Acknowledgements

I thank my advisor and chairman of my thesis committe, Prof. Gérard Medioni, for his guidance and advice throughout my studies at USC. His direction is like the fundamental stick field described in this thesis: while he identifies a “preferred” direction for me to pursue, this direction is not rigid at all, since it can decay with “proximity and higher curvature,” that is, within the tensor voting formalism, there is still a lot of freedom, and much detail still needs to be worked out, which are part of the fun as well as challenge in doing research with Gérard.

I also thank Prof. Ram Nevatia, Dr. Ulrich Neumann, Dr. Chris Kyriakakis, and Dr. Antonio Oretega, for finding the time to serve on my committee, and providing invaluable suggestions to my work, and their encouragement.

Many thanks also go to all the members of the computer vision group at IRIS, for all the discussion and corridor chat, whether they lead to fruitful outcome in my research, or simply some fun diversion from the hectic life as a Ph.D. student.
# Contents

Acknowledgements ii  
List Of Tables viii  
List Of Figures ix  
Abstract xiii  

## 1 Introduction  
1.1 Motivation and contribution ........................................ 2  
1.1.1 2-D sketch versus layered description .......................... 3  
1.1.2 Contribution of this thesis ....................................... 5  
1.2 Literature survey ................................................... 6  
1.3 Overview of the basic formalism ................................... 11  
1.3.1 Representation .................................................... 11  
1.3.2 Data communication .............................................. 13  
1.3.3 Overall approach ................................................ 14  
1.4 Outline of the dissertation ......................................... 14  
1.5 Notations .................................................................... 17  

## 2 Review of the Basic Formalism  
2.1 Overview of the salient structure inference engine ............... 20  
2.2 Tensor representation ................................................ 20  
2.2.1 Second order symmetric tensor .................................. 21  
2.2.2 Tensor decomposition .............................................. 23  
2.2.3 Uniform encoding .................................................. 25  
2.3 Tensor communication ................................................ 25  
2.3.1 Token refinement and dense extrapolation .................... 25  
2.3.2 The fundamental 2-D stick kernel ............................... 27  
2.3.3 Derivation of the stick, plate, and ball kernels ................ 29  
2.3.4 Implementation of tensor voting ................................. 31  
2.4 Feature extraction .................................................... 37  
2.4.1 Surface extremality ................................................ 38
2.4.2 Curve extremality .................................................. 39
2.4.3 Maximal feature extraction ..................................... 39
2.5 Complexity ............................................................. 40
2.6 Summary ............................................................... 41

3 Feature Extraction Algorithms ................................... 42
3.1 2-D curve extremality ............................................... 43
3.2 3-D surface extremality ............................................ 44
  3.2.1 Definitions ......................................................... 44
  3.2.2 Discrete version .................................................. 45
3.3 3-D curve extremality .............................................. 47
  3.3.1 Definitions ......................................................... 47
  3.3.2 Discrete version .................................................. 48
3.4 Maximal feature extraction ...................................... 51
3.5 Details of the modified Marching algorithms .................. 52
  3.5.1 Properties of an extremal surface patch ..................... 52
  3.5.2 Properties of an extremal curve segment ...................... 55
  3.5.3 Vector alignment .................................................. 59
3.6 Space and time complexity ....................................... 60
3.7 Summary ............................................................... 61

4 Feature Inference in 3-D ............................................ 63
4.1 Previous work ......................................................... 63
  4.1.1 Surface fitting ................................................... 64
    4.1.1.1 Deformable model approach ............................... 64
    4.1.1.2 Physics-based approach .................................. 65
    4.1.1.3 Functional minimization ................................ 65
    4.1.1.4 Computational geometry approach ....................... 66
    4.1.1.5 Level-set approach ....................................... 67
  4.1.2 Curve fitting ..................................................... 67
  4.1.3 Limitations of these methods ................................ 69
4.2 Elementary cases .................................................. 69
  4.2.1 Oriented and non-oriented input ............................... 69
  4.2.2 Oriented input ................................................... 70
    4.2.2.1 Surface inference from surfels ........................... 72
    4.2.2.2 Curve inference from curvels ............................ 74
    4.2.2.3 Surface inference from curvels ............................ 76
    4.2.2.4 Curve inference from surfels ............................ 77
  4.2.3 Non-oriented input ............................................. 79
    4.2.3.1 Surface inference from points ............................ 80
    4.2.3.2 Curve inference from points ............................. 82
4.3 Other applications and examples ................................ 84
  4.3.1 Fault detection ................................................. 86
4.3.2 Medical imagery

4.3.3 Terrain reconstruction

4.3.4 3-D object modeling from photographs

4.4 Summary

5 Extension to the Basic Formalism: Feature Integration

5.1 Motivation and overall strategy

5.2 Cooperative computations and hybrid voting

5.2.1 Feature inhibitory and excitatory fields

5.2.2 Curve trimming by inhibitory junctions

5.2.3 Curve extension toward excitatory junction

5.2.4 Surface trimming by inhibitory curves

5.2.5 Surface extension toward excitatory curve

5.2.6 Final curves and junctions from surfaces

5.3 Time and space complexity analysis

5.4 Limitations and proposed solution

5.5 Results

6 Application of the Basic Formalism: Visualization

6.1 Shock wave detection

6.1.1 Problem mapping and results

6.1.2 Insufficiency of iso-surface extraction

6.2 Vortex extraction

6.2.1 Vortex segmentation

6.2.2 Vortex extraction

6.2.3 Vorticity lines in Taylor vortices

6.3 Summary

7 Inference of Curvature Information: Second Order Geometric Properties

7.1 Previous work on curvature estimation

7.2 Overview and motivation

7.3 Estimation of the sign of principal curvatures

7.3.1 Representation of sign of curvature

7.3.2 Vote collection

7.3.3 Geometric interpretation

7.4 Principal direction estimation

7.4.1 Vote definition

7.4.2 Vote collection

7.4.3 Geometric interpretation

7.5 Curvature-based stick kernels

7.6 Evaluation

7.6.1 Accuracy of labeling

7.6.2 Grouping by curvature
7.6.3 Accuracy of estimated principal directions ............................. 164
7.6.4 Quality of surface fitting .................................................. 164
7.7 Results on real data ............................................................ 169
7.8 Feature integration using curvature information .......................... 173
  7.8.1 Overall approach ........................................................... 173
  7.8.2 Surface-curve integration ................................................. 176
  7.8.3 Surface-junction integration ............................................. 179
  7.8.4 Curve-junction integration .............................................. 180
  7.8.5 Results on integrated structure inference ............................. 180
7.9 Summary ............................................................................... 182

8 Higher Dimensional Tensor Voting and its Applications 189
  8.1 n-Dimensional Tensor Voting ............................................... 190
    8.1.1 Tensor representation .................................................... 190
      8.1.1.1 Second order symmetric tensor in n-D ......................... 190
      8.1.1.2 n-D tensor decomposition ....................................... 193
      8.1.1.3 Uniform encoding .................................................. 194
    8.1.2 n-D tensor communication ............................................. 194
      8.1.2.1 Token refinement and dense extrapolation ...................... 195
      8.1.2.2 Derivation of n-D voting fields ................................ 195
    8.1.3 Implementation of n-D tensor voting ................................ 197
  8.2 n-D feature extraction ....................................................... 200
    8.2.1 n-D hypersurface extremality ...................................... 201
    8.2.2 Discrete version .......................................................... 203
    8.2.3 Extraction of (n – 1)-D entity ....................................... 203
  8.3 Application to epipolar geometry estimation ............................. 205
    8.3.1 Previous work in epipolar geometry estimation .................. 206
    8.3.2 Review of epipolar geometry ........................................ 208
    8.3.3 Our approach ............................................................... 209
  8.4 Tensorization and local densification ..................................... 211
  8.5 Extrema detection and outlier rejection .................................. 213
    8.5.1 8-D extremality ............................................................ 213
    8.5.2 Grouping of detected zero crossings ................................. 214
  8.6 Other issues ......................................................................... 215
  8.7 Space and time Complexity .................................................. 217
  8.8 Results ................................................................................ 218
    8.8.1 Aerial image pairs ......................................................... 218
    8.8.2 Image pair with widely different views .............................. 219
    8.8.3 Image pairs with non-static scenes ................................... 219
  8.9 Summary .............................................................................. 220


9 Conclusion and Future Work

9.1 Summary ................................................. 226
9.2 Future Research .......................................... 227
  9.2.1 The scale issue ...................................... 227
  9.2.2 Multiresolution ...................................... 227
  9.2.3 Dealing with Images ................................ 228
  9.2.4 \( n \) Dimensions ................................. 228
  9.2.5 The tensor voting formalism ..................... 229

Appendix A

  \( N \)-Dimensional Marching ............................ 230
  A.1 Cell splitting ......................................... 230
    A.1.1 Splitting a 2-cell ............................... 231
    A.1.2 Splitting a 3-cell ............................... 232
    A.1.3 Splitting an \( n \)-cell ........................... 232
  A.2 Simplex contouring .................................... 233
    A.2.1 Contouring a 1-simplex .......................... 233
    A.2.2 Contouring a 2-simplex .......................... 234
    A.2.3 Contouring a 3-simplex .......................... 234
    A.2.4 Contouring an \( n \)-simplex .................... 234
  A.3 Contour triangulation ............................... 235

Appendix B

  Red-Black Tree ........................................... 236
List Of Tables

1.1 Notations .................................................. 17
5.1 Space and time complexities of feature integration (without using curvature) 121
7.1 Sign of curvature vote collected at $O$ ........................................ 151
7.2 Accuracy on curvature labeling ............................................. 163
8.1 Generalization of 2-D tensor voting to $n$-D ................................... 191
8.2 Space and time complexities .................................................... 217
8.3 Summary of the results on epipolar geometry estimation ................. 221
# List Of Figures

1.1 Geometric representation of second order symmetric tensor ........................................ 13
1.2 Overall approach ........................................................................................................ 15

2.1 Overview of the essential components of tensor voting .............................................. 19
2.2 An ellipsoid with its eigensystem .............................................................................. 22
2.3 A general, second order symmetric 3-D tensor ......................................................... 24
2.4 The fundamental 2-D stick kernel ............................................................................. 28
2.5 The design of fundamental 2-D stick kernel ............................................................. 28
2.6 3-D stick kernel .......................................................................................................... 32
2.7 3-D plate kernel .......................................................................................................... 33
2.8 3-D ball kernel ........................................................................................................... 34
2.9 3-D ball kernel (continued) ....................................................................................... 35

3.1 Curve extremality in 2-D ......................................................................................... 43
3.2 3-D surface extremality ............................................................................................ 45
3.3 Flow chart of extremal surface extraction algorithm .................................................. 46
3.4 Projection of curve saliency onto the $\hat{u}$-$\hat{v}$ plane ............................................. 48
3.5 Flow chart of extremal curve extraction algorithm ..................................................... 49
3.6 Further illustration of SingleSubVoxelCMarch ......................................................... 50
3.7 Seven cases of zero crossings .................................................................................... 53
3.8 Ten typical configurations of zero-crossing patch ..................................................... 54
3.9 Illustration of SingleSubVoxelCMarch .................................................................... 56
3.10 Linear interpolation gives a sub-voxel approximation where $\frac{ds}{du} = \frac{ds}{dv} = 0$ .... 58
3.11 Vector alignment ....................................................................................................... 60

4.1 Annotated version of the basic flowchart of Figure 2.1 ............................................... 71
4.2 Surface inference from surfels .................................................................................. 72
4.3 Four-point basic plane ............................................................................................. 73
4.4 Four-point basic ellipsoid ....................................................................................... 74
4.5 Four-point basic saddle ........................................................................................... 75
4.6 Flowchart of curve inference from curvels .............................................................. 76
4.7 Curve inference from curvels ................................................................................... 77
4.8 Another example of curve inference from curvels ..................................................... 78
4.9 Flowchart of surface inference from curvels ........................................................... 79
4.10 Surface inference from curvels
4.11 Flowchart of curve inference from surfels
4.12 Curve inference from curvels
4.13 Flowchart of surface inference from points
4.14 Surface inference from points
4.15 Flowchart of curve inference from points
4.16 Curve inference from points
4.17 Another example of curve inference from points
4.18 A surface is more salient a curve as we scale along the z-axis
4.19 Fault detection from seismic data
4.20 Inferred surface description for the femur data set
4.21 Data acquisition by a laser digitizer
4.22 Three slices of the original input, with mostly accurate but erroneous data
4.23 Data validation
4.24 Dental restoration
4.25 More results on dental restoration
4.26 Input data for terrain reconstruction
4.27 Automatic integration of detected ridge surfaces and crestine lines
4.28 More views of the DTM result
4.29 Layered description for Capital Building
4.30 Layered description for Labor Department
4.31 Inferred surfaces for the point set obtained from six stereo pairs

5.1 Inferring integrated high-level description
5.2 Detected features are not well localized
5.3 Incorrect curve may be obtained by considering the CMap alone
5.4 Overview of cooperative algorithm.
5.5 Illustration of overall strategy
5.6 Given initial curves and junctions, an incidence graph is constructed
5.7 Inference of the most probable extension
5.8 Surface extension
5.9 “Leakproof surface”
5.10 Irrelevant votes generated by the stick kernel
5.11 Plane and sphere
5.12 Three orthogonal planes
5.13 Triangular wedge
5.14 Inference of integrated surface, curve, and junction description from stereo

6.1 Velocity field of Blunt fin
6.2 Blunt fin.
6.3 Density field and the extracted λ-shock.
6.4 Histogram of density values on λ-shock
6.5 Consecutive snapshots of the wavy Taylor vortices)
8.7 Arena ......................................................... 222
8.8 Gable ....................................................... 223
8.9 House ......................................................... 223
8.10 Game-1 ....................................................... 224
8.11 Game-2 ....................................................... 225

A.1 Examples of $n$-cells and $n$-simplexes ($n = 1, 2, 3, 4$) ................. 231
A.2 Splitting a 2-cell (square) into four 2-simplexes (triangles) ............... 231
A.3 Splitting a 3-cell (cube) into 3-simplexes (tetrahedrons) ................. 232
Abstract

This dissertation is founded on the basic Tensor Voting Formalism. It provides a unified computational framework, making use of the continuity constraint to generate layered descriptions in terms of surfaces, regions, curves, and labeled junctions, from sparse, noisy, binary data in 2-D or 3-D. The method is non-iterative, does not depend on initialization, robust to spurious outlier noise, and the only free parameter is the size of the neighborhood, or the scale of analysis, which is indeed a property of visual perception.

In this thesis, feature extraction from tensor data is first studied. A number of new feature extraction algorithms is designed and implemented. Coherent features, such as a hole-free triangulation mesh, and a curve consisting of connected and oriented curve segments, are typical outputs of these algorithms.

While the basic formalism provides excellent results for smooth structures, it only detects discontinuities but does not localize them. A methodology for feature integration is proposed. This extended system is applied in a variety of visualization problems, and very encouraging results are obtained. The feature integration is further upgraded by the use of second order, curvature information, which is absent from the basic formalism.

The tensor voting formalism is also generalized to higher dimensions, and the 8-D version is applied to solve the problem of epipolar geometry estimation. Given a set of noisy point correspondences in two images as obtained from two views of a static scene without correspondences, even in the presence of moving objects, our method extract all good matches while rejecting all outliers.

The proposed theory and the implementation consolidate the existing tensor voting foundation. Promising further research and development in a wide variety of applications, and in any dimensions, are possible.
Chapter 1

Introduction

A stable and automatic computer vision system should generate compact scene description from one or more images. Owing to the projective nature of imaging, this problem is under-constrained. Vision researchers are hence confronted with the challenge of deriving the sufficient (description) from the insufficient (clues), and it is often complicated by the fact that outlier noise is unavoidable even in many state-of-the-art measurement phases.

Tensor Voting, a computational framework for feature extraction and segmentation, has emerged from the computer vision group at USC since the early 1990s [35, 72, 78]. Over the past several years, this approach has demonstrated promise and success on a number of computer vision problems, such as inference of segmented features from sparse and noisy 2-D and 3-D data [36], inference of surfaces and curves from stereo [71], and detection of motion discontinuities [31]. A detailed description of the computational framework can be found in the book by Medioni, Lee, and Tang [78].

This line of research was initiated by Guy and Medioni [36]. They proposed a feature extraction scheme based on a vector voting procedure. Issues of feature inference in 2-D and 3-D were addressed.
Lee and Medioni [72] then formalized the methodology into the tensor voting formalism. In essence, the methodology is grounded on two elements, tensor calculus for data representation, and voting for data communication, which together provide a unified framework for the robust inference of multiple curves, surfaces, regions and junctions from any combination of points, segments, and surface patch elements.

This dissertation is founded on this basic tensor voting formalism. Although the basic formalism has shown great promise, it shows some limitations. A number of components are missing from the basic theory. Therefore, the theme of this thesis to (1) extend, (2) augment, and (3) generalize the basic formalism in order to make the underlying theory more complete. The advancement of the theory made in this thesis results in an expanded applicability of the formalism to other problem domains. As a result, important progress has been made. We shall describe in detail the above contributions in this dissertation.

In this introductory chapter, we first discuss the motivation and goals behind this research in section 1.1. Then, in section 1.2, we review representative literature. Section 1.3 outlines the tensorial approach. Section 1.4 gives an overview of this dissertation. Finally, notations used in this dissertation are given in section 1.5.

1.1 Motivation and contribution

In order to reduce the complexity of computer vision problems, the most common approach used by vision researchers is the “divide-and-conquer” approach: the general and difficult vision problem is divided into a number of smaller sub-problems. These smaller sub-problems, such as matching, surface fitting, and removal of noise and outliers, are addressed independently. An intermediate representation, \(2\frac{1}{2}D\) sketch, is usually produced,
from which 3-D reconstruction is performed. Indeed, the $2\frac{1}{2}$-D sketch representational framework, which was presented by Marr in [76], represents the general reference paradigm for solving machine vision problems [78].

### 1.1.1 $2\frac{1}{2}$-D sketch versus layered description

The use of the $2\frac{1}{2}$-D sketch is appropriately summarized in [78]:

“At the heart of Marr’s representational framework is the intermediate representation called the $2\frac{1}{2}$-D sketch, which is a viewer-centered description of the visible surfaces (for which each pixel is assigned a single label). It serves as the main stepping stone toward the recovery of three-dimensional, object-centered description of object shapes and their spatial organization from images. This simplified representation, together with the modular approach to problem solving, seems to provide a handle to solve the difficult task of deriving scene description from images. In particular, many aspects of various vision problems can be formulated in the standard functional optimization framework, which can then be solved using well-known mathematical techniques. Accordingly, the main focus of computer vision research has been on finding the ‘right’ functional to optimize for each particular aspect of the vision problem. Complex situations are handled by incorporating additional criteria into the optimization framework.”

The limitations of the $2\frac{1}{2}$-sketch are, therefore, evident:

1. Since the $2\frac{1}{2}$-sketch is a viewer-centered description, it imposes additional and unnecessarily viewpoint dependent constraints. These constraints can be harmful because they may not have to be satisfied in the final solution. Specifically, in image segmentation, since each pixel is assigned a single label, a *partition* of the image will result. A single visible surface, if partially occluded, may be represented by several regions of the partition.
2. The $2\frac{1}{2}$-D sketch is harmful in dealing with multiple images. For instance, in stereo, a depth value is computed for every pixel. Depth discontinuities occur at the boundaries of overlapping visible surfaces. In motion grouping and segmentation, an optical flow vector is estimated at each pixel in the image. Again, discontinuities occur at the boundaries where visible surfaces overlap. In both cases, more constraints are needed to resolve these viewpoint-dependent features, whereas the results obtained are often a fragmented version of the desired one.

Therefore, while a proper enforcement of continuity constraint is needed (as observed by Marr and summarized in his “matter is cohesive” principle), exceptions to this rule, such as depth and motion discontinuities, should also be properly handled.

Hence, the desirable intermediate representation should be one of an object-centered description representing overlapping layers of visible surfaces and curves. The use of a viewpoint independent representation is essential to the proper implementation of the continuity and uniqueness constraints. A layered representation of visible curves and surfaces should be used so that the representation will not be overloaded with both shape and viewpoint information.

The next question we want to ask is: “how can we obtain a layered description from images?” In other words, a computational framework is needed. It turns out that through the use of tensor voting, we can effectively infer segmented, layered descriptions in terms of multiple surfaces, curves, regions, and junctions. In [78], we substantiated this statement by demonstrating the use of tensor voting for inference of layered descriptions from stereo, shading, and for other early to mid-level vision problems. Here, we also propose future research directions, several of which have been translated into the contributions of this thesis.
1.1.2 Contribution of this thesis

In this dissertation, we make important advances based on the the basic tensor voting formalism. The main contributions are outlined as follows:

1. Extension of the basic formalism

   In this thesis, non-trivial extensions are made to the basic formalism. They include the design and implementation of

   (a) feature extraction algorithms

   (b) feature integration

2. Application of the extended formalism to a range of visualization applications.

3. Augmentation of the basic formalism

   We augment the basic formalism with the capability of inferring second-order geometric properties, or curvature information. Using the augmented formalism, the feature integration process is upgraded, resulting in a more unified approach and cleaner methodology by the use of curvature information.

4. Higher dimensional tensor voting

   We generalize the basic formalism to $N$-dimensions for $N \geq 3$, and use it in applications, such as the epipolar geometry estimation.

We have made some important advances, and the results of this dissertation are recognized by the related scientific community, leading to publications in [101, 102, 103, 104, 105, 106].
1.2 Literature survey

In this section, we outline the representative approaches to address early to mid-level vision problems. Since this research also deals with other related problems, a review of related work will accompany the sections where the problems are addressed.

Problem formulation predominates the design of any algorithms. Here, we classify the major approaches found in computer vision literature into the following five “schools” of formulation, as in the book by Medioni et al. [78]. A more thorough description, comparison, and analysis of these approaches can be found there.

Regularization

In regularization approaches, early vision problems are identified as ill-posed inverse problems. Therefore, extra knowledge is needed to constrain the search space. Usually, a procedure that minimizes a functional is employed for the search of an optimal solution. However, as described earlier in this chapter, there is a discontinuities aspect inherent in the smoothness constraint commonly imposed in (early) vision. This aspect is difficult to express along with the smoothness in a functional optimization framework.

For instance, the Mumford-Shah model represents discontinuity by using a discrete set, thus incompatible with the continuous functional representation of the smoothness constraint.

Blake and Zisserman [9] have developed a weak membrane model that explicitly deals with discontinuities. Their method iteratively solves for the labeling of data and the model in a functional optimization framework. The complexity and the heuristic nature of their
approach indeed reflect the inherent difficulties involved in characterizing discontinuities in a regularization framework.

Zhao, Osher, Merriman and Kang propose the level-set approach in [127]. The main idea is to describe a curve as the zero level set of a function of higher dimension. Then, instead of evolving the curve, the evolution of the higher dimensional function is considered. Zero levels are extracted as the results. One advantage of this method is that it allows changes in topology, which has proved to be a powerful tool for many applications in physics, geometry, and computer vision.

**Consistent labeling**

In consistent labeling, computer vision problems are cast as one of finding the set of consistent labels for all the image pixels. There are three types of consistent labeling proposed: (1) discrete relaxation labeling [40, 41, 87], (2) continuous relaxation labeling [23, 53], and (3) stochastic relaxation labeling [32], or specifically Markov Random Fields.

Regardless of the formulation of the problem, all solutions for consistent labeling involve an intrinsically iterative process. As in all iterative processes, the main issues in defining a relaxation process are initialization, updating, and stopping condition. The different formulations just provide different justifications for setting the functions and parameters. Nevertheless, the labeling framework makes it relatively easy to incorporate terms to account for discontinuities.
Clustering and robust methods

As the goal of computer vision is to infer structures from images, one can also formulate vision problems in a data analysis framework.

In computer vision, data analysis techniques are applied in two different manners. Clustering methods use statistics to explore the inherent tendency of a point pattern to form compact groups of points in multidimensional space. Robust techniques use data analysis formalism to perform parametric model fitting.

In clustering techniques, a single partition of points is created such that points in each cluster/partition share similar properties. Partitional clustering methods [57] can be divided roughly into
1. error-square clustering
2. density estimation clustering
3. clustering by graph theory

The reader may refer to a standard text, for example, Duda and Hart [39], for details on these standard clustering methods.

Recently, Shi and Malik [98] devised a clustering by graph technique called normalized cuts and have applied it to image segmentation [98], and motion segmentation and tracking [99]. Most of the clustering techniques are iterative and sensitive to the a priori choice of the number of clusters in the feature space. If this choice is very poor, the final partition of the space may be incorrect as well. This sensitivity is an important limitation in computer vision, since the number of significant feature properties is known a priori only in rare situation.

In robust techniques, parametric model fitting is used to infer relevant visual information. The models are most always approximations only. Moreover, real data often needs
to be represented by multiple models, but not a single model. On the other hand, outliers are frequently structured themselves, such that they obey the same model with a different parameter. While techniques such as Hough Transform [52] can be very robust to noise, the parametric formalism of these techniques make it hard to deliver the same robustness in solving early vision problems.

**Artificial neural network approach**

One approach to solve the computer vision problem is inspired by our own biological, visual system. Since the objective of computer vision is to replicate our visual capabilities artificially, the computer vision problem might as well be solved if we can simulate vision on a computer, by using structures that neurologists have found. The connectionist approach to early vision makes use of an artificial neural network which simulates the features of a neuron.

A neural network is characterized by the large number of weighted connections between a large number of very simple processing (switching) elements, or neurons.

While there are methods that derive these weighted connections empirically [90, 91], the emphasis of connectionist approach has been on learning internal representation automatically. The absence of explicit models or confidence intervals makes reliability assessment of these methods difficult.

The work of Grossberg and Mingolla [34] describes a theory and neural network implementation of surface perception. It consists of two systems, the boundary contour system (BC) and the feature contour system (FC), and the interaction between them. We suspect that the limitations of the approach are due to the use of scalars only to accumulate information.
Perceptual saliency approach

Among the works that employ perceptual saliency approach, such as the works by Thornber and Williams [113], and by Sarkar and Boyer [94], Sha’ashua and Ullman [96] were the first to propose the use of a saliency map to guide the perceptual grouping process. A saliency map, derived from the input 2-D image, is a dense map where the value of each pixel corresponds to the degree of “perceptual importance,” such as smoothness, length, or and constancy of curvature. Their method uses an incremental optimization scheme to avoid the exponential complexity involved in picking subsets from a large set of data. A local operator chooses the optimal continuation from a given segment. However, this method cannot handle large gaps, and can be fooled by erroneous segments along a correct curve. Moreover, their method is iterative, running on a locally connected network.

Guy and Medioni [36] proposed the vector voting methodology to address the problem in 2-D and 3-D inference. This method makes use of perceptual constraints to group input features, reconstruct the underlying surfaces, and detect surface discontinuities explicitly, all at the same time. This is achieved by a convolution-like process where each input is aligned with predefined dense vector mask to produce dense feature maps. Surfaces, 3-D space curves, and junctions, which correspond to local extrema in these features maps, are derived from these maps.

Recently, Lee and Medioni [72] formalized Guy and Medioni’s methodology into tensor voting. This work combines the representational power of a tensor (introduced by Knutsson et al. [61] in the context of signal processing for computer vision), and the computational efficiency of voting, for salient structure inference. A unified computational
framework for dealing with various input types was derived. One advantage of this approach is that, although the problem of deriving the best scene description is itself an optimization problem, tensor voting does not perform iterative searching. Solution emerges directly from the data set, although it is not known what explicit functional is being optimized. This formulation therefore avoids most problems associated in iterative optimization. This method is non-iterative, depends on no critical threshold, allows any number of objects in the scene, each with any genus, and the only free parameter is the scale of analysis, which is a parameter of human perception ability.

1.3 Overview of the basic formalism

In this section, we give an overview of the tensor voting approach. A more in-depth review is given in chapter 2. The foundation of the formalism is grounded on the use of tensors for representation (section 1.3.1), and voting for data communication (section 1.3.2), and the implementation of the smoothness constraint in tensor fields.

1.3.1 Representation

In [72], two broad classes of representation for visible curves and surfaces are identified: global and local representations.

Global representations use parametric functions to capture surface and curve information. Parametric model fitting works well if there is no orientation discontinuity nor outliers. Otherwise, as demonstrated in numerous attempts, parametric modeling may fail. Worse, errors caused by outliers, and errors caused by orientation discontinuities, are often indistinguishable.
Local representations, on the other hand, are more general as they are capable of describing shapes in a uniform encoding. The derivation of global curve and surface is possible, and in fact even significantly simplified, if an adequate local representation is available which describes overlapping layers of visible surfaces and curves sufficiently [72].

Based upon the “matter is cohesive” property, a visible curve or surface is smooth everywhere, except at locations where discontinuities occur. Such discontinuities are locus of points where

- two (resp. three or more) surface patches intersect, resulting in a curve (resp. point) junction, or
- two or more curve elements intersect, resulting in a point junction

A point on a smooth structure has high certainty about its orientation (i.e., normal or tangent direction), while a point at a junction has high orientation uncertainty. Therefore, a desirable local representation should be able to encode variations of orientations, should adequately represent the continuum between these two extremes.

It turns out that a second order symmetric tensor possesses this property, as illustrated in Figure 1.1. This particular type of tensor can be geometrically visualized as an ellipsoid. The shape of the tensor encodes (un)certainty of orientation. For example, to encode absolute orientation certainty, the ellipsoid is essentially a stick. To encode absolute orientation uncertainty, the corresponding ellipsoid becomes a sphere.

