DIGITAL VIDEO MOVING OBJECT SEGMENTATION USING TENSOR VOTING: A NON-CAUSAL, ACCURATE APPROACH

By
Ian Guest

SUBMITTED IN FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
UNIVERSITY OF CAPE TOWN
CAPE TOWN
AUGUST 2009

Abstract

Motion based video segmentation is important in many video processing applications such as MPEG4. This thesis presents an exhaustive, non-causal method to estimate boundaries between moving objects in a video clip. It make use of tensor voting principles. The tensor voting is adapted to allow image structure to manifest in the tangential plane of the saliency map. The technique allows direct estimation of motion vectors from second-order tensor analysis. The tensors make maximal and direct use of the available information by encoding it into the dimensionality of the tensor. The tensor voting methodology introduces a non-symmetrical voting kernel to allow a measure of voting skewness to be inferred. Skewness is found in the third-order tensor in the direction of the tangential first eigenvector. This new concept is introduced as the *Tensor Skewness Map* or *TS map*. The TS map gives further information about whether an object is occluding or disoccluding another object. The information can be used to infer the layering order of the moving objects in the video clip. Matched filtering and detection are applied to reduce the TS map into occluding and disoccluding detections. The technique is computationally exhaustive, but may find use in off-line video object segmentation processes. The use of commercial-off-the-shelf Graphic Processor Units is demonstrated to scale well to the tensor voting framework, providing the computational speed improvement required to make the framework realisable on a larger scale and to handle tensor dimensionalities higher than before.

Declaration

This thesis is being presented for the degree of Doctor of Philosophy in the Department of Electrical Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at any university. I confirm that it is my original work. Portions of the work have been published in condensed form in several conference papers [18, 19]. I confirm that I am the primary researcher in all instances where work described in this thesis was published under joint authorship.

Ian Guest
August 2009

Acknowledgments

I would like to acknowledge persons that have played a role in the formulation of this thesis over the years. I would like to thank my supervisor Fred Nicolls and co-supervisor Gerhard de Jager who provided valuable input into the structure and direction of the thesis. They also helped in seeking funding, as well as facilitated my attendance of the local conferences. Also to all the members of the Digital Image Processing group at UCT whom I had the pleasure of interacting with over the years. In the industry, Thomas Landgrebe and Pavel Paclik helped keep the enthusiasm up.

I would like to thank my family Lysette, Lucas, Sawn and John for their patience and support over a very long five years of study. I would also like to thank my mother Mavis Guest for her encouragement over the years.

I would like to acknowledge the financial assistance of:

- The National Research Foundation towards this research. Opinions expressed in this work, or conclusions arrived at, are those of the author and are not necessarily to be attributed to the National Research Foundation.
- De Beers Consolidated Mines Limited.

Table of Contents

Chapter 1

Introduction

This chapter gives a brief basic background in image and video segmentation and explains the motivation to improve and augment the methodologies in the field of moving object segmentation by using tensor voting strategies. We give an overview of the objectives of the research and indicate the contribution made by this thesis in the field of moving object segmentation using tensor voting. The various chapters are summarised to allow selective reading of the thesis and to give the mathematical conventions and notations used throughout the thesis.

1.1 Overview and motivation

In a video sequence, a human observer can identify a background or backdrop against which the scene plays out. The human observer is also able to identify several objects in the video scene which move in relation to the background and each other. They may at times move over each other obscuring one of the objects giving a sense of depth. Motion itself in the video scene attracts attention as the observer would most likely focus on that which is moving compared to that which does not.

If the sequence was reduced to a single frame or still shot, the observer would be able to differentiate objects based on their semantic inference such as knowing what a human looks like and identifying a human in the frame. Computer vision usually does not have this semantic inference — or at least it does not form part of low level algorithms — but it is more part of a high level vision problem [70] or content-based analysis [20]. On a single frame, low level vision usually makes use of high contrast edges to segment images. Examples of such algorithms are the Canny edge detector [4] and the watershed algorithm [69]. In highly complex or textured images, these segmentations may bear little resemblance to actual objects. Another method is to use a form of clustering based on the image pixel colours to determine the objects [39, 7].

By introducing the other frames of a video sequence, the relative motion of points or areas in the image can provide good cues to delineate objects. This is referred to as moving object segmentation and is the focus of this thesis. By being able to separate moving objects in the video sequence, these objects can be separated allowing the background to be represented as a single image reconstructed over several frames. This is called a mosaic and has been researched by many authors [5, 29, 52, 54] as it can be used in video compression by sending the background once, and then superimposing the moving objects on the background. The remaining moving objects can also benefit from this process as the rate of change within them is small compared to the object movement rate allowing better compression. The MPEG4 standard [25, 37] makes use of the moving objects as AV object data with object motion.

Most moving object approaches make use of the previous frame to determine motion. This thesis looks at a non-causal approach where frames from both the past and future are used to estimate moving objects. This type of encoding is suitable for off-line processing of video clips in preparation for compression. The non-causality usually implies that time is not a factor in estimating the moving object boundaries. This is also motivated by the extremely rapid growth in computing capability. The algorithms explored are massively parallel and computationally intensive. This may present a computational problem now, but not in the future.

A typical application has been researched by Guest [19] where MPEG4 compression methodologies can be optimised to give better compression for a similar image quality if object boundaries are

known. This approach allows an object to "bleed" over a motion boundary. Motion boundaries are typically very abrupt changes in colour/intensity causing high frequency components in the Direct Cosine Transform (DCT). By allowing a smooth or gradual intensity variation on the edge, these components can be made a lot smaller which allows better compression. When decoding the video, the object boundaries are used to delineate the object once reconstructed, which reinstates the crisp edge.

There are many motion segmentation techniques using edge techniques, region techniques, combined edge and region techniques coupled with motion vector estimation. The techniques are either point based where each pixels motion is estimated (usually using image gradients) or region based where a *Region of Support* is used to determine the motion [55]. From these two techniques, motion can be estimated. The point-based method has the advantage of being able to operate close to motion boundaries without being affected, but are highly susceptible to noise in the sequence. The region-based method is far more robust against noise, but lacks the fine resolution required on moving object boundaries.

An effort to try adjust the region-based method to have a more relevant Region of Support instead of typically a block of pixels, led to tensor voting techniques pioneered by Medioni and several other researchers [43, 21, 30, 34, 33, 36, 45, 46, 49, 48, 50, 58, 61, 60, 62, 59, 63, 64, 65, 67, 66]. The tensor voting techniques provide the baseline from which this thesis develops extensions. The tensor voting approach is relevant in that it combines *association* and *feature extraction* in a single step. It also has very few parameters that require adjustment making it a unified robust approach.

This thesis aims at presenting a theory which may yield a different approach to motion segmentation, but does not attempt to finalise this approach. Several examples are explored to indicate the potential of the techniques developed in the thesis.

1.2 Research objectives

The thesis objective is to extend the tensor voting framework as applied to motion segmentation. To do this new algorithms and encoding structures are developed and higher-order tensor analysis is explored to reveal additional geometric features in a video sequence that have bearing on motion segmentation.

The objective of not being constrained by the number of adjacent frames (non-causal) in a video sequence allows the algorithms to be applied to video sequences in an off-line application such as video compression of pre-recorded material. This currently does occur in the industry where movies are encoded as efficiently as possible to reduce digital bandwidth on digital TV systems and to cram as many channels as possible into the available bandwidth. The objective of accurate moving object segmentation fits in with current compression standards such as MPEG4 [25], but is often not applied due to the difficulty in doing the segmentation.

With non-causality and accuracy comes the burden of computational load. Algorithms need to look at more data and information in order to make decisions on segmentation. This thesis allows more data to enter the algorithms and the objective is to be able to make more-optimal selections of which data contains information. Of course — this happens for each pixel making it a vast computational and memory intensive problem.

Within the problem of computation, the thesis research looks at resolving and adapting algorithms developed on a highly parallel computation engine in the form of a Graphical Processing Unit (GPU) that is generally available. The research also looks into the segmentation of the algorithm on these units to allow expandability into the future where GPUs will and already are (i.e. the Nvidia Tesla architecture and computers) becoming super-computers of the future.

1.3 Contributions

Although the thesis builds on existing tensor frameworks in the motion segmentation application, several concepts thought to be novel are introduced.

1. **Tangential Voting**. In order to fit the idea of motion traces through a *spatio-temporal volume*, the tensor voting framework is adapted to allow tangential voting. The normal tensor voting framework makes use of surface geometric features with normal voting. This concept is generalised into the N dimensional tensor voting framework.

2. **Direct Data Encoding**. The pixel and surrounding pixel colour information and relations are directly encoded into the tensor resulting in high dimensionality. The reason for approaching the problem with direct encoding is to allow the tensor voting framework to be driven directly from the data as well as to make data association more selective.

3. **Non-Symmetrical Kernel**. The normal tensor voting framework voting kernel is adapted to be asymmetrical. This allows higher-order tensor analysis to reveal other useful geometric features to be extracted without disturbing the second order geometrical features in use.

4. **Tensor Skewness**. The third-order tensor geometrical property investigated reveals *occluding* and *disoccluding* properties. These properties allow motion boundaries to be found as well as giving layering information in the video scene.

In terms of implementation, the algorithms developed need an implementation framework suitable for high throughput computation. Due to the highly parallel and uncorrelated nature of the tensor voting framework, a novel Graphic Processing Unit (GPU) implementation for high-dimensional tensor voting is developed using a commercially available and affordable solution.

1.4 Thesis organisation

The remainder of **Chapter 1** provides the mathematical notations used in the thesis including the concept of dimensionality, the symbols used and common operations and conventions. It goes on to define terms concerned with images and video sequences with the concept of colour encoding. Tensors are introduced and defined in the context of this thesis. The concept of tensors is expanded to second- and third-order tensors, and the geometric features of second-order tensors are described. The thesis relies heavily on random numbers, which need to be on unit spheres. The methodology of obtaining numbers lying on the unit sphere is described and discussed. Lastly, the counter-intuitive property of surface area on unit spheres is covered.

Chapter 2 discusses current tensor voting methods in the literature with several examples and results cited. The tensor voting framework as it is used in the literature is described and defined. The generalised tensor voting framework as applied to motion segmentation as used by Nicolescu [49] is presented and commented on.

Chapter 3 looks at motion traces in a spatio-temporal volume. This leads to an analysis of where the information to do with motion segmentation lies in the spatio-temporal volume including both pixel information and pixel inter-relation. In the 3D case, a solution to tangential voting is looked at, but due to the dimensional limitations a more generalised solution is developed. The concept of a skew kernel and a skewness measure is introduced to allow third order tensor features to emerge. The data encoding into the tensors is formulated and explained. To easily evaluate the performance, a single dimensional example is presented with simulation results. The concept of skewness is expanded to the two-dimensional image plane in a spatio-temporal volume, and Tensor Skewness maps (*TS maps*) are explored.

Chapter 4 looks into the implementation of the algorithms on a GPU. All the architectural issues are discussed, and an implementation framework given. The algorithms are presented as they are implemented, including implementation specific methodologies to allow higher order dimensions to be accommodated. The GPU is also benchmarked against the Personal Computer (PC) Central Processor Unit (CPU) to determine its performance enhancement.

Chapter 5 looks at some real-world examples. Motion estimation and motion segmentation problems using standardised sequences are investigated and discussed. Ground truth is also presented and the results compared and discussed.

Finally, **Chapter 6** concludes the thesis giving a summary of contributions, and reflects on results and potential future work in the algorithms developed.

1.5 Mathematical notations

1.5.1 Defining the dimensionality

In the following work a lot of traversing between the *scalar* representations and *multi-dimensional* representations is made. It is fitting to clarify mathematical notations here as it will ease the readability of the manuscript.

We define scalars as non-bold italic letters such as $a, b, \ldots$. This denotes a scalar a that is a real number, $a \in \mathbb{R}$.

We define vectors as bold letters such as $\mathbf{a}, \mathbf{b}, \ldots$. The dimensionality or length of the vectors is defined by p where $p \in \mathbb{N}^+$ where $\mathbb{N}^+$ denotes positive natural numbers. A vector $\mathbf{a}$ is related to real number space of dimension n by $\mathbf{a} \in \mathbb{R}^n$. A vector $\mathbf{a}$ where $p = 1$ is a scalar a. For $p = 2$, we have a 2D vector defined as $\mathbf{a} = (a_1, a_2)$. Higher values of p result in higher dimensional vectors following the same pattern.

We define multi-valued vectors and matrices as upper case bold letters such as $\mathbf{A}$,$\mathbf{B}$,.... Each element of the vector or matrix has a dimensionality n where $n \in \mathbb{N}^+$. If $n = 1$, then the multi-valued vector reduces to a normal vector. A multi-valued vector or matrix $\mathbf{A}$ is related to real number space of dimension $n \times p$ by $\mathbf{A} \in \mathbb{R}^{n \times p}$.

1.5.2 Symbols

Scalars, vectors, matrices and high-order tensors

a , b , ...	Representation of scalars or random variables of dimension $p = 0$.
$\mathbf{a}$, $\mathbf{b}$, ...	Representation of vectors or random variable vectors of dimension $p = 1$. *unitary* (unit length) vectors are represented as $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, ...
$\mathbf{A}$, $\mathbf{B}$, ...	Representation of matrices of dimension $p = 2$.
$\mathcal{A}$, $\mathcal{B}$, ...	Representation of tensors of dimension $p \geq 3$.

Region of support

A Region $\mathcal{R}$ is defined as a space in the metric chosen that is used to support or nullify an hypothesis about a point. This is normally done in Euclidean space around a particular point.

Metrics and spaces

Ω	Generic name of the definition domain of images, curves, surfaces, lines etc.
$\mathbb{N}^+$	Dimension of the underlying space. Generally $p \in \mathbb{N}^+$ (natural positive integers).
$\mathbb{R}^p$	Closed spatial domain of dimension p. In our case $\Omega \subset \mathbb{R}^p$.

Inner and outer products

A dot product of two vectors $\mathbf{u}$ and $\mathbf{v}$ is given as $\mathbf{u} \bullet \mathbf{v}$ and results in a scalar. The outer product or tensor product of two vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$ is given as $\mathbf{u} \otimes \mathbf{v}$ and results in a matrix of size $m \times n$. An inner product is also expressed as $\langle \mathbf{u}, \mathbf{v} \rangle$.

Normalising

The Euclidean distance between two vector points $\mathbf{a}$ and $\mathbf{b}$ is defined as $|\mathbf{a} - \mathbf{b}|$. The length of a vector $\mathbf{a}$ is similarly defined as $|\mathbf{a}|$. To indicate that a vector $\mathbf{a}$ has unit length, the unit vector is defined as $\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|}$. In the text, the unit vector may be used without indicating the normalisation process — in this case it must be assumed that the vector has been normalised. The use of the norm of $\mathbf{a}$ may also be used as an extension to the Euclidean distance and is denoted as $\|\mathbf{a}\|$.

Dropping of subscripts and arguments

In order to make the formulations more readable and compact, subscripts and arguments may be dropped as formulations are developed. An example of this would be N_{ij} becoming N and $\mathbf{v}(s, \kappa)$ becoming $\mathbf{v}$. This is only done in cases where no confusion may exist on the formulation interpretation.

(a) Stacked images. (b) Spatio-temporal volume.

Figure 1.1: A Spatio-temporal volume represented of part of the *Mobile Calender* sequence.

1.5.3 Defining image and video representations

The thesis deals with *images* and *spatio-temporal volumes*. Spatio-temporal volumes are representations of images that include the temporal (time aspect) by stacking the images of a video sequence along the time axis as shown in Figure 1.1.

We define Ω as the domain for all the images, volumes and structures that we work with based on Tschumperle [68]. We let $\Omega \subset \mathbb{R}^p$ be a closed spatial domain of dimension p. It is important to note that we define it as a spatial domain which would be characteristically defined as the row (y) and column (x) position in an image as well as the z or time (t) dimension of a spatio-temporal volume. The dimensionality p is defined as a positive natural number, $p \in \mathbb{N}^+$, and in the case of x, y, z, p has a value of 3.

For a scalar image or spatio-temporal volume, we define the intensity image/volume I as:

$$I : \left| \begin{array}{l} \Omega \in \mathbb{R}^p \rightarrow \mathbb{R} \\ \mathbf{x} \rightarrow I(\mathbf{x}) \end{array} \right.$$

where the domain of I is $\mathbb{R}^p$ and the co-domain (what I maps into) is into the real number $\mathbb{R}$ domain. Furthermore, each point in Ω has an assigned intensity I based on its location $\mathbf{x}$. Several cases for different p emerge:

$\mathbf{x}$	$=$	x	where $p = 1$ (one-dimensional image)
$\mathbf{x}$	$=$	(x, y)	where $p = 2$ (normal two-dimensional image or slice of spatio-temporal volume)
$\mathbf{x}$	$=$	(x, y, z)	where $p = 3$. (spatio-temporal volume or set of video frames)
$\mathbf{x}$	$=$	$(x, y, z, \ldots)$	where $p > 3$. (multi-dimensional spatial vector or tensor).

This is a continuous representation. In tensor analysis that follows, this is relevant due to estimation of points that may not be discrete. This representation has spatial dimensionality, and as such only represents grey scale images and spatio-temporal volumes. The last case where $p > 3$ may be used where further dimensions are added as tensor representations.

For a vector-valued image or spatio-temporal volume, we define the intensity image/volume $\mathbf{I}$ as:

$$\mathbf{I} : \left| \begin{array}{l} \Omega \in \mathbb{R}^p \rightarrow \mathbb{R}^n \\ \mathbf{x} \rightarrow \mathbf{I}(\mathbf{x}). \end{array} \right.$$

Two common cases for different n emerge. When $n = 1$, the representation becomes a scalar representation. For colour images and volumes $n = 3$ which represents the (R, G, B) of a colour image. Other colour spaces can also be used and will be described in the text.

In many cases, colour images may be channelised with each colour is represented separately. In this case $I_i : \Omega \rightarrow \mathbb{R}$ which is the ith component of the vector image where $1 \leq i \leq n$. The vector image $\mathbf{I}$ can be represented in terms of I_i as:

$$\forall \mathbf{x} \in \Omega, \mathbf{I}(\mathbf{x}) = (I_1(\mathbf{x}), I_2(\mathbf{x}), \ldots I_n(\mathbf{x})).$$

For the sake of brevity, the functional parts may be dropped if the equation remains unambiguous, such as using $\mathbf{I}$ instead of $\mathbf{I}(\mathbf{x})$. The images used all form part of a regular grid, so individual pixels for $p = 1$ are referred to in discrete form using $\mathbf{I}_i$, for $p = 2$ as $\mathbf{I}_{i,j}$ and for $p = 3$ as $\mathbf{I}_{i,j,k}$, where i refers to the column, j refers to the row, and k refers to the plane. The correspondence to the continuous form already given would be x,y and z respectively.

When n moving objects $\Theta_i, \quad i = (1, 2, 3, \ldots, n)$ are segmented, they form a complete segmentation:

$$\begin{array}{ll} (\Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_n) & \equiv \Omega \\ \Theta_i \cap \Theta_j & = \emptyset \, \forall \, i \neq j \end{array}$$

For the data-driven purposes of the thesis, when an object is occluded it is not deemed to be part of the analysis anymore.

1.5.4 Defining tensors

Tensors are extensions to *scalars*, *vectors*, *matrices*, and *multidimensional matrices*. Tensors represent a linear quantity in the form of a n-dimensional array relative to a basis in which it is defined. It becomes a tensor when it is able to define itself without having to refer to this basis. This normally implies that a tensor is a set of vectors of the dimensionality of the basis it occupies, and is representable as a linear combination of these. Another way of expressing this is that if a transform is applied to the tensor, the vectors will transform to a different basis without collapsing any of these vectors. The order of a tensor gives some information of its form. A tensor of order 1 can be represented as a vector $\mathbf{x}$ with $p = n$ representing the length of this vector. Tensors of higher order can be represented as linear combinations of first-order tensors $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

$$\mathbf{T}_{ijk\ldots} = a_i b_j c_k \ldots \tag{1.5.1}$$

First-order tensors

First-order tensors are generally represented as a vector $\mathbf{x} \in \mathbb{R}^n$ where the length of the vector represents the length of the order 1 tensor with $p = n$. First-order tensors allow us to infer positional information by being able to describe a point in a specific metric. The usefulness of this is analogous in statistics to an averaging operation (first moment) and in this thesis has limited use, other than to construct higher-order tensors.

Second-order tensors

Second-order tensors have been extensively used in image and video moving object segmentation [28, 43, 45, 49]. Second-order tensors allow geometric features other than position to be described, such as curvature and direction. The equivalent operation in statistics is the covariance or second-order moment. Following Equation 1.5.1, they can be described by matrices with two indices and are *symmetric* and *positive semidefinite*. A two-dimensional matrix is defined as:

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{12} & \ldots & t_{1n} \\ t_{21} & t_{22} & \ldots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & \ldots & t_{nn} \end{pmatrix}.$$

The tensors that are used in tensor voting are made up by the *outer product* of a first-order tensor (vector) $\mathbf{a} \in \mathbb{R}^n$ with itself. This is equivalent to $\mathbf{T}_{ij} = a_i b_j$ which is the second-order representation

of Equation 1.5.1:

$$\mathbf{T} = \mathbf{a} \otimes \mathbf{a} = \mathbf{a}\mathbf{a}^\top = \begin{pmatrix} a_1^2 & a_1 a_2 & \dots & a_1 a_n \\ a_2 a_1 & a_2^2 & \dots & a_2 a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n a_1 & a_n a_2 & \dots & a_n^2 \end{pmatrix}.$$

Considering that the elements of $\mathbf{a}$ are real, the tensor $\mathbf{T}$ is considered *real*. The commutative property holds in that $a_i a_j = a_j a_i$ and as such the tensor $\mathbf{T}$ is *symmetric*:

$$\mathbf{T} \quad \text{is } \textit{symmetric} \iff \forall i, j \in [1, n], \quad t_{ij} = t_{ji}.$$

A positive semidefinite (or *nonnegative semidefinite*) matrix has the following characteristic:

$$\mathbf{T} \quad \text{is } \textit{positive semidefinite} \iff \forall \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x}^\top \mathbf{T} \mathbf{x} \geq 0.$$

One of the only conditions that a symmetric matrix be positive semidefinite is that it is a *Gram* matrix. A matrix $\mathbf{A}\mathbf{A}^\top$ is a Gram matrix of the rows of $\mathbf{A}$. In our case the matrix $\mathbf{A}$ is the vector $\mathbf{a}$, which is a matrix where one dimension is a singleton. This confirms that the tensor $\mathbf{T}$ is indeed positive semidefinite. Another important factor used in the construction of tensors in this thesis is that if $\mathbf{A}$ and $\mathbf{B}$ are positive semidefinite, then $\mathbf{A} + \mathbf{B}$ is also positive semidefinite.

An important aspect of the positive semidefinite property is held in the *spectral decomposition* into eigenvectors $\hat{\mathbf{e}}_k \in \mathbb{R}^n$ and its eigenvalues λ_k. A set of n eigenvectors $\hat{\mathbf{e}}_i$ and right-hand eigenvalues λ_i exist that satisfy the equation:

$$\mathbf{T}\mathbf{v} = \lambda \mathbf{v}.$$

The tensor $\mathbf{T}$ is positive semidefinite and this implies:

$$\mathbf{T} \quad \text{is positive semidefinite} \iff \forall k \in [1, n], \quad \lambda_k \geq 0.$$

Due to $\mathbf{T}$ being *real* and *symmetric*, the eigenvectors form an *orthonormal vector basis* in $\mathbb{R}^n$:

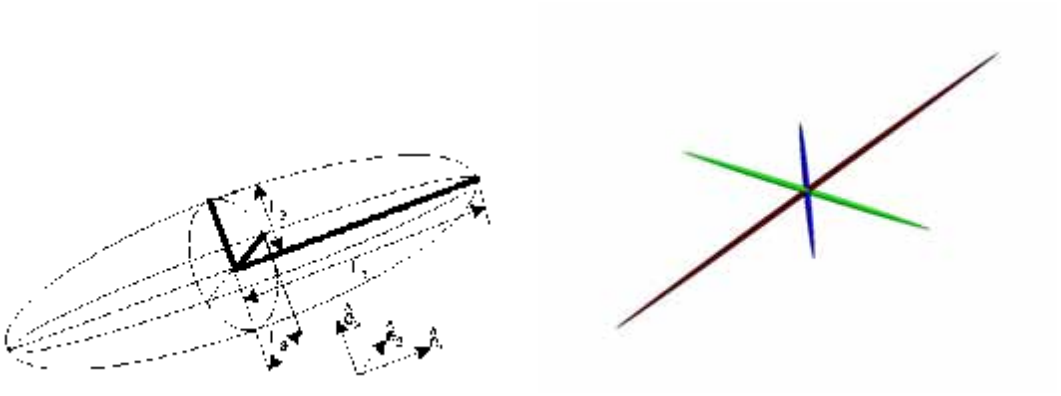
$$\mathbf{T} \quad \text{is } \textit{real} \text{ and } \textit{symmetric} \iff \forall k, l \in [1, n], \quad \hat{\mathbf{e}}_k \cdot \hat{\mathbf{e}}_l = \delta_{kl} = \begin{cases} 1 & (\text{if } k = l) \\ 0 & (\text{if } k \neq l) \end{cases}$$

The tensor $\mathbf{T}$ can now be written as:

$$\mathbf{T} = \mathbf{R}\boldsymbol{\Gamma}\mathbf{R}^\top$$

where $\boldsymbol{\Gamma}$ is a diagonal matrix with the eigenvalues down the diagonal:

$$\boldsymbol{\Gamma} = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n \end{pmatrix}.$$



(a) Tensor $\mathbf{T}$ represented as an ellipsoid.

(b) The tensor $\mathbf{T}$ represented as several thin ellipsoids.

Figure 1.2: Tensor $\mathbf{T}$ representations. The red line indicates the vector $\lambda_1\hat{\mathbf{e}}_1$, the green line indicates the vector $\lambda_2\hat{\mathbf{e}}_2$ and the blue line represents $\lambda_3\hat{\mathbf{e}}_3$.

and $\mathbf{R}$ is a *rotation matrix* or *reflection matrix* which is characterised by $det(\mathbf{R}) = \pm 1$ and the preservation of distances (isometry) in the Euclidean space. In this case it consists of column vectors $\tilde{\mathbf{u}}_k$ which are the eigenvectors:

$$\mathbf{R} = (\tilde{\mathbf{u}}_1|\tilde{\mathbf{u}}_2|\dots|\tilde{\mathbf{u}}_n) \qquad \text{where} \qquad \forall k = 1\dots n, \quad \tilde{\mathbf{u}}_k = \hat{\mathbf{e}}_k.$$

The rotation or reflection has orientation given by the eigenvectors. If it is a reflection ($det(\mathbf{R}) = -1$) the directionality of the orientation could be reflected. The fact that the reflection does not have directionality on the eigenvectors is important to note as further analysis looks at orientation, but not necessarily direction. For this reason we refer to $\mathbf{R}$ as the *orientation*, while the diagonal of $\mathbf{\Gamma}$ indicates the *strength* of the orientation. The eigenvalues are also ordered and real in that $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$. This ordering ensures that the first eigenvector $\hat{\mathbf{e}}_1$ has the largest associated eigenvalue λ_1.

As a way of visualising the tensor $\mathbf{T}$, we can use an ellipsoid of the dimensionality of the tensor. The principal axis will than be defined by the eigenvalues and eigenvectors. The 3D case is shown in Figure 1.2(a). Due to the orthogonality of the eigenvectors, the tensor $\mathbf{T}$ can be represented as the weighted sum of unit orthogonal tensors $\hat{\mathbf{e}}_k\hat{\mathbf{e}}_k^\top$:

$$\mathbf{T} = \mathbf{R}\mathbf{\Gamma}\mathbf{R}^\top = \sum_{k=1}^{n} \lambda_k \hat{\mathbf{e}}_k \hat{\mathbf{e}}_k^\top$$

The unit orthogonal tensors $\hat{\mathbf{e}}_k\hat{\mathbf{e}}_k^\top$ do not have direction, but do have orientation. They can be visualised as thin ellipsoids of length 1 and all other dimension widths being 0. Multiplying these with the λ_k weights and summing them together presents the tensor $\mathbf{T}$ as a combination of these thin ellipsoids as in Figure 1.2(b).

In the case where all the eigenvalues are equal, we can see that no direction is preferred. This is called an *isotropic* tensor and the weighted sum of unit orthogonal tensors reduces to a weighted identity matrix:

$$\mathbf{T} = \mathbf{R}\Gamma\mathbf{R}^\top = \sum_{k=1}^{n} \lambda \hat{\mathbf{e}}_k \hat{\mathbf{e}}_k^\top = \lambda \mathbf{R}\mathbf{R}^\top = \lambda \mathbf{I}.$$

The representation of this tensor in 3D would be a *sphere* of radius λ, and in N dimensions a *hypersphere* of radius λ. We refer to it as a *ball* tensor. Expanding on this concept we can represent the tensor $\mathbf{T}$ as a linear combination of ascending tensor dimensionality:

$$\mathbf{T} = (\lambda_1 - \lambda_2)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top + (\lambda_2 - \lambda_3)(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^\top) + \cdots + (\lambda_{n-1} - \lambda_n)\sum_{i=1}^{n-1} \hat{\mathbf{e}}_i\hat{\mathbf{e}}_i^\top + \lambda_n \sum_{i=1}^{n} \hat{\mathbf{e}}_i\hat{\mathbf{e}}_i^\top. \quad (1.5.2)$$

Each of these terms represent *geometric features* of the tensor. The strength or *saliency* of each feature is given by the values $\lambda_1 - \lambda_2, \lambda_2 - \lambda_3, \ldots, \lambda_{n-1} - \lambda_n, \lambda_n$.

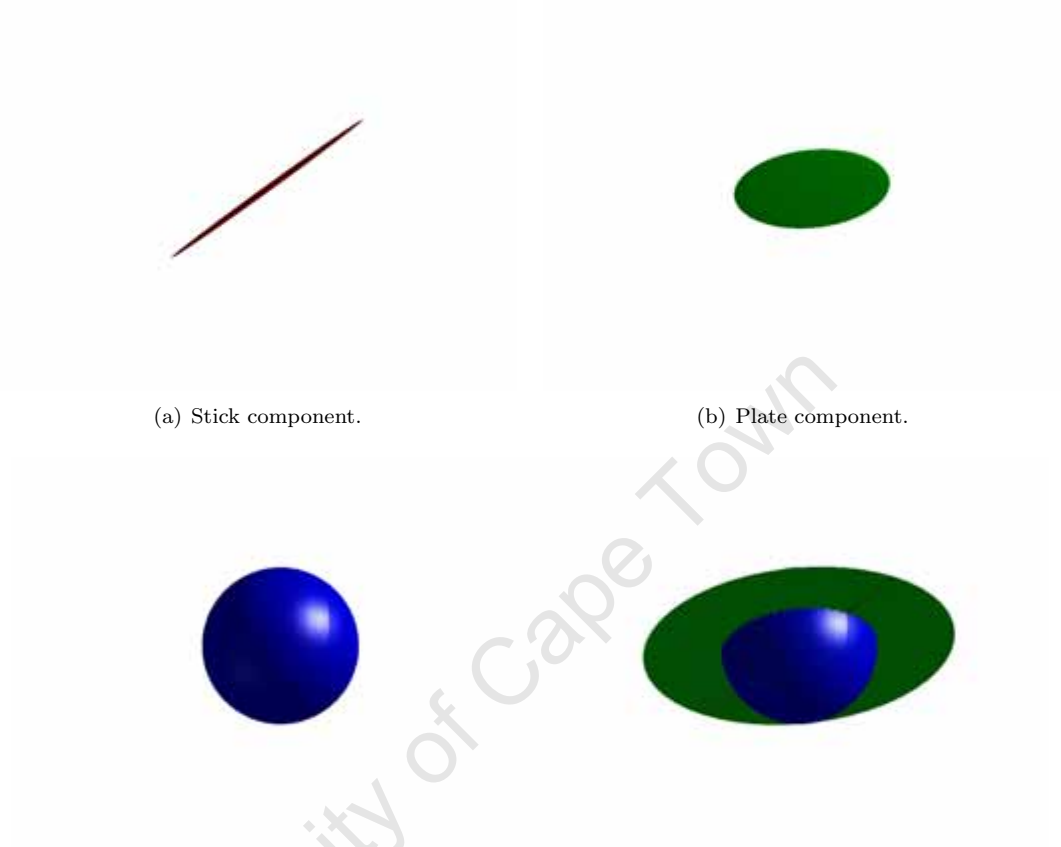
We can represent this decomposition graphically for the 3D case in Figure 1.3 as the combination of 3 geometric features of a stick (thin ellipsoid), a plate and a ball, where the stick tensor is given as $(\lambda_1 - \lambda_2)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top$, the plate tensor is given as $(\lambda_2 - \lambda_3)(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^\top)$, and the ball tensor is given as $\lambda_3(\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top + \hat{\mathbf{e}}_2\hat{\mathbf{e}}_2^\top + \hat{\mathbf{e}}_3\hat{\mathbf{e}}_3^\top)$.

Further significance of Equation (1.5.2) is that the linear combinations of eigenvectors give an indication of *normal* and *tangent* vectors associated with the term. For example, the first term $(\lambda_1 - \lambda_2)\hat{\mathbf{e}}_1\hat{\mathbf{e}}_1^\top$ is represented as a stick or thin ellipsoid and can be visualised for the 3D case in Figure 1.3(a). The ellipsoid has a single tangential vector which can lie up or down the length of the ellipsoid, and several normal vectors. If we constrain the normal and tangent vectors to be orthonormal, the number of normal vectors will be limited to $n - 1$.

The number of normal and tangential vectors can be extended to the remaining terms where the number of tangential vectors increases and the number of normal vectors decreases to the last term $\lambda_n \sum_{i=1}^{n} \hat{\mathbf{e}}_i\hat{\mathbf{e}}_i^\top$ which is representative of a ball tensor. In this case it is not really possible to infer tangent and normal vectors as there is no actual orientation. The meanings of the saliencies and vectors is given in Table 1.1. The geometric features and tensor names are as described by Nicolescu and Medioni [49], and generally refer to a 3D or 4D case. The geometric features are extensible to N dimensions by focusing on the normal and tangential properties of the structure.

Third-order tensors

Third-order tensors are introduced as a means of extracting further geometric information from the tokens. The reason for introducing the third order tensor is to be able to look for a geometric equivalent to skewness in statistics.



(a) Stick component.

(b) Plate component.

(c) Ball component.

(d) Combination of stick, plate and ball components.

Figure 1.3: A linear representation of tensor $\mathbf{T}$.

Table 1.1: General second-order tensor relationships.

Tensor	Saliency	Normals	Tangents	Feature
Ball	λ_n	none	none	Point
C-plate	$\lambda_{n-1} - \lambda_n$	$\hat{\mathbf{e}}_1 \dots \hat{\mathbf{e}}_{n-1}$	$\hat{\mathbf{e}}_n$	Curve
S-plate	$\lambda_{n-2} - \lambda_{n-1}$	$\hat{\mathbf{e}}_1 \dots \hat{\mathbf{e}}_{n-2}$	$\hat{\mathbf{e}}_{n-1}, \hat{\mathbf{e}}_n$	Surface
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Stick	$\lambda_1 - \lambda_2$	$\hat{\mathbf{e}}_1$	$\hat{\mathbf{e}}_2 \dots \hat{\mathbf{e}}_n$	Volume

Third-order tensors have the form $\mathcal{T}_{ijk} = a_i b_j c_k$ where the first order tensors **a**,**b**,**c** have a length n. In tensor voting we construct the tensors from a single first-order tensor $\mathbf{a} = (a_1, a_2, a_3, \ldots, a_n)$ and obtain the third-order tensor:

$$\mathcal{T}_{ijk} = a_i a_j a_k. \tag{1.5.3}$$

The form of the third-order tensor $\mathcal{T}$ is a three-dimensional matrix. The equivalent in statistics is skewness or the third-order moment. The three-dimensional form is not suitable for eigen analysis in the normal sense, but work in the Magnetic Resonance Imaging (MRI) field by Zhang [71] has identified Z-eigenvalues that perform a similar function. It is also difficult to visualize as the concept of the ellipsoid is not valid anymore.

Zhang identified an important property of the third-order tensor, and that is the rotational invariance of the largest and smallest z-eigenvalues, which is similar to the second-order case. An important aspect of this is the definition of apparent *skewness* in a projected direction, defined as

$$S_{app} = \frac{\mathcal{T}x^3}{\|x\|^3} \tag{1.5.4}$$

where

$$\mathcal{T}x^3 \equiv \sum_{i,j,k=1}^{3} \mathcal{T}_{ijk} x_i x_j x_k.$$

The apparent skewness is a scalar and can be equated to the eigenvalue in the second-order case.

In the tensor voting framework, the concept of skewness needs to be looked at in terms of a direction given by one of the eigenvectors of the second-order tensor. Using one of the the unitary eigenvectors $\hat{\mathbf{e}}_i$ that is aligned to the geometric feature in which we are interested, Equation 1.5.4 reduces to

$$S_{\mathbf{e}_i} = \sum_{j,k,l=1}^{3} \mathcal{T}_{jkl} e_{i_j} e_{i_k} e_{i_l}. \tag{1.5.5}$$

The sign of the skewness measure usually indicates whether the first-order tensors are clustered on either the one or the other side of the projection onto the eigenvector. The magnitude $\|S_{\mathbf{e}_i}\|$ is used as a measure of skewness in the orientation of $\hat{\mathbf{e}}_i$.

1.5.5 The problem with random numbers on n-spheres

In the cases where the orientation of a tensor is fully or partially unknown, it becomes necessary to generate a set of random tensor orientations in order to numerically determine its effect over a voting field (introduced later). The intuitive way of doing this in N dimensions is to use a random vector $\mathbf{x} \in \mathbb{R}^{n+1}, \quad \|\mathbf{x}\| = 1$, where the elements of $\mathbf{x}$ belong to the uniform distribution $U[-0.5, 0.5]$. This vector is normalised using $\mathbf{u} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ to the unit *n-sphere* or *hypersphere* represented as S^n. This approach has a problem in that the random variable $\mathbf{u}$ is not uniformly distributed over the n-sphere S^n [53].

A trivial solution to this problem is the intuitive solution to the S^1 sphere in $\mathbb{R}^2$ where a uniform random variable θ of dimension 1 and distribution $U[0, 2\pi]$ is transformed using the rotation matrix $[\cos(\theta) \quad \sin(\theta)]^\top$ to produce a random variable $\mathbf{x} = [\cos(\theta) \quad \sin(\theta)]^\top$. This random variable can be seen to be uniform on the perimeter of the S^1 sphere (circle). Extending this to the 3D unit sphere, through the use of rotational matrices in 3 dimensions needs two random variables ϕ and θ as a polar representation of the unit S^2 sphere. The distribution of θ is $U[0, 2\pi]$ as in the case of the 1-sphere, and the distribution of ϕ is $U[0, \pi]$. This will result in a non-uniform distribution on the S^2 sphere as the points will cluster at the pole and be more sparse on the equator of the S^2 sphere. By weighting the distribution of ϕ with a cos weight, this can be rectified but the process becomes involved at higher dimensional S^n spheres.

There are several solutions for higher dimensions given in the literature such as the *subgroup algorithm* of Diaconnis and Shashahani [8]. This solution needs QR factorisations in order to transform a uniform random variable onto the n-sphere and would not be appropriate in numerical Monte Carlo studies. For numerical methods, there are three solutions.

Monte Carlo rejection method
The rejection method uses uniform random variables $\mathbf{x} \in \mathbb{R}^{n+1}$ with individual distributions $U[-1, 1]$. The random points are evaluated to see if they fall inside the n-sphere by evaluating $\|\mathbf{x}\| \leq 1$. If this inequality does not hold, then the point is rejected. If the inequality holds, the point is retained and the random variable $\mathbf{u} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ lies on the unit n-sphere and is uniformly distributed over S^n. This method is easy to implement, but the number of points within the unit n-sphere drops exponentially as the dimensionality increases which makes this hit-and-miss technique badly suited for dimensionalities over 10. In tensor analysis, the dimensionality can reach this level fairly quickly.

Coordinate method
The coordinate method looks to determine the distribution of each coordinate of the n-sphere uniformly distributed variables. This is a complex undertaking in N dimensions [53] that involves an

iterative approach as each dimension is transformed. In the algorithm, the inverse of a cumulative distribution function needs to be solved numerically as no closed-form solution exists. The solution to the inverse of the cumulative distribution function is done using an approximation method such as Newton's method. The method is fairly complex but scales approximately linearly as the dimensionality of the n-sphere increases.

Gaussian method

The Gaussian method was proposed by Knuth [32] and relies on the characteristics of a multi-variate independent Gaussian distribution:

$$f(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^n} e^{-\frac{1}{2}\langle \mathbf{x}, \mathbf{x} \rangle} \tag{1.5.6}$$

which has a density of $N(0,1)$. What is of interest is that if we transform the random variable $\mathbf{x}$ using a rotation matrix $\mathbf{R}$ such that $\mathbf{y} = \mathbf{R}\mathbf{x}$, then we find that the distribution of $\mathbf{y}$ is $N(0,1)$ as well. This is due to the $\langle \mathbf{x}, \mathbf{x} \rangle$ inner product in Equation 1.5.6 being transformed to $\langle \mathbf{R}\mathbf{x}, \mathbf{R}\mathbf{x} \rangle$ and knowing that $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$. This invariance to rotation leads to a Monte Carlo method where we choose a random vector $\mathbf{x} \in \mathbb{R}^{n+1}$ with distribution $N(0,1)$ and project it onto the unit n-sphere S^n by $\mathbf{u} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ which are uniformly distributed on the n-sphere. Numerically, the Box-Muller method [3] can be used and the computational complexity is linear. This method scales well for high dimension n-spheres and can be appropriate for this thesis.

Another candidate as a normal random number generator is the one that MATLAB uses. MATLAB uses a method based on the Ziggurat [41] algorithm. This algorithm makes use of preset lookup tables and layered equal area rectangles covering the target Gaussian density. Several variants of this algorithm exist, but the one described by Marsaglia [41] provides an efficient method running 99% of the time from look-up tables.

When working with higher-order tensors, the results are sensitive to non-ideal random number generators. Care needs to be taken in choosing a pseudo-random number generator such that the results in the Monte Carlo analysis are well behaved. For the purposes of this thesis, the Ziggurat method as described by Marsaglia [41] is adopted. This was validated during the course of the thesis by simulation.

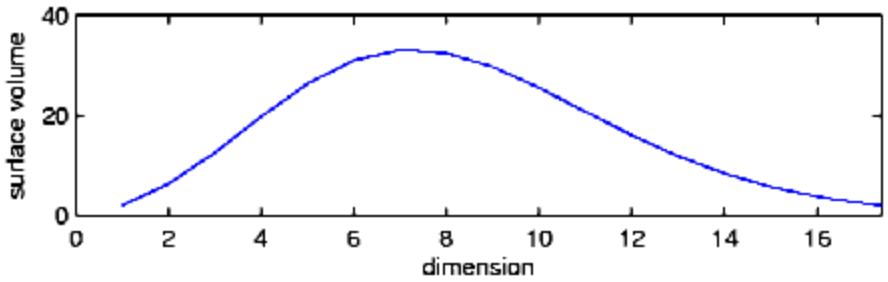


Figure 1.4: *surface volume* S_{n-1} as a function of dimensionality.

1.5.6 The shrinking n-sphere

In the tensor analysis to follow, the dimensionality of the problem is often high. In order to generate sufficient random numbers on the unit n-sphere, the concept of n-sphere surface area need to be introduced as this is the value that represents the number of random numbers to be generated for stable results, as well as the relationship between several dimensions where the cummulative effect of the random points must be balanced.

In Monte Carlo analysis, it is necessary to balance the number of random points for a specific unit sphere S^n of dimensionality n. This is done by weighting the number of iterations used in the Monte Carlo analysis by the unit *n-sphere surface area*, or more correctly the *surface volume* of dimensionality $n-1$.

The surface volume can be denoted as:

$$S_{n-1} = \frac{2\pi^{n/2}}{\Gamma(n/2)} \tag{1.5.7}$$

where $\Gamma(\cdot)$ is the gamma function. When this surface volume is determined for varying n as in Figure 1.4, we observe an inflection showing that the surface area diminishes as $n > 7$. This is counter-intuitive, and must be borne in mind in higher-dimensional problems that can and do occur in tensor voting. The solution to this problem is to increase the number of Monte Carlo iterations accordingly to obtain sufficiently good voting results, or to fix the iterations if it is only a ball vote.

Chapter 2

Discussion of current methods

We look specifically at tensor voting and allied methods that have been developed in the literature. We reproduce the main line of algorithm development and look briefly at the various successes in the line of video segmentation that have been obtained using tensor voting. After surveying the methods, we return to the particular problem of moving-object segmentation in video sequences and discuss these techniques applied to this problem, focusing on whether there is room for improvement.

2.1 Introduction

In the previous chapter the mathematical background necessary for tensor voting was discussed. In this chapter we look at relevant literature limited to the field of tensors applied in the image and video field. Literature on generalised motion segmentation is not covered as the area is not within the confines of the thesis and would not be entirely relevant.

The work is referenced in chronological order to preserve the evolution of tensor techniques.

The generalised tensor voting framework is introduced in this section as it appears in numerous references cited. The tensor voting framework is expanded on in subsequent chapters to include tangential voting, direct data encoding and skewness measures.

2.2 Previous work on tensors in the video segmentation field

The motion estimation problem approached by using orientation tensors has been done by Farnebäck [13, 12] in 2001 in his thesis [14] and previous formulations [11, 9, 10]. He makes use of the popular Yosemite series (without the clouds) with a technique that uses an affine motion model with simultaneous motion estimation and segmentation. In this paper he attains average error of 1.14° for 100% density of pixels, which was better than several existing methods of the time.

