial classification process and refinement process. In the initial classification process, we define the classification only for a small sample of the complete point set. Then we analyze and determine the parameter values based on this small sample. In this way, we can cover all true SurfaceCrease/SurfaceCorner points, and it may also cover some FaceInterior points that are located around the true SurfaceCrease/SurfaceCorner points, but we will reduce these points during the refinement process. The refinement process is to identify the classification again based on the local neighbourhood of the points.

4.1.1 Initial Classification Process

Our classification process uses the statistical values computed by an eigen value analysis technique applied to a point by considering all the points within a small neighbourhood. These statistical properties are then used to classify the point into one of the four categories: FaceInterior, SurfaceBorder, SurfaceCrease and SurfaceCorner.

Let us denote $\lambda_0, \lambda_1, \lambda_2$ ($\lambda_0 \leq \lambda_1 \leq \lambda_2$) as the three eigen values.

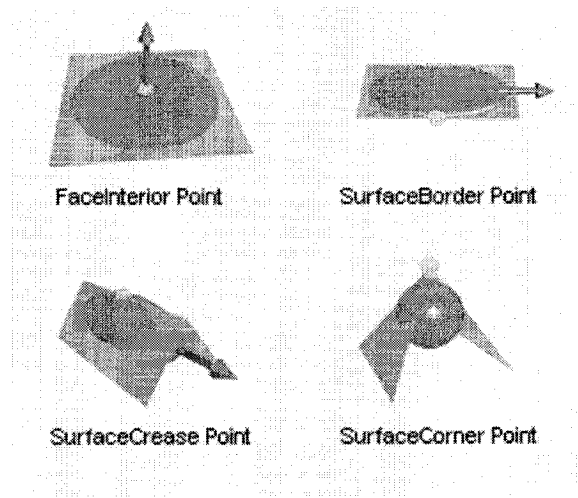


Figure 4-1: Initial Classification

4.1.1.1 FaceInterior Points

FaceInterior points are the points that are interior to a face and within a local neighbourhood are part of a locally flat plane, i.e, they are surrounded by other points on the same plane. Here we assume that the properties associated with the mean of all neighbor points of one point are approximately the same as the properties of the point itself. For a point to be classified as FaceInterior, the eigenvalue of first PC and second PC are almost same, and the eigenvalue of third PC is almost zero comparing to the eigenvalue of other two PCs because all neighbor points are on the same plane.

$$\lambda_0 / (\lambda_0 + \lambda_1 + \lambda_2) \approx 0$$

4.1.1.2 SurfaceBorder Points

SurfaceBorder points are also the points which are on locally flat plane, but they are not inside the face, but are on the edge of the face, with no points on the other points outside the face. For a point to be classified as a SurfaceBorder point, all neighbor points must be on one side of the border of the face, or along the border edge. Hence, the first PC vector has to be along the border of the face, and the second PC

vector has to be orthogonal to the first PC vector but still on the locally flat plane, and the third PC vector is the normal vector to the flat plane. The length of the first PC vector is approximately double of the length of the second PC vector, and the length of the third PC vector is close to zero in comparison to the other two vectors.

$$\lambda_0 / (\lambda_0 + \lambda_1 + \lambda_2) \approx 0$$

and $\lambda_1 * 2 \approx \lambda_2$

For the a closed surface model, there is no border in theory, but some points could be erroneously identified as SurfaceBorder points. This could be due to noisy data which causes one side to be dense, and another side sparse.

4.1.1.3 SurfaceCrease Points

SurfaceCrease points are shared between two faces. The neighbor points of SurfaceCrease points are two groups; each group belongs to a face of the object. Hence they are classified as FaeInterior points. Hence, the centre point of all neighbor points should move down to the center of two planes. For classifying a point as SurfaceCrase, the vector of first PC points in the direction of the SurfaceCrease line. The eigenvalue of the second PC reduces a little, and the eigenvalue of the third PC increases in comparison to the FaceInterior points.

$$\lambda_0 \approx \lambda_1$$

and $\lambda_0 + \lambda_1 \approx \lambda_2$

Actually, the SurfaceCrease type is very hard to determine. Firstly, λ_0 is not close to zero because the neighborhood points have certain distance in the direction of the normal. We also cannot define a range for λ_0 of the SurfaceCrease type because it is really very closely related to the intersecting angle of the two surfaces of the

adjacent faces. As the angle is decreases from 180 degree to 0 degree, λ_0 is increasing.

If it is a very small angle, λ_0 can increase to become λ_1 , in this case it may get erroneously classified as a SurfaceBorder point. Secondly, λ_1 will increase when the angle is increasing from 0 degree to 180 degree. It may be close to λ_2 when the angle is increasing to close to 180 degree. Hence the criterion above is applicable only in the regular case. It needs to be considered in conjunction with the criteria for SurfaceBorder and SurfaceCorner point classification.

4.1.1.4 SurfaceCorner Points

SurfaceCorner points are on the intersection of more than two faces, and the principal components are more complicated. The neighbor points of SurfaceCrease points may consist of many groups; each group belonging to a different face. Locally these will be on different planes. Hence, the centre point of all neighbor points should move down to the center of all planes. For classifying a point as SurfaceCorner, all eigen values of the three PCs must be similar.

$$(\lambda_2 - \lambda_0) / (\lambda_0 + \lambda_1 + \lambda_2) \approx 0$$

4.1.2 Classification Distribution:

In the process of the initial classification, the program always needs the predefined parameter thresholds for the eigen values to classify the points. People setup the threshold directly either by experience or interactively through user input. We have not chosen this method. Instead we estimate the values according to the sampled data model based on the distributions of all points during the processing.

4.1.2.1 Planar Estimation Distribution

Planar Curvature Estimation Value is used to identify the SurfaceCrease/SurfaceCorner points from FaceInterior/SurfaceBorder points. The planar estimation value for FaceInterior/SurfaceBorder points should be very small because the neighborhood points of a FaceInterior or SurfaceBorder point are approximately on a plane. On the contrary, the planar curvature estimation value for a SurfaceCrease or SurfaceCorner point should have certain value because the neighborhood points of a SurfaceCrease or SurfaceCorner point are not apparently on the same plane.

$$\text{Planar Estimation} = \lambda_0 / (\lambda_0 + \lambda_1 + \lambda_2)$$

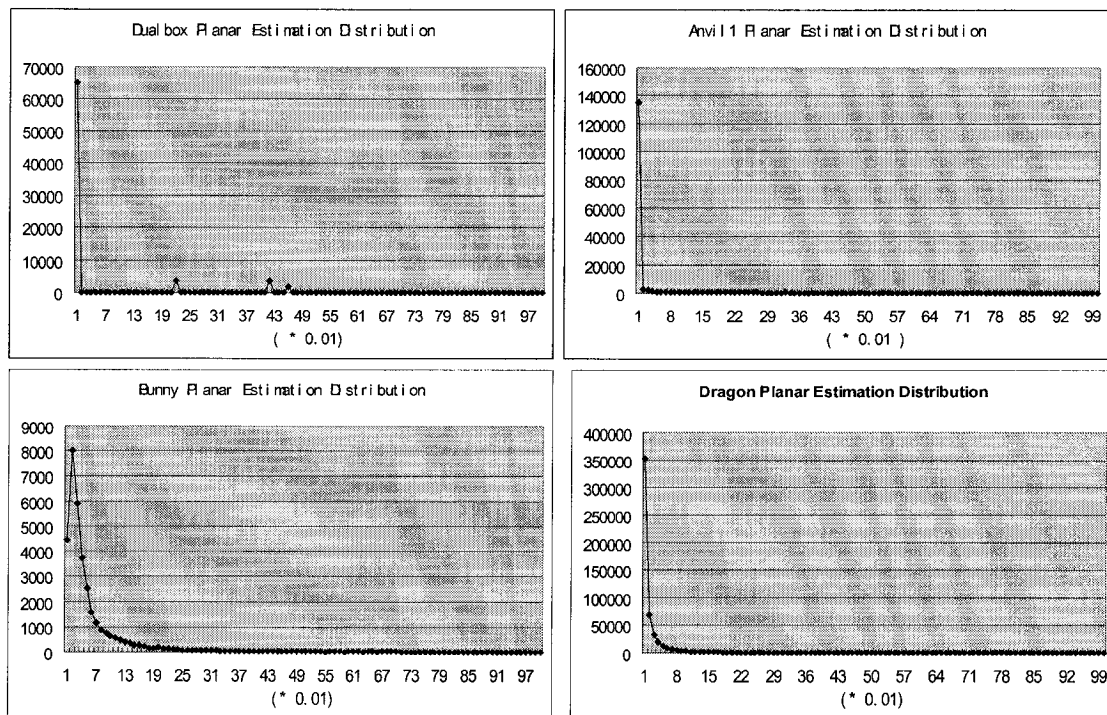


Figure 4-2: Planar Estimation Distributions.

Figure 4-2 shows the planar estimation distributions for a) Dualbox, b) Anvil, c) Bunny and d) Dragon. The planar estimations of most models vary continuously, so we

prefer to use the distribution of the planar estimation to identify the SurfaceCrease/SurfaceCorner points from FaceInterior/SurfaceBorder instead of one predefined value.

4.1.2.2 SurfaceCrease Estimation Distribution

SurfaceCrease Estimation Value is used to distinguish the SurfaceCrease points from the SurfaceCorner points. The SurfaceCrease points should have significant SurfaceCrease estimation value, but the SurfaceCrease estimation value for a SurfaceCorner point should be close to zero.

$$\text{SurfaceCrease Estimation} = 1 - \lambda_1 / \lambda_2$$

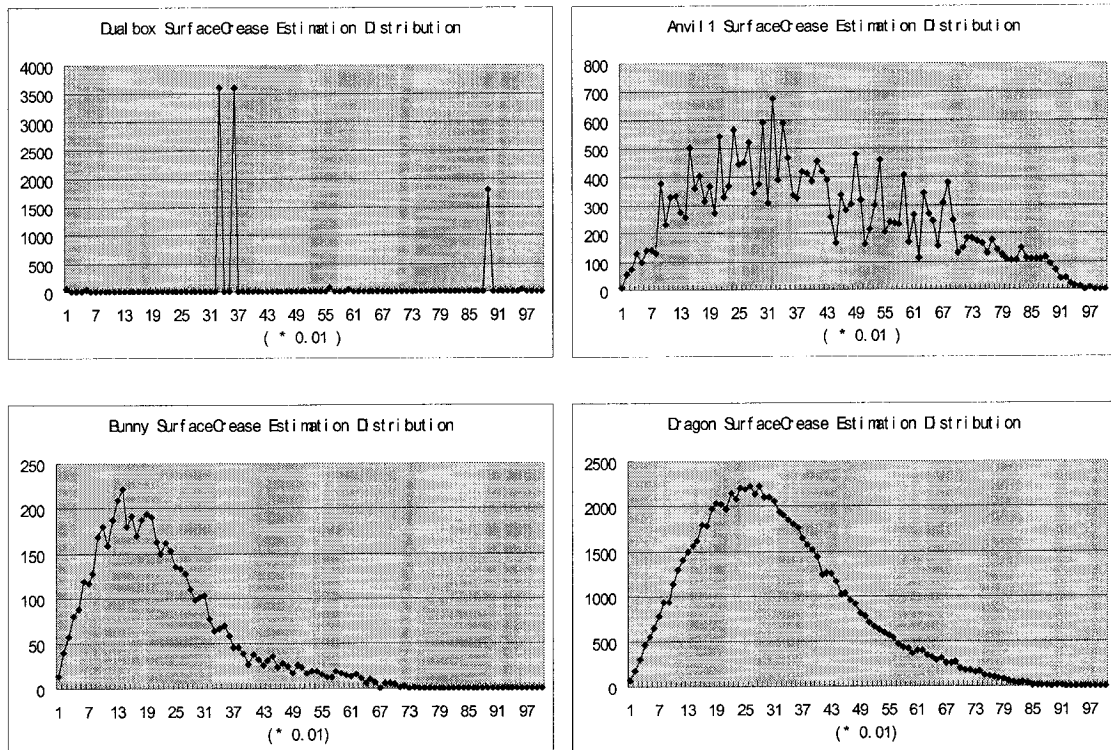


Figure 4-3: SurfaceCrease Estimation Distributions.

Figure 4-3 shows the SurfaceCrease estimation distributions for a) Dualbox, b) Anvil, c) Bunny and d) Dragon. The SurfaceCrease estimations of most models vary

continuously, so we prefer to use the distribution of the SurfaceCrease estimation to identify the SurfaceCrease points from SurfaceCorner points.

4.1.2.3 SurfaceBorder Estimation Distribution

SurfaceBorder Estimation Value is used to distinguish the SurfaceBorder points from the FaceInterior points. The SurfaceBorder points should have significant SurfaceBorder estimation value, but the SurfaceBorder estimation value for a FaceInterior point should close to zero.

$$\text{SurfaceBorder Estimation} = \lambda_1 / \lambda_2$$

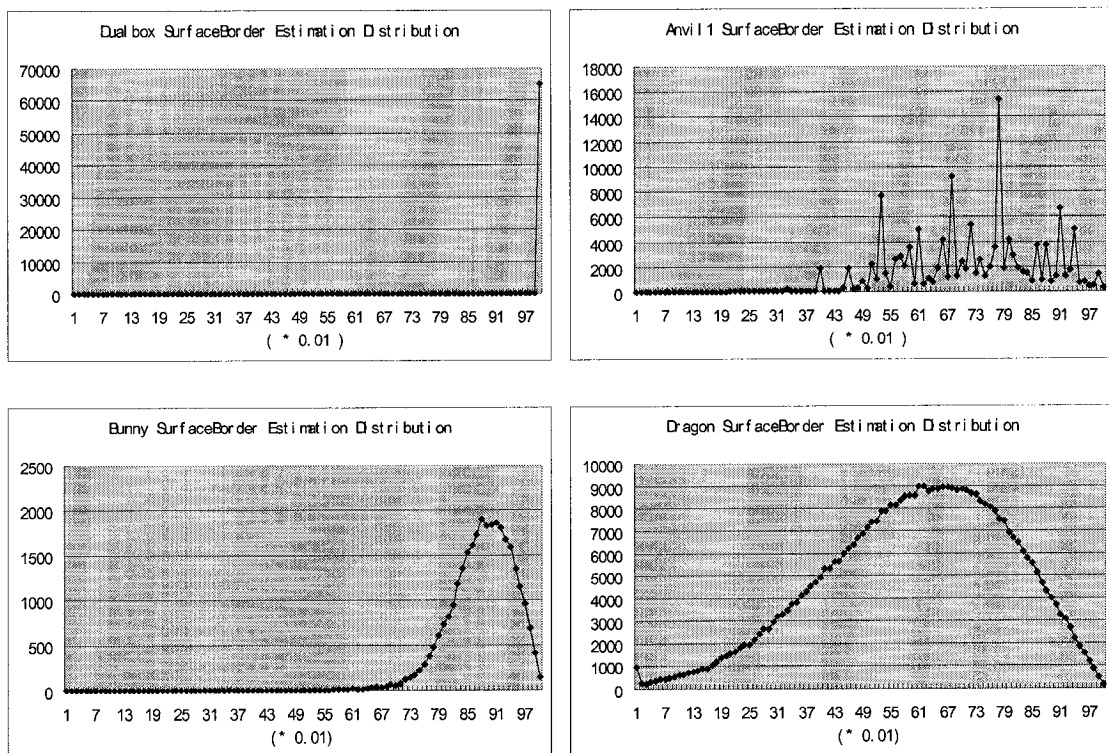


Figure 4-4: SurfaceBorder Estimation Distributions.

Figure 4-4 shows the SurfaceBorder estimation distributions for a) Dualbox, b) Anvil, c) Bunny and d) Dragon. The SurfaceBorder estimations of most natural models varies continuously, but the SurfaceBorder estimations of the engineering models like

dualbox and anvil only have several values. It is very hard to find out the proper SurfaceBorder estimation value to identify SurfaceBorder points for different models, so we prefer to use the distribution of the SurfaceBorder estimation to identify the SurfaceBorder points.

4.1.3 Refining Points Classification:

The initial classification is to determine the thresholds for eigen values for use in the classification process. For each class of the points, we give global criteria for all points of the sample model. The global criteria are based on the eigen values of the neighborhood points. However, the eigen values of the close points are also very similar because the close points share most neighborhood points and they have very similar local surface properties. Moreover, the different parts of the sample surface have different variation. Some parts of the surface vary with sharp discontinuities, but some may vary very smoothly. The classification of points belonging to objects with sharp edges is easy, but the points of smooth surfaces are hard to be classified. For proper classification of points of natural or sculptured objects, it will be necessary to extend the criteria for the SurfaceCrease/ SurfaceCorner/ SurfaceBorder. This is beyond the scope of our present research.