The size of the second order symmetric tensor encodes its importance, or more precisely, feature saliency. For example, the longer the stick, the higher the confidence level that the corresponding point should lie on a smooth structure.
Figure 1.1: Geometric representation of second order symmetric tensor

Note that due to symmetry, the second order tensor representation of an orientation \( \mathbf{v} \) is the same as that of the orientation \( -\mathbf{v} \). To capture orientation direction, which is a first order orientation information, we make use of first order tensors, which are commonly known as vectors. The data representation hence includes both a first order tensor for encoding polarity saliency, and a second order tensor for encoding orientation saliency.

### 1.3.2 Data communication

The use of tensors enables the encoding of local variations of orientations. In the work by Guy and Medioni [36], they first proposed the use of the following statistical measure in a vector-voting scheme: a large number of orientation estimations is collected by fitting simple model in a local neighborhood. By analyzing the consistency of all the collected orientation estimations, and also the amount of support, the type of feature, as well as its corresponding saliency, can be inferred simultaneously. This technique is related to Hough Transform, but more powerful since feature saliency measure is also derived from the voting scheme. The incorporation of tensor as representational scheme in a voting computational scheme, proposed by Lee and Medioni in [72], provides a unified computational framework. The inference of salient features from a set of points, curve elements, surface elements, or any combination of them, is possible.
1.3.3 Overall approach

Figure 1.2 summarizes the basic tensor voting methodology, depicting the 3-D system as an example. The terms introduced below are defined and explained in Chapter 2.

Typically, the system accepts as input a noisy point set, with or without orientation information. The input is first encoded as a set of perfect stick, plate, and/or ball tensors.

Then, tensor voting is applied to the input in tensorization: each input site votes by propagating its information in a neighborhood, by using predefined voting fields. Each site collects all the tensor votes received. This voting process results in a set of generic tensors. The shape and orientation of the derived tensors indicate preferred surface normal as well as curve tangent information.

These tensors, inferred at each input site, vote again to fill the volume in densification. Dense feature maps, or saliency maps, are produced, from which salient features are derived. In 3-D, these features correspond to surfaces, 3-D space curves, and junctions.

In the flowchart shown in Figure 1.2, we also show the feature integration step which integrates detected features into a coherent 3-D model.

1.4 Outline of the dissertation

The organization of this dissertation is as follows: In Chapter 2, we review the tensor voting formalism.

Chapter 3 describes the algorithms for feature extraction in 2-D and 3-D.

Chapter 4 illustrates the tensor voting approach with basic examples.

Chapter 5 addresses feature integration, an extension to the basic formalism.
Figure 1.2: Overall approach
In Chapter 6, we apply the tensor voting methodology to visualization, and present results on applications such as flow visualization, vortex extraction, terrain visualization, and medical imagery.

In Chapter 7, we augment the underlying tensor voting theory with the capabilities for inferring second order geometric properties.

We generalize the computational framework into higher dimensions in Chapter 8, and apply this multidimensional version to the problem of epipolar geometry estimation.

Finally, we conclude this dissertation in Chapter 9, and propose future direction of this research.
1.5 Notations

Throughout this dissertation, we use the notation scheme of Table 1.1 unless otherwise stated.

<table>
<thead>
<tr>
<th><strong>Expression</strong></th>
<th><strong>Example</strong></th>
<th><strong>Type Setting</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>scalar</td>
<td>$s$</td>
<td>italics</td>
</tr>
<tr>
<td>vector</td>
<td>$\vec{v}$</td>
<td>overlined italics</td>
</tr>
<tr>
<td>unit vector</td>
<td>$\hat{v}$</td>
<td>italics with hat</td>
</tr>
<tr>
<td>zero vector</td>
<td>$0$</td>
<td>bold zero</td>
</tr>
<tr>
<td>vector array</td>
<td>${\vec{v}}$</td>
<td></td>
</tr>
<tr>
<td>point</td>
<td>$P$</td>
<td>capital italics</td>
</tr>
<tr>
<td>matrix</td>
<td>$\mathbf{R}$</td>
<td>capital bold</td>
</tr>
<tr>
<td>second order symmetric tensor</td>
<td>$\mathbf{T}$</td>
<td>capital bold</td>
</tr>
<tr>
<td>voxel</td>
<td>${(s_{i,j,k}, \hat{v}_{i,j,k})}$</td>
<td></td>
</tr>
<tr>
<td>complexity</td>
<td>$O(n)$</td>
<td>capital italics</td>
</tr>
</tbody>
</table>

Table 1.1: Notations
Chapter 2

Review of the Basic Formalism

This chapter reviews the core component of tensor voting: the *Salient Structure Inference Engine*. This is the fundamental aspect of the core theory described in a book by Medioni, Lee, and Tang [78].

In this chapter, more emphasis is given to the 3-D version. The input to this engine is a sparse set of tokens. These tokens encode position, but may also contain orientation information, and possibly a confidence measure. The output of this engine is a compact description of the input in terms of *layers* of curves, regions, surfaces, and junctions.

The outline of this chapter is as follows: Section 2.1 gives an overview of the approach. The tensor representation scheme is described in section 2.2. Section 2.3 explains the tensor communication scheme. Section 2.4 outlines the modified marching process for feature extraction. Section 2.5 analyzes the time and space complexity of the overall scheme. Section 2.6 summarizes the results presented in this chapter.
Figure 2.1: Overview of the essential components of tensor voting
2.1 Overview of the salient structure inference engine

Figure 2.1 illustrates the overall approach in this methodology. Each input token is first encoded into a second order symmetric tensor. For instance, if the input token has only position information, it is transformed into an isotropic tensor (a ball) of unit radius.

In a first voting stage, tokens communicate their information with each other in a neighborhood, and refine the information they carry. After this process, each token is now a generic second order symmetric tensor, which encodes confidence of this knowledge (given by the tensor size), curve and surface orientation information (given by the tensor orientations).

In a second stage, these generic 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, the domain space is digitized into a uniform array of cells. In each cell the tensor can be decomposed into elementary components which express different aspects of the information captured.

The resulting dense tensor map is then decomposed. Surface, curve, and junction features are then obtained by extracting, with subvoxel precision, local extrema of the corresponding saliency values along a direction. The final output is the aggregate of the outputs for each of the components.

2.2 Tensor representation

The goal is to extract geometric features such as regions, curves, surfaces, and the intersection between them. Here, we first summarize the various differential properties of these
entities, and their behavior at singularities. Then, we proceed to propose the use of tensor as the representation scheme.

Points can be represented by their coordinates.

For curve inference, a first order local description of a curve is given by the point coordinates, and its associated tangent. A second order description would also include the associated curvature.

For surface inference, a first order local description of a surface patch is given by the point coordinates, and its associated normal. A second order description would also include the principal curvatures and their directions.

It is important to note that a second order symmetric tensor only encodes first order differential geometry properties.

However, we do not know in advance what type of entity (point, curve, surface) a token may belong to. Furthermore, singularities occur when these entities intersect with each other. For example, when two smooth surfaces intersect, there is no associated normal information at the point of intersection. However, if we consider a small finite neighborhood around a curve junction, it has an associated tangent corresponding to the intersection curves.

2.2.1 Second order symmetric tensor

To capture first order differential geometry information and its singularities, a second order symmetric tensor is proposed, which captures both the information and its confidence, or saliency. Such a tensor can be visualized as an ellipse in 2-D, or an ellipsoid in 3-D.

Currently, there are two missing components in the basic formalism:

1. second order differential geometry information (curvature)
2. first order polarity information (tensor orientation)

In chapter 7, we augment our current basic formalism with (1) in order to make the theory more powerful. We have already done some work on (2), and we propose to complete the theory by including the first order tensor in the second edition of our book.

Intuitively, the shape of the tensor defines the type of information captured (point, curve, or surface element), and the associated size represents the saliency. For instance, in 2-D, a very salient curve element is represented by a thin ellipse, whose major axis represents the estimated tangent direction, and whose length reflects the saliency of the estimation. Therefore, it encapsulates both orientation and certainty information at the same time.

In 3-D, the second order symmetric tensor is an ellipsoid, which is fully described by its associated eigensystem, with three eigenvectors $\hat{e}_1$, $\hat{e}_2$, and $\hat{e}_3$, and the three corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ (see Figure 2.2), i.e.,

$$
\begin{bmatrix}
\hat{e}_1 & \hat{e}_2 & \hat{e}_3 \\
\end{bmatrix}
\begin{bmatrix}
\lambda_1 & 0 & 0 \\
0 & \lambda_2 & 0 \\
0 & 0 & \lambda_3 \\
\end{bmatrix}
\begin{bmatrix}
\hat{e}_1^T \\
\hat{e}_2^T \\
\hat{e}_3^T \\
\end{bmatrix}
$$

(2.1)

Rearranging the eigensystem, the 3-D ellipsoid is given by

$$(\lambda_1 - \lambda_2)S + (\lambda_2 - \lambda_3)P + \lambda_3B,$$

(2.2)
where $S$ defines a stick tensor, $P$ defines a plate tensor and $B$ defines a ball tensor, as shown in Figure 2.3.

These tensors define the three basis tensors for any general 3-D ellipsoid. By equation (2.1), a linear combination of these basis tensors defines any second order symmetric tensor.

### 2.2.2 Tensor decomposition

The eigenvectors encode orientation (un)certainties: surface orientation (normal) is described by the stick tensor, which indicates certainty along a single direction. Orientation Uncertainty is abstracted by two other tensors: curve junction results from two intersecting surfaces, where the uncertainty in orientation only spans a single plane perpendicular to the tangent of the junction curve, and thus described by a plate tensor. At point junctions where more than two intersecting surfaces are present, a ball tensor is used since all orientations are equally probable.

The eigenvalues, on the other hand, effectively encode the magnitudes of orientation (un)certainties, since they indicate the size of the corresponding 3-D ellipsoid.

Therefore, we can decompose a tensor token into the following 2-tuples $(s, \hat{v})$, where $s$ is a scalar indicating feature saliency, and $\hat{v}$ is a unit vector indicating direction:

\[
\begin{align*}
S &= \hat{e}_1 \hat{e}_1^T \\
P &= \hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T \\
B &= \hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T + \hat{e}_3 \hat{e}_3^T
\end{align*}
\]
Figure 2.3: A general, second order symmetric 3-D tensor
• *Surface-ness*: $s = \lambda_1 - \lambda_2$, and $\mathbf{\hat{v}} = \mathbf{\hat{e}}_1$ indicates the normal direction.

• *Curve-ness*: $s = \lambda_2 - \lambda_3$, and $\mathbf{\hat{v}} = \mathbf{\hat{e}}_3$ indicates the tangent direction.

• *Junction-ness*: $s = \lambda_3$, $\mathbf{\hat{v}}$ is arbitrary.

We have now explained the information encoded in a second order symmetric tensor, which consists of three independent elements and a measure of feature saliency.

### 2.2.3 Uniform encoding

The basic formalism allows the unified description of a variety of input feature tokens, such as points, curves elements or surface patch elements.

In 3-D, if the input token is a point, it is encoded as a ball tensor ($\lambda_1 = \lambda_2 = \lambda_3 = 1$), since, initially, there is no preferred orientation. If the input token is a curve element, it is encoded as a plate tensor ($\lambda_1 = \lambda_2 = 1, \lambda_3 = 0$). If the input token is a surface patch element, then it is encoded as a stick tensor ($\lambda_1 = 1, \lambda_2 = \lambda_3 = 0$), which is essentially a very elongated ellipsoid.

### 2.3 Tensor communication

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

#### 2.3.1 Token refinement and dense extrapolation

The input tokens are encoded as tensors as described in section 2.2.3. These initial tensors communicate with each other in order to
• derive the most preferred orientation information, or refine the initial orientation if given, for each of the input tokens (token refinement), and

• extrapolate the above inferred information at every location in the domain for the purpose of subsequent coherent feature extraction (dense extrapolation).

These two tasks can be implemented by a voting process, which involves having each input token aligned with predefined, dense voting kernels (or voting fields). This alignment is simply a translation followed by rotation. The dense voting kernels encode the basis tensors (i.e. stick, plate, and ball tensors in the 3-D case). The derivation of the voting kernels is given later in this section. This voting process is similar to a 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. By using an appropriate voting kernel, each token broadcasts the information in a neighborhood. The size of the neighborhood is given by the size of the voting kernel used. 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 kernels in 3-D. All voting kernels can be derived from the fundamental 2-D stick kernel.
2.3.2 The fundamental 2-D stick kernel

Voting fields of any dimensions can be derived from the 2-D stick tensor, and therefore it is called the fundamental 2-D stick kernel. Figure 2.4 shows this 2-D fundamental stick kernel.

The design of this field is given below. Note that, in 2-D, a direction can be defined by either the tangent vector, or the normal vector, which are orthogonal to each other. We can therefore define two equivalent fundamental fields, depending whether we assign a tangent or normal vector at the receiving site.

Here, we describe the normal version of the 2-D stick kernel. The tangent version is similar. Given a point at the origin with a known normal ($\mathbf{n}$), we ask the following question: for a given point $P$ in space, what is the most likely normal (at $P$) to a curve passing through $O$ and $P$, and normal to $\mathbf{n}$? (Figure 2.5 illustrates this situation.) We claim that the osculating circle connecting $O$ and $P$ is the most likely one, since it keeps the curvature constant along the hypothesized circular arc. For a detailed theoretical treatment, please refer to [35]. The most likely normal is given by the normal to the circular arc at $P$.

The length of the normal vector at $P$, representing the saliency of the vote, is inversely proportional to the arc length $OP$, and also to the curvature of the underlying circular arc. In doing so, both the proximity and the smoothness (or lower curvature) constraints are effectively encoded as the corresponding saliency measure, representing the likelihood of a smooth curve passing through that point.

Note that the connection as given by osculating circle becomes less likely if the angle subtended by $\mathbf{n}$ and $OP$ is less than $45^\circ$. Therefore, we only consider the set of orientations for which the angle defined above is not less than $45^\circ$. See Figure 2.4.
2 views of the 3-D plots of the field strength

Figure 2.4: The fundamental 2-D stick kernel

Figure 2.5: The design of fundamental 2-D stick kernel
In spherical coordinates, the decay of the 2-D stick takes the following form:

\[
\overline{DF}(r, \varphi, \sigma) = e^{-\left(\frac{r^2 + \varphi^2}{\sigma^2}\right)}
\]  

(2.6)

where \( r \) is the arc length \( OP \), \( \varphi \) is the curvature, and \( \sigma \) is the scale of analysis, the only free parameter in the basic formalism.

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 from 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.

Polarity information, which encodes the orientation of the tensor, is captured in the first order tensor.

### 2.3.3 Derivation of the stick, plate, and ball kernels

A second order symmetric tensor \( T \) can be parameterized by \( T(\lambda_1, \lambda_2, \lambda_3, \gamma, \beta, \alpha) \), where \( \gamma, \beta, \) and \( \alpha \) are angles of rotation about the \( z, y, \) and \( x \) axis respectively for defining its orientation. Without loss of generality, we consider below the derivation of the stick kernel oriented at \([1 0 0]^T\), and the plate with normal oriented at \([0 0 1]^T\). The other orientations can be obtained by a simple rotation in 3-D.

Using the above notation, we can express the three basis tensors by a parameterization, as follows:

\[
S(1, 0, 0, \gamma, \beta, \alpha)
\]  

(2.7)
\[ \mathbf{P}(1, 1, 0, \gamma, \beta, \alpha) \]  
\[ \mathbf{B}(1, 1, 1, \gamma, \beta, \alpha) \]  
(2.8)
(2.9)

Hence, \( \mathbf{S}(\cdot) \) describes the orientation \([1 0 0]^T\), \( \mathbf{P}(\cdot) \) describes a plane with normal \([0 0 1]^T\), and \( \mathbf{B}(\cdot) \) describes an isotropic tensor field.

Let us denote the fundamental 2-D stick kernel by \( V_F \).

\( \mathbf{S}(\cdot) \) is obtained by revolving the normal version of \( V_F \) 90° about the \( z \)-axis (denote it by \( V_F' \)), then integrating the contributions by rotating \( V_F' \) about the \( x \)-axis. The resulting voting field is a stick tensor which describes the orientation \([1 0 0]^T\) in world coordinates:

\[
\mathbf{S}(1, 0, 0, \gamma, \beta, \alpha) = \int_0^\pi V_F' d\alpha|_{\beta=0, \gamma=0} \]  
(2.10)

\( \mathbf{P}(\cdot) \) can be obtained by rotating \( \mathbf{S}(\cdot) \) about the \( z \) axis, and integrating the contributions, as follows:

\[
\mathbf{P}(1, 1, 0, \gamma, \beta, \alpha) = \int_0^\pi \mathbf{S} d\gamma|_{\alpha=0, \beta=0} \]  
(2.11)

\[
(2.12)
\]

\( \mathbf{B}(\cdot) \) can be obtained by rotating \( \mathbf{S}(\cdot) \) about the \( y \) and \( z \) axes, and integrating the contributions, as follows:
\[ \mathbf{B}(1, 1, 1, \gamma, \beta, \alpha) = \int_0^\pi \int_0^\pi \mathbf{S} \, d\beta d\gamma |_{\alpha=0} \]  

(2.13)

Figure 2.6, Figure 2.7, Figure 2.8, and Figure 2.9 illustrates the 3-D stick, plate, and ball voting kernels.

### 2.3.4 Implementation of tensor voting

To simplify the discussion, we restrict the tensor voting process to the token refinement case here, and in 2-D. The extension to the voting process of extrapolating directional estimates for feature extraction, and to higher dimensions, is straightforward.

The tensor voting process aggregates tensor contribution from a neighborhood of voters by using tensor addition. Tensor addition is implemented as follows. Suppose that we have only two input tokens. Initially, before any voting occurs, each token location encodes the local associated tensor information. Denote these two tensors by \( \mathbf{T}_{0,1} \) and \( \mathbf{T}_{0,2} \).

**Tensor Encoding:** The input can be a point or a tangent \( \hat{\mathbf{t}} \). A point is encoded as a ball, and a tangent as a stick, as follows.

- **Stick.** \( \lambda_1 = 1, \lambda_2 = 0 \), with \( \hat{e}_1 = \hat{\mathbf{t}} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \hat{e}_2 = \begin{bmatrix} -t_y \\ t_x \end{bmatrix} \).

- **Ball.** \( \lambda_1 = \lambda_2 = 1 \), with \( \hat{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \), with \( \hat{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \).

In both cases, the input is converted into a tensor \( \mathbf{T}_{0,j}, j = 1, 2 \) in our two point case, by
Figure 2.6: 3-D stick kernel
another cut of the plate kernel, showing the orientation of curve tangents

intensity-coded saliency of the 2 cuts

two views of the field strength of the cuts

Figure 2.7: 3-D plate kernel
Figure 2.8: 3-D ball kernel

Since the 3-D ball kernel is isotropic, only one half of the 3 corresponding components are shown.
Figure 2.9: 3-D ball kernel (continued)
\[
\mathbf{T}_{0,j} = \begin{bmatrix}
\hat{e}_1 & \hat{e}_2 \\
\lambda_1 & 0 \\
0 & \lambda_2 \\
\end{bmatrix} \begin{bmatrix}
\hat{e}_1^T \\
\hat{e}_2^T \\
\end{bmatrix}
\]
\[
= (\lambda_1 - \lambda_2)\hat{e}_1 \hat{e}_1^T + \lambda_2(\hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T)
\]
\[
= T_{0,j}^S + T_{0,j}^B
\]

(2.14)
(2.15)
(2.16)

where \(T_{0,j}^S, T_{0,j}^B\) are the respective stick and ball components. Both \(T_{0,j}^S, T_{0,j}^B\) are symmetric, positive semi-definite, \(2 \times 2\) matrices.

**Tensor Voting:** An input site \(j\) collects the tensor vote cast from the voter \(i\). This vote consists of a stick and a ball component.

- **Stick vote.** Let
  \[
  \begin{bmatrix}
  v_x \\
v_y
  \end{bmatrix}
  \]
  be the stick vote collected at site \(j\), which is cast by voter, which is cast by voter \(i\) after aligning the 2-D stick voting field (by translation and rotation) with the \(\hat{e}_1\) component of the tensor \(T_{0,i}\) at \(i\) (obtained in the tensor encoding stage). Then,

  \[
  T_{1,j}^S = T_{0,j}^S + (\lambda_1 - \lambda_2) \begin{bmatrix}
v_x^2 & v_x v_y \\
v_y v_x & v_y^2
\end{bmatrix}
\]

(2.17)

- **Ball vote.** Let \(T_B\) be the ball vote collected at site \(j\), which is cast by voter \(i\). Then,

  \[
  T_{1,j}^B = T_{0,j}^B + \lambda_2 T_B
  \]
Therefore, $T_{1,j} = T_{1,j}^S + T_{1,j}^B$ is obtained. Note that $T_{1,j}$ is still symmetric and semi-positive-definite, since $T_{1,j}^S$ and $T_{1,j}^B$ are both symmetric and positive definite. Hence, $T_{1,j}$ produced by the above is also a second order symmetric tensor.

**Tensor Decomposition:** After $T_{1,1}, T_{1,2}$ have been obtained, we decompose each of them into the corresponding eigensystem.

Note that the same tensor sum applies to the tensor voting process for extrapolating directional estimates, with the following changes:

- first, site $j$ may or may not hold an input token. For non-input site $j$, $T_{0,j}$ is a zero matrix.

- second, we do not propagate the ball component from voting sites, but only generate stick votes.

### 2.4 Feature extraction

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. Each voxel of these maps has a 2-tuple $(s, \hat{v})$, where $s$ is a scalar indicating feature saliency, and $\hat{v}$ is a unit vector indicating direction:

- **Surface map** ($SMap$): $s = \lambda_1 - \lambda_2$, and $\hat{v} = \hat{e}_1$ indicates the normal direction.

- **Curve map** ($CMap$): $s = \lambda_2 - \lambda_3$, and $\hat{v} = \hat{e}_3$ indicates the tangent direction.

- **Junction map** ($JMap$): $s = \lambda_3$, $\hat{v}$ is arbitrary.

These maps are dense vector fields which are then used as input to the extremal algorithms (chapter 3) in order to generate features such as surfaces, curves, and junctions.
The definition of point maximality, corresponding to junctions, is straightforward: it is a local maxima of the scalar value $s$.

We outline below the definition of surface and curve extremality, and provide full details of the methods in the next chapter.

### 2.4.1 Surface extremality

Let each voxel in the SMap hold a 2-tuple $(s, \hat{n})$ where $s$ indicates surface saliency and $\hat{n}$ denotes the normal direction. The vector field is continuous and dense, i.e., is defined for every point in 3-D space.

A point is on an extremal surface if its strength is locally extremal along the direction of the normal, i.e.,

$$\frac{ds}{d\hat{n}} = 0. \quad (2.19)$$

This is a necessary condition for 3-D surface extremality. 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 $\vec{g}$ as,

$$\vec{g} = \nabla s = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \end{bmatrix}^T \quad (2.20)$$

and project $\vec{g}$ onto $\hat{n}$, i.e.,

$$q = \hat{n} \cdot \vec{g} \quad (2.21)$$

Thus, an extremal surface is the locus of points with $q = 0$. 

38
2.4.2 Curve extremality

Each voxel in the CMap holds a 2-tuple \((s, \hat{t})\), where \(s\) is the curve saliency and \(\hat{t}\) indicates the tangent direction. This field is continuous and dense, in which \((s, \hat{t})\) is defined for every point \(P\) in 3-D space.

A point \(P\) with \((s, \hat{t})\) is on an extremal curve if any displacement from \(p\) on the plane normal to \(\hat{t}\) will result in a lower \(s\) value, i.e.,

\[
\frac{ds}{d\hat{u}} = \frac{ds}{d\hat{v}} = 0
\]

(2.22)

where \(\hat{u}\) and \(\hat{v}\) are two unit orthogonal vectors which define the plane normal to \(\hat{t}\) at \(P\). This is a necessary condition for 3-D curve extremality. A sufficient condition, which is used in implementation, is defined in terms of zero crossings in the \(\hat{u}\)-\(\hat{v}\) plane normal to \(\hat{t}\). Define

\[
R(\hat{t} \times \overline{g})
\]

(2.23)

where \(R\) defines a rotation to align a frame with the \(\hat{u}\)-\(\hat{v}\) plane, and \(\overline{g}\) is defined earlier.

By construction, an extremal curve is the locus of points with \(\overline{q} = 0\).

2.4.3 Maximal feature extraction

In the above sections, we described how to extract features from the vector maps decomposed from the tensor field obtained after the dense extrapolation stage of tensor voting.

Please refer to the flowchart of the tensor voting formalism as depicted in Figure 2.1 to integrate the feature extraction algorithms we have just described into the flow of the
whole formalism. Note that this chapter only outlines the definition of feature extremality. Details in implementation, including the extraction of maximal feature, will be described in Chapter 3. (Note that extremal feature can be maximal and minimal as well, and we are only interested in maximal feature that corresponds to maximal feature saliency.)

We now proceed to analyze the space and time complexity of the formalism.

### 2.5 Complexity

We analyze the space and time complexity in this section. Let

- \( n \) = number of input features (tokens)
- \( k \) = maximum dimension of 3-D voting kernel \( \approx 3\sigma \)
- \( s \) = total output surface size (in voxels)

For space complexity, the size of the vote store is directly proportional to size of the voting envelop, which is \( \Theta(s) \). Voting envelope is the union of finite and non-zero neighborhoods that contains all the input features. In practice, the input heap, of size \( \Theta(n) \), which is used for (sparse) input quantization is much smaller than the envelope. Also, the vote store needs to store features in sub-voxel precision. Therefore, the storage requirement for the input heap is not as substantial when compared with the vote store. In implementation, SMap, CMap, and JMap are actually embedded in the vote store. Therefore, the total space requirement of tensor voting is \( \Theta(s) \).

For time complexity, the token refinement step takes \( O(nk^3) \) time. Dense extrapolation takes \( O(sk^3) \). Vote interpretation takes \( O(1) \) time per site. Therefore, tensor voting runs in \( O((n + s)k^3) \) time in total.
The above analysis implies that tensor voting is a linear operation, both in time and space. We shall defer the details of the algorithms for feature extraction, whose time and space complexity is linear with the size of the output.

2.6 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 data, we produce features such as junctions, curves and surfaces.

The approach consists of three elements

- encoding the information using tensors,
- communication and computation using tensor fields,
- feature extraction using tensor decomposition and local marching algorithms.
Chapter 3

Feature Extraction Algorithms

In this chapter, we describe a methodology to extract salient features from the dense tensor map produced after the voting process for dense extrapolation, by formulating it as one of a modified version of the marching cubes algorithm [69]. We shall call this methodology the extremal feature extraction algorithms. The input to these algorithms is a set of dense vector maps, decomposed from the tensor map as obtained by tensor voting. The output is a compact layered description in terms of junctions, curves, and surfaces. Each output curve is connected and oriented; and each output surface is a hole-free triangulation mesh. Thus, the output is useful not only for visualization, but also as a compact coherent means suitable for further interpretation and processing.

In the following, we first describe the 2-D version in section 3.1, is then generalized into 3-D.

Sections 3.2 and 3.3 detail the 3-D version of extremal feature extraction. Section 3.4 explains the surfaces and curves produced by the above algorithms are in fact maximal features. Section 3.5 gives the details on the properties on the output extremal surfaces
and curves as produced by our algorithms. Section 3.6 gives the space and time complexity analyses of the feature extraction algorithms. Finally, section 3.7 summarizes this chapter.

### 3.1 2-D curve extremality

Before considering 3-D surface and curve extremality, let us consider the simpler, 2-D version first. Figure 3.1(a) depicts a synthetic 2-D circular potential field \( \{(s, \hat{v})\} \). Since it is circular, let \( \ell \) be a diameter joining \( A \) and \( B \), where \( \ell \) is normal to the corresponding tangents (arrows) at \( A \) and \( B \). We plot the field strength along the line \( \ell \), Figure 3.1(b), where two extremal points exist. We seek to link all such extremal points present in this field into a connected 2-D curve, approximated by a set of connected polyline segments (Figure 3.1(c)). Therefore, in the 2-D case, an extremal curve is defined by the locus of points for which field strength is locally extremal along the normal direction, i.e.,

\[
\frac{ds}{d\ell} = 0
\]  

(3.1)

![Figure 3.1: Curve extremality in 2-D](image)

(a) A synthetic circular field, (b) plot of field strength along \( \ell \), (c) the extremal curve.
This definition, being a necessary condition for 2-D curve extremality, is equivalent to projecting the field strength onto a line normal to \( \hat{v} \). In implementation, we discriminate a response satisfying Equation (3.1) as being extremal by \textit{zero crossing detection}, so that a 2-D extremal curve can be sufficiently characterized by a zero-crossing curve.

Note that this problem is not exactly equivalent to a conventional zero crossing curve extraction, as the sign (or the “inside” or “outside” in the surface case) is not required to be known in advance, since we assume having access to the tangent direction only. In the following, we generalize this notion to surface and curve extremality in 3-D.

