ROD-TV: SURFACE RECONSTRUCTION ON DEMAND
BY TENSOR VOTING

By

NG HO LUN

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science

June 2003, Hong Kong
Authorization Page

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

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

[Signature]

NG HO LUN
ROD-TV: SURFACE RECONSTRUCTION ON DEMAND
BY TENSOR VOTING

By

NG HO LUN

This is to certify that I have examined the above MPhil thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Dr. CHI-KEUNG TANG, Supervisor

Prof. LIONEL NI, Head of Department

Computer Science
13 June 2003
To my parents and sisters
Acknowledgments

I want to express my sincere appreciation to my advisor Dr. Chi-Keung Tang for his unfailing guidance and support during this research. After working under him for a year, I still find it hard to believe that he is so knowledgeable in the field of Computer Graphics and Vision. I have been fortunate enough to gain invaluable advice from him throughout my MPhil studies; it was truly an once-in-a-lifetime experience to receive his support and encouragement. No advisor could have been better than Dr. Chi-Keung Tang.

I also wish to thank Dr. Sheung-Han Gary Chan and Dr. Chiew-Lan Tai for using up their valuable time to serve on my defense committee as well as making important comments about this thesis. Special thanks to Dr. Sheung-Han Gary Chan for being my supervisor in my final year project and sharing with me his valuable experience during my postgraduate study.

Furthermore, I would like to thank Oscar Au, who gave me lots of ideas and explanations about computer graphics; Dickson Tong, who gave me a brief introduction to tensor voting formalism; Venus To, who encouraged me while we were writing our thesis; and the support of my fellow MPhil classmates Herman Chan, Jason Chan, Tommy Lam, Terence Li, Ivor Tsang, and David Wong.

Last but not least, I would like to thank my family for their support. Especially my mother, who has always prepared a bowl of soup for me when I go back home from the hostel each week throughout my postgraduate study. Without them, this thesis would never have come to fruition. Once again, thank you everybody.

Stanley Ng

The Hong Kong University of Science and Technology

June 2003
Table of Contents

AUTHORIZATION PAGE .......................................................... II

ACKNOWLEDGMENTS ............................................................. V

TABLE OF CONTENTS ............................................................ VI

LIST OF FIGURES .................................................................. X

LIST OF TABLES ....................................................................... XIV

LIST OF EQUATIONS .............................................................. XV

LIST OF ALGORITHMS ............................................................ XVI

ABSTRACT ........................................................................ XVII

CHAPTER 1 INTRODUCTION ...................................................... 1

1.1 Problem in Rendering Scanned 3-D Dataset ......................... 1
1.2 On Demand Reconstruction ............................................... 1
1.3 Motivation ..................................................................... 3
  1.3.1 Reconstruction and Rendering .................................... 3
  1.3.2 Robustness and Efficiency ....................................... 6
1.4 Scope ........................................................................... 14

CHAPTER 2 RELATED WORK .................................................. 15

2.1 Algorithm Grouping ......................................................... 15
2.2 Algorithm Description .................................................... 16
CHAPTER 3 REVIEW OF BASIC TENSOR VOTING FORMALISM ............ 22

3.1 Tensor Voting System ........................................ 22
   3.1.1 Introduction .................................... 22
   3.1.2 Stage One – Information Encoding ............... 24
   3.1.3 Stage Two – Sparse Tensor Voting ............... 25
   3.1.4 Stage Three – Dense Tensor Voting ............... 26
   3.1.5 Stage Four – Features Extraction ............... 26

3.2 Information Representation .................................. 27
   3.2.1 Introduction .................................... 27
   3.2.2 Second Order Symmetric Tensor .................. 28
   3.2.3 Tensor Decomposition ............................ 32

3.3 Tensor Communication ....................................... 33
   3.3.1 Introduction .................................... 33
   3.3.2 Token Refinement and Dense Extrapolation ....... 33
   3.3.3 The Fundamental 2-D Stick Kernel ................ 35
   3.3.4 Derivation of the Stick, Plate and Ball Voting
        Field ............................................. 37

3.4 Features Extraction ......................................... 39
   3.4.1 Introduction .................................... 39
   3.4.2 Maximal Surfaces in 3-D ......................... 40
   3.4.3 Maximal Curves in 3-D .......................... 42
   3.4.4 Maximal Junctions in 3-D ....................... 43
   3.4.5 Maximal Features Extraction ..................... 43

3.5 Tensor Voting Operations .................................. 43

Summary ......................................................... 47

CHAPTER 4 DATA STRUCTURE FOR ROD-TV .................. 48

4.1 Introduction ................................................ 48

4.2 Spatial Data Structures .................................. 48
   4.2.1 Overview ...................................... 48
   4.2.2 Spatial Subdivision and Object Subdivision .... 50
4.3 Rod-TV Data Structure 1: Grid Pyramid .................. 53
  4.3.1 Introduction ........................................ 53
  4.3.2 Construction of Grid Pyramid ..................... 54
  4.3.3 Grid Pyramid Operations: Token Smoothing and
        Cell Quantization .................................. 58
  4.3.4 Pyramid Grid Benefits and Limitations .......... 58

4.4 Rod-TV Data Structure 2: Octree ......................... 59
  4.4.1 Introduction ........................................ 59
  4.4.2 Construction of Octree ............................ 60
  4.4.3 Octree Benefits and Limitations .................. 61

4.5 Experimental Results .................................... 64

4.6 Summary ................................................. 70

CHAPTER 5 VISIBILITY ALGORITHMS FOR ROD-TV ............. 71

5.1 Introduction ............................................ 71

5.2 Rod-TV Visibility Algorithm I: View Frustum Culling 72
  5.2.1 Viewing Frustum in a 3-D scene .................... 72
  5.2.2 Plane-Token Intersection .......................... 73
  5.2.3 Frustum-Token Intersection ....................... 77
  5.2.4 Frustum-(Bounding Volume) Intersection .......... 78
  5.2.5 Hierarchical View Frustum Culling .............. 80

5.3 Rod-TV Visibility Algorithm II: Occlusion Culling .... 82
  5.3.1 Image Space Algorithm ............................. 83
  5.3.2 Occlusion Culling: Construction of Occlusion
        Map and Depth Map ................................ 84
  5.3.3 Occlusion Culling: Depth Test ..................... 85
  5.3.4 Problem in Occlusion Culling ..................... 85
  5.3.5 Contribution Threshold ............................ 86

5.4 Experimental Results .................................... 89

5.5 Summary ................................................. 96

CHAPTER 6 LEVELS-OF-DETAIL RECONSTRUCTION FOR ROD-TV .... 97

viii
List of Figures

Figure 1.1 - Reconstruction and Rendering of ROD-Tv ......................... 5
Figure 1.2 - Noise Generation in LUNG Dataset .................................... 9
Figure 1.3 - Robustness of ROD-Tv [part 1] ........................................ 10
Figure 1.4 - Robustness of ROD-Tv [part 2] ........................................ 10
Figure 1.5 - Robustness of ROD-Tv [part 3] ........................................ 11
Figure 1.6 - Robustness of ROD-Tv [part 4] ........................................ 11
Figure 1.7 - Robustness of Tensor Voting ........................................... 12
Figure 1.8 - Robustness of ROD-Tv [part 5] ........................................ 12
Figure 1.9 - Efficiency of ROD-Tv [zoom-in part 1] ............................... 13
Figure 1.10 - Efficiency of ROD-Tv [zoom-in part 2] ............................. 13
Figure 1.11 - Efficiency of ROD-Tv [zoom-out] .................................... 14
Figure 2.1 - Algorithm grouping in both graphics and visions .................. 15
Figure 3.1 - Overview of the essential components of tensor voting .......... 23
Figure 3.2 - Tensor Voting Stage One - Information Encoding .................. 24
Figure 3.3 - Tensor Voting Stage Two - Sparse Tensor Voting ................. 25
Figure 3.4 - Tensor Voting Stage Three - Dense Tensor Voting ............... 26
Figure 3.5 - Tensor Voting Stage Four - Features Extraction ................. 27
Figure 3.6 - Geometrically describe second order symmetric tensor in 3-D space ......................................................................................... 30
Figure 3.7 - Surface patches elements (SURFEL) ................................ 30
Figure 3.8 - Curve Segment Element (CURVEL) ................................... 31
Figure 3.9 - Point case .................................................................32
Figure 3.10 - A generic second order symmetric tensor ..................34
Figure 3.11 - The Design of Fundamental 2-D Stick Kernel ..........36
Figure 3.12 - Discrete version on the 3-D stick voting field ..........38
Figure 3.13 - Discrete version on the 3-D plate voting field ..........38
Figure 3.14 - Discrete version on the 3-D ball voting field ..........39
Figure 4.1 - Hierarchical Bounding Sphere and Hierarchical Octree ..50
Figure 4.2 - The construction of Hierarchical Bounding Sphere ....52
Figure 4.3 - Grid Pyramid in 3-D space ......................................53
Figure 4.4 - Grid Pyramid in 2-D space ......................................54
Figure 4.5 - The construction of Grid Pyramid in bottom-up fashion ..56
Figure 4.6 - Grid Pyramid operations: smoothing operator and
            quantization operator ...................................................57
Figure 4.7 - Octree in 3-D space ..............................................59
Figure 4.8 - Quadtree in 2-D space ..........................................61
Figure 4.9 - The construction of Octree in top-down fashion .........62
Figure 4.10 - A subdivision problem in Octree construction ........64
Figure 4.11 - Data multi-resolution on the PregnantWoman dataset ..66
Figure 4.12 - A comparison between Grid Pyramid and Octree on the
            PREGNANTWOMAN dataset ............................................67
Figure 4.13 - Data multi-resolution on the MONKEYSADDLE dataset ..68
Figure 4.14 - A comparison between Grid Pyramid and Octree on the
            MONKEYSADDLE dataset ..............................................69
Figure 6.8 - Surface Patches Extraction and Surface Connectivity ...... 112
Figure 6.9 - ROD-Tv with -connectivity and 1-connectivity ............... 113
Figure 6.10 - Shading in ROD-Tv ............................................. 114
Figure 6.11 - Experimental result on the surface connectivity of PLANE dataset ......................................................................... 115
Figure 6.12 - Experimental result on HAND dataset (part 1) .............. 116
Figure 6.13 - Experimental result on HAND dataset (part 2) ............... 117
Figure 6.14 - Experimental result on HAND dataset (part 3) ............... 118
Figure 6.15 - Experimental result on HAND dataset (part 4) ............... 118
Figure 6.16 - Experimental result on HAND dataset (part 5) ............... 119
Figure 6.17 - Experimental result on HAND dataset (part 6) ............... 119
Figure 6.18 - Experimental result on comparison between QSplat point rendering and ROD-Tv in HAND dataset ........................................ 120
List of Tables

Table 3.1 - Input groups to the Tensor Voting ........................................24
Table 3.2 - Feature saliency and orientation in a second order symmetric
tensor .................................................................................................33
Table 3.3 - Feature saliency and orientation in a tensor map ..............40
Table 4.1 - Comparison between Bounding Volume Hierarchy (BVHs) and
Spatial Volume Hierarchy (SVHs) .........................................................51
Table 4.2 - Comparison on the number of cells between Grid Pyramid and
Octree ....................................................................................................63
Table 4.3 - Experimental results on the PREGNANTWOMAN dataset ....67
Table 4.4 - Experimental results on the MONKEYSADDLE dataset ....69
Table 5.1 - Three Difference Cases in Plane-Token Intersection Test.....76
Table 5.2 - Comparisons between the Numbers of Tokens Survival after
Frustum Culling ..................................................................................89
Table 5.3 - Comparisons between the Numbers of Tokens Survival after
Occlusion Culling ................................................................................89
Table 6.1 - Comparison of ROD-TV in two different views ...............120
List of Equations

Equation 3.1 - Representing second order symmetric tensor by eigensystem in matrix form ......................................................... 29

Equation 3.2 - Representing second order symmetric tensor by eigensystem in polynomial form ......................................................... 29

Equation 3.3 - Representing second order symmetric tensor by stick tensor, plate tensor and ball tensor ......................................................... 32

Equation 3.4 - Energy decay function in the 2-D fundamental stick kernel ....................................................................................... 37

Equation 3.5 - Formula to generate plate voting field ......................................................... 38

Equation 3.6 - Formula to generate plate voting field ......................................................... 39

Equation 5.1 - Equation of Plane in General Form ......................................................... 74

Equation 5.2 - Equation of Plane in Hessian Normal Form ......................................................... 75

Equation 5.3 - Token-Plane Distance ............................................................................ 75

Equation 6.1 - The estimation of the radius of the projected sphere ...... 108
List of Algorithms

Algorithm 3.1 - Operation to generate vector vote between voter site and votee site ................................................................. 45

Algorithm 3.2 - Operation to generate tensor vote between voter site and votee site .................................................................. 46

Algorithm 3.3 - Operation to combine weighted vector vote into current tensor vote .................................................................. 46

Algorithm 4.1 - Constructing Hierarchical Bounding Sphere Algorithm ............................................................. 52

Algorithm 5.1 - Computing Distance between Token and Plane ..................................................................................... 76

Algorithm 5.2 - Frustum-Token Intersection ........................................................................................................... 78

Algorithm 5.3 - Hierarchical View Frustum Culling .................................................................................................. 82

Algorithm 5.4 - General Occlusion Culling Algorithm .................................................................................................. 88

Algorithm 6.1 - Hierarchical Traversal for ROD-Tv ........................................................................................................ 100

Algorithm 6.2 - General Hierarchical Traversal ........................................................................................................ 100

Algorithm 6.3 - Front-to-Back Hierarchical Traversal in 1-D Binary Tree .................................................................................. 102

Algorithm 6.4 - Hierarchical Traversal with Visibility Cull ................................................................................................ 104

Algorithm 6.5 - Hierarchical Traversal with Levels-of-Detail .......................................................................................... 106
ROD-TV: SURFACE RECONSTRUCTION ON DEMAND BY TENSOR VOTING

By

NG HO LUN

Computer Science Department
The Hong Kong University of Science and Technology

Abstract

In this thesis, a “graphics for vision” approach is proposed to tackle the problem of surface reconstruction from a large and imperfect data set: surface reconstruction on demand by tensor voting (ROD-Tv). ROD-Tv simultaneously delivers good efficiency and robustness by adapting to a continuum of primitive connectivity, view dependence, and levels of detail ( LOD). Locally inferred surface elements are robust to noise and better capture local shapes. By positioning and inferring per-vertex normals at sub-voxel precision on the fly, we can achieve interpolative shading to produce superior quality rendering. Since this missing information can be recovered at the present levels-of-detail, our result is not upper bounded by the scanning resolution.

ROD-Tv consists of a spatial hierarchical data structure that encodes various levels of detail. The local surface reconstruction algorithm is tensor voting. It is applied on demand to the visible subset of data at a desired levels-of-detail, by traversing the data hierarchy and collecting tensorial support in a neighborhood. We compare our approach and present encouraging results.
Chapter 1

Introduction

1.1 Problem in Rendering Scanned 3-D Dataset

High resolution laser scanners have become more popular due to advances in 3-D scanning technology [3DSCANNERS] [CYBERWARE]. Even with some of the most state-of-the-art range finders, the large volume of scanned data produced are seldom perfect: the range data are noisy due to measurement errors, so an approximation of the surface is created. Without addressing noise explicitly, pure point rendering approach without tactful handling may fail on erroneous input, or produce significant artifact when crucial data are missing. Robustness and efficiency are therefore important issues. A robust 3-D application should provide interactive speed and the appropriate LOD for efficient processing.

1.2 On Demand Reconstruction

To save processing time, view dependent reconstruction from a noisy dataset can be considered. That is, when a user visualizes a certain region of interest at high resolution, only the visible subset of data inside the viewing frustum needs to be processed. To visualize the whole data set, the processing should be done at an appropriate resolution. Otherwise, computation time may simply be wasted on rendering a triangle whose projected area is less than one pixel.

To achieve these requirements, we propose to defer vision reconstruction until view-dependent information is available at run-time. We call this on-the-fly “graphics for vision” approach ROD-TV, reconstruction on demand
by tensor voting. ROD-Tv uses view dependent and LOD control to drive
tensor voting to deliver robust and efficient reconstruction, given a large
and imperfect 3-D dataset. Therefore, ROD-Tv is most suitable for
applications that require a balance of quality reconstruction, rendering
and response latency.

Since this is a new approach, we use Figure 2.1 to position and compare
ROD-Tv with related work, which will be detailed in Chapter 2. ROD-Tv
adapts to a continuum of view dependence, levels of detail, and primitive
connectivity to perform reconstruction:

*View Dependent*

Representation or description can be viewer-centered or object-centered.
Depth maps [Curless 00] derived from stereo and motion are classical
viewer descriptions. Overlapping layers [Medioni 00] are regarded as
either an object centered or a view independent representation.

*Levels of detail*

MipMap [Williams 83] and Gaussian pyramids [Burt 83] are some
pioneering hierarchical representations.

*Primitive Connectivity*

Depending on the types of approaches taken and applications, output
ranges from a set of isolated points to a fully connected mesh. The
computational geometry approach [Hoppe 92] [Boissonnat 84] treats an
input point set as a connected graph to produce a surface mesh. It is
important to note that noisy input needs to be filtered first.

ROD-Tv contributes the following dimensions in computer graphics and
computer vision: it has led to a better rendering result through
reconstructing per-vertex normals, a robust and efficient reconstruction system which rejects noise and performs reconstruction on-the-fly.

1.3 Motivation

ROD-Tv is inspired by QSplat [Rusuinkiewicz 00] and its limitations. The inspiration is manifested into a ROD-Tv adaptation to a continuum of view dependence, levels-of-detail, and primitive connectivity for speeding up reconstruction. By making use of view dependence, ROD-Tv is a more efficient system with good noise robustness compared to the original tensor voting [Medioni 00]. Image resolution indicates the needed LOD, which drives how much detail ROD-Tv should reconstruct on-the-fly.

1.3.1 Reconstruction and Rendering

A point rendering system is adequate for noise-free input, in which fine details are usually viewed at scales near the scanning resolution [Rusuinkiewicz 00]. Can we go beyond this? QSplat and related point rendering techniques can be improved upon by plugging in ROD-Tv to reconstruct a more robust shading primitive on-the-fly.

As shown in Figure 1.1, for zoom-in viewing, our rendering result is better than those produced by simple point splatting. We are capable of reconstructing per-vertex normals efficiently using tensor voting. Interpolative shading in the screen space is therefore possible.

In QSplat, a single normal is kept (in a tree node), which is used to orient shading primitives. Though it can be argued that per-vertex normals can be obtained through a separate process, or by keeping more normals obtained from the input mesh if available, ROD-Tv is more attractive,
since such information can be inferred on-the-fly, in the presence of noise and in the absence of a connected mesh.