During the process of the initial classification, there are many FaceInterior points that are identified to other types because of two reasons described above. Hence, we add a refinement process after initial classification in order to correct the point classification. The refining logic is based on the neighborhood points, and the refining is done on a case by case.

There are three major cases:

- There are many points classified as SurfaceCrease points besides the actual edge points on both sides.
- Some FaceInterior points are identified to be SurfaceBorder points.
- Some SurfaceCrease points on the edge are identified to be SurfaceCorner points.

4.1.3.1 SurfaceCrease Points Refinement Process

The initial classification finds not only exactly SurfaceCrease points along the edge, but also the points besides the true SurfaceCrease points on both sides because all of them have very similar local surface properties. In order to extract the real edge points from the neighborhood SurfaceCrease points, the curvature estimation is a good property to deal with it.

We use the value similar to the curvature estimating method described by Gumhold et al. [14]. The curvature K_i is:

$$K_i = \frac{2d_i}{u^2}$$

u is the average distance between the point to other neighbor points, and d is the distance on the normal direction from the point to the local plane which passes the center point of the neighborhood points.

$$d = v_i v_c \cdot N_i.$$

If the point is close to the spine; the curvature estimation value K_i becomes larger. If we take a plane that is orthogonal to the edge direction and it cuts the neighborhood points like below on the right side, the point on the spine line will have the biggest curvature value.

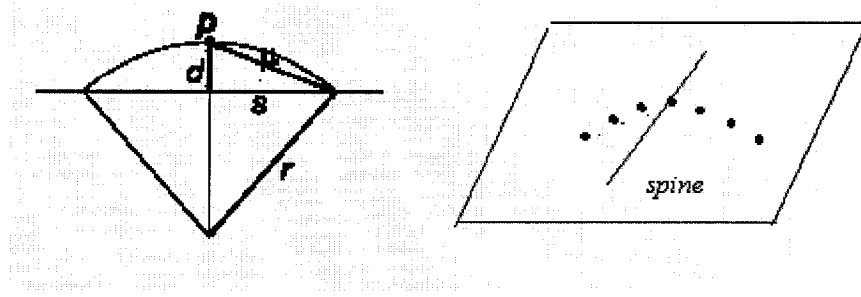


Figure 4-5: Curvature Estimation

4.1.3.2 SurfaceBorder Points Refinement Process

The SurfaceBorder points and the FaceInterior points are easy to be mixed up. They both have eigen values with very short length along the normal direction comparing to other two PC vectors. The only difference is the variance between 1st and 2nd PC vectors. As a SurfaceBorder point, the length of 1st PC vector may be close to double of the 2nd PC vector, and as a FaceInterior point, the length of these two PC vectors are approximately same. In fact, the variance between the length of these two types points changes gradually, so we can not simply distinguish SurfaceBorder and FaceInterior points only by the length ratio of the 1st PC vector with respect to the 2nd PC vector.

Gumhold et al. [14] introduced a very good property of the SurfaceBorder and FaceInterior points. After calculating the eigen matrix for a SurfaceBorder point v_i , we can get a plane (e_0, c_i) . The e_0 is the normal, and the c_i is the mean of the neighborhood points. We can project all neighbor points on this plane, and then connect all projected neighbor points to the projected SurfaceBorder point on the plane; each pair of the adjacent neighbor points on either clockwise or anti-clockwise direction have an angle. The maximum open angle is a very good property for the SurfaceBorder points.

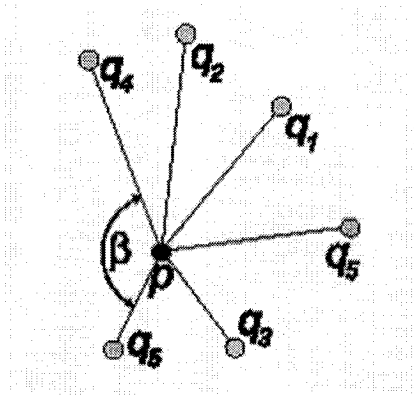


Figure 4-6: Maximum Open Angle

For the FaceInterior points, the neighbor points should be distributed around them randomly; for the SurfaceBorder points, the neighbor points should be located on one side, less or none are located on another side, so the maximum open angle for a SurfaceBorder point is much bigger than a FaceInterior point. We predefine a value β for the maximum open angle; and a SurfaceBorder point will change to the FaceInterior point if it's maximum open angle is less than the predefined value β .

4.1.3.3 SurfaceCorner Points Refinement Process

The logic to identify SurfaceCorner points is based on the ratio of the length of 1st PC vector to the length of 2nd PC vector. This means comparing the 2nd PC vector and the 1st PC vector. If the difference between these two PC vectors is large; it is more likely to be a SurfaceCrease point. If the difference between these two PC vectors is less; it is more likely to be a SurfaceCorner point. Hence, some SurfaceCrease points on the edge are identified to be SurfaceCorner points for two reasons. One is noise data. The noise data will increase the variation on 2nd PC vector if noise points are located along 2nd PC vector. Another reason is the criteria for identifying the SurfaceCorner

points is weak; and it causes more points to be classified as SurfaceCorner, than the actual case.

The SurfaceCorner point on the edge has following property:

- All close points have the same direction as the 1st PC vector; which is also the edge direction.

We change the point from the SurfaceCorner point to the SurfaceCrease point if the SurfaceCorner point has above property.

4.1.4 Neighborhood size:

The neighbor points are the sample points that are close to the point being classified. The neighborhood size is the number of closest sample points' contributing to the local surface variation. Increasing the neighborhood size means the contribution of any one sample point to the local variation is less and the variation of the local surface becomes smoother. Decreasing the neighborhood size means the contribution of any one sample point to the local variation is more. If there are some noise data in the neighborhood, they cause more damage to the local surface estimation for a smaller neighborhood size than for a bigger neighborhood size.

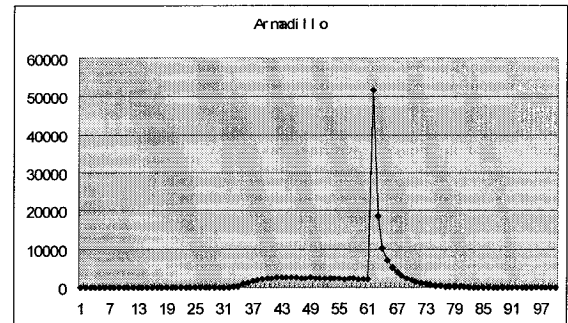
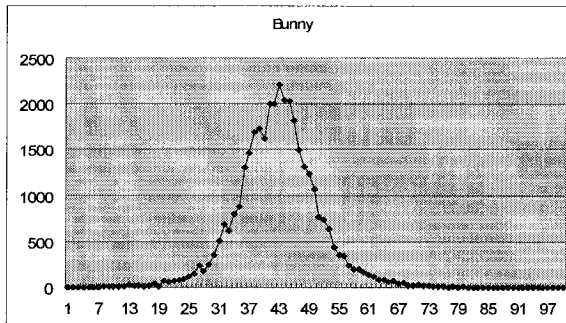
A good neighborhood size is very hard to be estimated. If the data is dense, the neighborhood size should be bigger and vice versa. The data is noisier, the neighborhood size should be bigger. Taking more neighbor points can reduce the noise data contribution to the local surface variation.

Most of algorithms published in the literature for feature detection in point clouds use the Delaunay Triangulation method to get the relationship between the neighbor points, and then they use the k-nearest neighbor method to estimate the local

neighbourhood. However, 3D Delaunay Triangulation for the surface of a 3D point cloud model is a complex and expensive process, particularly for a very large point set. We have chosen to use a distance function to choose the neighbourhood.

- The average distance distribution:

For the sampled data model, the average distance of the adjacent points is always unknown. In some papers, they get the average distance as user input. In our approach, we estimate the average distance by calculating the minimum distance distribution. We calculate the minimum distance of the neighbor point for each sample point first, and then split the minimum distance range into hundred segments, and distribute all sample points into the hundred segments. Two cases of the noise data affect the average distance estimation, one is duplication sample which is caused by the process of registering the multiple range data sets; another case is that some noise data is far away from the surface, so we skip the minimum 1/10 sample points and the maximum 1/10 sample points, and only calculate the average of distance from the middle 80 percent points.



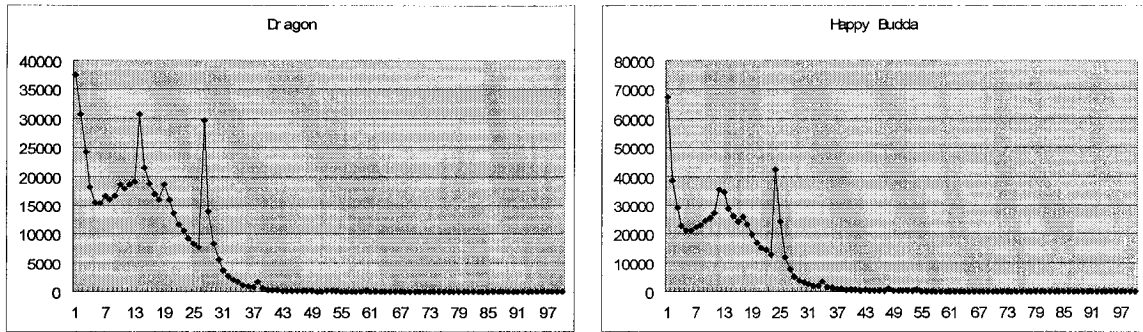


Figure 4-7: The average distance distributions

4.2 Edge Traverse

4.2.1 SurfaceCorner Point Adjustment:

There may be more than one SurfaceCorner points within a small neighbourhood after point classification, whereas there should be only one. Since edges are between two SurfaceCorner points, incorrect SurfaceCorner points will in turn generate very short edges, and cause the edge analysis to go astray or involve a very complex process. Hence, the first step is to adjust the SurfaceCorner points classification and rectify this situation to the extent possible.

For the SurfaceCorner point, V_{corn} is defined as below:

$$V_{\text{corn}} = (\lambda_2 - \lambda_0) / (\lambda_0 + \lambda_1 + \lambda_2)$$

The values of V_{corn} are compared if there are more than one SurfaceCorner points in the small range. The minimum V_{corn} of the SurfaceCorner point is the best SurfaceCorner point because its V_{corn} is more close to zero. Other SurfaceCorner points of the neighbor points are changed to the SurfaceCrease points.

By recursively selecting the best SurfaceCorner point, the process finally retains only one SurfaceCorner point in any small neighborhood.

4.2.2 Direction of Edge Points

According to the points classification, the eigenvector of the first PC is the direction of the edge at that SurfaceCrease point. The vector between two adjacent edge points should be fitting the direction of the edge at the SurfaceCrease points. Not only does the vector from current point to next point fit the direction of SurfaceCrease points, but also it should fit the direction of edge tangent vector of next point if next point is SurfaceCrease or SurfaceBorder point.

The traversing algorithm only traverses the SurfaceCrease, SurfaceBorder and SurfaceCorner points. If current point is SurfaceCrease or SurfaceBorder type, the first PC vector is the edge direction. If the current point is SurfaceCorner point, the edge direction cannot use the first PC vector; it has to be the vector from the previous point to the current point. For each neighborhood point, three conditions have to be satisfied to be next point on the edge.

- It should be SurfaceCrease or SurfaceBorder or SurfaceCorner point.
- The angle between the edge direction and the vector from current point to neighbor point should be \leq predefined angle β_1 .
- The angle between the vector from current point to neighbor point and the edge direction of neighbor point \leq predefined angle β_2 if the neighbor point is SurfaceCrease or SurfaceCorner point.

The algorithm is described below:

edge_traverse(S:current traversing point)

- *Get the previous traversing point P.*
- *Get the vector from P to S as V_{PS} .*
- *If S is SurfaceCrease point or S is SurfaceCorner point*

- Set Edge Traversing Direction(ETD) = the first PC vector of S
- If ETD dotproduct $V_{PS} < 0$ then
- Set $ETD = ETD * (-1)$ /* Change direction */
- Else if S is SurfaceCorner point
- Set Traversing Direction(ETD) = V_{PS}
- Else
- Return with error message because S can not be FaceInterior point.
- End if
- Clear the next traversing list TL.
- Get the neighborhood points set NGP(S).
- For each point N of the NGP(S)
- If V is not SurfaceCrease/SurfaceBorder/SurfaceCorner point then
- Skip. /* next neighbor point loop */
- End if
- Set $A_1(N)$ = the angle between V_{PS} and V_{SN}
- If $A_1(N) > \beta_1$ then
- Skip. /* next neighbor point loop. */
- End if
- If S is SurfaceCrease or SurfaceBorder point then
- Set C_N = the first PC vector of N as C_N
- Set $A_2(N)$ = the angle between V_{SN} and C_N
- If $A_2(N) > \beta_2$ then
- Skip. /* next neighbor point loop */
- End if
- Append N, $A_1(N)$ into TL.
- End Loop
- Order TL by the angle in descent
- For each point T in TL
- myPush(T) /* The stack to keep all traversing points */
- Set ret = edge_traversing(T)
- myPop()
- If ret = Succeeded then
- Return Succeeded
- End if

- *End Loop*
- *For each point T in TL*
- *If T is SurfaceCorner point then*
- *Append_edge()* */* Make a traversing edge from the stack */*
- *Return Succeeded*
- *End if*
- *End Loop*
- *Return Not_Found* */* No edge is found */*
- */* end of function edge_traversing() */*

4.3 Edge Analysis:

4.3.1 Edges Analysis Objective:

In order to get the geometry information from these edges, several rules are defined as given below:

I. No vacancy along the edge:

Any two adjacent points of an edge must be close to each other, which means there are not any other points approximately lying on the middle between two adjacent edge points.

II. No edges intersect or cross each other.

The edge is used to split the surface into different faces later. It will cross more than one surface if it intersects other edges.

III. No parallel edges.

There must not be two edges that are between the same start and end points, and all points of one edge are very close to second edge.

IV. No short edges loop.

V. Each edge has to connect to other edges on both sides.

All edges have to be on one or more edge loops; otherwise, it is useless for face recovery.

VI. Any adjacent edges must connect each other at the end point.

It will be done after the SurfaceCorner points merge operation is completed.

4.3.2 No gap along the edge

No gap along the edge means any two adjacent points of an edge must be close to each other, and there are not any other points lying in the middle of two adjacent edge points.

For example, there are two adjacent edge points A and B, and there is one point C which is very close to the segment of A and B.

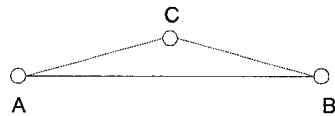


Figure 4-8: Vacancy Example.

Hence, C will be added into the edge between A and B. The old edge segment is AB, and the new edge segment is ACB.

The reason for this function is to make the edge more solid on the surface. The edge of a point cloud model has to be along the points of the surface and it should separate the points of the surface into two faces on either side of the edge. Actually, one line/curve cannot separate the points into two sets in 3D, but the plane can. The edge separates the points of the surface into two sets only because we only consider these points as close to the edge, and we are assuming the points and the edge are on

the same surface and locally on the same plane. Hence, if there are two adjacent points whose distance is big, it may cause the edge to be far from the surface, and it cannot separate the nearby points into two sets.

- How to decide about the third point that lies in between adjacent edge points?

In order for it to be easy for checking intersection or crossing of the segments, we assume the distance of the adjacent edge points should be less than 2 times of the average distance. The point cloud model is like a grid that has the same distance approximately, the third point between two points should be found if the distance of two points is more than 2 grid distance.

For example: There are two adjacent edge points A and B in three cases. The distances of A and B in each of the three cases is not less than 2 times of the average distance, and we can find third point C and insert C into the edge between A and B.

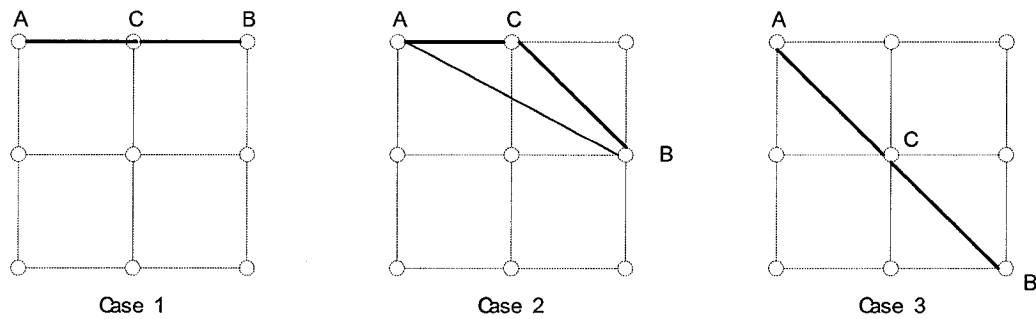


Figure 4-9: Three cases of segment vacancies.