### 3.2 3-D surface extremality

#### 3.2.1 Definitions

Let each voxel in the given vector field hold a 2-tuple \((s, \hat{n})\) where \(s\) indicates field strength and \(\hat{n}\) denotes the normal direction. Suppose the vector field is continuous, in which \((s, \hat{n})\) is defined for every point \(P\) in 3-D space. A point is on an \textit{extremal surface} if its strength \(s\) is locally extremal along the direction of the normal, i.e.,

\[
\frac{ds}{d\hat{n}} = 0
\]

(3.2)

This is a necessary condition for 3-D surface extremality. A sufficient condition, which is used in implementation, is defined in terms of zero crossings along the line defined by \(\hat{n}\) (Figure 3.2).
Figure 3.2: 3-D surface extremality
(a) a normal vector (with an imaginary surface patch drawn), (b) field strength along the normal, and (c) the derivative of field strength.

To this end, we compute the gradient vector $\mathbf{g}$ as,

$$\mathbf{g} = \nabla s = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \end{bmatrix}^T \tag{3.3}$$

and project $\mathbf{g}$ onto $\hat{n}$, i.e.,

$$q = \hat{n} \cdot \mathbf{g} \tag{3.4}$$

Therefore, an extremal surface is the locus of points for which $q = 0$.

3.2.2 Discrete version

In implementation, we can define the corresponding discrete version of $\mathbf{g}$ and $q$, i.e.,

$$\overline{g}_{i,j,k} = \begin{bmatrix} s_{i+1,j,k} - s_{i,j,k} \\ s_{i,j+1,k} - s_{i,j,k} \\ s_{i,j,k+1} - s_{i,j,k} \end{bmatrix} \tag{3.5}$$
\[ q_{i,j,k} = \hat{n}_{i,j,k} \cdot g_{i,j,k} \]  

(3.6)

Therefore, the set of all \( \{q_{i,j,k}\} \) constitute a scalar field which can be processed directly by the Marching Cubes algorithm [69]. Given a cuboid consisting of eight \( \{q_{i,j,k}\} \) of the vertex voxels, an abstract procedure which we shall call SingleSubVoxelMarch produces as output a zero-crossing patch (if it exists), which is a triangulation of the local surface patch for that cuboid. Detail construction of this procedure is given later.

![Flow chart of extremal surface extraction algorithm](image)

**Figure 3.3: Flow chart of extremal surface extraction algorithm**

*An adapted version of the Marching Cubes algorithm is abstracted as SingleSubVoxelMarch.*

Figure 3.3 shows the overall extremal surface algorithm. It picks the seed voxel whose \( s \) value is largest so far, computes a zero-crossing patch by using SingleSubVoxelMarch, and aggregates the surface in its neighborhood until the current \( s \) value falls below a low threshold. If there are multiple surfaces, it then picks the next available seed, and performs the same patch aggregation process until the \( s \) value of the next seed falls below a high threshold. A polygonal mesh is thus produced. Note that the thresholds are not critical and they are used solely for efficiency purposes and rejecting noisy features.
Despite the intricate details, SingleSubVoxelMarch is still an $O(1)$-time operation, and can be regarded as a plug-in component of the extremal surface algorithm. Owing to its complexity, we shall describe SingleSubVoxelMarch as a component in section 3.5.

### 3.3 3-D curve extremality

#### 3.3.1 Definitions

Each voxel in the given potential vector field holds a 2-tuple $(s, \hat{t})$, where the $s$ is field strength and $\hat{t}$ indicates flow or tangent direction. Suppose the field is continuous, in which $(s, \hat{t})$ is defined for every point $P$ in 3-D space. A point $P$ with $(s, \hat{t})$ is on an extremal curve if any displacement from $P$ on the plane normal to $\hat{t}$ will result in a lower $s$ value, i.e.,

$$\frac{ds}{d\hat{u}} = \frac{ds}{d\hat{v}} = 0$$

where $\hat{u}$ and $\hat{v}$ define the plane normal to $\hat{t}$ at $P$ (Figure 3.4).

This is a necessary condition for 3-D curve extremality. A sufficient condition, which is used in implementation, is defined in terms of zero crossings in the $\hat{u}$-$\hat{v}$ plane normal to $\hat{t}$.

Define

$$\bar{q} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T = R(\hat{t} \times \overline{g})$$

where $R$ defines a rotation which aligns a frame with the $\hat{u}$-$\hat{v}$ plane (shown in Figure 3.4, with its detailed construction given in section 3.5), and $\overline{g}$ is defined in Equation (3.3).
Field strength is projected onto a plane perpendicular to the tangent. When a change in signs occurs in the derivatives in both \( \hat{u} \) and \( \hat{v} \), a curve passes through the point.

cross product is used to project \( \vec{g} \) onto the \( \hat{u}-\hat{v} \) plane. \( \mathbf{R} \) is used to re-position the elements in this cross product, so that \( q_x \) and \( q_y \) correspond to the \( \hat{u} \) and \( \hat{v} \) components of Equation (3.7) respectively, and \( q_z = 0 \) after applying \( \mathbf{R} \). Therefore, by construction, an extremal curve is the locus of points for which \( \vec{q} = 0 \).

### 3.3.2 Discrete version

In implementation, we can define the corresponding discrete version of \( \vec{g} \) and \( \vec{q} \), i.e.,

\[
\overline{q}_{i,j,k} = \mathbf{R}(\hat{t}_{i,j,k} \times \overline{g}_{i,j,k})
\]  \hspace{1cm} (3.9)

Therefore, the set of all \( \{\overline{q}_{i,j,k}\} \) constitute a vector array, which can be processed by our novel adaptation to the Marching Cubes, abstracted as a procedure which we shall call
SingleSubVoxelCMarch. Given a cuboid, this procedure takes the eight \( \{ q_{i,j,k} \} \) computed at the vertex voxels as input, and produces an extremal point where an extremal curve passes through (if it exists) as output.

Figure 3.5 shows the overall extremal curve algorithm. It picks the seed voxel with \((s, \hat{t})\) whose \(s\) value is largest so far. Then, it

1. computes an extremal point (if it exists) by using `SingleSubVoxelCMarch` and,

2. predicts the next cuboid the curve will pass through by using the current directed curve segment (e.g. \((P_i - P_{i-1})\) in Figure 3.6), or \( \hat{t} \) if it is unavailable at the very beginning of curve tracing.

The marked components are further explained in Figure 3.6.
Figure 3.6: Further illustration of SingleSubVoxelCMarch

The above steps are repeated (i.e., the marked portion in Figure 3.5 which is further illustrated in Figure 3.6) until the current $s$ value falls below a low threshold. Denote this curve thus obtained by $C_1$. Then the algorithm returns to the seed and repeats the whole process above with direction $-\hat{t}$. Denote the curve thus obtained by $C_2$. It outputs

$$Reverse(C_2) \cup C_1 \quad \text{(3.10)}$$

as a connected and oriented extremal curve. If there are multiple curves, then it picks the next available seed and performs the same curve aggregation process until the $s$ value of the next seed falls below a high threshold. Note that these thresholds are only used for efficiency.
Owing to the complexity of detail, we describe SingleSubVoxelCMarch as a separate component in section 3.5. However, it is still an $O(1)$-time operation, and can be regarded as plug-in component of the extremal curve algorithm.

On close examination, our algorithm resembles the previously reported Predictor-Corrector scheme for extracting a vorticity line by Banks and Singer [3]. However, ours differs in two important aspects. First, by making use of saliency gradient, we do not need to perform the time consuming Lagrangian interpolation to produce the sub-voxel extremal point. Also, since the $u$-$v$ plane does not necessary intersect at exactly four points in a cuboid, their four-point Lagrangian interpolation scheme can only be regarded as a rough approximation of the true sub-voxel coordinates.

### 3.4 Maximal feature extraction

The above definition and discussion are for extremal feature extraction, in which the problem is transformed into one of the zero-crossing surface and curve extraction. However, we are only interested in extracting maximal features. For example, a maximal surface in 3-D is defined as the locus of points for which the surface saliency is maximal (c.f. extremal) along the normal at that point. A maximal curve can be defined similarly.

To distinguish a response as maximal or minimal, a plausible solution is to compute second-order derivatives. But, it is noted that second-order derivatives is quite error-prone. It is not a stable solution.

Here, we argue that the extremal surface patches and curve segments produced by the above algorithms are in fact maximal. In surface and curve extraction, a seed voxel with the maximum feature saliency is first chosen. A feature, such as a surface patch or a curve
segment, is produced from it. The feature produced at the seed voxel must therefore be a maximal feature.

Surface patches, or curve segments, are then aggregated according to connectivity. Therefore, unless at a discontinuity (which is detected by our methodology), since “matter is cohesive,” there is no reason to believe that any surface patch or curve segment as produced by the algorithms is minimal, except for those produced in the vicinity of discontinuities. This is owing to the violation of the smoothness constraint around discontinuity: during the tracing of a maximal feature, the locus may become a point of inflexion around a discontinuity, resulting in the emergence of minima.

Feature integration, to be described in Chapters 5 and 7, deals with the proper handling of discontinuities.

### 3.5 Details of the modified Marching algorithms

#### 3.5.1 Properties of an extremal surface patch

We show in the following: A local zero-crossing patch computed by SingleSubVoxelMarch, if it is not at the boundary, is connected to its neighboring patch in the adjacent cuboid to produce a global hole-free triangulation.

If the boundary of a zero-crossing patch computed at a cuboid face is not at the domain boundary, then by the construction of SingleSubVoxelMarch described next, this patch boundary is shared *exactly* by two adjacent voxels, or by two zero-crossing patches. This is because either a patch boundary is computed without ambiguity, or ambiguity is resolved in a deterministic way. Thus, all edges in the global triangulation are shared by exactly two zero-crossing patches, except for those edges at the boundary domain.
Hence, it remains to construct a deterministic SingleSubVoxelMarch. Consider the four voxels which constitute a face of a cuboid. Each voxel is labeled ‘+’ if $q_{i,j,k} \geq 0$ and ‘−’ otherwise. Hence there are $2^4 = 16$ possible configurations, which can be reduced to seven by rotational symmetry (Figure 3.7).

![Figure 3.7: Seven cases of zero crossings](image)

Ambiguities in configurations (6) and (7) are resolved using the method proposed in [83, 112]. The zero four crossings define two orthogonal (dotted) lines (in Figure 3.7). For example, the $x$-coordinate of the vertical dotted line is the mean of the $x$-coordinates of the hollow circles shown. Let the two lines intersect at $(x, y)$ inside the face, and let $q_1, q_2, q_3, q_4$ be the values of $q$ at the cuboid vertices. Then we let

$$v' = v(x,y) = (1-y)[(1-x)q_1 + xq_2] + y[(1-x)q_4 + xq_3] \quad (3.11)$$

be our disambiguity function. This definition of $v'$ resembles that of bilinear interpolation which intuitively aims at better connectivity [112]. If $v' \geq 0$, we choose configuration (7). Otherwise, (6) is chosen.

We examine each face of a candidate cuboid as above in order to extract a zero-crossing patch, which in fact is a set of closed loops (polygons) formed by ordering the segments obtained in each face (Figure 3.8).
Since there are at most two segments per face, the maximum number of segments per voxel is 12 (note that 10 and 11 are impossible). Also, since we restrict these segments to be part of a cycle, and a non-trivial cycle must have length at least 3, it further reduces the number of possibilities to 20, as shown in [112].

Some of the more common configurations are shown in Figure 3.8. (Note that a configuration with its rotationally symmetric counterparts are counted as one configuration.) Therefore, zero-crossing patches are produced unambiguously, implying that each segment of their boundary is shared by exactly one other segment of another patch boundary, unless at the domain boundary.

Note that SingleSubVoxelMarch will not produce any output in the vicinity of a surface orientation discontinuity (curve junction), where the differential property described in section 3.2 no longer applies. However, such a discontinuity results from the intersection between two extremal surfaces. We extract the two extremal surfaces away from this discontinuity, as described above, and then integrate the detected surface and curve junctions.
3.5.2 Properties of an extremal curve segment

In the following, we show that each local curve segment connecting two extremal points, computed by \texttt{SingleSubVoxelCMarch}, is connected to its neighboring curve segments, if these extremal points are not any of the endpoints of the extremal curve. Moreover, all curve segments must be non-intersecting.

By the following construction, \texttt{SingleSubVoxelCMarch} assigns the values of \( \{q_{i,j,k}\} \) deterministically without ambiguity. So there is either zero or one extremal point in any cuboid. Also, by construction of the extremal curve algorithm, the locus of extremal points is traced in strict order, and thus, the resulting polyline segments must be connected.

We give a valid construction for \texttt{SingleSubVoxelCMarch} as follows. Consider the eight voxels \( V_r \) with \((s, \hat{r})\), \(1 \leq r \leq 8\), which make up a cuboid. Define the cuboid tangent at the cuboid center, denoted by \( \hat{t} \), to be the mean tangent of the eight \( \hat{r} \)'s. We compute the 3-D sub-voxel coordinates of the point that an extremal curve with tangent \( \hat{t} \) passes through by the followings:

\textbf{STEP 1}:  
(a) Translate the unit cuboid to the world origin. Let \( T \) be the translation matrix.

(b) Compute \( \vec{g} \) for the eight \( V_r \)'s, using Equation (3.9).

(c) Compute the cuboid tangent \( \hat{t} \) by interpolating the aligned \( \hat{r} \)'s, \(1 \leq r \leq 8\). (Note: two tangents are aligned if their dot product is non-negative.) Therefore,

\[ \hat{t} = \sum_{r=1}^{8} \hat{r}/8 \]  
(3.12)
Thus, \( \hat{t} \) defines a \( \hat{u}\hat{v} \) plane through which an extremal curve with tangent \( \hat{t} \) passes. We assume that this plane passes through the cuboid center (Figure 3.9(b)).

### Figure 3.9: Illustration of SingleSubVoxelCMarch

**STEP 2:** (a) For each cuboid edge \((P_0^k, P_1^k), 1 \leq k \leq 12\), we compute the intersection point (if it exists) with the \( uv \) plane. Solving the corresponding ray-plane equation, an intersection point \( Q_k \) on a cuboid edge is given by the parameter \( u_k \) (Figure 3.9(b)):

\[
\begin{align*}
  u_k &= -\frac{\hat{t} \cdot P_0^k}{\hat{t} \cdot (P_1^k - P_0^k)} \\
  Q_k &= P_0^k + (P_1^k - P_0^k)u_k
\end{align*}
\]  

(3.13)  

(3.14)

If \( \hat{t} \cdot (P_1^k - P_0^k) = 0 \) or \( u_k < 0 \) or \( u_k > 1 \), there is no intersection.

(b) Order all intersection points \( \{Q_k\} \) so that they form a cycle. Since \( \{Q_k\} \) lie on the \( uv \) plane, this problem is equivalent to computing a 2-D convex hull for
\{Q_k\}. Several known algorithms are available in any standard algorithms text such as [15]. Let the ordered set be \(\{Q_1, \cdots\}\).

(c) Define a frame which aligns with \(u-v\) plane by a rotation matrix \(R:\)

\[
R = \begin{bmatrix}
\hat{x}^T \\
\hat{y}^T \\
\hat{z}^T
\end{bmatrix}
\]  

(3.15)

with

\[
\hat{z} = \hat{i}
\]  

(3.16)

\[
\hat{x} = Q_1/||Q_1||
\]  

(3.17)

\[
\hat{y} = \hat{z} \times \hat{x}
\]  

(3.18)

We then transform the ordered \(\{Q_k\}\) to frame \(R\). So for all intersections \(Q_k\), we assign

\[
Q_k \leftarrow RQ_k
\]

See Figure 3.9(b). Note that after applying \(R\) to \(Q_k\) as above, \((Q_k)_z\) will become zero.

Step 3: For each (ordered) intersection point \(Q_k\) which lies on a cuboid edge \((P_{0k}, P_{1k})\) connecting two voxels, we compute the \(\overline{q_k}\) w.r.t. the frame \(R\) (Figure 3.9(c)) as follows:
(a) Compute the interpolated gradient vector for \( Q_k \), denoted by \( \overline{g_k} \). Let the gradient vector at \( P^k_0 \) and \( P^k_1 \) be \( g^k_0 \) and \( g^k_1 \), respectively. If \( Q_k = P^k_0 + (P^k_1 - P^k_0)u_k \) is given by Equation (3.14), then by linear approximation, we have

\[
\overline{g_k} = g^k_0 + (g^k_1 - g^k_0)u_k \tag{3.19}
\]

(b) Compute \( \overline{q_k} \) for each \( Q_k \):

\[
\overline{q_k} = R(\hat{i} \times \overline{g_k}) \tag{3.20}
\]

Step 4: (Marching cycle) Now, \((\overline{q_k})_x\) corresponds to the \( \hat{u} \) component, and \((\overline{q_k})_y\) corresponds to the \( \hat{v} \) component of Equation (3.9), with \((\overline{q_k})_z = 0\). We march along the sides of the cycle in order given by the ordered set \{\( Q_k \)\}, and compute zero crossings (Figure 3.10). Because we approximate a zero crossing by linear interpolation, if there exists an extremal point, the positive and negative \((\overline{q_j})_x\) (resp. \((\overline{q_j})_y\)) should be linearly separable and thus four zero crossings, and subsequently two 2-D straight lines,

Figure 3.10: Linear interpolation gives a sub-voxel approximation where \( \frac{ds}{du} = \frac{ds}{dv} = 0 \)
will be produced. Their intersection corresponds to the extremal point in frame $\mathbf{R}$. Denote this intersection point in frame $\mathbf{R}$ by $P^R$.

Step 5: Transform $P^R$ back to the world frame $\mathbf{W}$, i.e.,

$$P^W = T^{-1}R^{-1}P^R$$ (3.21)

Both $T^{-1}$ and $R^{-1}$ are easy to compute since they are pure translation and rotation matrices, respectively. $P^W$ is the extremal point with sub-voxel precision through which an extremal curve will pass.

Curve intersection and branching are characterized by the presence of point junctions, which are not handled by SingleSubVoxelCMarch because they are curve orientation discontinuities (since the differential property no longer applies). Thus, no extremal point will be produced in the vicinity of junctions, which is the intersection of two or more converging extremal curves. Therefore, as described in Chapter 5, we propose to detect junctions and then properly localize them, using the extremal curves produced as initial estimates.

3.5.3 Vector alignment

An important issue in ensuring the consistency of the evaluation at cuboid voxels in both extremal algorithms is the alignment of the voxel vectors. In computing the covariance matrix during tensor voting, the original orientations of the input vectors are decoupled as they are represented as a stick, a second order symmetric tensor. Therefore, we need to
locally align the vertex vectors in the dense field \( \{(s, \hat{v})\} \) before applying them to Equations (3.6) and (3.9). The alignment involves a simple test of the sign of the dot product of all voxels against an arbitrary vertex voxel vector \( \hat{v} \), and flipping the vector if the sign is negative (Figure 3.11).

### 3.6 Space and time complexity

We give the analytic space and time complexity in this section. In the implementation, the set of candidate seeds are kept by a Fibonacci heap (F-heap), in which the seeds are prioritized by their associated saliencies. Insertion to and seed extraction from an F-heap is very efficient. For a more detailed amortized analysis of F-heap, see [15]. Let

\[
\begin{align*}
    f &= \text{maximum number of seed voxels} \\
    s &= \text{total output surface size (in voxels)} \\
    c &= \text{total output curve size (in voxels)} \\
    t &= \text{total output size (in voxels)}
\end{align*}
\]
Recall that the input to both extremal algorithms is a dense vector map whose size is a constant factor of the envelop surrounding the input features. Therefore, the input size of also of $\Theta(t)$.

**Extremal curve algorithm.** We preprocess the CMap in $O(t)$-time to build a Fibonacci heap (F-heap) for $O(\log t)$-time per seed extraction afterward. For extremal curve algorithm, computing zero crossings using SingleSubVoxelCMarch only involves a constant number of choices, so it takes $O(1)$ time to produce a point on an extremal curve (if any). Therefore, in the usual case where $f \ll c$, the extremal curve algorithm runs in linear time, i.e. at most $O(t + c)$.

**Extremal surface algorithm.** As above, seed extraction takes $O(\log t)$ time with preprocessing. Also, SingleSubVoxelMarch takes $O(1)$ time to compute a zero-crossing patch (if any) in a voxel. Therefore, if $f \ll s$, which is usually the case, the extremal surface algorithm runs in linear time, i.e. at most $O(t + s)$.

Note that the extremal algorithms are insensitive to choices of seeds since multiple seeds are kept by a prioritized list (so multiple curves and surfaces can be extracted as intended). If one seed fails to generate the feature (e.g. local extrema fails to exist because of numerical errors) another will do the job, as long as the high threshold is exceeded.

### 3.7 Summary

In this chapter, we have described how we extract curve and surface features from the dense tensor field produced after the voting process. We use a modified local marching process, which is more general than techniques used in conventional zero crossing curve or surface
detection because we only assume access to the direction but not the orientation, information. The methodology consists of the following elements:

- definition of curve and surface saliency extremalities
- zero crossing detection
- discrete implementation issues
Chapter 4

Feature Inference in 3-D

In this chapter, we illustrate the concepts described in previous chapters, by applying the basic formalism and the feature extraction algorithms, to perform inference of simple geometric structures.

In section 4.1.1, we first review related, previous work in surface and curve fitting, in 3-D. Then, in section 4.2, we first give an overview for the most general case, for which the input consists of a mixture of oriented and non-oriented data. Then we proceed to illustrate the formalism with specific, elementary examples. In section 4.3, we demonstrate our formalism with other more complicated examples, and in different application domains.

4.1 Previous work

In this section, we review related work on surface and curve fitting in 3-D. The inference of space curves and surfaces have been studied in various fields such as computer vision, computer graphics, and visualization. Various techniques have been proposed to deal with the problems of extracting specific features, such as surfaces or 3-D space curves.
4.1.1 Surface fitting

Much work has been done in surface fitting to clouds of points. They can be roughly classified into the following (not necessary exclusive) categories:

- Deformable models
- Physics-based approach
- Functional minimization
- Computational geometry approach
- Level-set approach

4.1.1.1 Deformable model approach

Terzopoulos et al. [111] proposed the deformable model approach, which attempts to deform an initial shape so that it fits a set of points through energy minimization. Kass et al. [59] proposed the use of similar model in 2-D. Pentland [88] proposed the automatic extraction of deformable part models. A part-based segmentation of an object is generated in which each part is described by using a deformable model representation. Boult and Kender [6] addressed the sparse data problem specifically. They analyzed and compared four theoretical approaches, and demonstrated a method using minimization over Hilbert spaces. Poggio and Girosi [89] formulated the single-surface approximation problem as a network learning problem. Blake and Zisserman [9] addressed similar problems dealing explicitly with discontinuities. Others (e.g. [122], [124]) also studied similar problems. Fua and Sander [29] have proposed a local algorithm to describe surfaces from a set of points. Their method relies heavily on local properties in initial estimate of normals, and cannot work in the presence of noise.
4.1.1.2 Physics-based approach

Szeliski et al. [100] have proposed the use of a physically-based dynamic local process to deal with the problem. Each data point is subject to various forces. They are able to handle objects of unrestricted topology, but assume a single connected object.

Terzopoulos et al. proposed the physics based approaches ([109], [110], and [111]) that rely on the Governing Equation which is typically used to describe multi-particle systems. The model itself is thought of either as a mesh-like surface composed of a set of nodes with springs connecting them, or as a surface represented by an implicit function. The initial model (or surface) is first in equilibrium without external forces exerted on it. Then, the input data points are attached to their counterparts on the model by springs. This interrupts the equilibrium, and requires the system to converge to a new steady state, which is considered to be the desired surface. In addition to the external force from the data points, there are two more forces in this dynamic system. One is from the stiffness of the spring (or the surface), and the other is the damping for dissipating the energy of the whole system. Some approaches based on this might need the correspondence relation between the collected measurements and the points on the model, which are not always available.

4.1.1.3 Functional minimization

The functional minimization approach (e.g. Han and Medioni [37], Liao and Medioni [64]) always starts with an initial surface, which is then made to fit the data. An energy function is associated with the surface which indicates the present energy of the surface. The energy comes mainly from (1) the smoothness constraint imposed on the fitting surface and (2) the distance between the fitting surface and the data. A numerical method for function
minimization is then applied to conform the fitting surface to the data by reducing the energy of the fitting surface. The underlying surface of the collected data is obtained when the function reaches the minimum. In fact, work based on the deformable model approach fall into this category.

### 4.1.1.4 Computational geometry approach

The algorithms in the computational geometry category (for example, in Boissonnat [5] and Hoppe et al. [49]) treat the collected data as vertices of a graph. The graph is constructed by adding edges between nodes, based on estimated local properties. The algorithms in this category usually produce results in a polyhedral form where each face is a triangle, and the vertices on the polyhedron are the input data points themselves. Using local information only, computational geometry based approaches cannot handle noise. They can, however, describe complex objects with any topology.

Another approach, alpha-shapes [21], has also attracted much attention in the computational geometry community. A finite set of points in 3-D space and a real parameter alpha uniquely define a simplicial complex. It consists of a collection of vertices (0-D), edges (1-D), triangles (2-D), and tetrahedra (3-D) embedded in the 3-D space. This is termed as the alpha-complex of the point set. The alpha-shape is the geometric object defined as the union of the elements in the complex.

Alpha shapes can be viewed as generalizations of the convex hull of the point set. The most crude shape is the convex hull itself, which is obtained for very large values of alpha. As alpha decreases, the shape shrinks and develops cavities that may join to form tunnels and voids. For each alpha, the alpha-complex is a subcomplex of the 3-D Delaunay triangulation. The parameter alpha is somewhat similar to our scale parameter.
4.1.1.5 Level-set approach

Recently, a new approach, known as the level-set approach, has produced very good results, and attracted significant attention, especially in the mathematics community. We have already given a brief description in Chapter 1. We refer the interested reader to Sethian [95] for a review. In the context of surface reconstruction, Zhao, Osher, Merriman and Kang [127] recently proposed an approach for shape reconstruction using the level set method.

4.1.2 Curve fitting

Curve fitting from an unorganized set of points has been studied extensively in the visualization community owing to its significance in trajectory extraction and flow visualization. More common approaches include the use of streamball techniques, line integral convolution, and curvature estimation.

Brill and Djatschin [10] propose the streamball techniques for flow visualization. Streamlines, traced from streamballs, are curve tangents at every point in a velocity field. However, streamlines may intersect at the same cuboid. The junction created by such intersection may cause problem.

Cabral and Leedom [11] developed the Line Integral Convolution (LIC), which has become a very attractive visualization technique in the field of computational fluid dynamics. The LIC algorithm takes as input a vector field lying on a Cartesian grid and a texture bitmap of the same dimensions as the grid, and outputs an image wherein the texture has been “locally blurred” according to the vector field. There is a one-to-one correspondence between grid cells in the vector field, and pixels in the input and output image. Each pixel in the output image is determined by the one-dimensional convolution of a filter kernel and
the texture pixels along the local streamline indicated by the vector field. However, the 3-D extension to track feature curves seems to be expensive. Also, while it may produce insightful 3-D flow visualization, it is inconvenient as a means of computation because the whole field is imaged and textured.

Koller, Gerig, Szekely, and Dettwiler [62] proposed a method for the multiscale detection of curvilinear structures in 2-D and 3-D image data. This method is designed for medical image analysis and visualization. This method requires an explicit, initial model.

Fruhauf [28] used raycasting for rendering volumetric vector field. The methodology makes use of “visualization objects” that are locally tangential to the vector field, and a directed light source. Streamlines are shaded at sampled points when a vector field is raycast. While it produces good visualization result, the output, being a set of rendered but unorganized points, is unsuitable for further processing.

Thirion and Gourdon [112] proposed the Marching Lines for extracting coherent feature space curves that correspond to intersection of two isosurfaces. Their method poses isosurface extraction as an “interface” problem, and guarantees good topological properties of the extracted curves.

Fidrich [26] proposes the extraction of space curves in scale space by mapping the problem into a 4-D space: as in Thirion and Gourdon [26], feature curves are considered as intersection of two isosurfaces in 3-D. So, their moving paths can be explicitly obtained in scale space as the intersection of two isosurfaces in 4-D. Both [26] and [112] require two intersecting isosurfaces, or two implicit functions in order to define a space curve. (Note that in Chapter 3, we have a direct definition for extremal curve.)
4.1.3 Limitations of these methods

The above methods are computationally expensive as an iterative process takes place. They also suffer from one or more of the following problems:

- only one genus-zero object – this is true to all deformable models approaches.
- a single $2\frac{1}{2}$-D surface can be described at any one time – spatial ordering should be known \textit{a priori}. Such methods fail when true 3-D data is present.
- smoothing – surface boundaries and discontinuities are usually smoothed out as most methods do not handle specifically surface boundaries.
- outlier noise – noisy data can severely interfere with the convergence rate to the correct surface. Methods that requires the resulting surface interpolate all input data points collapse when noise is present.

4.2 Elementary cases

In this section, we illustrate our basic tensor voting formalism with elementary examples in 3-D. We start with the most general case in which the input contains oriented and non-oriented data as well. Then, we describe in detail each specific case, each one of which fits well into our unified formalism.

4.2.1 Oriented and non-oriented input

We use the flowchart in Figure 2.1 which shows our overall approach, and produce from it an annotated version in Figure 4.1 to detail the process.
In the most general case, the input consists of points, edgels, and normals. They are first encoded as true tensors, as described in chapter 2, and a first pass of voting at the token locations only is performed. The result of this first voting pass is a set of generic tensor tokens.

Each tensor obtained is then decomposed into a ball, a plate, and a stick component. Then, in a second pass of voting, we discard the ball component (which corresponds to junction-ness, and therefore should not be propagated), and vote using the plate and stick fields only. The result is a dense tensor map, which is decomposed into vector maps. Junctions, curves, and surface are extracted by the extremal feature extraction process.