Han together with Medioni [22] put an initial foundation down in tensor voting by using the precursors to the tensor voting kernel on document line deskewing, where the orientation of the text lines are estimated.

Lee uses Tensor Voting in her PhD thesis [35] in 1998. The thesis defines the tensor voting framework and inference of geometric features from the saliences of the tensors. The thesis also introduces the sparse to dense method of densification that most further work in tensor voting in the field of image and video processing use.

In 1999, Gaucher together with Medioni [16] look into using Tensor Voting as extended in this thesis to do motion segmentation over 3 frames. Effort is spent on separation of the motion layers and occlusion. This technique is applied to the Yosemite sequence (where no occlusion exists) with an average error of 8.83° for 100% usage of pixels.

In a work using the z component further than the 2 or 3 frame problem, Haussecker [24] makes use of the 3D volume made up of an intensity image. He imposes the *brightness change constraint* on the volume to define a structure tensor. This tensor makes use of local gradients and local binomial

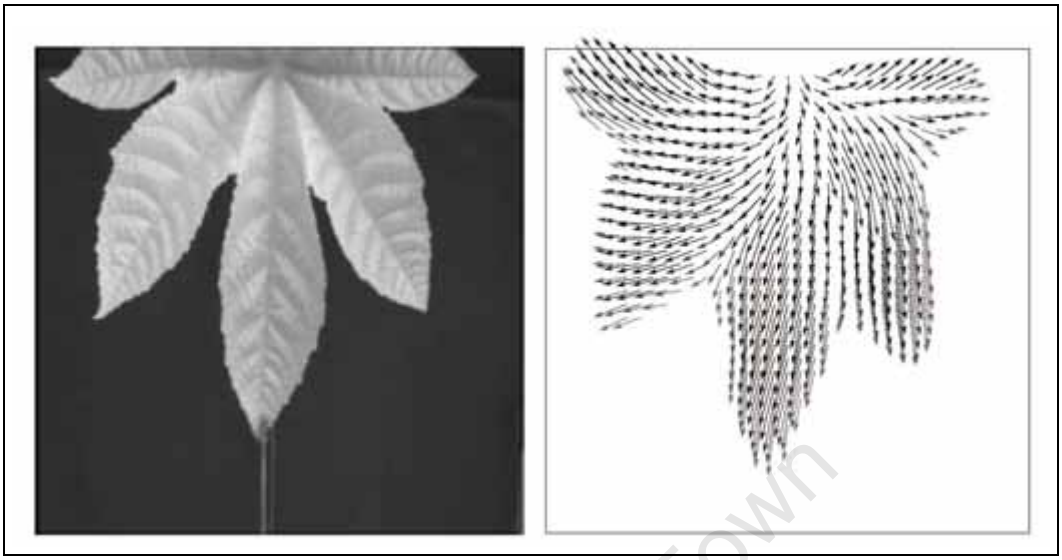


Figure 2.1: Use of structure tensor from Haussecker taken from [24].

smoothing. The formulation looks at coherency to define motions grouped together and an edge measure to detect the aperture problem. The resulting motion fields are shown in Figure 2.1.

Kang, Cohen and Medioni [30] use the tensor framework in joint image space to estimate several affine motions. This is compared to RANSAC and favorable results are shown by looking at image differences after motion compensation. A further paper [31] uses tensor voting over a video scene to track persons and vehicles. The tensor approach is used to filter and connect track trajectories in a multi-object, multi camera system. Kornprobst and Medioni [33] also look at the problem of tracking objects and the improvements that can be gained using tensor based voting to determine singular trajectories.

Tang and Medioni [58] use tensor voting to get saliency maps of surfaces and curves in 3D. After a densification step, their method of using extremals to delineate geometric features in 3D space is applied. This is effective in delineating surfaces and curves in noisy 3D data. In further work [60] they refine the algorithm by using tensor voting that takes curvature into account. This is built into three different kernels and is applied on the data separately to determine the saliency with which the data complies with the kernel. Tang, Medioni and Lee [63] use tensor voting in an 8D space to do epipolar geometry estimation from stereo images. This problem is effectively addressed using the concept of extremals and several passes through the tensor voting framework to refine the answers.

Mordohai and Medioni [46] use the tensor voting framework on stereo imaging to improve the disparity maps. This is further extended by Lee, Medioni and Mordohai [36], who use tensor voting on stereo image disparity maps to improve surface estimation of 3D objects. The tensor voting

deviates from other stereoscopic techniques in that it presents a unified approach in dealing with communication and interpretation of the disparity map. Figure 2.5 shows the results obtained on the disparity map by Lee et al.

Lui, Chellappa and Rosenfeld [38] use an adaptive structure tensor based on a parametric model to do dense motion estimation. The approach yields comparable results to other leading edge methods on numerous test examples. The results are reproduced on the Yosemite scene that Farnebäck also uses. The adaptive affine method proposed manages to produce a mean error of 1.39^o for 100% pixel density. The result is shown in Figure 2.6.

Massad, Balbos and Mertsching [42] apply tensor voting together with Gabor filtering on grey scale images to infer better contours and junctions. The saliency maps are used to extract edge information.

Medioni, Tang and Lee [43] present a consolidated paper of the theory and applications of tensor voting. This paper is a synopsis of the work done at University of Southern California over a period of several years. The range of applications includes motion flow estimation in video sequences, 3D shape from stereo imaging and 3D smooth shape restoration.

Tong, Tang, Mordohai and Medioni [67] introduce the concept of augmenting the second order tensor voting with first order tensor voting to get polarity from the regions edges. The first order voting consists of the addition of vectors at the votee site giving the direction of the majority of the voters weighted by distance and curvature. The method is very similar to the skew tangential voting introduced later except that the augmentation method uses two steps (outlier rejection followed by densification) to get the polarity, while the method to be proposed extracts edge characteristics in the second order voting process without densification. The augmentation method produces two results in the voting process which is the first and second order tensors. The skew tangential voting gives orientation from the second order voting process and then augments it with a more complex third order voting process to determine boundary points.

Nicolescu and Medioni [49] use tensor voting to do video motion segmentation on two frames. The resulting segmentation is good in selecting regions of similar motion, but led to a further paper [50] that includes a refinement of the boundaries using the image gradient and orientation in a 2D voting process. The results on a fish sequence are shown in Figure 2.3. These results are pertinent to this thesis, and do form part of the methodology.

Jia together with Tang [28] use the tensor voting approach and extend it into the N dimension space. This was primarily to infer missing data in images and 3D objects. The inference of missing data makes use of iterative stick voting. This gradually erodes the missing data with inferred data. Good results are obtained, as seen in Figure 2.7. Jia and Tang [29] also look at using tensor voting

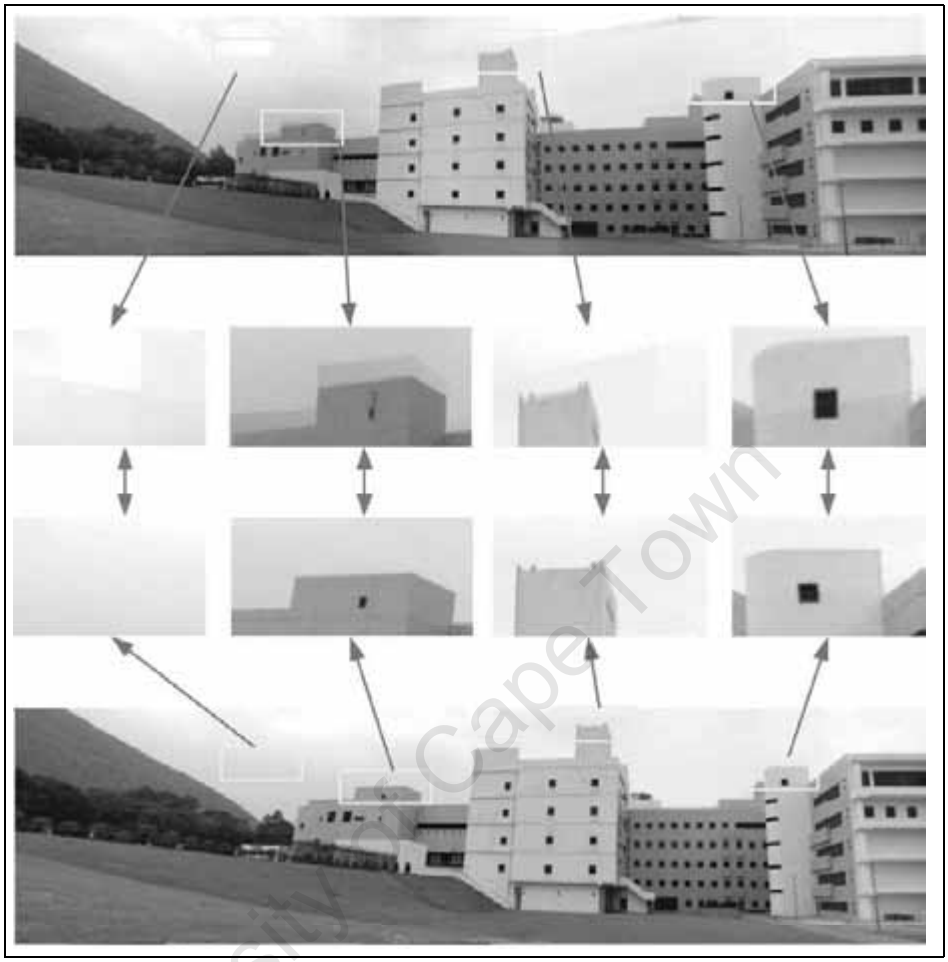


Figure 2.2: Correction of local intensities in mosaic image reconstruction taken from [29].

for local and global intensity alignment in multiple images. The technique is used to do mosaicing and image enhancement in a fashion that surpasses histogram correction techniques. The results are shown in Figure 2.2. Jia et al. [26] extend the tensor voting solution to inference of missing data to allow estimation of missing cyclic movement in video sequences, as shown in Figure 2.4.

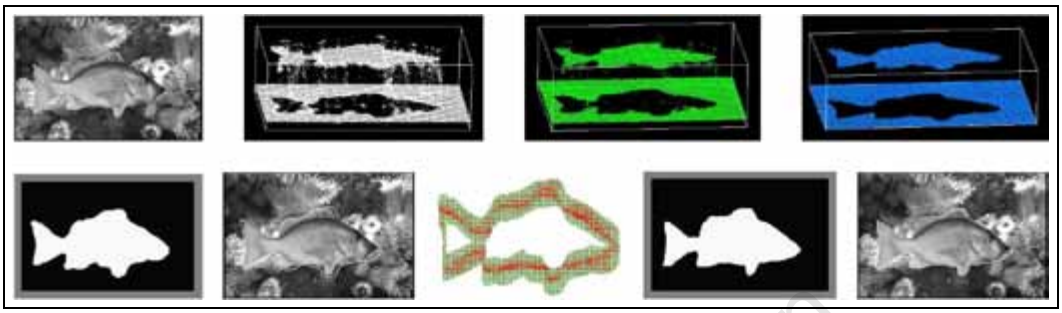


Figure 2.3: Results taken from [50] showing the tensor voting approach used in segmenting a fish from 2 frames of a video sequence and the subsequent boundary refinement.

Figure 2.4: Estimation of cyclic motion in a video sequence taken from [26].

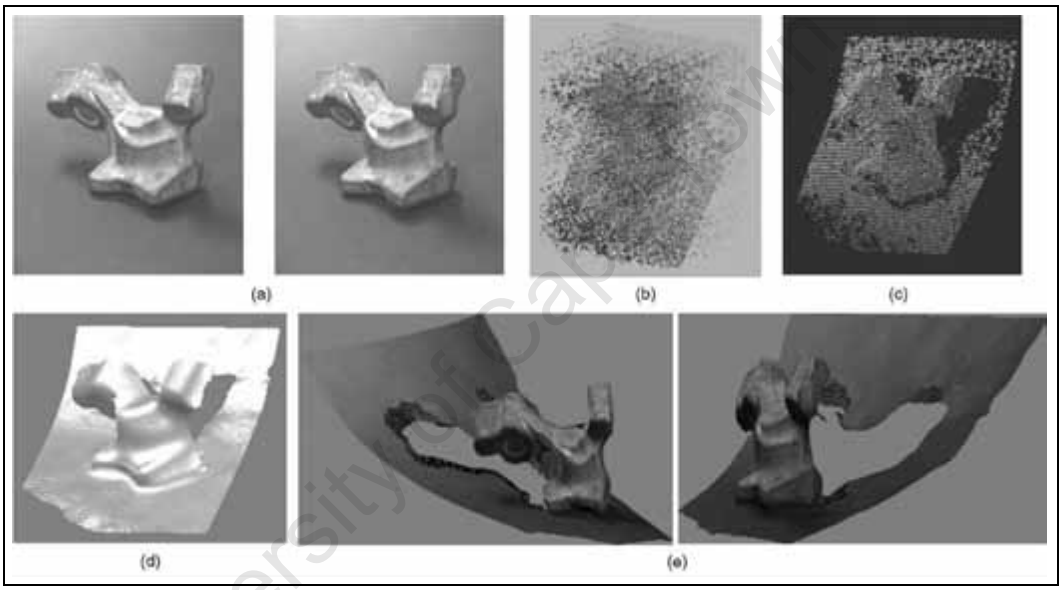


Figure 2.5: Results taken from [36] showing (a) the original images, (b) the initial correspondences, (c) the unique disparity assignments, (d) the inferred surface in disparity space and (e) the texture mapped views.

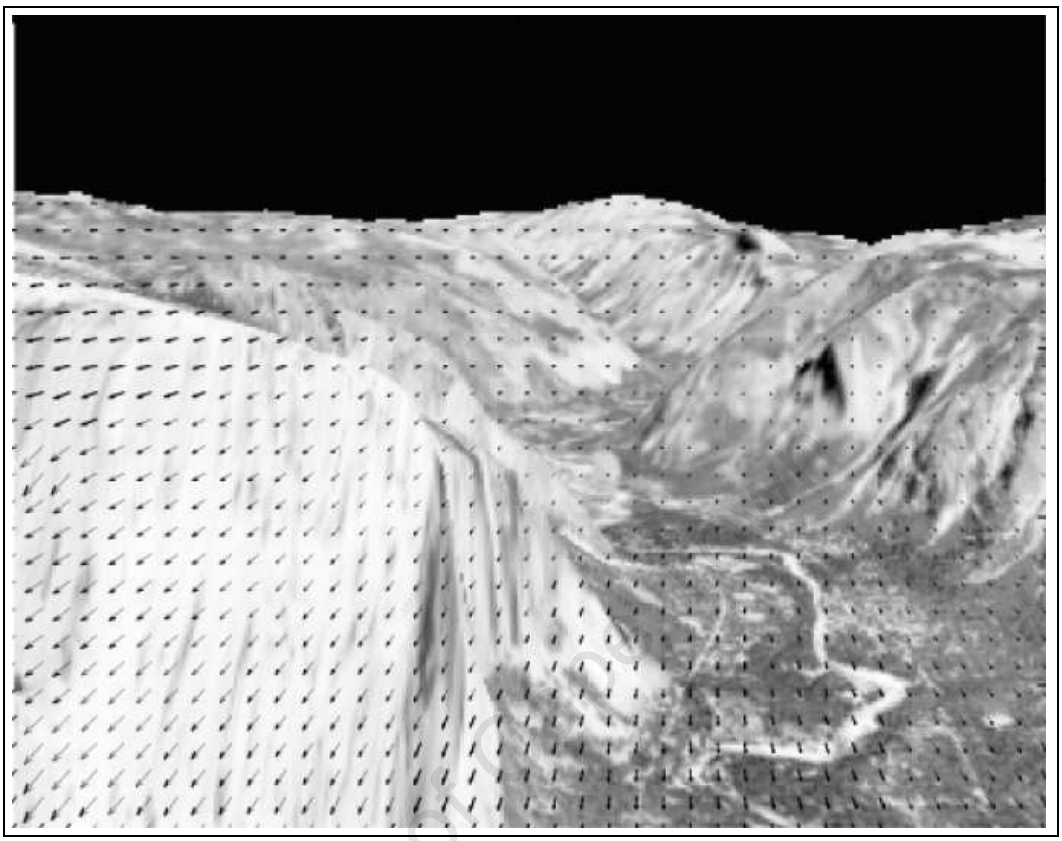


Figure 2.6: Algorithm reproduced from [38] showing the motion field estimation of the Yosemite sequence.

Figure 2.7: Inference of missing data taken from [28].

2.3 Tensor voting framework

The tensor voting framework is compiled from several sources, the most important being Medioni [43] and Nicolescu [49, 50].

A video or image sequence is made up of pixels defined in a specific basis frame (usually (x, y, t)) and for attributes of each pixel that may or may not be related to the positions (x, y, t), such as v_x, v_y, and colour or greyscale information. This information is partially or completely used in making up *tokens*. In the cases that follow, the components (x, y, v_x, v_y) are used and are basic geometric (positional) quantities.

The basis of tensor voting is to use a region of support $\mathcal{R}$ around a token to determine whether the token forms part of a geometric structure such as a curve, volume or junction while simultaneously allowing a measure of noise rejection. By using second-order tensor representations, the second moment allows curvature and tangents on curves and surfaces to be described. Tokens are represented in a tensorial way for first-order tensors:

$$\mathbf{t}_i = (x_{1_i}, x_{2_i}, ..., x_{n_i}). \tag{2.3.1}$$

Extending this to the second-order tensor, we get

$$\mathbf{T}_i = \mathbf{t}_i \mathbf{t}_i^\top. \tag{2.3.2}$$

The second-order tensor is positive semidefinite and this characteristic is retained when the second-order tensors are accumulated:

$$\mathbf{A} = \sum_i \mathbf{T}_i. \tag{2.3.3}$$

The accumulation allows meaningful Monte Carlo analysis to be done and allows eigenanalysis to yield good approximations of the geometric features much in the same way as a covariance matrix.

2.3.1 Second-order tensor data representation

The second-order tensor data representation has been described in 1.5.4. The generalised equation 1.5.4 can be interpreted in the 3D ($n = 3$) sense as:

$$\mathbf{T} = (\lambda_1 - \lambda_2)\mathbf{S} + (\lambda_2 - \lambda_3)\mathbf{P} + \lambda_3 \mathbf{B}, \tag{2.3.4}$$

where $\mathbf{B}$ is the ball component having no particular orientation. This can be visualised as a sphere and is defined by $\hat{e}_1\hat{e}_1^\top + \hat{e}_2\hat{e}_2^\top + \hat{e}_3\hat{e}_3^\top$. The plate component is given by $\mathbf{P}$ and has no orientation around two of the three axes. This assesment can be visualised as a disk/plate and is defined by $\hat{e}_1\hat{e}_1^\top + \hat{e}_2\hat{e}_2^\top$. The last component is the stick component given by $\mathbf{S}$, which is aligned to the $\hat{e}_1$ axis.

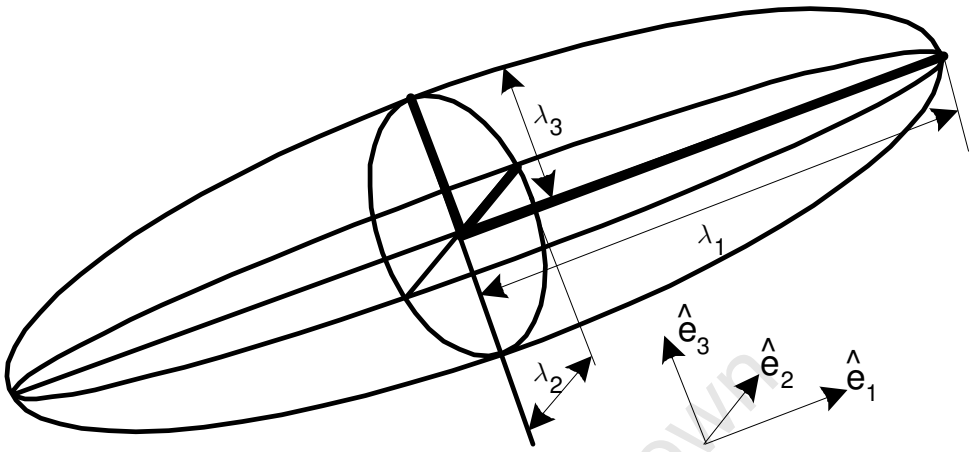


Figure 2.8: 3D ellipsoid tensor representation.

This can be visualised as a stick/line and is defined by $\hat{e}_1\hat{e}_1^\top$. In Equation 2.3.4 the eigenvectors and eigenvalues describe an ellipsoid as shown in Figure 2.8.

For the 3D case the different components allow us to differentiate geometric features. These consist of points that have no firm direction (**B**), points on surfaces or plates (**P**) and points that are on lines and curves (**S**). The coefficients of these terms are the saliency measures and have the following characteristics:

1. **Point saliency**. This is characterized by very similar eigenvalues ($\lambda_1 \approx \lambda_2 \approx \lambda_3$) and has no preferred direction. The saliency value is given by λ_3.

2. **Curve saliency**. This is characterized by two similar eigenvalues larger than the third ($\lambda_1 \approx \lambda_2 > \lambda_3$). The measure is $\lambda_2 - \lambda_3$ being large in value indicating a strong curve directionality (belonging to a line) with a tangent unit vector $\hat{e}_1$, and the normal unit vectors are given by $\hat{e}_2$ and $\hat{e}_3$. The saliency value is defined as $\lambda_2 - \lambda_3$.

3. **Surface saliency**. This is characterized by one eigenvalue larger than the second and third ($\lambda_1 > \lambda_2 \approx \lambda_3$). The measure is $\lambda_1 - \lambda_2$ being large in value indicating a strong surface affinity (belonging to a surface), and the normal unit vector is given by $\hat{e}_3$. The tangent unit vector is given by $\hat{e}_1$ and $\hat{e}_2$ The saliency value is defined as $\lambda_1 - \lambda_2$.

The 2D case, which has the smallest dimensionality, is defined as:

$$\mathbf{T} = (\lambda_1 - \lambda_2)\mathbf{S} + \lambda_2\mathbf{B} \tag{2.3.5}$$

where only points and curves are present, not surfaces as there are no plate tensors.

In the tensorial representation, the formulation of Equation 2.3.4 is extensible to N dimensions:

$$\mathbf{T} = (\lambda_1 - \lambda_2)\mathbf{S} + \sum_{i=2}^{n-1}(\lambda_i - \lambda_{i-1})\mathbf{P}_i + \lambda_n\mathbf{B} \tag{2.3.6}$$

where $\mathbf{P}_i$ are $n-2$ plates and $\mathbf{P}_i = \sum_{j=1}^{i} \hat{e}_j\hat{e}_j^\top$. These have been given names such as C-plate (Curve plate) and S plate (Surface plate) [49] in the 4D case, but these names are fairly arbitrary in the N dimensional case.

The various saliencies get modified to adapt to the N dimensional meaning:

- $(\lambda_1 - \lambda_2)$ gives the *hyper-surface* saliency with a single normal direction given by $\hat{e}_1$ and $n-1$ tangential directions given as $\hat{e}_i$ with $2 \leq i < n$.
- The $n-2$ plate tensors whose saliency is described as $(\lambda_i - \lambda_{i+1})$ where $2 \leq i < n$. The orientation uncertainty is described by $\sum_{j=1}^{i} \hat{e}_j\hat{e}_j^\top$. The tangent(s) are described by $\hat{e}_t$ where $i < t \leq N$, and the normal(s) by $\hat{e}_n$ where $0 < n \leq i$. This results in a $(n-i)$ dimensional feature in N dimensional space. In the case of 3D ($n = 3$ and $i = 2$), this would represent a 1D feature, which is a curved line with two normals $\hat{e}_1$ and $\hat{e}_2$ and one tangent $\hat{e}_3$. In 4D ($n = 4$ and $i = 2$), this would represent a 2D feature which is a surface, with two normal directions, $\hat{e}_1$ and $\hat{e}_2$, and two tangential directions, $\hat{e}_3$ and $\hat{e}_4$.
- λ_n refers to the *hyper-junction* saliency. In this case there is total uncertainty of orientation.

2.3.2 Second-order tensor data communication

In order to construct the second-order tensors and make use of their properties, it is necessary to allow communication between the tokens from the surrounding region of support $\mathcal{R}$.

Tensor voting makes use of tensor communication in order that tokens at various points can vote at other token points. This is described by a *kernel*. A typical case, and the one used most in the literature, has the vote decay with *distance* and with *curvature* according to a Gaussian (e^{-x^2}) function. A kernel that can describe this decay allows the scalar *vote strength* to be described as

$$VS_{stick}(s, \kappa) = \exp^{-\frac{(s^2+\alpha\kappa^2)}{\sigma^2}}, \tag{2.3.7}$$

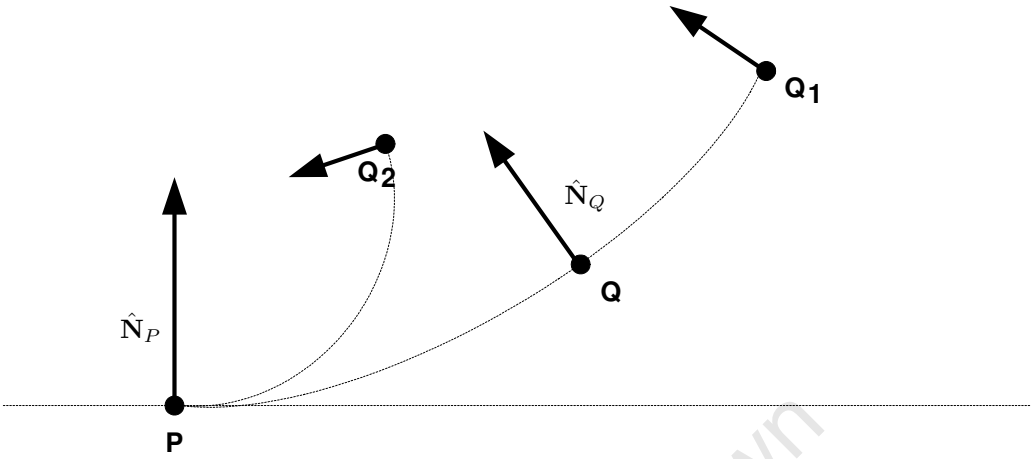


Figure 2.9: 2D vote strength if directional information is known.

where s is the arc length joining the tokens as shown in Figure 2.11, κ is the curvature, α is a curvature scaling factor and σ is the radial distance scaling factor. This form of vote strength is applicable if the orientation and position of the voter is known explicitly, as is the case of the stick data representation $\mathbf{S}$.

If we look at the 2D case, the vote strength in Figure 2.9 of voter P on votee Q is seen to get less for increased radial distance (Q_1) and increased curvature (Q_2). If the effect over the xy plane in 2D is mapped, then Figure 2.10 indicates very little strength broad-side to the voter (high curvature) and a general radial decay (greater distance). The arc length and curvature can be derived geometrically from Figure 2.11 and are given by:

$$r = \frac{l}{2\sin(\theta)}; \quad \kappa = \frac{1}{r}; \quad s = 2r\theta. \tag{2.3.8}$$

It is better to use equations containing $\cos(\alpha)$ as this parameter is easily calculate using dot products between unit vectors. The set of equations can be written as:

$$r = \frac{l}{2\cos(\alpha)}; \quad \kappa = \frac{1}{r}; \quad 2\theta = \pi - 2\arccos(|\cos(\alpha)|); \quad s = 2r\theta. \tag{2.3.9}$$

Furthermore, the *scale* is denoted by σ. This determines the decay of the vote strength with distance and curvature. An additional constant α scales the radial decay and curvature decay in relation to each other.

Up until now, we have been dealing with the vote *strength* which is a scalar. In order to determine direction, the normal unit vector at point $\mathbf{P}$ must be known. This is denoted as $\hat{\mathbf{N}}_P$. In the voting

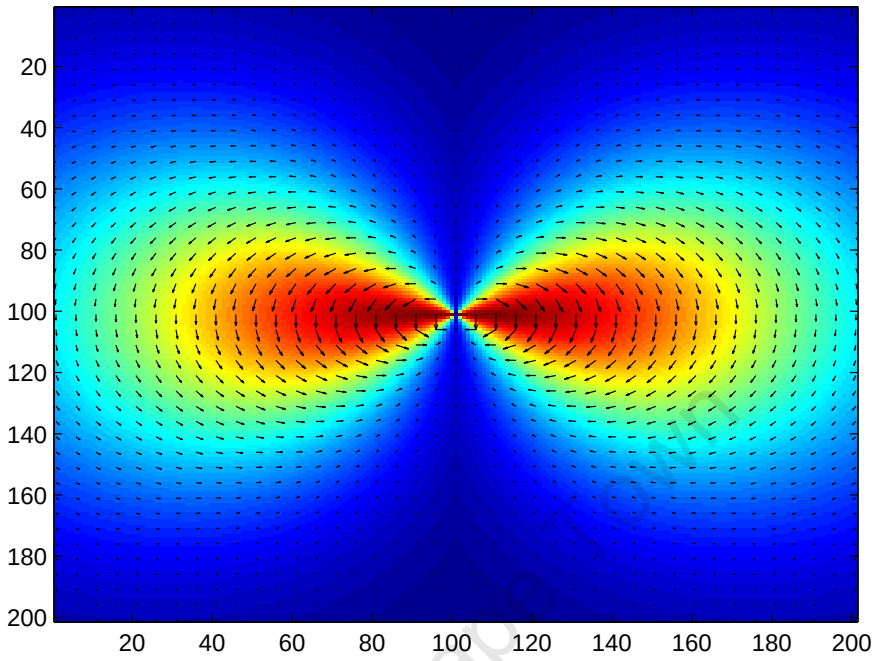


Figure 2.10: 2D $VS_{stick}(x, y)$ showing both the strength and the vector field. Blue indicates low strength and red indicates high strength.

process, $\hat{\mathbf{N}}_P$ is known, but $\hat{\mathbf{N}}_Q$ must be found. In order to do this, we extend the unit vector $\hat{\mathbf{N}}_P$ to the center point $\mathbf{C}$ and then align it back to point $\mathbf{Q}$:

$$\mathbf{C} = r\hat{\mathbf{N}}_P + \mathbf{P}; \quad \mathbf{N}_Q = \mathbf{C} - \mathbf{Q}. \tag{2.3.10}$$

The first-order tensor (unit vector) at point $\mathbf{Q}$ is denoted by $\hat{\mathbf{N}}_Q$. Including the vote strength we get:

$$\mathbf{v}_{stick}(s, \kappa) = VS_{stick}\hat{\mathbf{N}}_Q, \tag{2.3.11}$$

and extending to the second-order tensor:

$$\mathbf{V}_{stick}(s, \kappa) = \mathbf{v}_{stick}\mathbf{v}_{stick}^{\top}. \tag{2.3.12}$$

The above equations are in terms of (s, κ). Using Equation 2.3.8 and the relationships:

$$l = \sqrt{{x_Q}^2 + {y_Q}^2}; \qquad \theta = \arcsin(\frac{y_Q}{x_Q})$$

we can determine VS_{stick} at point Q in terms of (x_Q, y_Q) and we can denote this as vector $\mathbf{q}$. We can plot the kernel vote strength as a function of (x_Q, y_Q) as in Figure 2.10.

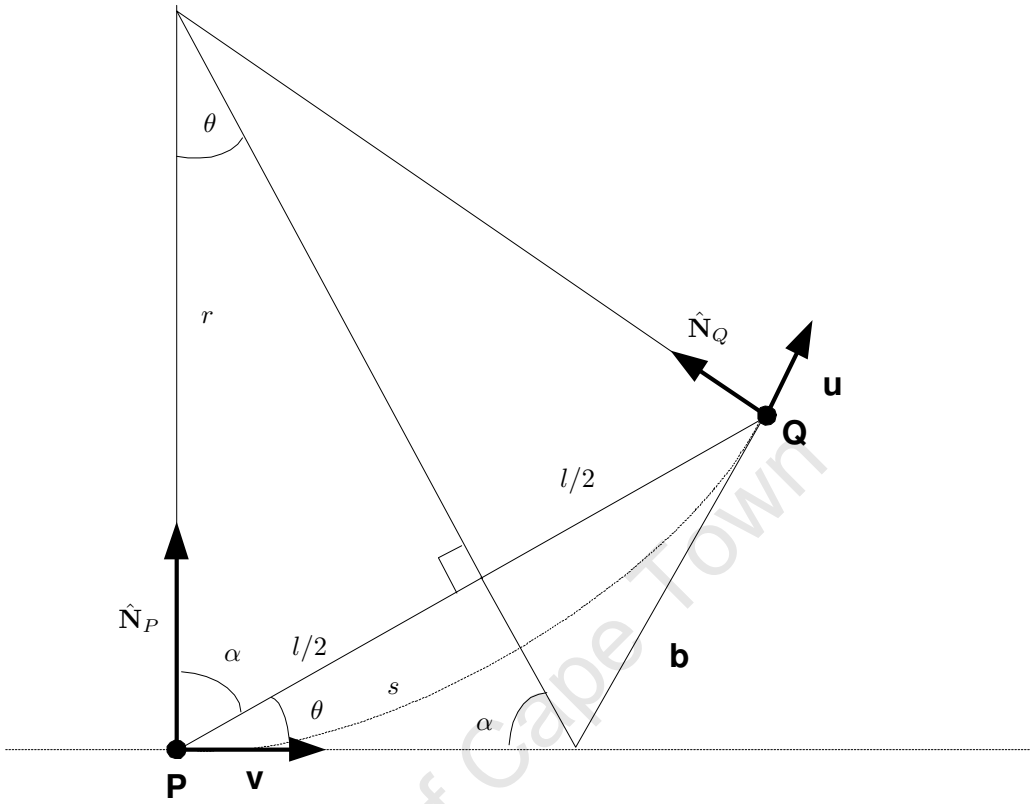


Figure 2.11: Geometry of 2D stick vote.

We can extend Equation 2.3.12 to 3D by noting that the stick vote strength VS_{stick} is radially symmetric around the x axis. This is indicated in Figure 2.12(a). Essentially, all dimensions higher than 2D can be reduced to a 2D problem by rotating the plane defined by the vector $\hat{\mathbf{N}}_Q$ and the receiver point Q such that it is the 2D plane defined by xy in Figure 2.11.

The stick vote is applicable when the normal vector $\hat{\mathbf{N}}_P$ is known. There are many instances where it is completely unknown, or is only known in some of the dimensions. In these cases, all possible unknown orientations are integrated. In the 3D case, the plate vote $\mathbf{V}_{plate}(\mathbf{q})$ can be found by integrating the stick vote around the z-axis to produce

$$\mathbf{V}_{plate}(\mathbf{q}) = \int_0^{2\pi} R^{-1}_{\theta\phi\gamma}(\mathbf{V}_{stick}(\mathbf{q}R_{\theta\phi\gamma}))R^{-T}_{\theta\phi\gamma}d\gamma|_{\theta=0;\phi=0}, \tag{2.3.13}$$

where the stick vote is first rotated with the rotation matrix $R_{\theta\phi\gamma}$ such that it is aligned with the $\hat{e}_1$ vector of the stick vote at point P. The rotations θ, ϕ and γ are around the x, y, z axis respectively. The rotation corresponds to the situation where there is certainty in one axis and not the other two. The integration takes the form of a circle in the xy plane. We can visualize the rotation as shown

in Figure 2.12(b).

If there is no certainty in direction, a ball vote $\mathbf{V}_{ball}(\mathbf{q})$ can be found by integrating over the whole sphere:

$$\mathbf{V}_{ball}(\mathbf{q}) = \int_0^{2\pi} \int_0^{2\pi} R_{\theta\phi\gamma}^{-1}(\mathbf{V}_{stick}(\mathbf{q}R_{\theta\phi\gamma}))R_{\theta\phi\gamma}^{-T} d\gamma d\phi|_{\theta=0}. \tag{2.3.14}$$

The result of the ball vote in 3D is shown in Figure 2.12(c). The integrations can be substituted with summations in the discrete case.

The votes are iteratively done on the votees, where the votee set comprises of all the elements $\mathbf{T}_i$. The voters are drawn from the same set within a radius (defines the region of support $\mathcal{R}$) such that contributions beyond this radius are negligible. The contributions become negligible due to the radial exponential decay defined by σ. Any points further than 3σ are deemed negligible. All the voter's second-order tensor votes are tensorially added and then the eigenvalues and eigenvectors are determined, allowing the mentioned saliencies to be computed. The saliencies determine the characteristics of the votee point in relation to its surrounding elements. In the 3D case, a votee may be characterised as being an independent point, as a point within a volume with its *point* saliency, as a point on a 3D surface by its *surface* saliency, or as a point on a curve or line with its *curve* saliency.

(a) Stick vote $VS_{stick}(x, y, z)$.

(b) Plate vote $VS_{plate}(x, y, z)$.

(c) Ball vote $VS_{ball}(x, y, z)$.

Figure 2.12: 3D representations of stick, plate and ball votes.

2.4 Tensor voting framework applied to motion segmentation

Tensor voting in moving object segmentation as it has been defined in the literature [35, 43, 45, 49] provides a good measure of finding geometric features. The tensor voting method almost always follows the flow indicated in Figure 2.13.

- *Selection of input tokens.* The selection of input tokens is sparse and is usually based on a feature detector such as a corner detector for 2D image position, and on correlation matching (using a region $\mathcal{R}$) to obtain subsequent frame displacement velocities (v_x, v_y). Normally [49], several match candidates using different block matching sizes are placed in this set.
- *Ball tensor voting.* An unaligned tensor vote is carried out, and the surface saliency is extracted for all the sparse points. Points that are below 10% of the maximum surface saliency are censored from the token set.
- *Refined tokens.* The refined tokens are a subset of the sparse input tokens containing only tokens that have a good surface saliency. Tensor voting can be applied again to further refine this subset, and to get better surface estimates by excluding outliers.
- *Densification.* In order to assign a velocity to every (x, y) point in the image, a set of candidates based on the minimum and maximum velocities are assigned to each open (x, y) location, and voting with censorship is applied repeatedly. This erodes the open (x, y) locations until the token map is complete in (x, y). Usually some apriori information is applied [49] to prevent impossible selections or uncontrolled growth of surfaces.
- *Dense saliency map.* After densification each (x, y) point in the 2D image has an assigned surface saliency.
- *Feature extraction.* Feature extraction is a heuristic process where surfaces are separated from each other using some dissimilarity algorithm, so that the surfaces can be individually labeled.
- *Features.* Each (x, y) point in the image is now labeled according to motion.

The work done by Nicolescu [49] succeeds in finding moving regions in a data set corrupted with many outliers. Using simple synthetic shapes such as disks, the segmentation is good, but when natural images are used the edges are indistinct as shown in Figure 2.14. This is to be expected due to the densification stage using no image data to make the decision as to where the edges of the object are, and the method is similar to a watershed system where two opposing seed areas approach each other. The densification populates the unassigned spaces equally fast preserving rounded boundaries well. By using a disk, the densification completes populating the unassigned spaces more-or-less on the boundary [47].

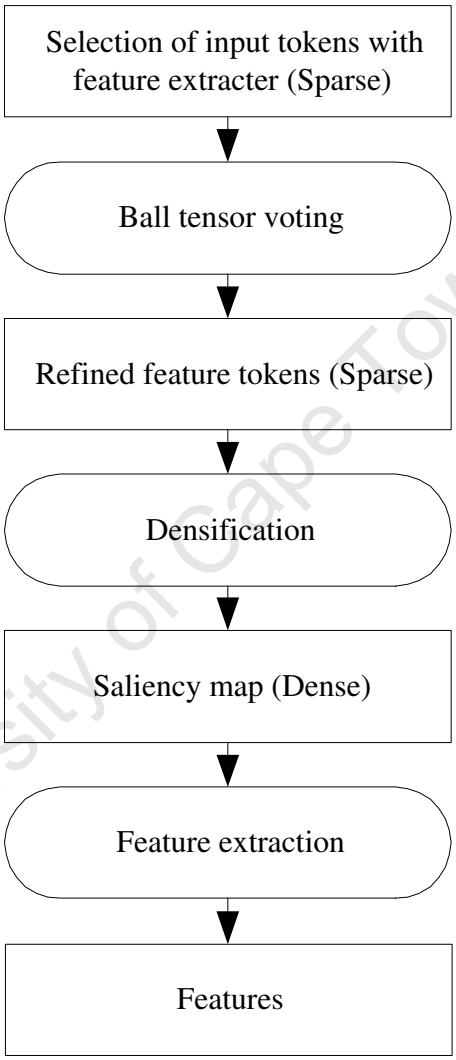


Figure 2.13: Generalised tensor voting flow diagram.

(a) *Flower Garden* sequence. (b) Segmentation of tree trunk.

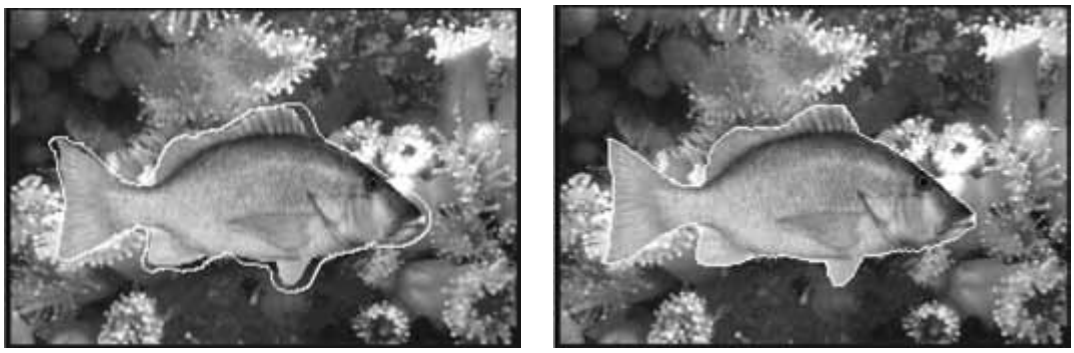
Figure 2.14: Indistinct edges found on the *Flower Garden* sequence taken from [49].

Nicolescu continued and proposes a solution by using tensor voting in the indistinct region between moving objects [50]. In this approach the described tensor voting approach is followed until the surfaces have been extracted. Knowing that the boundaries may be inexact, a refinement process is carried out by defining a horizontal zone of uncertainty centered around the detected boundary (the vertical uncertainty is dealt with separately) with the width matched to the largest correlation filter used in obtaining the initial v_x estimates. For a specific boundary point (x_c, y_c) the horizontal line is opened up, the horizontal image derivative D_x determined over the segment, and a Gaussian decay weight applied to introduce a bias towards the current estimate of the edge. The following mapping into the 2D tensor space is used:

$$\begin{aligned} e_1 &= (G_x, G_y) \qquad \text{(normal to edge)} \\ e_2 &= (-G_y, G_x) \qquad \text{(tangent to edge)} \\ \lambda_1 &= |G_x| \\ \lambda_2 &= 0, \end{aligned}$$

where the gradient in the y direction (G_y) is found by looking at adjacent rows. This represents a 2D stick voting process, where the desired geometric feature is a curve ($\lambda_1 - \lambda_2$). This process is repeated for the vertical zone of uncertainty. The maximum curve salient points are then used as seed to grow the ridges of a boundary through the uncertainty regions and assign to pixels their respective surface labels. Further refinement can be done using the surface tensor voting approach, but this time with apriori knowledge of the regions. This refines the velocity (v_x, v_y) estimates. The refinement seems to give good edge estimates, as shown in Figure 2.15.

The image gradient approach makes use of image gradients on the 2D image plane (no motion information) to refine the edges, which is a valid approach but may have difficulties in indistinct or aliased image areas.



(a) Unrefined edges. (b) Refined edges.

Figure 2.15: Refinement of edges on a fish sequence taken from [50].

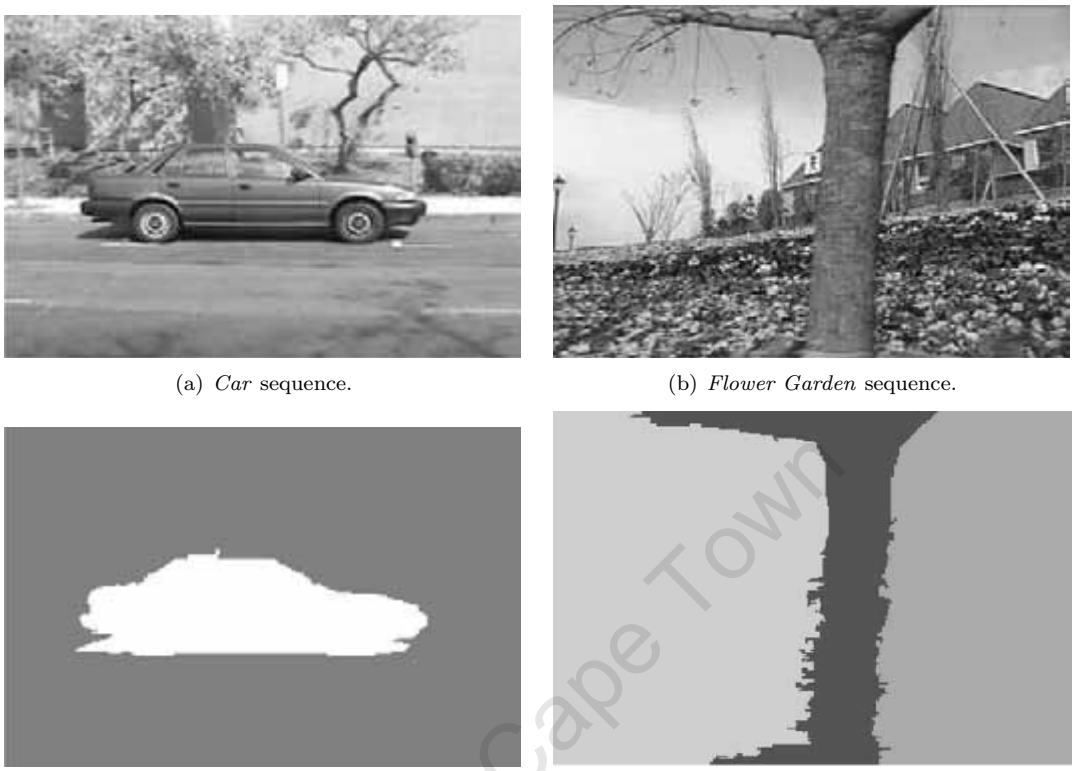
The methods by Nicolescu use two frames of a video sequence, even when more are available. Min and Medioni [45] recently propose extending the tensor voting process of Nicolescu [49] to spatio-temporal volumes, as indicated in Figure 1.1. This adds an extra dimension of time to the tensor voting problem. Although not central to the proposed formulation, cross-correlation techniques using local 2D neighborhoods is still used to determine (v_x, v_y). The matches are also limited to not have more than a single match between the adjacent frames and the reference frame due to the potential explosion of tokens resulting in a problem with too many mismatches and no clear solution.