This operation is done when a traversed edge is added into the edge list. It will check the gap along the new edge, and it will fill the points into the new edge if one or more points are found to fit the edge between any adjacent points in the new edge.

4.3.3 No intersecting edge

There are two cases of edge intersection. One is two edges intersect at the same point. Another case is two edges do not intersect at the same point, but they cross each other. Intersecting edges have to be split into three or four edges for both cases.

4.3.3.1 Edges Intersect at the same point

Edge intersecting at the same point is very easy to handle. Each edge is split into two edges from the intersecting point.

4.3.3.2 Edge Crosses Edge

Edges' crossing means that one edge cross another edge but they are not sharing any point. It takes four steps to process it:

- I. Two adjacent points on the same edge should be very close to two adjacent points on the crossing edge because crossing edges have to be on the same surface. There are two edges E_1 and E_2 ; A and B are two adjacent points on the edge E_1 ; C and D are two adjacent points on the edge E_2 . Based on the rule-I of edge analysis in chapter 4.3.1, A and B are less than 2 times of average distance; C and D are very close, less than 2 times of average distance; and these two edges are crossing each other. Hence, the distance from A to one of C and D is less than 2 times of the average distance. B is also in the same situation.
- II. Projecting four points onto the same plane, and check whether the projected lines intersect each other. Three of four points (A, B, and C) decide one plane PL, and the fourth point (D) is projected onto the plane PL as D_1 . Hence, the question to check whether two lines on different planes intersect is changing to the question to check whether

two lines on the same plane intersect. It is easy to check whether the projecting lines (A-B and C-D1) are intersecting each other on the same plane.

- III. In some special case, different selections of three out of four points to decide one plane gives different results. Hence, if the result of step II is no-intersection, another set of three of four points (A, B and D) are selected to decide another plane PL1, and the fourth point(C) is projected onto the plane PL1 as C₁, and checked again as described earlier.
- IV. If these two edges are intersecting each other according to step II or step III , then the best splitting point is selected from these four points.
- V. Each edge is split into two edges at the best splitting point.

4.3.4 No parallel edges

The parallel edges are not exactly parallel. The parallel edges are two cases.

- *Both edges have the same start and end edge points, and at least one edge is a short edge.*
- *Each point on either edge is very close to at least one point on another edge.*

The shorter edge will be removed in both cases. (In our experiments, the short edge is the case when the number of the points is less than 6.)

4.3.5 SurfaceCorner Points Merge Operation

The face recovery process has to be based on detection of edge loops, and any two adjacent edges on the edge loop connect each other on the same edge end point.

We have the rule-VI for this constraint in the edge analysis. The rule-VI will remove

the edge from the edge list if that edge does not connect to other edges at either edge end point.

In the edge point traversal process, the traversing always begins at a SurfaceCorner point and also ends at a SurfaceCorner point, assuming that these points have the connectivity on the sampled 3D object. However, the edge traversing may not be successful if the surface is irregular; specially, the areas that are close to a convex vertex as shown in the figure below. The SurfaceCrease points do not have the clear edge direction if it is in the area around the convex vertex, or the valley of the surface because there are more than two surfaces merging together in that area. The points belonging to different surfaces will affect the first PC vector of the SurfaceCrease point. Hence, the SurfaceCrease points traversed in the SurfaceCorner area may not succeed. We need to create additional edges to connect two close by SurfaceCorner points.

The process always selects the closest pair of the SurfaceCorner points first, and creates a new edge with two close SurfaceCorner points. Creating a new edge with two end points does not create a vector any more, it follows the same rule of identifying out the gaps in the new edge as described earlier in section 4.3.2. The process always takes the middle point between any two adjacent edge points on the new edge. The process is illustrated in the figure below.

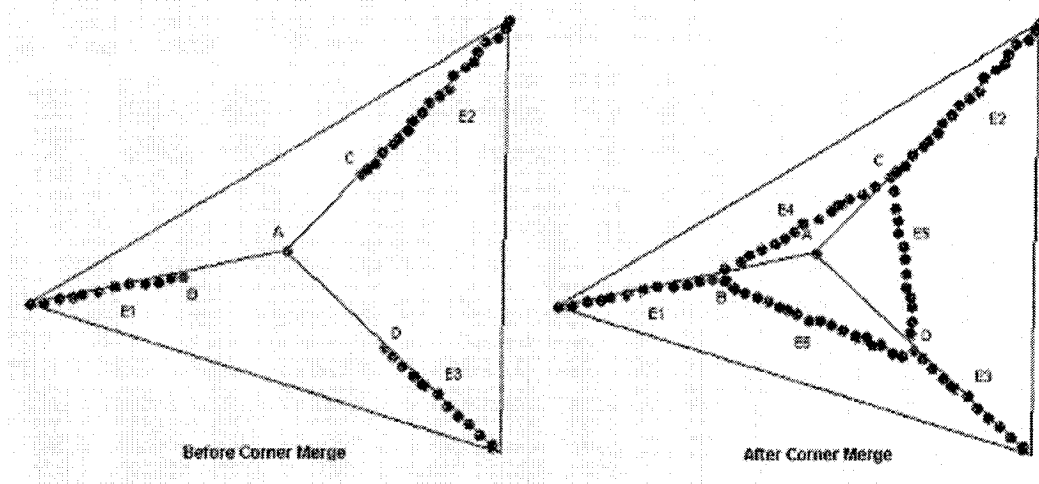


Figure 4-10: The first step of the SurfaceCorner points merge operation.

On the left side, there are three surfaces merging at the point A. AB, AC, AD and their extended line for each is the common edge between two faces. E1, E2 and E3 are three existing edges after the SurfaceCrease points are traversed, but the edge traversing has failed to find the edges AB, AC and AD. B, C and D are identified as SurfaceCorner points; and they are very close to each other. E carries out the SurfaceCorner points merging process. On the right side, is the result of the SurfaceCorner points merging process without considering the short edge loop. E4 is the new edge creating between B and C; and E5 is the new edge creating between C and D, and E6 is the new edge creating between D and B.

SurfaceCorner Points Merging can help the edge loop, but it may cause more small faces because the short edge loop may form new loops after creating the new edge between the close SurfaceCorner points as illustrated in the righthand side picture in the figure above. Hence, several rules have to be applied in the SurfaceCorner Points Merging Process. These rules are given below:

- Two SurfaceCorner points are close

The new edge of two SurfaceCorner points should be on the same surface.

If they are too far, they may cross another face.

- Always process closest pair of SurfaceCorner points

Closest pair of SurfaceCorner points has more chance to be on the same surface.

When the short edge loop forms after a SurfaceCorner points merging operation, the edge with closest pair of SurfaceCorner points should be kept, and the edge with further pair of SurfaceCorner points should be removed.

- No intersection with any existing edges

This pair of SurfaceCorner points should be connected to an edge because one existing edge intersects it.

- No short edge loop must be created

It can avoid detection of very small faces.

The first two rules are used before creating the new edge; and 3rd and 4th rules are used to cancel the new edge after it has been generated.

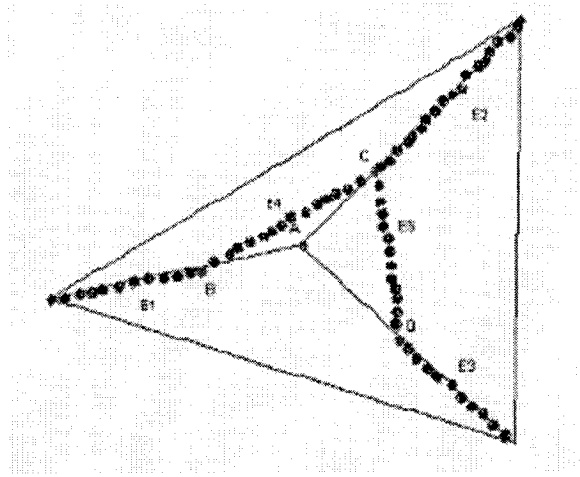


Figure 4-11: Final Result of SurfaceCorner points merge operation.

Hence, the new edge E6 between B and D is cancelled after checking 4th rule; and the final result of the SurfaceCorner points merging process is as shown in the figure 4-11.

4.3.6 Edge Combination

This process is to combine two lines together if they are connected to each other; and they can fit the feature of the edge. Right now, we only consider combination of the straight lines, but we can easily extend our method so that curves can also be combined.

We are using the Least Square Fitting approach to combine the edges. Different formula can be defined for fitting different features. For 3D straight lines, we have used Linear Fitting Using Orthogonal Regression in Eberly et al. [11].

4.3.6.1 General Linear Least Squares Fitting

We have already described this in section 2.2.3. for a more detailed discussion, please see Feddima et al. [12].

4.3.6.2 Linear Fitting Using Orthogonal Regression

Let a line be $L(t) = tD + A$ where D is unit length, and A is the reference position on the line.

Define X_i to be the sample points,

$$X_i = A + d_i D + p_i D^\perp \dots\dots\dots(1)$$

Where $d_i = D \bullet (X_i - A)$, and D^\perp is some unit length vector perpendicular to D with appropriate coefficient p_i . Define $Y_i = X_i - A$. The vector from X_i to its projection onto the line is

$$Y_i - d_i D = p_i D^\perp \dots\dots\dots(2)$$

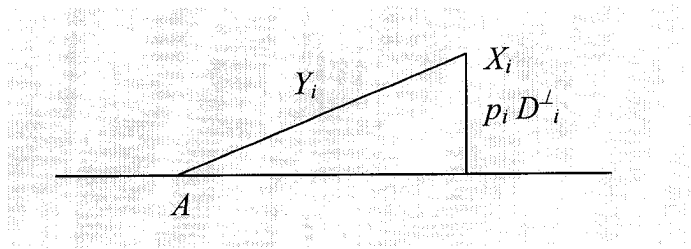


Figure 4-12: The distance from one point to 3D line.

The squared length of this vector is $(p_i)^2 = (Y_i - d_i D)^2$, The energy function for the least squares minimization is $E(A, D) = \sum_{i=1}^m (p_i)^2$. Two alternate forms for this function are

$$E(A, D) = \sum_{i=1}^m (Y_i^t \bullet [I - D D^t] \bullet Y_i) \dots\dots\dots(3)$$

And

$$E(A, D) = D^t (\sum_{i=1}^m [(Y_i \bullet Y_i) I - Y_i \bullet Y_i^t]) D = D^t M(A) D \dots\dots\dots(4)$$

Using the first form of E in the previous equation (3), take the derivative with respect to A to get

$$\partial E / \partial A = -2[I - D D^t] \sum_{i=1}^m Y_i \quad \dots\dots\dots(5)$$

This partial derivative is zero whenever $\sum_{i=1}^m Y_i = 0$

in which case $A = (1/m) \sum_{i=1}^m Y_i$ (the average of the sample points).

Given A , the matrix $M(A)$ is determined in the second form of the energy function. The quantity $D^t M(A) D$ is a quadratic form whose minimum is the smallest eigen value of $M(A)$. This can be found by standard eigen matrix solvers. A corresponding unit length eigenvector D completes our construction of the least squares line.

For $n = 2$, if $A = (a, b)$, then matrix $M(A)$ is given by

$$M(A) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{\begin{bmatrix} \sum_{i=1}^m (x_i - a)^2 & \sum_{i=1}^m (x_i - a)(y_i - b) \\ \sum_{i=1}^m (x_i - a)(y_i - b) & \sum_{i=1}^m (y_i - b)^2 \end{bmatrix}}{\sum_{i=1}^m (x_i - a)^2 + \sum_{i=1}^m (y_i - b)^2}$$

For $n = 3$, if $A = (a, b, c)$, then matrix $M(A)$ is given by

$$M(A) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{\begin{bmatrix} \sum_{i=1}^m (x_i - a)^2 & \sum_{i=1}^m (x_i - a)(y_i - b) & \sum_{i=1}^m (x_i - a)(z_i - c) \\ \sum_{i=1}^m (x_i - a)(y_i - b) & \sum_{i=1}^m (y_i - b)^2 & \sum_{i=1}^m (y_i - b)(z_i - c) \\ \sum_{i=1}^m (x_i - a)(z_i - c) & \sum_{i=1}^m (y_i - b)(z_i - c) & \sum_{i=1}^m (z_i - c)^2 \end{bmatrix}}{\sum_{i=1}^m (x_i - a)^2 + \sum_{i=1}^m (y_i - b)^2 + \sum_{i=1}^m (z_i - c)^2}$$

where

$$\delta = \sum_{i=1}^m (x_i - a)^2 + \sum_{i=1}^m (y_i - b)^2 + \sum_{i=1}^m (z_i - c)^2 .$$

4.4 Remarks

Detecting features such as vertices and edges are the first step in the process of segmenting a 3D point cloud model into point subsets that can be identified as faces, edges and corners. The techniques are sensitive to a number of model related properties. These are the type of object (engineering class or the class consisting of natural or sculptured shapes), the sampling density, the variation in sampling density over the entire model and finally the noise in data set. We have devised techniques that try to take into account all the above characteristics. These techniques use statistical properties of the point data set and adaptively derive the required thresholds needed for reasonably accurate classification. This is followed by fairly complex methods for refining the initial classification and then traversing the point set space for getting edge points. These techniques have been implemented and tested on a wide variety of models. The results are presented in chapter 6 of this thesis.

5 Face Recovery Techniques for Point Cloud Models

In Chapter 4 we have discussed the techniques that we have devised for detecting points that can be identified as vertices and points that can be identified as making up the different edges in the 3D object. At the end of the edge detection process all the unidentified points will be of type `FaceInterior`. The next step in reverse engineering the boundary representation (B-rep) of the 3D object is to segment the entire point set into different faces that make up the object. It may be noted that in a B-rep model of the surface of a 3D object, each face may be bounded by one or more edge loops. In case of a closed surface, each edge will belong to exactly 2 faces. In the case of an open surface, the edge made up of `SurfaceBorder` points will belong to only one face. In this chapter we describe the techniques we have devised for detecting edge loops and for segmenting the point set into different faces bounded by edge loops. We use the popular half-edge data-structure to represent the B-rep recovered by this process.

5.1 Half Edge Data Structure

The half-edge data structure is an edge-centered data structure capable of maintaining incidence information of vertices, edges and faces. The basic half-edge data structure is as follows:

Edge Item Definition:

- Edge next.
- Edge opposite_halfedge
- Vertex end_vertex
- Face face

Face Item Definition:

- Edge start_edge

Vertex Item Definition:

- Double x
- Double y
- Double z

The basic half edge data structure cannot handle hole loops, i.e., faces with multiply connected boundary loops. For our surface structure, we do consider holes. For a face with multiple edge loops, one loop is the outside boundary, and other loops make up the inside boundary of the face. We extended the basic half edge data structure in order to handle this case. Hence, we add another layer between the face and the half edge layer. This additional layer is half edge loops which consist of all half edges in the loop.

The face recovery process has three stages. The first stage is to construct the half edge node list; and second step is to detect half edge loops from the half edges; and the third step is to segment the point set into different faces from the half edge loops.

Our half edge data structure has four major nodes: vertex, half edge, half edge loop and face. In order to make an efficient structure, we have to add more links to the whole structure.

5.1.1 Vertex/Half Edge Structure:

There are several types of node structures in the Vertex/Half Edge Structure in order to query the information quickly. They are Vertex Node, Start Vertex Node, Edge list, Half Edge Node. Vertex Node just keeps the 3D coordinates information. Edge list is the source data of the half edge construction; and it comes from the previous process of edge recovery. Each edge has a start vertex and end vertex. In the half edge node, we only keep the start vertex reference. Start Vertex Node is to collect all half edge nodes which have the same start vertex. All Half Edge Nodes are stored in one list which is used to generate the half edge loop later. Each Half Edge Node has a pointer to Start Vertex Node, a pointer to the opposite Half Edge Node and a pointer to Loop Element. Loop Element is valid only if the half edge has been inserted into a half edge loop. The Loop Element will be described in next section.

These structures are used in the first stage of the face recovery process. This process takes the edge array as the source, and it generates other three data structures.