### 4.2.2 Oriented input

We first consider the “basic” case of a surfel, in which every input feature encodes a position and a direction, whose direction encodes the normal to the local surface patch. This oriented data is first encoded as true tensor as described in Chapter 2. According to our formalism, this type of feature corresponds to perfect stick tensor. Figure 4.2 depicts the specialized version of Figure 4.1 which shows the steps involved for this case.

In this section, we illustrate, with examples, the different feature inference capabilities from a range of oriented data, such as:

- Surface-ness from surface elements, surfels
- Curve-ness from curve elements, curvels
- Surface-ness from curve elements, curvels
- Curve-ness from surface elements, surfels

Each case has real applications, which are mentioned in the following chapters or in subsequent sections.
Figure 4.1: Annotated version of the basic flowchart of Figure 2.1
4.2.2.1 Surface inference from surfels

This is the basic case for surface inference, in which oriented data (in the form of surfels) are available at each input site. We show the specialized flowchart in Figure 4.2. Here, we also illustrate this process with small cases as running examples.

**Four-point basic plane**

Figure 4.3(a) shows an input which consists of four coplanar surfel tokens with an associated, constant, normal. Note that the direction of these normals is not consistent (i.e., they do not point toward the same side of the underlying plane). Global orientation consistency is not a requirement, owing to the use of second-order symmetric tensor in our
formalism. Each surfel is thus a perfect stick, which is aligned with the stick voting field for surface inference in order to propagate normal votes in its neighborhood. When each site has collected the directed votes from its neighborhood, its surface saliency is measured by $(\lambda_1 - \lambda_2)$, with normal direction estimated as $\hat{e}_1$. The set of points with extremal $(\lambda_1 - \lambda_2)$ along $\hat{e}_1$ is extracted as an extremal surface, represented by a triangular mesh. Figure 4.3(b) shows the resulting planar surface inferred.

**Four-point basic ellipsoid**

Figure 4.4(a) shows an input consisting of four surfel tokens whose orientations are consistent with an underlying ellipsoid. Again, each input surfel is a perfect stick. The resulting surface saliency, after voting, is measured by $(\lambda_1 - \lambda_2)$, with normal direction estimated as $\hat{e}_1$. The locus of points with extremal $(\lambda_1 - \lambda_2)$ along $\hat{e}_1$ is extracted as an extremal surface, represented by a triangular mesh. Figure 4.4(b) shows the resulting global surface inferred.
Four-point basic saddle

Figure 4.5(a) shows an input consisting of four surfel tokens whose orientations are consistent with an underlying saddle surface. Again, each input surfel is a perfect stick. The resulting surface saliency, after voting, is measured by \((\lambda_1 - \lambda_2)\), with normal direction estimated as \(\hat{e}_1\). The locus of points with extremal \((\lambda_1 - \lambda_2)\) along \(\hat{e}_1\) is extracted as an extremal surface, represented by a triangular mesh. Figure 4.4(b) shows the resulting saddle surface inferred.

4.2.2.2 Curve inference from curvels

Here, oriented data (i.e. curvels) are available at each input site. A curvel is a curve element consisting of a point associated with tangent information. Figure 4.6 shows the flowchart. Curve inference from curve elements has direct application, such as trajectory extraction (e.g. see Chapter 6 on vorticity line extraction). Figure 4.7(a) shows the input tangents. Each input tangent is encoded into a plate tensor, and the direction of the tangent
(a) two views of four points with associated normals consistent with a saddle surface

(b) two views of the inferred saddle surface

Figure 4.5: Four-point basic saddle

vector is aligned with \( \hat{e}_3 \), the normal to the plane defined by \( \hat{e}_1 \) and \( \hat{e}_2 \). The difference with the previous section resides in the input information, and in the voting field used. When each site has received the directed votes collected in its neighborhood, curve saliency is measured by \( \frac{\lambda_2 - \lambda_3}{\lambda_1} \), with tangent direction estimated as \( \hat{e}_3 \). The locus of points with extremal \( \lambda_2 - \lambda_3 \) along \( \hat{e}_3 \) are extracted as an extremal curve, represented by a set of piecewise connected linear curve segments. Figure 4.7(b) shows the resulting curve obtained.

Figure 4.8(a) shows another example input set of tangents, whose direction is not necessarily consistent. The orientations of the input curves are consistent with a helix, which
is defined by the parametric equation \((40 \cos t, 40 \sin t, 20t)\) in this case. The resulting curve is shown in Figure 4.8(b).

### 4.2.2.3 Surface inference from curvels

This is the case when the input consists of curvels lying on a surface, and we want to infer the underlying surface. The use of digital 3-D scanner usually results in a set of curvels (curve elements lying) on the object being scanned. Figure 4.9 shows the specialized flowchart, and we have the following running example to explain this process. Figure 4.10(a) shows 4 curvels, whose orientations are consistent with a planar surface. These input edgels denote perfect plate tensors, with direction encoded as \(\hat{e}_3\). We need to
infer the normal at each site for surface inference. Each input plate is aligned with the plate voting field. When each site has received the directed votes, surface saliency is measured by $(\lambda_1 - \lambda_2)$, with normal direction estimated as $\hat{e}_1$, as shown in Figure 4.10(b), and the resulting surface is shown in Figure 4.10(c).

### 4.2.2.4 Curve inference from surfels

When two surfaces intersect each other, a curve junction is produced. The flowchart summarizing this curve inference process is depicted in Figure 4.11. Figure 4.12(a) shows two views of the input set of surfels whose orientations are consistent with two intersecting
Figure 4.8: Another example of curve inference from curvels

planes. Each input surfel is a perfect stick, which is aligned with the stick voting field for propagating votes in its neighborhood. When each site has collected all the directed votes from its neighborhood, its curve junction saliency is measured by $(\lambda_2 - \lambda_3)$, with tangent estimated as $\hat{e}_3$. The set of points with extremal $(\lambda_2 - \lambda_3)$ along $\hat{e}_3$ is extracted as an extremal curve, represented by a set of piecewise connected linear curve segments. Figure 4.12(b) shows the resulting junction curve inferred.
4.2.3 Non-oriented input

We now describe the case where the input tokens are points (without any associated orientation). In these cases, our approach consists of a preliminary step of voting which infers a tensor at each input site. Since there is no orientation information available, the input set of points vote with the 3-D ball field. This is done so regardless of surface or curve inference. After this voting step, we get a tensor, which can be decomposed into a ball, a
Figure 4.10: Surface inference from curvels

plate, and a stick component. Then, we vote with the stick voting field for feature inference, depending on whether the desired smooth feature is curve or surface. This second voting step is identical to the one used for the case when oriented data is available.

In this section, we illustrate feature inference capabilities from non-oriented data. This comprises of

- Surface-ness from points
- Curve-ness from points

### 4.2.3.1 Surface inference from points

Figure 4.13 depicts the flowchart. The running example is a set of 4 points sampled from a planar surface. Figure 4.14(a) shows the input. One point is made to lie farther from the
other 3 points. Since no initial orientation is given, these points are initially encoded into perfect balls. The ball voting field is thus placed at each point, propagating ball votes in each neighborhood. After the preliminary step of ball field voting, a tensor is produced at each point. The stick components (or $\hat{e}_1$) of these tensors, which denote the normal directions, are shown in Figure 4.14(b). Note that although surface saliency ($\lambda_1 - \lambda_2$) at the farthest point is relatively small, the normal direction is still correctly estimated. These inferred sticks then vote again, with the stick voting field for surface inference, exactly as
4.2.3.2 Curve inference from points

The process is summarized in Figure 4.15. Figure 4.16(a) shows 4 collinear points. Again, since no initial orientation is given, these points are encoded into perfect balls. The ball voting field is thus placed at each point, propagating ball votes in each neighborhood. After the preliminary step of ball field voting, a tensor is produced at each point. The tangent component (or $\hat{e}_3$) of these tensors, which denote the tangent directions, are shown in...
Figure 4.13: Flowchart of surface inference from points

Figure 4.16(b). These inferred tangents then vote again, now with the plate voting field. Figure 4.16(c) shows the inferred curve.

Another example of a helix is shown in Figure 4.17. It is interesting to note that, if we continuously scale the $z$ coordinates of the input points (so that the helix coils more tightly) up to a certain point, a cylindrical surface will become a more salient description than a helical curve one. Such a cylindrical surface is indeed produced by our system when we make such a tighter helix, which is in agreement with human perception (Figure 4.18).
4.3 Other applications and examples

More results on complex examples are presented in this section. These examples are drawn from real application domains, such as

- Seismic data processing
- Medical imagery
- Terrain reconstruction
- 3-D object modeling from photographs
Figure 4.15: Flowchart of curve inference from points

More results can be found in Chapter 6 on visualization, and also in [103, 104].
4.3.1 Fault detection

In the exploration for oil and gas in the subsurface, seismic data are gathered. Modern seismic acquisition yields a 3-D image of the subsurface. Due to the fact that water is denser than oil and gas, water tends to push oil and gas formed in the subsurface upwards. If no barrier stops the petroleum, it will eventually migrate to the surface and disappear. However, some oil and gas gets trapped in reservoirs made up by some kinds of barriers. Typical
barriers are impermeable sediments and faults. In order to determine the presence of oil and gas, identification of the barriers is important. Automated tools for interpreting layers have been around for several years. However the interpretation of faults is not automated to the same extent.

We apply our methodology for the extraction of fault surfaces and junction curves which are inherent in a sample set where 3-D points on fault surfaces have been extracted from seismic data. Seismic data provides useful information on the structure of sediments in the
Figure 4.18: A surface is more salient a curve as we scale along the \( z \)-axis subsurface of the earth. The faults represent potential oil and gas reservoir boundaries, and are therefore of major interest to identify. Up to now fault interpretation has been a work-intensive manual process. The junction curves, which indicate the intersection of different faults, are also important in allowing an automatic or semi automatic interpretation process to detect fault intersections. Figure 4.19 shows two views of the data, the extracted junction curves, and two views of the extremal surfaces we extract from the data, in which the overlapping layers are faithfully extracted.
Figure 4.19: Fault detection from seismic data

(a) two views of the input data corresponding to faults (courtesy of Schlumberger Geco-Prakla, Stavanger, Norway) (b) extracted junction curves overlayed on the extracted surfaces (c) two views of the overlapping surfaces we extract, which represent the faults corresponding to relative Earth movement.
4.3.2 Medical imagery

A set of 18224 points is sampled from Femur, the proximal bone of the lower limb. We introduce about 400 outliers into the data set. We infer the surface description directly from the noisy data, as shown in Figure 4.20.

Figure 4.20: Inferred surface description for the femur data set

Courtesy of Gregoire Malandain, INRIA, Sophia, France

Dental CAD/CAM is in the process of revolutionizing dentistry, and is appearing in every major dental groups and laboratories today. Its main components are data acquisition, a modeling, and a milling system. The only widely used commercial system is the CEREC
system, produced by Siemens Inc. It is a self-contained unit with an imaging camera, a monitor, and an electrically controlled machine to mill inlay and onlay restorations from ceramic blocks. The accuracy is not good, and significant manual processing is required. Another system of interest, developed by Duret et al. [19, 20], was able to produce crowns with an average gap of 35 µm. The system is no longer commercially available, and suffered from lack of speed and a cumbersome interface.

An alternative approach, followed here, is to perform the restoration design manually, in a laboratory, by using a conventional medium such as wax; then to transfer this physical design into a digital model. This information can be directly used to control a CAM machine to mill the restoration from a ceramic (or other) block. Typically, digital measurements are sampled from a wax model, using palpation or optical sensing [19]. Figure 4.21 shows the set up of one such commercial, multiple-view registration system, from which our data was obtained. As we shall see, though mostly accurate, the point sets obtained also contain many erroneous outlier readings. Given such noisy measurements, the challenge is to derive an accurate shape description automatically. Otherwise, even slight errors may result in artifacts which need to be corrected manually, or worse, may make the restoration unusable.

While filter-based techniques, such as the discrete Fourier transform, are effective in suppressing spurious samples, they often “oversmooth”, degrading sharp discontinuities and distinct features that correspond to important anatomical (preparation) lines and features. At present, unless intensive human intervention is used, it is impossible to construct an accurate representation that respects both medical and dental criteria requiring accuracy of up to 5 to 10 µm [20]. Here, we provide an alternative approach, based on tensor voting and extremal feature algorithms, to infer faithful dental models in the form of
surfaces (for capturing smoothness) and 3-D curves (for preserving shape discontinuities) from noisy dental data. Using our approach, we can also perform data validation by removing spurious outliers in the original noisy data set, using the intermediate SMap and CMap representations.

We tested our system on a variety of crowns and inlays. An inlay is a cast filling that is used to replace part of a tooth, while a crown is a larger restoration. The data is acquired using the set-up shown in Figure 4.21. The wax shape, obtained from the dental laboratory, is rotated about the $x$-axis in $15^\circ$ (or $90^\circ$) increments so that 24 (or 4) successive views are visible to the sensor.

**Mod-4.** We first demonstrate in this and the next two examples the graceful degradation of tensor voting in the presence of spurious and missing samples. Because of this, feature integration (as outlined in Chapter 5) is skipped. A set of only 4 views of a $Mod$ are digitized and quantized in 100x100x100 array, which contains 5217 points. This data set is difficult because it has a complicated shape, and has many missing and misleading data resulting from fewer registered views and self occlusion. Using the inferred surface and
curve model, we can perform *data validation* (Figure 4.23(a)): after voting, we validate each input point by simple thresholding of its associated surface or curve saliency. This shows a simple application of spurious noise elimination with the availability of a faithful model. The extracted extremal surfaces and curves are depicted in Figure 4.24(a). Note that surfaces are only correct away from the discontinuity curves, for which they have low surface saliencies. However, these discontinuities are detected and marked since they are characterized by high curve saliencies. These detected junction curves allow further refinement or intervention to take place.

**Inlay-4.** A set of 4 views of an *Inlay* are digitized, having 2447 data points quantized in a 70x50x70 array. This is a very complicated surface, and the data set is noisy, and contains many missing and erroneous readings. Result of data validation is shown in Figure 4.23(b). The extremal surfaces and curves extracted are shown in Figure 4.24(b).

**Crown-4.** A sparse set of only 4 views of a *Crown* are digitized and quantized using 100x100x100 voxel array. This data set contains 8844 points. We can recover the underlying surface model, which is automatically marked with curves representing grooves and preparation lines. With a faithfully inferred model, we can perform data validation (Figure 4.23(c)). Figure 4.25(a) shows the resultant extremal surfaces and curves.

**Crown-24.** A set of 24 views of a *Crown* are registered. The data set contains 60095 points, quantized in a 100x100x100 array. Figure 4.22 depicts three slices of the data, showing mostly accurate, but also spurious and missing data. Again, since it is a scalar field, tensor voting is used to produce the dense CMap and SMap. In particular, grooves
Figure 4.22: Three slices of the original input, with mostly accurate but erroneous data
(a) $x = 42$, (b) $x = 65$, and (c) $x = 80$

and anatomical lines can be abstracted as intersection between extremal surfaces which is coherent with the definition of the CMap.

We can detect the upper and lower surfaces of the Crown. The detected preparation line and the grooves are in turn used to produce a coherently integrated surface and curve description. Result of data validation is shown in Figure 4.23(d). Figure 4.25(b) shows the extracted extremal surfaces and curves.

### 4.3.3 Terrain reconstruction

This experiment demonstrates the visualization of a digital terrain model (DTM), and the detection and extraction of the crestline inherent in the noisy scalar field. A section of the East Pacific Ridge is sampled as a low resolution DTM from [84], and two views of the noisy scalar depth field are depicted in Figure 4.26. The area covered begins at the northwest corner at $(-8.75, -127.75)$ progressing eastward for 205 values, then stepping 5 minutes south for the next row, ending at the southeast corner at $(-24.25, -104.75)$, where $(-x, -y)$ denotes south latitude $x^o$ and west longitude $y^o$, respectively. We seek to
Figure 4.23: Data validation
A middle slice of extremal surfaces, the original noisy data, and the validated data set for 
(a) Mod-4, (b) Inlay-4, (c) Crown-4, and (d) Crown-24.
Figure 4.24: Dental restoration

Two views of the original noisy data, the extremal discontinuity curves and surfaces inferred for (a) Mod-4 and (b) Inlay-4.
Figure 4.25: More results on dental restoration

Different views of the original noisy data, the extremal discontinuity curves and surfaces inferred for (a) Crown-4 and (b) Crown-24.
extract the ridge in the form of an extremal surface, and represent the crestline as an extremal curve. Being a scalar depth field, tensor voting is applied to the DTM to produce the SMap and CMap, which are in turn directly input to the extremal surface and curve algorithms respectively for feature extraction. In particular, we assert that the crestline should be localized as the intersection of between the two halves of the ridge, where two distinct distributions of directed votes on surface normals should be present. Thus, this is in coherence with the definition of the CMap.

Results are shown in Figures 4.27 and 4.28. Given the CMap, the extremal curve algorithm detects and extracts the crestline and other discontinuities which are only implicit in the noisy data. Also, the underlying sea bed representing the ridge is faithfully extracted from the SMap by the extremal surface algorithm.

Note that differential properties on surface extremality no longer applies in the vicinity of the crestline, which is a surface discontinuity (as shown in Figure 4.27(b), in which a
Figure 4.27: Automatic integration of detected ridge surfaces and crestine lines
(a) The crestline of the ridge is detected as an extremal curve. (b) extracted ridge before integration, showing a “gap” in the surface description. (c) result after integrating intermediate ridge surfaces and crestline “gap” is present in the surface description). However, this singularity is exactly detected by the extremal curve algorithm. Since both extremal surfaces and curves are coherent computation means, they can readily be further interpreted to produce a coherently integrated surface description in which the two halves of the ridge surfaces intersect at junction curve precisely, as shown in Figure 4.27(c). Two more views of the extracted curves and surfaces are depicted in Figure 4.28.

We also processed some other DTM data taken from satellites. Shown in Figures 4.29 and 4.30 are the input DTMs and the processing results, for the Capital Building and the Labor Department at Washington D.C. respectively. Note that in the Capital Building results, we can segment the input depth data into three surfaces: the dome, the roof, and the ground. The color shown there only denotes the depth or z-coordinates of the data. We can also segment the Labor Department DTM into several surface layers, each of them is
The crestline of the ridge and other surface discontinuities are detected and extracted by the extremal curve algorithm. Shown here are the detected junction curves overlayed with the original data. The extremal surface algorithm extracts the surface representing the ridge.

also shown color-coded according to depth. But input DTMs contain a moderate amount of outliers.

4.3.4 3-D object modeling from photographs

A stereo system was developed by Chen and Medioni [14]. Given a pair of images, the system automatically recovers the epipolar geometry. Then, a dense disparity map is produced, from which the 3-D coordinates can be inferred. To obtain a model, a Styrofoam
head was placed on a rotary table and six pairs of stereo images were collected. The rotary table was arranged so that the frontal face has more coverage than the back. The point sets resulted from the six stereo pairs are registered into a single coordinate system, and the resulting point set is then used as input to our 3-D system. Note that the resulting point set is very noisy. Figure 4.31 shows two views of the input point set, and the faithful result produced by our system.
4.4 Summary

In this chapter, we have illustrated the basic formalism and feature extraction algorithms with elementary examples, such as

- surface inference from surfels
- curve inference from curvels
- surface inference from curvels
- curve inference from surfels
- surface inference from points
• curve inference from points

Note that we do not explicitly direct the system to extract either surfaces, or curves, or junctions, or any combination. Instead, our system produces the most salient features. We also applied the system to a variety of applications, such as fault detection from seismic data, dental restoration, and surface inference from dense stereo data.
Figure 4.31: Inferred surfaces for the point set obtained from six stereo pairs
Chapter 5

Extension to the Basic Formalism: Feature Integration

Figure 5.1: Inferring integrated high-level description
(a) Input sparse data points with normals. (b) Surface orientation discontinuity is localized as an intersection curve. (c) Output surface with discontinuity preserved.

Our human visual system can perform an amazing job of perceiving surfaces from a set of 3-D points. We not only can infer surfaces, but also segment the scene, detect surface orientation and discontinuities. For example, Figure 5.1 shows a sparse set of points (with normals) sampled from a planar surface intersecting with a sphere. The circle represents the intersection contour which is not explicit in the data. When the data is presented to us as a sequence of projections, we ourselves have no problem in inferring such surface discontinuities (i.e. the circular intersection curve) and segment the scene into two components,
a spherical and a planar surface. Earlier work by Guy and Medioni [36], which was later formalized by Lee and Medioni [72], proposed to detect the presence of junctions and intersection curves from such data. While it did a good job of it, it did not try to integrate them into a unified representation, but instead produced three independent representations: one for surfaces, one for curves, and one for junctions. It can be readily observed from their results that the surfaces inferred are only correct away from curves and junctions, and that curves and junctions are not properly localized.

In this chapter (also in [101, 102]), we describe the work done on feature integration for proper feature localization. This work is a sequel to Guy and Medioni [36]. The main contribution is to show that this voting process, when combined with our novel surface and curve extraction processes, can be readily extended to unify these three independent representations to produce an integrated description of surfaces, curves, and junctions.

In this chapter, we start by motivating this work, then describe the overall approach and detail on the integrated feature inference process. We finally analyze the time complexity and show results on complex data.

### 5.1 Motivation and overall strategy

An integrated high level description requires that surface orientation discontinuities be explicitly preserved. However, as readily seen from [36], generating surfaces (resp. curves and junctions) using the SMap (resp. CMap and JMap) independently does not guarantee precise localization of discontinuities. Discontinuities, though detected, may be:

- Smoothened out. For example, although a salient curve may be detected in CMap, it is still possible that the surface saliency gradient across the corresponding voxels...
Figure 5.2: Detected features are not well localized
(a) Input data obtained from a digitized surface sweep on a triangular wedge, (b) and (c) two views of the reconstructed surfaces, 3-D curves and junctions. Surface (resp. curve) orientation discontinuities, though detected, are not well localized because SMap (resp. CMap) is used alone for surface (resp. curve) tracing.

in SMap varies smoothly and thus a smooth surface will be traced if the SMap is alone considered, and surfaces, curves, and junctions are not coherently integrated, as shown in Figure 5.2.

- left “undecided”. Voxels around discontinuities may have a low surface saliency and thus no surface is produced, creating a gap (Figure 5.3(a)).

- Incorrect. Because of data sparsity, using the CMap alone to trace a curve may produce incorrect result. A sphere intersecting with a plane should give a circle. However, if we use the CMap alone in this case to infer the intersection contour, it will not be circular (as shown in Figure 5.3 where one of the hemispherical surfaces is also shown).
Figure 5.3: Incorrect curve may be obtained by considering the CMap alone (a) gaps are produced around region of low surface saliency, (b) and (c) two views of the incorrect curve owing to data sparsity.

We have illustrated through examples the limitations of noncooperative feature inference. In this chapter, we show (constructively by proposing an algorithm) that while the original work by Guy and Medioni [36], and by Lee and Medioni [72], does not handle an integrated description, their voting approach can be cleanly extended into a cooperative framework such that surfaces, curves, and junctions can be inferred cooperatively. The underlying idea is two-fold:

- Extending the use of the 3-D stick, plate, and ball voting fields in the tensor voting formalism to infer surface/curve orientation discontinuities.

- Making use of the extremal surface and curve extraction processes described in Chapter 3 for initial estimate generation and subsequent refinement.

Our overall strategy is as follows: For preserving precise surface orientation discontinuity, we treat the curve as a surface inhibitor: the curve will not be smoothed out while a surface is being traced using the SMap. Once an explicit initial surface estimate is obtained, we treat the very same curve as exciter for computing precise discontinuity. A similar procedure applies to curve orientation discontinuity. We show in Figure 5.4 the major steps of
the cooperative algorithm, and illustrate them in Figure 5.5 using one face of our triangular wedge (Figure 5.2) as a running example:

1. **Curve trimming by inhibitory junctions.** Initial junctions vote with a curve inhibitory field so that the detected discontinuity (indicated in the JMap) will not be smoothed out when a developing curve is evolving. (Figure 5.5-(1)).

2. **Curve extension toward excitatory curves.** Initial junctions and curve vote with curve excitatory fields so that the curves obtained in (1) are brought to intersect with the junctions (Figure 5.5-(2)).
3. **Surface trimming by inhibitory curves.** Extended curves obtained in (2) vote with a *surface inhibitory field* so that the detected discontinuity (indicated in the CMap) will not be smoothed out when a developing surface is evolving. (Figure 5.5-(3)).

4. **Surface extension toward excitatory curves.** The extended curve and the trimmed surface are convolved with *surface excitatory fields* so that the latter can be naturally extended to hit the curve (Figure 5.5-(4)).

5. **Final curves and junctions from surfaces.** The set of surface boundaries obtained by extended surface intersection produces a set of refined curves and junctions which are coherently integrated and localized. (Figure 5.5-(5)).

We want to emphasize here that our cooperative approach is *not* iterative (though each step makes use of results produced in previous step(s)). Also, the geometric locations of all junctions, 3-D curves, and surfaces may change considerably after the cooperation process.

### 5.2 Cooperative computations and hybrid voting

We extend vector voting in [36] to cooperatively integrate initial junction, curve, and surface estimates generated by extremal feature extraction algorithms in order to obtain an
integrated description. Slight modifications are needed for both feature extraction algorithms, which are described in the following sections.

5.2.1 Feature inhibitory and excitatory fields

In essence, the process of feature integration is to define feature inhibitory and excitatory fields and to use them for feature localization. We have curve and surface inhibitory fields. Curve (resp. surface) inhibitory field is a inhibition mask for inhibiting curve (resp. surface) growing as intended by its respective extremal algorithms. No curve segment (resp. surface patch) is possible in a voxel masked by a inhibitory field.

We also have excitatory fields for inferring feature extension. These fields are essentially the stick, the plate, and the ball fields (defined in Chapter 2) for feature extension toward the detected orientation discontinuity. In particular,

- Curve excitatory fields are defined by the ball kernel and the plate kernel (defined in Chapter 2),
- surface excitatory fields are defined by the plate and the stick kernels.

between $O$ and $P$, because it keeps the curvature constant. The “most likely”

5.2.2 Curve trimming by inhibitory junctions

The extremal curve extraction algorithm (Chapter 3) is modified to take not only the CMap but also the detected junction estimates (from JMap) as input. The process is explained in the following:

1. The voxels in CMap corresponding to initial junctions are inhibited by a curve inhibitory field to protect detected curve orientation discontinuity detected in JMap. It
is done simply by putting the corresponding inhibition mask over the detected junction locations. The typical size of this inhibition mask is 5x5x5.

2. Curves are traced exactly as described in (Chapter 3).

Since curve growing is inhibited around a junction, but not by the low threshold in the original extremal curve extraction, its orientation discontinuity is smoothed out, and no spurious curve features are created around a junction. This step results in a set of trimmed curves.

5.2.3 Curve extension toward excitatory junction

The detected junctions and trimmed curves obtained in the previous phase are used to produce an “extended” curve for which curve orientation discontinuities are preserved. We group these features by using an incidence graph (Figure 5.6(b)), and process curve extension one by one.

First, an incidence graph $G = (V, E)$ is constructed with $E$ corresponding to curves and $V$ to incident junctions (Figure 5.6). This graph is created by checking the distance between every endpoint of the trimmed curves and the detected junctions to determine to which junctions the curve endpoints should be connected. By such grouping, we can avoid unnecessary and unwanted interaction among excitatory fields (defined shortly).

Then, for each curve (edge in $E$) and its incident junctions (vertices in $V$), we quantize them as input (recall that this intermediate curve is of sub-voxel precision). Two excitatory fields are used to extend the trimmed curve toward the detected junctions such that it will intersect the junction precisely in a single pass of voting (see Figure 5.7):

1. (Excitatory plate kernel) Each curve segment (in $E$) votes with the plate kernel.
Figure 5.6: Given initial curves and junctions, an incidence graph is constructed

Figure 5.7: Inference of the most probable extension

2. (Excitatory ball kernel) The junctions (in $V$) vote with the ball kernel, where each tensor vote in the ball kernel is increased (i.e. excitatory) in order to "attract" the curve toward the junction. Its size is related by a constant factor (its choice is not critical; typical factor is two) to the curve inhibitory fields in section 5.2.2.

Note that we use two types of voting fields (namely, the plate and ball kernel) in a single voting pass. Since both of them are vector fields, the voting proceeds in exactly the same way as described in Chapter 2.

For vote interpretation, we assign a 2-tuple $(s, \hat{r})$ in each voxel by the following. Since the voting ellipsoid is very elongated if there is a high agreement in one direction (Figure 5.7(c)), $s = \lambda_2 - \lambda_3$ is chosen as the saliency, and $\hat{r} = \hat{e}_3$ gives the direction of the
curve segment tangent. With this map, a slight modification of the extremal curve extraction algorithm described suffices to extract the desired extended curve. The low threshold is eliminated and curve tracing continues until the junction is exactly hit. This extended curve preserves curve orientation discontinuity.

5.2.4 Surface trimming by inhibitory curves

In this phase, the extremal surface algorithm is modified to take not only the SMap but also the extended curve obtained in the previous phase to produce a triangular mesh:

1. First, the voxels in the SMap corresponding to the location of the extended curve vote with a surface inhibitory field to protect surface orientation discontinuity detected in the CMap.