The 5D tokens are allowed to vote in such a way that the feature sought is defined by the 3D trajectory of a test pixel point through space. Due to the fact that 5D spaces are difficult to visualize, the parametric equations of the relationship between (x, y, t, v_x, v_y) are investigated, yielding the required geometric feature of the 3D traces as a feature with 2 normal vectors $(\hat{e}_1, \hat{e}_2)$, tangential vectors $(\hat{e}_3, \hat{e}_4, \hat{e}_5)$ and saliency measure $\lambda_2 - \lambda_3$.

The formulation follows the same process given in Figure 2.13 except that the densification process is guided by an oversegmentation of the 2D reference plane image using an image based algorithm (such as watershed). Each of the oversegmented regions are checked to see whether they must be fused by checking whether the following conditions apply:

- The average of the derived velocities (v_x, v_y) along the adjoining boundaries are similar and
- The normals of (v_x, v_y) along the adjoining boundaries are similar.

The computation in a 5D space leads Min to use Graphic Processor Units to speed up the computation. The results, which can be seen in Figure 2.16, show good segmentation of the car, but fairly corrupted segmentation on the *Flower Garden* sequence. The reason for not finding the small



(a) *Car* sequence. (b) *Flower Garden* sequence.

(c) *Car* segmentation. (d) *Flower Garden* segmentation.

Figure 2.16: 5D tensor voting applied on two images taken from [45].

branches and the smearing of the trunk is due to the 2D image segmentation failing in highly complex and textured regions.

2.5 Summary

In this chapter we have outlined the current work done by other authors in the tensor voting field as applied to image and video segmentation and presented some of their results. The generalised tensor voting framework is outlined forming the basis of the continuation of the thesis and the formulation of new voting strategies. The tensor voting framework as applied to motion segmentation is described and commented on.

Chapter 3

Formulating a method of estimating motion traces

This chapter discusses and proposes methods of finding accurate traces of moving pixels in video scenes using tensor voting. Preliminary work produced rudimentary results, and this was further developed with a new kernel to extend the work into N-dimensional tokens. Several experiments with easily visualised sequences allow a foundation to be made that is extended into the N-dimensional domain. The various problems that require extension of the tensor voting framework are presented.

3.1 Introduction

In the previous chapter current methods in tensor voting as applied to motion segmentation developed by other authors were presented and commented on. The generalised tensor voting framework was presented that forms the basis of further work in this thesis.

In this chapter the motivation for using trace-based tensor voting as opposed to the surface-based tensor voting of the previous chapter is presented. An analysis of which data is important in the spatio-temporal volume is done and the importance of preserving the information in the data is discussed.

An initial 3D trace solution is formulated and presented. The results and limitations of the proposed trace solution are discussed.

A novel method of using a tangential voting kernel is presented and analysed. The kernel is extended to be a non-symmetrical tangential kernel and the effect of the kernel is presented and analysed.

In order to be able to visualise the effect of tensor voting on motion estimation, a single-dimensional synthetic *tissue earth* sequence is generated and analysed under various tensor encodings. The analysis is done both for an ideal sequence as well as for an interpolated sequence. From the single-dimensional analysis, the geometric feature of skewness is extracted and analysed.

A discussion of the relevance of the geometric feature of skewness is presented and its relationship to occlusion and disocclusion is discussed. Matched filtering techniques are proposed to be able to detect the edges of occlusion and disocclusion.

The analysis is extended into the two dimensional spatial domain, and new tensor encodings are introduced. The encodings are analysed in terms of sufficiency in representing motion vector orientation, and the occlusion and disocclusion detection capabilities verified on an ideal synthetic *tissue earth* sequence.

Lastly, the problem of high dimensionality Monte Carlo analysis is looked into and a simplified stick vote solution is proposed that can be relevant to high dimensional problems.

3.2 Motivation for using motion traces

3.2.1 Introduction

In the introduction, numerous results of relevant work have shown a few areas where improvement in segmentation of video sequences can be effected. All of these works are aligned to the tensor voting framework and use standard image processing techniques in transforming the image into tokens. This is a crucial step in trying to preserve information that is later used in the voting process. Suggestions and formulations are made as how to keep as much data as possible in the preparation of the tokens.

3.2.2 Where is the information?

In order to maximally utilize the data in a video sequence one needs to look at where the moving information is held in the sequence.

Fundamentally, if one only has one frame, one would not be able to achieve any moving object segmentation at all unless one has a clear concept of the objects being looked at. An example is if there is a car in the sequence and it is on a road, it is probably moving. For an algorithm to know this it is necessary to classify and assign characteristics to objects, which is not possible with low level vision algorithms.

Most algorithms use two or three frames to determine moving objects. This provides a degree of information, but if there is aliasing or if there are boundaries of homogeneous colour, the algorithms will either need to guess the boundaries in these regions or fail.

If one is able to look at a full video clip then much more information can be utilised to find moving objects. There are still problems concerning occlusions and objects entering and leaving the video scene, and these need to be dealt with using higher level vision algorithms. From this discussion, it makes sense to use a spatio-temporal volume to represent the video sequence.

One of the fundamental constraints on moving object segmentation is the *optical flow* constraint, which is represented as

$$(\nabla \mathbf{g})^\top \mathbf{v} = 0 \tag{3.2.1}$$

where $\nabla \mathbf{g} = (\frac{\partial g}{\partial x}\,,\, \frac{\partial g}{\partial y}\,,\, \frac{\partial g}{\partial t})^\top$ is the gradient of the intensity $g(x, y, t)$ at a 3D point (x, y, t), and $\mathbf{v} = (v_x\,,\, v_y\,,\, v_t)^\top$ and represents the 3D flow or velocities. The optical flow constraint is based on

- *Constant illumination.* In the tensor voting framework, the algorithm looks for a geometric

feature that is smooth or slowly changing. This relaxes the constant illumination limitation somewhat and also frees it from the rigid body formulations such as affine rigid body motion assumptions.

- *Lambertian reflectivity.* This assumption is generally true for natural video. This thesis will not cover highly specular objects.
- *Brightness variation only due to motion.* In a video scene the lighting will rarely change. It will probably change from one scene to the next. If this was the case, the video clip would be highly disturbing to view.

The optical flow constraint is used in most low-level vision algorithms used in estimating motion in image sequences. Correlation techniques using block matching make direct use of optical flow from one frame to the next. Most techniques use a region of support $\mathcal{R}$ consisting typically of pixels in the vicinity of the reference pixel or group of pixels. The region of support has the following characteristics:

- *Large region of support* $\mathcal{R}$. If the motion estimation of a reference pixel is determined by the surrounding data, the measurement is integrating the surrounding data in an effort to robustify the estimate. If the region of support is within the moving object, the estimate of motion is good. If the region of support straddles a boundary, the estimate is corrupted due to several different motion behaviors within the region.
- *Small region of support* $\mathcal{R}$. When using a small region of support, the robustness of the motion estimate is compromised. Areas with aliasing and featureless areas can result in erroneous estimates. Due to the low level of integration, the intrinsic errors are not well suppressed. With a small region of support, the erroneous estimates on boundaries become fewer.
- *Directional region of support* $\mathcal{R}$. In order to change the region of support as a potential motion boundary is included, various methods of distorting the region of support to try and stay within a moving object can be used [17, 15]. This potentially has the problem that knowledge of the solution is needed prior to the algorithm being applied.

In terms of tensor voting tokens, the approach in the 4D case [49] has been to use several different sizes of the region of support $\mathcal{R}$. The motion estimates from all of these are encoded as tokens, and the tensor voting process eliminates the incorrect candidates — but the edges of objects still need to be handled in a different way [50].

Spatial data

It is important to note that motion $(v_{x_{ref}}, v_{y_{ref}})$ of a reference pixel (x_{ref}, y_{ref}) cannot be inferred by only looking at the reference frame raw image data. Motion information is contained in differences between the image frames in a video sequence. There are several "clues" as to what may be moving, such as high contrast boundaries, but they themselves do not show that the object is moving. Normal motion estimation techniques exploit the differences between frames by using techniques to minimise the difference between two regions where there is a displacement (v_x, v_y) between these regions of support. The operation of differentiation or subtraction destroys part of the image data information by removing the absolute image data and reducing it to only the difference. In most cases, such as block matching, this reduction in information is acceptable due to the region of support being large enough for integration to allow good motion estimates.

It would be better to retain the image data, and tensor voting provides an avenue in that the raw data value may be utilised, and not the difference data that has currently been employed by [49, 50, 45].

In block matching techniques, a *matching search region* is also defined within which to look for correspondences. The size of the region in (x,y) is determined by:

- *Frame rate*. The frame rate determines the scale of the t axis. For low frame rates, the inter-frame t increases. This would effectively increase the *matching search region* in (x,y). Higher frame rates will decrease it and reduce the scale.
- *Expected dynamics*. If a highly dynamic video clip, such as motor racing, is being used, the matching search region will also increase.

For block matching, the matching search region is defined as a square and is defined in [2]. A more rigorous estimate would be a circle, due to the apriori unknown direction of movement. Extending this concept into the spatio-temporal volume, the matching search region can be described as a *matching cone volume*, which describes an uncertainty region emanating from the reference frame into the preceding and succeeding frames. The cone diameter increases at the rate of the matching search region as the frame becomes more distant from the reference frame, as shown in Figure 3.1.

What has been described is a model for the selection of tokens in (x,y,t) for a given $(x_{ref}, y_{ref}, t_{ref})$. In terms of tensor voting, this implies that given a *votee* token, the *voter* tokens can only be selected if they comply with the model.

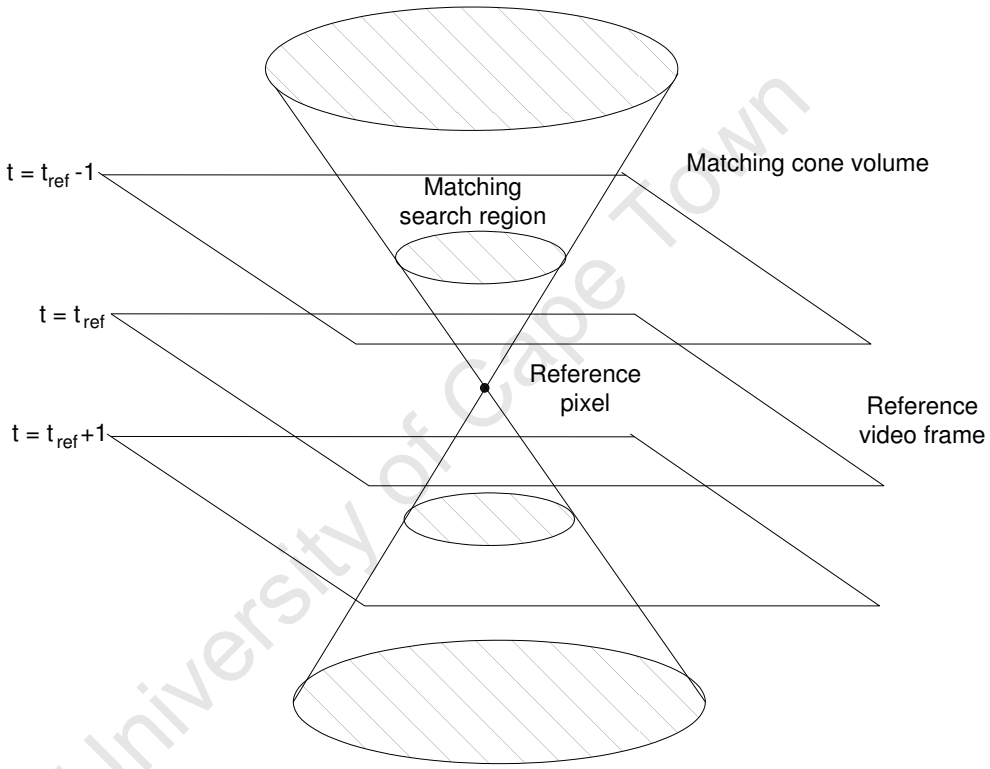


Figure 3.1: Representation of a *matching cone volume*.

Pixel information

Each pixel in an image or image sequence has an intensity which can either be a scalar (gray-valued image) where $n = 1$ or a vector where $n = 3$. The intensity represents an *attribute* of a specific spatial position (x, y, t). The characteristics of this attribute are directly used by the optical flow constraint, in trying to keep the differential of this attribute close to zero. In the current implementations of tensor voting, correlation techniques (differencing methods) are used to derive (v_x, v_y). If the colour attribute is used directly, the data will be used directly. Caution must be exercised as the scale of the pixel attribute may be different from the scale of the spatial coordinates.

Colour information is normally given as:

- *Gray scale.* This is a single intensity value. Video sequences are rarely gray-scale, but this formulation is useful in developing theories.
- *RGB Colour.* RGB colour has 3 components (R, G, B) and is the usual representation in computing and images. Using RGB values is often the preferred format, as it is the representation space of colour images.
- *YCbCr Colour.* YCbCr colour has 3 components (Y, Cb, Cr) and is the usual representation in video processing based on strong edges such as JPEG. The first component (Y) denotes the grey-scale value which contains a lot of the edge information. The Cb component holds the blue difference component, and the Cr component holds the red difference component. These two components are referred to as the chrominance components and are usually sub-sampled to reduce the amount of data required to represent an image. The $YCbCr$ colour space is often used, as it is the measurement space of colour sensors and is adapted to the human visual system. By remaining in the sensor colour space, the noise characteristics will not be disturbed.
- *CIELAB Colour.* CIELAB colour has 3 components (L, a, b) described by Chen [6]. These components have been matched to the human perception of colour. As in YCbCr, images would need to be transformed into this space, and the noise characteristics of the colour channels will be disturbed.

Using colour attributes, obtaining the data inferring motion would still mean matching similar colours from different spatial positions. If the colours and spatial properties form smooth geometric features, then they may be related to each other.

Colour information in the image sequences is perceptually the information used in natural images to discern objects and movement. The importance of this information is crucial and some characteristics

are noted:

- *Complete.* The pixel information presents a smooth and complete representation of the information. If some pixels are missing or are corrupt, the video sequence would be deemed as damaged. The focus of the tensor voting is not in this case used for in-painting and image repair, although it has been used for this application in other works [27].
- *Non-jarring.* The representation of pixel flow is smooth. There may be times where an abrupt change of direction is observed, such as when a ball bounces off a surface, but this is not true for the greater portion of time in a video sequence.
- *Stationary noise.* The noise superimposed on the image due to non-ideal sensors remains constant during the video sequence. If we assume the noise to be roughly Gaussian over one or two pixels, it becomes an advantage as it prevents abrupt gradients in the image and is the basis of many non tensor voting motion estimation techniques, such as an adaptive structure tensor flow estimation algorithm by Liu [38].

The implication of these points is that colour is a very important data source, and that the decay in importance when colour matching is used to determine motion should follow a gradual decay. As a first-order estimate, the decay is deemed as a Gaussian decay in line with the tensor voting *kernel*.

Consolidating the data

A token possesses spatial information (x, y, t) as well as attributes (R, G, B) and is scaled to form the first-order tensor $t_i = (k_x x_i, k_y y_i, k_t t_i, k_R R_i, k_G G_i, k_B B_i)$. The scale factors affect the *tensor communication* in defining the importance of the various components of the tensor on the voting process. Generally the scale in the x and y direction is the same as the image has a square aspect grid in x and y, meaning that $k_x = k_y$. The same argument applies to R, G and B, meaning that $k_R = k_G = k_B$. The number of scales can therefore be reduced to k_t, $k_{xy} = k_x = k_y$ and $k_{RGB} = k_R = k_G = k_B$. As an initial start point, the scales are all set to one although this is not necessarily optimal. Normally the colour information occupies the space of $R, G, B \in [0, 255]$ giving colour a higher weighting in the tensor voting than the spatial coordinates. Note that the inclusion of a scale on the components does not destroy any information as it is a linear operator.

3.2.3 How is the data grouped?

We need to determine which data is related to other data. The data can be grouped according to:

- *Attribute.* The colour of a pixel as seen over several frames and its relation to the optical flow constraint.
- *Position.* The data forms traces over time which is contained within the *matching cone volume.*
- *Regions.* Data over discrete regions moves similarly and smoothly when the data is within an object.

In the spatio-temporal volume, the optical flow constraint requires that the pixels (x_i, y_i) representing a point in the image on the reference plane, are represented as points in preceding and succeeding frames (other time instants) given that no occlusion or disocclusion occurs. These traces can be thought of as smoothly changing trajectories that change according to the object motion velocities (v_x, v_y), which are also functions of time. These are referred to as *fiber bundles* in [45]. Tensor voting is well suited to the problem of smoothly changing geometric features, such as these traces, and does not require any further assumption about the rigidity of motion or apply any models, such as affine, on the movement.

Once some estimate of motion (v_x, v_y) has been inferred from traces, data adjacent to each other allows regions to be grouped. This is a second step in solving the motion estimation problem. The regions must not contain boundaries as this will corrupt region motion estimation.

3.3 Initial trace solution

An early attempt at locating motion traces tries to localise high contrast points found using corner measures such as the SUSAN corner detector [57, 56] or the Harris detector [23]. The first-order tensor tokens are made up as $\mathbf{P}_i = (x_i, y_i, t_i)$. The sparse selection of these points includes any measure of motion (v_x, v_y) that is found using block matching techniques with sub-pixel refinement. The positions are given by P_i, and the direction of the tensor is determined from (v_{x_i}, v_{y_i}) encoded as a vector E_i with $v_{t_i} = 1$. This vector is a tangential vector in the direction of motion.

A voter is determined at point P and a votee at point Q (both elements from P_i), the respective tangential unit vectors are $\hat{\mathbf{v}}$ and $\hat{\mathbf{u}}$ (both orthogonal unit vector elements from E_i). The whole axis system is shifted such that P is at the origin. A vector $\mathbf{b} = \mathbf{p} - \mathbf{q}$ now describes the votee position. The tangential vector $\mathbf{v}$ is converted into the normal form using $\mathbf{n}_P = (\mathbf{v} \times \mathbf{b}) \times \hat{\mathbf{v}}$. Using the normal unit vector $\hat{\mathbf{n}}_P$, the normal vote strength at the votee point Q can be computed according to Equation 2.3.7. The stick vote becomes:

$$\mathbf{V}_{stick} = VS^2_{stick}\mathbf{v}\mathbf{v}^\top. \tag{3.3.1}$$

In this formulation, all votes are collected per votee and summed, including the votee voting for itself (perfectly aligned — zero distance in the kernel). The eigenvectors and eigenvalues are found for all tensor elements $\mathbf{P}_i$. In this formulation, with all the vectors $\mathbf{v}$ pointing in the same direction, the saliency sought is a high line saliency $(\lambda_1 - \lambda_2)$, and the direction given by $\hat{e}_1$. The maximum saliency is found over all i, and used to set a threshold of 1/3 of the maximum saliency. Any votee locations found to have saliencies below this value are discarded (noise rejection), and the rest have their E_i replaced with $\hat{e}_1$ for each i. This forms the new improved 3D set of directions E_{ref_i}.

A computer simulation in MATLAB was written to process a synthetic image created using the *tissue* image as shown in Figure 3.3. The central disk section of this image was rotated from frame to frame, while the rest was held static. For the synthetic image, the exact ground truth was determined for comparative results and for each P_i an actual E_{actual_i} was determined.

The synthetic image was processed with the control point extraction, block matching, and tensor voting, to get a refined set E_{ref_i} for each P_i that was not rejected as having a saliency that was too low. Comparisons were made between the raw and actual and refined and actual motion vector fields as contained in $E_{\ actual_i}$. This is shown in Figure 3.4.

The method of comparison between the actual motion vectors and calculated motion vectors was to look at the dot product between the two motion vectors as given by $\alpha_{ref_i} = \arccos((u_{ref_i}, v_{ref_i}, 1) \bullet (u_{actual_i}, v_{actual_i}, 1))$. The same can be found for the raw values as $\alpha_{raw_i} = \arccos((u_{raw_i}, v_{raw_i}, 1) \bullet (u_{actual_i}, v_{\ actual_i}, 1))$. The mean and standard deviation values in degrees are given in Table 3.1

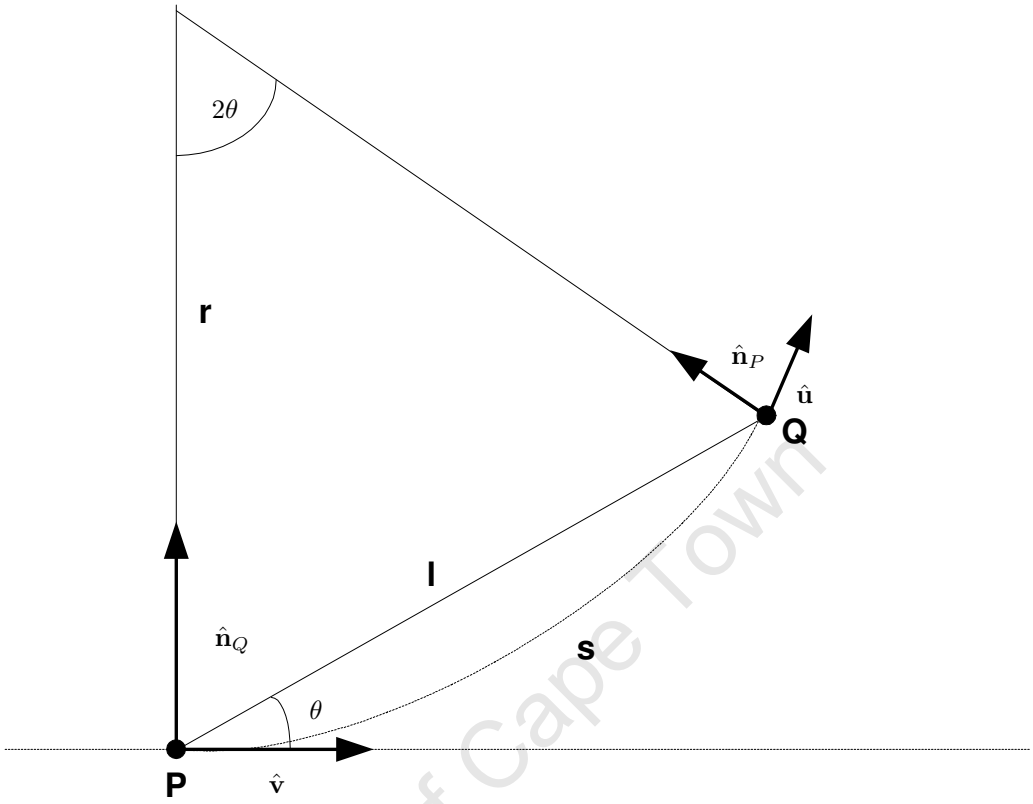


Figure 3.2: 2D stick geometry with tangential vectors.

where the parameter α indicates the weighting coefficient of the curvature excluding the distance as given later in Equation 2.3.7.

Table 3.1: Comparative angular errors (α is defined in Equation 3.4.1).

Case	Mean Error	STD Error
Raw,$\sigma = 15$,$\alpha = 1000$	10.0	9.8
Ref,$\sigma = 15$,$\alpha = 1000$	7.3	8.8

An improvement of 37% is noticed compared to the raw measurements. This method is suitable for 3D tensors, but fails for any other dimensionality due to the fact that the cross product is only defined in 3D. A generalised form of the cross product exists called a *wedge product*, which is valid in N-dimensions, but the resultant is not a vector, but a *2-vector* which is an order 2 tensor.

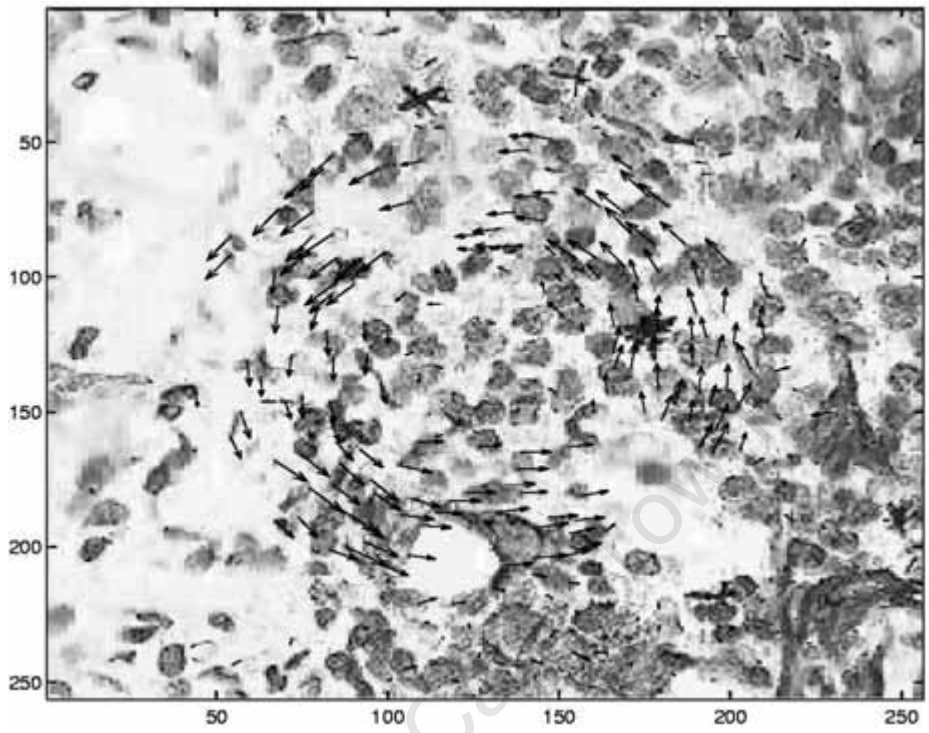
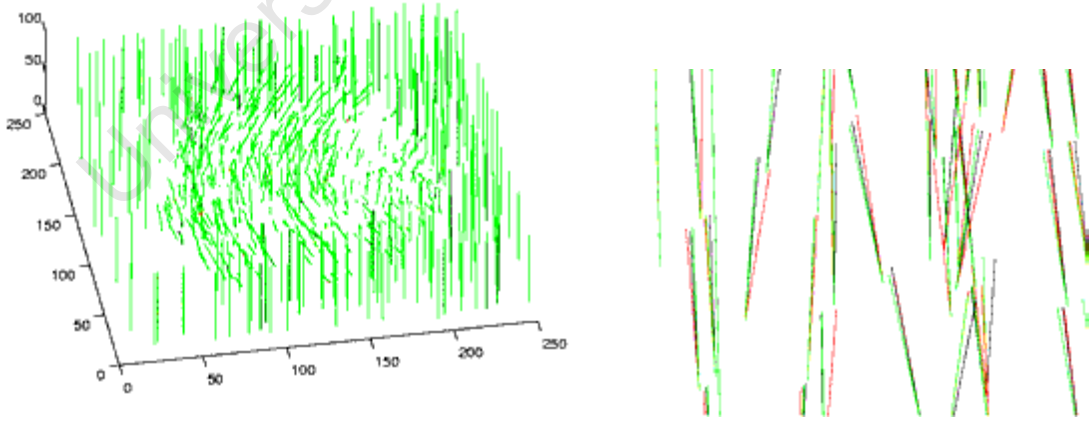


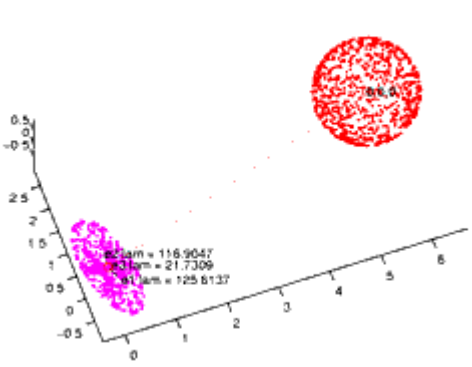
Figure 3.3: Raw Motion Flow Field with refinement used as input to P_i and E_i .



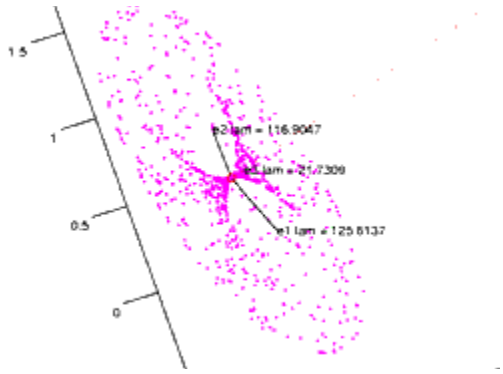
(a) Full volume traces.

(b) Detail of traces.

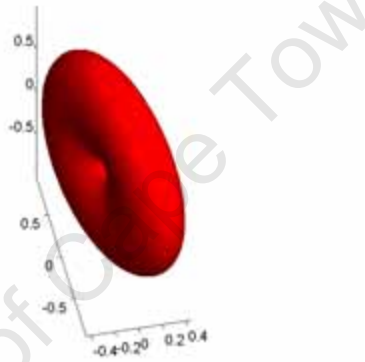
Figure 3.4: 3D traces based on *tissue* where the red traces indicate the raw measurements, black represents ground truth and green represents the tensor results.



(a) Voter (red) and votee (violet) point clouds.



(b) Votee detail indicating the directions and strengths $(\lambda_1\hat{\mathbf{e}}_1, \lambda_2\hat{\mathbf{e}}_2, \lambda_3\hat{\mathbf{e}}_3)$ of the spectral decomposition of the second-order tensors.



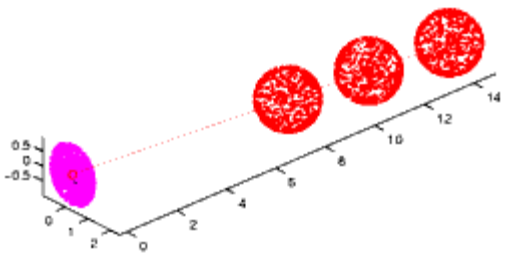
(c) Surface rendition of the cloud of votes at the votee.

Figure 3.5: 3D representations of the simulated ball voting process using the normal kernel.

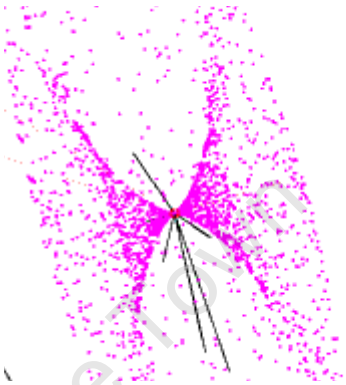
3.4 Symmetrical tangential voting in N dimensions

The initial attempt in Section 3.3 adjusted the kernel to do tangential voting by making use of a cross product, thus limiting its use to 3D. Using normal tensor voting, the stick votes are normal to the connecting vector between the voter and votee. When a ball vote is simulated, it is implemented as uniform random direction unit vectors at a voter site which then use the tensor voting field to produce a vote vector at the votee site for each voter vector. The ball vote for the normal kernel given in Section 2.3 is shown in Figure 3.5. The surface renditions in the figures do not represent actual voting tokens, but rather give an indication of the shape and structure of the underlying kernel.

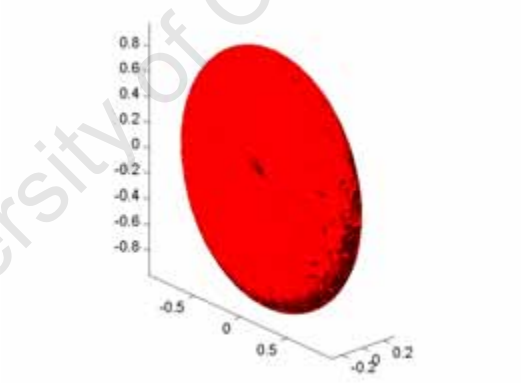
The eigenvector aligned to the *connecting line* between the voter and votee is the eigenvector $\hat{\mathbf{e}}_3$ associated with the smallest eigenvalue λ_3. The other more dominant eigenvectors ($\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$) are



(a) Voter (red) and votee (violet) point clouds.



(b) Votee detail indicating the directions and strengths $(\lambda_1\hat{\mathbf{e}}_1, \lambda_2\hat{\mathbf{e}}_2, \lambda_3\hat{\mathbf{e}}_3)$ of the spectral decomposition of the second-order tensors from three voters.



(c) Surface rendition of the cloud of votes at the votee.

Figure 3.6: 3D representations of the simulated ball voting process using the normal kernel and three voters in a line.

normal to the connecting line with the disconcerting attribute that they may be orthogonal to each other, but are not fixed or unique due to the radial symmetry of the votee cloud since they can spin around the connecting line. In Figure 3.5(b) the calculated strengths of the eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ are (125,116,21). These absolute values depend on the number of votes received by the votee. What is of greater importance is the relationship of $\lambda_1 \approx \lambda_2 \ll \lambda_3$. Repeating the voting process in a tangential mode by stringing voters in a line, the contribution at the voters at the votee can be seen in Figure 3.6. Figure 3.6(b) shows the individual spectral decompositions of the three voters separately: the only aligned eigenvector directions are $\hat{\mathbf{e}}_3$. The other two eigenvectors ($\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$) do not align themselves due to the circular symmetry. The small magnitude of the last eigenvalue indicates that errors in tensor location will have large effects on the integrity of the voting as the two dominant orthogonal components will adversely affect the result when the spectral decomposition is carried out on the combined second-order tensor.

To change the behavior of the voting, a different voting kernel needs to be formulated that has a dominant direction along the tangential direction, instead of in the normal direction as indicated in Figure 2.10. Graphically, it would simply mean rotating this plot by $\pi/2$, but still preserving the desirable radial decay and cone-like properties. This needs to be done without restricting the dimensionality as was done in Section 3.3. The method proposed is to rotate the voter vector direction by $\pi/2$. Due to the rotational symmetry we only need to deal with angles and dot products, allowing the formulation to be applicable to N dimensions.

Using the geometry given in Figure 3.7, the vote strength is given in the same form as Equation 2.3.7:

$$VS_{stick_{tan}}(s, \kappa) = \exp^{-\frac{(s^2+\alpha\kappa^2)}{\sigma^2}}, \tag{3.4.1}$$

where we define the scalar strength variables as

$$r = \frac{l}{2|\sin(\alpha)|}; \quad b = \frac{l}{2\cos(\alpha)}; \quad \kappa = \frac{1}{r}; \quad 2\theta = \pi - 2\arccos|\sin(\alpha)|; \quad s = 2r\theta. \tag{3.4.2}$$

In order to correctly define the direction such that the vector field is smooth, symmetrical and rotates in the same fashion as in Figure 2.10, we need to define the voter direction. The voter direction is defined as $\mathbf{v}$ and will result in a direction given as $\mathbf{u}$. The direction is found by determining the lower center point (defined as the intersection the two lines defined by these vectors), which is a distance b from the voter in the direction of the voter. The direction of $\mathbf{u}$ can be found as:

$$\mathbf{C} = r\hat{\mathbf{v}} + \mathbf{P}; \quad \mathbf{u} = \mathbf{Q} - \mathbf{C} \tag{3.4.3}$$

and subsequently the unit vector $\hat{\mathbf{u}}$ can be found. The vote strength and direction is plotted in Figure 3.8. The vector field is smooth and rotated $\pi/2$ radians compared to Figure 2.10.

Observing a simulated ball vote for the tangential vote as in Figure 3.9, it is noticed that the eigenvector aligned with the connecting line between the voter and votee is the eigenvector $\hat{\mathbf{e}}_1$

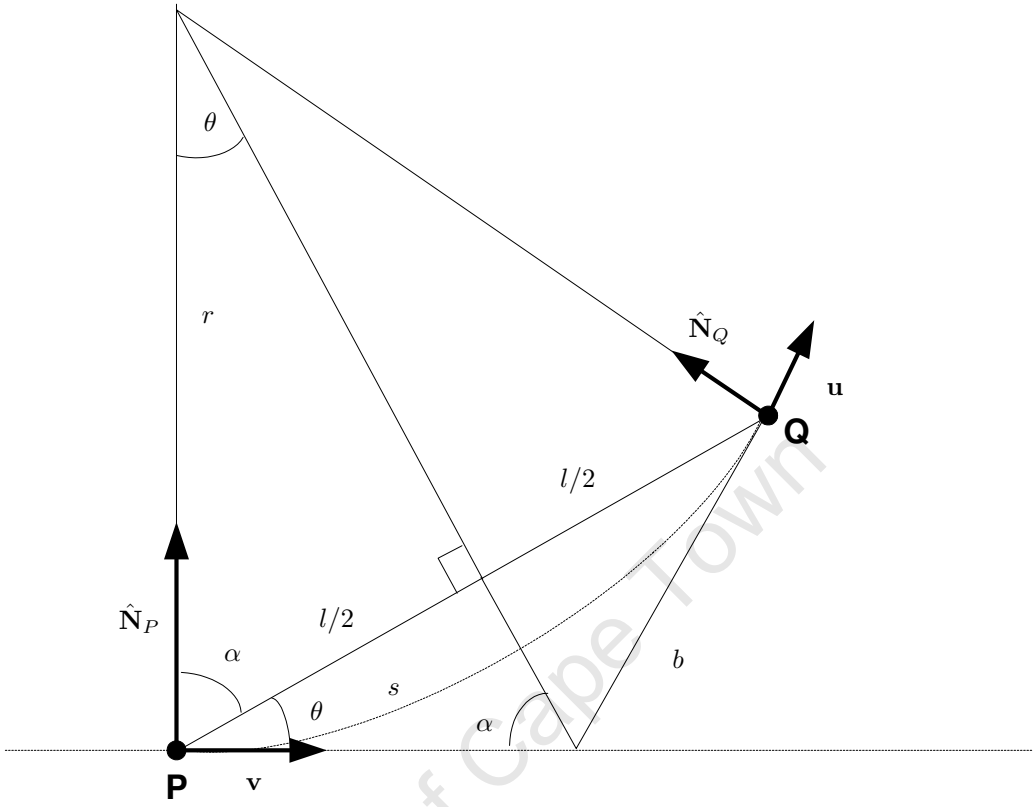


Figure 3.7: Geometry of 2D tangential stick vote.

associated with the largest eigenvalue λ_1. The other eigenvectors ($\hat{\mathbf{e}}_2$ and $\hat{\mathbf{e}}_3$) are normal to the connecting line but are much smaller that λ_1. In Figure 3.9(b) the calculated strengths of the eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ are (37,4,4). The absolute values depend on the number of votes received, so the relationship between the eigenvalues given as $\lambda_1 \ll \lambda_2 \approx \lambda_3$ better defines the geometric characteristics of the voting. Repeating the experiment where the voters are placed in a line, we obtain results given in Figure 3.10. Figure 3.10(b) shows the individual spectral decompositions of the three voters separately; the aligned eigenvector directions are $\hat{\mathbf{e}}_1$. The other two eigenvectors ($\hat{\mathbf{e}}_2$ and $\hat{\mathbf{e}}_3$) do not align themselves due to the circular symmetry but are now small in relation to λ_1 so the effects of errors in tensor location will not have large effects on the integrity of the voting. The surface rendition in Figure 3.10(c) has two lobes which is desired structure aligned to the eigenvector $\hat{\mathbf{e}}_1$ with the largest eigenvalue λ_1.

The graphical results presented in this section describe a novel method of tangential tensor voting as applied to motion trajectories that avoids using higher-order eigenvectors and eigenvalues to determine the motion vector. By using a new voting kernel, the motion trajectory is aligned to the

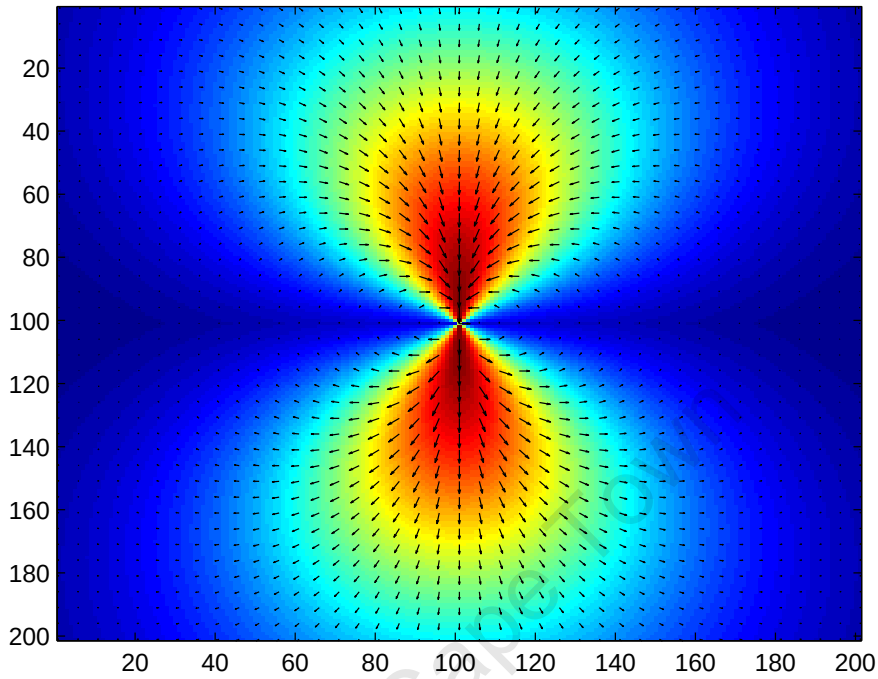
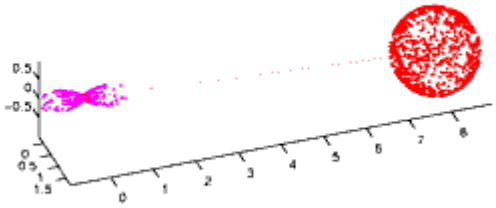


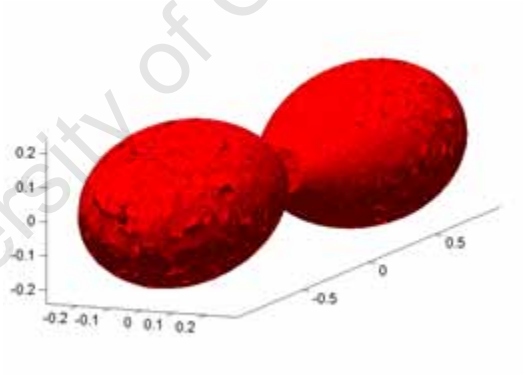
Figure 3.8: 2D $VS_{stick_{tan}}(x, y)$ showing both the strength and the vector field. Blue indicates low strength and red indicates high strength.

first eigenvector and eigenvalue which increases stability of the tangential tensor voting process.



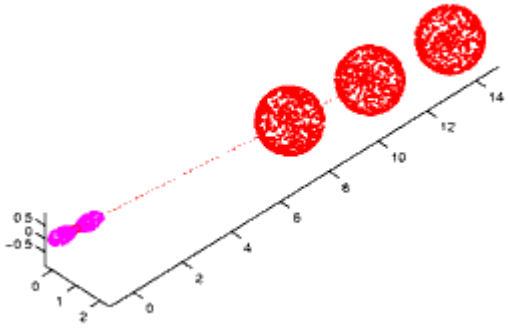
(a) Voter (red) and votee (violet) point clouds.

(b) Votee detail indicating the directions and strengths $(\lambda_1\hat{\mathbf{e}}_1, \lambda_2\hat{\mathbf{e}}_2, \lambda_3\hat{\mathbf{e}}_3)$ of the spectral decomposition of the second-order tensors.

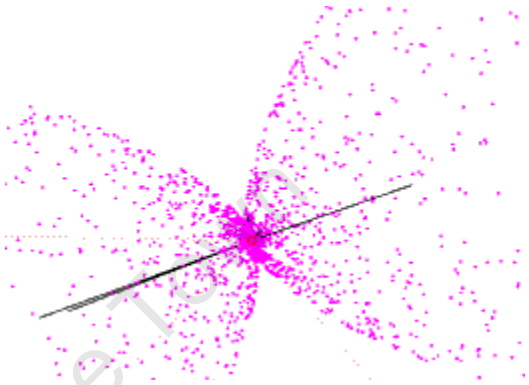


(c) Surface rendition of the cloud of votes at the votee.

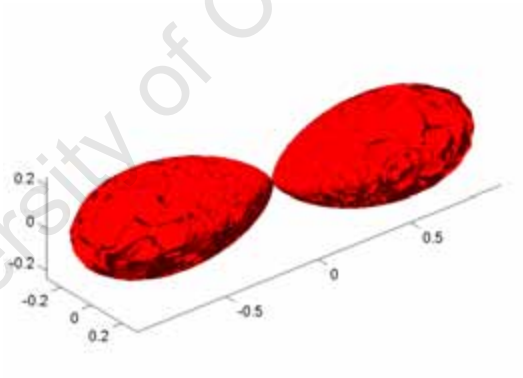
Figure 3.9: 3D representations of the simulated ball voting process using the tangential kernel.