Vert/Edge/Half Edge Organization

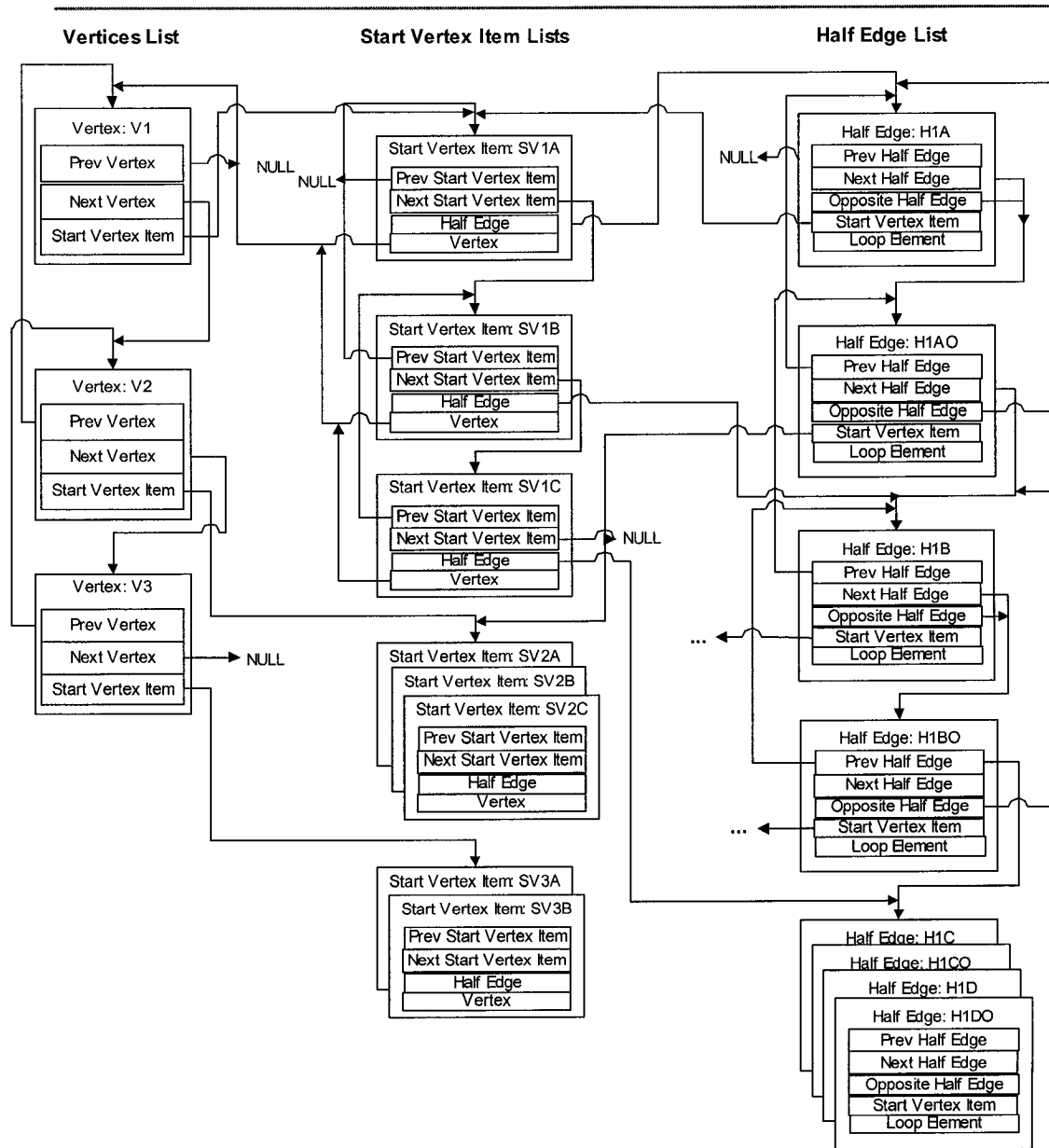


Figure 5-1: Vertex/Start Vertex/Half Edge Organization.

The Start Vertex Item is a data structure that supports query on the next half edge during the process of the half edge loop. The query on Half Edge A can be done easily using the following steps:

- A -> Opposite Half Edge -> StartVertexItem
- All items except opposite half edge in this Start Vertex Item list are the next half edge of A.

5.1.2 Half Edge/Half Edge Loop/Face Structure:

The face consists of at least one half edge loop; a half edge loop consists of at least two half edge elements; a half edge element is a reference to half edge node.

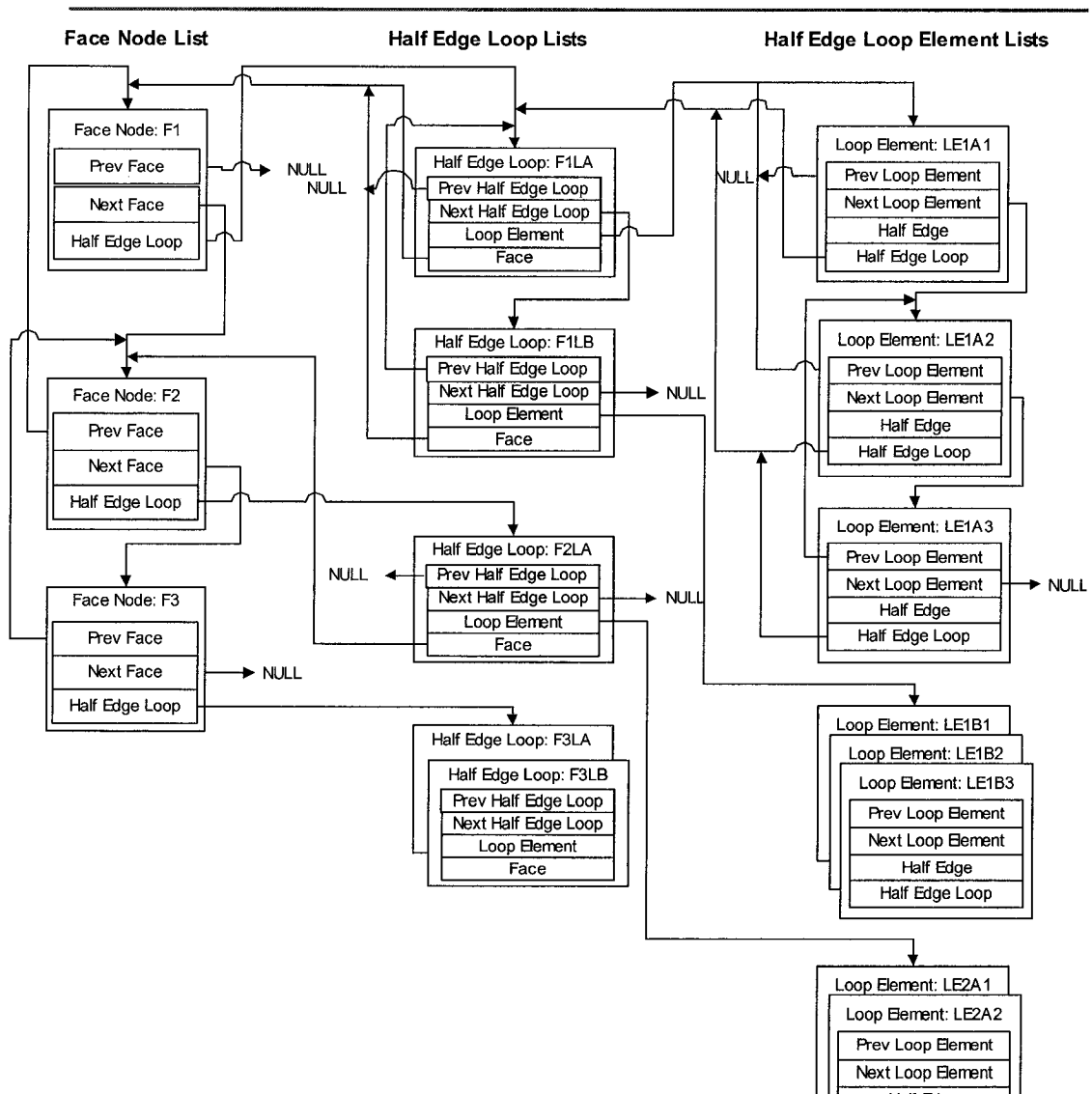


Figure 5-2: Half Edge/Half Edge Loop/Face Organization.

5.1.3 Three types of half edge loops to split surface:

The half edge structure for the entire 3D object is built incrementally by considering one edge at a time. Introduction of a new half edge will cause changes to the current state of face structure. Each face is usually surrounded by one half edge loop, but some faces may consist of more than one half edge loop.

One half edge can be part of a half edge loop or may be the start of a new half edge loop; A half edge cannot be the start and end half edge of the same loop. It will connect to another half edge loop. There are 3 types of loop endings.

5.1.3.1 Half Edge Loop Type A:

The new half edges ends at the beginning of the loop itself.

If the new half edge loop ends at the beginning of itself, all opposite half edge of this loop's half edge should connect to build another loop. If the first loop is outside, then the opposite loop is inside. Both loops split the old face into two faces.

There are two cases of type A. One is that one face is surrounded by one half edge loop. Second case is that one face is surrounded by more than one half edge loops.

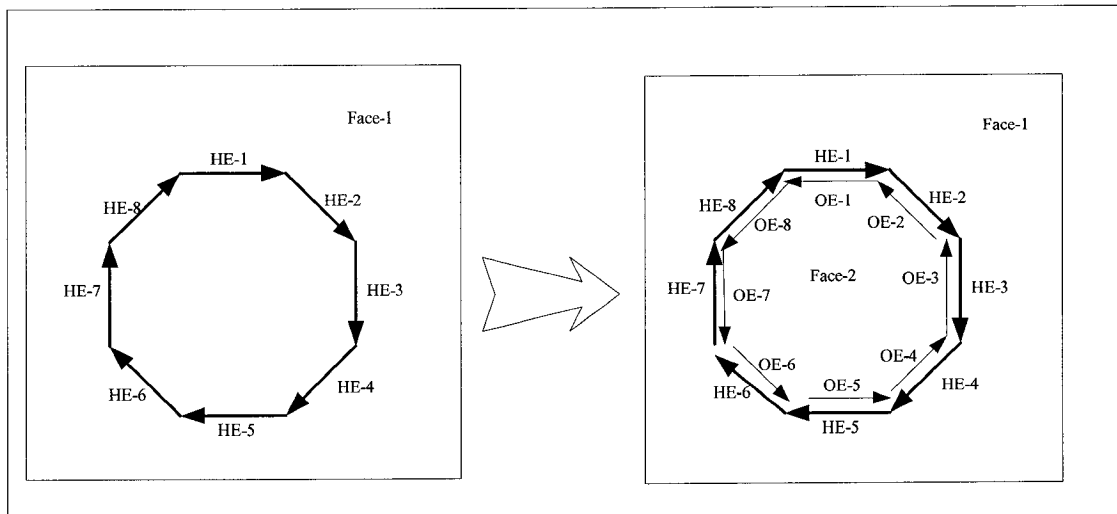


Figure 5-3: First case of Half Edge Loop Type A.

In this case one new half edge loop ($HE-1 - HE-8$) is found. It is beginning at $HE-1$, and is ending at $HE-1$. Its opposite half edges ($OE-1 - OE-8$) form another loop.

Both loops split old *Face-1* into two faces: *Face-1* and *Face-2*. *Face-1* has the loop (*HE-1* – *HE-8*) as the boundary, and *Face-2* has the loop (*OE-1* – *OE-8*) as the boundary.

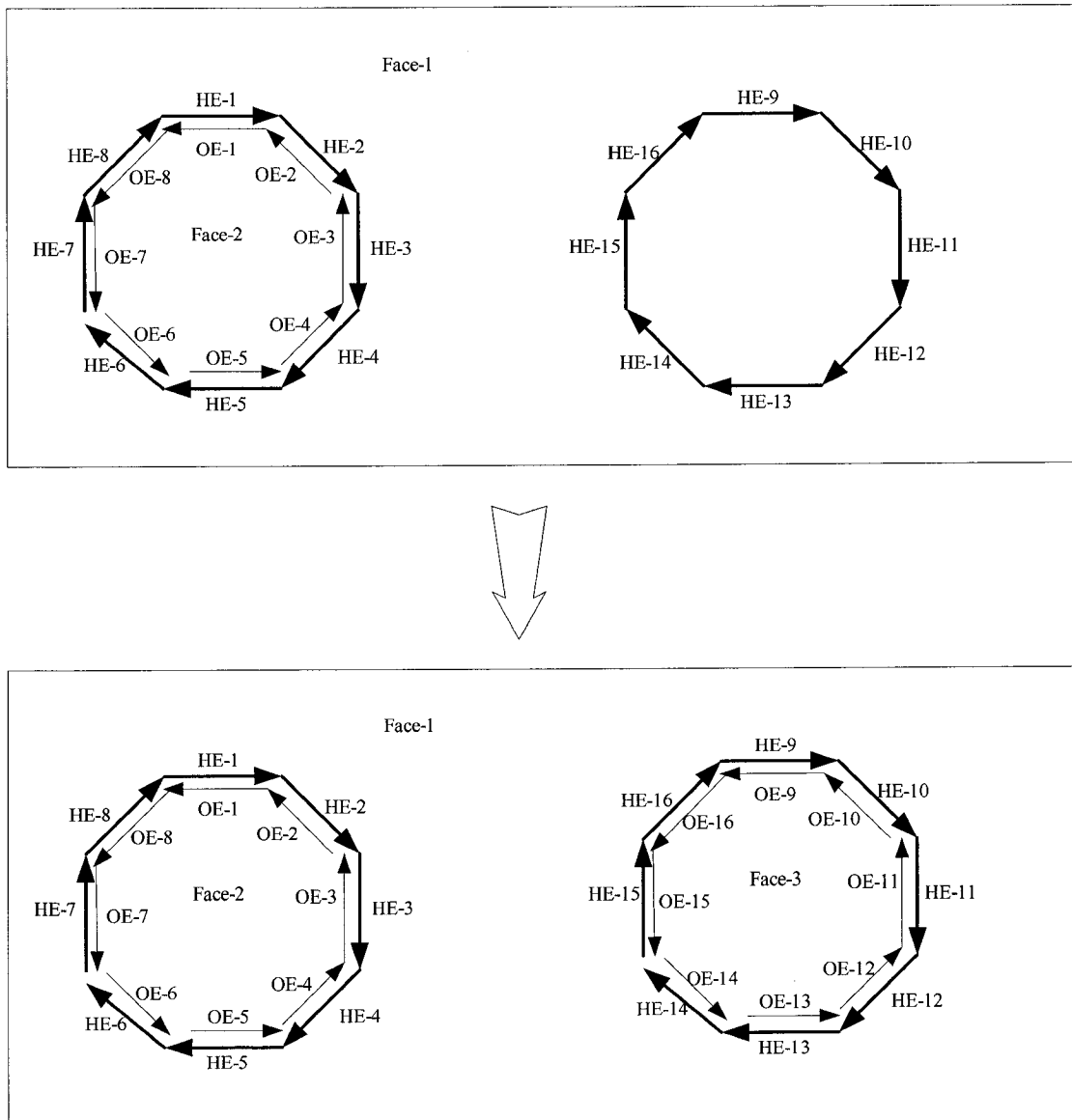


Figure 5-4: Second case of Half Edge Loop Type A.

In the second case, the new Face-1 is surrounded by two half edge loops (from *HE-1* to *HE-8* and from *HE-9* to *HE-16*) after the old *Face-1* is split into new faces, *Face-1* and *Face-3*.

If another isolated half edge loop is found like the loop (from *HE-9* to *HE-16*) in the *Face-1* again, the result will be three loops surrounding the new face.

5.1.3.2 Half Edge Loop Type B:

The new half edge loop begins at one existing loop, and ends at the same existing loop.

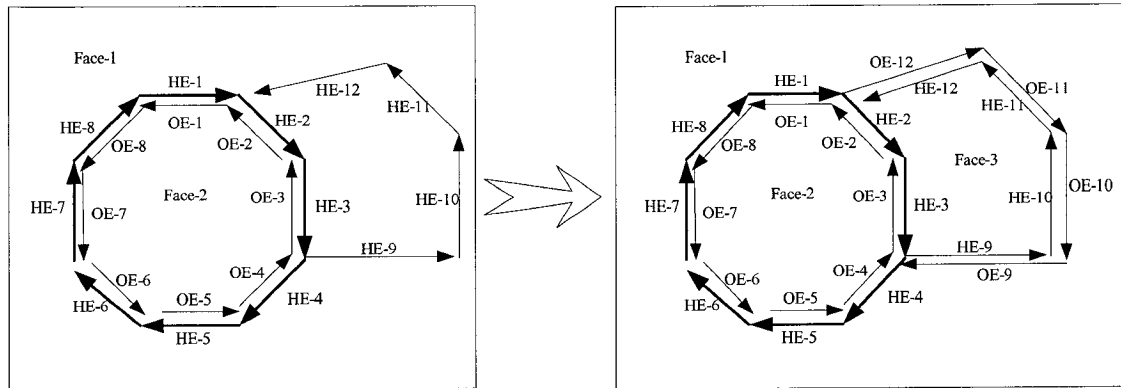


Figure 5-5: Half Edge Loop Type B.