One of the motivations on this thesis is that our contribution, ROD-TV can produce a smooth result in any viewing resolution [Figure 1.1.G-H]. It is useful especially when a viewing resolution is greater than a scanning resolution. If point rendering is chosen, an artifact result is returned [Figure 1.1.A-F]. However, ROD-TV spends extra time on inferring missing normal information in between points. If the viewing resolution is less than or equal to the scanning resolution, then we prefer to use point rendering. The reason is that no extra time is needed to infer missing information and faster rendering result is returned. Since our goal is to get a smooth result in any viewing resolution, instead of point rendering, ROD-TV is used. In ROD-TV, an “on demand” visible point set is queried by using some graphical techniques, like levels-of-detail and visibility culling. Then, “on demand” surface is reconstructed from this point set by using tensor voting. The advantage of reconstructing “on demand” surface is that no extra CPU time is wasted on the invisible part. Also, this is a fast reconstruction compared to traditional view-independent surface reconstruction.
Figure 1.1 - Reconstruction and Rendering of ROD-Tv

Comparison of sample results produced by QSplat and ROD-Tv, rendered at a higher resolution than the scanning resolution. The shading primitives used in QSplat is (A) points, (B) quadrilaterals, (C) ellipses, (D) spheres, (E) circles, and (F) round points. Shading primitives produced by ROD-TV are shown in (G), where per-vertex normals are not reconstructed, and in (H), where per-vertex normals are reconstructed.
1.3.2 Robustness and Efficiency

The inadequacy of point rendering without reconstruction is illustrated by comparing our results to the noisy LUNG dataset, for which QSplat and point rendering techniques are not well suited for producing convincing results.

In our comparison, a noisy LUNG dataset is being used. This dataset consists of tomographic images. It is composed of 34 cross-sections. In each cross-section, contours of a lung at a specific depth are outlined. We manually sample the contour and finally a noise-free LUNG dataset with 3,305 tokens is produced.

The generation of a noisy dataset

Usually, it is hard to produce a noise-free dataset even when it is obtained from a laser scanner. In order to demonstrate robustness between point rendering techniques and our method, a noisy dataset is being used. To obtain a noisy dataset, a process of noise generation is performed as shown in Figure 1.2. Basically, this process consists of two components. The first component is the original noise-free dataset, which contains 3,035 tokens. The second component is a noise-token generation. In our comparison, we make use of the impulse noise model, where noise-tokens are randomly generated within a given range. For 25% of the dataset noise, there are 759 random tokens in total (3,035 * 0.25 = 759). Apart from the impulse noise model, the Gaussian noise model can be used to generate better noise distribution. Finally, the Noisy LUNG dataset is produced by combining the true data and noise together.

Point rendering without specific normal direction

In point rendering, if the normal direction is not specified in each token site, then a sphere is used as a rendering primitive. Figure 1.4 illustrates
point rendering in a sphere mode. To generate a hole-free surface, the spheres in each site should be big enough to touch their neighbors. The advantage of using a sphere as a rendering primitive is that no specific orientation alignment has been made. Rendering without alignment is always accelerated. Nevertheless, when we look closely at the surface, an artifact surface is returned [Figure 1.1.A-F]. In order to make a fair comparison, a quadrilateral rendering primitive is used instead of a sphere.

Point rendering with specific direction
If a quadrilateral is used, then we need to know the normal direction of each token site. We can achieve this normal information through a process of sparse tensor voting. Apart from the normal direction, surface saliency is also obtained after sparse tensor voting. To eliminate potential noise tokens, we can set a cutoff on the surface saliency, which indicates the likelihood of a point lying on a smooth surface. Figure 1.3 illustrates point rendering in a quadrilateral mode with a surface saliency threshold equal to 0. As you can see in the figure, quadrilaterals in each site are being aligned in their normal direction. Noise tokens exist as its surface saliency cutoff is too low.

Saliency threshold adjustment in point rendering
To eliminate unwanted noisy tokens, we need to set a higher threshold value. Two different threshold values are being tested and corresponding results are displayed in Figure 1.5 and Figure 1.6. Even though we can make use of normal saliency information, the point rendering result is still unsatisfactory. This is because the optimal threshold value is hard to obtain. If we select a threshold value lower than the optimal, then we cannot eliminate noise-tokens completely. On the other hand, if we select a threshold value higher than the optimal, then some true tokens are being removed and holds are produced on the surface. Hence, it is inefficient for handling noisy input in point rendering.
Noise robustness in tensor voting algorithm

In order to make a smooth and hole-free surface, we need a noise robust surface reconstruction algorithm. Hence, tensor voting is used and the results of reconstructing a noisy dataset are shown in Figure 1.7 and Figure 1.8. In both figures, whether noises are present or not in a dataset, tensor voting can fully utilize surface saliency information during reconstructing and finally a smooth and hole-free surface is returned.

Surface Reconstruction on demand through tensor voting – ROD-Tv

Even though tensor voting is a robust algorithm, surface reconstruction on the whole dataset is a costly operation, where some tokens are entirely invisible. To save processing time, reconstruction on a visible subset at the current levels-of-detail is proposed in this thesis. We call it “Surface Reconstruction on Demand by Tensor Voting” or ROD-Tv – The local surface reconstruction is tensor voting and a visible subset is obtained by doing visibility analysis on a hierarchical dataset.

Efficiency in ROD-Tv

The general idea of the ROD-Tv is that we only reconstruct surfaces on the visible dataset. For example, when we look at a zoom-in view of a dataset, some tokens are inside the viewing volume and some are not. To accelerate the whole reconstruction, only visible tokens are processed. Two different zoom-in views on the LUNG dataset are illustrated in Figure 1.9 and Figure 1.10. Clearly, only half of the dataset is being processed during reconstruction (Figure 1.9 – 1,823 tokens out of 3,035 tokens and Figure 1.10 – 1,805 tokens out of 3,035 tokens). Also, a high resolution surface is being output since the distance between the viewer and the dataset is comparatively small. However, when the viewer moves further away from the dataset, the surface reconstruction resolution will decrease accordingly, and therefore time is saved during reconstruction. Figure 1.11 illustrates a zoom-out view on the LUNG dataset. A coarse surface
can be found in a close-up view and its appearance does not change much when you observe it on a screen.

Our motivation in this thesis is using ROD-TV instead of point rendering to reconstruct a surface from a noisy dataset. ROD-TV is similar to the original tensor voting algorithm and it is noise robust. Further speed-up on the reconstruction is made by queuing a visible subset in a data hierarchy. That is, the coarse surface reconstruction is performed when the viewer is far away and the fine surface reconstruction is made when the viewer is close. Furthermore, interpolative shading can be inferred on-the-fly and result in a smooth, rendered surface. In the following section, the scope of this thesis is outlined.

![Diagram of Noise Generation in LUNG Dataset](image)

**Figure 1.2 - Noise Generation in LUNG Dataset**

A 25% LUNG dataset consists of two input components. The first component is the original noise-free LUNG dataset (3,035 tokens). The second component is the noise generation. In this case, we make use of the impulse noise model, where noises randomly occur within a given range. For a 25% noise dataset, there are 759 random tokens \((3,035 \times 0.25 = 759)\).
Figure 1.3 - Robustness of ROD-TV [part 1]

The original LUNG dataset (3,305 tokens) with 25% noise (759 tokens) is illustrated in the left. After processing sparse tensor voting, each token site gets the surface saliency information, which indicates the likelihood of a token lying on a smooth surface. The figure on the right shows a point rendering in quadrilateral mode without any surface saliency cut-off.

Figure 1.4 - Robustness of ROD-TV [part 2]

The above figure shows a point rendering using sphere as a rendering primitives on LUNG dataset with 25% noise. Since we don't know the direction of the site, sphere is used as a rendering primitive. In close-up view, we can visualize the artifact surface.
Figure 1.5 - Robustness of ROD-Tv [part 3]

The above figure illustrates a point rendering with quadrilateral as primitives. The surface saliency cutoff is set at 0.35. Compared to Figure 1.3, a large portion of noise is being removed. However, this saliency threshold is too low to reject all the noise point in point rendering. The surface normal of each token site can be visualized in a close up view.

Figure 1.6 - Robustness of ROD-Tv [part 4]

The above figure illustrates point rendering with a quadrilateral as primitives. The surface saliency cutoff is set at 0.53. Compared to Figure 1.3, all noise tokens are being removed. Compared to Figure 1.5, some true tokens are also being eliminated and we conclude that this saliency threshold is too high. Holds are produced during point rendering. The surface normal of each token site can be visualized close up.
Figure 1.7 - Robustness of Tensor Voting

We reconstruct a surface from a noisy/noise-free dataset with tensor voting. (A) A LUNG dataset with no noise (B) LUNG dataset with 25% noise (C) A LUNG dataset with 100% noise. Compared to the point rendering result, tensor voting is more noise robust.

Figure 1.8 - Robustness of ROD-Tv [part 5]

The original LUNG dataset (3,305 tokens) with 25% noise (759 tokens) is illustrated on the left. The figure on the right shows our result by means of the noise robust tensor voting algorithm. The surface reconstructed by tensor voting is much smoother than the point rendering one.
A surface reconstructed on the whole dataset is inefficient. To speed up the computation process, surface reconstruction on demand is proposed, so-called ROD-TV. In the figure, a zoom-in view of the LUNG dataset is shown, which only consists of 1,823 tokens (Full dataset is 3,305 tokens). The surface is reconstructed under high resolution.

This figure shows another zoom-in view of the LUNG dataset which consists of 1,805 tokens only. The high resolution surface can be found in the screen-space view.
Compared to Figure 1.8 and Figure 1.9, we move the viewing position further away the dataset. A low-resolution surface is then constructed as shown in the above figure. The whole surface consists of only 232 tokens. The low resolution surface can be visualized in the close-up view.

1.4 Scope

The goal of this thesis is to develop a robust and efficient surface reconstruction system of acceptable quality in rendering given noisy data. The next chapter surveys relevant prior work and tries to define a position for ROD-Tv. Chapter 3 gives a general overview on the original tensor voting system, which is used for a local surface reconstruction. Data structure alternatives are described in Chapter 4. A visibility analysis between the viewer and the input dataset can be found in Chapter 5. Chapter 6 presents the levels-of-detail algorithms for the surface reconstruction on demand. Finally, Chapter 7 summarizes this thesis, draws conclusions and points out some possible future extensions.
Chapter 2

Related Work

2.1 Algorithm Grouping

ROD-TV uses "graphics for vision" which is a new approach for addressing the problem of reconstruction from a large and imperfect dataset. It consists of three elements in both graphics and visions, namely, view dependence, levels-of-detail and primitive connectivity, and it adapts to a continuum of them. In order to give a better picture on ROD-TV and other related works, we use Figure 2.1 to position ROD-TV with each element on each axis accordingly.

![Figure 2.1 - Algorithm grouping in both graphics and visions](image)

Axis of View-Dependence – representation or description can be either viewer-centered or object-centered, axis of levels-of-detail axis – dataset can be represented in a single-scale or at multi-scale, axis of Primitive Connectivity – input points are fully connected to each other or isolated.
2.2 Algorithm Description

Figure 2.1 shows three axes based on view dependence, levels-of-detail and primitive connectivity. Algorithms are simply grouped into eight classes in both graphics and visions. In the following, a description of each class can be found.

CLASS 0 – [VIEW-DEPENDENCE× PRIMITIVE-CONNECTIVITY× LEVELS-OF-DETAIL×]

Many computer vision algorithms are situated at Class 0, at which the description is a 3-D isolated point set that supports a single-scale. Tensor voting [Medioni 00] [Tang 98] runs in sparse mode and belongs to this category. The input to the tensor voting is either a vector or a scalar field. First, the field points are being encoded into tokens. Second, each token is being refined after processing the sparse tensor voting. Third, a dense voting algorithm is carried out and a feature saliency map is obtained throughout a 3-D space. Finally features such as the surface patches and the curve segments can be extracted through an operation of non-maximal suppression. We can find more detail on the tensor voting in chapter 3. One of the advantages of using tensor voting is noise robustness – a smooth and hole-free triangular surface patch can be generated from the noisy input dataset. If traditional surface fitting algorithms [Hoppe 92] [Hoppe 94] are used, errors occur due to the presence of noise. Hence, our contribution, ROD-TV is entirely based on tensor voting in the local surface reconstruction. However, one of the disadvantages of the tensor voting is that it only supports a single-scale on the input dataset and some processing time is wasted on the invisible parts. Plugging in some graphics techniques – levels-of-details control and visibility analysis makes improvements on the tensor voting.

CLASS 1 – [VIEW-DEPENDENCE× PRIMITIVE-CONNECTIVITY× LEVELS-OF-DETAIL×]
Algorithms using a purely view-dependent description are grouped together into Class 1. Stereopsis [Barnard 82] [Marr 76] [Marr 79] is a classical computer vision problem belong to this category. The input to the stereopsis consists of two or more images taken from different viewpoints, and the output is the ability to infer information on the 3-D structure and the distance of a scene. First, given two different viewpoint images, a search problem is carried out in order to find all corresponding points between a left and right image. The correlation-based approach and feature-based approach are two common methods used to solve these corresponding points problem. Second, a fundamental matrix [Luong 96] [Torr 97] or an essential matrix is estimated from the corresponding points. You can make use of the eight-point algorithm [Hartley 95] [Longuet-Higgins 81] if you have at least eight pairs of the corresponding points, which is by far the simplest and numerical stable in computation. Third, the geometry of stereo: epipolar geometry is determined and rectification [Ayache 91] [Faugeras 93] is done on the stereo pairs. Finally, a 3-D reconstruction [Sparr 93] from two views is obtained based on the intrinsic and the extrinsic parameters of the stereo geometry. One of the properties in the view-dependent description is that different input viewpoint images leads to different reconstruction results. Also, it has the advantage that processing time is saved since time is spent only on the visible information. In our ROD-TV, we are interested in the view-dependent description instead of view-independent description, and therefore we are trying to reconstruct visible 3-D points given a particular viewpoint.

**Class 2 - [View-Dependence] [Primitive-Connectivity] [Levels-of-Detail]**

Algorithms converting unorganized points into a mesh description are grouped together into Class 2. The Delaunay triangulation algorithm [Aurenhammer 91] [Avis 83] [Lee 80] is the most prominent for mesh generation in computational geometry, which belongs to this category.
First, the algorithm starts from the hull edge, which is an initial edge we build in the convex hull. Second, a point-free triangle is constructed by finding the mate* of the initial hull edge. In other words, we need to find the smallest circumscribed circle passing through the mate point and the two vertex of the hull edge. Third, the point-free triangles are constructed one by one until there are no more Delaunay triangles in the point set. Finally, a collection of triangles, namely a fully connected mesh, is generated. No matter which algorithms is used, the same set of Delaunay triangles is returned if no assumption is violated. One of the disadvantages in mesh generation is that it is outlier sensitive. Hence, a noisy input dataset needs to be filtered in advance. In our ROD-TV, a fully or partially connected mesh can be obtained by adjusting the connectivity factor during surface reconstruction. An input point set to the ROD-TV can be either noisy or noise-free.

*mate point – a circumscribed circle is composed of three points. Two of them come from the edge AB and the remaining one comes from the mate of the edge AB. Hence, point C is defined as mate point. The following illustration depicts the idea.

CLASS 4 – [VIEW-DEPENDENCE* PRIMITIVE-CONNECTIVITY* LEVELS-OF-DETAIL*]

Algorithms changing from a single-scale to a multi-scale representation belong to Class 4. Form a 2-D image processing points of view, the MipMap [SOLOMON 01] [TANNER 98] [WILLIAMS 83] is a classical example belongs to this category. The underlying idea on the MipMap is to
generate a series of textures ranging from high to a low-resolution, and then renders an appropriate texture resolution based on the current levels-of-detail. For instance, for a closer surface, a high-resolution texture is rendered and vice versa. The advantage of doing multiple texture mapping is that the rendering performance is increased due to small memory consumption in the low-resolution texture mapping. However, the disadvantage is that we need to pre-compute each texture resolution offline. To generate a spectrum of texture, re-sampling and smoothing are two common operations, and therefore the Gaussian Pyramid [Burt 83] representation is introduced. For a 3-D case, a multi-scale data representation can be achieved using spatial hierarchical data structures. The low-resolution data representation is obtained in the root level, and the high-resolution data representation is obtained in the leaf level. Examples on the hierarchical data structure include Bounding Sphere [Rusinkiewicz 00], Octree [Samet 89a] [Samet 89b] and Grid Pyramid. In our Rod-Tv, appropriate level of detail of the dataset is obtained by traversing data hierarchy. In Chapter 3, details on the Rod-Tv data structure are described.

CLASS 3: [VIEW-DEPENDENCY, PRIMITIVE-CONNECTIVITY, LEVELS-OF-DETAIL] In CLASS 1, we point out that stereopsis is a classical computer vision problem which takes the viewing parameters into consideration based on the isolated point sets. If we make use of the viewing parameters into fully connected points (i.e., meshes), then it leads to a typical computer graphics problem and is being grouped into CLASS 4. In this category, different viewing positions will result in different resolution of meshes. Hence, the view-dependent mesh is named. Examples can be found in [Hoppe 97], which selectively refines an arbitrary mesh according to changing the viewing parameters. The finer mesh is obtained if 1) it lies inside viewing frustum 2) its triangular face normal is points towards to the viewer 3) its triangular face projection covers large area on the screen.
plane and vice versa. The view frustum, surface orientation and screen-space geometric error are three key refining criteria in the view-dependent mesh problem. One of the reasons for using view-dependent mesh is that graphics workload is minimized as the coarse mesh is obtained after being refined. Additional attention needs to be drawn when handling mesh resolution transition.

CLASS 5 — [VIEW-DEPENDENCE✓ PRIMITIVE-CONNECTIVITY✓ LEVELS-OF-DETAIL✓]

Algorithms involving view-dependent control and levels-of-detail control belong to this category. QSplat [rusinkiewicz 00] is a characteristic example, which renders a huge and unstructured 3-D scanned dataset hierarchically using costless polygons. QSplat and other related point rendering techniques do not perform reconstruction or maintain connectivity [zwicker 01] [pfister 00] [grossman 98]. Simple geometrical shapes such as points and ellipses are used as rendering primitives, which are splatted to cover more pixels in the screen space and to make holes invisible. Point rendering techniques without user intervention are not suitable for noisy input dataset, which is commonly found in many affordable range finders. Our on demand strategy is, however, inspired by their view-dependent LOD control to speed up surface reconstruction. On the other hand, the scientific visualization community is also interested in a grid-based hierarchical representation where explicit connectivity is not kept or used. The Marching Cubes algorithm [lorensen 87] extracts isosurface or isocontours. If the input scalar field is noise-free, of adequate resolution and regularly sampled, the Marching Cubes algorithm is sufficient. Enhancements such as multi-resolution capability and view-dependent speed-up were reported, in which a certain amount of neighborhood or connectivity information is considered. In [gerstner 00], a multi-resolution isosurface extraction based on hierarchical tetrahedral meshes generated by recursive bisection
is used. Topology preservation and controlled topology simplification are achieved.

**Class 6: View-Dependence ⊕ Primitive-Connectivity ⊕ Levels-of-Detail ⊔**

Algorithms generating a serious of mesh resolution belong to this class. It is a common technique for improving rendering performance – a detailed mesh is used when the object is close to the viewer, while a coarser approximation is substituted as the object recedes from the viewer. To produce different resolution of meshes, [Hoppe 96] introduced progressive mesh (PM) representation. The progressive mesh is the one in which the vertex information is stored internally in a special tree that can be accessed to render the mesh with any number of vertices. The progressive meshes not only capture a continuous sequence of meshes optimized for a view-independent LOD control, but also allow a fast traversal of the sequence at runtime. It is efficient, lossless and continuous-resolution representation. The advantages of using progressive meshes for LOD description.