2. The surface is then traced as described in Chapter 3. A set of trimmed surfaces is produced.

5.2.5 Surface extension toward excitatory curve

The extended curves (section 5.2.3) and the trimmed surface (section 5.2.4) computed in previous phases are used together to produce an extended surface with preserved surface orientation discontinuity.

First, an incidence curve list is constructed for each trimmed surface. This list corresponds to the set of extended curves with which a trimmed surface will intersect (Figure 5.8). Similar to the use of incidence graph, an incidence curve list is used to group relevant surface and curve features so that we can process surface extension in a divide-and-conquer manner and avoid unnecessary interaction among excitatory fields. To create
this list for each trimmed surface, we examine each curve, and a curve closest to the mesh boundary of the trimmed surface will be assigned to the list.

![Diagram](image)

Figure 5.8: Surface extension
(a) For surface extension, the corresponding normals from this trimmed surface mesh are convolved with stick kernel. Curve segments are convolved with plate kernel. (b) An incidence curve list for the surface.

Then, each trimmed surface with its (enclosing) curves is treated as input to our voting procedure (i.e., we quantize both the curve (tangents) and the surface (normals). This quantization is needed because both the extended curves and the trimmed surfaces are of sub-voxel precision). Two excitatory fields are used in a single voting pass (Figure 5.8(a)).

1. (Excitatory plate kernel) The tangents vote with the plate kernel, in which vector weight in each vote is increased in order to “attract” the trimmed surface toward the curve. Again, its size is related by the size of the surface inhibitory field used in section 5.2.4.

2. (Excitatory stick kernel) The normals vote with the stick kernel for inferring the most natural extension for filling the gap.

Note that the voting process is exactly the same as described in Chapter 2, even though we use hybrid vector fields to vote. For vote interpretation, we assign to each voxel a 2-tuple \((s, \hat{n})\), where \(s = \lambda_1 - \lambda_2\) is the saliency and \(\hat{n} = e_1\) gives the direction of normals.
The resulting map, which is a volume of surface normals with their saliencies, is fed into the extremal surface extraction algorithm with the following modification (Figure 5.9):

Since the extended curve may be *inaccurate* (recall that no surface information is used at the time when the curve was computed), the evolving surface may extend beyond, or “leak” through, the extended curves if the saliency of the currently visiting voxel in the SMap exceeds the low threshold of the extremal surface algorithm. (Note that, it does not suffice to simply increase the low threshold, nor to inhibit the neighboring voxels around the possibly incorrect curve.)

![Diagram](image)

**Figure 5.9: “Leakproof surface”**

A *curve lies on a leakproof surface, which is obtained by voting with the stick kernel, followed by normal estimation. Leakproof surface is also used later to infer a more accurate curve by surface/surface intersection.*

To prevent such leakage, we infer a *leakproof surface* by using our voting procedure, as in the following:

1. For all tangents of an extended curve, we compute the cross product with its closest surface normal \( \pi \) obtained above (Figure 5.9).
2. These estimated normals constitute another set of sparse data, which vote with the stick kernel. Extremal surface extraction is then performed, which explicitly produces the leakproof surface. This triangular mesh approximates the surface on which the extended curves are lying, and is used to inhibit the extremal surface algorithm from extraneous extension or “leakage.”

At this point, the reader may ask why there is no need for leakproof curve or surface in the case of curve extension toward excitatory junction (section 5.2.3). First, we do not have enough information to infer such leakproof curve or surface in that phase. Also, for curve extension, we set a distance threshold to prevent the curve from missing the junction during extension. This heuristics may not give the best intermediate result; but the final result will improve when surface information is taken into account, as in the last phase of the integration to be described in the following.

5.2.6 Final curves and junctions from surfaces

The extended surfaces obtained in the previous phase are most reliable because they are inferred from cooperating intermediate junction, curve and surface estimates together. These surfaces are used in turn to generate better intersection curves and junctions which are lying, or localized, on the surfaces. (Recall that intermediate junctions and curves are obtained without taking surfaces into consideration, since the surfaces as detected around those junctions and curves are unreliable.)

By construction, these intermediate curves and their junctions lie on the leakproof surface. Therefore, it suffices to compute the surface/surface intersection, or the precise surface boundaries, between the extended surface (the most reliable cue) and the leakproof
surface (on which curves and junctions are lying). A set of line segments results which, after cleaning up, is the set of coherent curves and junctions where our final surfaces should intersect precisely.

Note that since a curve is usually shared by more than one salient surface, the most salient of all extended surfaces is used to compute the surface/surface intersections with the leakproof surface. And the resulting final curve will be marked so that it will not be computed more than once.

### 5.3 Time and space complexity analysis

Let

- \( n \) = number of input points
- \( k \) = maximum dimension of voting field
- \( s \) = number of voxels occupied by output surfaces
- \( c \) = number of voxels occupied by output curves
- \( j \) = number of junctions

In our implementation, the input is quantized but not stored in a 3-D voxel array. Such a voxel array is very expensive and wasteful if the data set is large and the inferred surface is (usually) thin. Instead, a heap is used for storing the quantized input. Heap insertion and search, which are all we need, take \( O(\log n) \) in time [15].

Since a 3-D voxel array is not available, an efficient data structure is needed to maintain the vote collection (i.e. SMap, CMap, and JMap). Realizing that only insertion, search, and deletion are all we need for maintaining the vote store, a 3-D red-black tree [15] (which
is a specialization of the more general \( k-d \) tree) is used for storing votes. Insertion, search, and deletion can be done in \( O(\log(s + c + j)) \) time (or \( O(\log s) \) because \( s \gg c + j \)). Note that deletion is also needed for the maintenance of the vote store, because indiscriminate growth of the tree will lead to severe memory swapping that degrades the computing system. In the current implementation, we limit the maximum size of the vote store to be 20 MB. When this threshold is exceeded, the whole tree will be purged for freeing the memory.

Therefore, for space complexity, the heap and the vote store together takes \( O(s + n) \) space. For sparse data, \( n \ll s \). Also, the vote store stores surface patches of sub-voxel precision. Therefore, the storage requirement for the initial heap is not as substantial when compared with that of the red-black tree vote store. Therefore, the total space requirement is \( O(s) \) in practice.

Next, we analyze the time complexity of our method. Note that when dense data are available (during surface extension toward excitatory curves) for which small voting fields can be used, the effective neighborhood for each site is also small. Therefore, each site can gather all the votes cast by its effective neighborhood (size of voting field) only, perform smoothing, compute the eigensystem and surface patch (if any) for that site, all on-the-fly. The result produced by vote casting (as described in Chapter 2) is equivalent to that produced by vote gathering.

The total time complexity is analyzed as follows:

1. **Computing SMap, CMap, and JMap.** It takes \( O(sk^3) \) time for vote convolution, vote aggregation and interpretation, because only voxels occupied by surfaces (and their finite neighborhood that contains curves and junctions) are considered.
2. *Initial junctions from JMap.* $O(s)$ time for local maxima extraction.

3. *Curve trimming by inhibitory junctions.* We preprocess the CMap in $O(s)$ to build a Fibonacci heap (F-heap) for $O(\log s)$-time per seed extraction afterward. For extremal curve algorithm, computing zero crossings using SingleSubVoxelCMarch only involves a constant number of choices, so it takes $O(1)$ time to produce a point on an extremal curve (if any). Therefore, the extremal curve algorithm runs in linear time, i.e. at most $O(s + c)$, or $O(s)$ because $s \gg c$ in practice.

4. *Curve extension toward excitatory curves.* It takes $O(c + j)^2$ time to compute the incidence graph. Total vote convolution, aggregation and interpretation takes at most $O((c + j)k^3)$ time.

5. *Surface trimming by inhibitory curves.* Like (3), seed extraction takes $O(\log s)$ time. Also, SingleSubVoxelMarch takes $O(1)$ time to compute a zero-crossing patch (if any) in a voxel. Therefore, the extremal surface algorithm runs in linear time, i.e. at most $O(s)$.

6. *Surface extension toward excitatory curves.* Incidence curve list can be constructed in $O(s + c)^2$ time. Total vote convolution, aggregation, and interpretation takes at most $O((s + c)k^3)$ time. The extremal surface extraction algorithm runs in time linear with the output size, i.e. at most $O(s)$ time.

7. *Final curves and junctions from surfaces.* The surface/surface intersection routine can be embedded in (6), where each intersection between two voxel surfaces takes $O(1)$ time.
In all, the most time consuming part is step (6), because voting is performed on dense normals given by SMap, a thick set of points with normal information. Because we have dense information, the voting field used (i.e., $k$) in step (6) is small (typical size is about 5x5x5). Our program runs in about 15 minutes on a Sun Sparc Ultra 1 for 1000 input points.

In summary, we tabulate the space and time complexities in Table 5.1.

<table>
<thead>
<tr>
<th>Space Complexity</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap (for input quantization)</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Fibonacci heap (for seed store)</td>
<td>$O(s)$</td>
</tr>
<tr>
<td>Red-black tree (for vote collection)</td>
<td>$O(s)$</td>
</tr>
<tr>
<td><strong>Total space complexity</strong></td>
<td>$O(n + s) \approx O(s)$</td>
</tr>
</tbody>
</table>

| Computing SMap, CMap, and JMap             | $O(sk^3)$                   |
| Initial junctions                          | $O(s)$                      |
| Curve trimming                             | $O(s + c)$                  |
| Curve extension                            | $O((c + j)k^3)$             |
| Surface trimming                           | $O(s)$                      |
| Surface extension                          | $O((s + c)k^3)$             |
| Final curves and junctions                 | $O(s)$                      |
| **Total time complexity**                  | $\approx O(sk^4)$           |

Table 5.1: Space and time complexities of feature integration (without using curvature)
5.4 Limitations and proposed solution

The actual running time for feature localization is quite long. The main reason is that no curvature information is used in the integration process.

As described in section 5.3, in order to localize surface orientation discontinuity precisely, the whole intermediate surface (i.e. step 6) need to vote, so that the extended surface curves in the desirable direction. Otherwise, if only the boundary of the surface domain votes, the localization will not be accurate because of the design the stick kernel (See Figure 5.10): irrelevant votes are produced, which can be harmful to nearby features.

Precise localization of discontinuity can be affected by irrelevant votes if curvature at the surface boundary is not considered.

However, differential geometry [18] implies that we are faced with the duality of principal curvatures and surface, i.e., principal curvatures are known if and only if the underlying surface is known. In our surface inference problem, we therefore need a compromise. Instead of performing quantitative curvature estimation, for which second-order derivatives are often unstable, we propose to estimate the signs of the principal curvatures instead. If we known a priori this sign information, we do not need to vote for the other half...
of the stick kernel, and therefore can eliminate unnecessary votes. Also, for feature integration, discontinuity localization can be achieved more efficiently by only voting at the boundary of the intermediate surface.

In fact, the basic formalism described in Chapter 2 mainly concerns with the inference of first order differential geometry structures (i.e. normal or tangent information). There is no provision for inference of second order differential geometry properties, such as curvature information, which indeed provides unique, view-point independent description for local shape. A surface can be unambiguously reconstructed up to second order if the two principal curvatures at each point is known [24].

In Chapter 7, we shall augment the basic formalism with the capability to infer second order curvature information, by proposing an approach based tensor voting that fits nicely in the existing, basic framework.

## 5.5 Results

We produce synthetic and real sparse input data and present different views of the extracted surface. (The real data are sampled using a 3-D digitizer in sweep mode.) Each example emphasizes one or more aspects as described below.

### Plane and sphere

A total of 342 data points with normals are sampled from a sphere intersecting with a plane. Figure 5.11 shows two views of the input data, the extracted intersection curve, and integrated result obtained using our cooperative approach. The intersection curve is precisely inferred. The integrated surface description consists of an upper and lower hemispherical
surfaces, a square planar surface with a circular hole, and a circular planar surface inside the sphere (not visible).

Figure 5.11: Plane and sphere

**Three planes**

A total of 225 data points with normals are sampled from three orthogonal and intersecting planes. Initial estimates are cooperatively refined using our approach. The result is shown in Figure 5.12. There are six extremal curves and one 6-junction which are coherently integrated with twelve extremal planar surfaces. The junction curves are properly localized
Figure 5.12: Three orthogonal planes

*Triangular wedge*

A digitizer is made to sweep over a real triangular wedge, and produces a set of 1266 data points. Successive digitized points form a curvel, so we tensorize the input for obtaining the normals. Then, the cooperative computation is run as described. Depicted in Figure 5.13 is the automatically inferred integrated description in terms of junctions, curves, and surfaces. Six 3-junctions and nine extremal curves, and six extremal surfaces are integrated. The real object is also shown here.
We test our approach to infer segmented and integrated surface, curve, and junction description from stereo. Details can be found in Lee and Medioni [71]. Figure 5.14 depicts the input intensity stereo images and the resulting integrated description. First, an estimate of the 3-D disparity array is obtained in the traditional manner. Potential edgel correspondences are generated by identifying edge segment pairs that share rasters across the images (Figure 5.14(a)). Initial point and line disparity estimations are then made. To infer salient structure from the disparity array, we perform voting for each matched point, from which the SMap is computed. The most salient match are kept along each line of sight (Figure 5.14(b)), using the unique disparity assignment. This filtered, though still noisy, point set is then used as input to our program and the integrated description and the texture mapped surfaces are shown.
Figure 5.13: Triangular wedge
Figure 5.14: Inference of integrated surface, curve, and junction description from stereo (a) input stereo images, (b) noisy point set after assigning unique disparity, (c) resulting surfaces, curves, and junctions. Courtesy of Lee and Medioni.
Chapter 6

Application of the Basic Formalism: Visualization

In this chapter, we apply the essential formalism and feature extraction algorithms presented in previous chapters in the application domain of visualization.

Traditionally, feature extraction and visualization are addressed as separate issues by researchers in different communities. Visualization addresses the proper display of raw data, and processing is left to human users. Feature extraction deals with the automatic or semi-automatic means for structure inference from data, which may be sparse and/or corrupted by outlier noise. Visualization coupled with feature extraction not only allows users to “see” the data, but also provide users with “tangible” means of what to visualize, since the extracted features can be used for further processing and interpretation.

Feature extraction results in a geometric model, such as a surface mesh and/or a set of connected curve segments. With a faithful inferred shape description, we can also perform data validation, and filter out spurious noise which may be unavoidable in many measurement phases. Therefore, visualization coupled with feature extraction should enable users to not only “see” the data, but also “touch” the visualization results, i.e. further processing and interpretation are possible with the availability of inferred features.
We have already reviewed related work in surface and curve fitting in Chapter 4, which is also related to the application domain of visualization. In this chapter, we focus on how we formulate and solve some of the visualization problems by mapping them into our formalism, and the use of additional constraints imposed by specific problems.

We apply the tensor voting formalism in the following applications: (Some of the results were presented in Tang and Medioni [103, 104].)

- Flow visualization
- Vortex extraction
Figure 6.1: Velocity field of Blunt fin
The snapshots show four different slices of the velocity field. As the flow and geometry is symmetrical about the blunt fin, only one half of them is shown (reproduced from FAST, courtesy of NASA).

6.1 Shock wave detection

The first example demonstrates the detection, extraction, and visualization of shock waves, given as input a flow field such as a velocity field in a viscous medium. Figure 6.1 depicts four slices of the velocity field. Note the abrupt change in the flow pattern that creates a shock, as the fluid hits the blunt end of the fin. Figure 6.2 depicts the experimental setup of a Blunt Fin [54], and the velocity field. Air flows over a flat plate with a blunt fin rising from the plate. The free stream flow direction is parallel to the plate and the flat part of the fin, i.e., entirely in the $x$ component direction [54]. Note that the display of the velocity field is basically useless, since it is impossible for users to visualize the presence of a shock wave inherent in this velocity field.
Air flows over a flat plate with a blunt fin rising from the plate, showing the free stream flow direction. Display of the velocity field.

Figure 6.2: Blunt fin.

6.1.1 Problem mapping and results

The presence of a shock wave is characterized by local maxima of the density gradient [85], which is coherent with the definition of surface extremality (Chapter 3), and thus are extracted as extremal surfaces. First, we compute the density field (the left column of Figure 6.3 shows two views of different slices of the density field). Raw local density maxima are extracted in the density field, which results in a sparse set of points. Also, the original data set is sampled on a curvilinear grid. Therefore, a tensor voting pass is needed. Each site in the resulting dense field holds a 2-tuple \((s, \hat{n})\) where \(s\) is the magnitude of density and \(\hat{n}\) denotes the estimated normal. The dense field is input to the extremal surface algorithm. The resulting extremal surface, corresponding to the shock wave known as a “\(\lambda\)-shock” [54] due to its branching structure and shape, is faithfully and explicitly extracted (c.f. [85]) and shown in the right column of Figure 6.3.
The left column depicts two views of different slices of the density field (reproduced from FAST, courtesy of NASA), and the right shows the $\lambda$-shock wave, extracted as extremal surface using our methods, that corresponds to local density extrema having high density.

6.1.2 Insufficiency of iso-surface extraction

To show the inadequacy of iso-surface for such extraction, we sample, at the vertices of the triangulation of the resultant extremal surface, the density values at these extrema. These values are sorted and plotted as a histogram as shown in Figure 6.4.

Density values range from 0.2957 to 4.2266 units, showing that a single iso-density value cannot account for such extraction. Note that only one half of the $\lambda$-shock is extracted due to symmetry; the other half is obtained as a mirror reflection.
6.2 Vortex extraction

This experiment visualizes interacting Taylor vortices, in which the wavy and periodic properties of oscillating vortex cores are properly extracted as an extremal surface. Thus, these interesting properties can be visualized as a single spatio-temporal snapshot, rather than stacks of successive spatial, 2-D data slices, as in Figure 6.5. Using particle image velocimetry, Wereley and Lueptow [123] measured 185 time slices of velocity measurement for the Taylor vortices. Each successive slice represents a sequential time interval of 66.66 msec. This data set constitute the input velocity fields to our system.

While many valid definitions are possible, a single vortex core can conveniently be expressed as the locus having maximal “tangential” velocity (Figure 6.6). However, this characterization may become obsolete when there exists more than one vortices, as in the case of interacting Taylor vortices.
Figure 6.5: Consecutive snapshots of the wavy Taylor vortices. Dots (resp. crosses) indicate current (resp. past) vortex centers. One period of oscillation is shown. (Courtesy of Wereley and Lueptow)

Shown in Figure 6.5(a)–(h) are eight consecutive snapshots of the wavy Taylor vortex flows, averaged over one entire period of oscillation. (Note that Figure 6.5 (a) and (h) show the same field.) The measured period of oscillation is 2.2017 sec. Each image in Figure 6.5 shows a total of three oscillating and interacting vortices, shifting back and forth in the horizontal direction. Vortices A and C are rotating clockwise, while vortex B in anti-clockwise direction. Such adjacent, counter-rotation results in a maximal fluid flow being present between adjacent vortices (Figure 6.7). This maximal flow presents a maximal uncertainty for vortex core localization, since a point on the locus of maximal tangential velocity can legitimately “belong” to both adjacent vortex cores. Therefore, the
algorithm proposed by Ma and Zheng will only work if it is presented with only one, isolated vortex [73]. Basically, this algorithm first locates the vortex center, and computes the extent of the vortex by setting certain velocity thresholds.

### 6.2.1 Vortex segmentation

We can make use of this maximal uncertainty to *segment* or isolate the vortices so that Ma and Zheng’s algorithm for vortex core tracing can be applied. For each data slice, we do the following (see Figure 6.8):

1. apply the 2-D extremal curve algorithm to extract the “front” corresponding to maximal velocity or uncertainty (Figure 6.8(b));
2. extract the local velocity minima which correspond to vortex centers and compute the extent of the vortex cores (Figure 6.8(c)), using the “front” instead of setting arbitrary velocity thresholds. Note that adjacent vortex cores may overlap in the region of maximal uncertainty.

3. apply Ma and Zheng’s algorithm for each isolated spatial region to trace the vortex core, which results in a set of sparse and noisy points approximating each vortex core (Figure 6.8(d)).

### 6.2.2 Vortex extraction

Once a spatial approximation of the vortex core for each slice is obtained as above, we stack the resulting point sets from all temporal slices to produce a spatio-temporal data set in 3-D (shown in left column of Figure 6.10). Note that the resulting point sets produced by Ma and Zheng’s algorithm only approximate the vortex core boundary, using a set of (points constituting the) contours. Many of them are overlapping and misaligned, making most
Figure 6.9: Overlapping and misalignment of the initial, noisy contours

*tiling* techniques for successive contours not suitable in this case (Figure 6.9). Therefore, the interacting vortex cores are extracted as an extremal surface.

Tensor voting is applied to this noisy cloud of points to give a dense vector map (SMap), followed by surface extraction using the extremal surface algorithm. The extremal surface approximating the vortex cores are depicted in the right column of Figure 6.10. Note that the waviness and the periodicity of Taylor vortices depicted in Figure 6.5(a)–(h) are faithfully and conveniently captured in a single, spatio-temporal snapshot.

By observation, there are roughly 5.5 oscillations present in the result. Since there are 185 time slices, each corresponding to 66.66 msec or 1/15 sec, the period of oscillation is $184/15/5.5 = 2.230$ sec which is quite close to the measured period 2.2017 sec given in [123].
6.2.3 Vorticity lines in Taylor vortices

The trajectory of a vortex is best traced by its vorticity line, which is the locus of velocity minima in the case of Taylor vortices (i.e., the red dots in Figure 6.5(a)–(h)). A straightforward way to extract the vorticity line is to join local velocity minima present in successive time slices, producing a linear approximation of the curve (see Figure 6.11(a)). Despite the convenience, the curve thus obtained is very bumpy owing to the low data resolution. Here we seek to derive a smoother, linear approximation with sub-voxel accuracy of the curve (Figure 6.11(b)), by using the 3-D extremal curve algorithm.

![Figure 6.11: Rough versus smooth trajectories](image)

(a) rough approximation by joining local velocity minima in successive data slices, (b) the desired smooth linear approximation.
We regard the rough approximation as shown in Figure 6.11(a) as a sparse velocity field, where each directed segment joining local minima at $t = i$ and $t = i + 1$ as a velocity vector. Therefore, this field needs densification.

The plate voting field for curve is used to densify the sparse velocity field, producing a dense vector field $(s, \hat{\nu})$, where $s$ is the strength corresponding to $\lambda_2 - \lambda_3$, and $\hat{\nu} = \hat{e}_3$. This field serves as input to the extremal curve algorithm. Two views of the extracted vorticity lines, in terms of three extremal curves, are depicted in Figure 6.12. They are smooth with sub-voxel precision. Yet we do not oversmooth since there is still an average of roughly 5.5 complete oscillations present in our result, which agrees with the measured period of oscillation.

![Figure 6.12: Vorticity lines extracted as 3-D extremal curves](image)

(a) and (b) show two views of the bumpy input; (c) and (d) show two views of the refined vorticity lines extracted as 3-D extremal curves, with sub-voxel precision.
6.3 Summary

In this chapter, we have explained how we apply the core theory described in previous chapters for feature inference in 3-D in real, visualization applications. The unified framework makes use of the extended tensor voting formalism and feature extraction algorithms to extract features in 3-D, such as surfaces, 3-D curves, and junctions. This unified computational framework is capable of handling oriented as well as non-oriented data.

We have applied our 3-D system in a range of scientific visualization (such as flow visualization and vortex extraction) to show the general usefulness of our system.
Chapter 7

Inference of Curvature Information: Second Order

Geometric Properties

In this chapter, we augment the basic tensor voting formalism with the ability to infer curvature information (second order geometric property), which is absent from the original, basic tensor voting formalism. This information is then used in the voting process, resulting in a faster and better overall framework.

Curvature information gives a unique, viewpoint independent description of local shape. In differential geometry, it is well known that a surface can be reconstructed up to second order (except for a constant term) if the two principal curvatures at each point is known, by using the first and second fundamental forms [24]. Therefore, curvature information provides a useful shape descriptor for various tasks in computer vision, ranging from image segmentation and feature extraction, to scene analysis and object recognition. The previous effort described in Chapter 5 and also in [101, 102] proposes a procedure to integrate smooth surfaces and junction curves. While good results are obtained, without the use of explicit curvature information, a rather complex coordination process among detected features needs to be implemented.
The contribution of this research is twofold:

- **Improved results on surface fitting:**
  A robust method is proposed for the accurate estimation of sign and direction of principal curvatures. Experiments to qualitatively analyze the results are performed on synthetic and real data with large amount of outlier noise (as much as 500% noise, i.e., one out of six points is good). As a result of the use of curvature information, results on surface fitting are improved, because of the reasons we described earlier in this chapter about curvature.

- **Simplified feature integration:**
  By using explicit curvature information, the process of integrating smooth surfaces and junction curves can be simplified (c.f. Chapter 5 and also in [101, 102]), since we know which side w.r.t. each estimated surface normal (oriented or not) the surface to be inferred should locally curve to. This allows tensor votes to propagate in the proper and preferred directions.

  When considered alone, our curvature estimation can be regarded as a plug-in to other applications, which labels each data point as locally planar, elliptic, parabolic, or hyperbolic, as a discontinuity or an outlier. It runs fast, in $O(nk)$ time where $n$ is the input size and $k$ is neighborhood size (a detailed complexity analysis is given in Chapter 2, which also applies here since our curvature estimation is also implemented as tensor voting). When used as described in this chapter, it provides a stronger mathematical basis to integrate smooth surfaces and junction curves, or surface orientation discontinuities.

  We start by reviewing related work on curvature estimation in section 7.1. Then an overview of the methodology, and our motivation are described in section 7.2.
Section 7.3 and 7.4 detail the tensor voting based approach for estimation of the signs and directions of principal curvatures.

Section 7.5 describes the curvature-based voting kernels used in the dense extrapolation step for surface and curve extraction.

We give both qualitative and quantitative evaluations in section 7.6, and present results on real data in section 7.7.

Finally, in section 7.8 we propose a new feature integration procedure that makes use of the inferred curvature information. It results in a less complex process and a stronger mathematical basis for localization of smooth structures and discontinuities.

### 7.1 Previous work on curvature estimation

A detailed treatment of curvature can be found in a classical differential geometry text by do Carmo [18]. Despite the extensive study on recovery of curvature information from range data and other data sources, results are still not satisfactory. One approach involves fitting a local surface patch, and computing partial second order derivatives from it [4, 93, 98, 119]. Derivative computation is unstable in real data, and the estimated curvature is thus very noise-sensitive. Another approach recovers principal curvatures and direction from range data, by collecting four directional curvatures at 45° apart [24]. Unfortunately, these directional curvatures also rely on accurate, local first and second order partial derivatives, which are often difficult to estimate from real data.

Another methodology involves the recovery of surface normal vectors from the data, and is then usually followed by a local surface patch fitting. With normal information, a better fit, and thus a better curvature estimate, may be obtained. For example, Shi et al. [98]
diagonalize a scatter matrix for normal estimation from a set of sampled surface points. A least square process for fitting a local quadric patch is then followed. Unfortunately, parameterizing such local patch requires that the orientation of the estimated normals be consistent throughout the whole surface, which is either unavailable, or has to be estimated separately.

Rather than numerically computing curvature information, another approach involves the estimation of the sign of Gaussian curvature. For example, Angelopoulou and Wolff [2] compute the sign of Gaussian curvature, without surface fitting, local derivative computation, nor normal recovery. The sign of Gaussian curvature is determined by checking the relative orientation of two simple, local closed curves (one from the surface and one from its corresponding curve on the Gaussian sphere) is preserving or otherwise reversing. However, zero Gaussian curvature areas need to be first located in a separate process.

Parent and Zucker [86] propose to infer 2-D curves based on curvature information. In particular, to enforce the co-circularity constraint (analogous to co-surfacity in 3-D), they use curvature class partitioning in their trace inference stage, which bears some resemblance to our use of voting kernels. They select the highest support among the curvature classes at each local site, and then use a disambiguity step to ensure curvature consistency (smoothness). In our case, local orientation estimation, as well as the enforcement of the smoothness constraint, is unified by our voting process. Singularities are detected as disagreement in oriented votes collected at each site.

Empirical analyses on curvature estimation are reported in literature. The classical paper by Flynn and Jain [27] evaluates five methods of curvature estimation. The conclusion of the experiments performed by Trucco and Fisher [117] agree with [27]: qualitative curvature properties (e.g. sign of Gaussian curvature) can be more reliably estimated than
quantitative ones (e.g. curvature magnitude). Our method agrees with their conclusion, in addition we show (by example) that principal directions can also be estimated fairly robustly by our method, while this estimation is not addressed in [27, 117].

7.2 Overview and motivation

Our method, which is based on the tensor voting formalism, robustly recovers the sign and direction of principal curvatures for surface reconstruction directly from 3-D data, without surface fitting nor partial derivative computation of any kind. Zero curvature areas are detected and handled uniformly, by using homogeneous coordinates (section 7.3). The basis of our method is grounded on two elements: local structures are uniformly represented by a second order symmetric tensor, which effectively encodes preferred direction, while avoiding early decision on normal orientations and maintenance of global orientation consistency. Data communication is accomplished by a linear voting process, which simultaneously ignores outlier noise, corrects erroneous orientation (if given), and detects surface orientation discontinuities. While approaches at one extreme completely trust the estimated normals (e.g. [98]), and methods at the other extreme completely bypass the normal recovery process (e.g. [2]), our method is more flexible: it makes use of reliably inferred surface orientation information, corrects erroneous normals, and ignores inconsistent votes.