Based on the example of case-1 in Half Edge Loop Type A, there is the new half edge connectivity (*HE-9 – HE-12*) which is beginning after *HE-3*, and is ending at *HE-1*'s end vertex. Hence, it breaks the old half edge loop (*HE-1 – HE-8*)

5.1.3.3 Half Edge Loop Type C:

The new half edge loop begins at one existing loop, and ends at another existing loop. It does not form another face, and it breaks both existing loop, and connects both broken loops into another loop plus itself and its opposite edges.

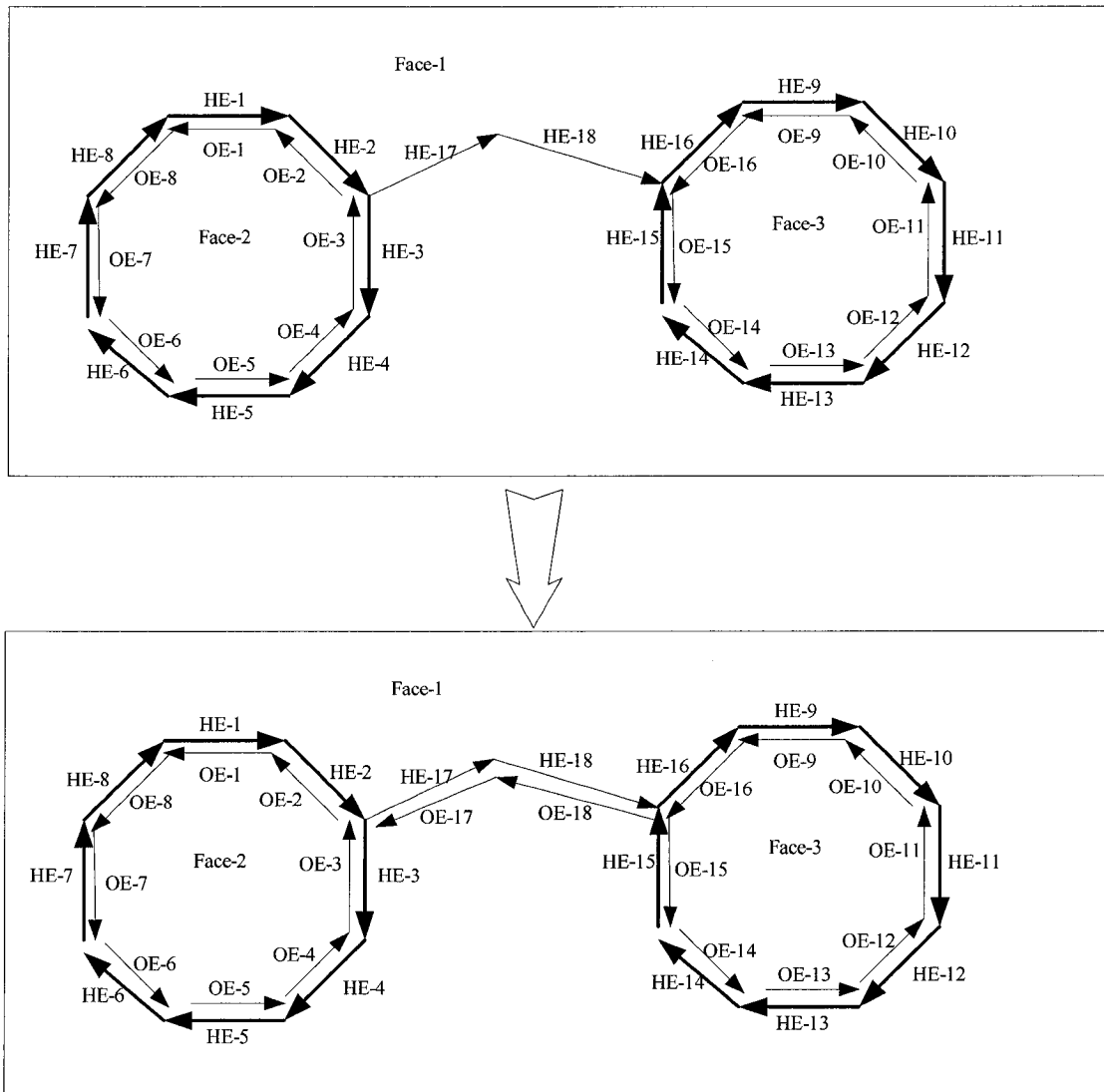


Figure 5-6: Half Edge Loop Type C.

Based on the example of the case-2 in Half Edge Loop Type A, there is the new half edge connectivity ($HE-17 - HE-18$) which begins after $HE-2$ of the first edge loop, and ends after $HE-15$ of the second edge loop.

It breaks the first edge loop from $HE-1 - HE-8$ to $HE-3 - HE-8 - HE-2$, and it breaks the second edge loop from $HE-9 - HE-16$ to $HE-16 - HE-9 - HE-15$, and it forms the new edge connectivity ($HE-3 - HE-8 - HE-2 - HE-17 - HE-18 - HE-16 - HE-9 - HE-15$). After adding the opposite edges of the edge $HE-17$ and $HE-18$, the new edge connectivity becomes the new edge loop: $HE-3 - HE-8 - HE-2 - HE-17 - HE-18 - HE-16 - HE-9 - HE-15 - OE-18 - OE-17$.

However, the new edge loop does not split the old face: *Face-1*. It just combines two old edge loops into one edge loop, and it keeps the same face: *Face-1*.

Hence, Half Edge Loop Type A and Half Edge Loop Type B always split one face into two faces, but Half Edge Loop Type C does not split the face.

5.2 Surface Split Algorithm

5.2.1 Types of points used in surface splitting:

5.2.1.1 Point Types after Face Segmentation

- *Processed Internal Face Vertex (PDIFV)*
- *Processed Edge End Vertex (PDEEV)*
- *Processed Edge Vertex (PDEV)*

Processed Internal Face Vertex (PDIFV) is the internal point inside the face. Processed Edge End Vertex (PDEEV) and Processed Edge Vertex (PDEV) are the points on the boundary of the face.

5.2.1.2 Types Used During Processing

- *Processing Internal Face Vertex (PGIFV)*
- *Processing Close Boundary Vertex (PGCBV)*
- *Processing Old Edge End Vertex (PGOEEV)*
- *Processing New Edge End Vertex (PGNEEV)*
- *Processing Shared Edge End Vertex (PGSEEV)*
- *Processing Old Edge Vertex (PGOEV)*
- *Processing New Edge Vertex (PGNEV)*
- *Processing Shared Edge Vertex (PGSEV)*

Processing Internal Face Vertex (PGIFV) is the initial value of the points belonging to the old face which is going to be split.

When the new edge splits an old face into two new faces, each new face will have its own edge loops. One new face may only have one edge loop, and it may also have more than one edge loop like the second case of Half Edge Loop Type A. We assume one of the new faces is still the old face because it uses the old face structure and face id, and another face is the new face created after the split. The points of the old face boundary are Processing Old Edge End Vertex if they are the end of the edges or Processing Old Edge Vertex if they are not the end of the edges. The points of the new face boundary are designated as Processing New Edge End Vertex if they are the end of the edges or Processing New Edge Vertex if they are not the end of the edges. However, both half edges are the shared edge between the old and new face if the opposite half edge of the one old boundary edge is also the new boundary edge; and

the end points of the shared edges are Processing Shared Edge End Vertex (PGSEEV), and other points of the shared edges are Processing Shared Edge Vertex (PGSEV).

During the first phase of the face split process, the point is called Processing Close Boundary Vertex (PGCBV) if it is very close to the shared edges, and it is not clear which face it belongs to. It will be identified later in Close-Boundary Vertex Processing.

5.2.2 A 3D Seed Fill Algorithm:

5.2.2.1 How Does the 3D Seed Fill Algorithm Work

We have extended the idea of the 2D seed fill algorithm used extensively in computer graphics for raster filling a region given the region boundary and a seed that is guaranteed to be interior to the region. This algorithm works by recursively growing the region around the seed until a boundary pixel is hit. Our 3D version takes one FaceInterior point as the seed, and starts from the seed point to extend each neighboring point. Each FaceInterior point in the neighbourhood is now considered as belonging to this face. This process is continued recursively by considering each newly added point as the new seed. The process terminates when a non FaceInterior point is reached.

The boundary of the face in this 3D Seed Fill Algorithm means the loop of shared edges. The new or old edge type vertices do not affect 3D Seed Fill Algorithm because they are the edge between the split faces. Meantime, if there is at least one neighbor vertex whose type is Processing Shared Edge End Vertex (PGSEEV) or Processing Shared Edge Vertex (PGSEV), Seed Fill Algorithm marks the seed as

Processing Close Boundary Vertex (PGCBV) and stops processing of the neighbor point of the current seed. The pseudo-code for this algorithm is given below.

3DSeedFillAlgorithm(Seed)

- *Collect all neighbor vertices of the seed*
- *For each neighbor vertex*
- *If the vertex is Processing Shared Edge End Vertex (PGSEEV) or Processing Shared Edge Vertex (PGSEV)*
- *Set Seed to Processing Close Boundary Vertex (PGCBV)), and quit this function*
- *End if*
- *End Loop*
- *For each neighbor vertex*
- *If the neighbor vertex is Processing Internal Face Vertex (PGIFV)*
- *Set the same face for this neighbor vertex as the seed.*
- *SeedFillAlgorithm(the neighbor vertex)*
- *End if*
- *End Loop*

5.2.2.2 Efficiency of 3D Seed Fill Algorithm

This 3D Seed Fill Algorithm is very simple and efficient because it does not have much computation. It just searches the neighbor points for each seed, and checks the type of each neighboring point. Collecting neighboring points and checking the type are computationally quite inexpensive and fast.

Furthermore, since most of the points in the face are interior points, 3D Seed Fill algorithm identifies the face id for these points fast. Only a very small portion of vertices are Processing Close Boundary Vertex (PGCBV), and these vertices have to use a more complicated algorithm to identify the face id later. In our experience, usually less than 3% of the vertices need the Close-Boundary Vertex Process.

5.2.3 Close-Boundary Vertex Processing:

After the Seed Fill process, most points on the surface have been identified as belonging to one of the current set of faces in the half-edge structure, except all the points that are very close to any edge of new/old face which need further processing. We call these points as Close-Boundary Vertices. The Close-boundary Vertex Process is used to identify the faces for these points.

The neighbor points of Close-Boundary Vertex are very many different types, but only some types will affect the processing of Close-Boundary Vertex. The different types that are not used are:

- *Processed Edge End Vertex (PDEEV)*
- *Processed Edge Vertex (PDEV)*
- *Processing Old Edge End Vertex (PGOEEV)*
- *Processing New Edge End Vertex (PGNEEV)*
- *Processing Old Edge Vertex (PGOEV)*
- *Processing New Edge Vertex (PGNEV)*

PGOEV is the boundary of the old face; and it is not shared with new face.

PGNEV is the boundary of the new face; and it is not shared with old face. Both

PGOEV and PGNEV are used to identify which face the points belong to in the 3D

Seed Fill process. However, the edge cannot be used to identify the face if the edge is

shared by old and new faces. The types of the shared edge are PGSEEV and PGSEV

described below.

The only types that are used in the processing of Close-Boundary Vertex are:

- *Processed Internal Face Vertex (PDIFV)*

- *Processing Internal Face Vertex (PGIFV)*
- *Processing Close Boundary Vertex (PGCBV)*
- *Processing Shared Edge End Vertex (PGSEEV)*
- *Processing Shared Edge Vertex (PGSEV)*

Processing Shared Edge End Vertex (PGSEEV) and Processing Shared Edge Vertex (PGSEV) compose the edge between two faces. The Close-Boundary Vertex Process identifies the face for an unknown point based on the relationship between points with previously established identities and itself. There are three kinds of relationships between neighboring FaceInterior points and the seed vertex that the close-boundary process will identify.

These three types of relationships are same-face, different-face and unknown. The same-face relationship means the seed vertex stays on the same face as the neighboring FaceInterior point. The different-face relationship means the seed vertex stays on a different face as compared to the neighboring FaceInterior point. There are only two faces to be identified for the seed vertex: old face and new face, so the seed vertex belongs to old face if its' neighboring FaceInterior point belongs to the new face, and vice versa. The unknown relationship means it cannot be used to identify the face. The unknown relationship happens when one of the following situations exist.

- *The neighboring FaceInterior point is far away from the seed vertex.*
- *The neighboring FaceInterior point and the seed vertex are not on similar plane*
- *The segment from the neighboring FaceInterior point for the seed vertex crosses more than one segment on the shared edges.*

Two points are on similar plane if:

both points have similar normal directions;

and the distance between two vertices along the normal directions of either vertex \leq predefined distance.

The segment from the neighboring FaceInterior point for the seed vertex crosses more than one segment on the shared edges. Usually it means the segment crosses more than one edge, but it may also cross one shared edge twice sometime. However, we still say it only crosses the shared edge once if two crossed segments on the shared edge are adjacent and both crossing positions are close to the same edge vertex.

The relationship between a neighboring FaceInterior point and the seed vertex is the same-face relationship if all the following conditions hold:

- *Both vertices are close*
- *Both are on the same plane*
- *They do not cross any shared edges.*

The relationship between a neighboring FaceInterior point and the seed vertex is a different-face relationship if all the following conditions hold:

- *Both vertices are close*
- *Both are on the same plane*
- *They do cross a shared edge once or they cross a shared edge twice but both crossing positions are close to the same vertex or they cross two adjacent shared edges but both crossing positions are close to the same edge end vertex.*

The relationship between the neighborhood points and the seed vertex becomes an unknown relationship when either same-face relationship or different-face relationship conditions are not satisfied.

The close-boundary process gets the relationship between each neighboring FaceInterior point and the seed vertex. If there is any same-face relationship vertex or different-face relationship vertex among the neighborhood vertices, the seed vertex can be associated with a face according to these neighboring points. Nevertheless, there may be conflicts in the relationships of the neighborhood points. For example, two neighboring points may be identified with the same-face relationship to the seed; but these two points actually belong to different faces. So this can cause a conflicting situation if we try to identify the face for the seed. In this case, the process will not identify the face for the seed vertex and keep the original status of the seed vertex. The seed vertex could be identified later when it is a neighborhood vertex of other seed vertex and they have same-face or different-face relationship.

After the seed vertex is identified, the process continues to identify other unknown face vertices in the neighborhood to check if they have same-face or different-face relationship with the seed vertex.

5.2.4 Close-Boundary Vertex Processing Algorithm:

- *Get the neighborhood vertices set $NGP(S)$*
- *Get the Processing Shared Edge End Vertices($PGSEEV$) and the Processing Shared Edge Vertices ($PGSEV$) as $SV(S)$*
- *Find the adjacent edge vertices for each of $SV(S)$, and order it.*
- */* Check the relationship between each neighbor vertex and Seed vertex(S) */*
- *For each vertex V in $NGP(S)$*

- *If the distance of SV exceed the pre-defined distance of close-boundary checking.*
- *Skip the following process.*
- *Else*
- *If V and S are on the same plane*
- *Set SPL = YES.*
- *For each pair of adjacent shared edge vertices in SV(S) as SV1, SV2*
- *If the segment of SV1 and SV2 is crossing the segment of S and V*
- *If the number of crossing segments NCS= 1 AND previous shared edge vertices and the shared edge vertices(SV1,V2) are the adjacent edge vertices pair AND the crossing position is close to the middle edge vertex of these two pairs of shared edge vertices.*
- *;*
- *Else if the number of crossing segments NCS = 0*
- *NCS++*
- *Keep the pair of the shared edge vertices (SV1, SV2) and the crossing position.*
- *Else if the number of crossing segments NCS > 1*
- *NCS++*
- *If V and S are not on the same plane*
- *Set SPL = NO.*
- */* Determine the New/Old Face */*
- *For each vertex V in NGP(S)*
- *If the face of V = Unknown*
- *Skip it.*
- *If the distance of SV < the pre-defined distance of close-boundary checking AND SPL is YES AND NCS = 0*
- *If the face of S Face(S) = Unknown*
- *Set Face(S) = Face(V)*
- *Else if Face(S) <> Face(V) /* If the checking is inconsistency */*
- *Set Face(S) = Unknown_But_Processed*
- *Return.*
- *Else If the distance of SV < the pre-defined distance of close-boundary checking AND SPL is YES AND NCS = 1*

- *If the face of S $Face(S) = Unknown$*
- *Set $Face(S) = Opposite\ of\ Face(V)$*
- */*if $Face(V)$ is new, then $Face(S)$ is old, and vice versa.*/*
- *Else if $Face(S) = Face(V)$ /* If the checking is inconsistency */*
- *Set $Face(S) = Unknown_But_Processed$*
- *Return.*
- *Else If the distance of $SV < the\ pre-defined\ distance\ of\ close-boundary\ checking$*
AND SPL is YES AND $NCS > 1$
- *Skip it. /* It can not be used because of too many crossing shared*
edges./*
- *Else If the distance of $SV < the\ pre-defined\ distance\ of\ close-boundary\ checking$*
AND SPL is NO
- *Skip it. /* No crossing check because they are not on the same plane. */*
- *Else If the distance of $SV > the\ pre-defined\ distance\ of\ close-boundary\ checking$*
- *Skip it. /* No necessary for crossing checking because they are too far.*
**/*
- */* Update other close-boundary vertices in the neighbor vertices */*
- *If $Face(S) = Unknown$*
- *Set $Face(S) = Unknown_But_Processed$*
- *Return*
- *For each vertex V in $NGP(S)$*
- *If $Face(V) \neq Unknown$*
- *Skip it.*
- *If the distance of $SV < the\ pre-defined\ distance\ of\ close-boundary\ checking$ AND*
 SPL is YES AND $NCS = 0$
- *Set $Face(V) = Face(S)$*
- *Else If the distance of $SV < the\ pre-defined\ distance\ of\ close-boundary\ checking$*
AND SPL is YES AND $NCS = 1$
- *Set $Face(V) = Opposite\ of\ Face(S)$*

5.3 Remarks

Detecting edge loops is comparatively a straight forward task once the edges have been detected and the edge corner vertices are known. Segmenting the point set into different faces bounded by these edge loops is however a far more complex task, particularly, for points which are close to the edge points. The 3D seed fill algorithm that we have introduced in this chapter is simple and efficient for the more straight forward cases in labeling the points with their owner faces. For points that are very close to edges, more complex techniques have been devised. All the above techniques have been implemented and tested on a variety of models. The results are presented in the next chapter.