(a) Voter (red) and votee (violet) point clouds.



(b) Votee detail indicating the directions and strengths $(\lambda_1\hat{\mathbf{e}}_1, \lambda_2\hat{\mathbf{e}}_2, \lambda_3\hat{\mathbf{e}}_3)$ of the spectral decomposition of the second-order tensors from 3 voters.



(c) Surface rendition of the cloud of votes at the votee.

Figure 3.10: 3D representations of the simulated ball voting process using the tangential kernel and three voters in a line.

3.5 Non-symmetrical tangential voting in N dimensions

A geometric feature that can have a lot of value is one that describes whether a votee is flanked by tangential voters or whether it is at the end of a tangential line. When a votee is at the end of a line, we refer to it as being an *end-cap*. This feature specifically demarcates motion boundaries without having to look at motion estimation.

In order to *skew* the votee point cloud towards a voter in a tangential way, all that is needed is to fold the symmetry around the connecting line to face away from the voter. This can be accomplished by a subtle change of the direction of the tangential voting kernel. By referring to Figure 3.7 and Equation 3.4.3, $\mathbf{u}$ can be found as:

$$\mathbf{C} = r\hat{\mathbf{v}} + \mathbf{P}; \quad \mathbf{u} = \begin{cases} \mathbf{Q} - \mathbf{C} & (\text{if } \cos(\alpha) \geq 0) \\ \mathbf{C} - \mathbf{Q} & (\text{if } \cos(\alpha) < 0). \end{cases} \tag{3.5.1}$$

The strength of the vote remains identical to the symmetric tangential stick vote strength as in Figure 3.8, but the lower hemisphere directions are all reversed. The reversal can be seen by observing the lower hemisphere directions in Figure 3.11. The directions now all flow towards the center.

The skew tangential kernel is applied to a ball vote in Figure 3.12. The shape of the votee point cloud is teardrop shaped in a direction away from the votee. The direction of the tangential eigenvector $\hat{\mathbf{e}}_1$ may point either away from or towards (as in Figure 3.12(b)) the voter, and as such only gives orientation information, not direction information. In the case of using 470 random stick votes arranged as a ball, the second-order eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ are (400,33,33) and the skewness in the direction of $\hat{\mathbf{e}}_1$ as defined in Equation 1.5.5 is -260. As a test, the symmetric tangential kernel is used under the same conditions and yields the second-order eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ of (392,33,33) and the skewness in the direction of $\hat{\mathbf{e}}_1$ as defined in Equation 1.5.5 of -14. From the small differences in eigenvalues and the large difference in skewness, it is assumed that the non-symmetrical tangential voting yields similar results in the second-order case, but the skewness measure differs considerably.

When the votee is flanked on either side by a voter as in Figure 3.13, the votee point cloud looks similar to the symmetrical case. Using 470 random stick votes arranged in a ball from each voter, the second-order eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ are (805,69,66) and the skewness in the direction of $\hat{\mathbf{e}}_1$ is 20. The skewness has dropped considerably, as would be expected due to the symmetry of the votee point cloud around $\hat{\mathbf{e}}_1$. In the definition of skewness the sign would indicate clustering on either the one projection side or the other. The eigenvector used to project the skewness is an indication of orientation, not direction. For this reason the sign on the skewness losses meaning, but the magnitude indicates the degree to which the votee is flanked in a tangential fashion.

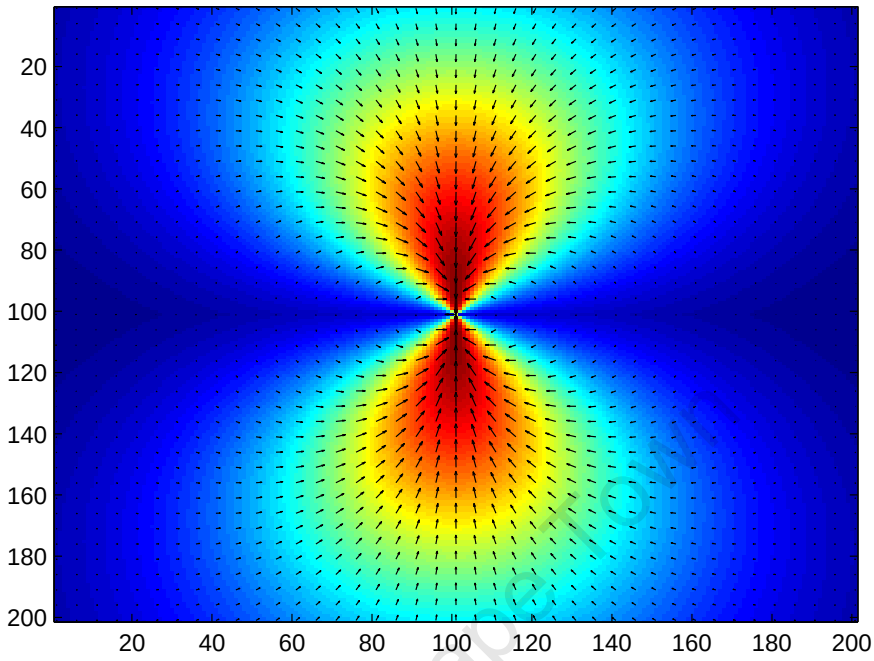
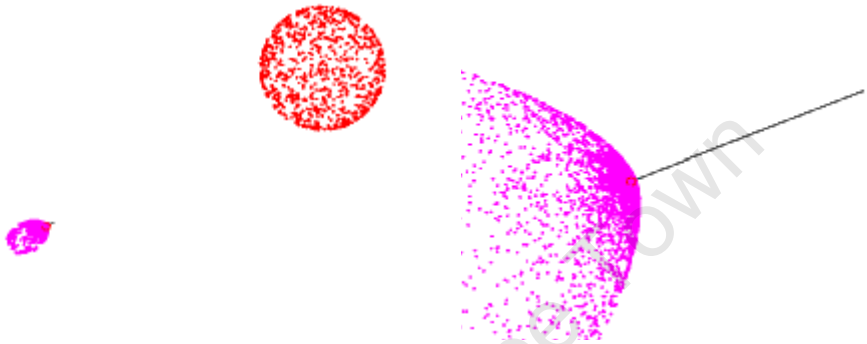


Figure 3.11: 2D 2D $VS_{stick_{skew}}(x, y)$ showing both the strength and the vector field. Blue indicates low strength and red indicates high strength.

3.6 Summary of differences between skew-tangential second-order voting and normal second-order voting

The primary reason for using a tangential representation compared to a normal representation of the second-order tensor voting process is to allow alignment of the first eigenvector with the trace of the motion of a pixel or group of pixels over a video sequence. In the normal representation, the direction of the trace is given by the third eigenvector which has a small eigenvalue associated with it which makes it more susceptible to noise as compared to the first eigenvector which has a large eigenvalue associated with it.

When the kernel produces a symmetrical cloud of votes at the votee, no polarity information is inferred even though orientation is inferred. By using a skew kernel, the polarity of the vote is indicated by the skewness of the cloud of votes at the votee. The skewness is determined by using a third-order tensor vote which has analogies to the third-order moment used in statistics to determine the skewness of a probability distribution function. The strength of the third-order tensor vote that is aligned with the first eigenvector of the second-order tensor vote determines the skewness parameter.



(a) Voter (red) and votee (violet) point clouds.

(b) votee detail indicating the directions of the spectral decomposition of the second-order tensors with the tangential direction $\hat{\mathbf{e}}_1$ visible down the axis of symmetry of the votee point cloud.

(c) Surface rendition of the cloud of votes at the votee.

Figure 3.12: 3D representations of the simulated ball voting process using the skew tangential kernel.

(a) Voter (red) and votee (violet) point clouds.

(b) votee detail indicating the directions of the spectral decomposition of the second-order tensors with the tangential direction $\hat{\mathbf{e}}_1$ visible down the axis of symmetry of the votee point cloud.

(c) Surface rendition of the cloud of votes at the votee.

Figure 3.13: 3D representations of the simulated ball voting process using the skew tangential kernel with adjacent voters in a line.

The skewness parameter rises as an edge is approached allowing for edge detection. Making use of a skew kernel does not seem to affect the properties associated with the symmetrical kernel in terms of the orientation of the first eigenvector.

3.7 Algorithm development with an ideal single dimension image

It is difficult to visualise the actions of an algorithm in a dimensionality that exceeds 3D. In order to develop the tangential voting concept further, the spatio-temporal volume is simplified by dropping one of the dimensions of the spatial term. This is done by observing a cut along a specific scan line in an image sequence, while holding all vertical motion at zero. This is difficult to achieve in a natural video sequence, so a synthetic sequence is used where two images are moved relative to each other. The *tissue* image is moved from right to left at a specific horizontal velocity v_{tissue} while the *earth* image is moved from left to right at a specific horizontal velocity v_{earth} over 50 frames. Figure 3.14 indicates the movements relative to each other. The movements are chosen as $v_{tissue} = -1$ pixels/frame and $v_{earth} = 2$ pixels/frame such that the extracted image over time remains crisp due to the synthetic movement always falling on pixel points in subsequent frames.

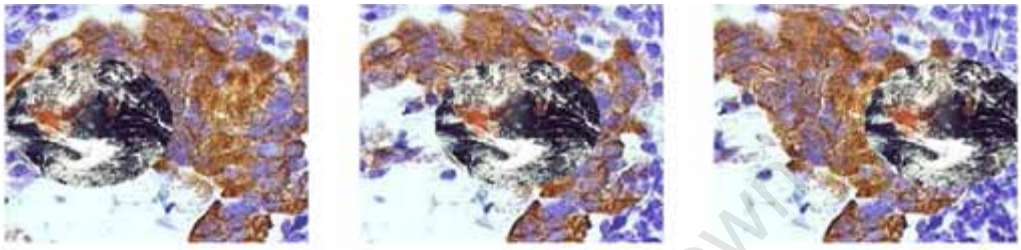
3.7.1 Second-order tangential voting

The votee points are made up of the central portion of frame 25. The sections close to the left and right hand edges are not used to prevent any biasing due to edge effects. The candidates for voters constitute all the other frames except for frame 25. In order to keep the problem tractable and implementable, we limit the number of voters per votee to 32. This also fits in neatly with the parallel computing architecture. These voters are selected to be:

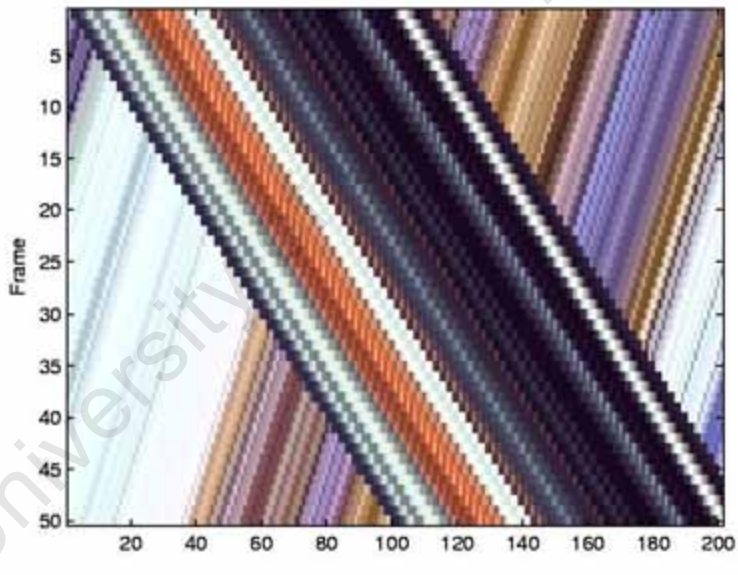
- The closest Euclidean points to the votee, and
- The points that fall within a cone of interest with a cone slope of 5 pixels/frame.

The first-order tensors are made up of $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$, which indicates that one dimensional position, frame, and colour are all used in the data. The components of the first-order tensor are chosen to be as close as possible to the available raw data and not use inferred parameters such as pixel velocities as this requires some form of flow estimation. The reduction of the 5D space to lower dimension subspaces is not consistently possible due to the changing R_i, G_i, B_i values. Tangential ball voting using the kernel described in Equation 3.4.2 is used with no alpha weighting ($\alpha = 1$) and a scale factor of $\sigma = 10$. The desired geometric feature is the feature that has a single tangential component and the rest as normal components. The saliency is therefore described by $\lambda_1 - \lambda_2$ and the direction is given by $\hat{\mathbf{e}}_1$.

To demonstrate the sensitivity of the tensor voting algorithm on a non-uniform random number generator on the unit 5-sphere S^5, the standard uniform random number generator described in

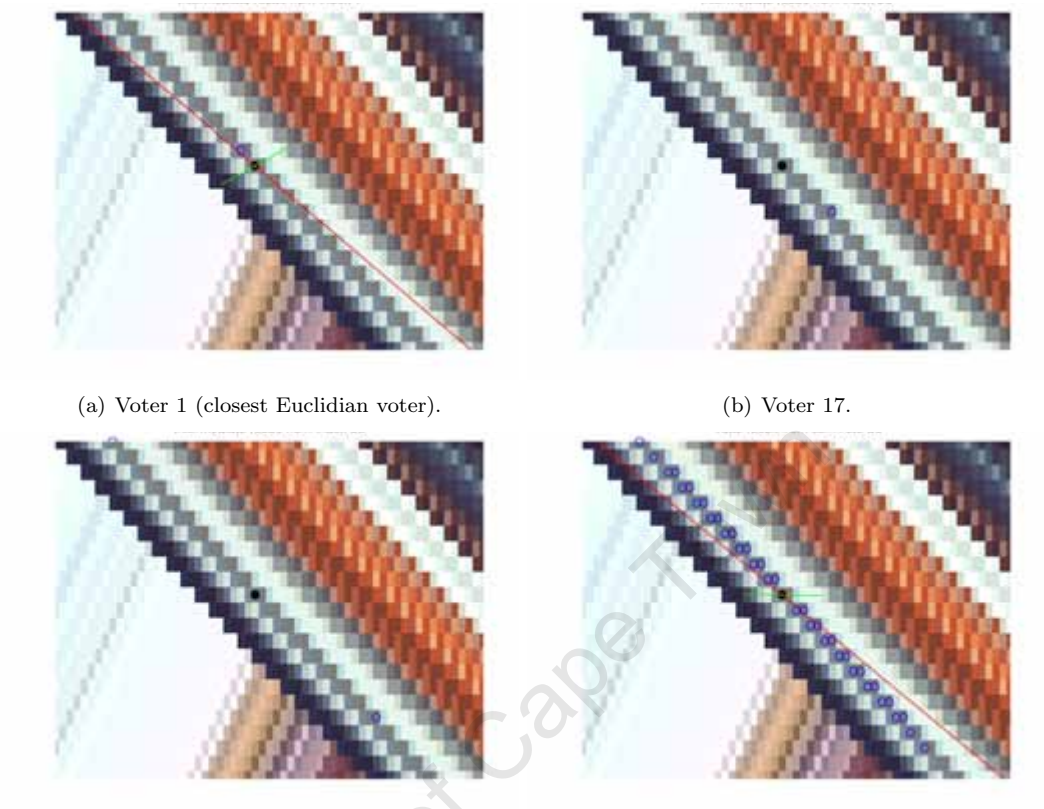


(a) frame 1. (b) frame 25. (c) frame 50.



(d) Scan line 100 extracted and plotted over columns and frames.

Figure 3.14: Single dimension movement of a synthetic *tissue earth* image sequence.



(a) Voter 1 (closest Euclidian voter).

(b) Voter 17.

(c) Voter 32 (furthest Euclidian voter).

(d) Cumulative effect of all voters.

Figure 3.15: Tangential ball vote process (using tensor representation $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$) on the votee marked as black disk with a non-uniform random number generator on the unit 5-sphere S^5. The voters are marked as blue circles, and the projection of the first two eigenvectors weighted with their respective eigenvalues are shown as red and green lines emanating from the votee respectively.

Section 1.5.5 is used with projection onto the unit sphere in Figure 3.15. It can be seen that the first eigenvector $\hat{\mathbf{e}}_3$ has a bias in direction as it should be more aligned to the voters.

By replacing the non-uniform random number generator on the unit 5-sphere S^5 with a uniform generator based on the Gaussian method described in Section 1.5.5 the results in Figure 3.16 look far better aligned, verifying the importance of correct random number generation being applied to the tensor voting problem.

The final result of 32 voters is also tested for the monochrome colour tensor $\mathbf{t}_n = (i, k, Y_i)$, the YCbCr colour space tensor $\mathbf{t}_n = (i, k, Y_i, Cb_i, Cr_i)$ and the CIELAB colour space tensor $\mathbf{t}_n = (i, k, L_i, A_i, B_i)$ in relation to the RGB colour space tensor $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$ on an individual votee level as shown in Figure 3.17. The analysis is done at this early stage to highlight the inclusion of voters in an

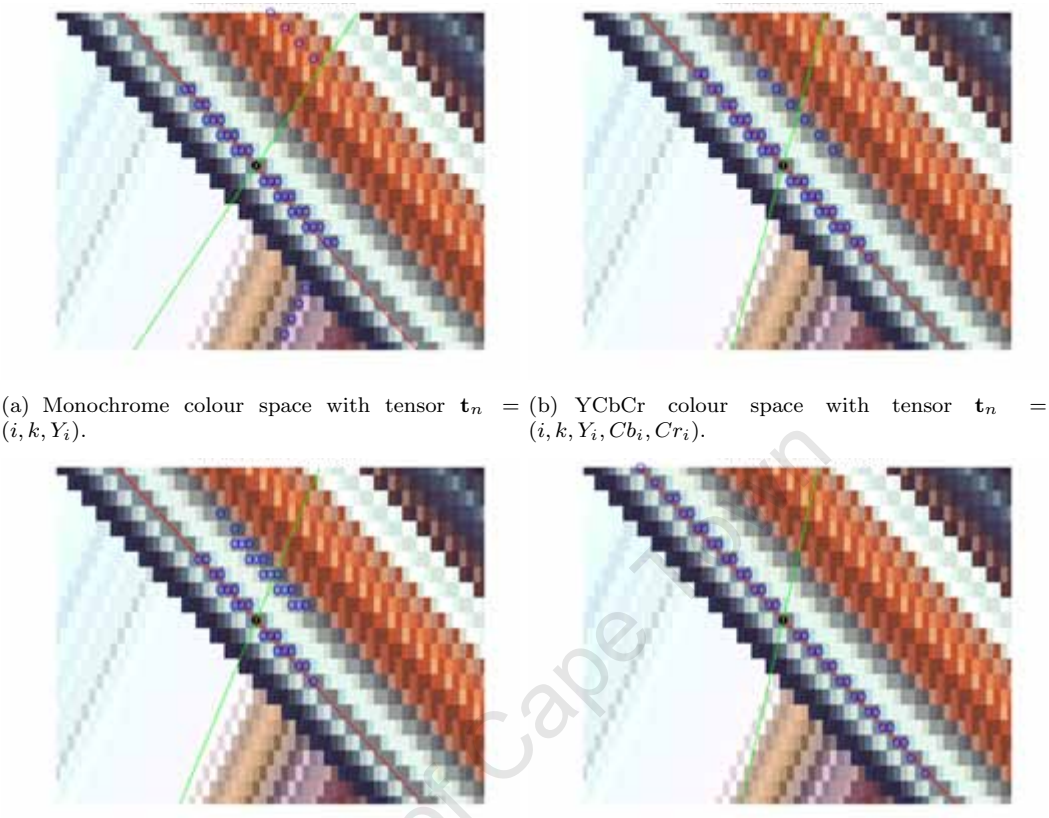
(a) Voter 1 (closest Euclidian voter).

(b) Voter 17.

(c) Voter 32 (furthest Euclidian voter).

(d) Cumulative effect of all voters.

Figure 3.16: Tangential ball vote process (using tensor representation $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$) on the votee marked as black disk with a uniform random number generator on the unit 5-sphere S^5. The voters are marked as blue circles, and the projection of the first two eigenvectors weighted with their respective eigenvalues are shown as red and green lines emanating from the votee respectively.



(a) Monochrome colour space with tensor $\mathbf{t}_n = (i, k, Y_i)$.

(b) YCbCr colour space with tensor $\mathbf{t}_n = (i, k, Y_i, Cb_i, Cr_i)$.

(c) CIELAB colour space with tensor $\mathbf{t}_n = (i, k, L_i, A_i, B_i)$.

(d) RGB colour space with tensor $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$.

Figure 3.17: Tangential ball vote process using several colour spaces on the votee marked as black disk. The voters are marked as blue circles, and the projection of the first two eigenvectors weighted with their respective eigenvalues are shown as red and green lines emanating from the votee respectively.

individual voting process that are noticeably incorrect. Care is taken that the scale of all the colour spaces remains similar to the RGB case to eliminate biases in favour of any single colour space. From the distribution of voters, it seems that the initial motivation to use the RGB colour space is valid.

Expanding on the single votee analysis on the one dimensional image as given in Figure 3.14(d), tangential ball voting is applied to more of the votees on line 25, resulting in Figure 3.18. In the figure, the saliency $\lambda_1 - \lambda_2$ is normalised to span $[0, 1]$ such that strong saliency is represented by white and low saliency is represented as black. The projection of the first eigenvector $\hat{\mathbf{e}}_1$ onto the first two dimensions (x, z) is normalised and displayed as a directionless bar indicating the frame-to-frame movement of colour, which is equivalent to v_x. A measure of correctness is visually seen by

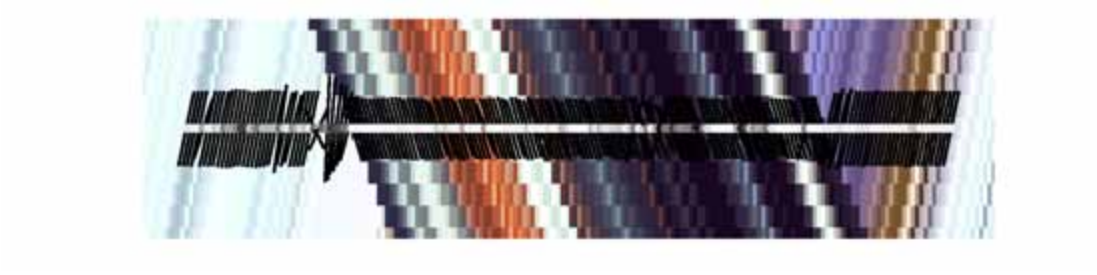
the alignment to the apparent motion in the underlying image. Where the saliency is strong (white dots), the orientations seem in good agreement with the motion. Weak saliency (darker dots) often includes definite errors in orientation. By censoring low saliency votees, a better sparse estimate of orientation is observed in Figure 3.18(d). An important feature of this vote is how close correct votes are obtained to a moving boundary as can be seen on the right hand motion boundary. The left hand motion boundary does not fare well due to the near-homogeneous colour at the boundary.

The motion estimates within the moving objects can be strengthened by formulating a dimension-11 tensor of the form $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (case 1). The tokens used for the dimension-11 tensor consist of the adjacent (left hand and right hand) pixel colour values. Choosing the adjacent pixels is based on trying to use the video frame data directly instead of using inferred values such as pixel velocities. The tokens are not orthogonal and an effort to try and find an orthogonal space using techniques such as Principal Components Analysis (PCA) or Karhunen Loeve Transform (KLT) would not be successful due to the varying correlation of the tokens over the video frame. The fact that the tensor is not orthogonal can have an effect on the basis of tensor fields being invalid. The differing tensor forms are enumerated as different *cases* later in the thesis given in Table 3.2. From Figure 3.19, the orientations are good within the boundaries of the moving object. The right hand motion boundary shows high saliency which is expected due to the general high contrast in this region, but the orientations are erroneous. The left hand motion boundary displays bleeding in that the orientation outside the boundary complies with the motion inside the boundary. This is due to the homogeneous colour on the one side of the boundary not contributing in a directional way to the tensor vote, allowing the $i + 1$ pixels to dominate.

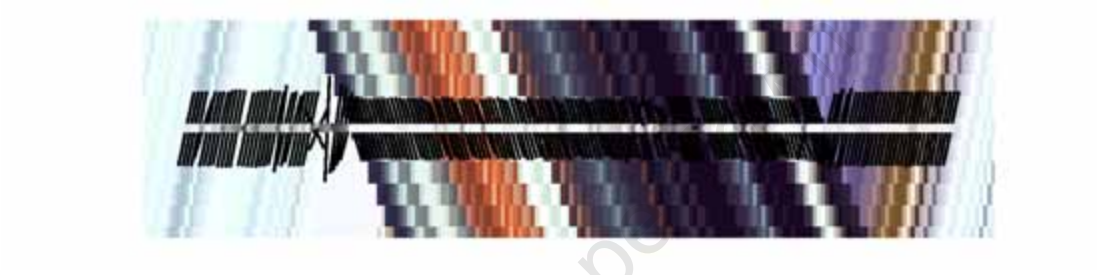
3.7.2 Third-order tangential voting

The non-symmetrical kernel of Section 3.5 is used on the dimension-5 tangential tensor vote on line 25 of the image. Using Equation 1.5.5 with the eigenvector $\hat{\mathbf{e}}_1$, the tensor skewness $\|S_{\mathbf{e}_1}\|$ is indicated in Figure 3.20. The tensor skewness graph is fairly erratic, although a correspondence to the moving object boundaries can be seen in Figure 3.20(b). The structure of the tensor skewness graph changes little when the number of voters are increased or the number of random ball vectors per voter is increased by increasing the iterations in the Monte Carlo analysis given as f. The normal variate pseudorandom number generator was compared to MATLAB's built in generator and the Box-Muller method. The Box-Muller method produced inferior results compared to the MATLAB generator, especially for the third-order tensor skewness. For this reason, the Ziggurat method [41] was substituted in the code for comparable results to MATLAB.

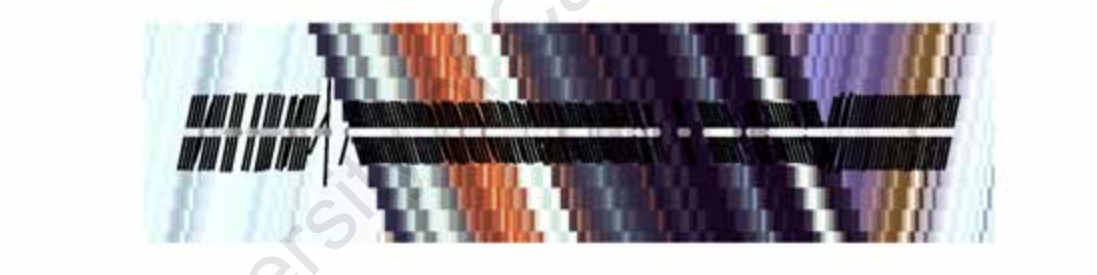
Extending the non-symmetrical kernel on the dimension-11 tensor (Case 1), similar tests are run resulting in Figure 3.21. Structure is noticeable in the skewness graph.



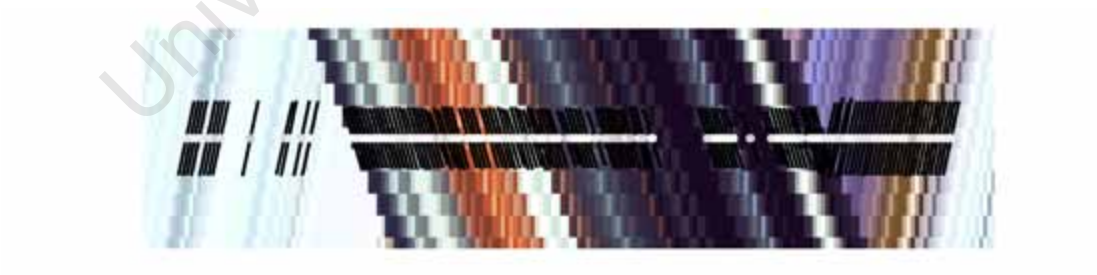
(a) Saliency threshold of 100% (no censorship).



(b) Saliency threshold of 75%.

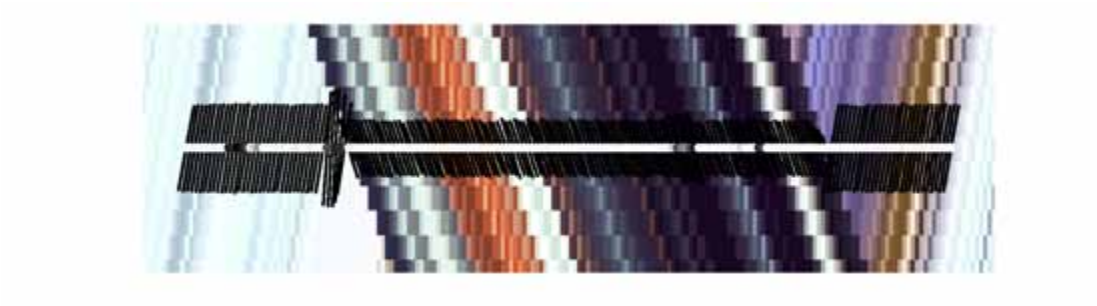


(c) Saliency threshold of 50%.

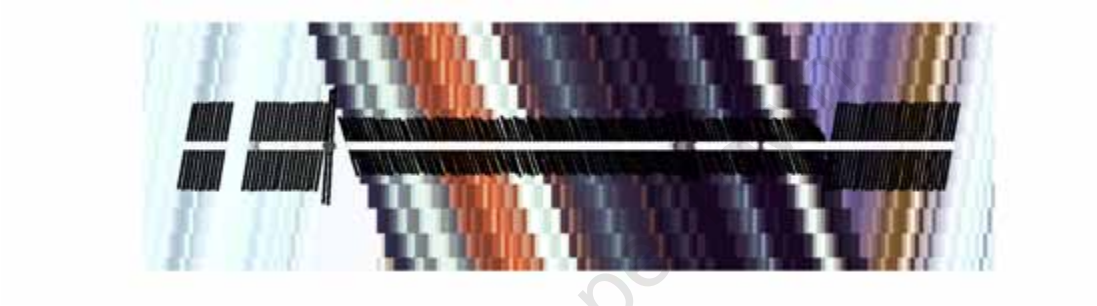


(d) Saliency threshold of 25%.

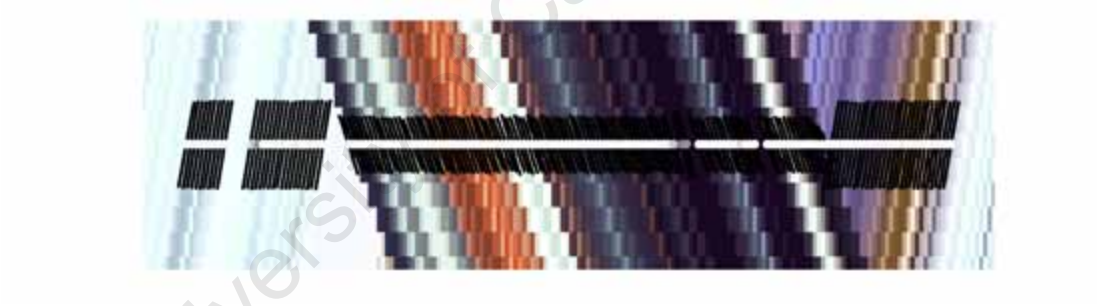
Figure 3.18: One dimensional tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$.



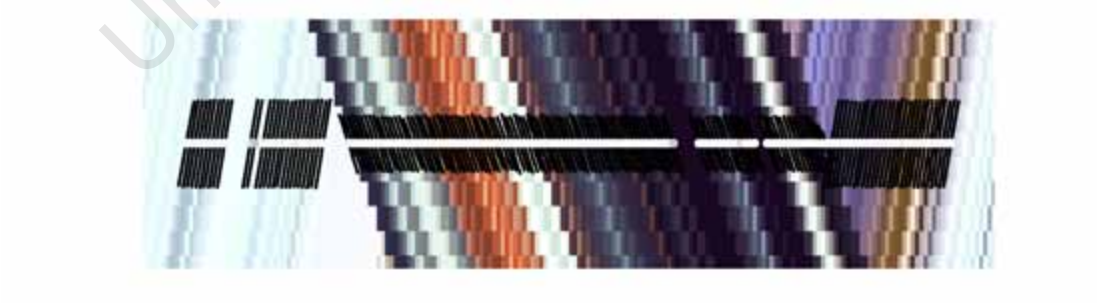
(a) Saliency threshold of 100% (no censorship).



(b) Saliency threshold of 75%.

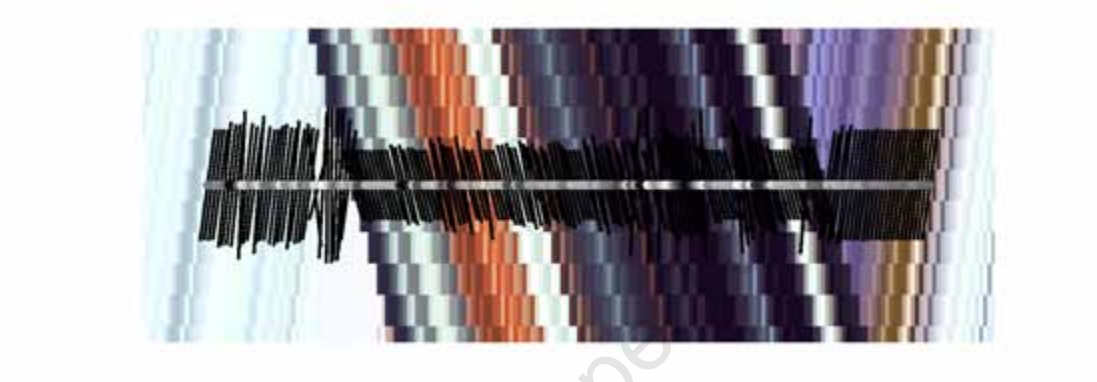


(c) Saliency threshold of 50%.

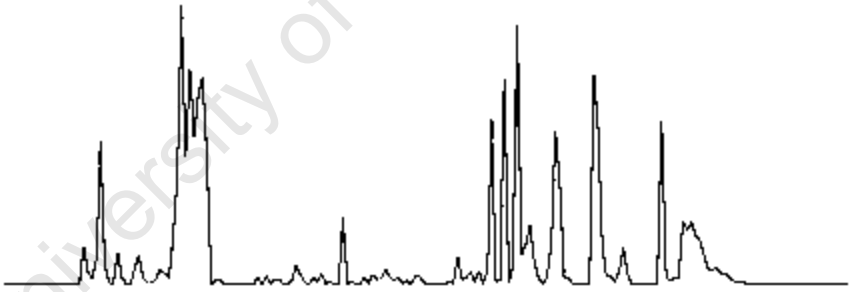


(d) Saliency threshold of 25%.

Figure 3.19: One dimensional tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (Case 1).

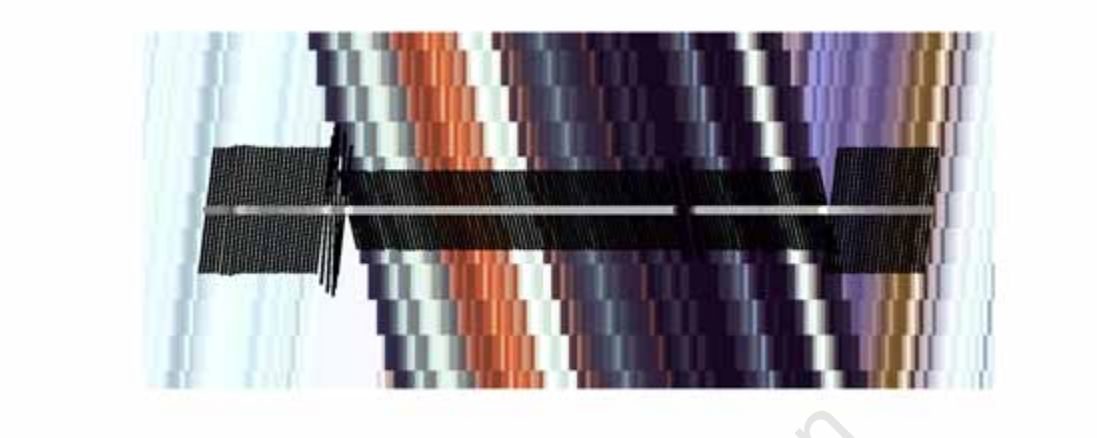


(a) Uncensored tensor vote using $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$.

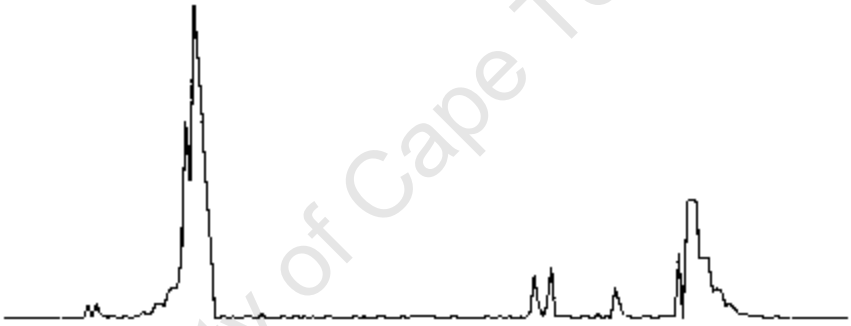


(b) Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$.

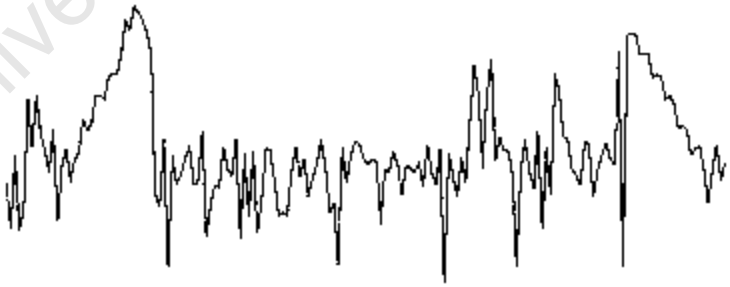
Figure 3.20: One dimensional skew tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i)$.



(a) Uncensored tensor vote using $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (Case 1).



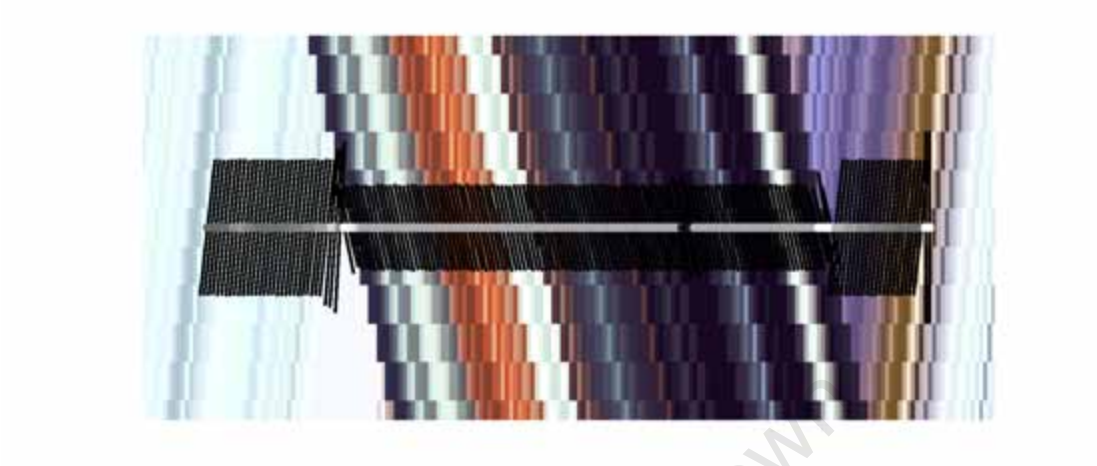
(b) Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$.



(c) Column aligned tensor skewness $\log_e(\|S_{\mathbf{e}_1}\|)$.

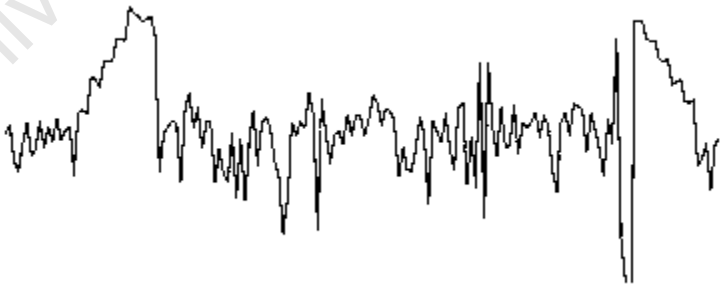
Figure 3.21: One dimensional skew tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (Case 1).

Extending the tensor further to include two adjacent pixels in the tensor representation, we get a dimension-17 tensor problem. Using two adjacent pixels horizontally maps to Case 4 given in Table 3.2. The same test is run resulting in Figure 3.22. Significant suppression of the noise results, but slight degradation in edge selectivity can be seen on the left hand edge of the tensor skewness graph. This is to be expected due to the averaging effect that the higher order tensor is causing due to adjacent pixel inclusion.



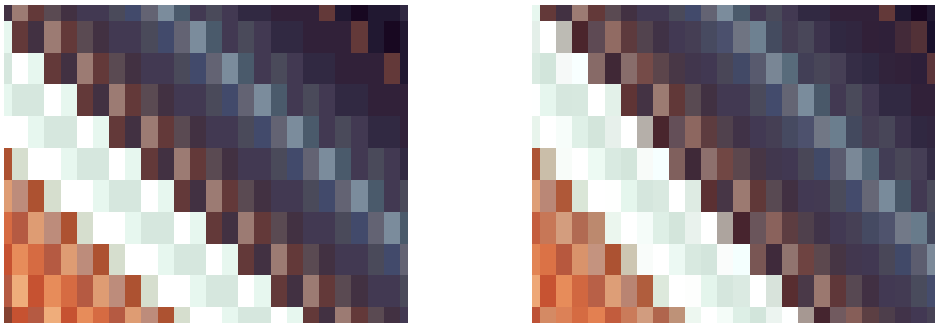
(a) Uncensored tensor vote using $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1}, R_{i-2}, G_{i-2}, B_{i-2}, R_{i+2}, G_{i+2}, B_{i+2})$ (Case 4).

(b) Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$.



(c) Column aligned tensor skewness $\log_e(\|S_{\mathbf{e}_1}\|)$.

Figure 3.22: One dimensional skew tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1}, R_{i-2}, G_{i-2}, B_{i-2}, R_{i+2}, G_{i+2}, B_{i+2})$ (Case 4).



(a) Ideal *tissue earth* image (b) Interpolated *tissue earth* image

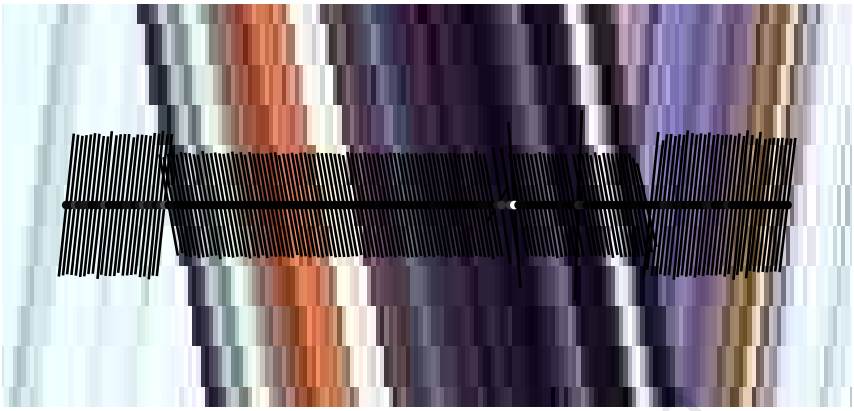
Figure 3.23: Segment of frame 25 of the synthetic *tissue earth* image sequence showing the effect of interpolation.

3.8 Algorithm development with an interpolated single dimension image

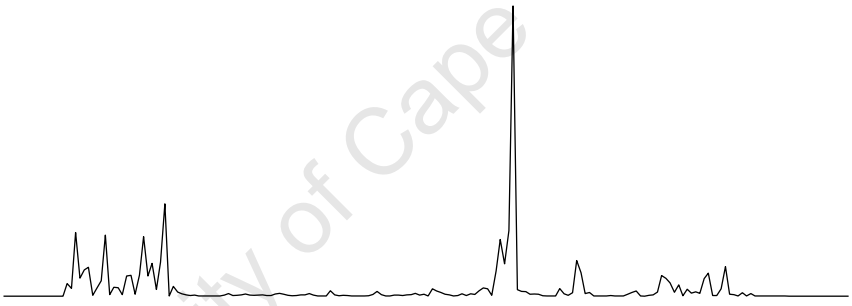
3.8.1 Introduction

In the previous section, an ideal synthetic sequence of images was used to develop the tensor voting algorithm. When the synthetic *tissue earth* image sequence is regenerated using non-integer object velocities, and the pixel values are generated using bicubic interpolation, the trace velocity lines do not intersect the discrete pixel points from image to image. In the regenerated *tissue earth* image sequence the *tissue* image is moved from right to left at a horizontal velocity $v_{tissue} = -1.12$ while the *earth* image is moved from left to right at a horizontal velocity $v_{earth} = 1.65$ over 50 frames. The effect of the interpolation on the pixel colour values can be seen in Figure 3.23 by looking down the motion traces and observing the slow blending of colours.