**Class 7: View-Dependence ⊔ Primitive-Connectivity ⊔ Levels-of-Detail ⊔**

Mesh algorithms using both the viewing parameters and the levels-of-detail control are grouped together into this category. It takes the advantages of both Class 3 and Class 5. Usually, a better visible subset is obtained after taking the viewing parameters and the levels-of-detail control into consideration. In other words, a number of triangular faces are largely reduced without having a noticeable change in the final appearance. As the original complex meshes are being simplified, the computation load is reduced.
Chapter 3

Review of Basic Tensor Voting Formalism

3.1 Tensor Voting System

3.1.1 Introduction

Extracting salient and structured information from a noisy dataset is a problem in computer vision. In order to capture salient structures such as junctions, curves, regions and surfaces efficiently, tensor voting [MEDIONI 00], a unified computational framework for the inference of multiple salient structures in both 2-D and 3-D spaces is developed. Tensor voting consists of two elements. The first element is a tensor calculus, which is used for information representation. The other element is a voting process, which is used for information communication between input token sites and its neighborhood. The details on these elements will be described later in this chapter. Tensor voting is a non-iterative approach, requires no initial guessing on parameters, can infer salient structure under extremely noisy environment and the only free parameter is the scale (neighborhood size). An overall illustration of tensor voting is shown in Figure 3.1.

The input of tensor voting can be dense or sparse dataset with noise or not involved. The goal of tensor voting is to extract geometric features such as regions, curves, surfaces and intersections. The whole progress can be simply grouped into four stages, namely, information encoding, sparse tensor voting, dense tensor voting and feature extraction.
Figure 3.1 - Overview of the essential components of tensor voting
3.1.2 Stage One – Information Encoding

![Diagram of Stage One - Information Encoding](image)

Figure 3.2 - Tensor Voting Stage One - Information Encoding

Information encoding is the first stage [Figure 3.2]. The main idea is to use a tensor as an information representation before passing it to the second stage for the information exchange. The input to the tensor voting is treated as token sites and can be grouped into four categories. They are summarized in Table 3.1:

<table>
<thead>
<tr>
<th>Group No.</th>
<th>Group Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>Points</td>
</tr>
<tr>
<td>Group 2</td>
<td>Edges (Points with associated tangent)</td>
</tr>
<tr>
<td>Group 3</td>
<td>Surface patches (Points with associated normal)</td>
</tr>
<tr>
<td>Group 4</td>
<td>Any combination on the above</td>
</tr>
</tbody>
</table>

Table 3.1 - Input groups to the Tensor Voting

After information encoding, the output is the encoded tensor tokens. From the geometry points of view, input points are encoded in ball-like shape tensor. Edges are encoded in a disc-like shape tensor. Surface patches are
encoded in a stick-like shape tensor. Details on the various tensor representations will be described in Section 3.2 – information representation.

3.1.3 Stage Two – Sparse Tensor Voting

![Diagram](image)

Figure 3.3 - Tensor Voting Stage Two - Sparse Tensor Voting

In the second stage, it is sparse tensor voting [Figure 3.3]. Each input site propagates its information in a neighborhood with the help of a voting field*. The information is encoded in a tensor, (we call it a vote), and is determined by the predefined voting field**. Information in each input site is refined after getting votes from the neighborhood. Since we use the tensor as the information representation, we can get the confidence information from the tensor magnitude and we can get the orientation information from the tensor orientations [LEE 98]. The tensor confidence and orientation information will be described in details in Section 3.2 – information representation.

* In tensor voting, there are three types of voting field. They are ball voting field, plate voting field and stick voting field. More details will be found in section 3.3.4.

** We create a lookup table to store the all votes in range of the voting kernel.
3.1.4 Stage Three – Dense Tensor Voting

Tensor tokens are refined after processing sparse tensor voting in stage two. After that, dense saliency tensor fields are generated [Figure 3.4]. These refined tensor tokens propagate their information in their neighborhood, leading to a dense tensor map which encodes feature saliency at every point in the domain. In practice, we quantize the 3-D space into several small cubes, called voxel and cast the tensor votes at every cube vertex.

3.1.5 Stage Four – Features Extraction

In previous stage, we obtain a dense tensor map which encodes feature saliency at every point in the domain. In this stage, we can capture the features, for example, surfaces, curves, points by decomposing the tensor map into elementary components [Figure 3.5]. Since features are located at the local maxima in the saliency tensor map, we can extract them by non-maximal suppression.
3.2 Information Representation

3.2.1 Introduction

The goal of tensor voting is to extract geometric features such as surface, curves, and points in a robust and non-iterative way. It claims that global complex features can be composed of several simplified components. For example, complex smooth curve uses numerous simple curve segments to consolidate. The other example, complex smooth surface can be constructed by numerous simple surface-patches. In tensor voting, information is captured in a second order symmetric tensor. It is one type of the local representations. The advantages can be summarized as follows:

Geometrical generality

Local representations are more general as they describe different feature types in a uniform matter. We can determine the feature types by estimating their saliencies and geometrical properties locally. However, in global representation, it uses parametric functions to capture the whole geometric shape. Since we don't know what types of the feature we will be extracted in advance. Once we choose the functions to model inappropriately, the accuracy of the result may vary a lot.
**Geometrical singularity**

Local representations encode the complementary properties of smoothness and discontinuity properly. Since we use a second order symmetric tensor as an information primitive, it can capture the discontinuity with multiple orientations. In global representation, the parametric functions will fail to model if the smoothness and discontinuity exist at certain points in the model. It is because parametric functions can only capture one orientation at any location in the model.

**Geometrical feature inference**

Local representations estimate the features in a robust and non-iterative way. Features will be determined by a single pass voting process within a neighborhood. In global representations, we usually randomly pick an initial parameters and then running optimization algorithm to infer certain features. However, parametric models often misfit due to errors come from outliers and curve or surface discontinuities.

### 3.2.2 Second Order Symmetric Tensor

The tensor formalism was first developed for capturing variations of orientations in the study of fluid dynamics. Knutsson [Knutsson 89] [Granlund 95] and Westin [Westin 94] have used a second order symmetric tensor as data representation to solve a number of signal processing problems in computer vision and have obtained some promising results. Figure 3.6 depicts a geometric illustration of the second order symmetric tensor. We use the shape of the tensor to encode the uncertainty of orientation, and the size of the tensor to encode the feature saliency [Lee 98]. In the following, we will describe how to encode the information in both geometrical and algebraic way.
Algebraically, a second order symmetric tensor can be described by an eigensystem with eigenvalues $\lambda_{\text{max}} \geq \lambda_{\text{mid}} \geq \lambda_{\text{min}} \geq 0$, and corresponding unit eigenvectors $\hat{e}_{\text{max}}$, $\hat{e}_{\text{mid}}$ and $\hat{e}_{\text{min}}$. The matrix form is shown in Equation 3.1 and the expanded polynomial form is shown in Equation 3.2. In either case, the eigenvalues and eigenvectors are easily computed with some standard methods such as the Jacobi method [PRESS 96].

\[
\mathbf{S} = \begin{bmatrix} \hat{e}_{\text{max}} & \hat{e}_{\text{mid}} & \hat{e}_{\text{min}} \end{bmatrix} \begin{bmatrix} \lambda_{\text{max}} & 0 & 0 \\ 0 & \lambda_{\text{mid}} & 0 \\ 0 & 0 & \lambda_{\text{min}} \end{bmatrix} \begin{bmatrix} \hat{e}_{\text{max}} \\ \hat{e}_{\text{mid}} \\ \hat{e}_{\text{min}} \end{bmatrix}
\]

Equation 3.1 - Representing second order symmetric tensor by an eigensystem in a matrix form.

Geometrically, a second order symmetric tensor can be visualized as ellipse in 2-D space and ellipsoid in 3-D space [Figure 3.6]. With refer to Section 3.1 – tensor voting system; there are mainly three input elements to the tensor voting system. They are points, edges (or called CURVEL – curve segments element) and surface patches (or called SURFEL – surface patches element). In the following, we are going to make use of the second order symmetric tensor to encode each type of inputs for further processing.

\[
\mathbf{S} = \lambda_{\text{max}} \hat{e}_{\text{max}} \hat{e}_{\text{max}}^T + \lambda_{\text{mid}} \hat{e}_{\text{mid}} \hat{e}_{\text{mid}}^T + \lambda_{\text{min}} \hat{e}_{\text{min}} \hat{e}_{\text{min}}^T
\]

Equation 3.2 - Representing second order symmetric tensor by an eigensystem in a polynomial form.
Figure 3.6 - Geometrically describe a second order symmetric tensor in 3-D space

**Case 1: Surface patches element (SURFEL)**

![Diagram showing a surface patch element with normal N and position P = [x y z], encoding... with an output stick-like tensor marked \( \hat{e}_{\text{max}} \)].

---

-- Input: Surfel --

-- Output: Stick Tensor --

Figure 3.7 - Surface patches element (SURFEL)

Given position and normal, surface elements are encoded as stick-like tensor

30
Surface patches element (SURFEL) is represented by a stick-like tensor [Figure 3.7], which is a thin ellipsoid. The major axis of the tensor represents the direction of the normal to the patch. The length of the tensor represents the estimated saliency. Mathematically, the stick-like tensor can be formed by setting $\lambda_{\text{max}}=1, \lambda_{\text{mid}}=0, \lambda_{\text{min}}=0$ and $\hat{e}_{\text{max}}=\hat{n}$.

**Case 2: Curve segments element (CURVEL)**

![Figure 3.8 - Curve segment element (CURVEL)](image)

Given position and tangent, curve elements are encoded as plate-like tensor.

Curve segments element (CURVEL) is represented by a disc-like tensor [Figure 3.8]. The tangent direction of the curve element is aligned with the normal of the disc-like tensor. The radius of the disc-like tensor represents the estimated saliency. Mathematically, the disc-like tensor can be formed by setting $\lambda_{\text{max}}=1, \lambda_{\text{mid}}=1, \lambda_{\text{min}}=0$ and $\hat{e}_{\text{min}}=\hat{t}$.

**Case 3: Points element**

Isolated points element having no associated orientation is represented by a ball-like tensor [Figure 3.9], whose radius is proportional to the saliency of the estimation. Mathematically, the ball-like tensor can be formed by setting $\lambda_{\text{max}}=1, \lambda_{\text{mid}}=1$ and $\lambda_{\text{min}}=1$.  

31
Figure 3.9 - Point case
Given position, point elements are encoded as ball-like tensor

3.2.3 Tensor Decomposition

\[ T_{\text{symmetric}} = (\lambda_{\text{max}} - \lambda_{\text{mid}}) T_{\text{stick}} + (\lambda_{\text{mid}} - \lambda_{\text{min}}) T_{\text{plate}} + (\lambda_{\text{min}}) T_{\text{ball}} \]

\[ T_{\text{stick}} = \left( \hat{e}_{\text{max}} \hat{e}_{\text{max}}^T \right) \]

\[ T_{\text{plate}} = \left( \hat{e}_{\text{max}} \hat{e}_{\text{max}}^T + \hat{e}_{\text{mid}} \hat{e}_{\text{mid}}^T \right) \]

\[ T_{\text{ball}} = \left( \hat{e}_{\text{max}} \hat{e}_{\text{max}}^T + \hat{e}_{\text{mid}} \hat{e}_{\text{mid}}^T + \hat{e}_{\text{min}} \hat{e}_{\text{min}}^T \right) \]

Equation 3.3 - Representing a second order symmetric tensor by a stick tensor, a plate tensor and a ball tensor

From Equation 3.3, a second order symmetric tensor \( T_{\text{symmetric}} \) can be linearly decomposed in terms of stick tensor, plate tensor and ball tensor. Geometrical description is found in Figure 3.10. Since \( T_{\text{symmetric}} \) encapsulates the feature orientation and the saliency information at each location in 3-D space, we can extract the features by decomposing
$T_{\text{symmetric}}$ into following 2-tuple $(s, \hat{v})$, where $s$ is a scalar indicating the feature saliency and $\hat{v}$ is a unit vector indicating the orientation. Details on the orientation and the saliency information of each feature types can be found in Table 3.2.

<table>
<thead>
<tr>
<th>Feature Types</th>
<th>Orientation</th>
<th>Saliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface</td>
<td>Surface-ness normal is estimated by $\hat{e}_{\text{max}}$</td>
<td>$(\lambda_{\text{max}} - \lambda_{\text{mid}})$</td>
</tr>
<tr>
<td>Curve</td>
<td>Curve-ness tangent is estimated by $\hat{e}_{\text{min}}$</td>
<td>$(\lambda_{\text{mid}} - \lambda_{\text{min}})$</td>
</tr>
<tr>
<td>Point</td>
<td>Arbitrary</td>
<td>$(\lambda_{\text{min}})$</td>
</tr>
</tbody>
</table>

Table 3.2 - Feature saliency and orientation in a second order symmetric tensor

### 3.3 Tensor Communication

#### 3.3.1 Introduction

We now turn to our communication and computation scheme, which allows input tokens to exchange information with its neighbors, and infer new information.

#### 3.3.2 Token Refinement and Dense Extrapolation

The input tokens are first encoded as tensors. These initial tensors are capable to communicate with each other in order to:

- **Token Refinement**
  
  Derive the most preferred orientation information, or refine the initial orientation if it is given, for each of input tokens.
- Dense Extrapolation

Extrapolate the above inferred information at every location in the domain for the purpose of subsequent coherent feature extraction.

Figure 3.10 - A generic second order symmetric tensor

It can be linearly decomposed into ball-like tensor, plate-like tensor and stick-like tensor

These two tasks can be implemented by a voting process, which involves having each input tokens aligned with predefined dense voting kernels. This alignment is simply a translation followed by rotation. The dense voting kernels encode the tensors as votes. The derivation of the voting
kernels is given later in this section. Actually, this voting process is similar to convolution, except that the output of this process is a tensor instead of a scalar.

In the token refinement case, each token collects all the tensor values cast at its location by all the other tokens. The resulting tensor value is the tensor sum of all the tensor votes cast at the token location.

In the dense extrapolation case, each token is first decomposed into its independent elements, such as points, curves or surfaces. By using an appropriate voting kernel, each token broadcasts the information within a neighborhood. The size of the neighborhood is given by the size of the voting kernel used. It is the only parameter that you should input in tensor voting. As a result, a tensor value is put at every location in the neighborhood.

While they may be implemented differently for efficiency, these two operations are equivalent, which can be regarded as tensor convolution.

We now describe the design and the derivation of the voting kernel in 3-D space. All voting kernels can be derived form the fundamental 2-D stick kernel.

### 3.3.3 The Fundamental 2-D Stick Kernel

The voting field of any dimensions can be derived from the 2-D stick tensor, and therefore it is called the fundamental 2-D stick kernel. Figure 3.11 shows this fundamental 2-D stick kernel.

The design of this field is given below. It is based on assumptions in perceptual organization [Guy 96]. Note that, in 2-D space, a direction can
be defined by either the tangent vector or the normal vector, which is orthogonal to each other. We can therefore define two equivalent fundamental voting fields, depending whether we assign a tangent or normal vector at the receiving site. More details on plotting the tangent and normal version of the 2-D fundamental stick kernel can be found in [MEDIONI 00].

![Diagram](image)

**Figure 3.11 - The design of fundamental 2-D stick kernel**

Here, we describe the normal version of the 2-D stick kernel. The tangent version is similar. Refer to Figure 3.11, let us consider a normal direction \( \mathbf{N} \) is available at \( P \). Suppose a point \( Q \) is to be connected by a smooth curve with a low curvature to the origin \( P \). The most likely normal at \( Q \), which is unknown, is given by the normal to the circular arc at \( Q \). Such connection minimizes the total curvature and thus implicitly encodes the smoothness constraint. To encode the saliency, the magnitude of the normal is decayed with distance and curvature according to the Equation 3.4.

Using the above principle, we can generate the 2-D stick voting field by considering every point in the 2-space domain. Note that, although we use vectors to define the fundamental 2-D stick voting field, in the vote collection stage of tensor voting, we aggregate the second order moment contributions form each vector vote. This means that the resulting vote
collected denotes a direction along a line, and thus a second order symmetric tensor, but not an oriented vector.

\[ S_{PO}(r, j, \sigma) = e^{\frac{(r^2+c\varphi^2)}{\sigma^2}} \]

where
- \( r \) is the arc length from O to P
- \( c \) is a constant to control the decay with high curvature
- \( \varphi \) is the curvature parameter
- \( \sigma \) is the scale of analysis parameter

Equation 3.4 - Energy decay function in the 2-D fundamental stick kernel

Polarity information, which encodes the orientation of the tensor, is captured in the first order tensor. More details on the polarity information on the voting process can be found in [TONG 01].

3.3.4 Derivation of the Stick, Plate and Ball Voting Field

A 3-D stick voting field, 3-D plate voting field and 3-D ball voting field can be generated by 2-D fundamental stick kernel.

For the 3-D stick voting field, we first define the orientation of the 2-D fundamental stick kernel as \([1 \ 0 \ 0]^T\), i.e. aligning \( \hat{e}_{max} \) component with x-axis, and then rotating 180 degree about x-axis. The resulting voting field is the 3-D stick voting field. Figure 3.12 shows the discrete version on the 3-D stick voting field on the XY-plane, YZ-plane and ZX-plane.
For the 3-D plate voting field, we need to align the $\hat{e}_{\text{min}}$ component with $z$-axis and then rotate 180 degree about $z$-axis. The resulting voting field is the plate voting field, which describe a plate with normal pointing to $[0 \ 0 \ 1]^T$ direction. Figure 3.13 shows the discrete version on the 3-D plate voting field on the $XY$-plane, $YZ$-plane and $ZX$-plane. Mathematically, the equation of the 3-D plate voting field can be described as:

$$\text{Field}_{\text{plate}} = \int_0^\theta \text{Field}_{\text{stick}} d\theta$$

Equation 3.5 - Formula to generate plate voting field

(note that $\theta$ is the rotation angle about $z$-axis)
For the 3-D ball voting field, we need to sample the 3-D space uniformly in advance. To ensure this, we can make use of a platonic solid, for instance, dodecahedron or icosahedron. Dodecahedron has 20 polyhedron vertices and icosahedron has 12 polyhedron vertices. All vertices are distributed uniformly in 3-D space. To generate ball voting field, we align \( \hat{e}_{\text{max}} \) of the 3-D stick field with vertices and then accumulating all the tensor votes. Finally, 3-D ball voting field is generated. Figure 3.14 shows the discrete version on the 3-D ball voting field on the XY-plane, YZ-plane and ZX-plane.

\[
\text{Field}_{\text{ball}} = \int_0^\theta \int_0^\alpha \text{Field}_{\text{stick}} d\theta d\alpha
\]

Equation 3.6 - Formula to generate plate voting field

(note that \( \theta \) is the rotation angle about z-axis and \( \alpha \) is the rotation angle about x-axis)

Figure 3.14 - Discrete version on the 3-D ball voting field
([[left]: XY-plane; [middle]: YZ-plane; [right]: ZX-plane])

### 3.4 Features Extraction

#### 3.4.1 Introduction

At the end of the voting process, we produce a dense tensor map, which is then decomposed into three dense vector maps in the 3-D case. They are
Surface Map (SMAP), Curve Map (CMAP) and Junction Map (JMAP). Each voxel of these maps has a 2-tuple \( (s, \hat{v}) \), where \( s \) is a scalar indicating the feature saliency, and \( \hat{v} \) is a unit vector indicating the direction. A brief summary on the tensor map can be found in Table 3.3.