6 Implementation Issues and Results

All the techniques described in chapters 3, 4 and 5 have been implemented and tested on a number of point cloud models. For purposes of testing our programs, many of the 3D data models were available for download from other research web sites. However, we also wrote our own programs for creating some of the simpler data sets such as the cube, two cubes and the anvil. These were needed so that we could test our programs using small and simple models. The total size of the programs is over 38,000 lines of code written in C, Open GL, and GLUT on the windows 2000 platform. All performance measures were measured on the same system consisting of a 2.8GHz CPU and 1GB of memory. In this chapter we describe the overall structure of our program and the results of running these programs on different point sampled surface datasets.

6.1 *Program Structure/Workflow*

The processing of the 3D sampled data is very complicated because of the huge scale of data to be processed and the visual aspects of the results. Due to these reasons, it is also very hard to debug such programs.

The boundary recovery techniques implementation has three stages: initial process, edge recovering and face recovering; and all major processes are included in Table 6-1. The workflow of the implementation is described in figure 6-1.

Stage	Process	Function
Initial	Sample Data Loading	Load the data
	Cube Establish Process	Average Distance Analysis
		Neighborhood Size Analysis
	Noise Data Process	Multi-Partition Analysis
		Duplication Vertices Analysis
		Standalone Vertices Analysis
	Eigen Matrix Process	Jacobi Process
Edge Reversing	Initial Classification Process	Flat/Border/Crease/Corner Classifying Process
	Classification Refining Process	Crease Refining
		Border Refining
		Corner Refining
	Edge Traversing Process	Recursively Traverse Edge Vertices
	Edge Refining	Edge Intersecting Analysis
		Short Edges Loop Analysis
		Parallel Edges Analysis
		Edge Merging Process
Face Reversing	HalfEdge Process	HalfEdge Strucutre Establish
		HalfEdge Loop Analysis
	Face Splitting Process	SeedFill Process
		CloseBoundary Process

Table 6-1: Processes of the whole system

System Overview

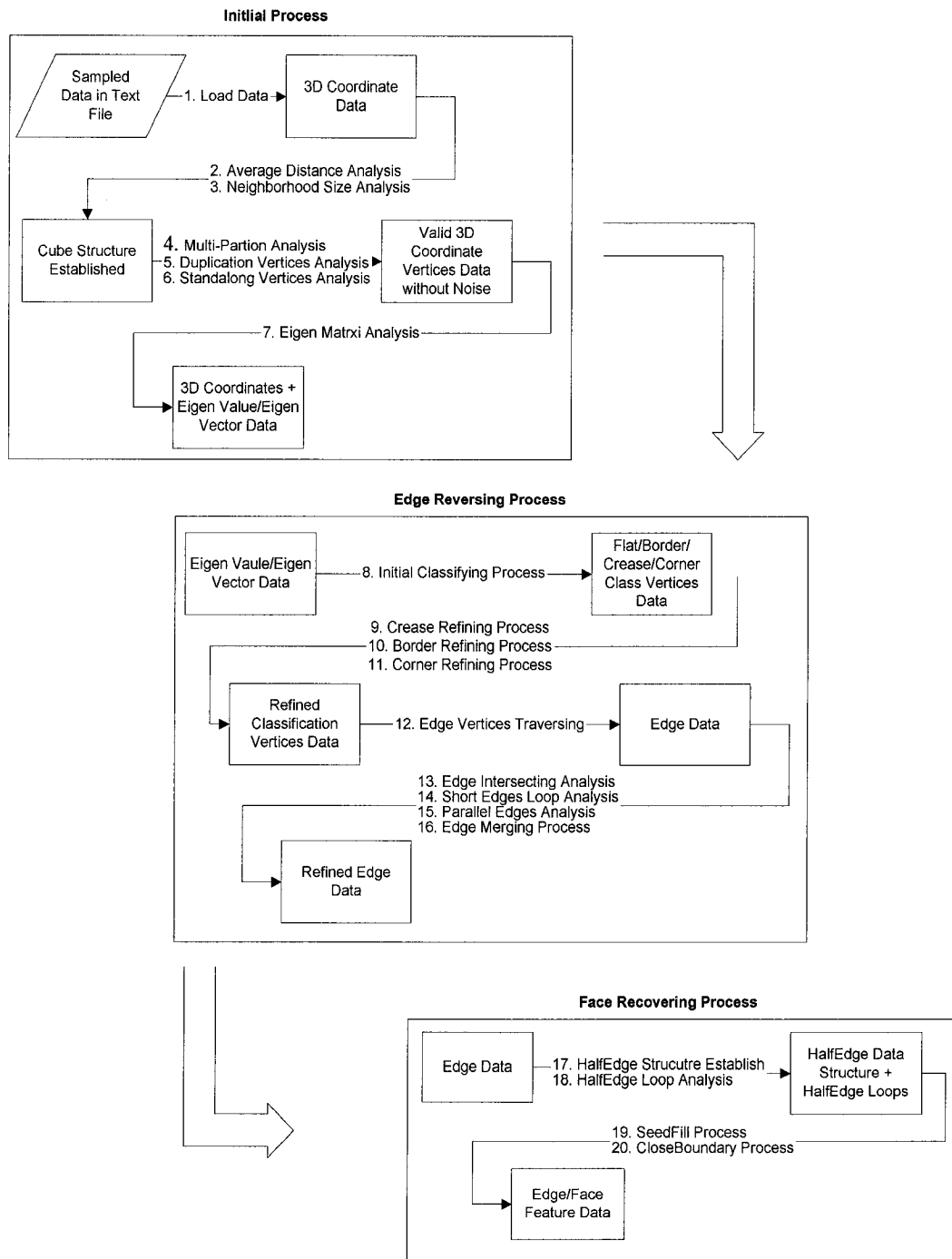


Figure 6-1: The overview of the system.

6.2 Information of each process

All models used in the tables of this chapter have been described earlier in Table 1-1.

6.2.1 Cube Establish Process

6.2.1.1 Cube Information

Model	NGP Size	Number of Valid Cubes	Cube Size	Average Vertices in Cube	Maximum Vertices in Cube
Single Box	3	1016	0.0144	59	169
Dual Boxes	3	1340	0.0072	55	169
Anvil	5	3794	2.5523751878	42	246
Armadillo	3	5904	0.0304893091	29	100
Bunny	3	1526	0.0374461665	23	60
Dragon	3	54695	0.0063659866	9	91
Dragon	5	22265	0.0106102264	24	172
Happy Budda	3	65364	0.0037658957	10	117
Happy Budda	5	29089	0.0062768406	24	184
Happy Budda	7	15501	0.0087875769	46	272

Table 6-2: The cube structure information.

Cube construction is the first process in the initial process. The cube size is the double of neighborhood size; hence, the neighbor point graphic for one point is computed in its cube and the eight cubes around it. Table 6-2 shows the cube structure information for each model. NGP Size is the estimating neighborhood radius based on the average distance.

6.2.1.2 Neighborhood Collection Information

Model	NGP Size	Neighborhood Radius	Average Neighborhood Number	Minimum Neighborhood Number	Maximum Neighborhood Number
Single Box	3	0.0072	25	19	33
Dual Boxes	3	0.0036	25	19	33
Anvil	5	1.2761875939	41	4	134
Armadillo	3	0.0152446545	21	12	69
Bunny	3	0.0187230832	16	5	24
Dragon	3	0.0031829933	9	2	74
Dragon	5	0.0053051132	22	2	130
Happy Budda	3	0.0018829478	10	2	92
Happy Budda	5	0.0031384203	23	2	170
Happy Budda	7	0.0043937884	44	2	249

Table 6-3: The Neighborhood vertices information.

Table 6-3 shows the average neighborhood vertices number, minimum neighborhood vertices number and maximum neighborhood vertices number according to the neighborhood size and the neighborhood radius for each model. The neighborhood radius is calculated based on the neighborhood size and the average distance of the model.

6.2.2 Processing Noise in Data

Model	Vertices	NGP Size	Partitions	Dup Noise	Standalone Noise
Single Box	60002	3	0	0	0
Dual Boxes	74402	3	0	0	0
Anvil	162882	5	0	0	0
Armadillo	172974	3	0	0	0
Bunny	35286	3	0	0	1
Dragon	566098	3	2082	25932	3041
Dragon	566098	5	55	25932	152
Happy Budda	727735	3	8844	7777	10024
Happy Budda	727735	5	1299	7777	794
Happy Budda	727735	7	85	7777	50

Table 6-4: Noise Analysis Data

Table 6-4 shows the separating partitions, duplication vertices and standalone vertices for each model. The separating partition means a part of vertices that can not be traversed based on the neighborhood size specified in the Table 6-4. Duplication vertices means two vertices are very close. The standalone vertices are the vertices that can not get any neighborhood vertices.



Figure 6-2: The noise analysis of the dragon.

Figure 6-2 shows the comparison between the dragon model with NGP 3 (a) and NGP 5 (b). The blue color points in figure 6-2 show the noise data. The dragon model with NGP 5 still has some noise data showing in the picture, but is has much less than its' NGP 3.

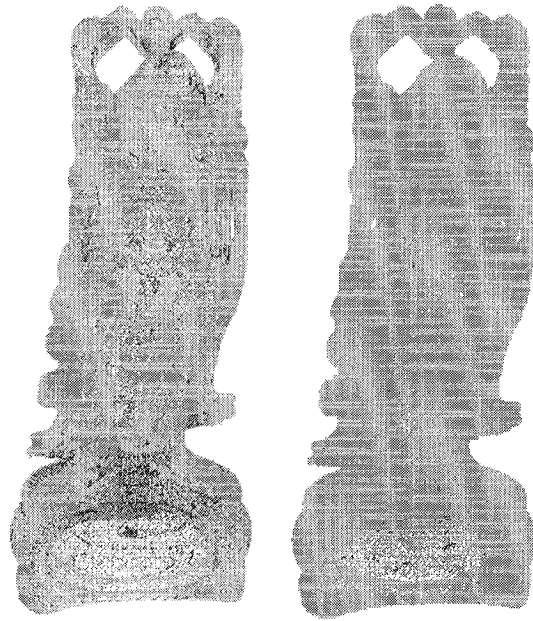


Figure 6-3: The noise analysis of Happy Budda model.

Figure 6-3 shows the comparison between the Happy Budda model with NGP 3 (a) and NGP 7 (b). The blue color points show the noise data. The Happy Budda model with NGP 7 still has some noise data showing in the picture, but it has much less than its' NGP 3. Happy Budda model with NGP 5 still has a lot of separating partitions.

Bunny has one point that is far away from any other points; and all other models have no noise point.

6.2.3 Eigen Matrix Computation Process

Model	NGP Size	Total Vertices	AVG Neighborhood Number	Time Cost
Single Box	3	60002	25	8.92s
Dual Boxes	3	74402	25	9.00s
Anvil	5	162882	41	34.50
Armadillo	3	172974	21	25.19
Bunny	3	35286	16	7.31s
Dragon	3	566098	9	42.0s
Dragon	5	566098	22	82.71s
Happy Budda	3	727735	10	74.91s
Happy Budda	5	727735	23	97.12s
Happy Budda	7	727735	44	174.77

Table 6-5: Eigen Matrix Calculation Information

Table 6-5 shows the time cost of the eigen vector/eigen value calculation for each model.

Time consumed = neighborhood vertices collecting time + eigen matrix calculation time.

6.2.4 Using Eigen Values in Rendering Process

- In chapter 3 we described our rendering algorithm that uses eigen values for efficient rendering of very dense point sampled surfaces.

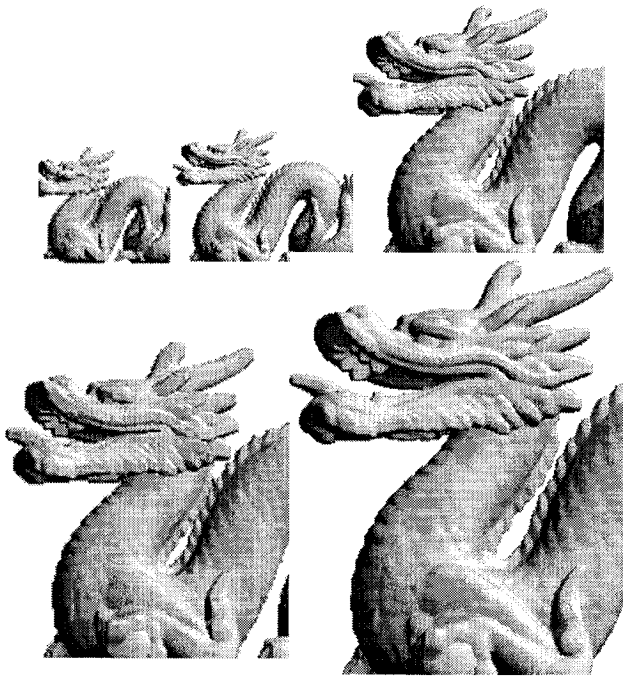


Figure 6-4: Rendering at different image sizes.

A dense region of the original surface is rendered using a small number of sample points. This selection uses multiple visual cues such as flatness of the region, presence of features such as an edge or silhouette.

A basic assumption that we have made is that for rendering a densely sampled flattish region without any other visually significant features we only need to render a few of the sample points. And further, every point in this region is “characteristically similar” to another. Since a large number of these points map onto the same pixel, there is nothing to choose one point from another. Hence we use uniform random sampling to select the subset of points to render. On the sample models we have used, the performance improvements are as shown in Table 3.2. One can observe that for smaller image sizes we get very good performance. This is as expected.

Continuing on the same line of thinking, we also assume that in a densely sampled region, the presence of any significant feature can be probabilistically determined by examining a smaller sample of the total set of points in the region. Accordingly in our algorithm we decide on the presence/absence of a feature using stochastic techniques rather than a totally deterministic approach that is used by all other algorithms. In all our experiments we have not found this causing any major problem. The figure above shows the results from our algorithm for different screen resolutions and zoom factors. Yet, there is the situation, however low its probability may be, that we could miss a feature and accordingly create not such an accurate rendering of the surface.

6.2.5 Initial Classification Process

When it comes to using the eigen value analysis in recovering the boundary representation for the 3D object, we have to carry out further analysis and classify the points into the different classes discussed earlier in chapter 4. Table below shows this classification for the different 3D objects we have experimented with.

Model	NGP Size	Flat Vertices	Border Vertices	Crease Vertices	Corner Vertices
Single Box	3	52114	2036	3516	2336
Dual Boxes	3	62517	2537	5612	3736
Anvil	5	129453	5373	21920	1915
Armadillo	3	140755	5822	24269	2128
Bunny	3	28727	1180	4943	435
Dragon	3	419824	17403	74811	6571
Dragon	5	439407	18168	75501	6661
Happy Budda	3	533214	21968	91068	7974
Happy Budda	5	578778	24099	99031	8707
Happy Budda	7	587045	24285	99589	8714

Table 6-6: Initial Classification Result.

Table 6-6 shows the result of the initial classification for each model. All points of each model are classified into four types: flat vertices, border vertices, crease vertices and corner vertices. The classification information will be used in the edge recovery process. The initial classification is based on predefined portion of each class.