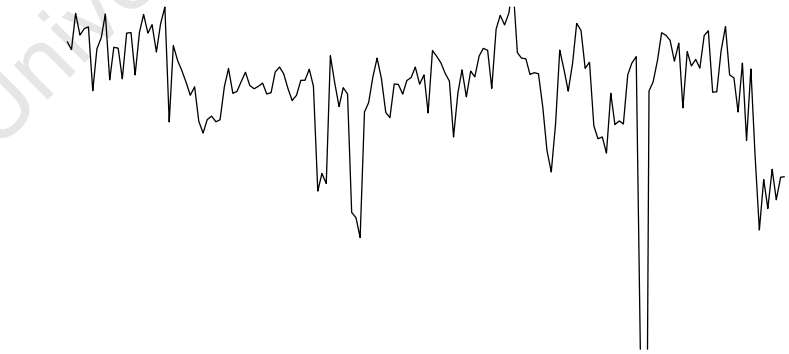
From the previous section, the best results were obtained using tensor representations of case 1 and case 4 from Table 3.2. Case 1 and case 4 are repeated with the interpolated synthetic *tissue earth* image sequence resulting in Figure 3.24 and Figure 3.25 respectively. The second-order tensor orientation is still well aligned to the underlying motion, but the skewness $\|S_{\mathbf{e}_1}\|$ does not indicate the boundaries in case 1 at all, and only vaguely represents the boundaries in case 4. The third-order tensor voting seems to be very sensitive to imperfections in the data introduced by the interpolation process.



(a) Uncensored tensor vote using $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (Case 1).

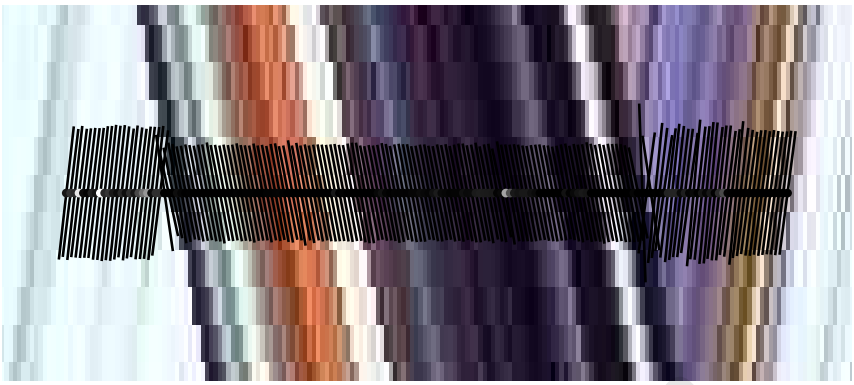


(b) Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$.

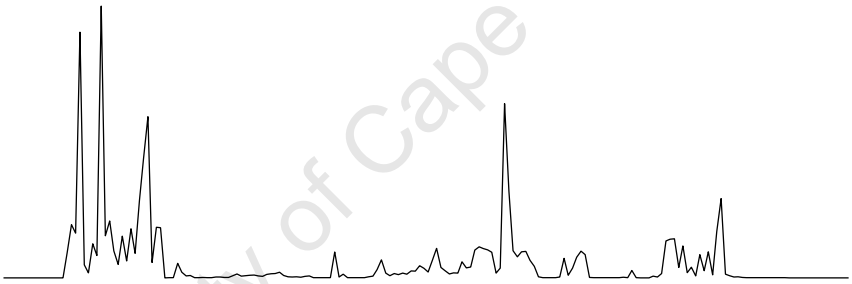


(c) Column aligned tensor skewness $\log_e(\|S_{\mathbf{e}_1}\|)$.

Figure 3.24: One dimensional skew tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1})$ (Case 1) on an interpolated image set.



(a) Uncensored tensor vote using $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1}, R_{i-2}, G_{i-2}, B_{i-2}, R_{i+2}, G_{i+2}, B_{i+2})$ (Case 4).



(b) Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$.

(c) Column aligned tensor skewness $\log_e(\|S_{\mathbf{e}_1}\|)$.

Figure 3.25: One dimensional skew tangential tensor vote using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1}, R_{i-2}, G_{i-2}, B_{i-2}, R_{i+2}, G_{i+2}, B_{i+2})$ (Case 4) on an interpolated image set.

3.8.2 Effect of varying the colour constant k_{RGB}

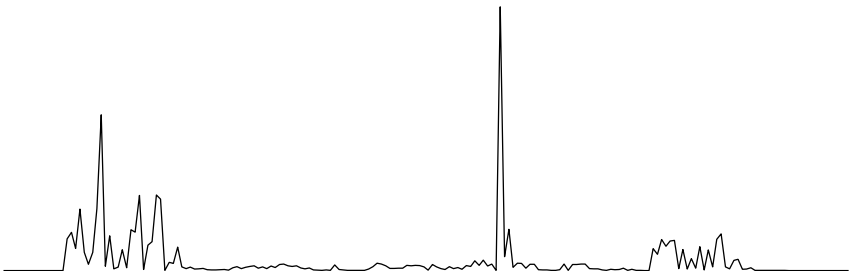
To try mitigate the imperfections in the colour RGB data, the constant k_{RGB} is reduced from $k_{RGB} = 1$ to several lower values. By reducing k_{RGB} the tensor colour data clusters closer together while the spatial x, y, t components of the tensor become more dominant in the tensor voting process. In Figure 3.26 the constant k_{RGB} is changed, and the skewness $\|S_{\mathbf{e}_1}\|$ observed. The effect of changing the constant k_{RGB} seems to allow a marginal improvement in the skewness $\|S_{\mathbf{e}_1}\|$ at $k_{RGB} = 0.25$ indicating only a minor effect on the skewness.

3.8.3 Effect of voter pre-alignment

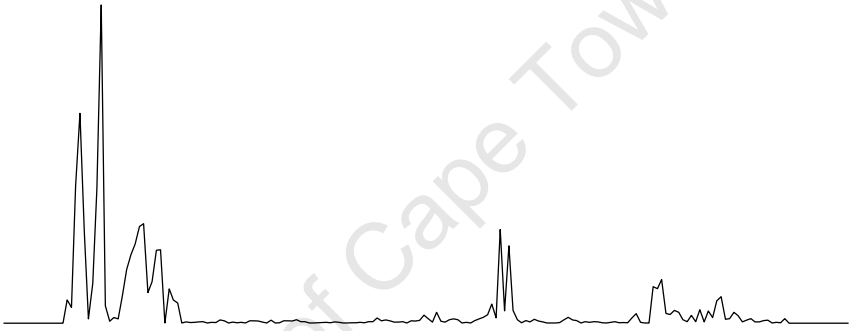
A method that can potentially increase the accuracy of the tensor voting approach is to allow two rounds of tensor voting to take place similar to the flow used by [49] and shown in Figure 2.13. In the discussion of the generalised tensor flow diagram in Figure 2.13, the process allows data points to be censored and filled in using densification. This thesis attempts not to infer information, but rather to use information held in the pixels. The two rounds of voting allow a select set of voter tensors to become votees. The tensor voting process allows the voter set to contain more orientation information. A flow diagram showing the stages of voting is shown in Figure 3.27 which makes use of the information available in the pixels, and does not infer information.

In the flow diagram shown in Figure 3.27 the various stages are described:

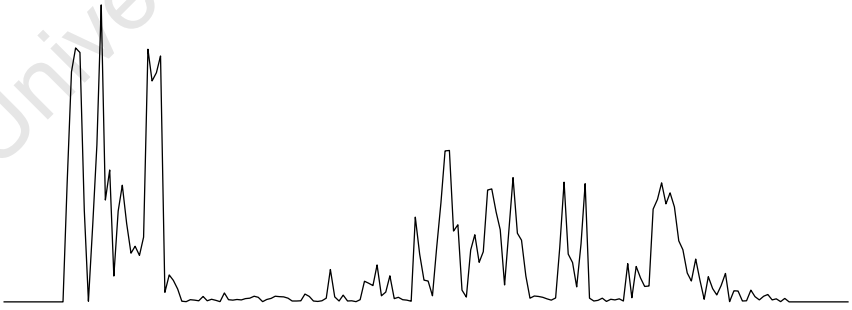
- *Selection of votee tokens.* The selection of input votee tokens is based on the requirement of the user but may include all the pixels in the regions under analysis.
- *Find m closest voter tokens per votee for all votees.* Using normal Euclidian distances between tokens (including the effects of k_t and k_{RGB}), the m closest voter pixel positions are chosen and collated into a set. The value of m is set at $m = 10$ empirically to both save on computation, but still have desired effects on the voting.
- *Use voter token set as votees.* A temporary votee set comprising of all the selected voters is compiled for tensor voting.
- *Ball tensor vote.* A ball tensor vote is carried out at all the temporary votee tokens to infer features to them. The features are held in the eigenvalues and eigenvectors at each temporary votee token.
- *Reinclude voter orientations into voter token set.* A new voter token set is made up with specific tokens containing the eigenvalues and eigenvectors to allow for orientated tensor voting.



(a) $k_{RGB} = 0.75$.



(b) $k_{RGB} = 0.5$.



(c) $k_{RGB} = 0.25$.

Figure 3.26: Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$ using a tensor representation of $\mathbf{t}_n = (i, k, R_i, G_i, B_i, R_{i-1}, G_{i-1}, B_{i-1}, R_{i+1}, G_{i+1}, B_{i+1}, R_{i-2}, G_{i-2}, B_{i-2}, R_{i+2}, G_{i+2}, B_{i+2})$ (Case 4) on an interpolated image set for different k_{RGB} values.

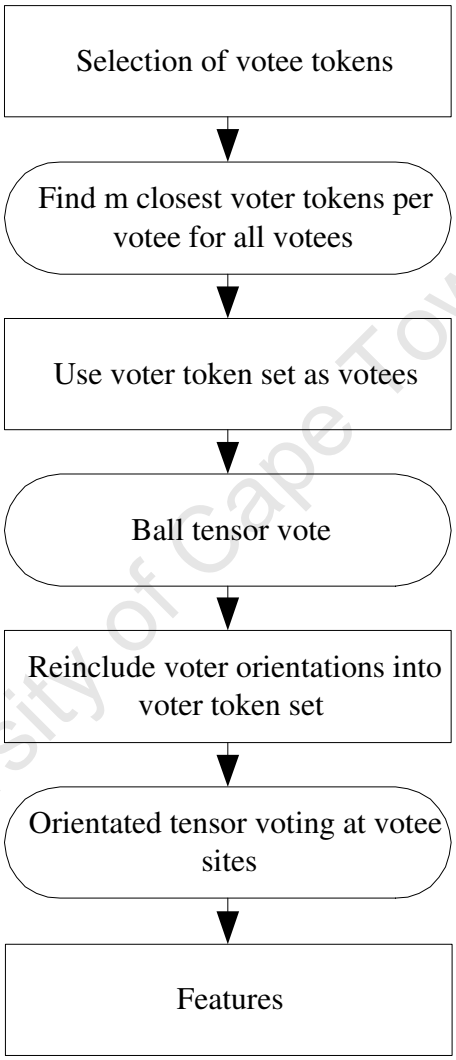


Figure 3.27: Voter pre-alignment tensor voting flow diagram.

- *Orientated tensor voting at votee sites.* The original set of votee tokens is used as votee tokens, and the new voter token set is used to vote at the votee sites. The number of orientated voters per votee does not make up the full complement of voters per votee (set at 32), so a mixture of ball and orientated votes are collected at each votee site.
- *Features.* The features such as skewness are collected at each votee site.

The selection of the closest voter tokens is based on the Euclidean distance from the original votee tensors and varies according to k_{RGB}. In Figure 3.28 the effects of the voter pre-alignment on a single votee orientation are shown in red. Also shown is the result of a standard ball vote at the votee site which is shown in green. The orientations of the pre-alignment and the ball vote are very similar. When k_{RGB} becomes small ($k_{RGB} = 0.25$), the voters start clustering around the voter with little regard to pixel colour information and the votee orientation is less aligned to the actual velocity.

In order to determine the effect on skewness, more votees are included and simulations run for several k_{RGB} values. The effect of k_{RGB} on linear skewness $\|S_{\mathbf{e}_1}\|$ is shown in Figure 3.29. The skewness using only ball voting shown in Figure 3.26 and pre-alignment ($m = 10$) of the voters shown in Figure 3.29 does not show any noticeable improvement in determining the motion boundaries as seen in Figure 3.22.

3.8.4 Skewness over spatial dimensions only

There may be an effect on skewness induced by the colour dimensions of the tensor encoding. The skewness calculation of Equation 1.5.5 is reduced by limiting the indices to the spatial dimensions (in this case 2 dimensions) and ignoring the rest:

$$S_{\mathbf{e}_i} = \sum_{j,k,l=1}^{2} \mathcal{T}_{jkl} e_{i_j} e_{i_k} e_{i_l}. \tag{3.8.1}$$

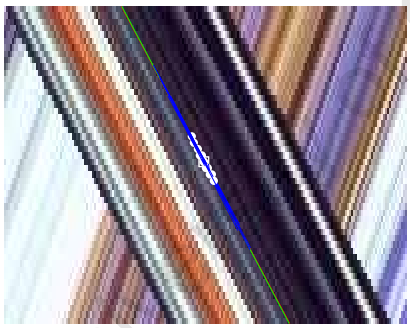
The simulation is run for both the pre-alignment of voters and the ball vote for case 4 with $k_{RGB} = 1$ in Figure 3.30. Both the pre-aligned vote and the ball vote do not show any improvement on localising the motion boundaries using the skewness measure.

3.8.5 Summary

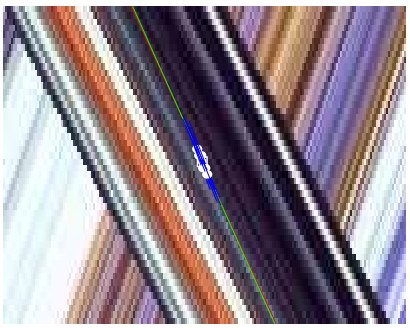
All the techniques used to mitigate the effect of interpolation in this section on the skewness $\|S_{\mathbf{e}_1}\|$ did not visually indicate the motion boundaries well.

(a) $k_{RGB} = 1$.

(b) $k_{RGB} = 0.75$.

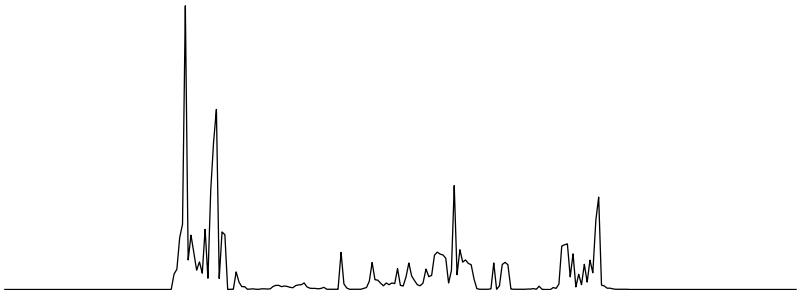


(c) $k_{RGB} = 0.5$.

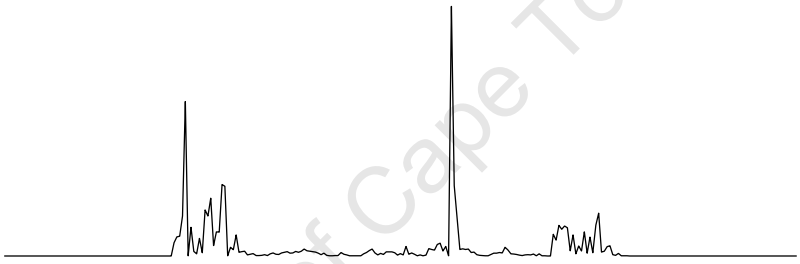


(d) $k_{RGB} = 0.25$.

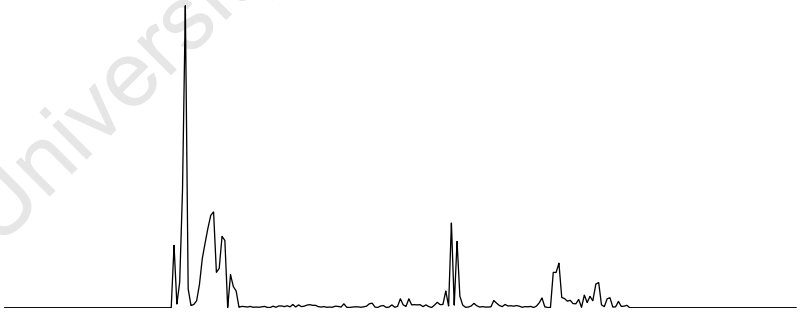
Figure 3.28: A votee (indicated with square marker) with the 10 closest Euclidean voters. The voter orientations are shown in blue, the normal ball vote (case 4) at the votee is shown in green, and the result from pre-alignment of voting at the votee (case 4) is shown in red.



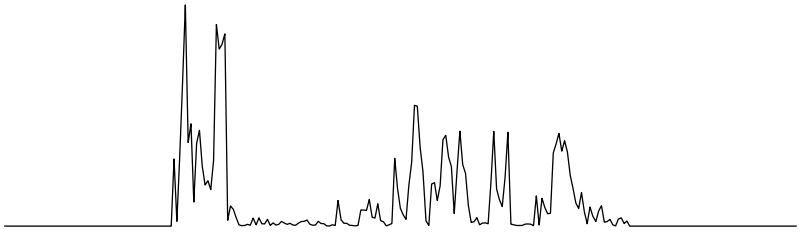
(a) $k_{RGB} = 1$.



(b) $k_{RGB} = 0.75$.

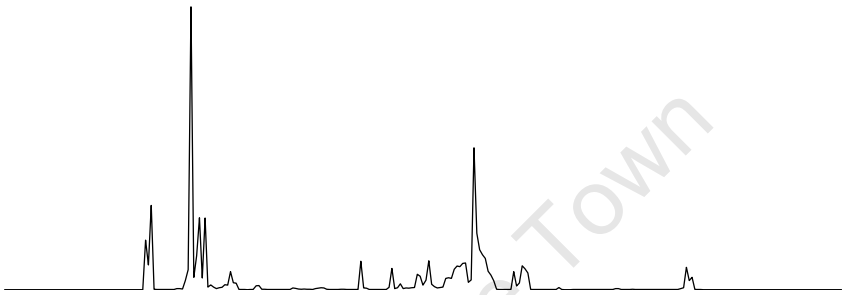


(c) $k_{RGB} = 0.5$.

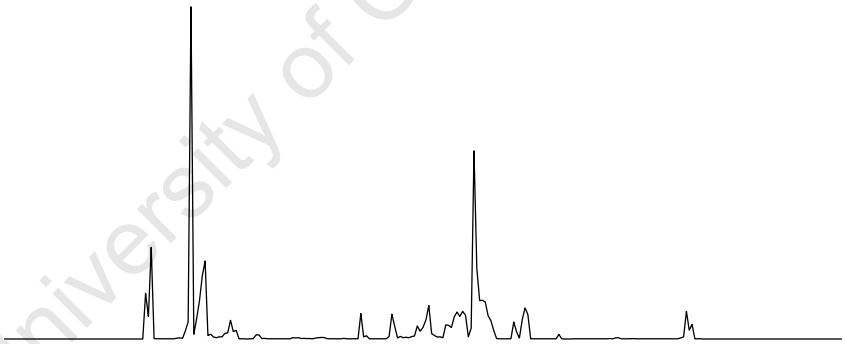


(d) $k_{RGB} = 0.25$.

Figure 3.29: Column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$ using pre-alignment ($m = 10$) tensor voting on a case 4 tensor representation of on an interpolated image set for different k_{RGB} values.



(a) Pre-alignment voters ($k_{RGB} = 1$).



(b) Ball voters ($k_{RGB} = 1$).

Figure 3.30: Case 4 column aligned tensor skewness $\|S_{\mathbf{e}_1}\|$ using only spatial dimensions in skewness calculation.

In classical block matching methods, the effects of noise and interpolation are mitigated by increasing the size of the matching region. A similar approach can be used for tensor voting by encoding more pixels into the tensor. The case 4 encoding already uses pixels that are two pixels away from the central pixel which decreases the ability to find boundaries accurately. Increasing the number of pixels used in the tensor is deferred to the two-dimensional analysis where pixels from the rows may also be used.

3.9 Skewness in tensor voting

The edge boundaries in Figure 3.21(b) display a characteristic shape due to the votee approaching a more and more skewed position which maximises on the boundary. The tensor skewness measure follows the path of the dominant eigenvector $\hat{\mathbf{e}}_1$. What is illustrated in the figures is the (x, z) projection of the dominant eigenvector $\hat{\mathbf{e}}_1$. In the case of a stationary object, the $\hat{\mathbf{e}}_1$ projection would be zero, and the characteristic tensor skewness graph would only show a large value if the analysed z was at a boundary. For this reason, the x axis analysis only provides an indication of structure in the case of moving objects.

The tensor skewness climbs as an approximate exponential e^x function towards the boundary and is only evident where the motion trace is ending (*occlusion*) or starting (*disocclusion*). A log plot of the tensor skewness in Figure 3.21(c) shows a linear region confirming the exponential nature of the tensor skewness graph.

The occlusion and disocclusion are relevant in causal analysis where no time reversal is possible. In the case of non-causal analysis, occlusion and disocclusion lose meaning as occlusion in forward time is disocclusion in reverse time. Due to the non-causal approach, the ending of a motion trace is referred to as a left hand trace, and the starting of a motion trace is referred to as a right hand trace. We refer to the traces as *left hand movement* and *right hand movement* respectively.

The one-sided nature of the climb in tensor skewness is due to the central moving part not experiencing any occlusion or disocclusion in line 25. This information can also be used to determine *layering* of the objects in a video sequence. In a video scene, various objects can occlude and disocclude each other. In the tensor framework, an image pixel $\mathbf{I}_{i,k} \in \Omega$ is assigned to be part of a motion object Θ_i and not as part of an edge of a moving object. Figure 3.31 shows a simple scenario of a foreground object Θ_1 moving over a background Θ_n. The tensor skewness graph would be able to indicate where a foreground object would be occluding or disoccluding the background. It is also possible to determine which pixels belong to Θ_i.

It is also evident that there are small portions of a foreground object Θ_1 where neither occlusion or disocclusion takes place. This is where the relative motions are parallel to each other. In this region, tensor skewness provides no information of a moving object boundary. If the projection of the direction of the eigenvector $\hat{\mathbf{e}}_1$ is taken, it is normally found that the directions in the two adjacent objects are opposite and parallel. This is due to the skew kernel adding a higher degree of direction to the eigenvector than the symmetric eigenvector. This information could be used to bridge the gaps where the tensor skewness measure fails.

The decay on the tensor skewness is directly related to the scale constant σ in the direction of the

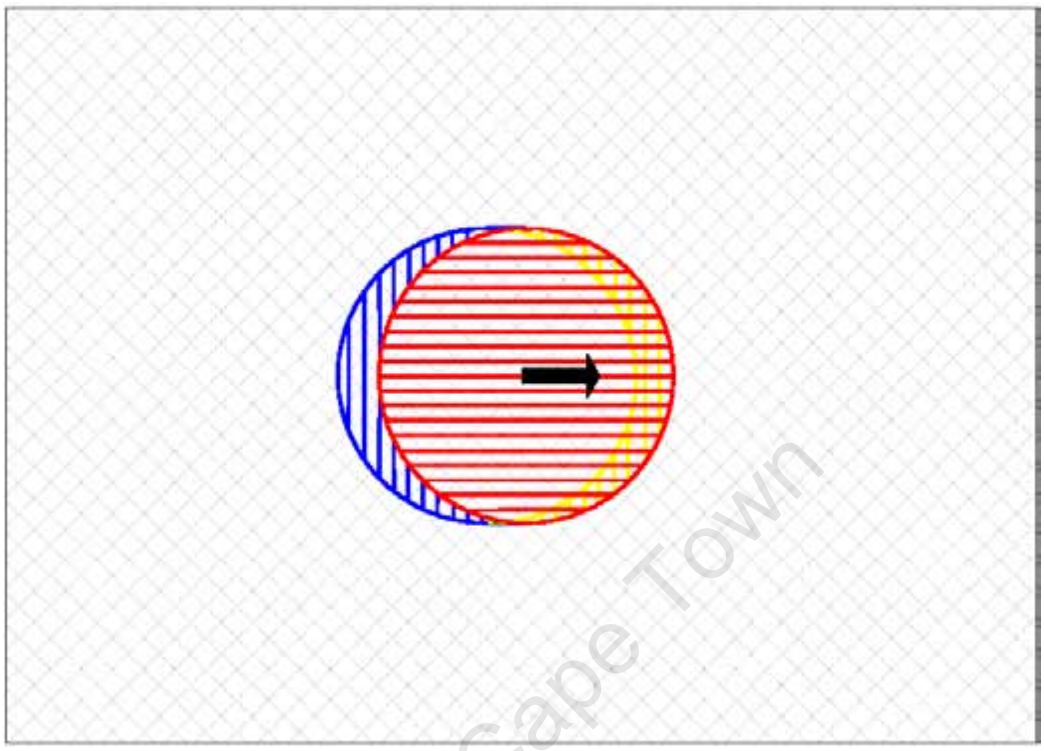
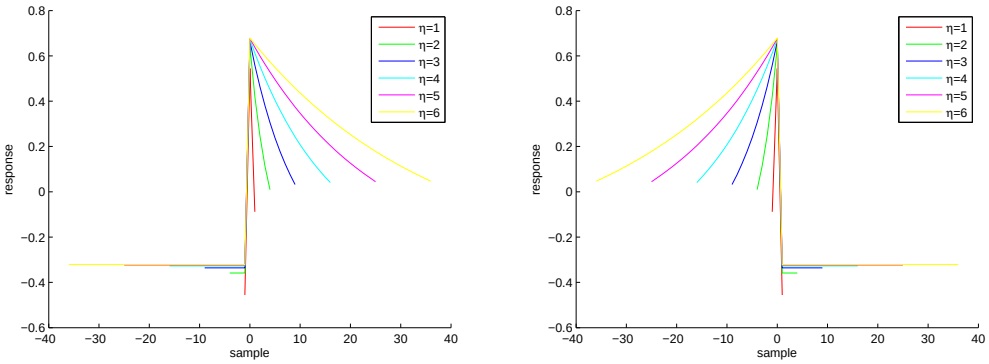


Figure 3.31: A foreground object Θ_1 (red) moving over a background object Θ_n (grey) causing occlusion or left hand movement (yellow) and disocclusion or right hand movement (blue)

eigenvector $\hat{\mathbf{e}}_1$. The larger the value of σ, the longer the tail of the skewness graph. Observing Figure 3.20(b), Figure 3.21(b), and Figure 3.22(b), it is noted that the exponential decay does not seem to be related to the dimensionality of the problem.

In filtering theory, a matched filter is deemed optimal in retrieving a signal of known shape in a noisy environment. The shape of the impulse response of the filter corresponds to the shape of the desired signal. Left hand and right hand traces are reversed shapes that can be derived from the same filter. The impulse response of a matched filter would be a filter with one half an exponential decay, and the other half a flat negative value as shown in Figure 3.32. The exponential section follows $y = e^{-\frac{x}{\eta^2}}$, and the whole filter is made zero mean.

Applying this filter in the direction of $\hat{\mathbf{e}}_1$ is difficult due to the data only being available on a regular grid. In order to simplify this, the filter is applied in the $\hat{x}$, $\hat{y}$ and $\hat{z}$ directions. Their orthogonality allows a linear recombination of the result. For the purposes of the analysis, the filter is applied on the tensor skewness graph in the x direction. This results in a filtered domain graph for both the left hand and right hand movement cases.



(a) Left hand movement filter.

(b) Right hand movement filter.

Figure 3.32: Matched left hand and right hand movement filters for various η.

Once the filter has been applied, the local peaks in the filtered domain are found. If the peak is significant ($> 50\%$ of the maximum of the filtered skewness values), then an occlusion or disocclusion boundary is declared. These results are applied to the two dimensional case in the following section.

3.10 Algorithm development with a two dimensional image

Now that the visualization of the single dimension has indicated a fairly generic algorithm, it is extended to a two dimensional image that is part of a spatio-temporal volume.

3.10.1 Encoding of tensors

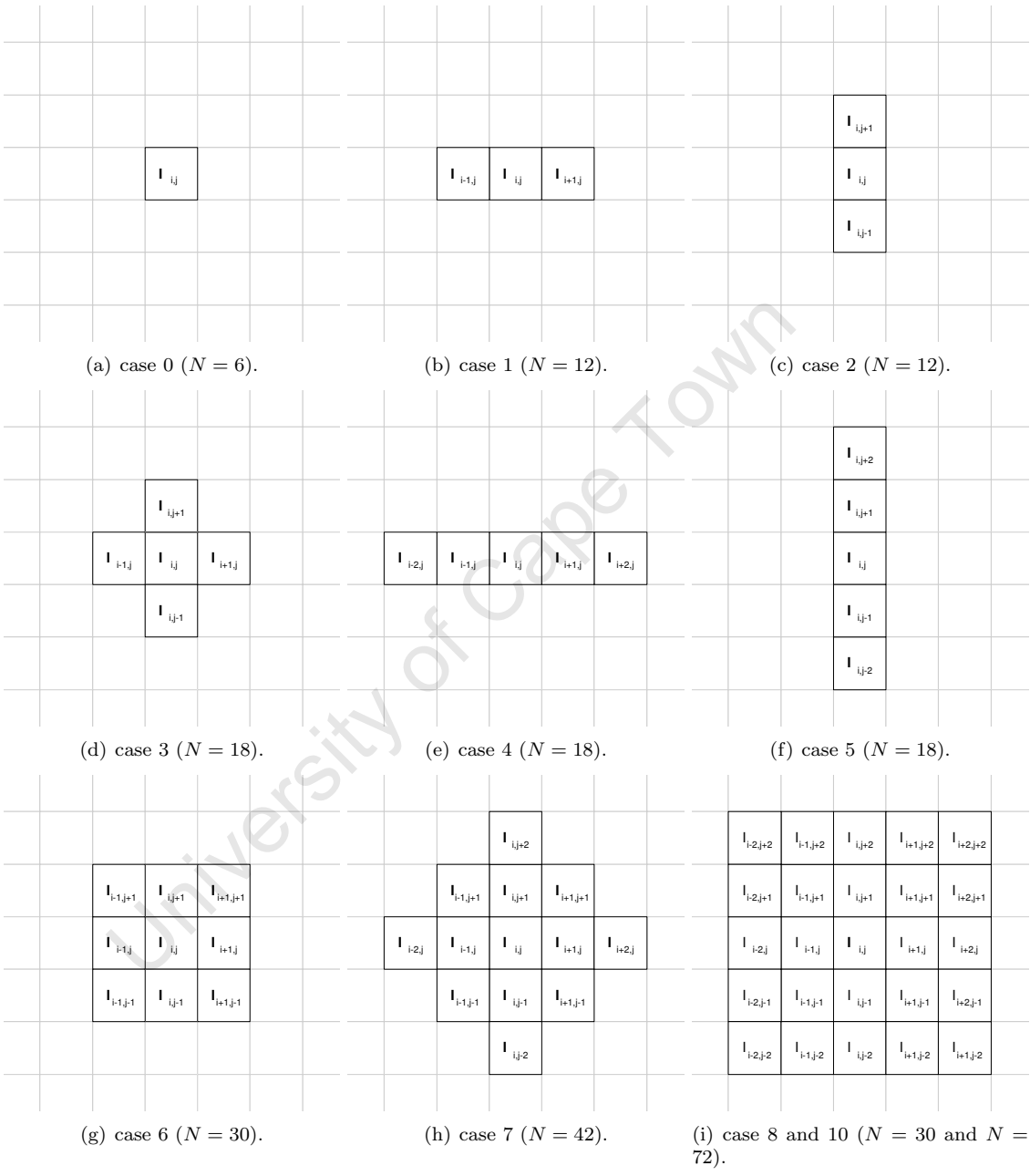
There are several methods of encoding the pixel information into a tensor representation. It is important to keep the elements of the tensor encoding for a pixel $\mathbf{I}_{i,j,k}$ extremely localised around the pixel. The pixels encoded all belong on the same x, y plane. The z plane is not used as motion information will be encoded into the tensor, which may not be stationary. Several arrangements of increasing tensor dimensionality are proposed in Figure 3.33 and are enumerated in Table 3.2. As the number of adjacent pixels increases, the dimensionality as given in Table 3.2 in brackets rises rapidly. For one dimensional analysis, cases 0, 1 and 4 are used as they have no y component. In the two dimensional analysis, cases 0, 3, 6, 7 and 8 are looked into. Case 8 is special in that the RGB colour information is not encoded in the adjacent pixels in order to reduce the dimensionality at the cost of selectivity. Using case 8 with all the RGB colour information included is used in a simplified voting method and is denoted as case 10.

As the dimensionality rises, the computational load and memory requirements rise. The highest implemented dimensionality for ball voting in this thesis is 42 (case 7). A simplified stick vote is used in this thesis up to a dimension of 72 (case 10).

3.10.2 Voter selection in the image volume

For the different cases, the voter selection for an arbitrary point in the moving earth sequence is chosen. Using the Euclidean distance between the votee and the other 3D pixel points, the closest 32 voters are chosen to participate in the vote. In [49], the tensor is made up from potentially good matches using a correlation technique with differing size windows. The normal correlation techniques determine the correlation of a potential voter with the votee. Including the adjacent pixels in the tensor encoding automatically incorporates this form of correlation and it extends it in including 32 of the closest candidates.

The relationship of voter selection based on the closest 32 Euclidean voters to the votee are shown in Figures 3.34 to 3.38. This is indicated for both a single votee showing the 3D positioning in x, y, z of the voters and their distance which is indicated as the greyscale colour of the voter. The closer the voter, the brighter the voter.



(a) case 0 ($N = 6$). (b) case 1 ($N = 12$). (c) case 2 ($N = 12$).

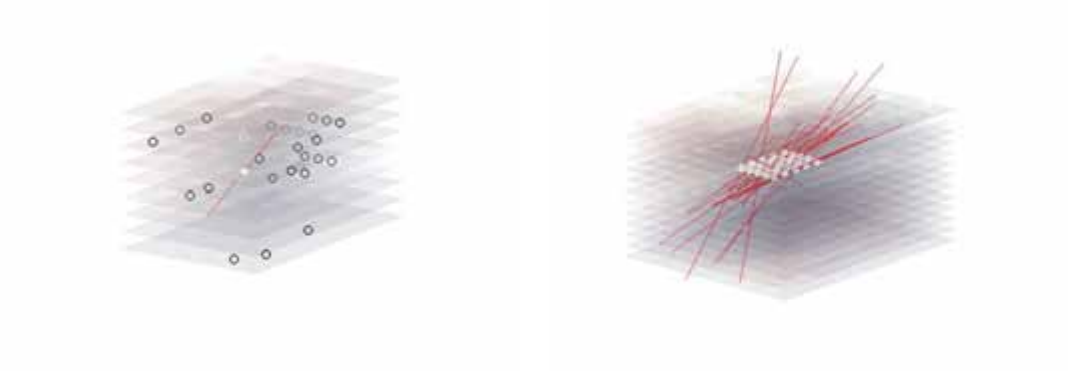
(d) case 3 ($N = 18$). (e) case 4 ($N = 18$). (f) case 5 ($N = 18$).

(g) case 6 ($N = 30$). (h) case 7 ($N = 42$). (i) case 8 and 10 ($N = 30$ and $N = 72$).

Figure 3.33: Colour tensor encoding pixel arrangements.

Table 3.2: Tensor encoding cases.

Case	Colour tensor encoding (dimension-N)	Monochrome tensor encoding (dimension-N)
0	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$ (N=6)	$i, j, k, Y_{i,j}$ (N=4)
1	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$ (N=12)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}$ (N=6)
2	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}$ (N=12)	$i, j, k, Y_{i,j}, Y_{i,j-1}, Y_{i,j+1}$ (N=6)
3	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}$ (N=18)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}, Y_{i,j-1}, Y_{i,j+1}$ (N=8)
4	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$, $R_{i-2,j}, G_{i-2,j}, B_{i-2,j}, R_{i+2,j}, G_{i+2,j}, B_{i+2,j}$ (N=18)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}, Y_{i-2,j}, Y_{i+2,j}$ (N=8)
5	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}$, $R_{i,j-2}, G_{i,j-2}, B_{i,j-2}, R_{i,j+2}, G_{i,j+2}, B_{i,j+2}$ (N=18)	$i, j, k, Y_{i,j}, Y_{i,j-1}, Y_{i,j+1}, Y_{i,j-2}, Y_{i,j+2}$ (N=8)
6	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}$ $R_{i-1,j-1}, G_{i-1,j-1}, B_{i-1,j-1}, R_{i-1,j+1}$, $G_{i-1,j+1}, B_{i-1,j+1}, R_{i+1,j-1}, G_{i+1,j-1}$, $B_{i+1,j-1}, R_{i+1,j+1}, G_{i+1,j+1}, B_{i+1,j+1}$ (N=30)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}, Y_{i,j-1}, Y_{i,j+1}$, $Y_{i-1,j-1}, Y_{i-1,j+1}, Y_{i+1,j-1}, Y_{i+1,j+1}$ (N=12)
7	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}$, $R_{i-1,j-1}, G_{i-1,j-1}, B_{i-1,j-1}, R_{i-1,j+1}$, $G_{i-1,j+1}, B_{i-1,j+1}, R_{i+1,j-1}, G_{i+1,j-1}$, $B_{i+1,j-1}, R_{i+1,j+1}, G_{i+1,j+1}, B_{i+1,j+1}$ $R_{i,j-2}, G_{i,j-2}, B_{i,j-2}, R_{i,j+2}, G_{i,j+2}, B_{i,j+2}$, $R_{i-2,j}, G_{i-2,j}, B_{i-2,j}, R_{i+2,j}, G_{i+2,j}, B_{i+2,j}$ (N=42)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}, Y_{i,j-1}, Y_{i,j+1}$, $Y_{i-1,j-1}, Y_{i-1,j+1}, Y_{i+1,j-1}, Y_{i+1,j+1}$, $Y_{i,j-2}, Y_{i,j+2}, Y_{i-2,j}, Y_{i+2,j}$ (N=16)
8	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}$, $Y_{i-1,j}, Y_{i+1,j}, Y_{i,j-1}, Y_{i,j+1}$, $Y_{i-1,j-1}, Y_{i-1,j+1}, Y_{i+1,j-1}, Y_{i+1,j+1}$, $Y_{i,j-2}, Y_{i,j+2}, Y_{i-2,j}, Y_{i+2,j}$, $Y_{i-2,j-2}, Y_{i-2,j+2}, Y_{i-2,j-2}, Y_{i-2,j+2}$, $Y_{i-1,j-2}, Y_{i-1,j+2}, Y_{i+1,j-2}, Y_{i-1,j-2}$, $Y_{i-2,j-1}, Y_{i+2,j-1}, Y_{i-2,j+1}, Y_{i+2,j-1}$ (N=30)	$i, j, k, Y_{i,j}, Y_{i-1,j}, Y_{i+1,j}, Y_{i,j-1}, Y_{i,j+1}$, $Y_{i-1,j-1}, Y_{i-1,j+1}, Y_{i+1,j-1}, Y_{i+1,j+1}$, $Y_{i,j-2}, Y_{i,j+2}, Y_{i-2,j}, Y_{i+2,j}$, $Y_{i-2,j-2}, Y_{i-2,j+2}, Y_{i-2,j-2}, Y_{i-2,j+2}$, $Y_{i-1,j-2}, Y_{i-1,j+2}, Y_{i+1,j-2}, Y_{i-1,j-2}$, $Y_{i-2,j-1}, Y_{i+2,j-1}, Y_{i-2,j+1}, Y_{i+2,j-1}$ (N=28)
10	$i, j, k, R_{i,j}, G_{i,j}, B_{i,j}, R_{i-1,j}, G_{i-1,j}, B_{i-1,j}, R_{i+1,j}, G_{i+1,j}, B_{i+1,j}$, $R_{i,j-1}, G_{i,j-1}, B_{i,j-1}, R_{i,j+1}, G_{i,j+1}, B_{i,j+1}, R_{i-1,j-1}, G_{i-1,j-1}, B_{i-1,j-1}$, $R_{i-1,j+1}, G_{i-1,j+1}, B_{i-1,j+1}, R_{i+1,j-1}, G_{i+1,j-1}, B_{i+1,j-1}$, $R_{i+1,j+1}, G_{i+1,j+1}, B_{i+1,j+1}, R_{i,j-2}, G_{i,j-2}, B_{i,j-2}, R_{i,j+2}, G_{i,j+2}, B_{i,j+2}$, $R_{i-2,j}, G_{i-2,j}, B_{i-2,j}, R_{i+2,j}, G_{i+2,j}, B_{i+2,j}, R_{i-2,j-2}, G_{i-2,j-2}, B_{i-2,j-2}$, $R_{i-2,j+2}, G_{i-2,j+2}, B_{i-2,j+2}, R_{i-2,j-2}, G_{i-2,j-2}, B_{i-2,j-2}$, $R_{i-2,j+2}, G_{i-2,j+2}, B_{i-2,j+2}, R_{i-1,j-2}, G_{i-1,j-2}, B_{i-1,j-2}$, $R_{i-1,j+2}, G_{i-1,j+2}, B_{i-1,j+2}, R_{i+1,j-2}, G_{i+1,j-2}, B_{i+1,j-2}$, $R_{i-1,j-2}, G_{i-1,j-2}, B_{i-1,j-2}, R_{i-2,j-1}, G_{i-2,j-1}, B_{i-2,j-1}$, $R_{i+2,j-1}, G_{i+2,j-1}, B_{i+2,j-1}, R_{i-2,j+1}, G_{i-2,j+1}, B_{i-2,j+1}$, $R_{i+2,j-1}, G_{i+2,j-1}, B_{i+2,j-1}$(N=72)	



(a) Voters around a votee. Voter markers get darker with increasing Euclidean distance from votee. Motion vectors shown as red line through votee.

(b) A small subset of votees indicating motion estimate consistency.

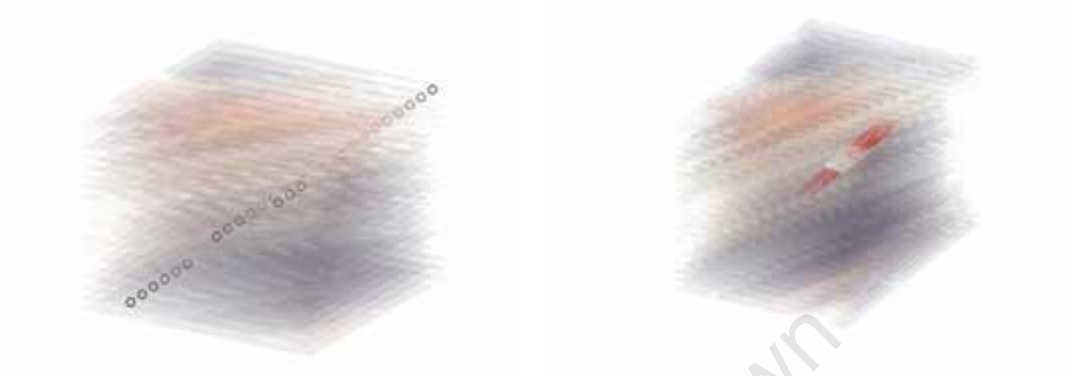
Figure 3.34: 3D image volume voter-votee relationship for case 0 encoding of the tensors.

The projection of the first eigenvector $\hat{\mathbf{e}}_1$ into the 3D space is also shown to indicate the estimate of the motion vector. The accuracy of the estimate is indicated by showing a small sub-set of votees and their motion vector estimates. In case 0, the motion vector estimates are fairly disorganised, but still have some degree of directionality. In case 3 and above, the motion vector estimates seem good with fairly little improvement as the dimensionality grows.

3.10.3 Analysis of two dimensional voting

In order to analyse the tangential tensors in the two dimensional plane the tissue-earth sequence used in the single dimensional analysis is used. The earth moves to the right, while the tissue moves slowly to the left. This motion is suitable for the further analysis as the skewness measure is analysed in the x direction only in the results. The motion vectors will be aligned in the x direction with no y component. This allows easy visual orientation checks of the tangential tensor voting approach.

Simulations on the tissue-earth sequence are run for cases 0, 3, 6, 7 and 8 which are the cases relevant to the two dimensional image in a spatio-temporal volume. In these simulations, the scale factor $\sigma = 20$, the number of iterations $f = 10000$ and the number of voters per votee is set at 32. A single ball vote pass is done with no removal or interpolation of data. In order to get an overall image of the effect of the tensor voting, only half the two dimensional image is shown. Figures 3.39 to 3.43 show the saliency of the first eigenvector $\hat{\mathbf{e}}_1$ as well as its projection on the two dimensional x, y plane. The projection is the orientation of the motion vector. Also shown are the tensor skewness measures in the two dimensional plane. This is referred to as the Tensor Skewness Map or *TS map*. Both the overall effect and the detail are shown. In the detail, the motion vector orientations can



(a) Voters around a votee. Voter markers get darker with increasing Euclidean distance from votee. Motion vectors shown as red line through votee.

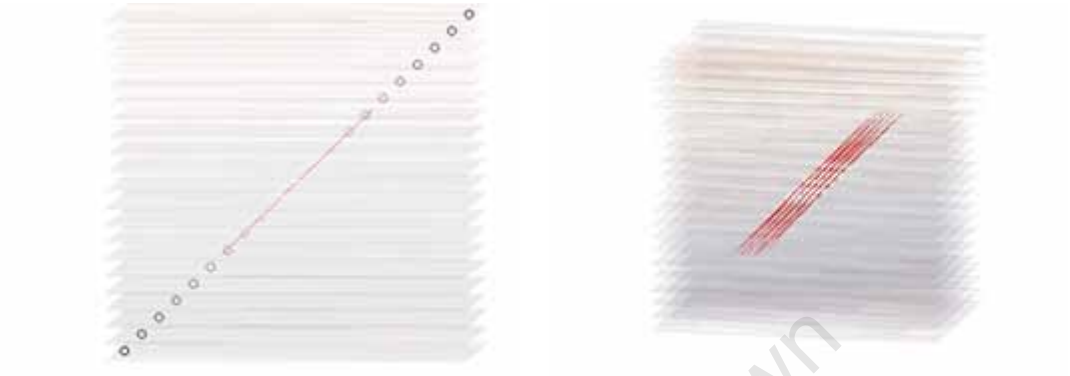
(b) A small subset of votees indicating motion estimate consistency.