<table>
<thead>
<tr>
<th>Tensor Map</th>
<th>Orientation [direction] ( (\hat{v}) )</th>
<th>Saliency [magnitude] ( (s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Map (SMAP)</td>
<td>( \hat{v} = (\hat{e}_{\text{max}}) ) indicating the normal direction</td>
<td>( s = (\lambda_{\text{max}} - \lambda_{\text{mid}}) )</td>
</tr>
<tr>
<td>Curve Map (CMAP)</td>
<td>( \hat{v} = (\hat{e}_{\text{min}}) ) indicating the tangent direction</td>
<td>( s = (\lambda_{\text{mid}} - \lambda_{\text{min}}) )</td>
</tr>
<tr>
<td>Junction Map (JMAP)</td>
<td>( \hat{v} = \text{Arbitrary no specific direction} )</td>
<td>( s = (\lambda_{\text{min}}) )</td>
</tr>
</tbody>
</table>

Table 3.3 - Feature saliency and orientation in a tensor map

These maps are dense vector fields which are then used as an input to the maximal features extraction algorithms in order to generate features such as surfaces, curves, and junctions.

### 3.4.2 Maximal Surfaces in 3-D

To determine maximal surfaces at point \( p = [x, y, z]^T \) in 3-D, we make use of Surfaces Map (SMAP). SMAP holds 2-tuple \( (s, \hat{v}) \), where \( s = (\lambda_{\text{max}} - \lambda_{\text{mid}}) \) indicating the saliency (magnitude) and \( \hat{v} = (\hat{e}_{\text{max}}) \) indicating the orientation (direction). We denote \( \hat{v} \) by \( \hat{n} \) since it indicates the surface normal.

Considering continuous version of the problem of the maximal surfaces, in which \( (s, \hat{n}) \) is defined for every point in 3-D space. A point is on a
maximal surface if its saliency \( s = (\lambda_{\text{max}} - \lambda_{\text{mid}}) \) is locally maximal along
the direction of the normal. That is, the point \( p = [x \ y \ z]^T \) lies on a
maximal surface if the differential property \( \frac{ds}{d\hat{n}} = 0 \) holds at point \( p \). This
is a necessary condition for the maximal surface. A sufficient condition,
which is used in implementation, is defined in terms of zero crossings
along the line defined by \( \hat{n} \). We therefore define the gradient vector \( \tilde{g} \) as,

\[
\tilde{g} = \nabla s = \left[ \frac{\partial s}{\partial x} \frac{\partial s}{\partial y} \frac{\partial s}{\partial z} \right]^T
\]

and project \( \tilde{g} \) onto \( \hat{n} \), i.e., \( q = \hat{n} \cdot \tilde{g} \). Thus, a maximal surface is the locus of
points with \( q = 0 \).

For the discrete case, we can define the discrete gradient vector as

\[
\tilde{g}_{h,j,k} = \left[ s_{h+1,j,k} - s_{h,j,k} \right]
\]

and the locus of the maximal surface normal becomes \( q_{h,j,k} = \hat{n}_{h,j,k} \cdot \tilde{g}_{h,j,k} = 0 \). Therefore, the set \( \{q_{h,j,k}\} \) constitute a scalar field which
can be processed directly by Marching Cubes algorithm [LORENSEN 87]
[NIKOLAIDIS 01]. Given a voxel, this procedure takes the eight
\( \{q_{h,j,k}\} \) vertices of the voxel as the input. Triangulation of the local surface
patches is produced through a zero-crossing detection in each voxel
locally. Details on the maximal surfaces extraction are found in [MEDIIONI
03].
3.4.3 Maximal Curves in 3-D

To determine maximal curves at point \( p = [x \ y \ z]^T \) in 3-D, we make use of Curves Map (CMAP). CMAP holds 2-tuple \((s, \hat{v})\), where \( s = (\lambda_{mid} - \lambda_{min}) \) indicating the saliency (magnitude) and \( \hat{v} = (\hat{e}_{min}) \) indicating the orientation (direction). We denote \( \hat{v} \) by \( \hat{e} \) since it indicates the curve tangent.

Considering continuous version of the problem of the maximal curves, in which \((s, \hat{e})\) is defined for every point in 3-D space. A point \( p = [x \ y \ z]^T \) is on the maximal curve if any displacement from \( p \) on the plane normal to \( \hat{e} \) will result in a lower \( s \) value. That is a point \( p \) lies on the maximal curve if the differential property \( \frac{ds}{du} = \frac{ds}{dv} = 0 \) holds at point \( p \).

This is a necessary condition for the maximal curve. A sufficient condition, which is used in implementation, is defined in terms of zero crossings in the \( U-V \) plane normal to \( \hat{e} \). To do this, we therefore define the gradient vector \( \tilde{g} \) as,

\[
\tilde{g} = \nabla s = \left[ \frac{\partial s}{\partial x} \ \frac{\partial s}{\partial y} \ \frac{\partial s}{\partial z} \right]^T
\]

and define \( \tilde{q} = (\hat{e} \times \tilde{g}) \times \hat{e} \). By construction, \( \tilde{q} \) is the projection of \( \tilde{g} \) onto the plane normal to \( \hat{e} \). Therefore, the maximal curve is the locus of point for which \( \tilde{q} = 0 \). We can define the corresponding discrete version of \( \tilde{q} \), and for all eight vertices of the voxel we compute \( \tilde{q}_{i,j,k} \). The signs of the
elements in $\tilde{q}_{ijk}$ indicate whether there is any zero-crossing and thus curve segment passing through that voxel. Details on the maximal curves extraction are found in [MEDIONI 03].

3.4.4 Maximal Junctions in 3-D

To determine maximal junctions in 3-D, we make use of Junctions Map (JMAP). JMAP holds 2-tuple $\left( s, \hat{v} \right)$, where $s = (\lambda_{\min})$ indicating the saliency (magnitude) and $\hat{v} = \text{(Arbitrary)}$ indicating the orientation (direction). 3-D junctions are isolated points. There are no specific orientations. Maximal junctions can be straightly extracted from the local maxima of the $s = (\lambda_{\min})$ values.

3.4.5 Maximal Features Extraction

Maximal features, namely, surface, curves and junctions have been defined in Section 3.4.1, Section 3.4.2 and Section 3.4.3 respectively. To extract the maximal junctions, we can directly select the local maximal of its saliency value. In Section 3.4.1 and 3.4.2, we define the maximal curves and surfaces in terms of zero crossings. We can extract them through non-maximal suppression and the details algorithm on their extraction can be found in [MEDIONI 00].

3.5 Tensor Voting Operations

In tensor voting formalism, an information inference is done by vote casting under a predefined voting kernel. There are three kinds of voting kernels, namely ball voting kernel, plate voting kernel and stick voting kernel. All kernels are directly derived from 2-D fundamental stick field. For example, the 3-D stick voting kernel is generated by rotating 2-D
fundamental stick field around z-axis with its $\hat{e}_{\text{max}}$ component pointing to x-axis. The 3-D plate voting kernel is generated by rotating the 3-D stick voting kernel around z-axis before aligning its $\hat{e}_{\text{min}}$ component with z-axis. The 3-D ball voting kernel is generated by rotating the 3-D plate field around x-axis. For the 2-D fundamental stick field generation, in [TANG 01], NormalVote operation [Algorithm 3.1] is introduced. All votes in fundamental stick casting are in vector form (vector vote), which encapsulate the polarity information. For the 3-D voting kernel, all votes are in form of covariance matrix (tensor vote). It does not encapsulate polarity information compared to vector vote. Details in the tensor vote generation is listed in TensorVote operation [Algorithm 3.2] [TONG 01]. Besides, the vector vote can be combined into the tensor vote by means of Combine operation [Algorithm 3.3].

Different geometric features are extracted if you choose different voting kernels during the sparse and dense tensor voting stage. The domain is not only restricted in 2-D space or 3-D space, but also can be extended to be more general framework for solving higher dimension problems [TANG 01]. One of the examples is estimating epipolar geometry in computer vision using 8-D tensor voting [TANG 01].
Algorithm 3.1: NormVoterVote(Voter, Votee)

1. \( v \leftarrow \text{Votee.pos} - \text{Voter.pos} \)
2. /* case 1: voter and votee are at the same position */
3. if \( \text{Voter.pos} = \text{Votee.pos} \) then
4. /* Vote is cast by Voter without decaying */
5. \( \text{StickVote.dir} \leftarrow \text{Voter.dir} \)
6. \( \text{StickVote.mag} \leftarrow \text{Voter.mag} \)
7. \( \text{StickVote.pos} \leftarrow \text{Voter.pos} \)
8. /* case 2: voter and votee are on the straight line */
9. else if \( \text{arg} (\text{Voter.dir}, v) = \pi/2 \) then
10. /* Vote is cast by Voter with decaying */
11. \( \text{StickVote.dir} \leftarrow \text{Voter.dir} \)
12. \( \text{StickVote.mag} \leftarrow \exp (- (c^2 \sigma^2)/\sigma ma)^2 \) [equation 3.4]
13. \( \text{StickVote.pos} \leftarrow \text{Voter.pos} \)
14. /* case 3: voter and votee are connected with a HIGH curvature value */
15. else if \( \text{arg} (\text{Voter.dir}, v) < \pi/4 \) then
16. /* Zero Vector */
17. \( \text{StickVote} \leftarrow \text{ZeroVector} \)
18. /* case 4: voter and votee are connected with a LOW curvature value */
19. else
20. /* Vote is cast by Voter in osculating circle */
21. /* compute radius of osculating circle */
22. \( \text{radius} \leftarrow 1/\cos (\text{arg} (\text{Voter.direction}, v)) \times \text{Voter.dir} \)
23. /* compute center of osculating circle */
24. \( \text{center} \leftarrow \text{radius} - v/2 \)
25. \( \text{StickVote.dir} \leftarrow \text{center} - \text{Voter.pos} \)
26. \( \text{StickVote.mag} \leftarrow \exp (- (c^2 \sigma^2)/\sigma ma)^2 \) [equation 3.4]
27. \( \text{StickVote.pos} \leftarrow \text{Voter.pos} \)
28. end if
29. return \( \text{StickVote} \)

Algorithm 3.1 - Operation to generate vector vote between voter site and votee site
for all $0 \leq i, j < 2$ do
    TensorVote[i][j] ← 0
end for

/* Compute Stick Component */
stickSaliency ← Voter.lambdaMax - Voter.lambdaMid
if stickSaliency > 0 then
    Vote ← NormalVote(Voter, Votee)
    TensorVote ← Combine(TensorVote, Vote, stickSaliency)
end if

/* Compute Plate Component */
plateSaliency ← Voter.lambdaMid - Voter.lambdaMin
if plateSaliency > 0 then
    /* uniform sampling points in a circle */
    sample[ ] ← GenRandomUniformPts()
    for all $0 \leq k < sample\_count$ do
        sample[k].dir ← sample[k].pos - Voter.pos
        VoterTransform ← Voter dot sample[k].dir
        Vote ← NormalVote(VoterTransform, Votee)
        TensorVote ← Combine(TensorVote, Vote, plateSaliency)
    end for
end if

/* Compute Ball Component */
bailSaliency ← Voter.lambdaMin
if ballSaliency > 0 then
    /* uniform sampling points in a sphere */
    sample ← GenRandomUniformPts()
    for all $0 \leq k < sample\_count$ do
        VoterTransform ← Voter x sample[k].dir
        Vote ← NormalVote(VoterTransform, Votee)
        TensorVote ← Combine(TensorVote, Vote, bailSaliency)
    end for
end if

return TensorVote

Algorithm 3.2 - Operation to generate tensor vote between voter site and votee site
Summary

In this chapter, we have reviewed the basic tensor voting formalism. We have described the elements of the approach, and the flow of processing through the system.

From initial, sparse, and noisy 3-D dataset, we produce features such as junctions, curves and surfaces. The approach consists of three elements

- Encoding the information using tensor
- Communication and computation using sparse and dense tensor fields
- Features extraction using tensor decomposition and local marching algorithm in [MEDIONI 00]
Chapter 4

Data Structure for ROD-TV

4.1 Introduction

To perform reconstruction on demand, a good data structure is needed. Such a data structure not only encodes different representations on any given input dataset, but also facilitates visibility culling to reduce the processing time. In the original version of the tensor voting [MEDIONI 00], a list with single representation is used to perform an object-centered surface reconstruction. It is a costly operation since an entire dataset is being processed even if points are either out of a viewing frustum or occluded by the foreground surface patches. To carry out just enough or sufficient surface reconstruction effectively, we propose to make use of a hierarchy spatial data structure, which is the first contribution of this thesis. Details are given in this chapter.

In this chapter, we first describe some background on the spatial data structure in Section 4.2; choices on the hierarchy data structure are discussed in Section 4.3 and Section 4.4. In Section 4.5, simple experiments are done on the data structure to show its advantages. Finally, summary is given in Section 4.6.

4.2 Spatial Data Structures

4.2.1 Overview

A spatial data structure can organize geometry in a high dimensional space. For example, in the 2-dimensional and 3-dimensional space, Quad-
tree and Octree are used respectively. To arrange geometry in a higher dimension, Kd-tree is used. In this thesis, we only focus on the geometry in 3-dimensional space.

The organization of a spatial data structure is usually hierarchical. It means that the input data are decomposed into several different levels. Normally different levels are organized in a tree structure. The topmost level encloses the level below it [Figure 4.3] [Figure 4.4], and the next level encloses the next one after it, and so on. The building process is usually constructed in a recursive manner. It should be noticed that the construction is expensive and is usually done in a preprocess stage.

The main reason for using a hierarchy is that different types of queries get significantly faster. Searching is done by traversal of the tree structure from root to leaf or from leaf to root. The performance is improved from $O(N)$ to $O(\log N)$, given that the tree structure is approximately balanced. Besides, the data hierarchy encodes different representations for the input data and so-called data multi-representation. By taking union of all the leaf nodes of the tree, the highest data representation is attained [Figure 4.5.B]. In contrast, the lowest data representation is found in the root node [Figure 4.5.E]. The data resolution transition is done by smoothing, down-sampling and storing it in internal nodes [Figure 4.5.C-D].
4.2.2 Spatial Subdivision and Object Subdivision

![Diagram showing input tokens, hierarchical bounding sphere, and hierarchical octree.]

Figure 4.1 - Hierarchical Bounding Sphere and Hierarchical Octree

The left part shows a sample input tokens, which are all isolated points with no geometry involved. The middle part shows that input tokens are organized by a bounding sphere hierarchy. The right part shows that input tokens are grouped together into regular cells with different size.

There are two different types of spatial data structure [Figure 4.1]. They are Bounding Volume Hierarchies (BVHs) and Spatial Volume Hierarchies (SVHs).

For the BVHs, subdivision is object-orientated. Usually, each object is enclosed by some simple geometrical shape. Examples of the shapes are spheres and boxes. A set of objects within each geometrical shape is simplified. As space subdivision is fully dependent on the input objects, it is irregularly partitioned and does not cover the entire space. Also, siblings of BVHs may overlap each other due to irregular splitting. Examples of the BVHs are bounding spheres [Figure 4.1], Axis-aligned Bounding Boxes (AABBs) and Oriented Bounding Boxes (OBBs).

For the SVHs, subdivision is regular, meaning that space is split in a uniform fashion. Recursive subdivision is performed in the entire space and is independent of the input objects positions. Since the space splitting
is regular, all siblings are disjoint and do not overlap each other. Examples of the SVHs are Quadtrees, Octrees [Figure 4.1], Kd-trees and Grids. In Table 4.1, it shows a brief comparison of the BVHs and SVHs.

<table>
<thead>
<tr>
<th>Subdivision</th>
<th>Bounding Volume Hierarchies (BVHs)</th>
<th>Spatial Volume Hierarchies (SVHs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regularity</td>
<td>Object</td>
<td>Space</td>
</tr>
<tr>
<td>Scene unity</td>
<td>Irregular</td>
<td>Regular</td>
</tr>
<tr>
<td>Sibling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Overlap</td>
<td>Disjoint</td>
</tr>
<tr>
<td>Examples</td>
<td>AABBs, OBBs, Spheres</td>
<td>Quadtrees, Octrees, Kd-trees, Grids</td>
</tr>
</tbody>
</table>

Table 4.1 - Comparison between Bounding Volume Hierarchy (BVHs) and Spatial Volume Hierarchy (SVHs)

In my thesis, I have implemented two SVHs. They are Grid Pyramid and Octree. In the following sections, we discuss both in detail. For the BVHs, bounding sphere hierarchy has been tried out in [Tong 02]. The general idea of the construction of a bounding spheres hierarchy is depicted in Figure 4.2. The algorithm on the construction is described in Algorithm 4.1.
Figure 4.2 - The construction of Hierarchical Bounding Sphere

Given a set of tokens (1), which are all isolated points, we can construct a Hierarchical Bound Sphere using a bottom-up approach. It means that the finest resolution is constructed first and then next resolution and so on. (2) – (6) show the details on the bounding sphere construction in each resolution level. You may notice that the number of representative tokens reduces in each level. Details of the algorithm can be found in [Tong 02].

Node BuildTree(tokens[begin,end])
   if tokens[first] == tokens[end] then
      position ← PARTITIONLINE(tokens[begin])
      return new Node(tokens[first], position, radius)
   else
      middle ← PARTITIONPOINTS(tokens[begin,end])
      return BuildTree(tokens[begin,middle])
      return BuildTree(tokens[middle,end])
   end if

Algorithm 4.1 - Constructing Hierarchical Bounding Sphere Algorithm

This is a recursive algorithm. The above shows the 1-D BUILDTREE algorithm and it is the standard recursive tree algorithm. For the 3-D case, we can replace the PARTITIONLINE with BOUNDINGSphere which returns 2-tuple (position, radius).
4.3 Rod-TV Data Structure I: Grid Pyramid

4.3.1 Introduction

![Diagram of Grid Pyramid in 3-D space](image)

**Figure 4.3 - Grid Pyramid in 3-D space**

The Grid Pyramid is a hierarchy of volume. Volume resolution decreases from the bottom of the pyramid to the top. (A) Lo-resolution – the entire scene is represented by a 1 grid cell (B) Mi-resolution – the entire scene is represented by 8 grid cells. The cell size is halved compared to (A). (C) Hi-resolution – the entire scene is represented by 64 grid cells. The cell size is halved compared to (B). In (D), the corresponding tree representation of the Grid Pyramid is shown. The lo-resolution (A) is located at the tree root and the hi-resolution (C) is located at the tree leaf.

In this section, we describe how to use Grid Pyramid to organize input tokens hierarchically. The Grid Pyramid is a hierarchy of volume. Volume resolution decreases from the bottom of the pyramid to the top. In Figure 4.3, it shows the Grid Pyramid in 3-D space. To represent the lowest resolution on the input data, a single cell is used to enclose the entire 3-D space. To attain the next lowest resolution, current single cell is subdivided into 8 cells, which is used to enclose the 3-D space. The highest resolution of the input data is obtained by repeating the subdivision recursively until all cells are homogenous, which means that
all the cells are either full or empty. The successive subdivisions can be represented as a tree as shown in Figure 4.3.D, where the lowest resolution is encoded in the tree root and the highest resolution is encoded in the tree leaves. The Grid Pyramid for the 2-D case is similar to that of a 3-D case. The only difference is that four cells is returned instead of eight cells for each subdivision. It is depicted in Figure 4.4.