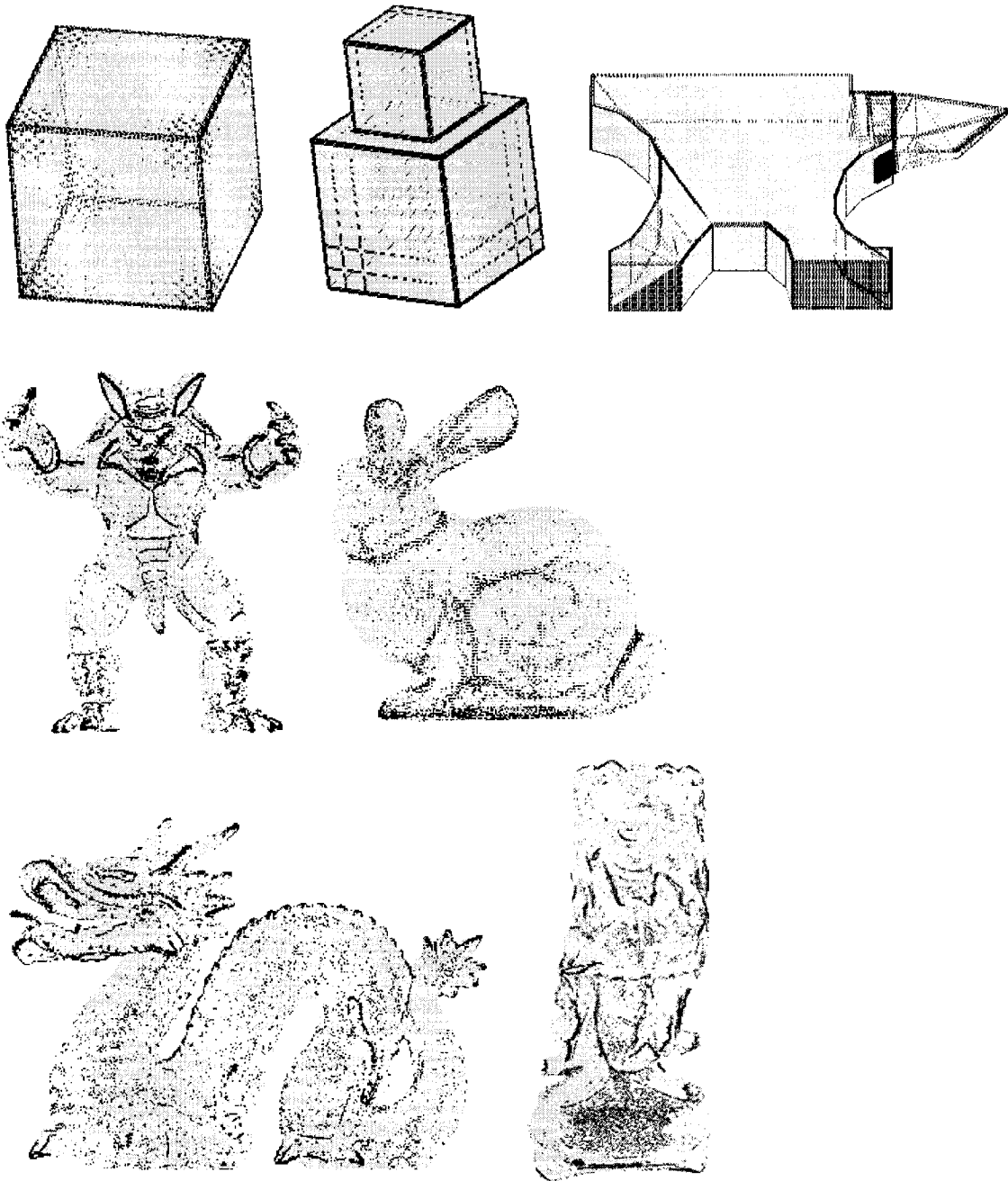


Figure 6-5: The initial classification information of each model.

Figure 6-5 show the result of the initial classification for a) Single Box b)Dual Boxes c)Anvil d)Armadillo e)Bunny f)Dragon using NGP 5 g)Happy Budda using NGP 7. The red points are the corner points; and the blue points are crease points; and the green points are border points; and the yellow points are the flat points in figure 6-5.

6.2.6 Classification Refining Process

Model	NGP Size	Flat Vertices		Border Vertices		Crease Vertices		Corner Vertices	
		Before	After	Before	After	Before	After	Before	After
Single Box	3	52114	58770	2036	0	3516	1212	2336	20
Dual Boxes	3	62517	72412	2537	0	5612	1938	3736	52
Anvil	5	129453	144077	5373	3219	21920	10390	1915	975
Armadillo	3	140755	160715	5822	11	24269	11198	2128	1050
Bunny	3	28727	32323	1180	96	4943	2626	435	240
Dragon	3	419824	455511	17403	14647	74811	44637	6571	3814
Dragon	5	439407	501974	18168	11766	75501	23518	6661	2479
Happy Budda	3	533214	579059	21968	18603	91068	51956	7974	4606
Happy Budda	5	578778	660059	24099	17011	99031	30366	8707	3179
Happy Budda	7	587045	686364	24285	12111	99589	18962	8714	2196

Table 6-7: The initial classification and the refined classification comparison.

Table 6-7 shows most of the border vertices in the initial classification are re-identified to the flat type; and a part of the crease vertices and the corner vertices also has been re-identified. These models should have no border vertices because they are close surfaces, but there still are a small number of border vertices in Happy Budda because the continuous of the planar estimation values.

The result after Classification Refining Process:

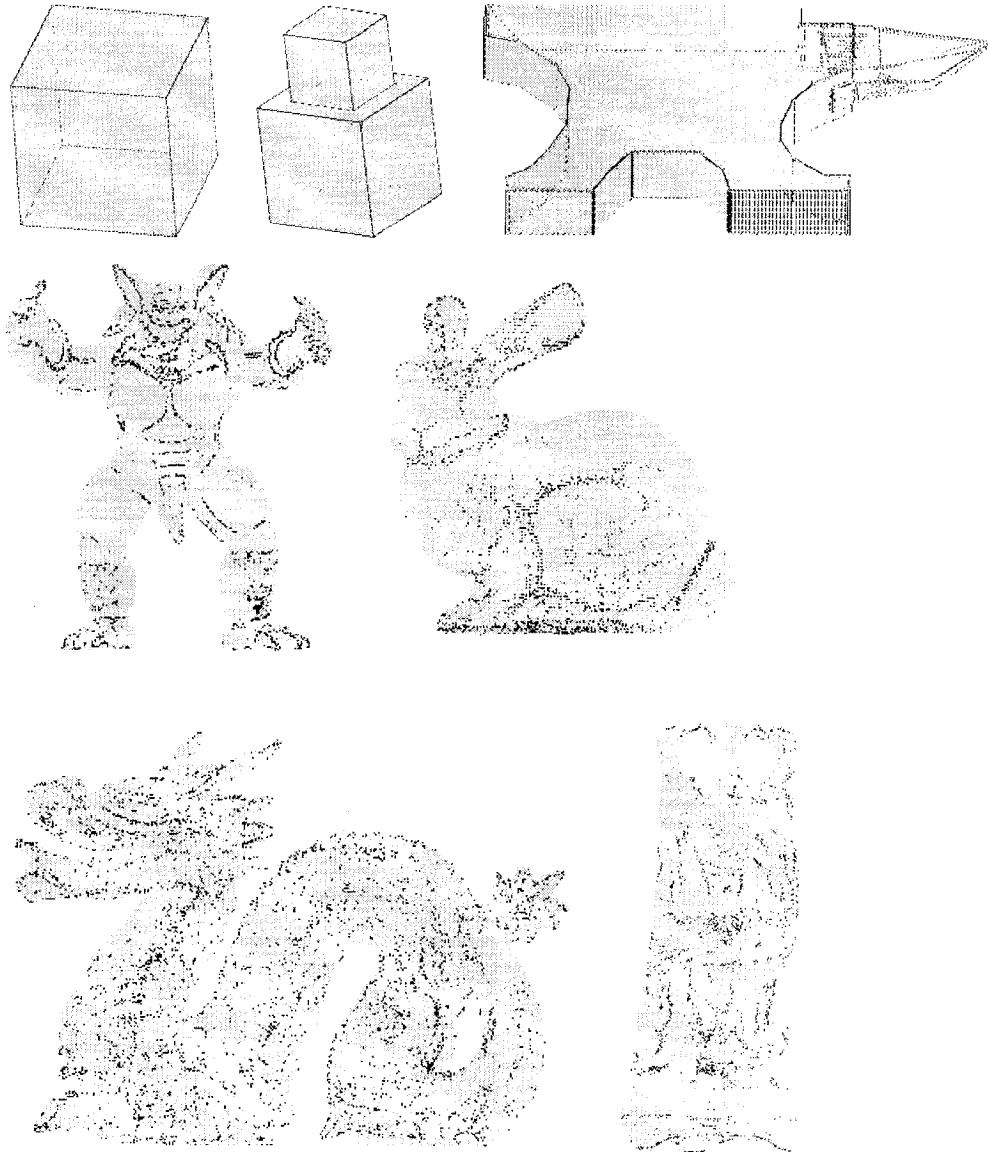


Figure 6-6: The refined classification information of each model.

Figure 6-6 shows the result of the classification refining for a) Single Box b) Dual Boxes c) Anvil d) Armadillo e) Bunny f) Dragon using NGP 5 g) Happy Budda using NGP 7. The red points are the corner points; and the blue points are crease points; and the green points are border points; and the yellow points are the flat points in figure 6-6.

The lines of the refined classification models become very slim comparing to the lines of the initial classification models.

6.2.7 Edge Traversing Process

Model	NGP Size	Number of Edges	AVG Number of Vertices	20>Edge >= 10 Vertices	Edge >=20 Vertices
Single Box	3	12	101	0	12
Dual Boxes	3	36	52	0	22
Anvil	5	175	17	31	36
Armadillo	3	290	8	64	13
Bunny	3	49	7	10	2
Dragon	3	886	7	156	19
Dragon	5	721	11	250	81
Happy Budda	3	835	6	93	6
Happy Budda	5	779	10	236	82
Happy Budda	7	632	16	256	164

Table 6-8: The edge data after traversing.

The edges are much shorter in the complicated models than the simple models in Table 6-8. The simple models like the engineer models have clear crease/corner attributes, so they can get good traversing edges. On the contrary, the crease/corner attributes of the complex models is very hard to be detected so it causes many crease vertices may be identified as the corner vertices, and then the long edges are broken by this kind of corner vertices.

6.2.8 Edge Refining

Model	NGP Size	Number of Edges	
		Before	After
Single Box	3	12	12
Dual Boxes	3	36	38
Anvil	5	175	335
Armadillo	3	290	47
Bunny	3	49	8
Dragon	3	886	227
Dragon	5	721	445
Happy Budda	3	835	168
Happy Budda	5	779	343
Happy Budda	7	632	823

Table 6-9: The comparison between the traversing edges and the refined edges.

The Table 6-9 shows the result of comparison of traversing edges and refined edges. The edge refining process increases the number of edges for the intersecting edges and the corner merging, but it also reduces the number of edges that are not in any edge loops.

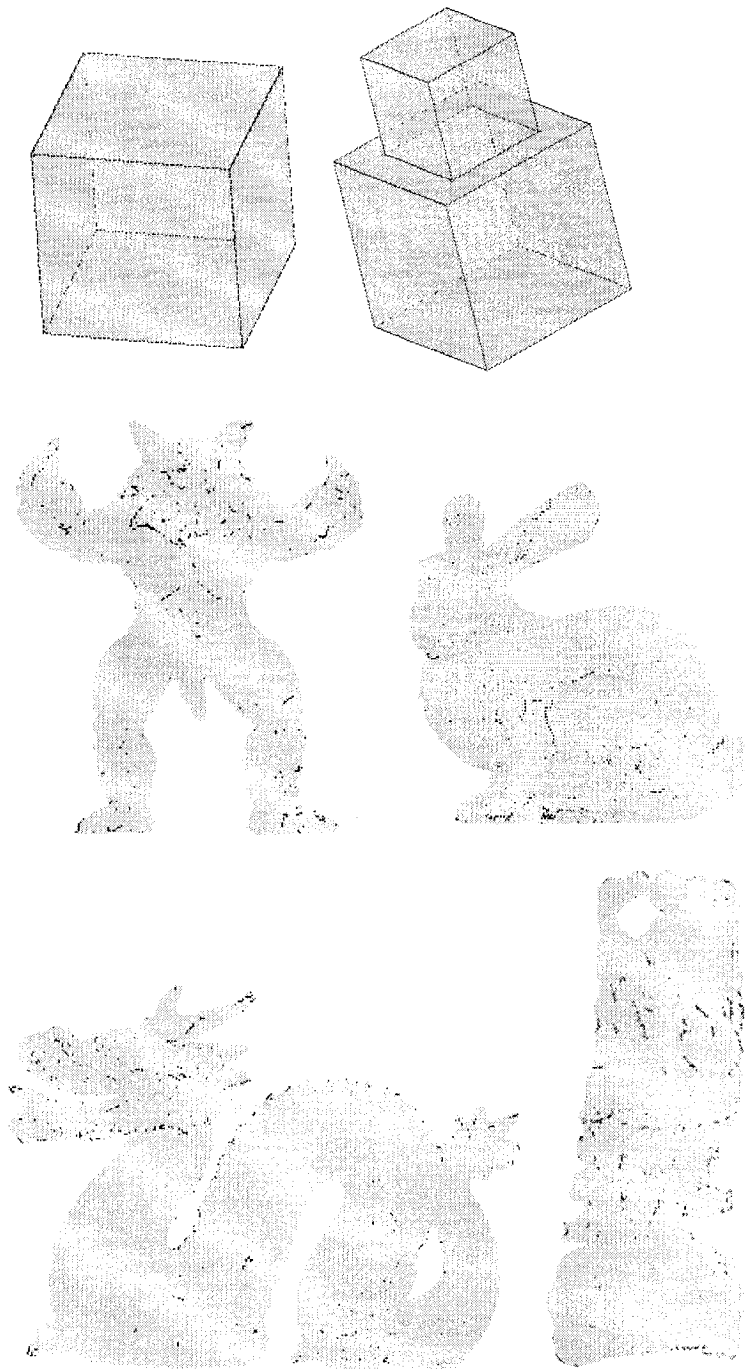


Figure 6-7: The result of the edge refining.

Figure 6-7 shows the result of the refined edges for a) Single box, b) Dual boxes, c) Armadillo d) Bunny, e) Dragon f) Happy Budda. The blue points are the edge end

points; and the blue points are edge points; and the yellow points are the internal points of the surfaces in figure 6-7.

6.2.9 Face Splitting Process

Model	NGP Size	Faces	Loop Edges
Single Box	3	6	6
Dual Boxes	3	11	12

Table 6-10: The face recovering information.

The result after face splitting process:

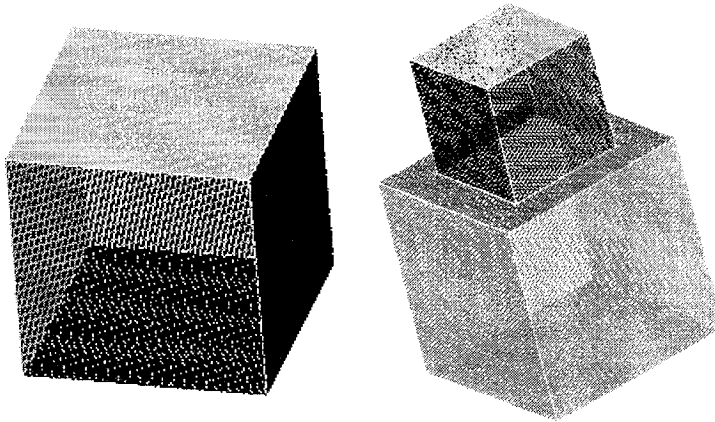


Figure 6-8: All faces are recovered.

Figure 6-8 shows the result of the face recovery for a) single box b) dual box. The yellow points are the edge points of the edge loop; and the other color points are surface internal points; and black points are unknown-type points. Fortunately, there are no black points appearing in the face recovering process.

6.2.10 Time consuming Process

Model	NGP Size	Total Vertices	Eigen Analysis	Classify Process	Edge Traverse	Edge Analysis	Face Recover
Single Box	3	60002	8.92s	0.96	0.21	0.46	4.86
Dual Boxes	3	74402	9.00s	1.48	0.37	0.93	6.47
Anvil	5	162882	34.50	3.61	2.06	98.34	X
Armadillo	3	172974	24.74	1.86	0.72	2.40	X
Bunny	3	35286	7.31s	0.57	0.18	0.14	X
Dragon	3	566098	42.0s	6.70	3.32	30.2	X
Dragon	5	566098	82.71s	11.90	5.48	63.54	X
Happy Budda	3	727735	74.91s	7.31	3.07	13.56	X
Happy Budda	5	727735	97.12s	15.35	5.54	229.34	X
Happy Budda	7	727735	174.77	28.18	10.64	230.79	X

Table 6-11: The time costs of each step.