Figure 3.35: 3D image volume voter-votee relationship for case 3 encoding of the tensors.

(a) Voters around a votee. Voter markers get darker with increasing Euclidean distance from votee. Motion vectors shown as red line through votee.

(b) A small subset of votees indicating motion estimate consistency.

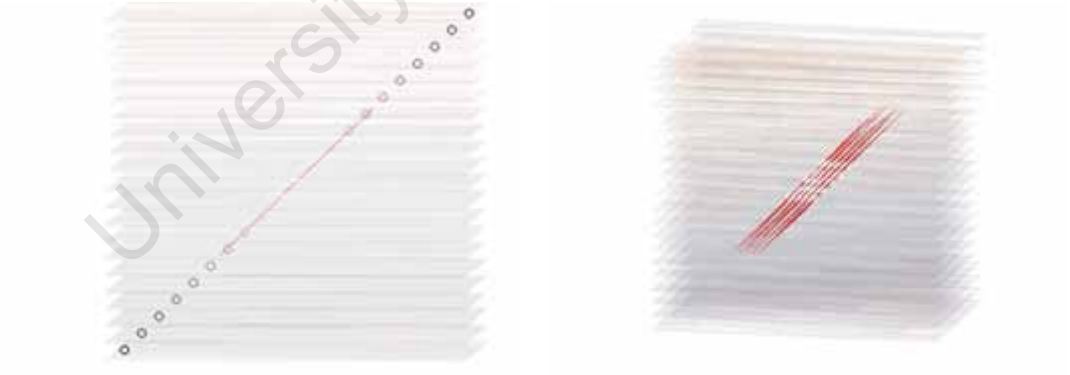
Figure 3.36: 3D image volume voter-votee relationship for case 6 encoding of the tensors.



(a) Voters around a votee. Voter markers get darker with increasing Euclidean distance from votee. Motion vectors shown as red line through votee.

(b) A small subset of votees indicating motion estimate consistency.

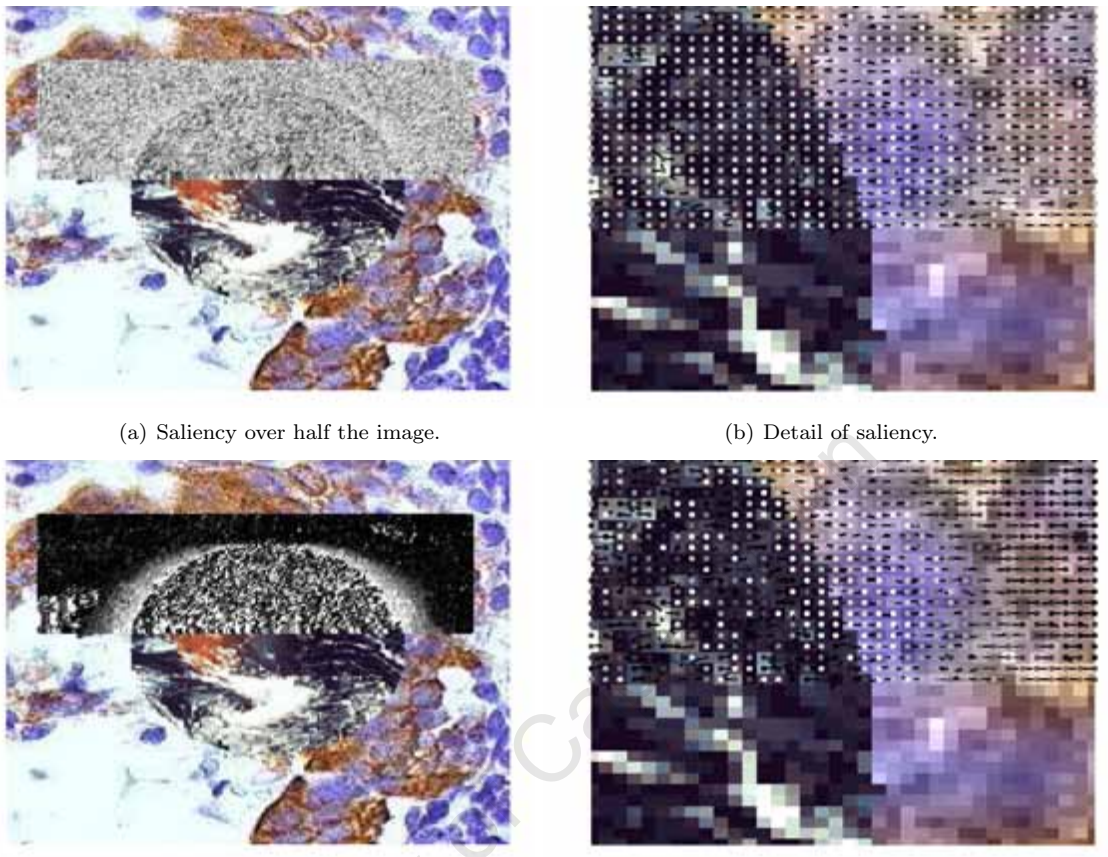
Figure 3.37: 3D image volume voter-votee relationship for case 7 encoding of the tensors.



(a) Voters around a votee. Voter markers get darker with increasing Euclidean distance from votee. Motion vectors shown as red line through votee.

(b) A small subset of votees indicating motion estimate consistency.

Figure 3.38: 3D image volume voter-votee relationship for case 8 encoding of the tensors.



(a) Saliency over half the image.

(b) Detail of saliency.

(c) *TS map* over half the image.

(d) Detail of *TS map*.

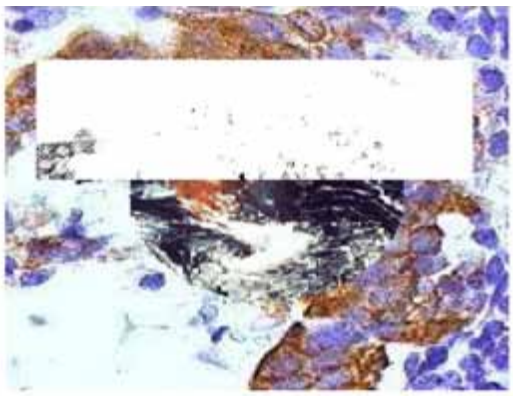
Figure 3.39: Case 0 tensor voting applied to the *tissue earth* ideal sequence.

be seen which are horizontal due to no y component in the motion.

The case 0 saliency in Figure 3.39 does not contain much structure, as can be seen by the motion vector orientations being inconsistent in places. Even though the motion vector estimation is not good, the *TS map* still shows structure at the occluding and disoccluding boundaries.

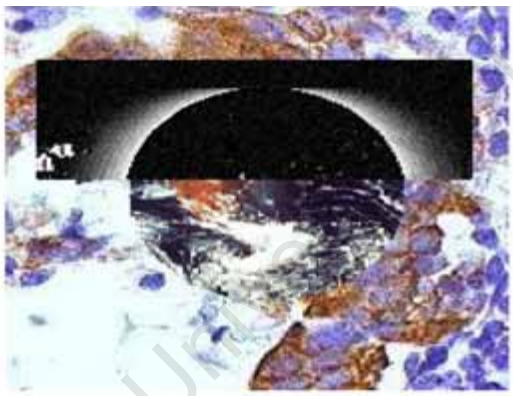
The case 3 saliency in Figure 3.40 shows good saliency, as evidenced by the bright saliency map. There is a section on the left where the saliency breaks down slightly. This is due to an area which has few features and can be expected. The motion vector orientations are consistent in the x direction, and the *TS map* shows good structure at the occluding and disoccluding boundaries.

The results of case 6 in Figure 3.41 show little improvement over case 3. In the cases of pixel aliasing and noise, there may be an advantage to using case 6.

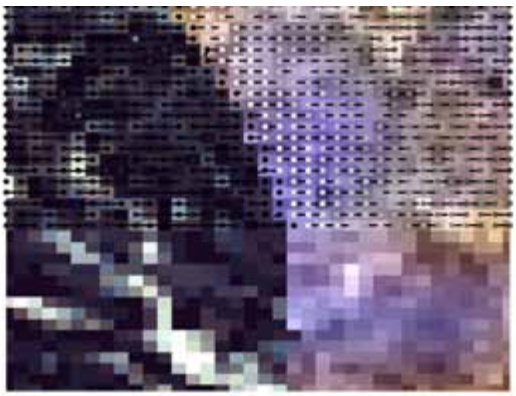


(a) Saliency over half the image.

(b) Detail of saliency.

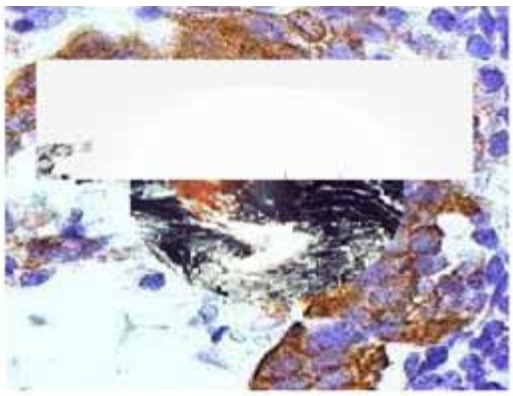


(c) *TS map* over half the image.



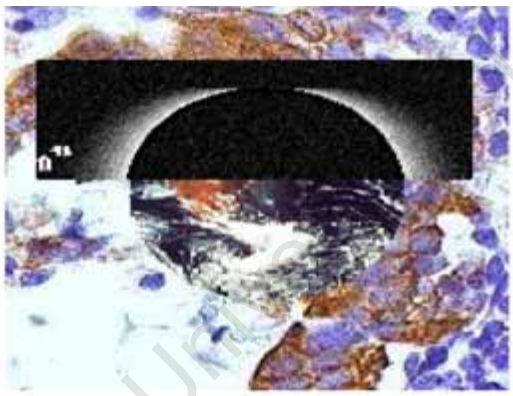
(d) Detail of *TS map*.

Figure 3.40: Case 3 tensor voting applied to the *tissue earth* ideal sequence.

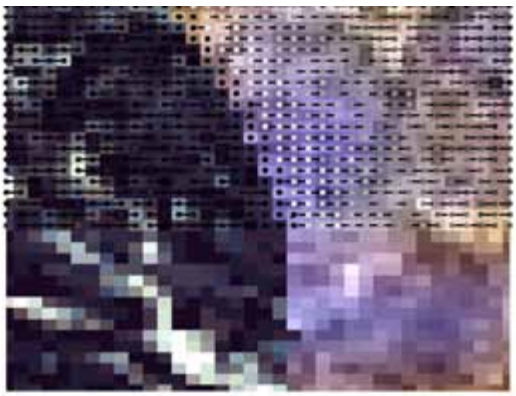


(a) Saliency over half the image.

(b) Detail of saliency.

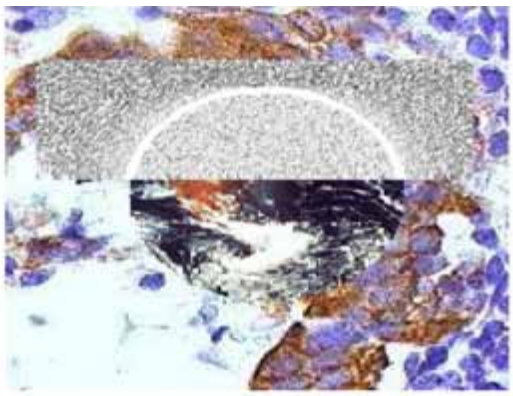


(c) *TS map* over half the image.



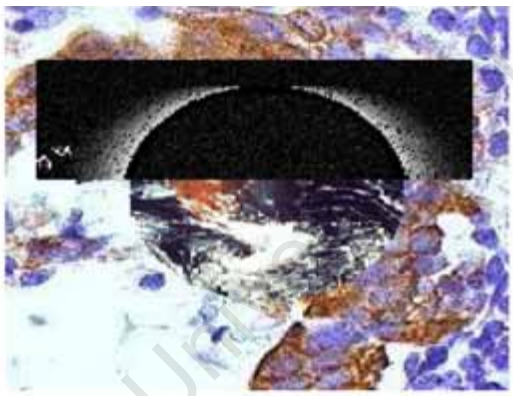
(d) Detail of *TS map*.

Figure 3.41: Case 6 tensor voting applied to the *tissue earth* ideal sequence.

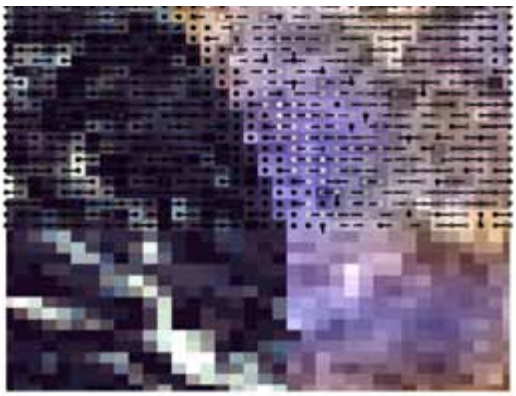


(a) Saliency over half the image.

(b) Detail of saliency.

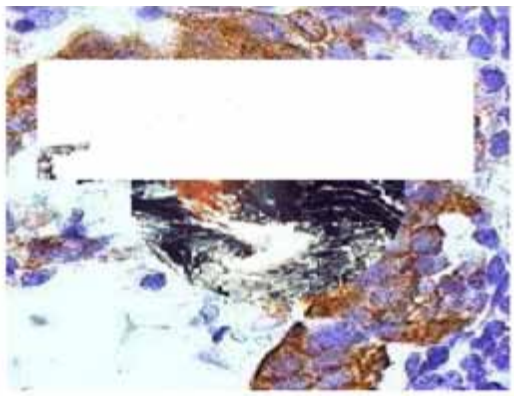


(c) *TS map* over half the image.



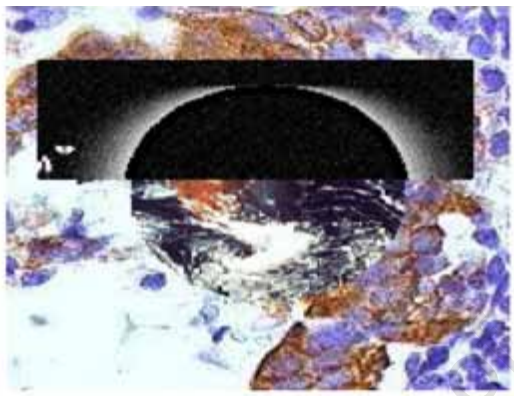
(d) Detail of *TS map*.

Figure 3.42: Case 7 tensor voting applied to the *tissue earth* ideal sequence.

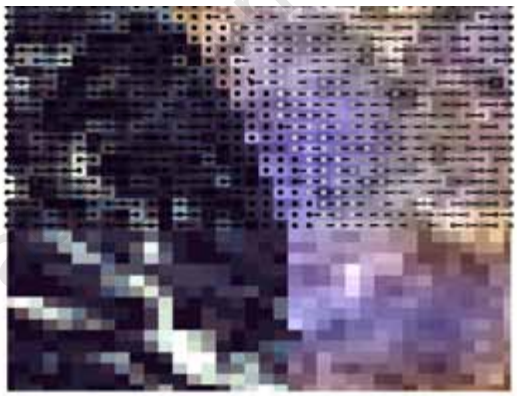


(a) Saliency over half the image.

(b) Detail of saliency.



(c) *TS map* over half the image.



(d) Detail of *TS map*.

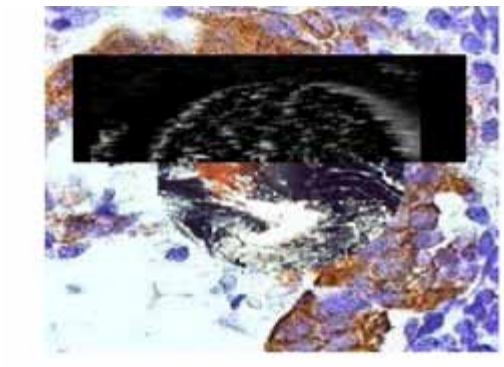
Figure 3.43: Case 8 tensor voting applied to the *tissue earth* ideal sequence.

The results of case 7 in Figure 3.42 show a degradation. This is probably due to the high dimensionality ($N = 42$) that causes inconsistencies in calculation. The effects of higher dimensionality on the Monte Carlo analysis are analysed in the next section.

The results of case 8 in Figure 3.43 show consistency with case 3 and case 6. The detail on the skewness also shows the averaging effect of the wider kernel in Figure 3.43(d), where there is a 2 pixel offset to the right. The effect of averaging is not wanted in trying to determine motion boundaries, and as such this case has limited use.

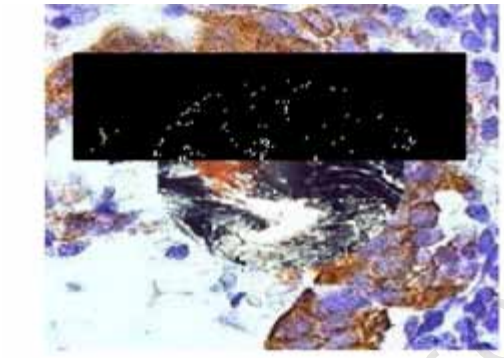
The occluding and disoccluding exponential filters are applied with $\eta = 2$. The exponential filters are only applied in the x-direction as the motion is only in that direction.

In case 0 the results of the left hand and right hand movement filters and detectors can be seen in Figure 3.44. The difference in detection of occlusion in Figure 3.44(a) and disocclusion in Figure 3.44(e)

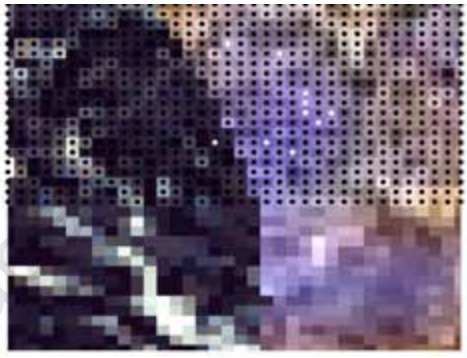


(a) Left hand movement filter results.

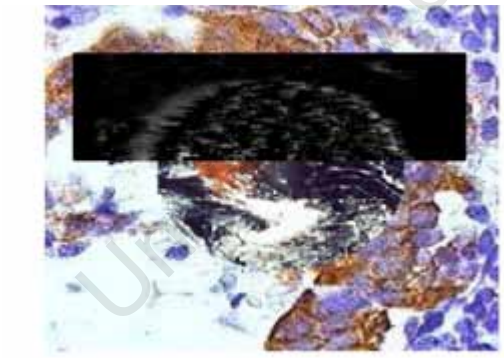
(b) Detail of left hand movement filter results.



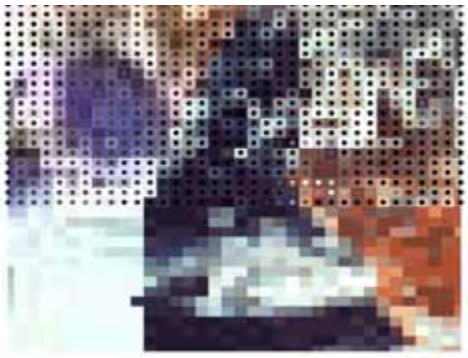
(c) Left hand movement filter detection at 50% of maximum.



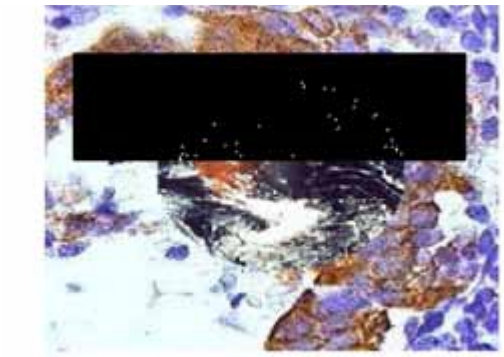
(d) Detail of left hand movement detection.



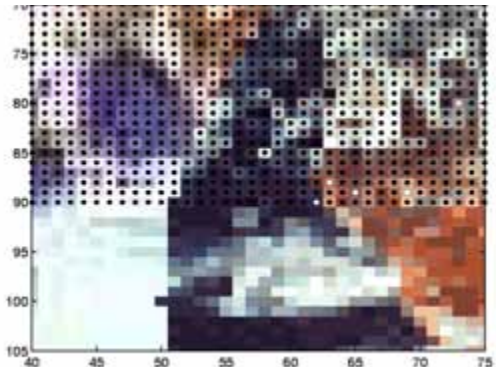
(e) Right hand movement filter results.



(f) Detail of right hand movement filter results.



(g) Right hand movement filter detection at 50% of maximum.



(h) Detail of right hand movement detection.

Figure 3.44: Case 0 *TS Map* passed through left hand and right hand movement filters.

is the wider detection swathe on the side of the object of interest. The swathe approximately borders the edge of the moving object, and the side which it occurs depends whether the motion is of an occluding or disoccluding nature. In case 0, the filters give a visual indication that the occluding and disoccluding structure is present, but the detections show fairly poor performance in discriminating the occluding and disoccluding boundary.

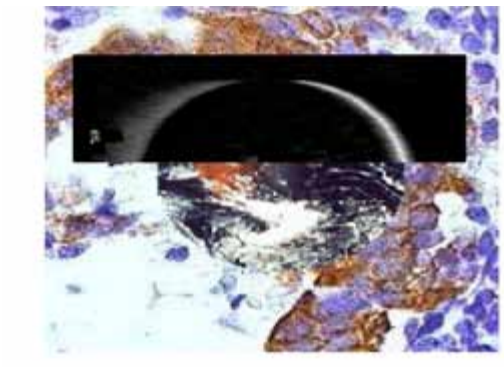
In case 3 the results of the occluding and disoccluding filters and detectors can be seen in Figure 3.45. When compared to case 0, the filter responses are much clearer in highlighting the occluding and disoccluding boundaries. The occluding and disoccluding detectors perform very well clearly demarcating the boundaries. A false detection is found on the left due to an ambiguous motion caused by aliasing.

In case 6 the results of the occluding and disoccluding filters and detectors can be seen in Figure 3.46. Case 6 gives very similar results to case 3. Considering that the images are ideal, case 6 is expected to perform better than case 3 in non-ideal (interpolation and noise) conditions. The occluding and disoccluding detectors perform very well, clearly demarcating the boundaries.

In case 7 the results of the occluding and disoccluding filters and detectors can be seen in Figure 3.47. The results of case 7 are not as good as case 3, 6 and 8 due to the high dimensionality. The poor result leads to more analysis later on the *Flower Garden* natural image sequence to try understand the problems associated with high dimensionality in tangential tensor voting.

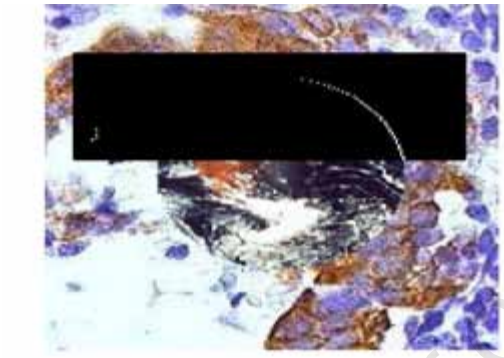
In case 8 the results of the occluding and disoccluding filters and detectors can be seen in Figure 3.48. Case 8 gives very similar results to case 3 and 6. The major difference is the effect of the filter width on the clear demarcation of the boundary. The offset on the detections is not only due to the apparent lag of the filter, but the filter seems to be averaging the edge confirming the hypothesis that larger regions of support affect the edge detection capability of the tensor voting framework.

In this section several of the different tensor encodings are applied to an ideal *tissue earth* image sequence. The reason for doing this is to determine at what stage does an increase in tensor encoding dimensionality have little effect on detecting occlusions and disocclusions. The increase in tensor encoding also affects the ability to discriminate edges accurately. From the results given in Figures 3.44 to 3.48, case 6 emerges as most appropriate in terms of detecting occlusions and disocclusions. Case 3 is also able to discriminate well, but may perform poorly under natural conditions.

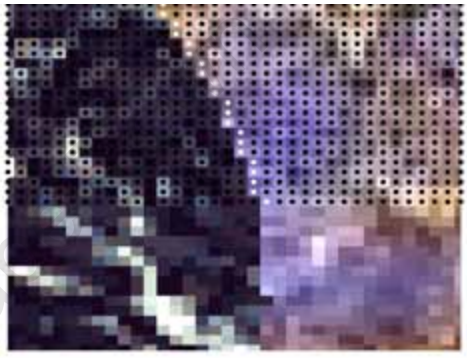


(a) Left hand movement filter results.

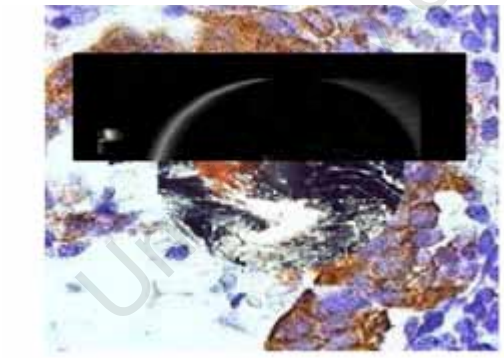
(b) Detail of left hand movement filter results.



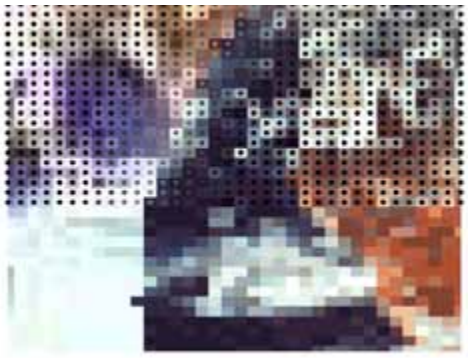
(c) Right hand movement filter detection at 50% of maximum.



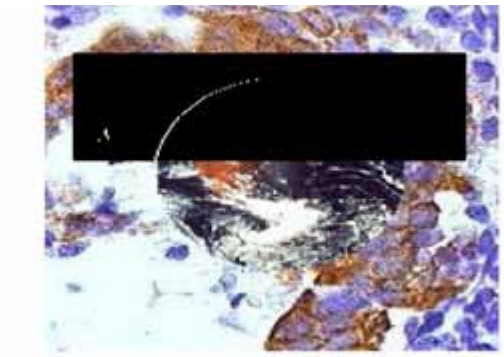
(d) Detail of right hand movement detection.



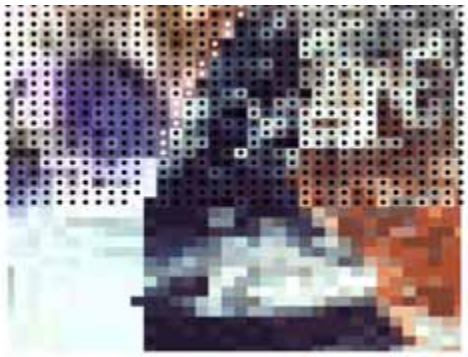
(e) Disocclusion filter results.



(f) Detail of right hand movement filter results.

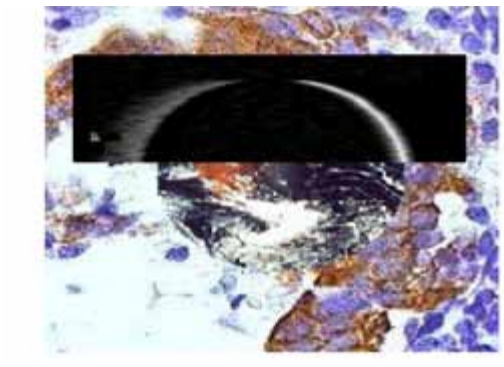


(g) Right hand movement filter detection at 50% of maximum.



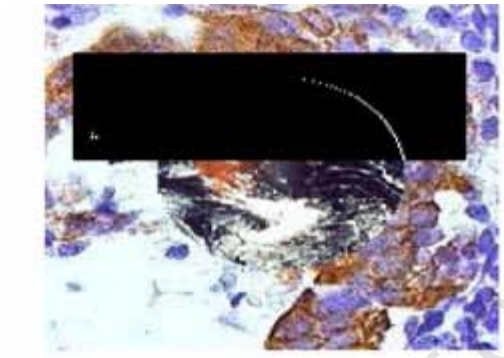
(h) Detail of right hand movement detection.

Figure 3.45: Case 3 *TS Map* passed through left hand and right hand movement filters.

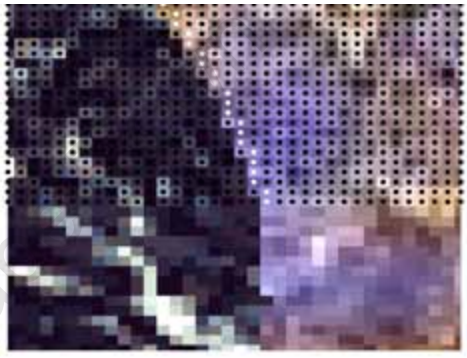


(a) Left hand movement filter results.

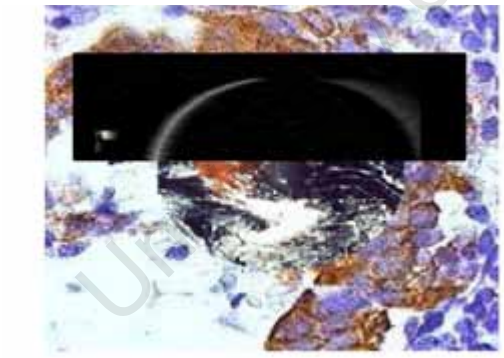
(b) Detail of left hand movement filter results.



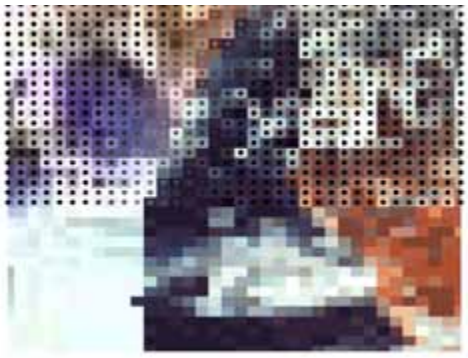
(c) Left hand movement filter detection at 50% of maximum.



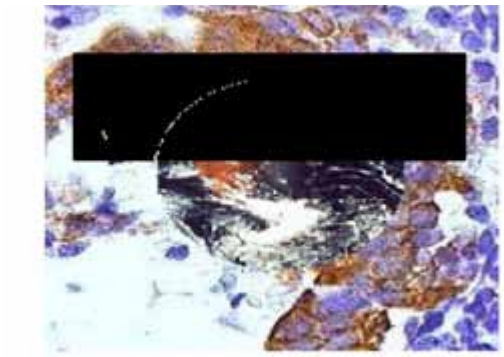
(d) Detail of left hand movement detection.



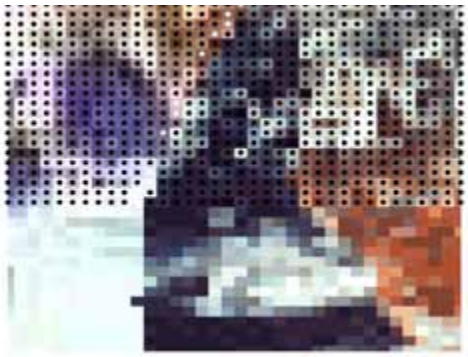
(e) Right hand movement filter results.



(f) Detail of right hand movement filter results.

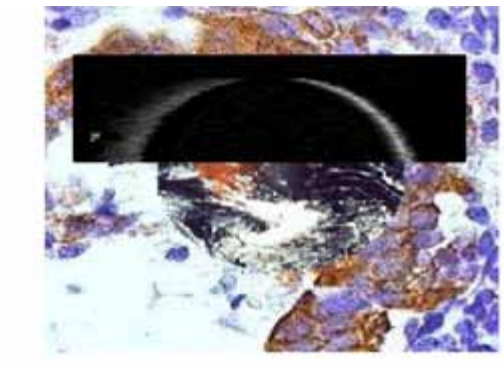


(g) Right hand movement filter detection at 50% of maximum.

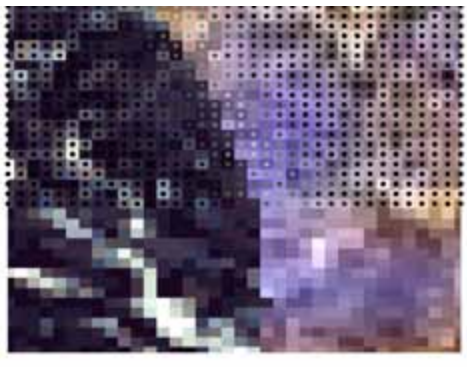


(h) Detail of right hand movement detection.

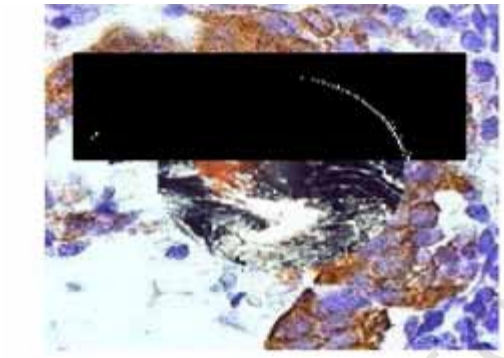
Figure 3.46: Case 6 *TS Map* passed through left hand and right hand movement filters.



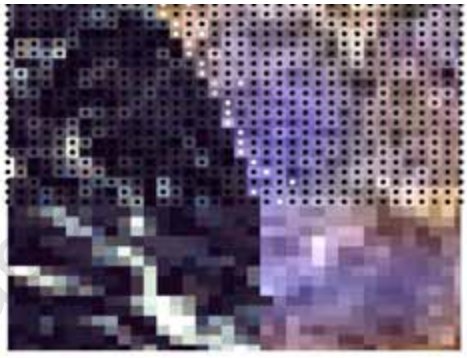
(a) Left hand movement filter results.



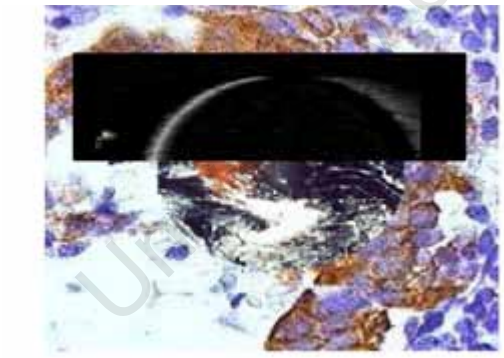
(b) Detail of left hand movement filter results.



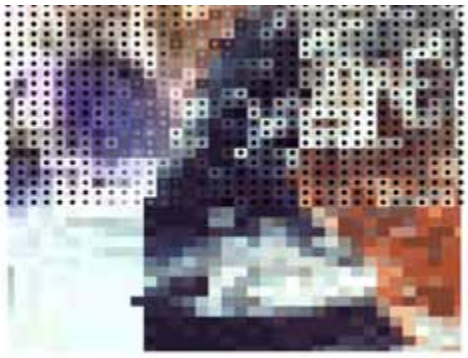
(c) Left hand movement filter detection at 50% of maximum.



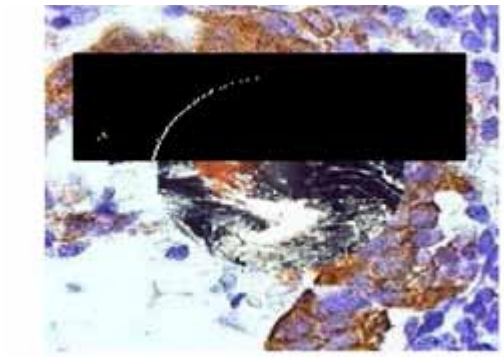
(d) Detail of left hand movement detection.



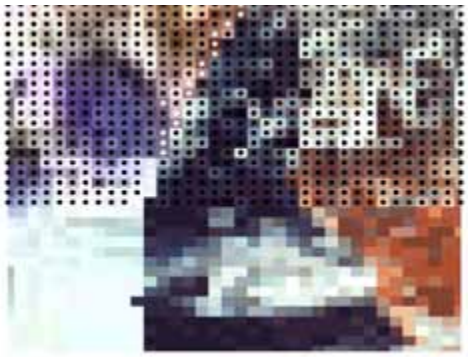
(e) Right hand movement filter results.



(f) Detail of right hand movement filter results.

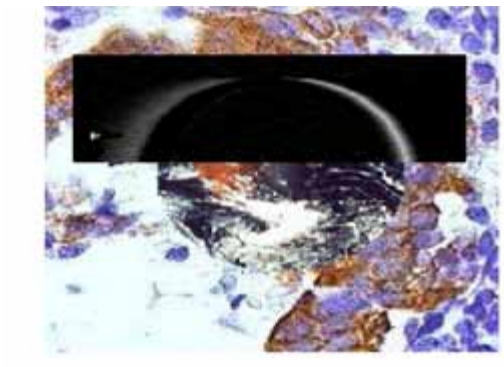


(g) Right hand movement filter detection at 50% of maximum.



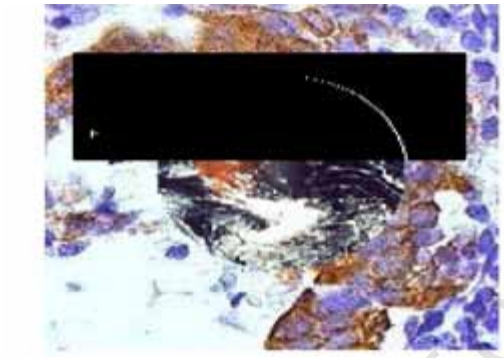
(h) Detail of right hand movement detection.

Figure 3.47: Case 7 *TS Map* passed through left hand and right hand movement filters.

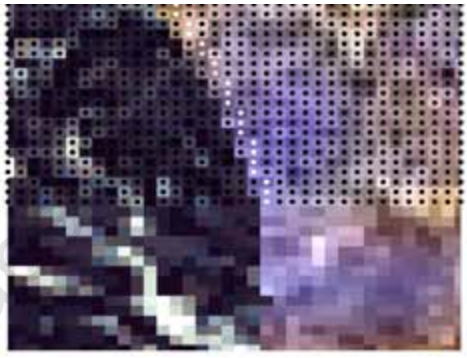


(a) Left hand movement filter results.

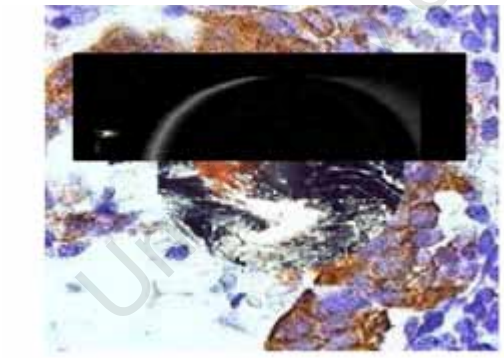
(b) Detail of left hand movement filter results.



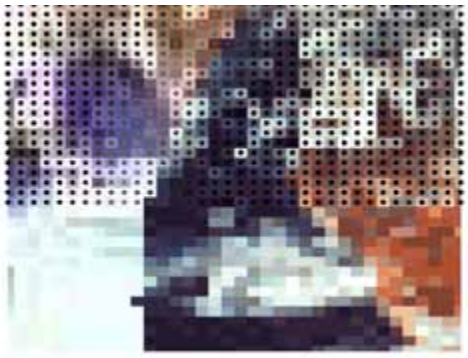
(c) Left hand movement filter detection at 50% of maximum.



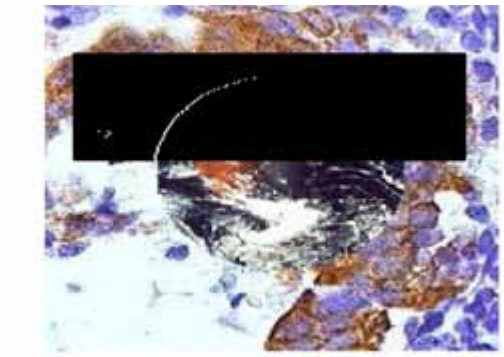
(d) Detail of left hand movement detection.



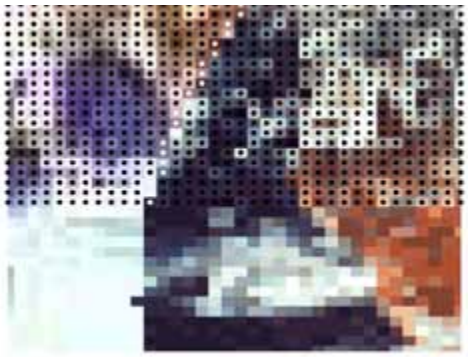
(e) Right hand movement filter results.



(f) Detail of right hand movement filter results.

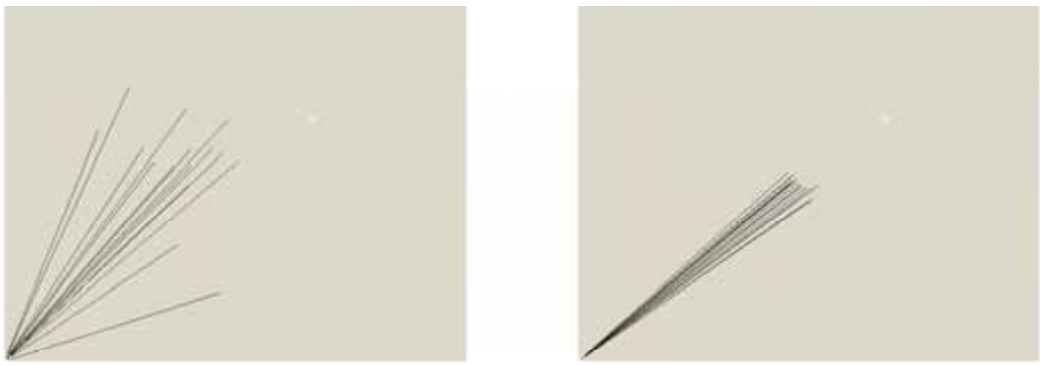


(g) Right hand movement filter detection at 50% of maximum.



(h) Detail of right hand movement detection.

Figure 3.48: Case 8 *TS Map* passed through left hand and right hand movement filters.



(a) $\hat{\mathbf{e}}_1$ orientation with $N = 42$ and $f = 1000$.

(b) $\hat{\mathbf{e}}_1$ orientation with $N = 42$ and $f = 30000$.

Figure 3.49: Twenty Monte Carlo estimations of the first eigenvectors $\hat{\mathbf{e}}_1$ orientation from the votee towards the voter.

3.11 High dimensionality tensor voting Monte Carlo analysis

A degradation of the skewness measure shown in Figure 3.37 for case 7 is observed. In order to understand this better, the interaction between a single voter and votee is analysed for varying dimensionalities using the Monte Carlo methods.

As a parameter to this analysis, the number of iterations f is varied. The other parameters are held constant at nominal values of $\sigma = 10$, $\alpha = 1$, $k_t = 1$, $k_{RGB} = 1$ and $k_{xy} = 1$. The distance between the voter and votee in N dimensional space is also kept constant. Twenty independent Monte Carlo runs are done to find the projection of the first eigenvector $\hat{\mathbf{e}}_1$ into 3D. Typical results are shown in Figure 3.49. The experiments are repeated for 18, 30 and 42 dimensions with several values of f resulting in angular errors (standard deviation) shown in Figure 3.50.

The angular errors climb as the dimensionality rises, but can be counteracted by increasing the number of iterations f in the Monte Carlo simulation. The reason for this is the reducing probability of randomly choosing a vector that will point to a small region in N dimensional space as N increases.

Running Monte Carlo simulations with large f and large N is difficult as computation time and storage requirements increase both with f and N. A simplified solution is to make use of a single vote aligned to the connecting line between the voter and votee. The simplified solution is equivalent to an aligned stick vote or a ball vote with $\alpha \to \infty$. Although this is not the case, the simplification can be warranted to greatly simplify the calculations needed in higher dimensional space. A similar approach is used by Jia [27], who uses high dimensional stick voting to infer sections of images that are missing.

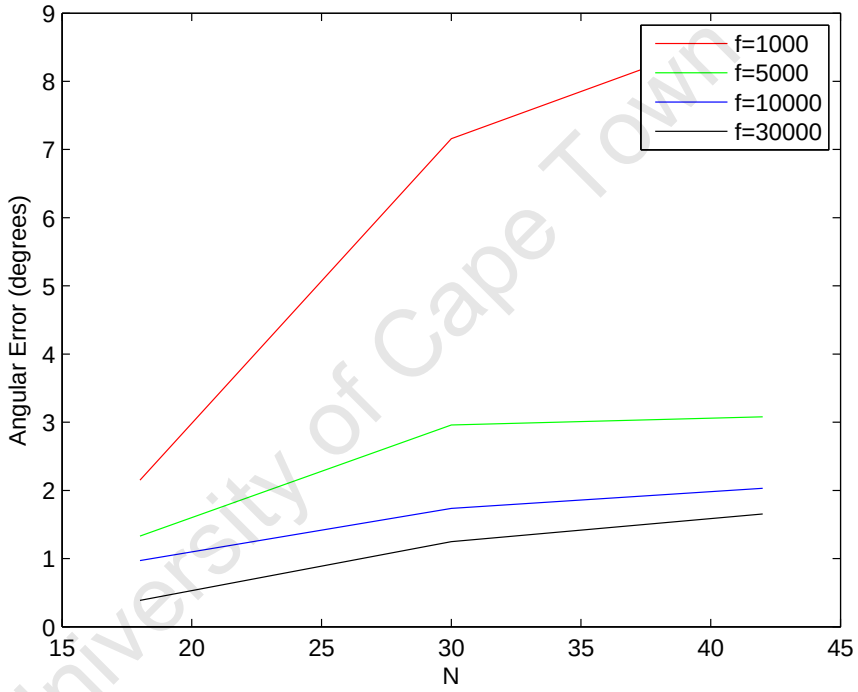


Figure 3.50: Monte Carlo angular error for a single voter—votee pair.