Many affordable laser range finders can now produce dense and mostly accurate information. However, outlier noise is still unavoidable in the measurement phase (for example, see Figures 7.14 and 7.17). With dense, mostly accurate but imperfect data, if we can
robustly estimate curvature information, and use it to tune the dense voting kernels accordingly in voting, more reliable surface reconstruction should result. To elaborate, suppose that we vote near the endpoints of a circular arc in 2-D (Figure 7.1) to fill the gap. Without the use of curvature information, using Guy and Medioni’s method [36], or the basic formalism by Lee and Medioni described in [72], a circle will not be produced since a straight connection (zero curvature) is preferred.

Figure 7.2 shows the augmented formalism. The flowchart is directly adapted from Figure 2.1 which depicts the basic formalism. We augment the flowchart with an additional voting pass for curvature estimation.

*First order orientation information* (normals or tangents) is inferred after the first tensor voting pass in the token refinement stage, which is performed exactly as described in Chapter 2.

*Second order curvature information* is obtained in a second tensor voting pass. This pass is applied to the inferred tensors at each input site. The resulting tensor field, together with the inferred curvature information, is “densified”, by using curvature-based voting fields, in the dense extrapolation stage for subsequent surface and curve extraction.

In the next section, we describe this additional voting pass for curvature estimation.
Figure 7.2: Augmented formalism (with curvature estimation)
7.3 Estimation of the sign of principal curvatures

Recall that, as described in Chapter 2, after the token refinement step, each input site is replaced by a set of true ellipsoids encoding preferred surface normal and curve tangent information. Each input tensor then votes again in order to estimate the sign and direction of the principal curvatures (the shaded process in Figure 7.2).

First, we analyze each input ellipsoid, by tensor voting, to label it as locally

- planar
- elliptic
- parabolic
- hyperbolic, an outlier, or a discontinuity

When this process is done, each input site will locally know which side w.r.t. its stick component the surface should locally curve to, except for the last case. Curvature-based stick kernels are then generated accordingly for the dense extrapolation step for feature extraction that follows. Thus, we do not need to vote on both sides of stick as shown in Figure 7.3, which illustrates the general shape of the 3-D stick voting field. Here, both sides of the stick vote, since no curvature information is used.
7.3.1 Representation of sign of curvature

For each input tensor, we arbitrarily pick an orientation out of the two choices of its stick component $\hat{e}_1$ as reference. Then, we align with it a local coordinate system. The sign of curvature is indicated by the side w.r.t. the oriented $\hat{e}_1$ the surface should locally curve to. *Homogeneous coordinates* are used to represent the sign of curvature so that zero curvature can be handled uniformly.

To elaborate, refer to Figure 7.4. Let $O$ be the input site with its arbitrarily oriented stick $\hat{e}_1$. W.l.o.g., consider $P$, $Q$, and $R$ in the neighborhood of $O$, $\text{nbhd}(O)$, as specified by the input scale or the size of the voting field. Let $N_p$ (resp. $N_q$ and $N_r$) be the stick component of the input ellipsoid at $P$ (resp. $Q$ and $R$) obtained after the token refinement stage.

We want to estimate the signs of principal curvatures at $O$, and hence $O$ is designated as the vote collector. We align the 3-D stick kernel at $O$ (with its oriented $e_1$, usual). Note that we are abusing the use of “vote collector” here, since so far in our discussion only voters are aligned with the voting kernels, but not vote collectors. However, the discussion is still valid.
Table 7.1: Sign of curvature vote collected at $O$

Now, $O$ collects the stick votes induced by its aligned stick kernel at $P$, $Q$, and $R$. Let the direction of these stick votes be $N^o_p$ (resp. $N^o_q$ and $N^o_r$). The direction and strength of these votes are defined as tabulated in Table 7.1.

In the table, $\text{dist}(\cdot)$’s denote the Euclidean distance.

Note that the dot product in the vote strength definition in Table 7.1 indicates vote consistency, i.e., if the magnitude of the dot product is close to zero, it means that the stick vote $N^o_p$ (resp. $N^o_q$ and $N^o_r$) is not consistent with $N_p$ (resp. $N_q$ and $N_r$). One or both of the following events occur:

- the ellipsoid (or its stick component) inferred during the token refinement stage at $O$ and/or $P$ is not accurate
- the smoothness constraint prescribed by the stick kernel is not satisfied

These situations may arise for severely corrupted data or at a discontinuity. In either case, the vote is unreliable and should be ignored by the dot product.

### 7.3.2 Vote collection

$O$ is a vote collector, which aggregates the sign of curvature vote (with its direction and strength defined above) cast by a point $P$ in its own neighborhood. Denote such a vote by
$v_p$, where $||v_p||$ defines the vote strength as above. The voting process is exactly the same as described in Chapter 2, but the formulæ for aggregating collected votes is different, as explained below.

We compute the (sample) mean $M$ and covariance matrix $S$ of the vote distribution.

\[
M = \begin{bmatrix} M_x \\ M_y \end{bmatrix} = \frac{1}{n} \sum_{p \in \text{nbhd}(O)} v_p \tag{7.1}
\]

\[
\mu = \frac{M_x}{M_y} \quad \text{(note that } M_y > 0) \tag{7.2}
\]

where $n$ is the number of points in $\text{nbhd}(O)$. For $p = 1, 2, \cdots, n$, define

\[
\overrightarrow{v}_p = v_p - M \tag{7.3}
\]

\[
B = \begin{bmatrix} \overrightarrow{v}_1 & \overrightarrow{v}_2 & \cdots & \overrightarrow{v}_n \end{bmatrix} \tag{7.4}
\]

Therefore, $\overrightarrow{v}_p$ represents the deviation of $v_p$ from the “weighted-averaged sign” $M$. Using $B$, we compute the covariance matrix $S$ as the $2 \times 2$ positive semidefinite matrix, and the total variance $\Sigma$ of $S$ is the trace of $S$, as follows:

\[
S = \frac{1}{n-1}BB^T \tag{7.5}
\]

\[
\Sigma = \text{trace}(S) \tag{7.6}
\]

### 7.3.3 Geometric interpretation

$\mu$ and $\Sigma$ together indicate which side w.r.t. to the stick at $O$ the (recovered) surface should locally curve to. We have the following cases (Figure 7.5):
Figure 7.5: Geometric interpretation of vote collection

- $|\mu| \approx 0$, $\Sigma \approx 0$. Curvature is zero (i.e. within an empirical tolerance) in $nbhd(O)$. It indicates that $O$ is locally planar (Figure 7.5(a)).

- $|\mu| \neq 0$, $\Sigma \approx 0$. Points in $nbhd(O)$ prefers smooth connection ($\Sigma \approx 0$), on only one side of the stick at $O$ ($|\mu| \neq 0$). It indicates that $O$ is locally elliptic (Figure 7.5(b)).

- $|\mu| \approx 0$, $\Sigma \neq 0$. Curvature votes cancel out each other, as indicated by non-zero $\Sigma$. Points in $nbhd(O)$ do not prefer either side of the stick, which indicates that $O$ is locally hyperbolic, or an outlier, or a discontinuity (Figure 7.5(c)).

- $|\mu| \neq 0$, $\Sigma \neq 0$. Curvature votes indicate that one side of the stick is preferred. Yet, there exists votes with zero curvature (implied by non-zero $\Sigma$). This indicates that $O$ is locally parabolic (Figure 7.5(d)).

If $O$ is inferred to be locally hyperbolic, an outlier, or a discontinuity, then we use the original dense stick kernel defined in Chapter 2 for densification, because of the inconclusive vote. Note that, although sign estimates are unreliable at regions of surface orientation
discontinuities, these singularities are detected as point and curve junctions at the token refinement stage, which are characterized by a high disagreement of oriented votes collected at such sites. No surface patch is produced in these regions of low surface saliencies where a surface saliency extremum does not exist.

If $O$ is labeled as locally planar, we use the original kernel, but redefine the decay function such that it decays more with higher curvature.

A detailed discussion on curvature-based stick kernel is given in section 7.5.

### 7.4 Principal direction estimation

#### 7.4.1 Vote definition

We define the vote for principal directions as follows. Refer to Figure 7.6.

Let $N_p$ be the stick component of the ellipsoid inferred at $P$ after the token refinement stage. It casts a vote (using the 3-D stick kernel) which is received at $Q$. Let $N$ be the direction of this stick vote (Figure 7.6(a)). Then, by the definition of the 3-D stick kernel, which is derived directly from the fundamental 2-D stick kernel (Chapter 2), $N$ at $Q$, $N_p$ at $P$ must lie on the same plane. We denote this plane by $\Pi_{PQ}$ (Figure 7.6(b)).

Refer to Figure 7.6(c). Let $N_q$ be the stick component of the ellipsoid inferred at $Q$ after the token refinement stage. (Note that $N_q$ and $N$ are not necessarily in the same direction.) Let $T_q$ be the tangent plane at $Q$, by assuming that $N_q$ is the direction of surface normal.

Then, $N_q \perp T_q$. Any violation of this assumption is indicated by vote inconsistency, as described below.
Figure 7.6: Illustration of the vote definition for principal directions
(a) Voter at P casts a vote at Q by using the stick kernel, (b) the tensor at P and the stick vote at Q define a plane $\Pi_{PQ}$, (c) the tensor at Q also defines a plane, which intersects with $\Pi_{PQ}$. Their intersection gives the direction that P “curves to” Q, or the curvature direction.
If \( T_q \cap \Pi_{PQ} \neq \emptyset \) (otherwise we simply skip the following), this intersection gives a line (indicated by the dotted line in Figure 7.6(c)). We define the direction and strength of the vote as follows:

- **Direction.** We put a vote \( \bar{v} \) along the line \( T_q \cap \Pi_{PQ} \), because \( P \) “curves to” \( Q \) along this direction (Figure 7.7).

![Figure 7.7: Principal direction vote on \( T_q \)](image)

- **Strength.** Scale and project the curvature along the circular arc (denote it by \( \rho \)) onto \( N_q \) by

\[
\frac{|\rho|}{dist(P, Q)} \frac{N}{||N||} \cdot \frac{N_q}{||N_q||}
\]

This definition prefers directions on the tangent plane \( T_q \) which curve *more* (as indicated by \( |\rho| \)), giving more weight to such directions.

In other words, we vote for the *maximum direction* (as the minimum direction is merely \( 90^\circ \) off).

If \( \frac{N}{||N||} \cdot \frac{N_q}{||N_q||} \approx 0 \), it means that the stick vote \( N \) cast by \( P \) is not consistent with \( N_q \) (this unreliable vote will thus be ignored).
7.4.2 Vote collection

The votes \( \mathbf{v}'s \) on the tangent plane \( T_q \) are collected as a second order symmetric tensor in 2-D, or equivalently as an *ellipse*. This topological ellipse (Figure 7.7) describes the equivalent eigensystem with its two unit eigenvectors \( \hat{e}_{mx} \) and \( \hat{e}_{mn} \) and the two corresponding eigenvalues \( \lambda_{mx} \geq \lambda_{mn} \). Rearranging the eigensystem, the ellipse is given by: \( (\lambda_{mx} - \lambda_{mn})S + \lambda_{mn}B \), where \( S = \hat{e}_{mx}\hat{e}_{mx}^T \) defines a *stick tensor*, and \( B = \hat{e}_{mx}\hat{e}_{mx}^T + \hat{e}_{mn}\hat{e}_{mn}^T \) defines a *ball tensor*, in 2-D.

7.4.3 Geometric interpretation

The *eigenvectors* denote the *principal directions*: \( \hat{e}_{mx} \) (resp. \( \hat{e}_{mn} \)) gives the maximum (resp. minimum) direction, since after the principal component (eigensystem) analysis, the eigenvector corresponding to the larger eigenvalues indicates the principal direction along which the underlying surface should curve.

However, the eigenvalues, do *not* indicate the magnitude of principal curvatures. In fact, if we align a local coordinate system at a point, the value of curvature at that point in direction \( \phi \) is given by [24]:

\[
    k_\phi = k_{max} \cos^2(\phi - \alpha) + k_{min} \sin^2(\phi - \alpha)
\]

(7.8)

where \( \alpha \) is the maximum curvature direction. Note that Equation (7.8) does *not* follow an elliptic distribution. Therefore, it cannot be described by a second order symmetric 2-D tensor, which follows an elliptic distribution.
In fact, the general shape of the distribution resembles a peanut [24], not an ellipse. Our experiments tends to over-estimate (resp. under-estimate) of maximum (resp. minimum) curvature if we assume an elliptic distribution.

However, given that the strength of votes is properly defined, (Equation (7.7)), this elliptic distribution assumption should still valid for approximating principal directions, if only the direction of the resulting eigensystem is considered (and confirmed by our experiments).

### 7.5 Curvature-based stick kernels

(b) stick kernel for locally parabolic or elliptic voxel

Since we can only estimate the signs and the directions of principal curvatures reliably, but not the magnitude, we only use the sign information to derive the curvature-based stick kernel for the dense extrapolation and coherent surface and curve extraction. The magnitude of curvature is known to be hard to estimate, because of the instability of second order estimates. Tensor voting does not involve such unstable, partial derivative computations. Also, knowing the principal directions should help recovering the magnitude, which is the subject of future research. According to the label inferred for each input ellipsoid, we have the following cases:

- **Hyperbolic region, outlier or discontinuity.**

  Voxels labeled as hyperbolic, outlier or discontinuity are characterized by inconclusive curvature votes. We use the original stick kernel as defined in Chapter 2 for dense extrapolation.
• *Planar.*

If voxels are labeled as locally planar, we use the same stick kernel, but impose more decay with high curvature, i.e. \( DF(P) = e^{-\left(\frac{c^2 + C\psi^2}{\sigma^2}\right)} \), \( C \gg c \) where \( c \) is the attenuation factor in Equation (2.6). Figure 7.8(a) shows one slice (at \( y = 0 \)) of this stick kernel. Note the “flatness” of the kernel, as opposed to the kernel in Figure 2.4. This kernel prefers planar connection.

• *Parabolic or elliptic.*

If voxels are labeled as locally parabolic or elliptic, we only consider the set of directions and strengths of the stick kernel for which \( \text{sgn}(\mu) \text{sgn}(\rho) > 0 \), i.e., one side of the stick kernel, upon kernel alignment. See Figure 7.9.

### 7.6 Evaluation

In this section, we evaluate the augmented tensor voting formalism in the following criteria:

- **Accuracy of labeling (quantitative):** By estimating the sign of principal curvatures as described in section 7.3, we can label each input site according to the detect curvature information.

- **Grouping by curvature (qualitative):** More complex scenarios, such as noisy input, are used as examples here to perform the curvature labeling and grouping.

- **Accuracy of estimated principal directions (quantitative):** Principal directions are estimated as described in section 7.4, using a synthetic input with various amount of
Figure 7.8: Stick kernel for voxel labeled as locally planar

Note the “flatness” of this stick field (c.f. Figure 2.6). This field prefers planar connection.
Figure 7.9: Stick kernel for voxel labeled as locally elliptic or parabolic

*Only one side of the original stick kernel (c.f. Figure 2.6) is considered.*
additive, uncorrelated noise. The resulting directions are quantitatively evaluated by comparing to the ground truth.

- Quality of surface fitting (qualitative): The quality and the degradation of the inferred surface in the presence of various amount of noise is studied. The results are also compared with those obtained from the original, no-curvature formalism.

### 7.6.1 Accuracy of labeling

We perform experiments on synthetic data to evaluate the accuracy and robustness of our tensor voting approach for curvature estimation.

We perform similar quantitative evaluation on the labeling accuracy of our method on primitive surfaces, as in [27]. Point samples are collected from three primitive surfaces: a spherical, a parabolic, and a hyperbolic surface. We then add a large amount of random noise to the bounding box of each shape.

(A) SPHERE. The curvature labeling for a sphere is over 95% correct. Curvature-based kernel (elliptic) is used for surface inference.

(B) CYLINDER. In this case, the boundary of the domain and region with sparser samples are more error prone, else we still have over 95% accuracy. Since parabolic vote is still the majority, by tensor voting, a faithful cylindrical surface can be produced.

(C) SADDLE. The saddle region can be detected. However, when a point gets farther away from the center of the saddle, locally it bears less resemblance to a saddle. But we can still infer a faithful surface.

Table 7.2 summarizes the result. Note that over 95% of accuracy is still reported with as much as 200% of outliers, i.e., only one out of three points is correct: the boldfaced
(resp. plain) text on the leftmost column indicates the percentage of correct (resp. incorrect) labels.

**Sphere (489 points)**

<table>
<thead>
<tr>
<th></th>
<th>50% noise</th>
<th>100% noise</th>
<th>150% noise</th>
<th>200% noise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>elliptic</strong></td>
<td>100.00%</td>
<td>100.00%</td>
<td>99.40%</td>
<td>95.88%</td>
</tr>
<tr>
<td><strong>parabolic</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>4.12%</td>
</tr>
<tr>
<td><strong>planar</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.60%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>hyperbolic</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

**Cylinder (3844 points)**

<table>
<thead>
<tr>
<th></th>
<th>50% noise</th>
<th>100% noise</th>
<th>150% noise</th>
<th>200% noise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>elliptic</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>parabolic</strong></td>
<td>97.89%</td>
<td>97.41%</td>
<td>99.60%</td>
<td>96.30%</td>
</tr>
<tr>
<td><strong>planar</strong></td>
<td>0.17%</td>
<td>1.85%</td>
<td>0.17%</td>
<td>2.72%</td>
</tr>
<tr>
<td><strong>hyperbolic</strong></td>
<td>1.94%</td>
<td>0.74%</td>
<td>0.23%</td>
<td>0.99%</td>
</tr>
</tbody>
</table>

**Saddle (605 points)**

<table>
<thead>
<tr>
<th></th>
<th>50% noise</th>
<th>100% noise</th>
<th>150% noise</th>
<th>200% noise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>elliptic</strong></td>
<td>0.00%</td>
<td>0.33%</td>
<td>0.67%</td>
<td>0.50%</td>
</tr>
<tr>
<td><strong>parabolic</strong></td>
<td>0.17%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>planar</strong></td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>hyperbolic</strong></td>
<td>99.83%</td>
<td>99.67%</td>
<td>99.33%</td>
<td>99.50%</td>
</tr>
</tbody>
</table>

Table 7.2: Accuracy on curvature labeling

**7.6.2 Grouping by curvature**

Here, we demonstrate the use of faithfully inferred curvature information for performing scene segmentation in more complex synthetic scenario. By making use of curvature, we can perform the integrated surface inference in a less complex way than in [101].
(A) Sparse Plane-Sphere. A sparse set of points is sampled from a sphere intersecting with a plane. Having accurate curvature estimation, we can segment the scene into the corresponding spherical and planar components (Figure 7.10).

(B) Noisy Saddle-Cylinder. We sample data from a hyperbolic surface intersecting with a cylindrical surface, and randomly add a considerable amount of outliers in the volume. We can recover curvature information and segment the input features (Figure 7.11).

7.6.3 Accuracy of estimated principal directions

A total of 3969 point samples are obtained from a toroidal surface, a genus-one object. We add a large amount (up to 1200%) of outlier noise in the corresponding bounding box. Figure 7.12 is a histogram showing the accuracy of the estimated principal directions at different noise levels.

7.6.4 Quality of surface fitting

Figure 7.13 shows the noisy input and the reconstructed surface. In each case, the principal directions at each input point can still be reliably estimated (as shown in the last section), and that our method degrades very gracefully with increasing amount of outliers. Note that the reconstructed surface can be extracted even up to 400% noise, and most parts of it can still be extracted in the presence of as much as 1200% noise. Here, we also include the results on the same noisy input run on the original, basic 3-D system without the use of curvature information. The latter results clearly show a faster rate of degradation, in terms of the quality of surface fitting.
Figure 7.10: Sparse plane-sphere
points labeled as hyperbolic

points labeled as parabolic

Figure 7.11: Noisy saddle-cylinder
Figure 7.12: Graceful degradation of the estimated curvature direction for a torus

The estimated direction is not adversely affected by outlier noise.
Figure 7.13: Noisy input and the corresponding reconstructed surface for a torus. The results on the same input using the basic formalism without curvature is also shown in the last column, showing a faster rate of degradation.
7.7 Results on real data

A good choice of real data for evaluating our system is data from the dental CAD/CAM area. A patient-specific dental restoration involves complex, distinct but close-by surfaces. For example, a crown restoration consists of an upper surface (the occlusal surface); and a lower surface which precisely adjusts to the preparation, along a very precise preparation line (which is only implicit in the input point set) that should not be smoothed out during the inference of the surface model.

(A) CROWN. A set of 9401 points is sampled from a crown restoration. Figure 7.14 shows the noisy input data, the extracted surfaces and 3-D crease curves. By using curvature information, we can segment the two distinct but very close-by surfaces, as shown in the two slices. Also, since we generate considerably less irrelevant votes by voting only on one side of the input stick in the preferred curvature direction, we made an improvement over a previous work [104] on the same data. Significantly less spurious surface patches are produced along the preparation line: a curve junction (low surface but high curve saliencies) corresponding to surface orientation discontinuity. In Chapter 5 and in [101, 102], a rather elaborate and complex process is described for feature integration that does not take curvature information into consideration, accounting for the use of over 60000 points for a comparable result.

(B) MOD. A set of 4454 points is sampled from an onlay restoration. Figure 7.15 shows the input data, the extracted surface model, and the inferred creases and other anatomical lines which are only implicit in the data.
Figure 7.14: Results on surface and curve inference from noisy *Crown* data
Figure 7.15: Results on surface and curve inference from noisy Mod data
(C) FEMUR. A set of 18224 points is sampled from femur (courtesy of INRIA), to which we add 400 outliers. A femur is the proximal bone of the lower limb. We infer the surface description from the noisy data, and label the regions detected as negative Gaussian curvature (saddle) as red, Figure 7.16.

![two views of the noisy data](image)

![three views of the reconstructed surface](image)

Figure 7.16: Results on surface reconstruction for femur data

(D) BUST. A set of 57712 points is obtained from dense stereo, bust, a bald human head model. This is a noisy data set, with very few data points on the top of the head (Figure 7.17). Here, interesting regions of negative Gaussian curvature are labeled, such as
the eyes, the area between the nose and cheek, the back of neck, and regions close to both ears. Figure 7.17 also shows the faithful reconstructed surface.

7.8 Feature integration using curvature information

In the previous section on evaluation, emphasis is on the accuracy of the curvature information obtained which results in the improved quality of surface fitting. Here, in this section, we further make use of the accurately inferred curvature information for performing feature integration.

In Chapter 5, we proposed an extension to the basic formalism for performing feature integration. Although the whole integration framework is based on the tensor voting formalism, the proposed method is still rather complex.

Here, we propose a new feature integration methodology based on the augmented formalism we have described in previous sections. By making use of curvature information to integrate detected features, the integration process is simpler, more efficient, and can achieve better results.

7.8.1 Overall approach

We now use the augmented tensor voting formalism described in this chapter, and the feature extraction algorithms, to perform feature integration. Therefore, after the token refinement and curvature estimation stages, a tensor inferred at each input site contains two pieces of geometric information:

- first-order, orientation information, as given by the stick component of the inferred tensor
Figure 7.17: Results on surface reconstruction for *bust* data
second-order, curvature information, which consists of the sign and directions of principal curvatures

The dense extrapolation step involves the use of curvature-based stick voting fields for inference of initial junctions from JMap, curves from CMap, and surfaces from SMap.

First, we need to mask off the voxels in the neighborhood in the CMap (resp. SMap) that correspond to salient junctions (resp. junctions and curves) detected in the JMap (resp. JMap and CMap). This is similar to the use of inhibitory fields introduced in Chapter 5. We shall call this neighborhood *mask off neighborhood*.

The JMap, and the modified CMap and SMap are then used in the integration process, which is summarized in the flowchart shown in Figure 7.18.
With the abuse of language, in this section and also the flowchart, we still refer the modified SMap (resp. modified CMap) as SMap (resp. CMap) without confusion.

Recall that the inferred surface patches away from the detected point and curve junctions are very reliable, because they correspond to extrema having a high surface saliency values. So, they are retained in the final integrated description.

Therefore, it remains to integrate smooth structures and detected discontinuities in the mask off neighborhood. This process is outlined as follows:

1. **Surface-curve integration.** Inferred tensors are used to localize curve junctions in the mask off neighborhood in SMap, so that the corresponding two intersecting surfaces will meet precisely at the localized curve junction.

2. **Surface-junction integration.** Inferred tensors are used to localize point junctions in the mask off neighborhood in SMap.

3. **Curve-junction integration.** Localized curve and point junctions obtained above vote again for inferring the curve extension toward the junction in the mask off neighborhood in CMap.

### 7.8.2 Surface-curve integration

Here, we have three steps in order to perform surface-curve integration. They are described in the following:
**Tensor grouping**

For each cuboid (made up of eight neighboring voxels) in the SMap in which a curve segment is detected in the CMap, there must be a corresponding mask off neighborhood. We partition the input tensors (inferred after the two tensor voting passes) just **outside** its mask off neighborhood into two groups.

Each group is a set of inferred tensors whose stick orientations are consistent with an underlying surface, and that the two respective underlying surfaces from each group should intersect to produce a curve junction.

A tensor voting pass, which runs in the token refinement mode, is done to perform such grouping. This is illustrated in Figure 7.19.

---

**Figure 7.19: Tensor grouping**

We can estimate the two possible surfels at the curve junction by using each inferred surface incident to the mask off neighborhood (indicated by the “hollow” arrow in one example above). Then, a tensor voting pass is performed. Tensors whose stick component orientations are consistent with the estimated surfel (“hollow” arrow) are categorized into the same group. This figure shows the different cases, in which the tensors are categorized into two disjoint groups.
Surface extension in the mask off neighborhood in SMap

Tensors in each respective group now vote with curvature-based stick kernel. Only the surface elements that lie inside the mask off neighborhood will be collected. The result is that two surfaces intersect with each other (Figure 7.20) in the mask off neighborhood. This intersection gives a curve segment, which corresponds to the locus of points where the two surfaces intersect. In other words, we have just localized one curve segment.

![Figure 7.20: Two intersecting surfaces inside the mask off neighborhood](image)

Connecting curve segments

The above two processes are applied to the subset of cuboids in the SMap that have a curve segment detected in the CMap. It produces in a set of curvels. A tensor voting pass, in dense extrapolation mode, is applied to these curvels for curve inference. A smooth curve representing the localized surface orientation discontinuity is thus obtained.
7.8.3 Surface-junction integration

Curvel grouping

For each cuboid (made up of eight neighboring voxels) in the CMap in which a point junction is detected in the JMap, there is a corresponding mask off neighborhood. We group the curve segments (curvels) incident to this mask off neighborhood by using the detected junction, as illustrated in Figure 7.21. Note that these curve segments are obtained after the surface-curve integration described in the previous section.

![Diagram showing mask off neighborhood and point junction with 3 groups: {{A,B},{B,C},{A,C}}](image)

Figure 7.21: Grouping curve segments incident to the mask off neighborhood

Surface extension in the mask off neighborhood in SMap

Refer to Figure 7.22. Curvels in each respective group will vote, in the token refinement mode, with the plate voting field for normal inference, Figure 7.22(a). The normals obtained then vote, now in the dense extrapolation mode, for inferring a surface. Only the surface elements that lie inside the mask off neighborhood are collected, Figure 7.22(b). The results are a set of surfaces intersecting with each other in the mask off neighborhood. This intersection gives a curve segment,
Figure 7.22: Surface extension inside the mask off neighborhood
(a) Curvels from curves A and B vote with the plate voting field for normal inference. The same operation is done for B and C, and for A and C. (b) A surface consistent with curvels from A and B is collected inside the mask off neighborhood. The same operation is done for B and C, and for A and C. (c) Junction localization.

Junction localization

The centroid of the smallest finite neighborhood that contains all the surfaces produced in the above step generates the localized junction. See Figure 7.22(c).

7.8.4 Curve-junction integration

The localized junction obtained above, and the curvels along the curve incident to the mask off neighborhood of the localized junction vote together to infer the curve extension toward the localized junction. See Figure 7.23.

7.8.5 Results on integrated structure inference

In this section, we present results on the upgraded version of feature integration, which makes use of the estimated curvature information.
We first use the two examples from section 7.6.2, the Sparse Plane-Sphere and the Noisy Saddle-Cylinder. Recall that in section 7.6.2, we have already performed the curvature labeling, which is very accurate even in the presence of outlier noise. We proceed here to infer the integrated surface descriptions, which are shown in Figure 7.24.

Next, we use two examples from [127]. Their level-set method produces very good results on the noiseless version of the same examples, Pipe and Two cones. These are difficult examples even in the noiseless case, because they involves topological changes, resulting in non-manifold areas and surface orientation discontinuities. Our new feature integration scheme not only detects the smooth structures, localizes the curve and point junctions, but also works in the present of a large amount of outlier noise. The level-set method, which uses a distance measure, is likely to fail on our noisy inputs.

**Pipe**

A set of 2455 points are randomly sampled on the surface of a solid pipe. The inner and the outer radii of the pipe are 25 and 35 units, respectively. Then, we add 100% random noise in the bounding box containing the sampled points.
Figures 7.25 and 7.26 show two views of the resulting noisy input, the inferred normals (first-order differential geometry information), the inferred signs of principal curvatures (second-order differential information), for which we label the input points as locally cylindrical (shown in green or lighter shade) or planar (shown in black), the localized junction curves, and the integrated surface description. Here, we also include the surface result without performing integration, showing that the curve junctions are smoothed out in the surface description without integration.