Table 6-11 shows the time costs of eigen calculation, classifying process, edge traversing process, edge analysis process and face recovering process. For the natural models, they have a lot of short edge loops and these loops form small faces. The program could not complete the small face splitting.

7 Conclusions and Potential for Further Work

In the last 6 chapters of this thesis we have discussed the emerging importance of point sampled surfaces in 3D graphics. We presented the current state of the art in processing these models and then described new techniques that we have developed for processing these models. These techniques are for efficient rendering of these models and for reverse engineering these models to recover their 3D boundary representation in the B-Rep format. The problem is a very difficult one due to a number of problems in the point sampled data – large volume, noise, sampling density variation. Below we discuss our successes and also the places where our techniques need more work before they can be made to work correctly.

Rendering technique

The rendering technique works well when the data is dense and uniformly sampled. For sparse data, we have to resort to splatting and our image quality may suffer. The salient features of this method are the following:

- Unlike many of the earlier techniques our method does not require a simplified subset to be pre-computed.
- Instead it selects a smaller subset on an as needed basis using multiple visual cues. Each time the viewing conditions change, a new subset is selected and rendered. A significant advantage is that the original point set is always available. In an extreme zoomed in situation, all the sample points within the visible region may be selected and rendered.

- At every sample point that is rendered, a correctly oriented normal is needed. For this, most other algorithms depend on having access to the underlying continuous surface either in the form of a polygon mesh or piecewise algebraic geometry representations such as quadric or spline surface patches. Our approach makes a significant departure from this. We do not need any underlying continuous surface representation. We also do not require that the normal orientation be computed at every sample point of the original set. We have described a method, which associates a representative normal with each flattish region, and a method of correctly orienting this representative normal. Using this representative normal for the region, the correctly oriented normal at any point in that region can be computed. For a 2-manifold surface this method will give correct results as long as the surface has been adequately sampled. In an irregularly sampled surface, there could be regions, where this may not give us the correct approach. We give an example. The surface is such that it almost folds into itself and touches itself; the touching point is nearer to this point than other points that are topologically nearer to this point. As a result when basing our decisions only on spatial proximity of the points we may associate an incorrect orientation for the normal at one of these touching places. In such a situation, knowledge of the underlying surface connectivity is essential. However, this problem is not specific to our approach. Any approach that has to determine the underlying surface connectivity – say triangulating the sample points or fitting a surface would also need this knowledge to be externally supplied to it. Otherwise the underlying surface could be created with inaccurate topological connectivity.

While the overall results seem quite good, there are a number of aspects that we would be considering for further improvement. We briefly discuss these below.

- Presently we use simple heuristics to determine the number of samples that represent a region. An adaptive approach to determine the sample size must be investigated, one in which the error is minimized.
- Since we use the oct-tree nodes our sampling is more of a stratified nature. Importance sampling, associating importance to different subsets of the original sample is another approach that may help considerably improve yield better results.
- Presently we traverse all the leaf nodes of the oct-tree and determine the sample points to render. This is single resolution rendering. The hierarchical structure already present should help to devise a multi-resolution rendering algorithm.
- We can also investigate the development of an out of core rendering technique with the oct-tree structure maintained in persistent storage, and neighboring nodes loaded into main memory on an as needed basis.
- Since our method is probabilistic, it is important to estimate the error in the rendered image. This would require clearly defining a metric for measuring error in rendered images.

Reverse Engineering

Specifically our approach works very well on the engineering sampled data model as box and anvil in Table 1-1. It needs clear separation of facial regions. In engineering objects we usually have sharp edges and relatively large face regions. For natural or

sculptured surfaces, the edges are curved and often small curved patches blend into neighboring surface patches. Hence our methods will need major refinements to work for such models as well.

There are some characteristics of our implementation which are worth mentioning here.

7.1 *Implementation Issues*

- This is very large implementation, and it includes fairly complex algorithms.
- For reasons of efficiency and correctness, we had to consider a large number of exceptional cases, making the program unwieldy and even more complex.
- There are many recursive functions in this program; the stack size is raised from 2MB to 32MB.
- Many complex data structures like Hash tables, FIFOs etc. are used in this program.
- Double/long base types are used in order to get more accurate calculations in the process.
- Large memory is required; we have to keep the eigen value/vectors information, and other status for more than 100,000 points.
- There is no easy way to debug it for two reasons. One reason is that these models have huge data to be processed. Second reason is that the environment of the running program is huge, especially, in the process of the edge traversing/edge analysis recursively.

7.2 Robustness Issues

- The entire aspect of detecting edge features or segmenting points into faces in 3D is relatively new and all its complexities are not yet well understood.
- As already mentioned, classification/face splitting process at the merging point works well for simple objects, but for multiple surfaces meeting at a corner vertex, this process is not as stable as one would like it to be.
- The noise in the data can significantly affect the performance of the algorithms. We have tried our best to ensure that our algorithms work in spite of noise in the data, however we have not used formal signal processing techniques of filtering out noise etc. It is certainly worthwhile to try out the effectiveness of such techniques.
- Some calculations can only be done approximately, not accurately, for example, we say two 3D segments intersecting on the surface, but they do not have any intersect point in 3D. This causes a lot of exceptions in the process of the edge analysis and face split process.

7.3 Potential Extensions

- Flexible Neighborhood Collection Method

I have used FNCM in the seed fill process. Once the seed touches the boundary, the size of the neighborhood will reduce a portion, for example, 1/10; and the size of the neighborhood will increase a portion if the seed does not touch the boundary in a number of times.
- For extension to natural and sculptured objects, edge processing needs further case based analysis.

For the natural model, the surface merging is smooth, but they always have more merging places than the engineering models, which causes a lot of short edges. The edge analysis(intersecting/splitting/corner merging) is much more complicated if there are more short edges.

- The correctness of the 3D segments intersection algorithm needs to be verified further.

3D point sampled surfaces involve dealing with very large point data sets. Due to the absence of any underlying structure, devising computational processes that work at all scales and in a generic manner is both challenging and exciting. Our research objective was to investigate the use of features in devising efficient and correct algorithms for processing such models. Towards this objective we devised algorithms for efficient rendering and for recovering the B-Rep representation of large point cloud models. We have implemented these algorithms and tested them for a wide range of point sampled data sets covering simple objects to very complex objects and also engineering objects to natural or sculptured objects. Clearly our reverse engineering techniques are limited to work for engineering class of objects. This is because we have yet to devise robust feature detection algorithms for this class of models. Nevertheless, we do believe that our work has convincingly shown that feature based techniques can be devised for processing large point cloud models and the benefits that could be gained therefrom.

8 References:

- [1] Nina Amenta, Marshall Bern, Manolis Kamvysselis. **A New Voronoi-Based Surface Reconstruction Algorithm**, Computer Graphics (Siggraph), pages 415-421, 1998
- [2] Nina Amenta and Marshall Bern. **Surface Reconstruction by Voronoi Filtering**, Discr. Computational Geometry, 22, pages 481-504, 1999.
- [3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. **Point Set Surfaces**, Proc. IEEE Visualization Conf., pages 21-28, 2001.
- [4] Sushil Bhakar, Liang Luo, and S.P. Mudur, **View Dependent Stochastic Sampling for Efficient Rendering of Point Sampled Surfaces**, Proc. WSCG 2004, Prague.
- [5] P. Benkő, L. Andor, G. K'os, R. R. Martin and T. Várady, "**Constrained fitting in reverse engineering**", submitted to Computer Aided Geometric Design, 2000
- [6] P. BENKŐ, T. VÁRADY: **Direct Segmentation of Smooth Multiple Point Regions**, Proc. GMP 2002, Eds: H. Suzuki, R. Martin, IEEE 2002, pp 169-178
- [7] Pál Benkő, Ralph R. Martin (*), Tamás Várady. **Algorithms for Reverse Engineering Boundary Representation Models**, Computer Aided Design, Vol 33, No 11, 2001, pp 839-851.
- [8] D. Brodsky, B. Watson, **Model simplification through refinement**, Proc. of Graphics Interface 2000, 2000
- [9] Oli Cooper, Neill Campbell and David Gibson. **Automated Meshing of Sparse 3D Point Clouds**. In: ACM SIGGRAPH: Conference Abstracts and Applications. San Diego, USA, ACM, July 2003.
- [10] P. Cignoni, C. Rocchini, , R Scopigno, **Metro: Measuring error on simplified surfaces**, Computer Graphics Forum, 17(2), 1998, pp 167-174
- [11] David Eberly, **Least Squares Fitting of Data**, Magic Software, Inc.
<http://www.magic-software.com>
- [12] J.T. Feddema, C.Q. Little; **Rapid world modeling: fitting range data to geometric primitives**. Robotics and Automation, 1997. Proceedings., 1997 IEEE

International Conference on , Volume: 4 , 20-25 April 1997 Pages:2807 - 2812
vol.4

- [13] Michael S.Floater, Matrin Reimers. **Meshless parameterization and surface reconstruction**. Comp. Aided Geom. Design 18, 2001
- [14] Stefan Gumhold, XinlongWang, Rob MacLeod. **Feature Extraction from Point Clouds**, Proceedings of the 10 th International Meshing Roundtable, Sandia National Laboratories, pp.293-305, 2001.
- [15] M. Gopi, S. Krishnan, C.T. Silva, **Surface Reconstruction based on Lower Dimensional Localized Delaunay Triangulation**, Computer Graphics Forum (Eurographics 2000).
- [16] M. Gross, **Graphics in Medicine: From Visualization to Surgery**, ACM Computer Graphics, Vol. 32, 1998, pp 53-56
- [17] J. Giessen, M. John, **Surface reconstruction based on a dynamical system**, *Proc. EUROGRAPHICS '02*, 2002
- [18] M. Garland, P. Heckbert, **Surface simplification using quadric error metrics**, *Proc. SIGGRAPH 97*, 1997
- [19] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. **Surface Reconstruction from Unorganized Points**, Computer Graphics (Siggraph), 26, pages 19-26, 1992.
- [20] H. Hoppe, **Progressive Meshes**, SIGGRAPH 96, 1996
- [21] A. Hubeli, M. Gross, **Fairing of Non-Manifolds for Visualization**, *Proc. IEEE Visualization 00*, 2000.
- [22] A. Hubeli, M.Gross, **Multiresolution Feature Extraction from Unstructured Meshes**, *Proc. IEEE Visusualization 01*, 2001
- [23] I. Jolliffe, **Principle Component Analysis**, Springer-Verlag, 1986
- [24] A. Kalaih and A. Varshney, **Modelling and Rendering Points with Local geometry**, *IEEE Trans. On Visualization and Computer Graphics*, 2002, pp 101-129.
- [25] Lutz Kettner. **Using generic programming for designing a data structure for polyhedral surfaces**. *Computational Geometry: Theory and Applications*, 13:65-90, 1999.

- [26] D. Levin. ***Mesh-Independent Surface Interpolation***, To appear in Geometric Modeling for Scientific Visualization. Edited by Brunnett, Hamann and Mueller, Springer-Verlag, 2003.
- [27] M. Levoy *et al.* **The Digital Michelangelo Project: 3D Scanning of Large Statues**, *Proc. SIGGRAPH 2000*
- [28] M. Levoy and T. Whitted. **The use of points as a display primitive**. In Technical Report 85-022, Computer Science Department, UNC, Chapel Hill, January 1985.
- [29] T. Lindeberg, **Feature Detection with Automatic Scale Selection**, Int. Journal of Computer Vision, vol. 30, no. 2, pp 77-116, 1998
- [30] M. Mantyla (1988) **An introduction into solid modelling**. Computer Science Press, ISBN 0-88175-108-1
- [31] Niloy J. Mitra, An Nguyen, **Estimating surface normals in noisy point cloud data**, Proceedings of the nineteenth conference on Computational geometry, June 08-10, 2003, San Diego, California, USA
- [32] Boris Mederos, Luiz Velho, and Luiz Henrique de Figueiredo. **Moving Least Squares Multiresolution Surface Approximation**, Technical Report TR-0303, IMPA, 2003.
- [33] Carsten Moenning, Neil A. Dodgson, **A new point cloud simplification algorithm**, The 3rd IASTED conference for Visualization, Imaging and Image Processing (VIIP 2003)
- [34] Mark Pauly, Markus Gross, **Efficient Simplification of Point-Sampled Surfaces**, IEEE Visualization 2002
- [35] Mark Pauly, Richard Keiser, Markus Gross **Multi-scale Feature Extraction on Point-sampled Surfaces (2003)** Computer Graphics Forum Volume22, Issue 3 (September 2003)
- [36] Mark Pauly, Richard Keiser, Leif P. Kobbelt, Markus Gross, **Shape Modeling with Point-Sampled Geometry** SIGGRAPH 2003 Proceedings, 641 – 650
- [37] M. Pauly, L. Kobbelt and M. Gross, **Multiresolution Modeling of Point-sampled Geometry**. Technical Report 378, ETH Zürich, Scientific Computing, August, 2002.

- [38] M. Pauly, M. Gross, **Spectral Processing of Point-Sampled Geometry**, Proc. SIGGRAPH 01, 2001
- [39] R. Peikert, M. Roth, **The Parallel Vectors Operator - A Vector Field Visualization Primitive**, IEEE Visualization '99.
- [40] H. Pfister, M. Zwicker, J. van Baar, M. Gross, **Surfels: Surface Elements as Rendering Primitives**, SIGGRAPH 2000.
- [41] G. RENNER, L. SZOBONYA: **Reconstructing surfaces from scanned data**, ICCVG, Int. Conf. on Computer Vision and Graphics, Zakopane, Poland, 2002, pp 610-616
- [42] Gerhard Roth, Eko Wibowo, **A Fast Algorithm for Making Mesh-Models from Multiple-View Range Data**, Proceedings of the Robotics and Knowledge Based Systems Workshop, 1995, pp. 349-355.
- [43] S. Rusinkiewicz, M. Levoy, **QSplat: A Multiresolution Point Rendering System for Large Meshes**, SIGGRAPH, 2000.
- [44] J. Revelles, C. Ureña, M. Lastra, **An Efficient Parametric Algorithm for Octree Traversal**, Journal of WSCG, vol 8, no. 2, pp. 212-219, ISSN 1213-6972
- [45] Ioannis Stamos and Peter K. Allen. **3-D Model Construction Using Range and Image Data**, Proceedings of Computer Vision and Pattern Recognition, June 2000, 1:531-536.
- [46] P. Savadjiev, FP. Ferrie , and K. Siddiqi, **Surface recovery from 3D point data using a combined parametric and geometric flow approach** , ENERGY MINIMIZATION METHODS IN COMPUTER VISION AND PATTERN RECOGNITION, PROCEEDINGS , LECTURE NOTES IN COMPUTER SCIENCE , vol. 2683 , pp. 325 -340 , 2003.
- [47] E. Shaffer, , M. Garland, **Efficient Adaptive Simplification of Massive Meshes**, *Proc. IEEE Visualization 01*, 2001
- [48] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, J. Snyder, **Silhouette Clipping**, Proc. SIGGRAPH 2000, pp. 327-334.
- [49] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion, **Conservative Volumetric Visibility with Occluder Fusion**, Proc. SIGGRAPH 2000.

- [50] Gerard L.G. Sleijpen, Henk A. Van der Vorst, **A Jacobi-Davidson Iteration Method for Linear Eigenvalue Problems**, SIAM Journal on Matrix Analysis and Applications 17(2):401-425, 1996
- [51] G. Taubin, **A Signal Processing Approach to Fair Surface Design**, *Proc. SIGGRAPH 95*, 1995
- [52] Tamás Várady, Pál Benkő. **Reverse Engineering B-rep models from Multiple Point Clouds**, Geometric Modelling and Processing 2000, IEEE Computer Society, 2000, pp 3-12.
- [53] G. Varadhan, D. Manocha, **Out-of-Core Rendering of Massive Geometric Environments**, *Proc. IEEE Visualization 2002*
- [54] Jan Weingarten, Gabriel Gruener, Roland Siegwart, **A Fast and Robust 3D Feature Extraction Algorithm for Structured Environment Reconstruction**, ICAR 2003
- [55] A. Witkin, P. Heckbert, **Using Particles To Sample and Control Implicit Surfaces**, *Proc. SIGGRAPH 94*, 1994
- [56] M. Zwicker, H. Pfister, J. van Baar, M. Gross, **Surface Splatting**, *Proc. SIGGRAPH 01*, 2001