(a) Ball vote.

(b) Simplified stick vote.

Figure 3.51: Twenty Monte Carlo estimations of the first eigenvectors $\hat{\mathbf{e}}_1$ orientation using 32 voters on the ideal *tissue earth* sequence ($N = 42$, $f = 10000$).

The effect of the ball vote and the simplified stick vote is applied to case 7 ($N = 42$) on the ideal *tissue earth* sequence in Figure 3.51. The simplified stick vote is well aligned to the voters in the ideal case. When a natural sequence such as the *flower garden* sequence is used, the stick vote gives a similar estimate of the motion vector as the ball vote. Simulations are conducted using both techniques on natural sequences in the experiments and measurements chapter.

3.12 Summary

This chapter developed the application of the skew tangential voting kernel applied to motion segmentation. The concept of skewness was introduced and expanded on. The concept of skewness is associated to occlusion and disocclusion using matched filters to detect motion boundaries proposed and used on a synthetic image sequence.

When non-ideal conditions are presented, such as with an interpolated synthetic image sequence, the skewness measure is adversely affected. Numerous methods were explored to try reinstate the skewness measure.

- *Voter pre-alignment.* A two-tier voting approach to refine the ball vote into an aligned vote.
- *Variation of k_{RGB}.* The effect of varying k_{RGB} was investigated.
- *Skewness only over spatial dimension.* The effect of only working the skewness measure out over the spatial dimensions.

(a) Ball vote. (b) Simplified stick vote.

Figure 3.52: Twenty Monte Carlo estimations of the first eigenvectors $\hat{\mathbf{e}}_1$ orientation using 32 voters on the *flower garden* sequence ($N = 42$, $f = 10000$).

None of these measures caused the skewness measure to visually improve.

A further problem with the increasing dimensions on Monte Carlo analysis was identified. As the dimensionality increases, the number of iterations needed for valid Monte Carlo results also rises. A simplified stick vote solution that is applied to one of the natural sequences is proposed later.

Chapter 4

Practical processing of the tensor voting framework

Even though the approach has been to ignore computational load, realistic results only occur after realistic simulations. Some of the results in the thesis require computational resources beyond that of a PC to complete within a reasonable time frame. Recent developments in the Graphic Processor field and the associated computational frameworks allow a reasonably priced solution to this vastly parallel formulation. Some insights are included.

4.1 Introduction

In the previous chapter, the theory of skew tangential tensor voting was proposed and analysed. A novel skewness measure was introduced as a method to determine motion boundaries based on occlusion or disocclusion.

In order to run Monte Carlo simulations using these techniques on image sequences it is necessary to implement the algorithm on a suitable architecture. Due to the highly parallel and independent nature of tensor voting and work done by Min [44], a Graphic Processor Unit (GPU) architecture is used.

In this chapter the implementation of the skew tangential tensor voting framework on such an architecture is described and discussed. The difficulties encountered and results achieved are also given.

4.1.1 Graphic Processor Units (GPUs)

The tensor voting problem presents a computational challenge especially when the number of dimensions rises. When the dimensionality is small ($N < 4$), then it is possible to use lookup tables. This is not feasible for the high dimensions used in the methods described in the thesis as the dimensionality usually exceeds $N = 5$. In order to accumulate votes cast by voter tensors at votee sites, Monte Carlo runs need to be done, resulting in a massively parallel as well as a massively sequential computational need.

GPUs have been fueled by the computer gaming industry to achieve massive parallel processing capability for rendering 3D graphics on screen. Tools have also become available for the GPU to allow results to be extracted from the GPU instead of rendering to the screen. Min has demonstrated the use of GPUs for Tensor Voting in [44] using a Nvidia GeForce 7800GTX. Min made use of leading technology at that stage, but the dimensionality was still limited to $N = 5$.

Subsequent advances in the graphic card environment have allowed the number of parallel processors to increase, and still be affordable. In this formulation, a Nvidia 260GTX was used which has an increased memory bandwidth from 55GByte/s to 100GByte/s as well as a processing increase from 175GFlop/s to 900GFlop/s. The number of transistors has also risen from 302 million transistors to 1.4 billion transistors on the die. This has occurred in the space of 3 years. For the tensor implementation, the new 260GTX supports double precision floating point natively and is one of the first GPU processors to do so.

The programming tools have also made large advances. The tools now use a well integrated environment called Compute Unified Device Architecture (CUDA) [51]. CUDA uses a C based environment allowing easy integration of existing algorithmic code with a few extensions to allow parallel computation in the framework. It also integrates easily in well known Integrated Development Interfaces (IDE) such as Microsoft® Visual C++. Compile times are short, but debug facilities are still problematic.

Table 4.1: GTX 260 capabilities.

Parameter	Value
Multiprocessors	24
Global Memory	896 MByte
Memory bus width	448 bit
Transistors	1.3 Billion
Core Speed	576 MHz
Memory Speed	1998 MHz
Available Threads	512
Shared Memory	16254 Bytes
Constant Memory	64544 Bytes

4.2 High dimensional tensor voting implementation

The approach taken to allow development on the GPU is a tiered approach:

- The algorithm is tested and developed in MATLAB without any optimisations, and is functionally correct. MATLAB is used to provide a convenient means to create data input and display data output, and remains in the software for this purpose in the GPU implementation.
- The core algorithm is written in C and introduced to MATLAB as a MEX file. Parameters are passed in and out of the C code, as well as using file interfaces for the input data which is difficult for MATLAB to manage.
- The C algorithm has predefined switches to call into CUDA, which uses the GPU to do the highly parallel and computationally intensive parts of the algorithm. Data preparation still occurs in the C environment as this is not easily scalable into the GPU architecture.

4.2.1 The GPU environment

The GPU used for this experimentation is the GTX260 which has the specifications [51] given in Table 4.1. The GPU is arranged to run parallel threads making use of its multiprocessors. The multiprocessors have a Single Instruction Multiple Thread (SIMT) architecture where a single processor can run several sets of threads with their own context through one instruction. The GPU is able handle divergent code in the threads by stopping instructions to the other threads until the threads instructions converge again. The number of stalled threads is related to the warp size of the processor, which is fixed at 32.

The thread arrangement is shown in Figure 4.1(a). The GPU is able to make 3D blocks of threads

that all execute concurrently as long as the number of threads does not exceed 512. Each thread needs local variables and registers which are taken out of shared memory. Thread blocks can also see shared memory in a communal way, where each one sees the same content. Problems arise when thread safe writing to this memory must occur as CUDA only has a couple of atomic instructions — none of which operate on double precision numbers. The shared memory, as seen in Figure 4.1(b), is a scarce resource limited to 16KByte. In optimising performance, due consideration of the memory constraint must be taken versus the thread count. The GPU also has very efficient synchronisation to allow all the threads to get back into step.

Over and above threads, thread blocks can be arranged in a 2D grid. The GPU schedules these thread blocks to execute as soon as there are open slots on the multiprocessors. There is no guarantee of the order of execution unless the host computer passes the GPU thread blocks and then waits for completion before passing the next block. The shared memory does not persist between thread blocks — it is only valid for the lifetime of a thread block.

The memory arrangement in the GPU is shown in Figure 4.1(b). Each thread has several local variables that use registers that reside in shared memory and have high speed access of about 4 clock cycles. When there are too many local variables, local memory is used which has a latency of 400–600 clock cycles as it is held in uncached global memory. To avoid this situation it is necessary to preallocate shared memory for arrays, as the compiler automatically places arrays in local memory.

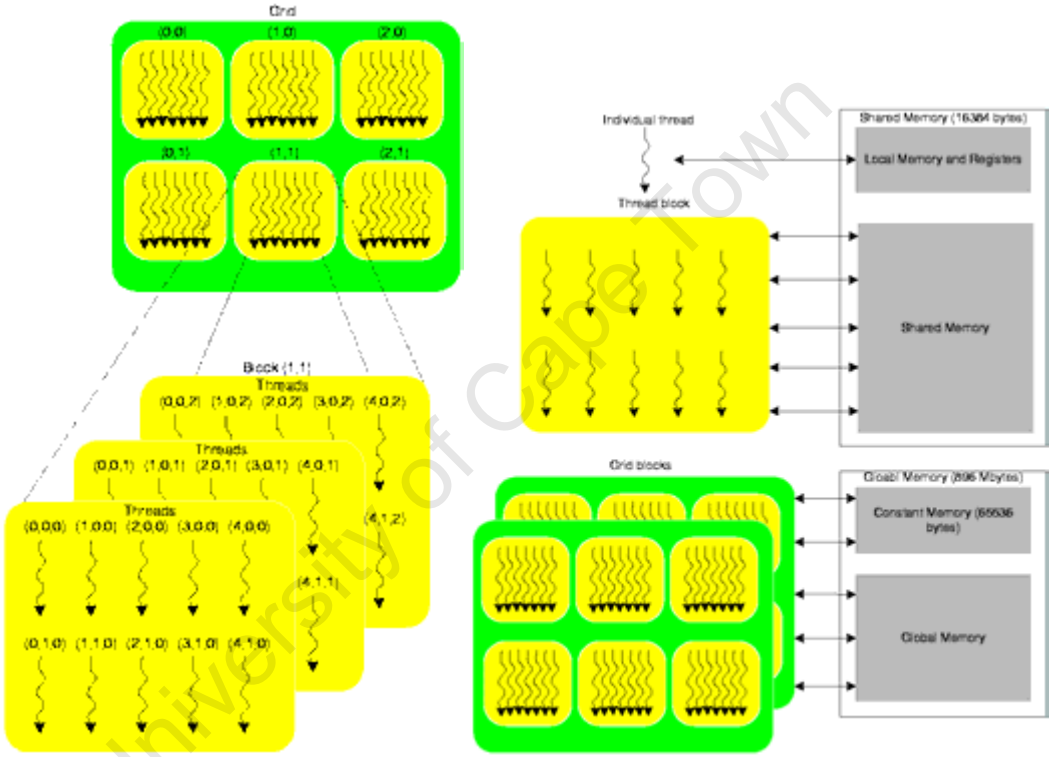
Shared memory is high speed memory on a single multiprocessor and can be used to allow several threads to read common variables or to allocate sections based on the identity of the thread in separate areas in shared memory.

Constant cache memory is a limited section of global memory where high speed read-only access can occur. Unlike shared memory, constant memory is persistent between thread blocks. Constant memory values can only be changed by the host and not the GPU.

Global memory is fairly large, but still small when looked at from the video processing perspective. Global memory has large latencies of 400–600 clock cycles in access. In order to counteract this, the GPU has several forms of memory access coalescing methods when threads access adjacent portions of memory simultaneously. The GPU compiler also allows instructions to be executed that do not rely on the outcome of the memory access before stalling the processor.

4.2.2 Crafting the algorithm for the GPU

The algorithm needs to be carefully coded into the GPU architecture to maximally use its resources. The algorithm code was based on TVLib Version 1.0, written by Tang at University of Southern



(a) Parallel threads and blocks.

(b) Tiered memory architecture.

Figure 4.1: Nvidia GTX260 architecture.

California. This code was extensively changed to cater for the new aspects of the algorithms proposed as well as the GPU architecture.

The algorithmic structure is outlined in Algorithm 1 to Algorithm 10. It is initiated in MATLAB in Algorithm 1, and moves into PC C code in Algorithm 2. Once the voting process is complete, the second order tensors at the votee sites are returned together with the skewness measure. MATLAB is used to display and manipulate the results into presentable formats.

Require: *image stack* $\wedge$ *votee positions* $\wedge$ σ $\wedge$ *case*
Ensure: *output tuple*$\wedge$ *graphics*
Create voteeTuple file
Invoke TensorVote *MEX file*
Read outputTuple file
Display orientation
Display skewness

Algorithm 1: MATLAB Front End

In Algorithm 2, the tensor voting process is entered. All the simulations are run with 32 voters and $f = 10000$. In this algorithm, the specified votee sites are segmented into sets of *voteen* voters based on the availability of GPU global memory. The largest component of memory is used by the storage area for all the ball vote first order tensors. These results are used both for the second and third order tensors and as such are not accumulated yet, with all iterations for all voters for all votees in the set stored. The closest 32 voters are calculated on the PC using getVoters, and are then transferred into GPU memory for FillTensorsGPU to operate on. When the GPU is finished, the results of the second order tensor votes at the votee sites as well as the skewness measures are transferred back into PC memory. The process is repeated until all the voting at the votee sites are completed.

Algorithm 3 determines the closest 32 voters to a specified votee site using an Euclidean metric. The *matching cone volume* constraints are also used to prevent voters on the same x, y plane from being chosen. The cone slope is normally chosen at 5 pixels/frame, which is adequate for the simulations and experiments being done.

The entry point in the GPU is FillTensorsGPU in Algorithm 4. Here the various parallel GPU algorithm kernels are launched with the required parameters. Each algorithm has its 3D threads defined by a *threadDim* call with the x, y, z dimensions specified. The grid blocks are similarly called using *gridDim* with the grid x, y dimensions specified. After each kernel launch, a GPU synchronise is executed to make sure all the results are completed and stored.

The GenTensorVote in Algorithm 5 uses the Ziggurat [40] random number generator with a seed derived from the thread number and the current timer value. This is used to generate random N

Require: *voteeTuple file* $\wedge$ σ $\wedge$ *case*
Ensure: *outputTuple file*
  $voteeTuples \Leftarrow voteefile$
  $majorvoteen \Leftarrow sizeof(voteeTuples)$
  $voteen \Leftarrow 0$
  $votern \Leftarrow 32$
  $count \Leftarrow 10000$
  $N \Leftarrow dimension(voteeTuple)$
  while *tensorTemp* will fit in GPU global memory and $voteen \leq majorvoteen$ **do**
    increase voteen
  $tensorTempGPU \Leftarrow GPUmalloc(voteen \times votern \times N \times count)$ {Used to hold 1st order tensors}
  $voteeTupleGPU \Leftarrow GPUmalloc(voteen \times sizeof(Tuple))$
  $outputTupleGPU \Leftarrow GPUmalloc(voteen \times sizeof(Tuple))$
  $voterTupleGPU \Leftarrow GPUmalloc(voteen \times votern \times sizeof(Tuple))$
  for $voteenumber = 0$ to $majorvoteen$ step $voteen$ **do**
    $voteeTupleGPU \Leftarrow portion(voteeTuple)$
    for $i = 0$ to $voteen$ **do**
      $voterTuple \Leftarrow$ getVoters {Only for the relevant votees}
    $voterTupleGPU \Leftarrow voterTuple$
    FillTensorsGPU
    $outputTuple \Leftarrow outputTupleGPU$ {Only for the relevant votees}
  $outputTuplefile \Leftarrow outputTuple$

Algorithm 2: TensorVote

Require: *voteeTuple* $\wedge$ *image stack* $\wedge$ σ $\wedge$ *case*
Ensure: *voterTuple*
  $voterList \Leftarrow []$
  for all $\mathbf{I}_{i,j,k}$ such that $\mathbf{I}_{i,j,k} \in matchingconevolume$ **do**
    $tuple \Leftarrow getTuple(\mathbf{I}_{i,j,k}, case)$ {Convert imagestack point to tuple}
    if $\|tuple - voteeTuple\| < max(voterList.distance)$ **then**
      $voterList[voterList.index] \Leftarrow tuple$
  $voterTuple \Leftarrow voterList$

Algorithm 3: GetVoters

Require: *voteeTupleGPU* $\wedge$ σ $\wedge$ *voterTupleGPU*
Ensure: *outputTupleGPU*
  $votees PerThread \Leftarrow 4$ {Fills the Multiprocessor}
  $threadDim \Leftarrow (votern, votees PerThread, 1)$
  $gridDim \Leftarrow (voteen/votees PerThread, 1)$
  genTensorVote {Launch the threads and blocks}
  threadSynch {Wait for them all to finish}
  $threadDim \Leftarrow (min(N, 20), min(N, 20), 1)$
  $gridDim \Leftarrow (N/20 + 1, N/20 + 1)$
  for $i = 0$ to *voteen* **do**
    collate2orderTensors {Launch the threads and blocks}
  threadSynch {Wait for them all to finish}
  $threadDim \Leftarrow (1, 1, 1)$
  $gridDim \Leftarrow (voteen, 1)$
  Eigen {Launch the threads and blocks}
  threadSynch {Wait for them all to finish}
  $maxThreads \Leftarrow maxthreads$ {memory constrained}
  $threadDim \Leftarrow (maxThreads, 1, 1)$
  $gridDim \Leftarrow (count/maxThreads, voteen)$
  FindSkewness {Launch the threads and blocks}
  threadSynch {Wait for them all to finish}
  $threadDim \Leftarrow (1, 1, 1)$
  $gridDim \Leftarrow (voteen, 1)$
  CollateSkewness {Launch the threads and blocks}
  threadSynch {Wait for them all to finish}

Algorithm 4: FillTensorsGPU

vectors on the unit N dimensional sphere. These are passed into the GenStickVote routine for tangential skew voting. The results are stored in the large area without accumulating the first order tensors. By doing this, the global memory latency is counteracted as no further calculations in this set of code rely on the result being stored – thus avoiding GPU multiprocessor stalls.

Require: *voteeLocation* $\wedge$ *voterLocation* $\wedge$ σ
Ensure: *tensorTemp*
  for *iterations* $= 0$ to f **do**
    $voterDir \Leftarrow randn()$
    $voterDir \Leftarrow norm(voterDir)$
    $tensorTemp++ \Leftarrow$ GenStickVote {Stores ND vectors sequentially}

Algorithm 5: GenTensorVote

The GenStickVote in Algorithm 6 implements Equation 3.4.2 with the direction given by Equation 3.5.1 to give the skew kernel depicted in Figure 3.12(c). In the GPU implementation, no early exits are allowed, and due to the possibility of denominators close to zero, checks need to be done as to whether a result is a real number or not. This routine is the inner computation of the tensor voting framework and contributes substantially to the computational load.

Require: $\mathbf{Q} =$ *voteeLocation* $\wedge$ $\mathbf{P} =$ *voterLocation* $\wedge$ $\sigma \wedge \hat{\mathbf{v}} =$ voterDir $\wedge$ *alphaFactor*
Ensure: *vote*
  $\mathbf{d} = \mathbf{Q} - \mathbf{P}$
  $l = |d|$
  $\cos(\alpha) = \hat{\mathbf{d}} \bullet \hat{\mathbf{v}}$
  $r = \frac{l}{2|\sin(\alpha)|}$
  $b = \frac{l}{2|\cos(\alpha)|}$
  $\hat{\mathbf{N}}_P = \hat{\mathbf{v}}\,|sin(\alpha)|$
  $\mathbf{C} = \mathbf{P} + (b\hat{\mathbf{N}}_P)$
  $\hat{\mathbf{u}} = (\mathbf{Q} - \mathbf{C})$
  $\theta = (\pi - \arccos(|\sin(\alpha)|)$
  $s = r\theta$
  $\mathbf{V}_{stick} = exp(-\frac{s^2 + alphaFactor * \theta^2/4}{\sigma^2})$
  $vote = \mathbf{V}_{stick}\hat{\mathbf{u}}$

Algorithm 6: GenStickVote

Once the unaccumulated first-order tensors are all in GPU global memory, the second-order tensor collation takes place where the outer product of the first-order tensor with itself $\mathbf{T} = \mathbf{a} \otimes \mathbf{a}$ results in a positive semidefinite second-order tensor. The second-order tensors are accumulated $\mathbf{A} = \sum \mathbf{T}$ over all voters and iterations to give a resultant second-order tensor $\mathbf{A}$ on which the eigen decomposition can take place for second-order feature extraction. This accumulation takes place in Collate2orderTensors in Algorithm 7.

A normal eigenanalysis is done on the second-order tensor $\mathbf{A}$ to determine both the eigenvectors $\hat{\mathbf{e}}_i$

Require: *tensorTemp*
Ensure: $eigenVector_{i,j}$
  $eigenVector_{i,j} \Leftarrow 0$
  for $i = 0$ to $f * votern$ **do**
    $t = tensorTemp[i]$
    $eigenVector_{i,j} \Leftarrow eigenVector_{i,j} + t_i * t_j$

Algorithm 7: COLLATE2ORDERTENSORS

and the eigenvalues λ_i of the second-order tensor. This is done in EIGEN in Algorithm 8.

Require: *eigenVector*
Ensure: $eigenVector \wedge eigenValue$
  $eigenVector, eigenValue \Leftarrow eig(eigenVector)$

Algorithm 8: EIGEN

The same set of first order tensors are used to find the accumulated third-order tensor $\mathcal{T}_{ijk} = a_i a_j a_k$ in Algorithm 9 . Due to space restrictions, the local storage of a $N \times N \times N$ third-order tensor is not desirable as memory is a scarce resource on the GPU. A simplification is made as the direction of the skewness has already been calculated and is the first eigenvector of the second-order tensor $\hat{\mathbf{e}}_1$. It is also known that the third-order tensor $\mathcal{T}_{ijk}$ is positive semidefinite which allows Equation 1.5.5 to be reduced to

$$S_{\mathbf{e}_1} = 6 \sum_{i=1}^{N} \sum_{j=i}^{N} \sum_{k=j}^{N} a_i a_j a_k e_{1_i} e_{1_j} e_{1_k}. \tag{4.2.1}$$

This simplification opens the implementation to the concept of skewness which would otherwise be difficult to implement, and would be computationally exhausting. The resulting skewness scalar is stored for each first-order tensor.

Require: *tensorTemp* $\wedge$ *eigenVector*
Ensure: *tensorTemp*
  $v \Leftarrow eigenVector_0$
  $skewness \Leftarrow 0$
  for $count = 0$ to $f * votern$ **do**
    $t = tensorTemp[count]$
    for $i = 0$ to N **do**
      for $j = i$ to N **do**
        for $k = j$ to N **do**
          $skewness \Leftarrow skewness + (t_i * t_j * t_k * v_i * v_j * v_k)$
    $tensorTemp_0 \Leftarrow skewness$

Algorithm 9: FINDSKEWNESS

Finally COLLATESKEWNESS in Algorithm 10 accumulates all the skewness measure scalars $S_{\mathbf{e}_1}$ for a particular votee.

Require: *tensorTemp*
Ensure: *skewness*
  $skewness \Leftarrow 0$
  for $count = 0$ to $f * votern$ **do**
    $t = tensorTemp[count]$
    $skewness \Leftarrow skewness + t_0$

Algorithm 10: CollateSkewness

4.2.3 Implementation results

The GPU architecture described has a single objective – to speed computation up in the high dimensional voting process. The voting algorithm code is compiled on both the PC CPU and the GPU for validation and performance evaluation. The two distinct time-consuming parts of the algorithm are the GetVoters part and the FillTensors part. The GetVoters part is not suitable for GPU implementation as there is a lot of decision logic surrounding the matching cone volume along with high memory requirements to store the whole spatio-temporal volume. This is kept on the PC CPU, and only the relevant voter tuples are passed into the GPU. In Figure 4.2, the voter selection is common to both the PC CPU and GPU. The PC CPU on which the simulations were run is a 3.0GHz Pentium 4 system with 3Gbytes of memory, while the GPU is the above-mentioned Nvidia GTX260 Graphics card. Both the implementations run code compiled for speed optimisation.

The performance in computation of the PC CPU and GPU is based on a 32 voter per votee problem with $f = 10000$, $\sigma = 10$, $\alpha = 1$, $k_{RGB} = 1$, $k_t = 1$ and $k_{xy} = 1$. Figure 4.2 plots the the time taken in seconds to complete the computation of one votee versus the dimensionality of the problem. The time taken to select voters in the PC CPU is superseded by the computation time at fairly low dimensionality, validating the idea of leaving the voter selection as a PC CPU-based activity. If the search area for the closest voters is opened up by increasing σ or decreasing the other parameters k_t, k_{RGB}, k_{xy}, then the voter selection time will increase as this process is done serially on the PC CPU.

The GPU code for the high dimensions is memory constrained, so the full number of multiprocessors are not used. Even so there is an improvement in runtime speed of the GPU compared to the PC CPU. This improvement ranges from five fold to seven fold at at higher dimensionality even though the clock speed of the GPU is less than the PC CPU. The computation speed could be improved more with the newer generation GPU cards that feature more on-board memory, as well as several GPUs on a card. The tensor problem is a largely uncorrelated parallel problem, and the GPU solutions will continuously increase the computing capability available at a reasonable cost.

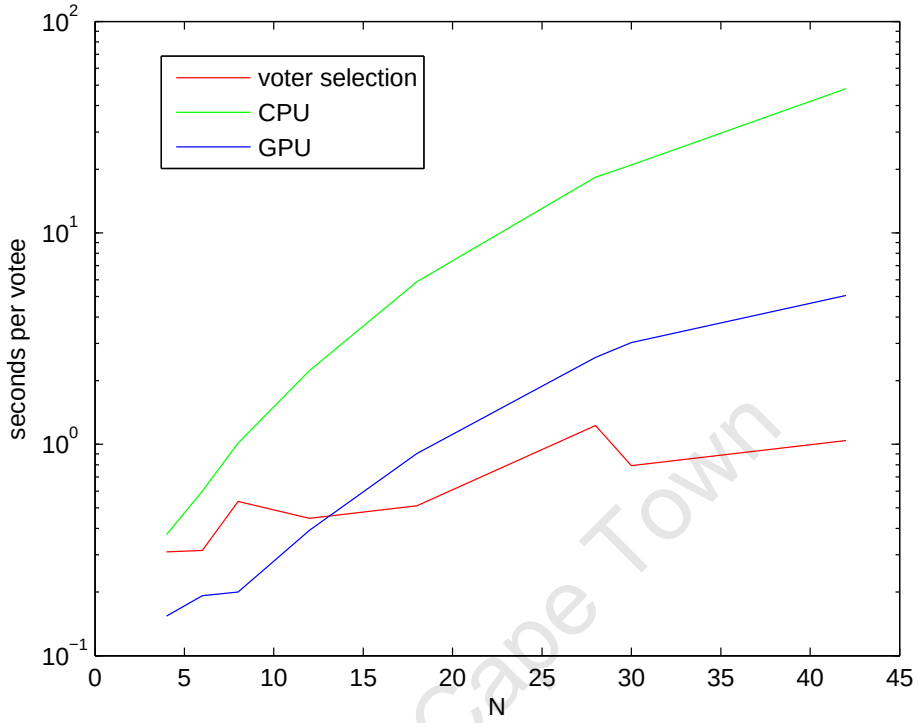


Figure 4.2: Processing performance comparison as a function of dimensionality between a PC CPU and GPU implementation of the same code.

The final GPU implementation has tried to glean the most out of the GPU capability by implementing the following points.

- *Minimal PC-GPU memory transfers.* Only the required input data and results are transferred between the PC and the GPU card as these transactions are expensive. In the tensor voting framework, only the applicable votees and voters are transferred as the voter selection occurs on the PC CPU and only the final results at the votee sites are returned to the PC.
- *Completely independent threads.* The threads may read common data but compute on independent transitional data and store results into independent memory locations.
- *Local thread data held in shared memory.* Due to the high latency of reading global memory, the input data required by the thread is read into shared memory only accessible by the thread.
- *Reuse of shared memory.* The scarce shared memory cannot be assigned and is not fully utilised. By reusing the memory the readability of the implementation suffers, but the number of threads can be maximised.

- *Maximising the number of threads.* Based on the required memory per thread, the maximum number of threads are launched. In the tensor voting framework, the threads are segmented based on a single votee-voter relationship. Thirty two threads cater for all the voters per votee. The set of 32 can be duplicated by doing several votees concurrently. The memory usage per thread is heavily reliant on the dimensionality of the problem and is the thread limiting factor. The achieved figures are shown in Figure 4.3. At the low dimensionalities the number of threads is limited by the GPU device maximum number of threads. As $N > 20$ the number of threads starts to collapse due to global and shared memory constraints.

- *Lengthy global memory write sequences.* The tensor voting Algorithm 5 writes the first-order tensor results into large sections of global memory dimensioned according to f, where $f = 10000$. Care is taken that the memory is non-overlapping between threads and that the algorithm only writes to the memory.

- *Optimal use of registers.* The CUDA environment is tuned to use registers (taken from shared memory) using compile directives instead of local memory which is slow. All the array memory needed for the thread is assigned at thread launch level. The CUDA environment does not seem to compile arrays at lower call levels into shared memory.

- *Buffered global memory reads.* When large sections of the global memory need to be read, a shared memory buffer is assigned that can take several data points at a time to minimise GPU memory stalling. Buffered reading takes place when the first-order tensors are collated in Algorithm 7 and Algorithm 9.

The PC CPU implementation uses the same code base as the GPU code except that all actions are serialised. The PC CPU code has a memory allocation limitation governed by the MATLAB MEX interface in use with the PC CPU code. As soon as the required memory on the PC CPU application side exceeds the amount requested by MATLAB from the operating system, the PC CPU applications exits to MATLAB with a memory allocation error. With the small runs done during the thesis this limitation did not pose a large problem.

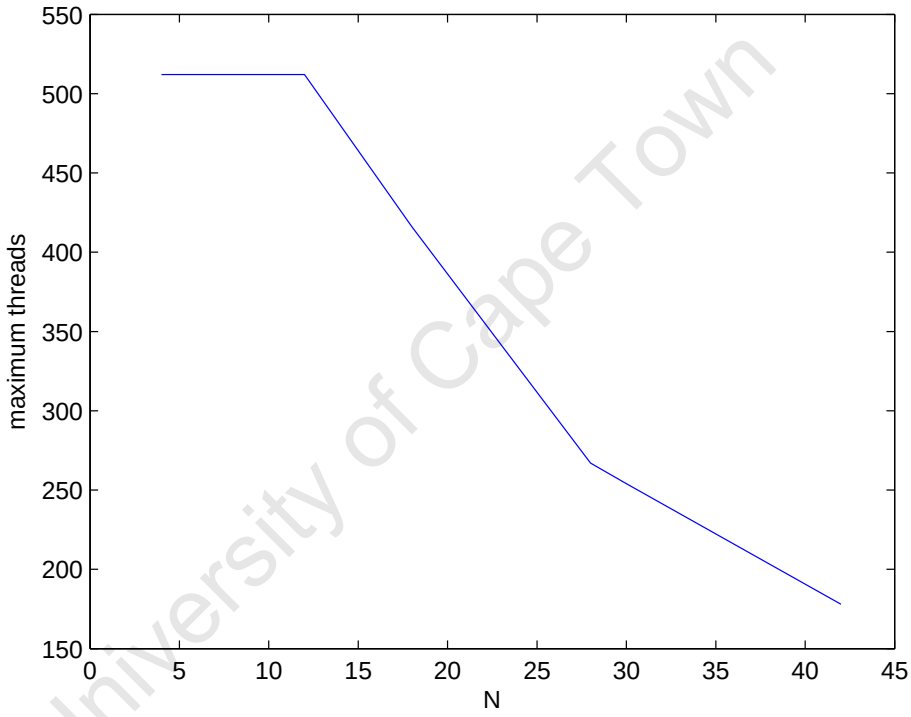


Figure 4.3: Maximum number of threads launched on GPU as a function of dimensionality.

4.3 Implementation Challenges

While implementing the high-dimensional tensor voting on the GPU, several implementation challenges arise. These challenges are usually very difficult to solve due to the unforgiving nature of the GPU, and inability to debug properly in the GPU environment. There are also several problems in the compiler which can be expected in the infancy of any compiler. Implementation problems were very seldom highlighted by the PC emulation of the GPU due to:

- The inability of the PC to deal efficiently with more than 20 threads at a time. In the GPU environment, the number of parallel threads reaches to 512.
- The subtle interaction between threads on memory conflicts and synchronisation.
- The emulation uses the PC C compiler – and as such the GPU compiler issues are not present.
- The emulation is not as resource constrained as the GPU card is, and does not exhibit memory depletion problems.

This led to *black box* testing where the GPU code is evaluated on outputs. The GPU outputs are compared to PC generated outputs for small simulation runs. Implementation problems were then found by isolating segments of code until the outputs from the GPU and PC compare correctly. Debugging in this fashion is an extremely time consuming process as the simulation runs need to be large enough to capture problems that do not occur often.

The implementation revealed several problems in the CUDA environment:

- *Compound statements.* Compound statements, especially with pre- and post-decrement or increment can cause errors if the variable affected is used in the next statement.
- *Conditionals.* Conditionals are generally not used in CUDA as this stalls the threads until all threads reach the same execution point. Errors were traced to conditionals inside the innermost loops. The code was restructured to rather complete without early exits.
- *Non-finite numbers.* In the case of non-finite numbers, CUDA propagates their presence. In tensor voting this is disastrous as the final answer will ends as non-finite as is typically the case when accumulating Monte Carlo results.
- *Incorrect type conversion.* In the case of converting from an integer to a double precision number, the GPU makes use of non-standard rounding. In some cases the only workaround is to make all the number double precision from the start.

- *Shared memory conflicts.* It seems that when the number of threads is small, the multiprocessor launches other blocks before a thread block is complete, allowing contamination of shared memory. This problem manifests as a locked computer.
- *Inadequate critical sections.* CUDA provides a few atomic instructions that allow a thread to complete a memory transaction before another thread can access the memory being changed. Unfortunately there are not any that support double precision floating point which is mandatory in the tensor voting framework to maintain Monte Carlo accuracy.

The GPU hardware also exhibits a major problem in the Windows operating system environment. When computation on the GPU exceeds 10 seconds without returning to the host program, the Windows operating system deems the display driver non-functional. Under these conditions the Windows operating system terminates non-functional drivers which leads to an operating system reset, or a general display hang-up. The only recovery is through a hard reset on the computer.

As has been described, development in the GPU field is still fairly hostile, but as evidenced by the advent of boxes containing GPUs for computational purposes only (Nvidia Tesla architecture), they will become more workable in the future.

Chapter 5

Experiments and measurements

This chapter looks into practical results of the proposed technique, and attempts to quantify the results. Use is made of known image sequences to be able to compare the results with ground truth. Although the main effort is not motion estimation, use is made of the Yosemite motion estimation problem as the ground truth is known. The full algorithm is applied to the Mobile Calender sequence and the Flower Garden sequence to determine its potential. The results on the natural sequences are discussed.

5.1 Introduction

In the previous chapter the GPU implementation of the skew tangential voting framework in higher dimensions was addressed. The GPU implementation of the voting framework gave a speed improvement of up to seven times compared to using a normal PC CPU.

In this chapter the skew tangential voting framework is applied to the *Yosemite* sequence to determine the performance of the algorithm on estimating the motion vectors. The results are compared to other authors results and are discussed.

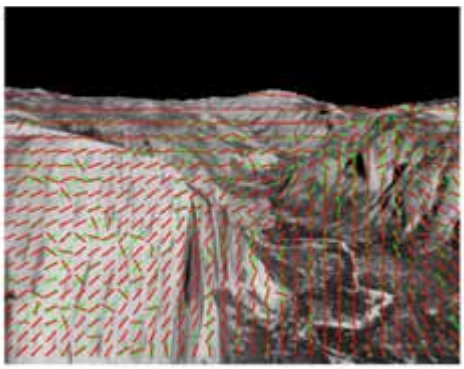
The skew tangential voting framework is then applied to the *Mobile Calender* sequence. The left hand and right hand movement detectors are used to determine the motion boundaries and this is compared with ground truth.

The voting framework is applied to the *Flower Garden* sequence with some analysis on the choice of parameters. The simplified stick vote method is also applied to the *Flower Garden* sequence and the results commented on.

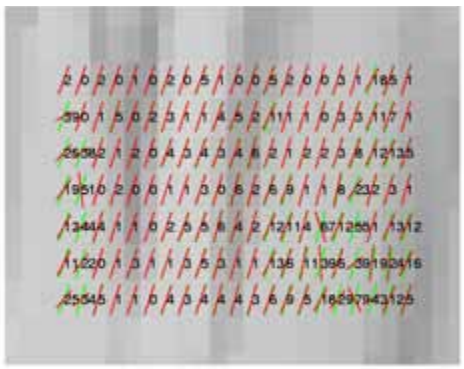
5.2 Motion vector estimation for the *Yosemite* sequence

Although the techniques being developed are for motion segmentation, motion vector estimation occurs automatically as the projection of the second-order tensor first eigenvector $\hat{\mathbf{e}}_1$ on the x, y plane. The *Yosemite* sequence is a useful synthetically generated test sequence used by many researchers to measure the performance of motion vector estimation algorithms. The sequence is 30 frames long, and the central frame is analysed allowing all the preceding and following frames to be used. Most researchers ignore the sky as the clouds exhibit non-rigid motion, and for purposes of comparison the sky is ignored in these results as well.

The tensor encoding uses case 8 encoding resulting in $N = 28$. The sequence is a gray scale image, so no colour information is encoded in the tensors. The tensor vote is a single pass ball vote using a GPU with 32 voters per votee, $\sigma = 10$, $\alpha = 1$, $k_{RGB} = 1$, $k_t = 1$, $k_{xy} = 1$ and $f = 10000$. The run time for the frame was 24hrs on the GPU. The results obtained are $12.1° \pm 17.3°$ for a 100% image coverage. The results in Figure 5.1 are not as good as Nicolescu and Medioni [49] for 100% coverage but it must be remembered that the vote was a single pass ball vote, and no saliency censorship and densification took place — all the results are from actual data. It can also be seen in Figure 5.1(b) that the errors seem to cluster. Looking at Figure 5.1(c) the error map can be seen. More prominant red areas indicate high errors showing that high errors occur in the darker area due to a high likelihood of aliasing. The high errors could be counteracted by increasing the number of



(a) Decimated motion vector field.



(b) Detail of motion vector field.

(c) Detail of motion motion vector error field.

Figure 5.1: *Yosemite* sequence where green represents the ground truth, red represents the tensor voting result and the numbering indicates the angular error in degrees between ground truth and the tensor voting result. The error is shown with high red value indicating a large angular error.

adjacent pixels in the tensor encoding at the cost of crisply detecting motion edges. In the *Yosemite* sequence the predominant flow is smooth without many edges and is not always representative of normal natural images such as the *Mobile Calender* or *Flower Garden* sequences.

Table 5.1: *Yosemite* motion vector results. The thesis result is in bold.

Technique	Average error	Standard deviation	Density
Farnebäck [13]	1.14°	2.14°	100%
Nicolescu and Medioni	3.74°	4.3°	100%
Gaucher and Medioni [16]	8.83°	10.6°	100%
Guest($\sigma = 20$,case 8)	**12.1°**	**17.3°**	**100%**
Anandan	15.54°	13.46°	100%
Uras et al. (unthresholded)	16.45°	21.02°	100%
Horn and Schunck	22.58°	19.73°	100%
Lucas and Kanade($\lambda_2 \geq 5.0$)	3.55°	7.11°	8.8%
Uras et al. ($\det H \geq 2.0$)	3.75°	3.44°	6.1%
Fleet and Jepson ($\tau = 2.5$)	4.29°	11.24°	34.1%
Fleet and Jepson ($\tau = 1.25$)	4.95°	12.39°	30.6%
Lucas and Kanade($\lambda_2 \geq 1.0$)	5.20°	9.45°	35.1%
Uras et al. ($\det H \geq 1.0$)	5.97°	11.74°	23.4%
Heeger	11.74°	19.0°	44.8%

5.3 Motion boundary detection in the *Mobile Calender* sequence

The *Mobile Calender* sequence is a well known test sequence used in evaluating moving object segmentation. This sequence also has a ground truth sequence against which to evaluate. The test sequence consists of 40 interlaced RGB frames with ground truth on every tenth frame. Due to a small time difference in the odd and even fields of the test sequence, the full image used in this analysis uses only the even field, and interpolates the missing lines with a bicubic interpolation. Due to the high complexity of the *TS map* extraction, only a small section of frame 20 of the sequence is evaluated over the train section.

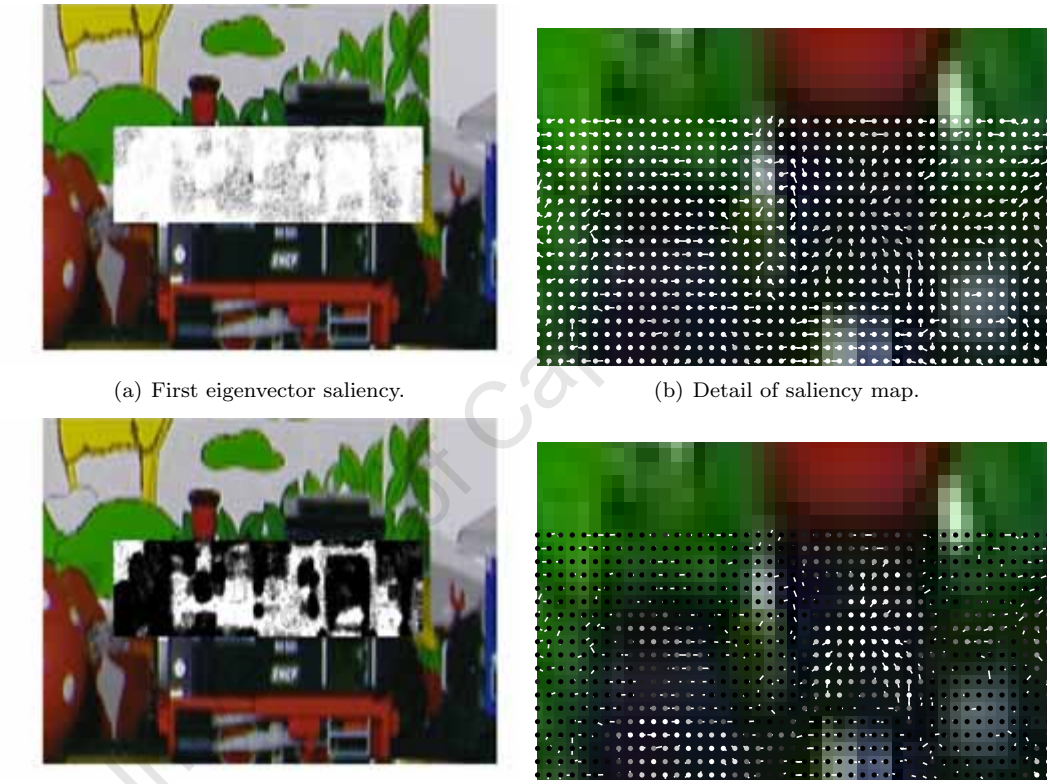
The TS map over the train is determined as outlined previously, with slightly different parameters. Only Case 3 and case 6 are evaluated as the ideal *tissue earth* sequence indicated that these cases present good motion vector estimation, as well as good skewness measure performance for detecting moving edge boundaries in Section 3.10. The progressive results for case 3 are shown in Figure 5.2, and the results for case 6 are shown in Figure 5.5.

For case 3, a lower scale factor $\sigma = 10$ is chosen as the occlusion and disocclusion distances are small in places and an increase in the σ factor increases the influence of voters further away, corrupting estimates across motion boundaries. The saliency is generally very high, and drops on the engine body due to homogeneous colour. The motion vector orientations in Figure 5.2(b) display chaotic behavior due to the homogeneous regions, but the TS map seems to display structure with the underlying image in Figure 5.2(c).

When the left hand and right hand movement filters in Figure 5.3 are used with a filter width of $\eta = 2$, the outputs of the filters in Figure 5.3(g) and Figure 5.3(c) still manage to show some moving edge structure. It seems as if the detection points have found their way into the object, due to the indistinct motion edges (smearing). It must be remembered that the algorithm has no edge detection aspect — it only relies on motion.

Combining the left hand and right hand movement detections, the result is compared to the ground truth image in Figure 5.4. As noted, the detections fall within the boundary of the object. It can also be seen that the ground truth is an approximation of the actual boundaries, and would make a poor evaluation tool for segmentation.

Case 6 is run with a higher scale factor $\sigma = 20$ as the dimensionality of the problem is higher, and Euclidean distances in this higher dimension are commensurately greater. Case 6 saliency and the TS map in Figure 5.5 show improvement when compared to case 3, although the homogeneity problem is still present on the train. The TS map and motion vector orientations in Figure 5.5 do



(a) First eigenvector saliency.

(b) Detail of saliency map.

(c) TS map.

(d) Detail of TS map.

Figure 5.2: Case 3 results with $\sigma = 10$, $\alpha = 10$, $k_{RGB} = 0.5$, $k_{xy} = 1$, $k_t = 1$ and $f = 10000$.

(a) Left hand movement filter with $\eta = 3$ applied to skewness measure in x direction.

(b) Detail of left hand movement filter result.

(c) Detection at 50% of left hand movement filter.

(d) Detail of left hand movement filter detection.

(e) Right hand movement filter with $\eta = 3$ applied to skewness measure in x direction.

(f) Detail of right hand movement filter result.

(g) Detection at 50% of right hand movement filter.

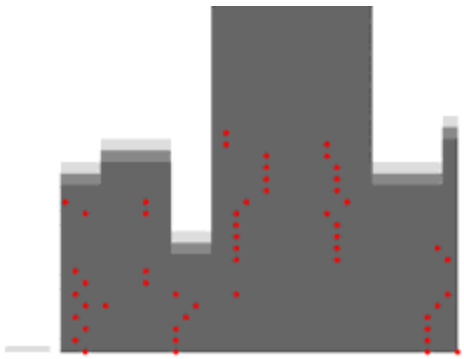
(h) Detail of right hand movement filter detection.