Two cones

A set of 4518 points are sampled on the surfaces of two solid cones, which touch each other at their respective apex. The heights and radii of the two cones are both 20 units. We add 100% random noise in the bounding box containing the sampled points.

Figures 7.27 and 7.28 show two views of the resulting noisy input, the inferred normals (first-order differential geometry information), the inferred signs of principal curvatures (second-order differential information), for which we label the input points as locally cylindrical (shown in green or lighter shade) or planar (shown in black), the localized junction curves, and the integrated surface description. Here, we also depict the result without performing integration, showing the smoothing out of the curve junctions, and the presence of spurious surface patches, in the resulting surface description without integration.

7.9 Summary

We have described an approach that infers sign and direction of principal curvatures directly from the input data, which may be sparse and noisy, and uses this information for
coherent surface and curve extraction. No partial derivative computation or local surface fitting is performed, which are very unstable in real data because outlier noise is not uncommon. Also, zero curvature is handled uniformly. Curvature information is incorporated into an existing computation framework, and we have shown results on real, noisy and complex data. We also propose the use of curvature information for feature integration which results in a simpler procedure and fits more nicely in the current tensor voting framework as well.
Figure 7.24: Integrated description for Sparse Plane-Sphere and Noisy Saddle-Cylinder
Figure 7.25: Pipe – one view

(i) noisy input points, (ii) inferred normals, (iii) inferred curvatures, (iv) inferred junction curves, (v) integrated description, (vi) surface description without integration (note that the curve junctions are smoothed out).
Figure 7.26: Pipe – another view
Figure 7.27: Two cones – one view
(i) noisy input points, (ii) inferred normals, (iii) inferred curvatures, (iv) inferred point and curve junctions, (v) integrated description, (vi) surface description without integration (note the spurious surface patches around discontinuities).
Figure 7.28: Two cones – another view
Chapter 8

Higher Dimensional Tensor Voting and its Applications

In this chapter, we generalize the tensor voting formalism into higher dimensions from the 2-D and 3-D versions, and propose useful applications of it.

In higher dimensions $n$, surface-ness becomes hypersurface-ness (a geometric variety of dimension $n-1$), which is defined by a stick tensor (a vector) normal to the hypersurface in the $n$-D space. The generalization of the concept can in fact be readily made. The challenge mainly lies on the implementation, since the dense voting stage produces values in the entire $n$-D space. Memory management and specialized data structures are issues, which need to be properly addressed.

Section 8.1 generalizes the tensor voting formalism into any dimensions. Section 8.2 generalizes the extremal feature algorithm to $n$-D.

In section 8.3, we motivate and propose an approach based on the higher dimensional tensor voting framework for solving the problem of epipolar geometry estimation. The corresponding 8-D tensor voting system for this problem is described in sections 8.4.

Section 8.5 describes the implementation issues and the solution for the dense extrapolation voting step, and extrema detection.
Other issues in epipolar geometry will be discussed in section 8.6. Complexity analysis for the high dimensional tensor voting is given in section 8.7. Finally, we present results in section 8.8.

8.1 \textit{n-Dimensional Tensor Voting}

In this section, we describe how to generalize the tensor voting formalism to \textit{n}-dimension for any \( n > 2 \), using the 2-D version as the basis. The 2-D tensor voting formalism, as reviewed in Chapter 2 and in the book [78], can be generalized to \textit{n}-D readily. For this reason, we shall relate the 2-D version to facilitate our \textit{n}-D discussion. Table 8.1 summarizes the key generalization components. In this section, we focus on the multidimensional version of voting as follows:

- \textit{n}-D tensor representation
- \textit{n}-D tensor communication

8.1.1 Tensor representation

8.1.1.1 Second order symmetric tensor in \textit{n}-D

A point in a 2-D space can either belong to a \textit{curve}, be a \textit{point junction} where intersecting lines meet, or be an \textit{outlier}. In the 2-D version, we use a second order symmetric tensor in 2-D as the representation. This can be visualized geometrically as an \textit{ellipse} (Figure 8.1). This ellipse can be described by the equivalent \( 2 \times 2 \) eigensystem, with its two unit eigenvectors \( \hat{e}_1 \) and \( \hat{e}_2 \), and the two corresponding eigenvalues \( \lambda_1 \geq \lambda_2 \), which is given by:

\[
(\lambda_1 - \lambda_2)S + \lambda_2 B,
\]  

(8.1)
<table>
<thead>
<tr>
<th>Feature to be inferred</th>
<th>2-D</th>
<th>$n$-D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representation</strong></td>
<td>curve</td>
<td>hypersurface</td>
</tr>
<tr>
<td><strong>Size of eigen-system</strong></td>
<td>$2 \times 2$</td>
<td>$n \times n$</td>
</tr>
<tr>
<td><strong>Stick basis</strong></td>
<td>absolute certainty in tangential direction along a single 2-D direction</td>
<td>absolute certainty in normal direction along a single $n$-D direction</td>
</tr>
<tr>
<td><strong>Ball basis</strong></td>
<td>absolute uncertainty in both 2 directions</td>
<td>absolute uncertainty in all $n$ directions</td>
</tr>
<tr>
<td><strong>Quantization unit</strong></td>
<td>2-D grid $(i, j)$</td>
<td>$n$-D hypercube $(i_1, i_2, \ldots, i_n)$</td>
</tr>
<tr>
<td><strong>Data structure for vote collection</strong></td>
<td>2-D grid</td>
<td>$n$-D red-black tree (linearized)</td>
</tr>
<tr>
<td><strong>Neighborhood</strong></td>
<td>grid distance $= 1$</td>
<td>Hemming distance $= 1$</td>
</tr>
<tr>
<td><strong>Local extremal feature</strong></td>
<td>curve segment</td>
<td>hypersurface patch</td>
</tr>
<tr>
<td><strong>Discontinuities</strong></td>
<td>intersection of lines (junctions)</td>
<td>intersection of hypersurfaces</td>
</tr>
</tbody>
</table>

Table 8.1: Generalization of 2-D tensor voting to $n$-D
where $S = \hat{e}_1 \hat{e}_1^T$ defines a stick tensor, and $B = \hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T$ defines a ball tensor, in 2-D. These tensors define the two basis tensors for any 2-D ellipse.

Analogously, a point in the $n$-D space can either be: on hypersurface (smooth), at a discontinuity (junction) of order between 2 and $N - 2$, or is an outlier. Consider the two extremes: an $n$-D point on a hypersurface is very certain about its normal orientation, whereas a point at a junction has absolute orientation uncertainty. As in 2-D, this whole continuum can be abstracted as a second order symmetric $n$-D tensor, or equivalently a hyperellipsoid. This hyperellipsoid can be equivalently described by the corresponding eigensystem with its $n$ eigenvectors $\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_n \geq 0$ and the $n$ corresponding eigenvalues, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Rearranging the $n \times n$ eigensystem, the $n$-D ellipsoid is given by:

$$
(\lambda_1 - \lambda_2)S + \sum_{i=2}^{n-1} (\lambda_i - \lambda_{i+1}) \sum_{j=1}^{i} \hat{e}_j \hat{e}_j^T + \lambda_n B,
$$

(8.2)

where $S = \hat{e}_1 \hat{e}_1^T$ and $B = \sum_{i=1}^{n} \hat{e}_i \hat{e}_i^T$ defines an $n$-D stick and ball, respectively, among all the $n$ basis tensors: Any hyperellipsoid in $n$-D can be represented by a linear combination of these $n$ basis tensors.
8.1.1.2 *n*-D tensor decomposition

Recall that in 2-D, we can decompose a second order symmetric tensor, or an ellipse, at each input site into the corresponding eigensystem.

The eigenvectors encode orientation (un)certainties: tangent orientation is described by the *stick* tensor, indicating *certainty* along a single direction.

At *point* junctions, where more than two intersecting lines converge, a *ball* tensor is used since there is no preferred orientation.

The eigenvalues, on the other hand, effectively encode the *magnitude* of orientation (un)certainties, since they indicate the size of the corresponding ellipse.

Hence, after each tensor has been decomposed into its elements, we can rearrange the resulting eigensystem into two components:

- \((\lambda_1 - \lambda_2) \hat{e}_1 \hat{e}_1^T\) corresponds to 2-D curve-ness
- \(\lambda_2 (\hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T)\) corresponds to 2-D junction-ness

Like the 2-D version, we can also decompose a second order symmetric tensor, or a hyperellipsoid in *n*-D, at each input site into the corresponding eigensystem.

Here, the eigenvectors effectively encode orientation (un)certainties: hypersurface orientation (normal) is described by the *stick* tensor, which indicates *certainty* along a single, *n*-D direction.

*Orientation uncertainty* is indicated by the *ball* tensor, where many intersecting hypersurfaces are present and thus no single orientation is preferred. The eigenvalues encode the magnitudes of orientation (un)certainties.

Similar to the 2-D case, we can rearrange the resulting eigensystem after the *n*-D tensor decomposition into the following *n* components:
- \((\lambda_1 - \lambda_2) \hat{e}_1 \hat{e}_1^T\) corresponds to n-D hypersurface-ness

- for \(2 \leq i < n, (\lambda_i - \lambda_{i+1}) \sum_{j=1}^{i} \hat{e}_j \hat{e}_j^T\) corresponds to orientation uncertainty in \(i\) directions, with a \((n-i)\)-D feature whose direction(s) are given by \(\hat{e}_{i+1}, \ldots, \hat{e}_n\) (for example, for \(n = 3, i = 2, (\lambda_2 - \lambda_3)(\hat{e}_1 \hat{e}_1^T + \hat{e}_2 \hat{e}_2^T)\) defines a plate tensor in 3-D, which describes a 1-D feature, a curve tangent, with direction given by \(\hat{e}_3\))

- \(\lambda_n \sum_{j=1}^{n} \hat{e}_j \hat{e}_j^T\) corresponds to n-D junction-ness

### 8.1.1.3 Uniform encoding

The formalism allows the unified description of a variety of input feature tokens.

In n-D, if the input token is a point, it is encoded as a n-D ball tensor \((\lambda_1 = \lambda_2 = \cdots = \lambda_n = 1)\), since initially there is no preferred orientation.

If the input token is a curve element in n-D, it is encoded as a plate tensor \((\lambda_1 = \lambda_2 = \cdots = \lambda_{n-1} = 1, \lambda_n = 0)\), with the direction of the curve tangent given by \(\hat{e}_n\).

If the input token is a n-D hypersurface patch element, then it is encoded as a stick tensor \((\lambda_1 = 1, \lambda_2 = \lambda_3 = \cdots = \lambda_n = 0)\), with \(\hat{e}_1\) equal to the direction of the surface normal to the given hypersurface patch.

### 8.1.2 n-D tensor communication

We are now ready to describe the voting algorithm for obtaining the tensor representation. As in the 2-D and 3-D case, each input token votes, or is made to align (by translation and rotation), with predefined, discrete versions of the basis tensors (or voting fields or kernels) in a convolution-like way. As a result of voting, preferred orientation information is propagated and gathered at each input site.
8.1.2.1 Token refinement and dense extrapolation

As in the case of 2-D or 3-D, given a set of input tokens, they are encoded as tensors as described in section 8.1.1.3. These initial tensors communicate with each other for token refinement and dense extrapolation.

These two tasks can also be implemented by a voting process, which in essence involves having each input token aligned with predefined, dense, $n$-D voting kernels (or voting fields). As in 2-D, the alignment is simply a translation followed by rotation.

The dense voting kernels encode the $n$ basis tensors. The derivation of voting kernels is given in the next section.

Recall that 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 $n$ elements. By using an appropriate voting kernel, each token broadcasts the information in a neighborhood. The size of the neighborhood is given by the size of the voting kernel used. As a result, a tensor value is put at every location in the neighborhood.

As in 2-D and 3-D, these two operations are equivalent, which can be regarded as tensor convolution.

8.1.2.2 Derivation of $n$-D voting fields

As in the 3-D case, the rotated, normal version of the fundamental 2-D stick voting field, denoted by $V'_F$ (section 2.3.3), is used to derive all $n$-D voting kernels.
Derivation of \( n \)-D stick kernel

W.l.o.g., as in Chapter 2, we derive the \( n \)-D stick kernel oriented at \([1 \ 0 \ 0 \ \cdots \ 0]^T\) in world coordinates. The other orientations can be achieved by a simple rotation in the \( n \)-D space. Therefore, using the parameterization introduced in Chapter 2, the \( n \)-D stick is given by

\[
S_n = \left[ \begin{array}{c} 1 \\ 0 \\ \cdots \\ 0 \\ \alpha_n \\ \alpha_{n-1} \\ \cdots \\ \alpha_1 \end{array} \right] = \int_0^\pi V_F d\alpha_1 |_{\alpha_n=\alpha_{n-1}=\cdots=\alpha_2=0} \tag{8.3}
\]

where \( \alpha_n, \alpha_{n-1}, \cdots \alpha_1 \) denotes the angle of rotation about axis \( n, n-1, \cdots, 1 \), respectively.

Derivation of \( n \)-D ball kernel

The \( n \)-D ball kernel can be obtained by rotating the above \( n \)-D stick kernel about the remaining \( n-1 \) axes, and integrating the contributions:

\[
B_n = \left[ \begin{array}{c} 1 \\ 1 \\ \cdots \\ 1 \\ \alpha_n \\ \alpha_{n-1} \\ \cdots \\ \alpha_1 \end{array} \right] = \int_0^\pi \int_0^\pi \cdots \int_0^\pi S d\alpha_2 d\alpha_3 \cdots d\alpha_n |_{\alpha_1=0} \tag{8.4}
\]

Derivation of other \( n \)-D kernels

For the other \( i \) kernels, where \( 2 \leq i < n \), we have

\[
P_n = \left[ \begin{array}{c} i \\ 1 \\ \cdots \\ 0 \\ 0 \\ \cdots \\ 0 \\ \alpha_n \\ \alpha_{n-1} \\ \cdots \\ \alpha_1 \end{array} \right] = \int_0^\pi \int_0^\pi \cdots \int_0^\pi S d\alpha_{n-i+2} \cdots d\alpha_n |_{\alpha_1=\alpha_2=\cdots=\alpha_{n-i+1}=0} \tag{8.5}
\]
which actually describes the rotation of the \( n \)-D stick field \( S \) about the \( i \) axes, and the integration of the contributions from all angles of rotation.

### 8.1.3 Implementation of \( n \)-D tensor voting

The \( n \)-D tensor voting process aggregates tensor contribution from a neighborhood of voters by using tensor addition. \( n \)-D Tensor addition is implemented analogously as in the 2-D case, as described below.

Suppose that we have only two input tokens. Initially, before any voting occurs, each token location encodes the local associated tensor information. Denote these two tensors by \( T_{0,1} \) and \( T_{0,2} \).

**Tensor Encoding:** The input is encoded into a perfect ball, a perfect stick, or a perfect “plate” (note that in the case of \( n \)-D, we have a total of \( n - 2 \) cases of plate tensors). One example is a “hyper-curve”: a space curve occupying in the \( n \)-D space.

- **Stick.** \( \lambda_1 = 1, \lambda_2 = \lambda_3 = \cdots = \lambda_n = 0 \), with \( \hat{e}_1 \) equals to the given orientation, \( \hat{e}_2, \cdots, \hat{e}_n \) are unit vectors which are orthonormal to each other, and also to the given \( \hat{e}_1 \). Therefore, a local, cartesian coordinate system aligned with \( \hat{e}_1 \) can be used to initialize \( \hat{e}_1, \cdots, \hat{e}_n \).

- **Plates.** For \( 2 \leq i < n \), \( \lambda_1 = \lambda_2 = \cdots = \lambda_i = 1, \lambda_{i+1} = \lambda_{i+2} = \cdots = \lambda_n = 0 \). We set \( \hat{e}_{i+1}, \hat{e}_{i+2}, \cdots, \hat{e}_n \) to be the given orientations, respectively. \( \hat{e}_1, \hat{e}_2, \cdots, \hat{e}_n \) are unit vectors chosen such that they are orthonormal to each other, and to \( \hat{e}_{i+1}, \hat{e}_{i+2}, \cdots, \hat{e}_n \). Again, a local, cartesian coordinate system aligned with \( \hat{e}_{i+1}, \hat{e}_{i+2}, \cdots, \hat{e}_n \) can be used to initialize \( \hat{e}_1, \cdots, \hat{e}_n \).
(For example, for \( n = 3, i = 2 \), we use \( \lambda_1 = \lambda_2 = 1 \), and \( \lambda_3 = 0 \) to represent a curve as a 3-D plate tensor, with \( \hat{e}_1 \) being equal to the given curve tangent direction. Having determined \( \hat{e}_1 \), we can align a local 3-D cartesian coordinate system with \( \hat{e}_1 \) to determine \( \hat{e}_2 \) and \( \hat{e}_3 \).)

- **Ball.** \( \lambda_1 = \lambda_2 = \cdots = \lambda_n = 1 \), with \( \hat{e}_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T \), and \( \hat{e}_2 = \begin{bmatrix} 0 & 1 & \cdots & 0 \end{bmatrix}^T \), 
  
  \[ \cdots, \text{and } \hat{e}_n = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}^T. \]

In all cases, the input is unified into a \( n \)-D second order symmetric tensor \( T_{0,j} \), \( 1 \leq j \leq 2 \) by

\[
T_{0,j} = \begin{bmatrix} \hat{e}_1 & \hat{e}_2 & \cdots & \hat{e}_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} \hat{e}_1^T \\ \hat{e}_2^T \\ \vdots \\ \hat{e}_n^T \end{bmatrix}
\]

\[= (\lambda_1 - \lambda_2)S + \sum_{k=2}^{n-1} (\lambda_k - \lambda_{k+1}) \sum_{j=1}^{k} \hat{e}_j \hat{e}_j^T + \lambda_n B \]

\[= T_{0,j}^S + \sum_{k=2}^{n-1} T_{0,j}^P + T_{0,j}^B \]

where \( S = \hat{e}_1 \hat{e}_1^T \) and \( B = \sum_{i=1}^{n} \hat{e}_i \hat{e}_i^T \) defines an \( n \)-D **stick** and **ball**, respectively. The \( n - 2 \) plate tensors are defined respectively by \( (\lambda_k - \lambda_{k+1}) \sum_{j=1}^{k} \hat{e}_j \hat{e}_j^T, 2 \leq k < n \). These \( n \) **basis** tensors define any hyperellipsoid in \( n \)-D, by a linear combination of them.

Note that all the \( n \times n \) matrices \( T_{0,j}^S, T_{0,j}^P, T_{0,j}^B \) are symmetric, positive semi-definite, and they describe a stick, plate, and ball tensor, respectively.

198
Tensor Voting: An input site $j$ collects the tensor vote cast from the voter $i$. This vote consists of a stick component, $n-2$ plate components, and a ball component.

- **Stick vote.** Let \( v_1, v_2, \ldots, v_n \) be the stick vote collected at site $j$, which is cast by voter $i$ after aligning the $n$-D stick voting field (by translation and rotation) with the $\hat{e}_1$ component of the tensor $T_{0,i}$ at $i$ (obtained in the tensor encoding stage). Then,

\[
T_{1,j}^S = T_{0,j}^S + (\lambda_1 - \lambda_2) T
\]

(8.9)

\[
= T_{0,j}^S + (\lambda_1 - \lambda_2) T
\]

(8.10)

where $T$ is $n \times n$ a symmetric, positive and definite matrix, consisting of the second order moment collection of the vote contribution.

- **Plate votes.** Let $T_{P_k}$ be the plate vote, $2 \leq k \leq n$, collected at site $j$, which is cast by voter $i$. Then,

\[
T_{i,j}^{P_k} = T_{0,j}^{P_k} + (\lambda_k - \lambda_{k+1}) T_{P_k}
\]

(8.11)

- **Ball vote.** Let $T_{B}$ be the ball vote, collected at site $j$, which is cast by voter $i$. Then,
\[
T_{1,j}^B = T_{0,j}^B + \lambda_n T^B
\]  

(8.12)

Therefore, \( T_{1,j} = T_{1,j}^S + \sum_{k=2}^{n-1} T_{1,j}^{P_k} + T_{1,j}^B \) is obtained. Note that \( T_{1,j} \) is still symmetric and positive definite, since \( T_{1,j}^S, \sum_{k=2}^{n-1} T_{1,j}^{P_k}, \) and \( T_{1,j}^B \) are all symmetric and positive definite. Hence, \( T_{1,j} \) produced by the above is also a second order symmetric tensor.

**Tensor Decomposition:** Similar to the 2-D case, after \( T_{1,1}, T_{1,2} \) have been obtained, we decompose each of them into the corresponding eigensystem.

Note that the same tensor sum applies to the tensor voting process for extrapolating directional estimates, with the following changes, as in the case in 2-D:

- first, site \( j \) may or may not hold an input token. For non-input site \( j \), \( T_{0,j} \) is a zero matrix.

- second, we do not propagate the ball component from voting sites, but only generate stick votes.

### 8.2 \( n \)-D feature extraction

As in the 3-D case, after the dense extrapolation stage, \( n \) dense structures, which are dense vector fields, are produced. Here, we only consider the two extreme cases.
Figure 8.2: $n$-D surface extremality

- hypersurface map: each $n$-D voxel in this map consists of a 2-tuple $(s, \hat{n})$, where $s$ indicate the hypersurface-ness, or hypersurface saliency, and $\hat{n} = \hat{e}_1$ denotes the hypersurface normal direction.

- hyperjunction map: it is a dense scalar map which denote the hyperjunction-ness, or hyperjunction saliency.

Since the topology of the other $n - 2$ vector maps is very complicated, and we have not used it in application, we shall not study the feature extraction from these maps in this thesis.

As in the 2-D and 3-D counterparts, the extraction of maximal junction in the hyperjunction map is very straightforward. it is a local maxima of the scalar value $s$.

### 8.2.1 $n$-D hypersurface extremality

Here, we generalize the notion of 3-D extremal surface to $n$-D extremal hypersurface. Recall that the hypersurface map is a dense vector field $(s, \hat{n})$ which encodes hypersurface normals $\hat{n} = \hat{e}_1$ associated with saliency values $s = \lambda_1 - \lambda_2$. 
To illustrate, suppose the dense structure as obtained after the dense extrapolation stage is dense and continuous (the discrete version for implementation will be described next), i.e. \( \{(s, \hat{n})\} \) is defined for every point \( P \) in the \( n \)-D space.

Imagine that we could traverse and look at the \( s \) values along the \( n \)-D vector \( \hat{n} \) (Figure 8.2(a)). By the definition of the \( n \)-D stick kernel, after tensor voting, an extrema (or maxima) in \( s \) (Figure 8.2(b)) should be observed at the point which lies on a salient hypersurface in the \( n \)-D space.

Therefore, we define an extremal hypersurface as the locus of points for which the saliency \( s \) is locally extremal along the direction of the \( n \)-D normal, i.e.,

\[
\frac{ds}{d\hat{n}} = 0
\]  

(8.13)

Again, this is only a necessary condition for \( n \)-D extremality. The sufficient condition, which is used in the implementation, is defined in terms of zero crossings along the line defined by \( \hat{n} \) (Figure 8.2(c)).

We therefore compute the saliency gradient

\[
\bar{g} = \nabla s = \begin{bmatrix} \frac{\partial s}{\partial x_1} & \frac{\partial s}{\partial x_2} & \cdots & \frac{\partial s}{\partial x_n} \end{bmatrix}^T
\]  

(8.14)

and then project \( \bar{g} \) onto \( \hat{n} \), i.e.,

\[
q = \hat{n} \cdot \bar{g}
\]  

(8.15)
Thus, an extremal hypersurface is the locus of points with \( q = 0 \). The extraction of maximal hypersurface is analogous to maximal surface in 3-D, described in Chapter 3.

### 8.2.2 Discrete version

We have discrete \( \{ (s_{i_1,i_2,\ldots,i_n}, \hat{n}_{i_1,i_2,\ldots,i_n}) \} \) in implementation. We can define the corresponding discrete versions of \( g \) and \( q \), i.e.,

\[
g_{i_1,i_2,\ldots,i_n} = \begin{bmatrix}
  s_{i_1+1,i_2,\ldots,i_n} - s_{i_1,i_2,\ldots,i_n} \\
  s_{i_1,i_2+1,\ldots,i_n} - s_{i_1,i_2,\ldots,i_n} \\
  \vdots \\
  s_{i_1,i_2,\ldots,i_n+1} - s_{i_1,i_2,\ldots,i_n}
\end{bmatrix}
\] (8.16)

and \( q_{i_1,i_2,\ldots,i_n} = \hat{n}_{i_1,i_2,\ldots,i_n} \cdot \overrightarrow{g_{i_1,i_2,\ldots,i_n}} \). Given an input point, we compute \( q_{i_1,i_2,\ldots,i_n} \) at each vertex voxel (a total of \( 2^n \) that makes up the hypercube quantization unit containing the input site). Thus, the set of all \( \{ q_{i_1,i_2,\ldots,i_n} \} \) constitutes a scalar field. If the signs of the \( q \)’s of two adjacent vertex voxels are different, a zero crossing occurs on the corresponding hypercuboid edge (there is a total of \( n2^{n-1} \) of them).

### 8.2.3 Extraction of \((n - 1)\)-D entity

When the zero crossings have been detected, in our \( n \)-D case, we need to group these zero crossings in order to find an \((n - 1)\)-D entity, or a hypersurface patch, that intersects with the \( n \)-D voxel.
Weigle and Banks [121] describe a contour meshing procedure which generalizes well to $n$ dimensions. We provide the details in Appendix A. We give a brief summary below. (The terms in italics below will be defined in the appendix.)

- First, a splitting operation is performed, which divides a hypercube voxel into a set of $2^{n-1}n$-simplexes.

- Zero crossings are then detected on the edges of these resulting simplexes.

- A contouring algorithm is applied recursively, starting by contouring 1-simplexes (edge) that gives a single point, and then 2-simplexes (triangle) that gives a line segment, so on.

A contour, made up of $(n-1)$-simplexes, should be produced if a hypersurface (an $(n-1)$-D entity) passes through that hypercube voxel. Therefore, the detection of hypersurface is translated into the following verification: if the candidate contour as produced by the above contour meshing procedure can be triangulated into a set of $(n-1)$-simplexes and nothing more, i.e., without any simplex of lower dimensions left behind, then, we can conclude that a hypersurface is detected.

We have now explained the generalized $n$-D version of tensor voting, and extraction of 0-D entity (corresponding to hyperjunction) and $(n-1)$-D entity (corresponding to hypersurface). Now, we are ready to apply this higher dimensional version of tensor voting into real application.
8.3 Application to epipolar geometry estimation

In this section, we propose to use the $n$-D tensor voting formalism to solve the problem of epipolar geometry estimation.

Epipolar geometry is a fundamental constraint used in computer vision whenever two images of a static scene are to be registered. Two issues needed to be addressed are: the correspondence problem, and the parameter estimation problem given a set of correspondences. The main difficulty stems from unavoidable outliers inherent in the given matches. Most robust techniques require that the majority of matches to be correct, or else some form of outlier detection and removal is usually performed before the actual parameter estimation.

Both the outlier detection and parameter estimation are often formulated as a non-linear optimization and search process in the parameter space. In the case of a full perspective camera model, this search space can be prohibitively large. Consequently, gradient-based and other non-linear heuristic search techniques have been proposed. The output, however, may not be independent of initialization, and poor initialization may seriously affect convergence rate. Simplifying assumptions, such as affine camera model and local planar homography [92], can drastically reduce the search complexity, but somewhat restrict the class of transformations which can be represented.

In fact, the general epipolar constraint can be rewritten as a linear and homogeneous equation that defines an 8-D hyperplane in its parameter space [126] (see section 8.3.2). Given a candidate set of (possibly noisy) matches, if we could visualize in 8-D, the subset of inlier matches should cluster onto a hyperplane in the corresponding 8-D space. Thus, analogous to line detection in 2-D, we can pose this outlier detection problem as one of
hyperplane inference. It is of great value and theoretical interest if we can pull out this salient hyperplane feature from the given sparse and noisy 8-D cluster, without performing any iterative or multidimensional search. Simplifying assumptions become unnecessary as the search space is no longer an issue.

An intuition to a non-iterative solution for hyperplane inference is inspired by the Hough Transform [52], which employs a voting technique that produces the solution receiving maximal support. However, as the dimensionality grows, the Hough Transform is extremely inefficient, and thus is impractical in most higher dimensional detection problems. We show in the following sections how to get around the drastic increase in time and space complexities associated with higher dimensions, by using $n$-D tensor voting.

### 8.3.1 Previous work in epipolar geometry estimation

If the input set of correspondences is already very good, then, the linear method such as the Eight-Point Algorithm [70], can be used for accurate parameter estimation. This algorithm is probably the most cited method for computing the essential (resp. fundamental) matrix from two calibrated (resp. uncalibrated) camera images. With more than eight points, a least mean square minimization is used, then followed by the enforcement of the singularity constraint [38]. Its obvious advantages are speed and ease of implementation.

Hartley [38] normalizes the data before using the Eight-Point Algorithm, and shows that this normalized version performs comparably with more complicated iterative techniques. Outlier rejection must be performed before the algorithm is used.

However, in practice, the input set of matches contains a considerable amount of outliers. More complicated, iterative optimization methods are proposed, some of them are described in [126]. These non-linear robust methods use certain optimization criteria, such
as distance between points and their corresponding epipolar lines, or gradient-weighted epipolar errors. Iterative methods in general require careful (or at least sensible) initialization for early convergence to the desired optimum. In particular, the method proposed by Zhang et al. [126] use the least median of squares, data sub-sampling, and certain adapted criterion, to discard outliers by solving a non-linear minimization problem. The fundamental matrix is then estimated. Note that robust methods require that a majority of the data to be correct, whereas we can tolerate much higher outlier to inlier ratio, as shown in the result section.