![Figure 4.4 - Grid Pyramid in 2-D space](image)

The Grid Pyramid is the hierarchy of a plane in 2-D space. Resolution decreases from the bottom of the pyramid to the top. (A) Lo-resolution – the entire scene is represented by 1 grid cell (B) Mi-resolution – the entire scene is represented by 4 grid cells. The cell size is halved compared to (A). (C) Hi-resolution – the entire scene is represented by 16 grid cells. The cell size is halved compared to (B). In (D), corresponding tree representation of the Grid Pyramid is shown. The lo-resolution (A) is located at the tree root and the hi-resolution (C) is located at the tree leaf.

### 4.3.2 Construction of Grid Pyramid

There are two approaches for constructing a Grid Pyramid – the top-down approach and the bottom-up approach. For the top-down case, the pyramid is constructed from the original volume (the entire scene). The construction then proceeds downwards by iteratively subdividing current
cells into eight equal cells in 3-D case and four equal cells in 2-D case. Such iteration ends when reaching the leaf node. For the bottom-up case, the pyramid is constructed from the original volume, then proceed upwards by iteratively consolidate a group of eight equal cells in 3-D case and four equal cells in 2-D case. Such iteration ends when reading the root node. In this thesis, we construct the grid pyramid in bottom-up fashion.

Figure 4.5 illustrates the construction of a Grid Pyramid in 2-D space. Given an 8x8 scene with 12 isolated tokens [Figure 4.5.A], the pyramid is construction in a bottom-up fashion. To build the leaf level of the Grid Pyramid, we quantize the entire scene into cells. The size of each cell is equal to 1. If there is more than one token within a cell, the smoothing operation [Figure 4.6.B] is carrying out. The idea of the smoothing operation is to find the average value among all tokens in a cell. After smoothing and quantizing, there are 11 tokens remaining in the leaf level of the Grid Pyramid [Figure 4.5.B]. To continue constructing upwards in the Grid Pyramid, we simply repeat quantization and smoothing operations until the root level is reached [Figure 4.5 B-D]. On level 0 (leaf level), there are 11 tokens remaining. On level 1, there are 8 tokens remaining. On level 2, there are 3 tokens remaining and finally there only 1 token remaining in level 3 (root level). As you can see, the number of tokens is reduced from bottom to top in the Grid Pyramid. Multi-resolution on the input tokens can be addressed through the hierarchical decomposition of the Grid Pyramid. Additionally, a tree can be used to represent the Grid Pyramid as shown in Figure 4.5.F. By traversing different height levels of the tree, the corresponding resolution of the input data is being returned. This hierarchical data structure is useful in visibility culling (Chapter 5) and Levels-of-Detail control (Chapter 6).
Figure 4.5 - The construction of Grid Pyramid in bottom-up fashion

(A) A sample scene in 2-D space is given. There are 12 tokens in the scene with a dimension of 8x8. (B) Building the highest resolution (level 3) in the Grid Pyramid. (No. of cells = 64 with size equals to 1) (C) Building the next resolution (level 2) of (B) in the Grid Pyramid. (No. of cells =16 with size equals to 2) (D) Building the next resolution (level 1) of (C) in the Grid Pyramid (No. of cells = 4 with size equal to 4) (E) Building the lowest resolution (level 0) in the Grid Pyramid (No. of cells = 1 with size equal to 1, which is the dimension of the entire scene. (F) A tree representation of the Grid Pyramid. Note that we can achieve different resolutions by traversing the height of the tree. (G) It is a key of the tree, which indicates the ordering of the child cells. It is in anti-clockwise fashion starting at the lower-left child cell. (H) Another key of the tree which tells you the node type of the tree. An empty cell is labeled as incomplete and colored in white. A full cell is labeled as complete and colored in black. A partial cell labeled as incomplete and colored in grey. Empty = does not contain any token in a cell. Full = contain exact one token in a cell. Grey = contains more than one token and has child cells.
Figure 4.6 - Grid Pyramid operations: smoothing operator and quantization operator

(A) An intermediate level of construction on a Grid Pyramid is shown. The input is level k-1 (hi-resolution) of the Grid Pyramid and the output is level k-1 (lo-resolution) of the Grid Pyramid. This involves two operations, namely the smoothing operation and the quantizing operation. (B) Smoothing operator – the input is level k-1 which consists of 3 tokens and 4 grid cells. After the smoothing process, the output is smoothed level k-1 which contains 1 token and 4 grid cells. In this case, the black box on the smoothing process is averaging. (C) The quantizing operator – the input is smoothed level k-1 with 1 token and 4 grid cells. After the quantizing process, the output is level k with 1 token and 1 grid cell. In this case, the black box on the quantizing process is condensing 4 smaller cells into 1 big cell.
4.3.3 Grid Pyramid Operations: Token Smoothing and Cell Quantization

To build a Grid Pyramid from level k-1 (higher resolution) to level k (lower resolution) involves two stages. The first stage is token smoothing. For each single cell in level k-1, we perform some metrics to simplify the tokens and return the most representative one as an output for level k. Smoothed level k-1 is obtained after smoothing and there is no change in its cell numbers. The whole process is depicted in Figure 4.6.B. The second stage is token quantization. For each single cell in a smoothed level k-1, grid cells are merged with its neighbor. Usually, in a 2-D case, merging involves four in a group and in 3-D case, merging involves eight in a group. It is a down-sampling progress and is depicted in Figure 4.6.C. After quantization, the number of grid cells in the output level reduces four times and doubles its cell size. By repeating the above two stages from the leaf layer to the root layer, a hierarchical Pyramid Grid is produced.

4.3.4 Pyramid Grid Benefits and Limitations

There are two benefits to be gained from a Grid Pyramid being used in data representation. The first benefit is its data multi-resolution capability. Significant regions are represented by a higher level of the tree. The most significant region is the root level. Data density is increased when a deeper level of the tree is traversed. Finally, the less significant region is located in the leaf level of the tree. By taking the union of all the leaf nodes, the whole input data is returned. The next benefit is the data compression capability. This is similar to jpeg compression in image processing. Some information is thrown out without loss of data appearance so that you can still visualize what they look like. In the Grid
Pyramid, data compression is achieved by keeping only the higher levels of the tree.

Nonetheless, the Grid Pyramid has some limitations. In Figures 4.3 and 4.4, every level in the Grid Pyramid is represented by regular cells. Each level has the same sized cells. Therefore, it leads to the problem of memory wasting. A large portion of space is tessellated with a large group of empty grid cells. Is it possible to make use of a large cell instead of a group of small empty grid cells? This introduces the concept of the topics about adaptive cell tessellation. In the coming section, Octree is chosen to perform adaptive cell tessellation.

4.4 Rod-TV Data Structure 2: Octree

4.4.1 Introduction

![Octree in 3-D space](image)

Figure 4.7 - Octree in 3-D space

Octree is a subset of the Grid Pyramid that completely spans the volume. (A) This is a hi-resolution representation as you can see in Figure 4.3.C. To perform snug fit in a 3-D scene, large octants are used to represent low density regions and small octants are used to represent high density regions. (B) The corresponding tree representation of (A). All internal nodes are colored in grey and labeled as incomplete nodes, which means that the node scalar value is computed by averaging all its child scalar values. All external nodes are colored in black and labeled as complete. Complete node means that it is a homogenous cell which contains the actual input value. Each parent has eight children exactly per subdivision, and therefore Octree is named.
To perform adaptive cell tessellation, Octree is used. Octree is one type of SVHs. It is designed to span the entire 3-D space with different octants of various sizes. Therefore, demanding memory storage requirement is met. In Figure 4.7.A, it shows an example on the Octree in 3-D space. Usually, the density in the 3-D scene is non-uniformly distributed, some regions get higher density and some regions get lower density. In an Octree representation, space spanning is adaptive. This means that high density regions are represented by small octants and low density regions are represented by large octants. In other words, if data points are too close together, we need to use smaller octants to separate them. Otherwise, larger octants are used in the separation. Figure 4.7.B shows the corresponding 3-D space partitioning in a tree representation. Every parent in the tree has exactly eight children after the subdivision process. Therefore, because of the 8-split, Octree takes its name.

Figure 4.8 illustrates an example of a 2-D space. The only difference is that every parent has 4 children instead of 8 in a 2-D space.

4.4.2 Construction of Octree

To construct Octree, we can extract it from Grid Pyramid by a single traversal of the pyramid from top to bottom. At each level of the pyramid, we examined the node types. There are totally three node types, namely complete node, incomplete node and semi-complete node. If the current node is a complete node, this means that there is only one token in the region, therefore no further subdivision is needed. A complete node is an external node which contains actual data from the input. If we collect all the complete nodes, the whole input dataset is returned. Furthermore, if the current node is an incomplete node, this means that there is no data in the cell, therefore no subdivision is needed. If the current node is a semi-complete node, this means that there is more than one token in the region, therefore subdivision on the current region is needed. By repeating the
subdivision from top to bottom in the Grid Pyramid, an Octree is produced. Figure 4.9 illustrates an example of the construction of an Octree from the Grid Pyramid.

Figure 4.8 - Quadtree in 2-D space

Quadtree is a subset of Octree in 2-D space. (A) This is a hi-resolution representation as you see in Figure 4.4.C. To perform snug fit in a 2-D scene, large quadrants are used to represent low density regions and small quadrants are used to represent high density regions. (B) The corresponding tree representation of (A). All internal nodes are colored in grey and labeled as incomplete nodes, which means that the node scale value is computed by averaging all its child scalar values. All external nodes are colored in black and labeled as compete. Complete node means that it is a homogenous cell which contains the actual input value. Each parent has four children exactly after subdivision, and therefore Quadtree is named.

4.4.3 Octree Benefits and Limitations

Through using Octree as a data structure, a lot of space is saved by tessellating cells of different size. A large region of space is being represented by a single large cell. Therefore, compared to Grid Pyramid, Octree is an efficient data structure to represent the entire 3-D space hierarchically. You can visualize this benefit in the example as shown in Figures 4.5 and 4.9. Table 4.2 points out the comparison of the number of cells consumed between Grid Pyramid and Octree. If
Figure 4.9 - The construction of Octree in top-down fashion

(A) A sample scene in 2-D space is given. There are 12 tokens in the scene with a dimension of 8x8. To build an Octree from Pyramid Grid, we need to take a look on the node type. There are three kinds of node types. They are complete, incomplete and semi-complete node. For complete node, if it contains exactly 1 token, then no further subdivision is carrying out on this node. For incomplete node, if it is an empty cell, then no subdivision is carrying out too. However, for remaining semi-complete node, since it contains more than 1 token, subdivision on the current node is needed. Complete node and incomplete node are classified as homogenous. Semi-complete node is classified as in-homogenous. (B) It shows the construction on the root level of Octree. It examined the root node which is an in-complete node, subdivision is carrying out and four children quadrants are produced. By repeating the subdivision progress from root to leaf as shown in (B) – (E). Finally Octree is produced. (F) its is a tree representation. You can find that the number of cells is lesser compared to Pyramid Grid (Figure 4.5.F). There are 25 cells in Octree and there are 85 cells in Grid Pyramid. Therefore Octree is a compact data structure for hierarchical data representation.
Octree is used, it consumes only 25 cells. However, if Grid Pyramid is used, it consumes 85 cells. There is a 60 cell difference between them and around 70% memory storage is saved on the regional space.

<table>
<thead>
<tr>
<th>Resolution level</th>
<th>No. of cells in Grid Pyramid</th>
<th>No. of cells in Octree</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution level 3</td>
<td>1</td>
<td>1</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Resolution level 2</td>
<td>4</td>
<td>4</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Resolution level 1</td>
<td>16</td>
<td>12</td>
<td>4 (4.71%)</td>
</tr>
<tr>
<td>Resolution level 0</td>
<td>64</td>
<td>8</td>
<td>56 (65.88%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>85</strong></td>
<td><strong>25</strong></td>
<td><strong>60 (70.59%)</strong></td>
</tr>
</tbody>
</table>

Table 4.2 - Comparison on the number of cells between Grid Pyramid and Octree

However, there are some limitations to the Octree data structure. First, if two tokens are close together, a large number of subdivision steps is needed to separate them. In other words, the tree is very high. Figure 4.10 depicts this case. To solve this problem, we can threshold the maximum number of subdivisions to limit the tree height. If the number of subdivisions reaches the threshold, then no more subdivisions are needed. All the tokens within a region are being condensed into one token and stored in a leaf node. Another method is to define another threshold on the minimum number of tokens within a region. After regional splitting, if the child regions reach the minimum threshold value, all tokens within these regions are condensed into one token and stored it in a leaf node.

Second, Octree is not usually a balanced tree during the construction. To construct an approximated balanced Octree, we are better to perform the first subdivision in the center of the scene. More details about the Octree construction can be found in [Samet 89A] [Samet 89B].

63
Figure 4.10 - A subdivision problem in Octree construction

(A) If two tokens get much too close together, then the number of subdivisions is huge. (B) In tree representation, the number of subdivisions can be reflected in the tree height. The deeper the tree, the larger number of subdivisions.

4.5 Experimental Results

In this section, we are going to present the results on the hierarchical data structure. We have made use of two datasets. They are PREGNANTWOMAN and MONKEYSADDLE. The PREGNANTWOMAN dataset consists of 41,836 data points. The input data can be decomposed into 12 levels of detail as shown in Figure 4.9. The highest resolution level is indexed at 0, which contains almost all the points from the input. The lowest resolution level is indexed at 12, which contains only one point. The whole sequence of the data resolution can be found from Figure 4.9.A to Figure 4.9.M. As we have described the Grid Pyramid and Octree in section 4.3 and section 4.4, an experiment on cells consumption in both Grid Pyramid and Octree is done and summarized in [Table 4.3]. After examining the table, you note
that Octree is a good hierarchical data structure for performing space partitioning in a 3-D scene. A large number of cells can be saved compared to Grid Pyramid. In other words, Octree consumes less memory over Grid Pyramid. The corresponding graph can be found in Figure 4.10. The shaded region on the graph indicates the number of cells or memory that can be saved by using Octree. The higher the resolution, the more memory can be saved. The MONKEYSADDLE dataset consists of 19,209 data points. The same experiment is done as with the PREGNANTWOMAN dataset. The experimental results are summarized in Table 4.4 and the corresponding graph plotting is shown in Figure 4.12.
Figure 4.11 - Data multi-resolution on the PregnantWoman dataset

(N) The original dataset, which consists of 41,836 data points. (A) The highest resolution level. (M) The lowest resolution level. (B) – (L) The transition levels between the highest and lowest resolutions.
<table>
<thead>
<tr>
<th>Multi-resolution level</th>
<th>Tree height</th>
<th>Cell size</th>
<th>No. of cells in Grid Pyramid</th>
<th>No. of cells in Octree</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>0.015625</td>
<td>$2^{12} \times 2^{12} \times 2^{12}$</td>
<td>41,580</td>
<td>68,719,435,130</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>0.03125</td>
<td>$2^{11} \times 2^{11} \times 2^{11}$</td>
<td>41,575</td>
<td>8,589,893,014</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.625</td>
<td>$2^{10} \times 2^{10} \times 2^{10}$</td>
<td>41,540</td>
<td>1,073,700,284</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0.125</td>
<td>$2^{9} \times 2^{9} \times 2^{9}$</td>
<td>41,425</td>
<td>134,176,303</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0.25</td>
<td>$2^{8} \times 2^{8} \times 2^{8}$</td>
<td>40,929</td>
<td>16,736,287</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0.5</td>
<td>$2^{7} \times 2^{7} \times 2^{7}$</td>
<td>32,447</td>
<td>2,064,705</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>$2^{6} \times 2^{6} \times 2^{6}$</td>
<td>10,363</td>
<td>251,781</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2</td>
<td>$2^{5} \times 2^{5} \times 2^{5}$</td>
<td>2,790</td>
<td>29,978</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>$2^{4} \times 2^{4} \times 2^{4}$</td>
<td>712</td>
<td>3,384</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>8</td>
<td>$2^{3} \times 2^{3} \times 2^{3}$</td>
<td>178</td>
<td>334</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>16</td>
<td>$2^{2} \times 2^{2} \times 2^{2}$</td>
<td>48</td>
<td>16</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>32</td>
<td>$2^{1} \times 2^{1} \times 2^{1}$</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>64</td>
<td>$2^{0} \times 2^{0} \times 2^{0}$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3 - Experimental results on the PREGNANTWOMAN dataset

---

**Figure 4.12** - A comparison between Grid Pyramid and Octree on the PREGNANTWOMAN dataset

67
Figure 4.13 - Data multi-resolution on the MONKEYSADDLE dataset

(M) The original dataset, which consists of 19,209 data points. (A) The highest resolution level. (L) The lowest resolution level. (B) - (K) The transition levels between the highest and lowest resolutions.
<table>
<thead>
<tr>
<th>Multi-resolution level</th>
<th>Tree height</th>
<th>Cell size</th>
<th>No. of cells in Grid Pyramid</th>
<th>No. of cells in Octree</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>0.03125</td>
<td>$2^{11} \times 2^{11} \times 2^{11}$</td>
<td>18,841</td>
<td>8,589,915,748</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0.625</td>
<td>$2^{10} \times 2^{10} \times 2^{10}$</td>
<td>18,836</td>
<td>1,073,722,988</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>0.125</td>
<td>$2^9 \times 2^9 \times 2^9$</td>
<td>18,789</td>
<td>134,198,939</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0.25</td>
<td>$2^8 \times 2^8 \times 2^8$</td>
<td>18,240</td>
<td>16,758,967</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0.5</td>
<td>$2^7 \times 2^7 \times 2^7$</td>
<td>15,792</td>
<td>2,081,360</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>$2^6 \times 2^6 \times 2^6$</td>
<td>10,036</td>
<td>252,108</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
<td>$2^5 \times 2^5 \times 2^5$</td>
<td>3,298</td>
<td>29,470</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>4</td>
<td>$2^4 \times 2^4 \times 2^4$</td>
<td>855</td>
<td>3,241</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>8</td>
<td>$2^3 \times 2^3 \times 2^3$</td>
<td>222</td>
<td>290</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>16</td>
<td>$2^2 \times 2^2 \times 2^2$</td>
<td>58</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>32</td>
<td>$2^1 \times 2^1 \times 2^1$</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>64</td>
<td>$2^0 \times 2^0 \times 2^0$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4 - Experimental results on the MONKEYSADDLE dataset

---

![Graph showing a comparison between Grid Pyramid and Octree on the Monkey Saddle dataset](image)

**Figure 4.14 - A comparison between Grid Pyramid and Octree on the MONKEYSADDLE dataset**
4.6 Summary

In this chapter, we first point out that a good hierarchical data structure is needed to perform surface reconstruction on demand. Then, two categories of hierarchical data structure are described – spatial subdivision and object subdivision. Through the merit of the hierarchy, the input dataset can be decomposed into several resolution levels. To archive this, Grid Pyramid and Octree have been discussed. After collecting the experiment results on the comparison between Grid Pyramid and Octree, we find that Octree is a good data structure for doing space division in 3-D scene space because of its adaptive cell size tessellation. In the following chapters, we are going to describe how to make use of Octree to perform visibility culling (Chapter 5) and Levels-of-Detail reconstruction (Chapter 6).
Chapter 5

Visibility Algorithms for ROD-Tv

5.1 Introduction

To perform reconstruction on demand, visibility algorithms are essential. Given a viewing position in a world space, such algorithms can determine a potential visible clique, which is a subset of the original input, for further processing in ROD-Tv pipeline. Usually, the size of this visible clique is small and therefore the "on demand" requirement of ROD-Tv is achieved. Compared to the original version of the tensor voting, all input tokens are involved in surface reconstruction whether they are visible or not. It is a time-consuming operation since most of the CPU time is spent on the invisible parts. To speed up the whole reconstruction process efficiently, various different kinds of visibility algorithms are introduced, which is the second contribution of this thesis. Details are outlined in this chapter.

In this chapter, various culling algorithms for speeding up tensor voting will be presented and explained. We start with view frustum culling (Section 5.2), followed by occlusion culling (Section 5.3). Finally, experimental results (Section 5.4) are discussed.
5.2 Rod-TV Visibility Algorithm I: View Frustum Culling

5.2.1 Viewing Frustum in a 3-D scene

The Viewing Frustum in a 3-D scene is usually defined by six clipping planes, namely the near, far, left, right, top and bottom clipping planes, which together form a cut pyramid. The region, which is enclosed by these planes, is called the viewing volume. If we ignore all the clipping planes, an infinite pyramid is generated. The general idea of the view frustum culling is as follows – If a token is inside the viewing volume, then it is classified as a visible token and added into a potential visible clique. Otherwise, they will be culled and classified as an invisible token. No further operation is processed on invisible tokens. Figure 5.1 depicts the infinite pyramid and viewing frustum.

---

![Image of infinite pyramid and viewing frustum]

Figure 5.1 - Infinite Pyramid and Viewing Frustum

The illustration on the right (B) is an infinite pyramid. The position of the camera is at the apex of the pyramid. The illustration on the left (A) is a viewing frustum. It consists of six clipping planes. They are the left, right, top, bottom, near, and far clipping planes. The enclosed region, which is shaded, is defined as a viewing volume. Objects are visible if they are inside the viewing volume. Otherwise, they will be culled without any further process.
The main advantage of using a view frustum culling is to select a visible subset from the original dataset. All off-screen tokens are removed and processing time is saved as we carry out surface reconstruction on the visible clique. To determine whether a token is visible or not, we generalize the whole culling process into a different geometrical test.

### 5.2.2 Plane-Token Intersection

![Diagram of Plane-Token Intersection](image)

#### Figure 5.2 - Intersection Test between Plane and Token

(A) Geometrically describe the perpendicular distance between a token and a plane in 3-D space. If the distance is positive, the token is located in front of the plane (B), which has the same region as the plane normal pointing to. If the distance is negative, the token is located behind the plane (D). The token is on the plane if the distance is zero (C).
To determine whether a token is inside or outside the viewing frustum, we need to carry out a plane-token intersection test. In this test, perpendicular distance between token and plane will be computed. Let \( \vec{P} = (x, y, z) \) be a point on the plane. In general, the equation of a plane with a normal \( \vec{N} = (a, b, c) \) passing through a well-known point on the plane \( \vec{Q} = (x_0, y_0, z_0) \) can be described as:

\[
\vec{N} \cdot (\vec{P} - \vec{Q}) = 0
\]

\[
(a, b, c) \cdot (x - x_0, y - y_0, z - z_0) = 0
\]

\[
a \cdot x + b \cdot y + c \cdot z - (a \cdot x_0 + b \cdot y_0 + c \cdot z_0) = 0
\]

\[
a \cdot x + b \cdot y + c \cdot z + d = 0
\]

where \( d = -(a \cdot x_0 + b \cdot y_0 + c \cdot z_0) \)

---

Equation 5.1 – The equation of a plane in general form

It is the General Form of an equation of plane in 3-D space. If the plane is being normalized, Hessian Normal Form can be obtained:

\[
\vec{N} = \frac{(a, b, c)}{\| (a, b, c) \|}
\]

\[
= \left( \begin{array}{ccc}
\frac{a}{(a, b, c)'(a, b, c)} & b & c \\
\end{array} \right)
\]

From Equation 5.1, let us divide \( \| \vec{N} \| \) on both sides and we get the following:
\[
\begin{align*}
\frac{a \cdot x}{\| (a,b,c) \|} + \frac{b \cdot y}{\| (a,b,c) \|} + \frac{c \cdot z}{\| (a,b,c) \|} + \frac{d}{\| (a,b,c) \|} &= 0 \\
\left( \frac{a \cdot x}{\| (a,b,c) \|} \frac{b \cdot y}{\| (a,b,c) \|} \frac{c \cdot z}{\| (a,b,c) \|} \right) \cdot (x, y, z) &= r \\
\text{where } r &= -\frac{d}{(a,b,c)} \\
\hat{N} \cdot \vec{p} &= r
\end{align*}
\]

\[
\hat{N} \cdot \vec{p} = r
\]

Equation 5.2 - The equation of a plane in Hessian Normal Form

Given a token \( \vec{T}_{token} = (x_{token}, y_{token}, z_{token}) \) in a 3-D space and a plane equation \( \hat{N}_{plane\_normal} \cdot \vec{p}_{pt\_on\_plane} = r_{distance} \), the Token-Plane Distance can be calculated by projecting \( \vec{W} \) onto \( \hat{N}_{plane\_normal} \), where \( \vec{W} \) is the vector from the plane to the token \( \vec{T}_{token} \), i.e.

\[
\vec{W} = \vec{T}_{token} - \vec{p}_{pt\_on\_plane} = (x_{token} - x, y_{token} - y, z_{token} - z)
\]

Let \( D \) be the perpendicular distance between the token and the plane, we get:

\[
D = \vec{W} \text{ ProjectOn } \hat{N} \\
D = \vec{W} \cdot \hat{N} \\
D = (\vec{T}_{token} - \vec{p}_{pt\_on\_plane}) \cdot \hat{N} \\
D = (\vec{T}_{token} \cdot \hat{N}) - (\vec{p}_{pt\_on\_plane} \cdot \hat{N}) \\
D = (\vec{T}_{token} \cdot \hat{N}) - r
\]

\[
D = \vec{T}_{token} \cdot \hat{N} + r
\]

Equation 5.3 - Token-Plane Distance
If the value $D$ is positive, we say that the token $T_{\text{token}}$ is in front of the plane, which lies in the same region as the plane normal. If the value $D$ is negative, we say that the token $T_{\text{token}}$ is behind the plane. If the value $D$ is equal to zero, we say that the token $T_{\text{token}}$ is on the plane. Figure 5.3 shows three different cases of the plane-token intersection test. Table 5.1 summarizes all the results and the corresponding algorithm can be found in Algorithm 5.1.

<table>
<thead>
<tr>
<th>Token-Plane Distance</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D &gt; 0$</td>
<td>Token is in front of the plane</td>
</tr>
<tr>
<td>$D = 0$</td>
<td>Token is on the plane</td>
</tr>
<tr>
<td>$D &lt; 0$</td>
<td>Token is behind the plane</td>
</tr>
</tbody>
</table>

Table 5.1 - Three Difference Cases in Plane-Token Intersection Test

```plaintext
Double TokenPlaneDistance(token, plane)

distance <- DOT(plane[normal], token[position]) + plane[distance from origin]

if distance >= 0 then
    /* token is in front of the plane and is in the same region with normal */
else
    /* token is behind the plane and is in the opposite region with normal */
end if

return distance
```

Algorithm 5.1 - Computing Distance between Token and Plane
The inputs to the function are the token position and plane. The output from the function is the perpendicular distance between token and plane. If the returned value is greater than zero, then the token is located in front of the plane. If the returned value is less than zero, then the token is behind the plane. If the returned value is equal to zero, then the token is on the plane.
5.2.3 Frustum-Token Intersection

Figure 5.3 - Frustum-Token Intersection
(A) Token inside the frustum (B) Token on the frustum (C) Token outside the frustum

To determine whether a token is inside or outside the viewing frustum, Frustum-Token Intersection Test is performed. The general idea of this test is simply computing a distance of each clipping plane with its normal pointing inward. In the other words, it performs Plane-Token Intersection Test, at most, six times. If all resultant distances from the planes are positive, this means that the token is in front of all clipping planes, and then it is classified as a visible token and added into a potential visible clique. Otherwise, the token is classified as an invisible token and is culled without any further process. The details of the Frustum-Token Intersection Test can be found in Algorithm 5.2.

```
Bool TokenInFrustum(token, plane[6]*)

    pos <- token[position]
    for all i such that 0 <= i < 6 do
        if ( PointPlaneDistance(pos, plane[i]) < 0 ) then
            /* token is behind the clipping plane */
            return false
```
end if
end for

/* token is in front of all clipping planes */
return true

* Frustum consists of six clipping planes and all plane normals are pointing inward.

Algorithm 5.2 - Frustum-Token Intersection

The inputs to the function are the token position and the clipping planes of the viewing frustum. The output from the function is to determine if the input token is being culled or not. If the return value is true, then the input token is not culled. Otherwise, the input token is culled.

5.2.4 Frustum-(Bounding Volume) Intersection

![Frustum-(Bounding Volume) Intersection](image)

Figure 5.4 - Frustum-(Bounding Volume) Intersection

A group of tokens is enclosed by a simple geometrical shape. A bounding sphere is used in this example. (A) (Bounding Volume) is inside the frustum (B) (Bounding Volume) intersects the frustum (C) (Bounding Volume) is outside the frustum.

Given an input dataset and the current viewing parameters, we can figure out the visible set by performing the Frustum-Token Intersection Test token by token. The running time is linearly proportional to the number of input data points. Can we improve on that? The answer is yes. To speed up the intersection test, we can use several simple geometrical shapes to group several tokens together, and then perform testing on these simple shapes instead of individual tokens. The simple shape, which encloses a group of tokens, is called the bounding volume and it is depicted in
Figure 5.4. The running time spent on the intersection gains little improvement since we only perform a test on the bounding volume instead of every token. Figure 5.5 shows the result of the intersection test. Basically, there are three returned cases after the intersection test. They are the INSIDE, OUTSIDE and INTERSECT frustums. As you can see in the figure, \( d \) is the perpendicular distance between the sphere center and the clipping plane and \( r \) is the radius of the bounding sphere. Suppose \( d \) is greater than zero. If \( d \) is greater than or equal to \( r \), then the bounding sphere is classified as INSIDE frustum. Otherwise, it is classified as INTERSECT frustum. However, suppose \( d \) is less than zero and if \( d \) is greater than or equal to \( r \), then the bounding sphere is classified as OUTSIDE frustum. Otherwise, it is INTERSECT. Both INSIDE and OUTSIDE are easy case to handle since a single test is done on the bounding sphere when these cases are returned. However, if the bounding sphere is INTERSECT frustum, and then we need to continue performing the test on its children until all the visible tokens are found. In order to perform an intersection test on the children of the current bounding sphere, it involves the hierarchical spatial data structure, which is described in chapter 3. In the following section, a logarithmic running time view frustum culling is discussed.
Figure 5.5 - Frustum-(Bounding Sphere) Intersection

Let \( r \) be the radius of bounding sphere and let \( d \) be the perpendicular distance between sphere center and the nearest clipping plane. There are three possible returned cases: OUTSIDE, INSIDE and INTERSECT. (A) Example scene on the Frustum-(Bounding Sphere) Intersection Test. (B) A close-up on the sample scene. Suppose \( d \) is greater than zero. The bounding sphere is classified as INSIDE frustum if \( d \) is greater than or equal to \( r \). Otherwise, it is classified as INTERSECT. However, suppose \( d \) is less than zero. The bounding sphere is classified as OUTSIDE frustum if \( d \) is greater than or equal to \( r \). Otherwise, INTERSECT is instead.

5.2.5 Hierarchical View Frustum Culling

In chapter 4, the hierarchical data structure for ROD-TV has been discussed. In this section, we make use of it to perform a hierarchical view frustum culling. The main advantage of using hierarchy is that the coarse geometrical description is encoded in the topmost layer and the finest geometrical description is encoded in the bottommost layer. In order to perform view frustum culling in logarithmical time, we start to perform the frustum test from the root to the leaf. If the root is entirely inside the frustum, all of its sub-tree are being added into a visible set. If the root is completely outside the frustum, all of its sub-tree are being culled. If the root is partially inside the frustum, then testing carries on to its children. The whole process performs in a top-down approach from the root to the leaf of the tree. The running time is \( O(\log N) \) instead of \( O(N) \).
as we found in the previous section. Figure 5.6 illustrates an example of the hierarchical view frustum culling. The corresponding algorithm can be found in [Algorithm 5.3].

Figure 5.6 - Hierarchical View Frustum Culling

A set of tokens and its bounding boxes are shown on the left (A). This scene is rendered with the hierarchical view frustum culling from the view point of the camera. The corresponding tree representation of the input tokens is shown on the right (B). The root of the tree intersects with the frustum, and the traversal continues with testing its children. Since parent of the token 1 and 2 is fully outside the frustum, all of its children are being culled without any further testing. However, the bounding box of the token 6 is entirely inside the frustum, it is not culled and its sub-tree is added into potential visible set directly. For the intersection case, the parent of tokens 3, 4, and 5 intersects the frustum, a test is carrying out on its children until the leaf node is reached. If the leaf node is entirely inside the frustum, the node is not culled. If the leaf node is entirely outside the frustum, the node is culled. If the leaf node is intersects with the frustum, Token-Frustum Intersection Test is carrying out.

PotentialVisibleSet ← {} /* empty set */
TraverseHierarchy( node )

\[
\text{if ( node is OUTSIDE frustum ) then}
\]

/* token is outside the frustum */
return
end if
if ( node is INSIDE or INTERSECT frustum ) then
    if ( node is leaf ) then
        if ( FrustumTokenIntersection( node ) ) then
            PotentialVisibleSet ← node
        end if
    return
end if
for each childnode do
    TraverseHierarchy( childnode )
end for
end if

Algorithm 5.3 - Hierarchical View Frustum Culling

It is a recursion algorithm. We start the test from the root node of the hierarchical spatial data structure. If the root complete inside the frustum, then all of its children are inserted into potential visible set. If the root is outside the frustum, the function terminates immediately. If the root is partially inside the frustum, test is recursive carrying out on its children from a top-down fashion.

5.3 Rod-TV Visibility Algorithm II: Occlusion Culling

After performing hierarchical frustum culling, a potentially visible set is generated where all tokens are inside the viewing frustum. Even though all tokens in the set are totally inside the frustum, some of them are being occluded by the others. In order to extract a "better" visible set, occlusion culling is carrying out. The general idea of the algorithm is to try to cull away (avoid processing) tokens that are occluded, that is, inside the viewing frustum but not visible in the final result. Figure 5.7 illustrates the big picture on the occlusion culling. An occluder set [Figure 5.7.C] will become a "better" visible set and further operations will be carried out in this set. However, a occludee set will be culled in order to save the processing time.
Figure 5.7 - General Idea on Occlusion Culling

The illustration on the left (A) shows a sample scene with six tokens. All of them are inside the frustum. However, some of them are invisible since they are being occluded by the others. To figure out the visible set, occlusion culling is performed. (C) An occluder set – a group of tokens we want to extract for further processing. (B) An occlude set – a group of tokens we want to cull after the occlusion test.

5.3.1 Image Space Algorithm

There are two classes of visibility algorithms – object space and image space. For the object space, a set of visible tokens is determined directly from the 3-D space. For the image space, tokens are first projected onto an image plane and a decision is made based on the projection and depth information. In this thesis, an image space occlusion culling algorithm is used. We figure out the resultant visible set in the image plane only without considering the object-space information. The details of the algorithm are shown in [algorithm 5.4].
5.3.2 Occlusion Culling: Construction of Occlusion Map and Depth Map

Figure 5.8 - Occlusion Culling: Occlusion Map and Depth Map
A sample scene is depicted in (A). The image plane is located between the near clipping plane and the camera. (B) There are 2 maps adhere with the image plane – d-map and o-map. (C) o-map (occlusion map) – it records the potential occluder set. (D) d-map (depth map) – it records the corresponding depth information on each token in the occluder set.

To begin with, we need to prepare two maps, namely an occlusion map and a depth map. Both of them are of 2-D array with the dimensions same as the image plane. For example, if the current viewing resolution is 500 by 500, then the array size of both maps are 500 by 500. This is shown in Figure 5.8. The depth map is used to record depth information on each projected token. Initially, all elements in the depth map are set to infinity. Geometrically, it means that tokens are positioned at infinity and cannot be visualized in the image plane. For the occlusion map, potential occluders are recorded and corresponding pixels are marked. This map not only indicates the object-projection distribution on the image plane, but also keeps track the potential occluder set. Figure 5.8.C and Figure 5.8.D illustrate the occlusion map and depth map respectively.
5.3.3 Occlusion Culling: Depth Test

To figure out the visible set from the input tokens, a depth test is carried out. The idea on the depth test is to compare the depth value for each token if they overlap each other along a particular line of sight. After the depth test, the closer token will be returned. Then, the corresponding occlusion map and depth map will be updated. The outdated one will be discarded. Figure 5.11 illustrates the idea on the depth test. After performing depth test on every input token, a better visible set is generated for further processing.

5.3.4 Problem in Occlusion Culling

If we use the image space algorithm, then we only produce a conservative visible set. It means that we cannot produce the exact visible set and some of the tokens are still invisible and cannot be culled. Figure 5.9 demonstrates one of the problems.

![Figure 5.9 - Occlusion Culling: Problem in Occluder Set](image)

After processing the occluder culling depth test, we can determine a better visible set. All most of the tokens
5.3.5 Contribution Threshold

To remove the tokens which are furthest from the camera, we propose to use a contribution threshold. After projecting the token on the image plane, we can set a threshold on the size or area of the projected region. If the projected region is below a certain threshold, then it will be culled. Figure 5.10 illustrates this idea. When the token is positioned at P, the projected region covers 24 pixels. When it is at position Q, the region covers 12 pixels. If the token moves further away from the camera, let say, at position R, the regions reduced to 4 pixels. To make matters worse, if the token is at S, its projection is less than 1 pixel and we cannot visualize it in the image plane. Therefore, it is trivial to set the threshold to at least 1 pixel. In the other words, if the token projection is less than 1 pixel of the image plane, then that token will be culled. As a result, this threshold can help us to determine a group of tokens which are closer to the viewer.