Torr and Murray propose the use of RANSAC [116]: Random sampling of a minimum subset (seven pairs) for parameter estimation is performed. The solution is given by the candidate subset that maximizes the number of points and minimizes the residual. It is, however, computationally infeasible to consider all possible subsets, since which grows exponentially in number. Therefore, additional statistical measures are needed to derive the minimum number of sample subsets. Extra samples are also needed to avoid degeneracy. Good results have been obtained in practice with this method.

Chai and Ma [13] propose the use of genetic algorithm to avoid the problem of local minima, by proper definition of genetic operators. The optimization process can be sped up through incorporating the ideas of evolution that properly guides the search process.

In [92], Pritchett and Zisserman propose the use of local planar homography (plane projective transformation). Homographies are generated by Gaussian pyramid techniques. Point matches are then generated using a homography. The set of matches is then enlarged, by using RANSAC for selecting a subset of initial matches consistent with a given homography. Besides its viewpoint invariance, homography drastically reduces the search space.
However, the homography assumption, as noted, does not generally apply to the entire image (e.g. curved surfaces), although local homography applies in most situations.

### 8.3.2 Review of epipolar geometry

Here, we briefly review the epipolar geometry (more details in [22]), and formulate the estimation problem as one of 8-D hyperplane inference.

Given two images of a static scene taken from two camera systems (see Figure 8.3), let \((u_l, v_l)\) be a point in the first image. Its corresponding point \((u_r, v_r)\) is constrained to lie on the epipolar line derived from \((u_l, v_l)\). This line is the intersection of two planes: the first is defined by: the two optical centers \(C_1, C_2\) and \((u_l, v_l)\), and the other plane is the image plane of the second image. A symmetric relation applies for \((u_r, v_r)\). This is known as the epipolar constraint. The fundamental matrix \(F\) that relates any matching pair \((u_l, v_l)\) and \((u_r, v_r)\) is given by

\[
 u_1^T F u_2 = 0 \quad (8.17)
\]
where \( \mathbf{u}_1 = (u_l, v_l, 1)^T \) and \( \mathbf{u}_2 = (u_r, v_r, 1)^T \). Note that \( \mathbf{F} \) is a rank 2, \( 3 \times 3 \) homogeneous matrix.

Equation (8.17) can be re-written as a linear and homogeneous equation in terms of the 9 unknown coefficients in \( \mathbf{F} \), or

\[
\mathbf{u}^T \mathbf{f} + F_{33} = 0, \tag{8.18}
\]

where

\[
\mathbf{u} = \begin{bmatrix} u_l & u_r & v_l & v_r & u_l v_r & v_l v_r & u_l & v_l \\ \end{bmatrix}^T \tag{8.19}
\]

\[
\mathbf{f} = \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{21} & F_{22} & F_{23} & F_{31} & F_{32} & F_{33} \\ \end{bmatrix}^T \tag{8.20}
\]

which defines a hyperplane equation in the 8-D space parameterized by \( u_l, u_r, v_l, v_r \). Note that \( \mathbf{u} \) is our measurement, which is a point in the 8-D parameter space defined by a match \( \mathbf{u}_1 \leftrightarrow \mathbf{u}_2 \). Note that although we have 9 unknowns \( F_{11}, F_{12}, F_{13}, F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33} \), since \( \mathbf{F} \) is homogeneous, \( F_{33} \) can be set to unity. Therefore, the problem space is an 8-D one.

The hyperplane normal (given by \( \mathbf{f} \)) and intercept (given by \( F_{33} \)) are unknowns, and they are estimated using the 8-D version of tensor voting.

### 8.3.3 Our approach

Figure 8.4 shows our overall approach, in which the blue processes are implemented as 8-D voting processes (described in section 8.4). The input set of point matches, obtained by automatic means such as cross-correlation technique, is first converted into a sparse
8-D point set as described in the previous subsection. This point set is quantized and “tensORIZED” into a discrete tensor field, which encodes the most preferred normal direction at each point. Then, this tensor field is locally “densified,” producing local dense structures suitable for extremA detection, from which the normal and intercept of the salient hyperplane containing all good matches can be estimated. Extrema detection (the pink process) is detailed in section 8.5. The input match is then checked against the inferred hyperplane, and a set of filtered inliers is produced. This process and other issues pertinent to epipolar geometry are described in section 8.6. Finally, the normalized eight-point algorithm (least mean square minimization followed by enforcement of singularity constraint) is applied to the verified matches for fundamental matrix estimation. Complexity analysis and results on a variety of image pairs are described in sections 8.7 and 8.8.

Figure 8.4: 8-D tensor voting approach to the epipolar estimation problem
8.4 Tensorization and local densification

**Tensorization** Each 8-D point, which corresponds to a potential match, is first encoded as an 8-D ball. Then, these input balls communicate with each other, propagating ball votes in a neighborhood. After each input site has collected all the 8-D tensor votes in its neighborhood, the resulting tensor is decomposed into the corresponding 8 components. The 8-D ball component is discarded as it corresponds to junction information, which should not be propagated in the dense voting stage.

**Local densification** After the input has been tensorized, the stick component at each input tensor is made to align with the stick kernel again for obtaining a densified structure $S\text{Map} \{ (s, \hat{n}) \}$, which indicates hypersurface-ness, as defined in section 8.2, and is used for extrema detection (section 8.5) that discard outliers. This voting process, is exactly the same as tensorization, except that directed votes are also collected at non-input sites in the volume. The same $n$-D tensor voting formalism also applies here. The only implementation differences are described in the following.

To draw an analogy, consider the 2-D case. For 2-D line extraction, we can afford to densify the whole 2-D domain, i.e., votes are cast and collected everywhere (Figure 8.5(a)). Or, more efficiently, since we have obtained saliency information after tensorization (in 2-D), densification starts out from the most salient site first. Votes are then propagated subject to connectivity (since a connected line should be extracted). Hence, only a slab of votes enveloping the line are computed (Figure 8.5(b)) during the extraction process.

In this 8-D version, two implementation differences are made to avoid the drastic increase in time and space complexities owing to the higher dimensionality.
Figure 8.5: Local densification in 2-D
(a) vote casting densifies everywhere; in vote gathering, we compute (b) a slab of votes only, or even perform (c) local densification.

For time efficiency, we do not even need to compute a slab of votes since we are performing outlier rejection, for which an explicit hyperplane represented as connected hypersurface patches (analogous to connected line segments) is not necessary. We pose this outlier rejection problem as one of extrema detection in 8-D (next section), which is performed at each input site only. Therefore, local densification (Figure 8.5(c)) suffices: a small volume centered at each input site gathers all the stick votes cast within the neighborhood (defined by the size of the voting field), performs smoothing, computes the eigensystem, interprets the vote, and produces a hypersurface patch (if any) for that site, all on-the-fly. The result produced by vote gathering is the same as vote casting, since they are reciprocal to each other.

For space efficiency, both local densification and vote gathering imply that it is unnecessary to keep the sparse input in an explicit 8-D voxel array, which would be very expensive. Since the input is quantized, we use a “linearized 8-D red-black tree” (simply an ordinary red-black tree, but we concatenate the 8 integer coordinates as the search key) to store each input site. This data structure is only of size $O(m)$, where $m$ is the input size,
much smaller than a whole 8-D array. Please refer to Appendix B for a summary of red-black tree.

8.5 Extrema detection and outlier rejection

Now, for each input site, we have computed a dense and local collection \{(s, \hat{n})\} of the SMap that encodes surface normals \(\hat{n}\) associated with saliency values \(s\). We want to infer a salient hyperplane (or estimate its normal and intercept), with sub-voxel accuracy, that contains the set of inliers. It is done by extrema detection, which indicates whether a salient hyperplane passes through that site.

8.5.1 8-D extremality

We specialize the \(n\)-D extremality to 8-D extremality in this section. In the 8-D implementation, we compute the saliency gradient \(\bar{g} = \nabla s = \left[ \frac{\partial s}{\partial x_1} \frac{\partial s}{\partial x_2} \cdots \frac{\partial s}{\partial x_8} \right]^T\), and then project \(\bar{g}\) onto \(\hat{n}\), i.e., \(q = \hat{n} \cdot \bar{g}\). Thus, an extremal hypersurface is the locus of points with \(q = 0\).

We have discrete \{(s_{i_1,i_2,\ldots,i_8}, \hat{n}_{i_1,i_2,\ldots,i_8})\} in implementation. We can define the corresponding discrete versions of \(\bar{g}\) and \(q\), and \(q_{i_1,i_2,\ldots,i_8} = \hat{n}_{i_1,i_2,\ldots,i_8} \cdot \bar{g}_{i_1,i_2,\ldots,i_8}\). Given an input point, we compute \(q_{i_1,i_2,\ldots,i_8}\) at each vertex voxel (a total of \(2^8 = 256\) that makes up the hypercube quantization unit containing that input point). Thus, the set of all \(\{q_{i_1,i_2,\ldots,i_8}\}\) constitutes a scalar field. If the signs of the \(q\)'s of two adjacent vertex voxels are different, a zero crossing occurs on the corresponding hypercuboid edge (there is a total of 1024 of them).
8.5.2 Grouping of detected zero crossings

The mere existence of zero crossings does not necessarily imply the presence of a salient hyperplane, because outlier noise can produce local perturbations of the scalar field. Therefore, as in 2-D and 3-D cases, we need to group the zero crossings detected at each input site into meaningful entities:

In 2-D, the Marching Squares algorithm can be used to link (order) all zero crossings detected on a grid edge to produce a curve (1-D entity).

In 3-D, the classical Marching Cubes [69] algorithm orders the detected zero crossings on the 3-D cuboid edges (a total of 12) to form non-trivial cycles, or surface patches (2-D entities). It can be done by a lookup table of all feasible configurations (i.e., the set of all zero crossings detected on cuboid edges should exactly form a set of cycles without any “dangling” zero crossing left).

The grouping of zero crossings detected on the hypercuboid edges in a discretized 8-D space is analogous to the 3-D case: We precompute once all feasible configurations (rotationally symmetric counterparts are counted as a single configuration), and store each template configuration as an ordered edge set (hypercycle) in a lookup table. Then, the subset of hypercuboid edges with zero crossings (detected as described in section 8.5.1) are matched against the stored templates. This template matching is very efficient since a configuration can be quickly discarded by a simple check on the number of edges in the template, followed by the ordering of edges. If a match occurs, then we conclude that a salient hyperplane passes through this 8-D site, or equivalently, an inlier is found.

Given the set of inliers found, we estimate the hyperplane normal and intercept as follows: the hyperplane normal is the saliency-weighted mean of the normals inferred at all
classified inliers. The hyperplane intercept is the saliency-weighted mean of the intercepts at all inliers, obtained using the estimated hyperplane normal.

### 8.6 Other issues

1. **Data Normalization.**

   A data normalization (translation and scaling) step as described in [38] is performed. The normalization step is performed independently for the two image pairs. The set of image points on one image, obtained from all potential point matches, is first translated so that their centroid is at the origin. Then, the point coordinates are scaled so that the mean distance from the origin is $\sqrt{2}$.

2. **Scaling.**

   In the epipolar case, the eight dimensions are independent, but not normalized or orthogonal. Because the tensor voting formalism assumes isotropic dimensions, scaling of the dimensions of the 8-D cube is needed so that the bounding box of the input is a hypercube (normalized in all dimensions).

   The smallest dimension of all the 8 dimensions of the bounding 8-D box is first computed. Then, a scaling factor is computed for each of the remaining 7 dimensions. The resulting bounding box is scaled to a bounding cube whose 8 dimensions are equal to the smallest dimension of the box before scaling. Therefore, the scale factor for each dimension can be different.

   Note that scaling does not make the epipolar space orthogonal. But, since the eight dimensions are already independent, the voting kernels should still be used over a
wide applicable range. Because of the scaling, the normal inferred at each inlier needs to be rescaled back before the estimated hyperplane normal and intercept are computed.


To improve accuracy, we need to run several passes to filter the output from the previous pass. The set of classified inliers is progressively refined as more outliers are rejected in each pass. Typically, only 4 to 5 passes are needed, and the refinement stops when the output inlier set is the same as the input. Since no multidimensional search is involved, a single pass is not very time-consuming.

In summary, the flow of all the working pieces is as follows:

Repeat {
    Normalize data points
    Convert each input pair into an 8-D point (section 8.3.2)
    Scale and quantize the 8-D input point set
    Tensorize each input site (section 8.4)
    Locally densify each input site (section 8.4)
    Use 8-D extremality for outlier rejection (section 8.5)
    Rescale the input
    Estimation hyperplane normal and intercept from inliers
    Classify inliers based on inferred hyperplane
} until (input inliers = output inliers)

Apply normalized Eight-Point Algorithm

Output fundamental matrix
8.7 Space and time Complexity

Except for tensorization, local densification, and extrema detection, other processes in Figure 8.4 are clearly linear in time and space. Since we use an efficient data structure to store the input, and only local densification is performed, the space complexity of these three processes is also linear, or $O(m)$, where $m$ is the input size.

The time complexity of local densification is only a constant factor that of tensorization. Kernel alignment takes $O(1)$ time since it is only a translation followed by a rotation. Therefore, the total time complexity for tensorization and local densification is $O(mk)$, where $k$ is the size of the voting field.

For extrema detection, since there is only a finite number of detected zero crossings and configurations, the total time is $O(m)$.

Table 8.2 gives the summary on complexities. Therefore, in contrast to the Hough Transform, the time and space complexities of the 8-D voting are independent of the dimensionality. The running time for an input of 100 matches is approximately 1 min on a Pentium II (450 MHz) processor.
8.8 Results

We demonstrate the general usefulness of our method by experimenting with a variety of image pairs. In terms of the accuracy of the estimated parameters, we note that all the methods reported in [126] (M-estimators, LMedS) fail on all of our set of input matches. This is the first quantitative evaluation. We provide a second one in the form of “distance” between the “ground truth” and our estimated fundamental matrices (in pixels). This distance measure is computed by randomly generating points in the images and computing the mean distance between points and epipolar lines. We use the program \textit{Fdiff} provided by Zhang [126]. The “ground truth” is obtained by using either Zhang’s implementation (in the case of aerial image pairs, we use the image pairs, not our noisy matches, as input), or the linear method by manually picked true correspondences. Table 8.3 summarizes the results of our experiments.

8.8.1 Aerial image pairs

In \textit{Pentagon} (Figure 8.6) and \textit{Arena} (Figure 8.7) experiments, we add a large amount of outliers, by hypothesizing \textit{all} pairs within 50 pixels of the correct matches in the corresponding images. We are still able to achieve high correct percentage despite the large number of wrong matches. The resulting filtered matches are numbered in the corresponding images, and a few corresponding epipolar lines are also drawn. In \textit{Gable} (Figure 8.6), we have approximately 100\% noisy matches, i.e., one match out of two is incorrect. The lighter (or yellow) crosses in Figure 8.6 denote classified outliers. The darker (or purple) crosses, alongside with corresponding matching numbers, indicate the filtered set of good
matches. This resulting set of match is passed into the normalized Eight-Point Algorithm. A few corresponding epipolar lines are shown.

8.8.2 Image pair with widely different views

In the *House* pair (courtesy of A. Zisserman), Figure 8.9, two very disparate views of the same static scene are taken. We manually pick only 16 matches and then add 32 wrong matches. Our method rejects all outliers and produce the accurate epipolar geometry.

8.8.3 Image pairs with non-static scenes

In the presence of moving objects, image registration becomes a more challenging problem, as the matching and registration phases become interdependent. Most researchers assume a homographic model between images, and detect motion by residual, or more than two frames are used. Torr and Murray [115] use epipolar geometry to detect independent motion. Here, we propose to perform true epipolar geometry estimation for non-static scenes using tensor voting in 8-D.

Two image pairs, *Game-1* (Figure 8.10) and *Game-2* (Figure 8.11), of a non-static basketball game scene are taken. The background of both image pairs is a 3-D static indoor stadium. There is a lot of independent motion due to moving players. This produces as much as 100% additional wrong matches to the already noisy set of matches due to moving players, as given by cross correlation technique. Since our method is designed to detect a salient hyperplane (contributed by the 3-D background) from a noisy 8-D cluster, and in this case the outliers are caused by non-stationary agents and their shadows cast on the floor, we should be able to pull out this hyperplane containing the inlier matches.
Game-2 we have some additional false matches on moving players. The results of our experiments show that we can indeed discard such wrong matches, retain true matches coming from the static background, stationary players and the audience. Therefore, we believe that our approach can extract multiple motions, mainly egomotion or possibly motion of large scene objects from an image pair (the subject of future research).

8.9 Summary

In this chapter, we have generalized the tensor voting formalism into any dimensions, and described a novel approach to address the problem of outlier detection and removal, in the context of epipolar geometry estimation.

The epipolar geometry estimation problem is posed as an 8-D hyperplane inference problem. Our method is more efficient than the Hough Transform in high dimensions. The computation, and the subsequent use of hyperplane saliency and extremal property in the spatial domain (versus parameter domain for orientation) are novel and effective, and are completely different from the Hough Transform. Since the methodology avoids searching in the parameter space, it is free of problems of local optima and poor iterative convergence. Our approach is initialization free (i.e., no initial fundamental matrix guess is needed). The pinhole camera model is the only assumption we make. No other simplifying assumptions are made about the scene being analyzed. By using an adequate data structure, higher dimensionality translates into a constant factor in processing time.

The future work of this research will, in addition to the application of multimotion analysis mentioned in section 8.8.3, focus on the investigation of quantization effect, scale of
<table>
<thead>
<tr>
<th>Class</th>
<th>$\times$ Size</th>
<th>Matches</th>
<th>Good</th>
<th>Bad</th>
<th>Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentagon</td>
<td>512 × 512</td>
<td>436</td>
<td>178</td>
<td>258</td>
<td>144.94%</td>
</tr>
<tr>
<td>Arena</td>
<td>817 × 591</td>
<td>486</td>
<td>197</td>
<td>289</td>
<td>146.70%</td>
</tr>
<tr>
<td>Gable</td>
<td>534 × 556</td>
<td>368</td>
<td>174</td>
<td>194</td>
<td>111.49%</td>
</tr>
<tr>
<td>House</td>
<td>768 × 576</td>
<td>48</td>
<td>16</td>
<td>32</td>
<td>200.00%</td>
</tr>
<tr>
<td>Game-1</td>
<td>638 × 219</td>
<td>252</td>
<td>85</td>
<td>167</td>
<td>196.47%</td>
</tr>
<tr>
<td>Game-2</td>
<td>643 × 288</td>
<td>150</td>
<td>95</td>
<td>55</td>
<td>57.89%</td>
</tr>
</tbody>
</table>

Table 8.3: Summary of the results on epipolar geometry estimation

analysis, and the application of this multidimensional voting approach for general hyper-surface extraction, and possible applications to other problem domains.
Figure 8.6: Pentagon

Figure 8.7: Arena
Figure 8.8: Gable

Figure 8.9: House
Figure 8.10: *Game-1*
Figure 8.11: Game-2
Chapter 9

Conclusion and Future Work

9.1 Summary

In this dissertation, we have extended the basic tensor voting formalism in several directions, which in fact can be incorporated into the basic formalism. We proposed the use of tensor voting for estimating second order differential geometry properties, such as sign and direction of principal curvatures evaluation. We have also extended the formalism into higher dimensions, and proposed applications, and presented results.

Here, we reiterate the contributions of this thesis as follows:

1. Extension of the basic tensor voting formalism, which includes

   (a) feature extraction algorithms
   (b) feature integration

2. Tensor voting for visualization

   We apply tensor voting in a range of visualization applications, such as flow and vortex visualization.
3. Augmentation of the basic formalism

We have augmented the basic tensor voting formalism with the ability to infer second-order geometric properties, or curvature information, resulting in a faster and better overall approach. Using the augmented formalism, the feature integration process is upgraded, resulting in a more unified approach and cleaner methodology by the use of curvature information.

4. $n$-dimensional tensor voting

We have generalized the basic formalism to $n$-dimensions for any $n$, and used the 8-D version to estimate epipolar geometry.

9.2 Future Research

9.2.1 The scale issue

The extent of the voting mask is the only free parameter in our framework. The same salient groups emerge across a range of scale values, which would seem to indicate that we only need to compute saliency at a discrete set of scales, and not a continuum. The proper choice of the particular scale values, and the integration of the results from multiple scales needs to be investigated further.

9.2.2 Multiresolution

A multiresolution model captures a wide spectrum on the levels of details of an object, which can be used to reconstruct any one of the levels on demand. It is therefore useful both in prototyping when a rough shape is sufficient for most purposes, and in the final
production phase. Multiresolution is related to the issue of scale, and we are interested in incorporating multiresolution capability to the tensor voting formalism.

### 9.2.3 Dealing with Images

As can be readily observed, our systems do not take images as input, but tokens instead. We therefore assume that some local process has produced these tokens. It would be tempting to simply pipeline our 2-D salient structure inference engine with an edge detector, to provide local orientation and contrast estimation of the edges in the image. This is not straightforward, however, as junction information is crucial for grouping, but this junction information is destroyed by edge detectors, such as Canny’s [12], or Laplacian of Gaussians (LoG) [75], which model intensity distributions as two-sided. Some recent approaches, such as the one proposed by Foerstner [25], may be more appropriate. The interested reader can refer to the book chapter by Medioni [77] for a broader overview of issues in low-level feature extraction.

### 9.2.4 $n$ Dimensions

We believe that our higher dimensional approach, when applied to the epipolar geometry estimation and motion analysis, can extract multiple motions, mainly egomotion or possibly motion of large scene objects from an image pair, which is a subject of future research. We also seek to apply the $n$-D system in other application domains.
9.2.5 The tensor voting formalism

Preliminary results were reported in [78] on the use of first order tensor and tensor calculus for inference of curve end-points and region boundaries. Together with the generalization of the approach to higher dimensions, and the capabilities of inferring second-order curvature properties, a revisit of the tensor voting formalism is needed to complete the whole, underlying theory.

So far, we have been using second order symmetric tensor to encode first order differential geometry information (i.e., orientation). But we also need first order tensor to encode polarities of the associated orientation.

Second order differential geometry information (curvature information) should also be included in the whole formalism in order to make the whole theory complete.
Appendix A

N-Dimensional Marching

In this appendix, we describe an algorithm to triangulate an implicit surface (or called a variety or contour), which occupies in a \( n \)-dimensional space. The algorithm can be found in Weigle and Banks [121]. The specialization into any dimensions, for example, into 8-D, can be readily done.

It consists of three parts:

- Cell splitting
- Simplex contouring
- Contour triangulation

A.1 Cell splitting

Recall that a square is a 2-cell, a cube is a 3-cell, a “hypercube” is a 4-cell, so on (Figure A.1(a)). Also recall that a 1-simplex is an edge, 2-simplex is a triangle, 3-simplex is a tetrahedron, so on (Figure A.1(b)).
We want to split an \( n \)-cell into simplexes in order to avoid ambiguities [83] that may arise when a contour crosses a cell. In other words, we want to triangulate the domain into a set of simplexes.

### A.1.1 Splitting a 2-cell

Splitting a 2-cell into four 2-simplexes (squares into triangles) is simple (Figure A.2): Each pair of adjacent vertices in the 2-cell can be combined with the mid-point to form a 2-simplex.

\[
\text{SplitC2 (square, simplex)} \\
simplex.p[2] \leftarrow \text{midpoint (square)} \\
\text{foreach edge in square do} \\
\quad \text{simplex.p[0..1] } \leftarrow \text{edge.p[0..1]} 
\]
A.1.2 Splitting a 3-cell

Next consider splitting a 3-cell. Each (square) face in a 3-cell can be combined with the midpoint of the cell to form a pyramid (Figure A.3(a)). The square base of the pyramid can be further subdivided so that four tetrahedrons, or 3-simplexes, are formed (Figure A.3(b)).

Note that the last step is exactly $\text{SplitC2}$:

$\text{SplitC3}$ (cube, simplex)

$\text{simplex.p}[3] \leftarrow \text{midpoint} \text{ (cube)}$

$\text{foreach square \ in \ cube \ do}$

$\text{SplitC2} \text{ (square, simplex)}$

A.1.3 Splitting an $n$-cell

This process generalizes into any $n$-cell for any $n$.

$\text{Split}$ (cell, simplex, $n$)

$\text{if } (n > 1) \text{ then}$

$\text{simplex.p}[n] \leftarrow \text{midpoint} \text{ (cell)}$

$\text{foreach subcell \ in \ cell \ do}$

$\text{Split} \text{ (subcell, simplex, } n - 1 \text{)}$

$\text{else}$

$\text{simplex.p}[0..1] \leftarrow \text{edge.p}[0..1]$
Note that this midpoint-splitting scheme generates $2^{n-1} n!$ simplexes from an $n$-cell. There are other more complicated techniques [1] that produce only $n!$ simplexes. But the scheme presented here is easy to implement, and the it is only computed once and stored as lookup-table, as what we do in the 8-D case for epipolar geometry estimation.

### A.2 Simplex contouring

When we have triangulated the domain by the above midpoint-splitting scheme, we need to obtain the scalar values at each vertices, including the new vertices generated by the introduction of mid-points during the splitting scheme. Then, we need to detect a zero crossing on an edge. By the intermediate value theorem, a zero crossing exists if there is a sign change at two adjacent vertices. Note that the reverse of the theorem is not true: the resolution should be sufficiently fine in order to resolve for a zero crossing.

#### A.2.1 Contouring a 1-simplex

A contour on a 1-simplex is a point. Therefore, an immediate application of the intermediate value theorem suffices. The “plug-in” procedure below, `contourPoint`, is actually a linear approximation of the location of the zero crossing on the edge. Therefore, it is of sub-cell (sub-voxel) accuracy.

```plaintext
contourS1 (edge)
    if zero-crossing (edge) then
        return contourPoint(edge)
```
A.2.2 Contouring a 2-simplex

A contour inside a 2-simplex (triangle) is approximated by a segment. Each endpoint of the segment is a point that lies on an edge of a triangle. Therefore, we loop over the edges (sub-complexes) of the triangle, and then connect them to form a contour-segment.

\( \text{contourS}_2 \) (triangle)
\[
\text{foreach edge in triangle do}
\]
\[
\text{point } \leftarrow \text{contourS}_1 \text{(edge)}
\]
\[
\text{contour } \leftarrow \text{contour } \cup \text{ point}
\]

Note that the resulting contour is a line segment.

A.2.3 Contouring a 3-simplex

To construct the contour-polygon in a tetrahedron, we can in fact enumerate all the possible configurations of the zero crossing that produce a physical surface patch. This is what we did in the 3-D case for surface extraction. Or, we can use a procedural approach as we are describing, which generalizes well into higher dimensions.

\( \text{contourS}_3 \) (tetrahedron)
\[
\text{foreach triangle in tetrahedron do}
\]
\[
\text{segment } \leftarrow \text{contourS}_2 \text{(triangle)}
\]
\[
\text{contour } \leftarrow \text{contour } \cup \text{ segment}
\]

Note that the resulting contour is a 3-edged or 4-edged “polygon”, or more correctly, \( \text{polytope} \) since the corresponding 4 points may not be coplanar. A polytope is defined as the convex hull of a finite domain.

A.2.4 Contouring an \( n \)-simplex

The above procedure can be subsumed by the following \( \text{contour} \) procedure, as follows:
**A.3 Contour triangulation**

If subsequently the contour polytope needs to be displayed, we can triangulate it into a set of simplexes. We can adapt the *Split* procedure, by looping over sub-polytopes along the boundary of the polytopoes:

```plaintext
triangulate (polytope, simplex, n)
  if (n > 1) then
    simplex[n] ← midpoint (polytope)
    foreach subpoly in polytope do
      triangulate (subpoly, simplex, n − 1)
  else
    simplex[0..1] ← polytope[0..1]
```

Note that the *foreach* in the above pseudocode in the case of $n$-simplex involves its decomposition into $(n − 1)$-simplexes, which is only a matter of combinatorics: since there are $(n + 1)$ vertices in an $n$-simplex, and there are $C_n^{n+1} = n + 1$ possible combinations of $(n + 1)$ vertices, the total number of $(n − 1)$-simplexes is $(n + 1)$. 
Appendix B

Red-Black Tree

A detailed description on red-black tree can be found in a standard algorithm text [15]. A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either red or black. By constraining the way the nodes can be colored on any path from the root to a leaf, red-black trees ensure that no path is more than twice as long as any other, and so therefore the tree is approximately balanced.

A binary search tree is a red-black tree if it satisfies the following red-black properties:

1. Every node is either red or black.

2. Every leaf is black.

3. If a node is red, then both its children are black.

4. Every simple path from a node to a descendent leaf contains the same number of black nodes.

A red-black tree with $n$ internal nodes has height at most $2 \log(n + 1)$, making them good search trees. The tree-insert and tree-delete operations with $n$ keys take $O(\log n)$ time. Because insertion and deletion can modify the tree, the resulting tree may violate
the red-black properties. Two operations, \textit{left-rotate} and \textit{right-rotate} are defined to restore
the red-black tree properties after insertion or deletion. The tree rotation operations take
$O(1)$-time. For details, please refer to [15].
Reference List


[21] H. Edelsbrunner and E. Mücke, “Three-dimensional alpha shapes,” ACM Transac-


242