Figure 5.10 - Occlusion Culling: Contribution Threshold

The illustration on the left (A) shows a token with its bounding sphere at Q. The corresponding projection on the image plane is depicted in (C). (B) The projection of the token at position P. (D) The projection of the token at position R. (E) The projection of the token at position S.
Figure 5.11 - Occlusion Culling: Depth Test

In the upper row, the original scene setup is shown. The original scene is shown in (A), it contains one token with bounding sphere P. Its d-map is depicted in (C) and had the depth value of 5. The corresponding o-map is shown in (B), the projected pixel is marked and the current occluder set is being kept track. Now in the middle row, a token is added into the scene with bounding sphere Q as shown in (D). After doing the perspective projection on the image plane. The corresponding d-map and o-map are shown in (F) and (E). After projection, the depth test is carrying out between these maps. First, overlapping test is done on o-map of (B) and o-map of (E). If the same pixels are being projected, a depth comparison between (C) and (F) is performed and the one which has a minimum distance from the camera is selected. After doing the depth comparison, o-map and d-map is updated as shown in (G) and (H).
OcclusionMap[screenX][screenY]
DepthMap[screenX][screenY]

IsOcclude(posX, posY, posZ)
/* pixel is recorded */
if ( ISRECORD( OcclusionMap[posX][posY] ) ) then
    if ( posZ < DepthMap[posX][posY] ) then
        return false
    else
        return true
    end if
else
    return true
end if

/* pixel is not recorded */
return false

OcclusionCulling( tokens )

INITIALIZE( OcclusionMap, BLACK )
INITIALIZE( DepthMap, INFINITY, NULL )

for each token in tokens do
    [posX posY posZ] ← PROJECT( token )
    if ( IsOcclude(posX,posY,posZ) ) then
        /* skip */
    else
        /* set occluder */
        OcclusionMap[posX][posY].ptr ← token
        /* update occlusion map and depth map */
        DepthMap[posX][posY] ← posZ
        OcclusionMap[posX][posY].color ← WHITE
    end if
end for

Algorithm 5.4 - General Occlusion Culling Algorithm
The input to the function is a group of tokens in 3-D space. The out of the function is a set of potential visible set, which is a set of occluders along the line of sight.
5.4 Experimental Results

Two culling algorithms have been carried out on two different dataset. They are Teapot dataset with 26,103 tokens and Hand dataset with 38,220 tokens. Table 5.2 compares the survival of the number of tokens after the frustum culling using two dataset of different view directions. From the table, it can be seen that large portions of the tokens are culled away. Figure 5.12.C and Figure 5.13.C show an image of visible tokens from Teapot dataset from their different view directions. Figure 5.15.C and Figure 5.16.C show an image of visible tokens from Hand dataset. Since the total number of tokens being produced is significantly reduced, we are able to save the rendering time without affecting the correctness of the final result.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>View</th>
<th>No. of Visible Tokens</th>
<th>No. of whole Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teapot</td>
<td>[Figure 5.12.C] - 1st</td>
<td>17,558</td>
<td>26,103</td>
</tr>
<tr>
<td>Teapot</td>
<td>[Figure 5.13.C] - 2nd</td>
<td>16,732</td>
<td>26,103</td>
</tr>
<tr>
<td>Hand</td>
<td>[Figure 5.15.C] - 1st</td>
<td>12,657</td>
<td>38,220</td>
</tr>
<tr>
<td>Hand</td>
<td>[Figure 5.15.C] - 2nd</td>
<td>25,819</td>
<td>38,220</td>
</tr>
</tbody>
</table>

Table 5.2 - Comparisons between the Numbers of Tokens Survival after Frustum Culling

Table 5.3 compares the survival of the number of tokens after the occlusion culling. From the table, it can be seen that a very large proportion of the tokens are removed. Figure 5.14.B and Figure 5.14.C show an image of the visible set from Teapot dataset with 2 different pixel thresholds during token projection. Figure 5.16.D shows an image of the visible set from Hand dataset after occlusion culling.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>View</th>
<th>No. of Visible Tokens</th>
<th>No. of whole Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teapot</td>
<td>[Figure 5.14.B] - 1st</td>
<td>2,654</td>
<td>26,103</td>
</tr>
<tr>
<td>Teapot</td>
<td>[Figure 5.14.C] - 2nd</td>
<td>1,062</td>
<td>26,103</td>
</tr>
<tr>
<td>Hand</td>
<td>[Figure 5.15.D] - 1st</td>
<td>3,848</td>
<td>38,220</td>
</tr>
</tbody>
</table>

Table 5.3 - Comparisons between the Numbers of Tokens Survival after Occlusion Culling
### Figure 5.12 Description

<table>
<thead>
<tr>
<th>(A)</th>
<th>Teapot Dataset - 26,103 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Current Screen-Space Viewing Window (view-port)</td>
</tr>
<tr>
<td>(C)</td>
<td>Frustum Culling is enabled – There are 17,558 tokens out of 26,103 tokens inside the frustum (67.26%). The remaining 8,545 tokens are culled (32.74%).</td>
</tr>
<tr>
<td>(D)</td>
<td>Frustum Culling is disabled – No tokens are culled. The full dataset (26,103 tokens) has been rendered. (100%)</td>
</tr>
</tbody>
</table>

### Figure 5.13 Description

<table>
<thead>
<tr>
<th>(A)</th>
<th>Teapot Dataset - 26,103 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Current Screen-space viewing window (view-port)</td>
</tr>
<tr>
<td>(C)</td>
<td>Frustum Culling is enabled – There are 16,732 tokens out of 26,103 tokens inside the frustum (64.10%). The remaining 9,371 tokens are culled (35.90%).</td>
</tr>
<tr>
<td>(D)</td>
<td>Frustum Culling is disabled – No tokens are culled. The full data (26,103 tokens) has been rendered (100%).</td>
</tr>
</tbody>
</table>

### Figure 5.14 Description

<table>
<thead>
<tr>
<th>(A)</th>
<th>Teapot Dataset - 26,103 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Occlusion Culling is enabled with pixel threshold equal to 1 pixel – There are 2,654 tokens out of 26,103 tokens have been chosen to be in a visible set (10.17%). The remaining 23,449 tokens are culled (89.83%). Since the threshold on the pixel size is too small. Some of the far tokens cannot be culled. One solution is to increase the threshold value.</td>
</tr>
<tr>
<td>(C)</td>
<td>Occlusion Culling is enabled with pixel threshold equal to 3 pixels – There are 1,062 tokens out of 26,103 tokens have been chosen to be in a visible set (4.68%). The remaining 25,041 tokens are culled (95.32%). Since the threshold on the pixel size has been increased. We can generate a better visible set effectively.</td>
</tr>
</tbody>
</table>

### Figure 5.15 Description

<table>
<thead>
<tr>
<th>(A)</th>
<th>Hand Dataset – 38,220 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Current Screen-space viewing window (view-port)</td>
</tr>
<tr>
<td>(C)</td>
<td>Frustum Culling is enabled – There are 12,657 tokens out of 38,220 tokens (33.12%). The remaining 25,563 tokens are culled (66.88%).</td>
</tr>
<tr>
<td>(D)</td>
<td>Frustum Culling is disabled – No tokens are culled. The full dataset (38,220 tokens) has been rendered.</td>
</tr>
</tbody>
</table>

### Figure 5.16 Description

<table>
<thead>
<tr>
<th>(A)</th>
<th>Hand Dataset – 38,220 tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B)</td>
<td>Current Screen-space viewing window (view-port)</td>
</tr>
<tr>
<td>(C)</td>
<td>Frustum Culling is enabled – There are 25,819 tokens out of 38,220 tokens inside the frustum (67.55%). The remaining 12,401 tokens are culled (32.45%).</td>
</tr>
<tr>
<td>(D)</td>
<td>Occlusion Culling is enabled with pixel threshold equal to 1 pixel – There are 3,848 tokens out of 38,220 tokens have been chosen to be in a visible set (10.07%). The remaining 34,372 tokens are culled (89.93%).</td>
</tr>
</tbody>
</table>
Figure 5.12 - Frustum Culling on the Teapot dataset [view point 1]
Figure 5.13 - Frustum Culling on the Teapot dataset [view point2]
Figure 5.14 - Occlusion Culling on the Teapot dataset
Figure 5.15 - Frustum Culling on the HAND dataset
Figure 5.16 - Visibility Test on the Hand dataset
Given a particular viewing location, a small visible set of input dataset can be generated after performing both frustum culling and occluding cull. This visible set is view-dependent since it is different if the viewing direction is changed. Later on, these visibility test will be used in levels of details surface reconstruction which are described in the coming chapter.

5.5 Summary

In this chapter, we first point out that a visibility test is essential to speedup reconstruction on demand. Then, two culling algorithms have been described – frustum culling and occlusion culling. Both algorithms can significantly reduce the original dataset into a better visible subset, which is small in size. In the following chapter, we make use of these visibility techniques to perform levels of detail reconstruction.
Chapter 6

Levels-of-Detail reconstruction for ROD-TV

6.1 Introduction

The fundamental concept of Levels-of-Detail (LOD) can be summed up in Figure 6.1. Before reconstructing a surface, we need to figure out a potential visible set. This set is closely related to the viewer. If a viewer is close to the object, then high resolution representation is shown. On the other hand, if the viewer is farther away, then less detailed representation is selected. Due to the distance, the low resolution version looks approximately similar to the high resolution. In this way, a significant speed up can be expected and this is the general idea on the LOD.

In general, LOD algorithms consist of three major parts, namely, generation, selection and switching. LOD generation is the part where a spectrum of a dataset is created with different details. The Octree hierarchical data structure discussed in Chapter 4 can be used to generate the desired number of LOD. The selection mechanism chooses a detailed level of a dataset based on a particular criteria, such as the projected area on the image plane or viewer distance in the world space. This mechanism is presented in Section 6.4. Finally, changing details from a higher resolution to a lower one is termed as LOD switching, which is done by traversing the Octree at various heights. Details on the LOD switching will be described in Section 6.3. In Section 6.5, surface reconstruction on selected LOD tokens are performed, which is similar to original tensor voting algorithm. Experimental results and summary are drawn in Section 6.6 and Section 6.7 respectively.

97
Figure 6.1 - The fundamental concept of LOD

(A) A complex dataset is simplified. In high resolution, the dataset consists of 10,242 tokens. Nevertheless, it contains 272 tokens only in a less detailed representation. (B) Showing the general idea on the LOD. Given camera/viewer position, a highly detailed dataset is shown when its location is close to the camera/viewer. However, the dataset resolution decreases as the distance from the camera/viewer increases.

6.2 LOD Generation

ROD-TV uses a hierarchy of Octree [FREITAG 99] [RUBIN 80] for visibility culling, levels of detail, and on-the-fly local surface reconstruction. Reorganizing dataset in a hierarchy is so-called LOD Generation. Chapter 4 describes a method of constructing Octree from a large volume of data. An example of the Octree multi-resolution is shown in Figure 6.1. Some properties of the Octree are stated in the following as a reference for the LOD switching and LOD selection.
The root of the tree captures the coarsest resolution of the dataset, whereas the finest resolution can be found in the leaf nodes.

- Each node of the tree contains its size and a token. The depth of the node can be determined from the node size.

- Each parent node in the Octree has eight children. All of them are of equal size and a so-called Octant.

- Octree is regular partitioning in 3-D space. We can pre-sort the octant in either front-to-back or back-to-front order during tree traversal.

6.3 LOD Switching

Once the hierarchy has been constructed as in the previous section, the following algorithm is used for selecting visible tokens given a particular camera position:

The input to the algorithm is a hierarchical dataset which is described in the previous section. The output is a visible token set. The whole algorithm is running in a recursive manner starting from the tree root until a leaf node is reached or terminated in a particular condition, such as a failure in some of the visibility tests.

Given a camera position and dataset in a 3-D space, the dataset resolution increases as a viewer moves closer to it. An example is given in Figure 6.1. The changing appearance can be achieved through toggling the tree at different depths. The higher tree depth you go, the more detailed dataset you get. This is the main inspiration of the LOD switching. In Section 6.3.1, a general hierarchical traversal is demonstrated. Two add-on components, visibility analysis and levels of detail analysis, are described in Section 6.3.2 and Section 6.3.3 respectively.
TRAVERSEHIERARCHY(node)
{
    if ( node is not or partial visible\(^1\) ) then
        skip this branch of the tree
    else if ( node is a leaf node ) then
        VisibleTokenSet = OUTPUTTOKEN( node )
    else if ( node is good enough in current LOD\(^2\) ) then
        VisibleTokenSet = OUTPUTTOKEN( node )
    else
        for each child in children( node ) do
            TRAVERSEHIERARCHY( child )
        end for
    end if
}

1 - visibility analysis
2 - levels of details analysis

Algorithm 6.1 - Hierarchical Traversal for Rod-Tv
The input is hierarchical dataset. The output is visible token set. The algorithm runs in a recursive manner in a top-down fashion. Two analyses are performed during traversal. They are visibility analysis and levels of details analysis.

6.3.1 General Hierarchical Traversal

TRAVERSEHIERARCHY(node)
{
    if ( node is a leaf node ) then
        VisibleTokenSet = OUTPUTTOKEN( node )
    else
        for each child in children( node ) do
            TRAVERSEHIERARCHY( child )
        end for
    end if
}

Algorithm 6.2 - General Hierarchical Traversal
Tokens in the dataset are organized in the hierarchical data structure. To query a set of tokens from the hierarchy, a general hierarchical traversal algorithm is used. Algorithm 6.2 demonstrates this idea. It starts from the root of the tree, if the current node is the leaf node, then it is added into a visible token set. Otherwise, the function will be recursively applied to its child if necessary. The running path is based on the depth-first manner.

In order to facilitate occlusion culling, we prefer to traverse the tree in front-to-back order instead of back-to-front order. It means that the closest octant to the camera is going to be visited before the furthest one.

---

Figure 6.2 - Front-to-Back Traversal in 1-D Binary Tree

(A) Two tokens are shown in 1-D scene. They are separated by plane 1 whose normal is pointing towards token B. (B) The 1-D binary tree representation on the sample scene. The token which has the same region with the plane normal is positioned in the right sub-tree and vice versa. (C) Front-to-back hierarchical traversal at camera position 2. (D) Front-to-back hierarchical traversal at camera position 1. Both (C) and (D) are inorder traversal.

Traversing a tree from front to back order is virtually identical to a conventional binary tree's inorder traversal. The only difference is the definition of which of the children is to be the first node visited. Figure 6.2 depicts a sample 1-D scene of two tokens. Its binary tree
representation is shown and tokens are separated by a plane. There are
two viewing poses, namely camera pose 1 and camera pose 2. When we
are at camera pose 1, the inorder hierarchical traversal will return a set
{A, B}. The nearest token is placed at the beginning of the set and the
farthest token is placed at the end of the set. If we change the viewing
position from pose 1 to pose 2, then the inorder traversal will return a set
{B, A}. Details on the front-to-back traversal method can be found in
Algorithm 6.3.

```
TRAVERSE_HIERARCHY_FRONT_TO_BACK(node)
{
    near = viewer[poses] dot node[plane_normal] ≥ 0;
    if near then
        TRAVERSE_HIERARCHY_FRONT_TO_BACK( node[Rchild] )
        VisibleTokenSet = OUTPUT_TOKEN( node )
        TRAVERSE_HIERARCHY_FRONT_TO_BACK( node[Lchild] )
    else
        TRAVERSE_HIERARCHY_FRONT_TO_BACK( node[Lchild] )
        VisibleTokenSet = OUTPUT_TOKEN( node )
        TRAVERSE_HIERARCHY_FRONT_TO_BACK( node[Rchild] )
    end if
}
```

Algorithm 6.3 - Front-to-Back Hierarchical Traversal in 1-D Binary Tree
It is similar to inorder traversal in 1-D binary tree. In standard inorder traversal, the order is either LVR or
RVL [Kruse 99]. Nevertheless, in our algorithm, the region which is close to the viewer is visited earlier. It
can be either L or R depending on the current plane orientation.

In a 2-D scene, Quadtree is used instead of Binary Tree. Our Quadtree
consists of two levels of Binary Tree. Each node in the Quadtree is
separated by two planes – one plane in each domain. An example is shown
in Figure 6.3. Plane X separates the whole scene into two halves and is
placed on the first level of the Quadtree. Plane Y1 and Plane Y2 separate
the remaining parts and are placed on the second level of the Quadtree. To perform Front-to-Back hierarchical traversal, the technique is the same as the Binary Tree's Front-to-Back inorder traversal. For the 3-D scene, Octree is used. Each node of the tree is separated by three planes and three levels of a Binary Tree are generated. To carry out Front-to-Back inorder traversal, it is the same as the 1-D and 2-D cases – the closest octant is visited first and the furthest octant is visited last. Finally, the sorted token set is produced given a fixed viewing point.

Figure 6.3 - Front-to-Back Traversal in 2-D Quadtree

(A) Four tokens are shown in 2-D scene. They are separated by 3 planes. The plane X divides the scene into two halves. The plane Y1 and Y2 separate the remaining in order to let one token in each Quadrant. (B) The 2-D Quadtree representation. '+' means the region have the same direction as the plane normal and vice versa. (C) Front-to-Back hierarchical traversal at camera position 1. The sorted token set is {A, B, D, C}. (D) Front-to-Back hierarchical traversal at camera position 2. The sorted token set is {B, A, C, D}.
6.3.2 Hierarchical Traversal with Visibility Culling

```
TRAVERSEHIERARCHY(node)
{
    if ( node is not or partial visible\(^1\) ) then
        skip this branch of the tree
    else if ( node is a leaf node ) then
        VisibleTokenSet += OUTPUTTOKEN( node )
    else
        for each child in children( node ) do
            TRAVERSEHIERARCHY( child )
        end for
    end if
}
```

\(^1\) - visibility analysis

Algorithm 6.4 - Hierarchical Traversal with Visibility Cull

In Chapter 4, two visibility algorithms, frustum culling and occlusion culling, are discussed. The general idea of using these algorithms is to generate a better visible set for a particular viewing direction. Therefore, a large amount of unimportant tokens are being culled after the visibility test. In our hierarchical traversal algorithm [Algorithm 6.4], these visibility analyses are included in order to speed up the whole surface construction process. In the following, a quick summary on both frustum culling and occlusion culling is given:

**Visibility Algorithm I – Frustum Culling**

As we recurse the Octree hierarchy, we cull nodes that are not visible. Frustum culling is performed by testing each Octant against the planes of the viewing frustum. If the Octant lies outside, it and its sub-tree are discarded and not processed further. If the sphere lies entirely inside the
frustum, this fact is noted and no further frustum is attempted on the children of the node.

*Visibility Algorithm II – Occlusion Culling*

We also perform occlusion culling during the tree traversal. Since we make use of image-space technique, every Octant is projected onto a 2-D viewing plane and corresponding pixels are recorded. If a pixel on the plane is being marked, then it means that there exists another Octant which projects the same pixel as before. In other words, two Octants are being interested along a particular line of sight. In this case, a depth test is performed – keeping the closer Octant and removing the furthest one. The depth value of Octants can be obtained by enabling the z-buffer in the hardware.

### 6.3.3 Hierarchical Traversal with Levels of Detail

Given a hierarchical dataset, we can extract different resolution levels by traversing the tree at different heights. When a viewer walks closer to the dataset, a higher resolution is returned. On the other hand, when a viewer moves farther away from the dataset, a lower resolution is returned. Different resolutions of the dataset retrieval are termed as LOD Selection. In general, there are two ways of performing it. The first one is based on a range that uses the distance from the viewpoint to the dataset as an evaluation. The second one is based on the projected area of the bounding volume on the image plane. Either one can be added into our hierarchical traversal as a levels of detail analysis as shown in Algorithm 6.5. The details of each method are described in the following Section.
TRAVERSHEIRARCHY(node)
{
    if ( node is not or partially visible\textsuperscript{1} ) then
        skip this branch of the tree
    else if ( node is a leaf node ) then
        VisibleTokenSet + OUTPUTTOKEN( node )

    else if ( node is good enough in current LOD\textsuperscript{2} ) then
        VisibleTokenSet + OUTPUTTOKEN( node )

    else
        for each child in children( node ) do
            TRAVERSHEIRARCHY( child )
        end for
    end if
}

1 - visibility analysis
2 - levels of details analysis

Algorithm 6.5 - Hierarchical Traversal with Levels-of-Detail

6.4 LOD Selection

6.4.1 LOD Selection I: Range-Based Method

A common metric for selecting a LOD is to associate the different LODS of a dataset with different ranges. The most detailed LOD has a range from zero to some user-defined RANGE-0. The next LOD has a range from RANGE-0 to RANGE-1, where RANGE-0 < RANGE-1. If the distance to the dataset is greater than or equal to RANGE-0 and less than RANGE-1, then this LOD is used, and so on. Examples of four different LODS with their ranges are illustrated in Figure 6.4.
Figure 6.4 - Range-Based Method in LOD Selection

It shows that how range range-based LOD work. RANGE-0, RANGE-1 and RANGE-2 are user-defined value in the object space. If the object-viewer distance is less than RANGE-0, dataset with level 0 (named LOD 0) is rendered. Similarly, if the distance is greater than RANGE-0 but less than RANGE-1, then LOD 1 is rendered. The detailed level in the dataset decrease as the distance increases.

6.4.2 LOD Selection II: Screen-Based Method

Another common metric for LOD selection is the projected area of the bounding volume. The desired resolution of a dataset can be determined by the number of pixels covered on the image plane. We project a sphere instead of an octant in order to neglect the axis alignment problem. Note that the approximate number of pixels is sufficient for the screen-based method. Supposing that an octant has size $2r$ with its centre at $C$, and then a fitting sphere will get a radius $r$ and centre at $C$. Even though the sphere is not fully fitted into an octant, it makes an acceptable estimation of the bounding volume projection. Figure 6.5 illustrates how the size of the projection of the sphere is halved when the distance is doubled and also describes the concept for estimating the projected radius of the sphere on the image plane. The equation to estimate the projected radius is shown in Equation 6.1.
\[ p_{\text{projected\_radius}} = \frac{n_{\text{plane\_distance}} r_{\text{sphere\_radius}}}{d_{\text{view\_direction}} \cdot (c_{\text{sphere\_centre}} - v_{\text{eye\_position}})} \]

Equation 6.1 - The estimation of the radius of the projected sphere

We start to traversal the dataset from the root of the hierarchy. If the rooted sphere projection on the current token exceeds the user-defined threshold on the image plane, the recursion is terminated. Otherwise, the projection continues to be carried out on its children recursively until the leaf node is reached. The deeper the tree is traversed, the higher the details of a dataset are achieved. This method is entirely making the LOD selection based on the 2-D image screen and it is a kind of image-space algorithm. The advantage of using an image-space algorithm is that a single threshold is used compared to the range-based method. Therefore, it is implemented in our hierarchy traversal algorithm as levels of detail analysis.

Figure 6.5 - Projected-Based Method in LOD Selection
(A) The size of the projection is halved if the distance from the viewer is doubled. (B)
We project a sphere instead of octant on the image plane. The sphere with radius $r$ is centred at $C$ and located in front of the image plane, which has distance $n$ apart from the eye. The estimated projected radius is equal to $p$ and its estimated projected area is equal to $\frac{1}{2}p^3$.

6.5 Surface Reconstruction on Demand

Given a predefined viewing position, a group of visible tokens is generated after the LOD selection on the hierarchical dataset. Then, surface reconstruction is performed on these tokens through a process of tensor voting. As we change the viewing position, the corresponding token group also changes. Therefore, it implies that it is a view-dependent surface reconstruction. The advantage of performing view-dependent surface reconstruction is that we can only concentrated on the visible token surfaces and thus the whole process can be noticeably sped up.

6.5.1 Token Neighbourhood

Since we do not require pre-processing, the first goal is to infer a surface normal at each token position, which is unknown. With this aim, we consider its neighbourhood. The size of its neighbourhood is related to the size of the voting field kernel. Figure 6.6 illustrates a token neighbourhood in a sample 2-D scene under different voting kernel size.

6.5.2 Token Communication

An unknown normal at each token position can be estimated within its neighbourhood by using a process of sparse tensor voting. Once the normal is estimated, dense tensor voting can be carried out to generate a dense feature map. Surface patches are attained from this map through a standard operation of non-maximal suppression. Sparse and dense tensor voting are grouped together as a token communication.
Figure 6.7 illustrates the idea of the sparse tensor voting. Suppose the normal in token $p$ is unknown. Within $p$'s neighbourhood, there are two candidates, namely $q$ and $r$. The normal of token $p$ can be estimated through a sparse tensor voting on its neighbourhood. The tensor votes received at $p$ are accumulated, by summing up the votes’ second order moment into the covariance matrix. For example, let $p$ receive a total of $k$ stick votes in its neighbourhood, after voting with the 3-D stick voting field. Denote a stick vote by $[v_x \ v_y \ v_z]^T$. The accumulated tensor at $p$

$$T_{\text{symmetric}} = \begin{bmatrix} \sum_k v_x^2 & \sum_k v_x v_y & \sum_k v_x v_z \\ \sum_k v_y v_x & \sum_k v_y^2 & \sum_k v_y v_z \\ \sum_k v_z v_x & \sum_k v_z v_y & \sum_k v_z^2 \end{bmatrix}$$

is estimated the normal direction and saliency can be extracted by eigen-decomposition.
6.5.3 Vote Interpretation

We perform eigensystem analysis on the second order symmetric tensor $T_{\text{symmetric}}$. $T_{\text{symmetric}}$ can be decomposed into the equivalent eigensystem with eigenvalues $\lambda_{\max} \geq \lambda_{\mid\text{mid}} \geq \lambda_{\min} \geq 0$, and corresponding unit eigenvectors $\hat{e}_{\max}$, $\hat{e}_{\mid\text{mid}}$ and $\hat{e}_{\min}$. After rearranging, we get:

$$T_{\text{symmetric}} = (\lambda_{\max} - \lambda_{\mid\text{mid}})T_{\text{stick}} + (\lambda_{\mid\text{mid}} - \lambda_{\min})T_{\text{plate}} + (\lambda_{\min})T_{\text{ball}}$$

$$T_{\text{stick}} = \left(\hat{e}_{\max} \hat{e}_{\max}^T\right)$$

$$T_{\text{plate}} = \left(\hat{e}_{\max} \hat{e}_{\max}^T + \hat{e}_{\mid\text{mid}} \hat{e}_{\mid\text{mid}}^T\right)$$

$$T_{\text{ball}} = \left(\hat{e}_{\max} \hat{e}_{\max}^T + \hat{e}_{\mid\text{mid}} \hat{e}_{\mid\text{mid}}^T + \hat{e}_{\min} \hat{e}_{\min}^T\right)$$
Hence, we can define surface saliency by \( s = (\lambda_{\text{max}} - \lambda_{\text{mid}}) \), with \( \hat{e}_{\text{max}} \) indicating the estimated normal direction.

### 6.5.4 Surface Element Extraction

We sample votes collected at all the quantized locations (voxels) in \( p \)'s neighbourhood within a distance \( c \), and decompose the accumulated tensor at each voxel into a corresponding eigensystem that encodes the surface saliency information. A modified marching process is used to extract local surface patches, which approximates the loci of points having extrema in surface saliency \( s \) along the normal direction \( \hat{n} \), i.e. \( \frac{ds}{dn} = 0 \). Examples of inferring surface patches of token \( P \) are shown in Figure 6.8.

![Figure 6.8 - Surface Patches Extraction and Surface Connectivity](image)

(A) Inferring surface patches of token \( p \) along the loci of points having extrema in surface saliency. The surface connectivity \( C \) is equal to 2. (B) Surface connectivity \( C = 1 \). (C) Surface connectivity \( C = 0 \).
6.5.5 Surface Connectivity

![Diagram showing 0-connectivity and 1-connectivity](image)

Figure 6.9 - ROD-TV with 0-connectivity and 1-connectivity

A set of triangular patches are inferred after surface element extraction. In ROD-TV, the patches can be extracted in a small neighbourhood c x c x c, for some integer c ≥ 1. The number of output surface patches and time complexity is $O(c^3N) = O(N)$, where N is the number of visible tokens after LOD selection. c is called connectivity factor and the output mesh is therefore only partially connected. Figure 6.11 shows the surface extracted with different connectivity.

To elaborate on partial mesh connectivity, consider one extreme when c = 1. Only the exact local surface patch at the token will be inferred, if at all. This is equivalent to running QSplat in the quadrilateral primitive mode [Rusinkiewicz 00], which uses normal direction to orient individual quadrilaterals of a unit size. As shown in Figure 6.9, as the surface connectivity becomes larger, the inferred surface elements local to each token begin to touch each other, producing a better rendering that costs more processing time, as desired by affordable users.
6.5.6 Shading Model

(A) Triangular surface patches are in flat shading, which is constant intensity on each facet. (C) Interpolative shading. Facet shading intensity is computed based on its vertex normal.

Figure 6.10 - Shading in ROD-TV

A set of triangular patches are produced after a token surface extraction. We can visualize those triangle patches in two different shading models, namely flat shading model and interpolated shading model. For the flat shading model, a single intensity value is used to shade an entire triangular patch. The intensity value of the illumination is highly dependent on the viewer's position and the patch facing a normal direction. For the interpolative shading model, normal is estimated at each corner of the triangular patch, which is called per-vertex normal. Therefore, patch shading is not constant and its intensity value is calculated based on the pre-vertex normal. The final result is much more impressive as shown in Figure 6.10.
6.6 Experimental Results

Figure 6.11 - Experimental result on the surface connectivity of PLANE dataset

(A) Input is a PLANE dataset with 256 tokens. After visibility analysis, there are only 128 tokens remaining and we do surface reconstruction on them. (B) Surface connectivity with 0-neighbourhood (C) Surface connectivity with 1-neighbourhood (D) Surface connectivity with 2-neighbourhood (E) Surface connectivity with 3-neighbourhood

Figure 6.11 demonstrates the connectivity component in ROD-Tv. This experiment is used on a PLANE dataset as an input Figure 6.11.A. It is an artifact dataset with a sampling size equal to 4 in range 64x64, and therefore 256 tokens are generated. After a visibility analysis in ROD-Tv, there are 128 tokens remaining (256-128 = 128) and we reconstruct the surface from these visible tokens with different connective parameters. Figure 6.11.B shows the connectivity factor equal to 0. It means that only the local surface patch that is exactly at the token will be inferred and is equivalent to QSplat run in the quadrilateral primitive mode [Rusinkiewicz 00]. The difference between QSplat and ROD-Tv is patch normal computation. In the former case, normals are pre-computed and extracted from an existing mesh. In the latter, normals are inferred on-the-fly regardless the existence of geometrical mesh structure exists or not. As you can see in the figure, a fully connected mesh is produced from
a sparse dataset if appropriate connectivity is adopted. As shown in Figure 6.11.B-E, as we increase the surface connectivity, the inferred surface elements on each token touching its neighbourhood, produces a better rendering that costs more processing time. As a result, we can adjust the ROD-TV performance (rendering and processing) by choosing a different connectivity factor given any input dataset.

Figure 6.12 - Experimental result on HAND dataset (part 1)
(A) Original hand dataset with 38,220 tokens. (B) Initiating viewer position in 3-D space with view-object distance equal to 17.46 units. (C) After visibility analysis and levels of detail (LOD) analysis in hierarchical traversal, there are 6,029 tokens remaining.

[Remark: tree height 8 / 12 – rod level 12 – 8 – 4]

Figure 6.13 and Figure 6.16 show the surface reconstruction from a different viewing position, where the former is closer to the dataset than the later. In this experiment, we make use of the HAND dataset Figure 6.12.A, which consists of 38,220 tokens. Given a particular viewing position, let say 17.46 units away from the dataset Figure 6.12.B, a visible token set at the desired level of detail is determined after hierarchical traversal. There are 6,029 tokens remaining for further processing Figure 6.12.C. Figure 6.13.B shows these perceptible tokens in
our viewing screen. Surface reconstruction on these tokens is depicted in Figure 6.13.A. Its corresponding screen mode can be found in Figure 6.13.C. To grab more information on the ROD-TV, surface normal and its wire-frame are described in Figure 6.14. Surface normals show the patch orientation and the wire-frame shows the patch complexity. The higher the resolution on the reconstruction, the larger the number of patches is produced. To get a better rendering result, an interpolative shading model is preferable instead of flat shading. A quick snapshot is located in Figure 6.15. Figure 6.16 shows the rendering result when the viewing distance is 112 units away from the dataset. After visibility analysis and level of detail analysis in hierarchical traversal, 37,412 tokens are being removed. The remaining survival tokens are being sent to a rendering engine for surface reconstruction. Results can be found in Figure 6.17.

Figure 6.13 - Experimental result on HAND dataset (part 2)
(A) Surface reconstruction on visible tokens, which is generated from hierarchical traversal. It shows the result in world space. (B) Visible tokens (6,029 tokens out of 38,220 tokens) in screen space. (C) Visible tokens' surface patches in screen space. There are 35,441 triangular patches in total.
Figure 6.14 - Experimental result on HAND dataset (part 3)

(A) zoom-in view on the visible tokens (B) close-up view of the visible tokens’ (B) surface (C) surface and its normal (D) surface and its triangular wire-frame

Figure 6.15 - Experimental result on HAND dataset (part 4)

(A) zoom-out a little bit on the visible tokens (B) Flat shading (C) Interpolative shading
Figure 6.16 - Experimental result on HAND dataset (part 5)
(A) Initiating viewer position in 3-D space with view-object distance equal to 112 units. (B) After visibility analysis and levels of detail (LOD) analysis in hierarchical traversal, there are 808 tokens remaining. (C) Only frontier surfaces are being rendered. All the occluded or back faces are culled. There are 7,281 surface patches as a result. (D) Another view of the visible tokens

Figure 6.17 - Experimental result on HAND dataset (part 6)
Rendering in (A) Tokens (B) Wire-Frame (C) Flat Shading (D) Interpolative Shading mode
<table>
<thead>
<tr>
<th></th>
<th>ROD-TV view 1</th>
<th>ROD-TV view 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>View-object distance</td>
<td>17.46 units</td>
<td>112 units</td>
</tr>
<tr>
<td>Number of visible tokens</td>
<td>6,029 tokens</td>
<td>808 tokens</td>
</tr>
<tr>
<td>Number of triangular</td>
<td>35,441 patches</td>
<td>7,281 patches</td>
</tr>
<tr>
<td>patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td>Hi-resolution surface reconstruction</td>
<td>Lo-resolution surface reconstruction</td>
</tr>
</tbody>
</table>

Table 6.1 - Comparison of ROD-TV in two different views

Figure 6.18 - Experimental result on comparison between QSplat point rendering and ROD-TV in HAND dataset

(A) QSplat point rendering using sphere as a primitive (B) ROD-TV in low-resolution [view-object distance = 112 units and total number of surface = 7,281 triangular patches] (C) ROD-TV in high-resolution [view-object distance = 17.46 units and total number of surface = 35,441 triangular patches]

Figure 6.18 shows a comparison between QSplat and ROD-TV. Obviously, QSplat point rendering using sphere as a rendering primitive is highly dependent on scanning resolution. This problem worsens if we zoom-in to the dataset. Hence, a non-smooth surface is being returned. Compared to our ROD-TV, we not only produce a smooth surface no matter what
scanning resolution we have, but also on demand reconstruction is determined on-the-fly. Table 6.1 shows the comparison of ROD-TV from two different views.

6.7 Summary

In this chapter, we introduce Levels-of-Detail surface reconstruction. First, we briefly explain what is Levels-of-Detail (LOD) IS. Basically, it consists of three major parts; generation, selection and switching. For the LOD generation, a single dataset representation can be decomposed into multi-representation. We make use of a hierarchical data structure to achieve this. Secondly, two methods are introduced to select an appropriate LOD level given a specific viewing position; the Range-based method partitions the whole scene into several LOD regions. LOD Selection is therefore regional based. The screen-based method uses the projection areas as a selection criterion. Distant object gets small projected areas on an image plane resulting in low LOD level representation and vice versa. Finally, interpolative shading is used instead of flat shading in order to a produce smooth and attractive rendering result.
Chapter 7

Conclusion and Future Work

In this thesis, we designed ROD-Tv to make tensor voting more practical for large and imperfect 3-D data sets by adapting it to a continuum of view dependence, level of details and primitive connectivity. ROD-Tv has demonstrated its robustness to noisy data. We also show that our reconstruction methodology can indeed improve rendering quality. In particular, interpolative shading is readily achieved by inferring per-vertex normals on-the-fly. To visualize fine details, we are no longer upper-bounded by the scanner resolution, since ROD-Tv is capable of reconstructing missing details. Visibility culling and levels-of-detail control drives how much detail ROD-Tv needs to reconstruct, depending on demand. Thus ROD-Tv delivers high robustness and efficiency. Though ROD-Tv is a linear-time algorithm in the number of output tokens, it is slower than traditional point rendering, which does not perform surface reconstruction.

There are a number of areas for future work, including:

- Memory management for large models, particular terrains.
- Finding other compact multi-resolution schemes, e.g., using details and approximate coefficient to drive the appropriate levels-of-detail.
- Parallel Octree data structure for surface reconstruction under a multi-processor environment [Freitag 99].
- Progressive reconstruction on triangular surfaces
- Plotting ROD-Tv into handheld device, e.g., PDA and Pocket PC [3DGraphics].
- An Optimal range search for finding tensorial support.
Bibliography

[3DSCANNERS]
ModelMaker X. Laser stripe triangulation scanner. 3D Scanners Ltd. Commercial software product.
http://www.3dscanners.com/

[3DGRAPHICS]
http://www.research.ibm.com/vgc/pdaviewer/

[AURENHAMMER 91]

[AVIS 83]

[AYACHE 91]

[BOISSONNAT 84]

[BARNARD 82]

[BURT 83]

[CYBERWARE]

[CURLLESS 00]

[ECK 95]

[FAUGERAS 93]

[ FREITAG 99]

[GERSTNER 00]

[G RANLUND 95]

[GROSSMAN 98]

[GUY 96]
G. Guy. Inference of Multiple Curves and Surface from Sparse Data. USC-IRIS technical report, Department of Computer Science, The University of Southern California, 1996.

[HARTLEY 95]

[HOPPE 92]

[HOPPE 93]

[HOPPE 94]

[HOPPE 96]

[HOPPE 97]

[KNUTSSON 89]

[LEE 80]

[LEE 98]

[LONGUET-HIGGINS 81]

[LORENSEN 87]

[LUONG 96]

[MARR 76]

[MARR 79]

[MEIDONI 00]

[MEIDONI 03]
[NIKOLAI DIS 01]

[PFISTER 00]

[PRESS 96]

[RUBIN 80]

[RUSINKIEWICZ 00]

[RUSINKIEWICZ 01]

[SAMET 89A]

[SAMET 89B]
Hanan Samet. The design and analysis of spatial data structures. Addison-Wesley, 1989.

[ SCHROEDER 92]

[SOLOMON 01]

[SPARR 93]

[TANG 98]

[TANG 01]

[TANNER 98]

[TONG 01]

[TONG 02]

[TORR 97]

[WESTIN 94]

[WILLIAMS 83]

[ZWICKER 01]