Figure 5.3: Case 3 TS Map passed through a left hand and right hand movement filter with detections shown.

(a) Combination of left hand and right hand movement detections.

(b) Detail of combination of left hand and right hand movement detections.

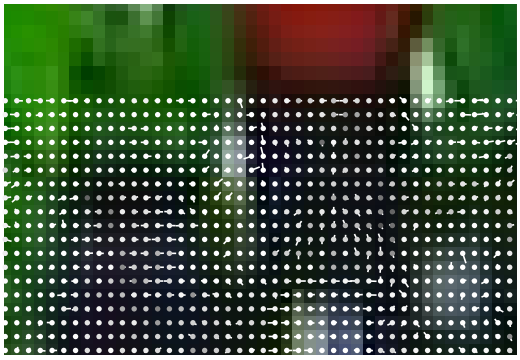
(c) Left hand and right hand movement detections superimposed on ground truth.



(d) Detail of left hand and right hand movement detections superimposed on ground truth.

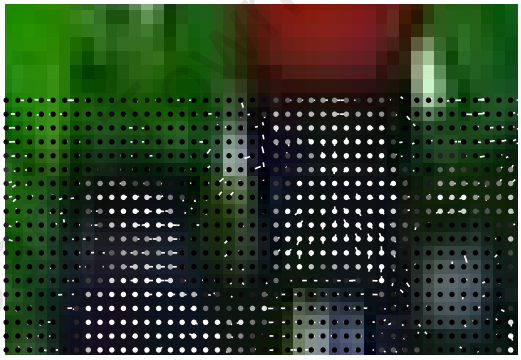
Figure 5.4: Case 3 full detection results. Both occluding and disoccluding detections are shown and compared to ground truth.

(a) First eigenvector saliency.



(b) Detail of saliency map.

(c) TS map.



(d) Detail of TS map.

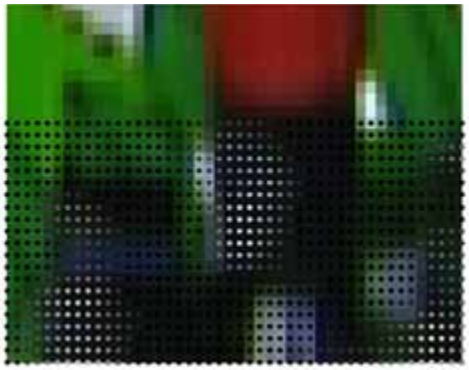
Figure 5.5: Case 6 results with $\sigma = 20$, $\alpha = 10$, $k_{RGB} = 0.5$, $k_{xy} = 1$, $k_t = 1$ and $f = 10000$.

look more consistent than in case 3.

The detection of occluding and disoccluding boundaries in case 6 are shown in Figure 5.6. The edge points are still within the boundaries of the object, but seem more consistent with motion boundaries.

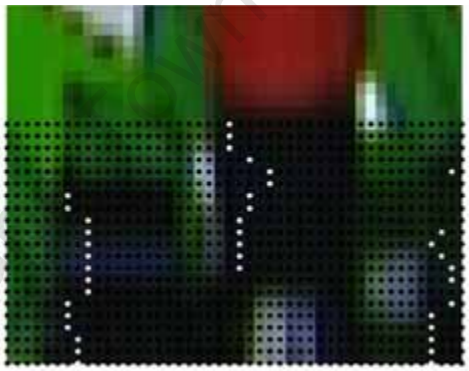
The occluding and disoccluding boundaries are compared with ground truth in Figure 5.7, and show improvement over case 3. It also seems that some of the oblique boundaries are found as well. The detail section only picks out a small portion of the region of interest. Toward the cab of the train, the method has managed to pick out small openings that allow the background to be visible. This is not noted in the ground truth, but are valid moving object edges.

(a) Left hand movement filter with $\eta = 3$ applied to skewness measure in x direction.



(b) Detail of left hand movement filter result.

(c) Detection at 50% of left hand movement filter.

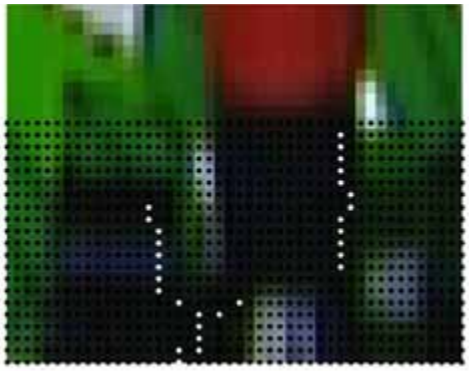


(d) Detail of left hand movement filter detection.

(e) Right hand movement filter with $\eta = 3$ applied to skewness measure in x direction.

(f) Detail of right hand movement filter result.

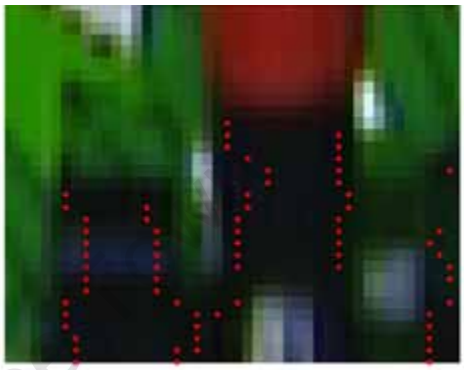
(g) Detection at 50% of right hand movement filter.



(h) Detail of right hand movement filter detection.

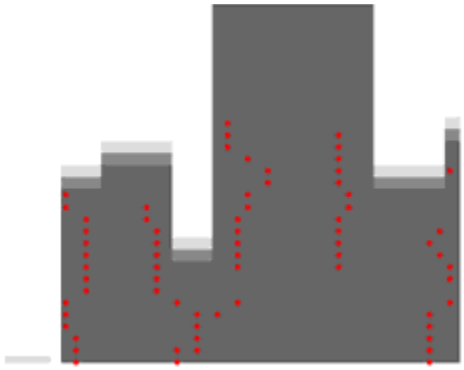
Figure 5.6: Case 6 TS Map passed through a left hand and right hand movement filter with detections shown.

(a) Combination of left hand and right hand movement detections.



(b) Detail of combination of left hand and right hand movement detections.

(c) Left hand and right hand movement detections superimposed on ground truth.



(d) Detail of left hand and right hand movement detections superimposed on ground truth.

Figure 5.7: Case 6 full detection results. Both occluding and disoccluding detections are shown and compared to ground truth.

(a) Frame 1. (b) Frame 30.

Figure 5.8: *Flower Garden* sequence showing the tree region tracked to give $v_x = -9.8$ pixels/frame and $v_y = -0.6$ pixels/frame.

5.4 Motion boundary detection in the *Flower Garden* sequence

The Flower Garden sequence is a well known test sequence used in evaluating moving object segmentation. A portion containing the tree trunk is used as this will display horizontal occlusion and disocclusion. The sequence is 40 frames long, and no ground truth is available. Even though ground truth is not available, the edges of the tree trunk are well defined. The tree trunk is slightly out of focus which makes the interpolation problem greater. In order to try quantify the parameters used, a portion of the tree trunk is manually tracked over 30 frames to determine the tree trunk motion vector and is shown in Figure 5.8. The flowers are also manually tracked in Figure 5.12.

5.4.1 Ball vote boundary detection in the *Flower Garden* sequence

Normal ball voting with case 6 tensor encoding is applied to a small section of votees within the tree trunk and the motion vector angles compared to the reference values of $v_x = -9.8$ pixels/frame and $v_y = -0.6$ pixels/frame as shown in Figure 5.9. Ball voting is conducted for various parameter values and the mean and standard deviation error values (using the same calculation as was used in the Yosemite results) are shown in Figure 5.10. The k_{xy} parameter is set at $k_{xy} = 1$ as the other parameters are believed to be relative to each other. The angular errors have local minima for all the σ values, and tend to occur at low k_{RGB} for low σ and as σ increases — the minimum moves to higher k_{RGB} as expected due to the coupling of the global parameter σ to the other parameters k_t and k_{RGB}. The local minima seen may be global minima as higher values of the parameters do not work well at all. From Figure 5.10, the best mean (10°) and standard deviation (7°) occurs

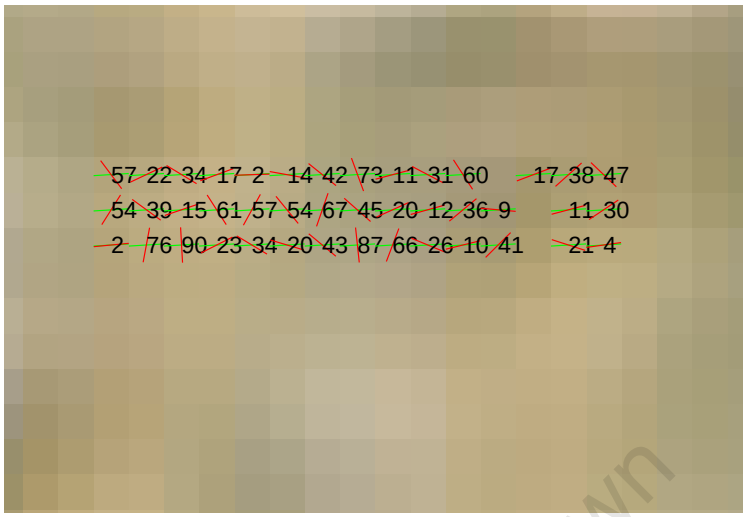


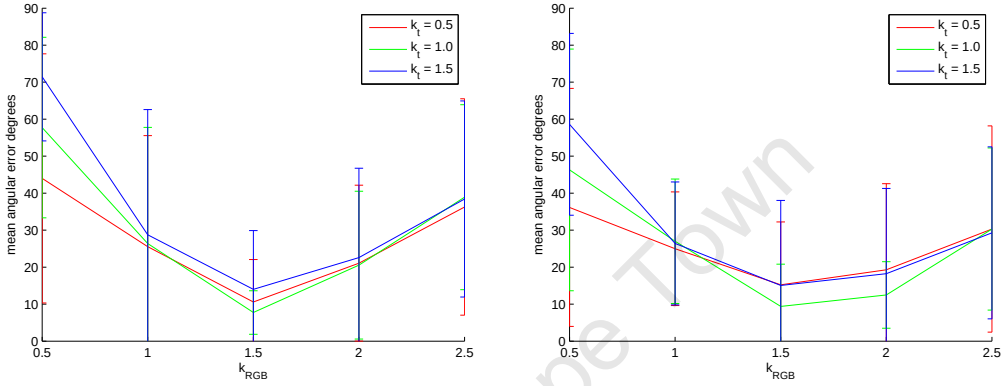
Figure 5.9: *Flower Garden* sequence detail of trunk showing the ground truth (green) and the case 6 ball vote results (red). Each votee site also has the angular error in degrees.

at $\sigma = 10$, $k_{RGB} = 1.5$ and $k_t = 1$. It must be remembered that the parameters have only been tested on a small portion of the trunk of the tree — and should be tested more comprehensively to confirm this choice. Due to the high computational load to do tests, the values obtained are deemed representative enough.

The same test of varying the parameters is run again using pre-alignment of 10 of the voters per votee to see if there is any noticeable improvement on the motion vector angular error in Figure 5.11. For the most part the results look similar comparing Figure 5.11 to Figure 5.10. The best mean (7°) and standard deviation (3°) occurs when the parameters are set at $\sigma = 10$, $k_{RGB} = 1.5$ and $k_t = 1.5$. This result is better than the ball vote, but comes at a large computational cost. The time used to compute the graphs for pre-alignment is 5 hours compared to 15 minutes for the ball votes. The computation only occurs on a small segment of the image frame. To use pre-alignment for a large portion of the frame yields a run time of several days and is not feasible within the constraints of the thesis.

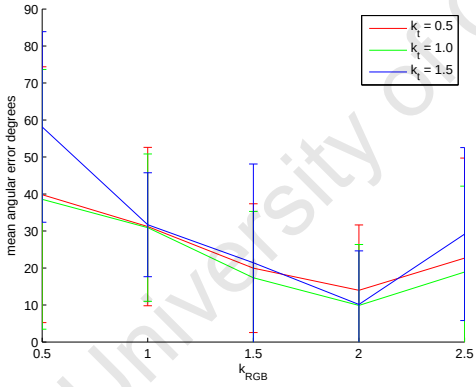
The ball voting is repeated on a segment of the flowers in the garden as shown in Figure 5.12. The manually determined velocities of the flower region are $v_x = -2.5$ pixels/frame and $v_y = -0.2$ pixels/frame. Ball voting is conducted in the same fashion as is done for the tree region and are shown in Figure 5.10.

In an effort to balance the parameters between the two moving objects, a compromised parameter selection of $\sigma = 20$, $k_{RGB} = 1.5$ and $k_t = 1.5$ is chosen and a ball vote is applied to a portion of the *Flower Garden* sequence in a similar fashion was done with the *Mobile Calender* sequence in the

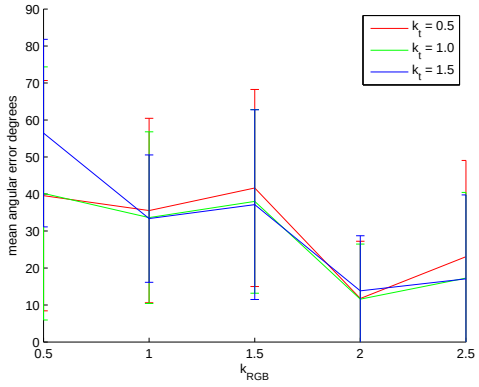


(a) $\sigma = 10$

(b) $\sigma = 20$

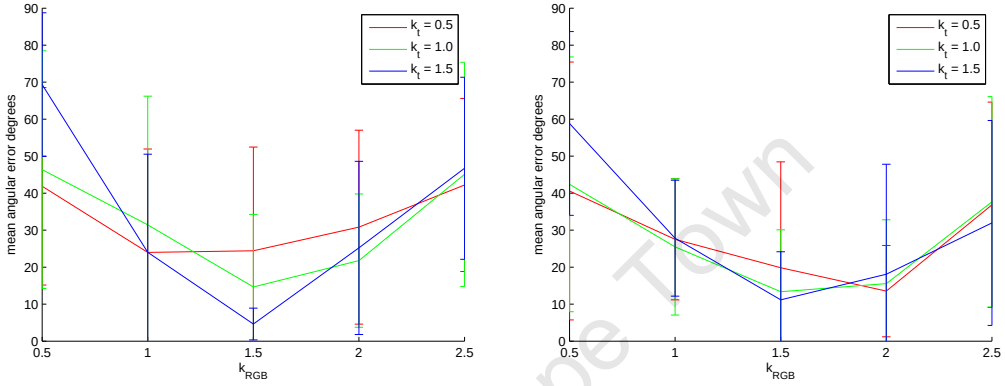


(c) $\sigma = 30$



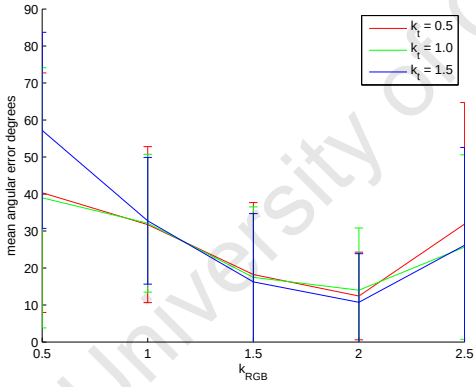
(d) $\sigma = 40$

Figure 5.10: Error bars on the average angular errors on the *Flower Garden* sequence tree trunk (case 6, ball voting) for various k_t, k_{RGB} and σ. The parameter $k_{xy} = 1$.

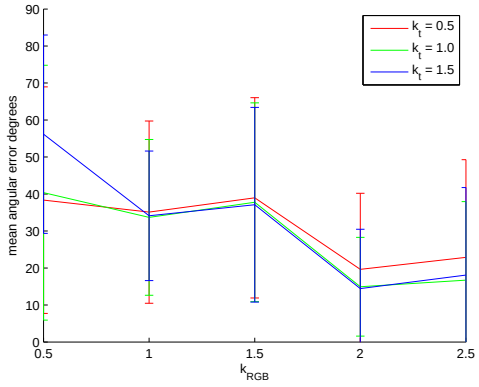


(a) $\sigma = 10$

(b) $\sigma = 20$

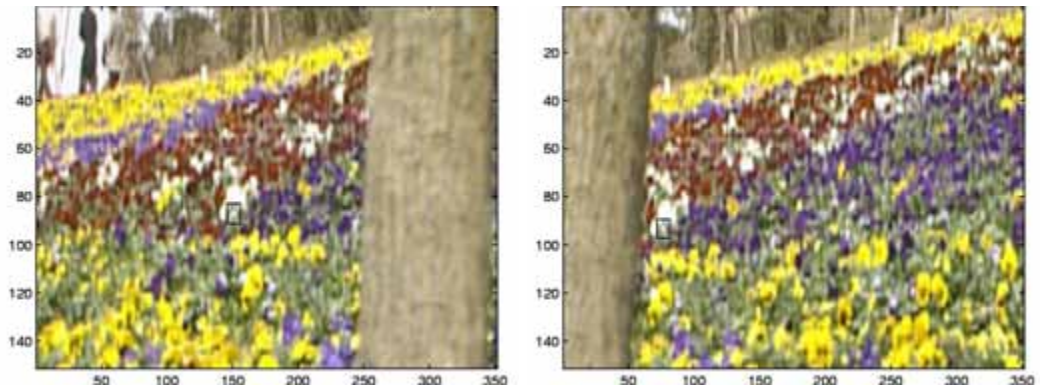


(c) $\sigma = 30$



(d) $\sigma = 40$

Figure 5.11: Error bars on the average angular errors on the *Flower Garden* sequence tree trunk (case 6, pre-alignment voting) for various k_t, k_{RGB} and σ. The parameter $k_{xy} = 1$.



(a) Frame 1. (b) Frame 30.

Figure 5.12: *Flower Garden* sequence showing the flower region tracked to give $v_x = -2.5$ pixels/frame and $v_y = -0.2$ pixels/frame.

previous section.

Even with the careful selection of parameters, the left and right hand movement detections shown in Figure 5.15 are poor. There are several reasons for the poor performance.

- *Lack of detail in tree trunk.* Close investigation of the tree trunk shows large homogeneous and aliased regions causing poor voter selection, especially with limited spatial matching given by case 6 encoding.
- *Rapid movement.* The tree trunk moves rapidly across the scene affecting the choice of parameters to cater for large movement between frames. By allowing for the rapid movement the parameters allow voter selection in aliased areas more easily.
- *Aliasing in the flowers.* The flowers are repetitions of themselves allowing poor voter selection based on aliasing.

The main counteraction of the reasons resulting in poor performance is to increase the number of pixels encoded in the tensor, which is done in the following section.

5.4.2 Simplified stick vote boundary detection in the *Flower Garden* sequence

The simplified stick vote is applied to a portion of the *Flower Garden* sequence to see if the method yields better results in finding boundaries. The simplified stick vote allows higher dimensionality

Figure 5.13: *Flower Garden* sequence detail of flowers showing the ground truth (green) and the case 6 ball vote results (red). Each votee site also has the angular error in degrees.

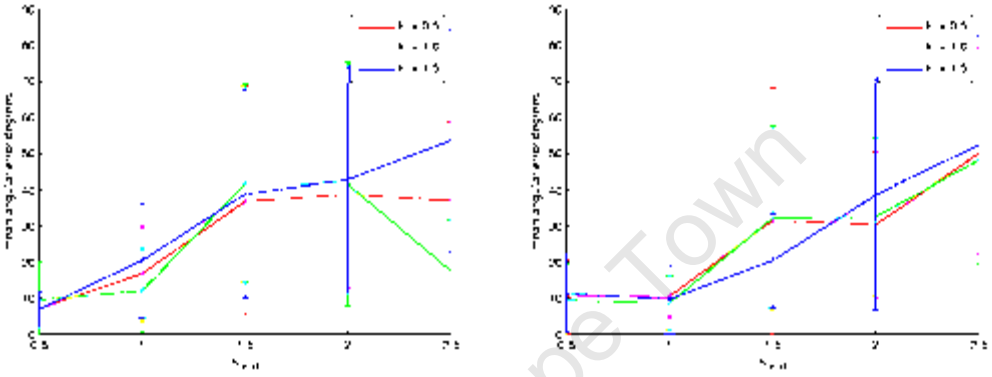
voting to be used, allowing more pixels to be incorporated in the tensor voting framework to counteract some of the aliasing and homogeneity issues in the *Flower Garden* sequence encountered with normal ball voting.

The parameter k_{xy} is made small to make the rapid tree trunk movement less important in determining voter selection, and the colour parameter k_{RGB} is made larger to try separate colour better. The simplified stick vote is a simple procedure where each voter votes once along the connecting line between the voter and votee. The strength of the vote is determined by Euclidean distance. The Monte Carlo run is greatly simplified as $f = 1$ so the dimensionality can be increased.

The simplified stick vote is first run on the case 8 tensor encoding in Figure 5.16 with a dimensionality $N = 30$, and then on a case 10 tensor encoding in Figure 5.17 with a dimensionality of $N = 72$.

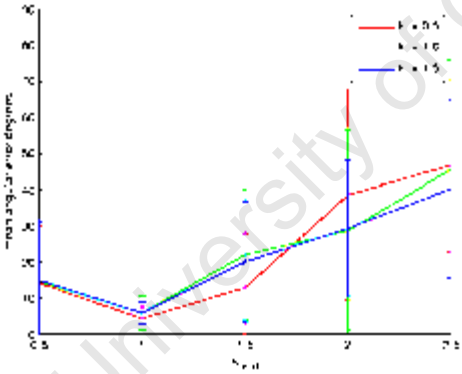
The simplified stick vote applied to the case 8 tensor encoding in Figure 5.16 is able to detect the disoccluding boundary well but detects many false boundaries as well. The performance of the occluding detector is partial with even more false boundaries detected. The false boundaries can be cleaned up in a post processing step using line or surface tensor voting.

Increasing the dimensionality to case 10 tensor encoding in Figure 5.17 yields a clear disoccluding boundary with few false boundaries, but the occluding detection does not pick up the occlusion edge well at all. It is believed that more comprehensive tensor encodings incorporating more adjacent pixels may improve this result.

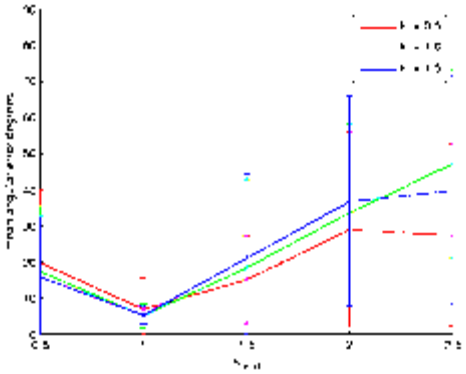


(a) $\sigma = 10$

(b) $\sigma = 20$



(c) $\sigma = 30$

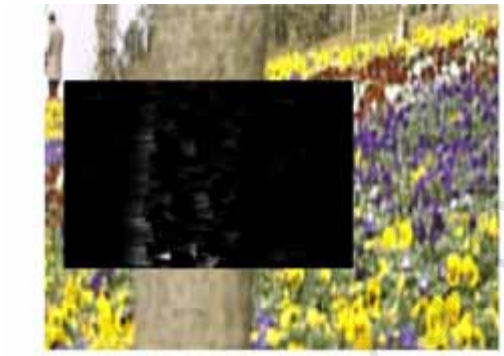


(d) $\sigma = 40$

Figure 5.14: Error bars on the average angular errors on the *Flower Garden* sequence flower garden (case 6, ball voting) for various k_t, k_{RGB} and σ. The parameter $k_{xy} = 1$.

(a) First eigenvector saliency.

(b) Skewness measure.



(c) Left hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

(d) Right hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

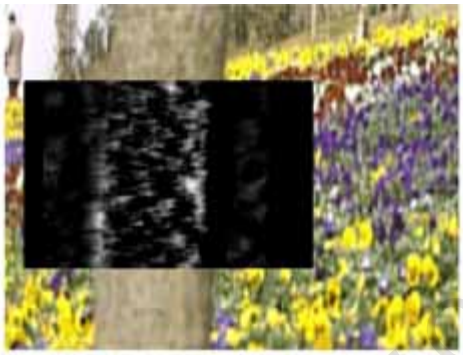
(e) Detection at 50% of left hand movement filter.

(f) Detection at 50% of right hand movement filter.

Figure 5.15: Case 6 ball vote applied to a section of the *Flower Garden* sequence. Parameters used are $f = 10000$, $\sigma = 10$, $\alpha = 1$, $k_t = 1$, $k_{RGB} = 2$ and $k_{xy} = 1$.

(a) First eigenvector saliency.

(b) Skewness measure.

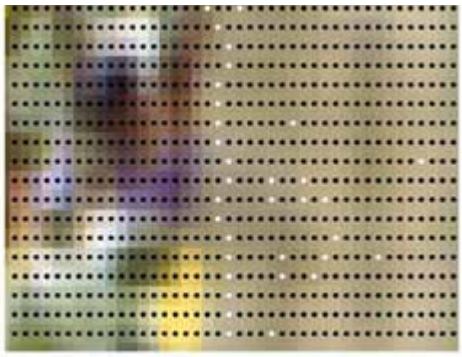


(c) Left hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

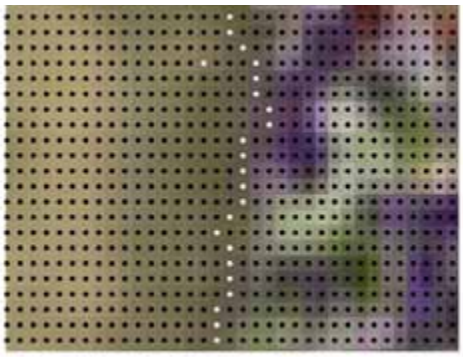
(d) Right hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

(e) Detection at 50% of left hand movement filter.

(f) Detection at 50% of right hand movement filter.



(g) Detail of left hand movement detection.



(h) Detail of right hand movement detection.

Figure 5.16: Case 8 simplified stick vote applied to a section of the *Flower Garden* sequence. Parameters used are $\sigma = 10$, $\alpha = 1$, $k_t = 1$, $k_{RGB} = 2$ and $k_{xy} = 0.1$.

(a) First eigenvector saliency.

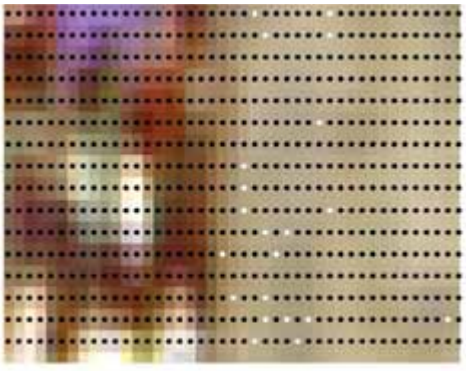
(b) Skewness measure.

(c) Left hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

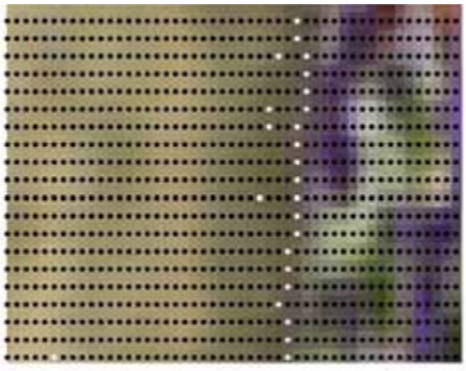
(d) Right hand movement filter with $\eta = 4$ applied to skewness measure in x direction.

(e) Detection at 50% of left hand movement filter.

(f) Detection at 50% of right hand movement filter.



(g) Detail of left hand movement detection.



(h) Detail of right hand movement detection.

Figure 5.17: Case 10 simplified stick vote applied to a section of the *Flower Garden* sequence. Parameters used are $\sigma = 10$, $\alpha = 1$, $k_t = 1$, $k_{RGB} = 1$ and $k_{xy} = 0.1$.

5.5 Discussion on parameter choice

Even though the objective is to reduce the number of free parameters, there are several parameters that need to be chosen for the skew tangential framework. Most can be chosen based on the underlying motion characteristics of the sequence.

- σ *parameter*. The σ parameter is the overall scale parameter. The parameter determines the "reach" of the voter. Higher values leads to voters further away affecting the outcome of the voting process at the votee. The high dimensionality of the tensors separates the tokens more, and σ needs to be increased generally as the dimensionality increases. A general rule empirically found is that an increase in dimension of 10 needs and increase in σ of 5. In highly aliased sequences the value is reduced to allow closer voters to outweigh further voters. Normal values are $10 \leq \sigma \leq 30$.

- α *parameter*. The α parameter is the "pointiness" parameter. This parameter only comes into play when directed voting is being done — otherwise its affect is minimal. In the simulations this parameter is usually $\alpha = 1$ or $\alpha = 10$.

- k_t *parameter*. The time or z scale parameter allows the effect of tokens from image frames further away to be accentuated or suppressed. Increasing k_t will attenuate the effect of voters from frames far away from the votee frame. This parameter is usually $k_t = 1$.

- k_{RGB} *parameter*. The colour scale parameter is useful in accentuating or suppressing the RGB elements of the tensor. In homogeneous areas the effect of slightly different colours can be accentuated by increasing k_{RGB} while still keeping desired spatial scales. Normal values are $1 \leq k_{RGB} \leq 2$.

- k_{xy} *parameter*. The xy scale parameter is related to the maximum expected motion in a sequence. In the *Flower Garden* sequence the trunk moves at almost 10 pixels/frame. To counteract this the xy scale parameter is set to $k_{xy} = 0.1$. In block matching motion estimation methods there is no penalty on matching blocks with great xy offset — as long as the blocks are within the search area. Similarly, k_{xy} can make all the tokens in the matching cone volume have little (but still finite) Euclidean distance penalties based on xy offset. Normal values are $0.1 \leq k_{xy} \leq 1$.

- η *parameter*. The left hand and right hand movement filter decay constant is matched to the σ value chosen. Observation of results indicate that the occlusion and disocclusion detection results are fairly robust with $2 \leq \eta \leq 4$.

5.6 Summary

In this chapter we have applied the skew tangential voting framework on portions of the *Yosemite* sequence, the *Mobile Calender* sequence and the *Flower Garden* sequence.

In the *Yosemite* sequence the motion vector angular error is determined and compared to several results in the literature. The performance of the algorithm is not very good in highly aliased areas where the angular errors peak, but in other more smooth areas the angular errors are low.

In the *Mobile Calender* sequence and the *Flower Garden* sequence the ability of the left hand and right hand movement detectors to detect the motion boundaries is variable from failure to fairly good. The simplified stick vote shows good performance on the *Flower Garden* sequence in detecting disocclusion but fails in detecting occlusion. This may be due to the aliasing problem with the *Flower Garden* sequence as well as the very fast movement of the tree trunk always allowing voters on either side of the votee and, in so doing, counteracting the skewness measure.

A more comprehensive method of optical flow evaluation is presented by Baker et al [1]. The evaluation method looks not only into angular error as used in the thesis on the Yosemite sequence, but also several other methods of evalauating optical flow accuracy such as motion vector end-point errors and sum of squared differences over the frame. Baker et al have prepared various test sets with novel methods of determining the ground truth optical flow and make the evaluation tool available over the internet to allow up-to-date performance comparisons with other estimation techniques. The method presented in the thesis is not subjected to these tests as the thesis objective is accurate boundary detection, not accurate optical flow estimation.

Chapter 6

Conclusions and Reflections

Closing the thesis, the work done is summarised and the contributions made are stated. Conclusions are drawn from the preceding work and future directions are reflected on.

6.1 Introduction

In the last chapters the concept of tensor voting used as a motion segmentation tool has been described. The theory has been explained, analysis done on various synthetic image sequences and testing of the algorithms conducted on natural image sequences.

This chapter concludes the thesis by summarising the work, discussing novel contributions made and suggesting potential future work.

6.2 Summary

The need for accurate motion segmentation in video compression techniques is still very even though data bandwidth has increased. There are still major content-providing networks that wish to place more video channels onto limited bandwidth such as on satellite television channels. For the most part the content can be compressed off-line, and the economies of scale do not avoid using super-computing capability to achieve high compression.

Tensor voting provides a framework where sequence features can be extracted while simultaneously rejecting outliers. The normal geometric features of point, lines and surfaces are well described in the literature. This thesis looks at extracting geometric features that are of interest to motion segmentation using tangential skew tensor voting. A novel geometric feature of skewness is identified, analysed and applied to image sequences. The algorithm uses all the data available in the sequence without doing censoring and densification. The framework maintains the single step extraction of features that makes the tensor voting approach attractive and adds skewness to allow occlusion and disocclusion to be detected.

The tangential skew tensor voting is aligned to characterising motion traces through a spatio-temporal volume. The kernel is modified to place the energy into the tangential direction, aligning its behavior to the desired need for the problem. The skewness is obtained by making the kernel asymmetric without adversely affecting any other attributes of the tangential tensor voting.

The encoding of the tensor is done directly from the data available and does not make use of block matching techniques to obtain motion information in the tensor. The data used is the spatio-temporal position of the pixel and the colour attributes of the pixel. Various cases are explored allowing adjacent pixels to be used in the token as well. The dimesionality of the tensor rises rapidly and the equivalent of a 5×5 block matching has a dimensionality of $N = 72$.

Special techniques on Graphic Processor Units (GPUs) are implemented to face the computational

challenge and the applicability of GPUs on tensor voting frameworks is demonstrated.

The algorithm developed is applied to natural sequences with varying degrees of success. Efforts to optimise the free parameters are made but the results are not in-line with the ideal case. The potential reasons for this are discussed and leave room for further work in the field.

6.3 Summary of contributions

This thesis has extended the use of tensor voting in moving object segmentation in video sequences. The tensor voting framework allows a one-step approach to extracting motion features in a video sequence. The extensions require extensive computational resources, which were developed during the course of the research. These extensions are discussed individually and the computational implementation is also discussed. Several concepts thought to be novel were introduced.

Tangential voting

The voting methods in the literature make use of a normal or surface feature obtained from the tensor voting process. This was adapted to provide a tangential or aligned feature voting process, firstly in a 3D case making use of cross products and then into the generalised N dimensional case. The voting kernel was changed appropriately to reflect the tangential voting in N dimensions and the validity of the kernel was checked in 3D.

Direct data tensor encoding

Instead of making use of normal block matching techniques to determine inter-frame motion, the tensors are constructed directly from the spatio-temporal volume pixels. Several cases were investigated with growing dimensionality. The direct encoding allows a small region of support to make a significant contribution to pixel associations in the tensor voting framework.

Non-symmetrical kernel

The kernels used in the literature are symmetrical. A non-symmetrical kernel was developed to allow the concept of skewness to be developed in the tensor voting framework. The non-symmetrical kernel was checked against its symmetrical counterpart to determine its validity in the tensor voting framework. It was found to behave in much the same way, except when the properties of skewness were desired.

Tensor skewness

In order to determine end caps of motion traces, where occlusion or disocclusion takes place, a concept from the MRI field was expanded on, and the third order tensor representation was developed in the tensor voting framework. It allows other useful features to become apparent, such as the start and end of motion traces in the spatio-temporal volume, by finding the N dimensional projection of the third-order tensor in the second-order space. The resulting Tensor Skewness map or *TS map* demonstrates motion edge boundary features and differentiates between left and right hand movement. Simple matched filter techniques reinforced the extraction of an often noisy skewness

parameter in the TS map to provide motion boundary detection.

Tensor voting implementation

With the advent of cost-effective Graphic Processor Units (GPU), the tensor voting framework was adapted to the GPU especially in the high-dimensional tensor voting problem. Speed increases of up to seven times that of a PC are observed.

6.4 Reflections

Using all the information in the vicinity of an image pixel without any transformation that distorts or reduces the information of the pixel is of great value. However, sensitizing the process to this information can lead to poor performance in noisy image data, especially when non-linear techniques such as eigen-decomposition are used. This becomes evident in the poor noise immunity exhibited when using natural images, and needs further investigation.

In natural images there are large sections where the colour information is fairly homogeneous. The tensor voting framework presented in this thesis does not remove low saliency areas and then re-interpolate them — it makes use of all pixels based on the underlying data. This leads to problems in these areas, and it is probable that a refining process similar to the tensor voting frameworks presented by other authors is necessary to address deficiencies in these areas.

The selection of tensors where the basis is not orthogonal and non-stationary could cause poor performance and needs to be investigated further.

The skewness measure in the spatio-temporal volume is derived from the specific tangential orientation in N dimensions. This thesis used the x projection of this skewness only. In a rigorous sense, this needs to be aligned to the orientation. It poses a problem in that the skewness measure is known only on the regular grid in the spatio-temporal volume. By making use of interpolation or nearest neighbor techniques, a more rigorous solution is possible. For the purposes of this thesis, the problems posed were made to have a significant x component in order to demonstrate the use of the skewness measure.

The introduction of the third-order tensor into the tensor voting framework has made a step forward in video motion segmentation. Due to the complexity of introducing a three-dimensional property into a high-dimensional problem, several simplifications were necessary. Even so the third order tensor used a fair portion of the processing capability in deriving the skewness measure. The memory constrained GPU also imposed upper limits in computation due to the third order tensor. As processing and memory capability increases in the GPU field, these problems will be lessened.

If object boundaries have motion vectors that are parallel but opposite, in a shearing arrangement, the skewness measure will not pick up a boundary. The method may need to be augmented with a form of decision logic similar to what Min [45] used in determining boundaries. A more comprehensive approach to cover the various failure modes of skewness would make a useful extension to this work.

Overall, the introduction of the skewness measure and direct data encoding in the tensor framework

opens a further avenue in moving object segmentation in video scenes. The motivation for using the tensor framework in this manner is to be able to segment closely adjacent (in 2D) portions of the image such as the branches in the *Flower Garden* sequence, as well as to declare openings in objects as demonstrated in the *Mobile Calender* sequence. It is useful in automatically giving layering information as part of the directly extracted information, as well as being able to accurately localise object boundaries through the use of small 2D kernels. The method currently does not interpolate data and only operates on the underlying data, maximising information use.

The GPU implementation presented fair gain on a CPU implementation, but if more effort is directed to segmenting the tensor voting problem in a way aligned to the GPU, better performance will be seen. Aspects that can be looked into that will cause substantial performance increase would be the use of memory coalescing between threads as the memory bandwidth is the limiting factor in the algorithm. Unfortunately, this is not trivial as the tensor voting problem requires large portions of memory that can only be found in the slow global memory. Currently the very small portions of fast memory are not sufficient for higher dimensional tensor voting.

Although the concept of skewness is an inherent geometric feature that can be extracted from the tensor voting process, the full impact needs to be explored further as a useful parameter in motion segmentation. The increase in parallel computing capability can allow exhaustive Monte Carlo simulation or more voters to be used per votee, which is currently limited in this thesis to 32.

Bibliography

[1] Simon Baker, Daniel Scharstein, J.P.Lewis, Stefan Roth, Michael Black, and Richard Szeliski, *A database and evaluation methodology for optical flow*, Proc. Eleventh IEEE International Conference on Computer Vision (ICCV 2007) (2007).

[2] V Bhaskaran and K Konstantinides, *Image and video compression standards — algorithms and architectures*, Kluwer Academic Publishers, 1997.

[3] G. Box and M. Muller, *A note on the generation of random normal deviates*, Annals of Mathematical Statistics **29** (1958), no. 2, 610–611.

[4] J. F. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986), no. 6, 679–698.

[5] Tat-Jen Cham and Roberto Cipolla, *A statistical framework for long-range feature matching in uncalibrated image mosaicing*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (1988), 442–447.

[6] Hsin-Chia Chen, Wei-Jung Chien, and Sheng-Jyh Wang, *Contrast-based color image segmentation*, IEEE Signal Processing Letters **11** (2004), no. 7.

[7] Y. Deng and B. S. Manjunath, *Unsupervised segmentation of color-texture regions in images and video*, IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001), no. 8, 800–810.

[8] P. Diaconis and M. Shashahani, *The subgroup algorithm for generating uniform random variables*, Probability in the engineering and informational sciences **1** (1987), 15–32.

[9] Gunnar Farnebäck, *Motion-based segmentation of image sequences*, Master's thesis, Linköping University, May 1996.

[10] ———, *Motion-based segmentation of image sequences using orientation tensors*, Proceedings of the SSAB Symposium on Image Analysis (1997), 31–35.

[11] ———, *Spatial domain methods for orientation and velocity estimation*, Lic. thesis, Dept. EE, Linköping University, March 1999.

[12] ———, *Fast and accurate motion estimation using orientation tensors and parametric motion models*, 15th IAPR International Conference on Pattern Recognition **1** (2000), 1135.

[13] ———, *Very high accuracy velocity estimation using orientation tensors, parametric motion, and simultaneous segmentation of the motion field*, Eighth IEEE International Conference on Computer Vision, 2001. ICCV 2001 **1** (2001), 171–177.

[14] ———, *Polynomial expansion for orientation and motion estimation*, Phd thesis, Linköping University, Sweden, 2002.

[15] R. Favam and S. Osber, *3D shape anisotropic diffusion*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03) **1** (2003), 179.

[16] Lionel Gaucher and Gerard Medioni, *Accurate motion flow estimation with discontinuities*, The Proceedings of the Seventh IEEE International Conference on Computer Vision **2** (1999), 695–702.

[17] Guy Gilboa, Nir Sochen, and Yehoshua Y. Zeevi, *Forward-and-backward diffusion processes for adaptive image enhancement and denoising*, IEEE Transactions on Image Processing **11** (2002), no. 7.

[18] I. W. Guest, *Tensor voting on sparse motion vector estimation*, Proceedings of the Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa (PRASA), 2004, pp. 7–11.

[19] ———, *Block DCT optimization on object edges*, Proceedings of the Sixteenth Annual Symposium of the Pattern Recognition Association of South Africa (PRASA), 2005, pp. 59–64.

[20] Ju Guo, Jongwon Kim, and C.C. Jay Kuo, *Fast and adaptive semantic object extraction from video*, Proceedings of SPIE Image and Visual Communications Processing (2000).

[21] Gideon Guy and Gerard Medioni, *Inference of surfaces, 3D curves, andjunctions from sparse, noisy, 3D data*, IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997), no. 11, 1265–1277.

[22] Song Han, Misuen Lee, and Gerard Medioni, *Non-uniform skew estimation by tensor voting*, Workshop on Document Image Analysis (1997).

[23] C. Harris and M. Stephens, *A combined corner and edge detector*, Proceedings of The Fourth Alvey Vision Conference (1988), 147–151.

[24] Horst Haußecker, Hagen Spies, and Bernd Jähne, *Tensor-based image sequence processing techniquesfor the study of dynamical processes*, Int. Symp. On Real-time Imaging and Dynamic Analysis (1998), 704–711.

[25] ISO/IEC, *Coding of audio-visual objects part 1: Systems*, Tech. report, ISO/IEC 14496-1:1999(E), 1999.

[26] Jiaya Jia, Yu-Wing Tai, Tai-Pang Wu, and Chi-Keung Tang, *Video repairing under variable illumination using cyclic motions*, IEEE Transactions on Pattern Analysis and Machine Intelligence **28** (2006), no. 5, 832–839.

[27] Jiaya Jia and Chi-Keung Tang, *Image repairing: Robust image synthesis by adaptive ND tensor voting*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition **1** (2003), I–643–I–650.

[28] ———, *Inference of segmented color and texture description by tensor voting*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004), no. 6, 771–786.

[29] ———, *Tensor voting for image correction by global and local intensity alignment*, IEEE Transactions on Pattern Analysis and Machine Intelligence **27** (2005), no. 1, 36–50.

[30] Eun-Young Kang, Isaac Cohen, and Gerard Medioni, *Robust affine motion estimation in joint image space using tensor voting*, 16th International Conference on Pattern Recognition, 2002. Proceedings **4** (2002), 256–259.

[31] Jinman Kang, Isaac Cohen, and Gerard Medioni, *Continuous multi-views tracking using tensor voting*, Proceedings of the Workshop on Motion and Video Computing (2002), 181.

[32] D. E. Knuth, *Seminumerical algorithms.*, 2 ed., The art of computer programming, vol. 2, Addison Wesley, 1997.

[33] Pierre Kornprobst and Gerard Medioni, *A 2D+t tensor voting based approach for tracking*, 15th International Conference on Pattern Recognition, vol. 3, Sept 2000, p. 7104.

[34] ———, *Tracking segmented objects using tensor voting*, IEEE Conference onComputer Vision and Pattern Recognition **2** (2000), 118–125.

[35] Mi-Suen Lee, *Tensor voting for salient feature inference in computer vision*, Phd thesis, University of Southern California, August 1998.

[36] Mi-Suen Lee, Gerard Medioni, and Philippos Mordohai, *Inference of segmented overlapping surfaces from binocular stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 6, 824–837.

[37] Jie Liang, *Informing science special issue on multimedia informing technologies - part 1 new trends in multimedia standards: MPEG4 and JPEG2000*, 1999.

[38] Haiying Liu, Rama Chellappa, and Azriel Rosenfeld, *Accurate dense optical flow estimation using adaptive structure tensors and a parametric mode*, IEEE Transactions on Image Processing **12** (2003), no. 10, 1170–1180.

[39] L. Luccheseyz and S.K. Mitray, *Color image segmentation: A state-of-the-art survey*, Proceedings of the Indian National Science Academy, vol. 67, 2001, pp. 207–221.

[40] Jose L. Marroquin, Edgar Arce, and Salvador Botelli, *Markov random measure fields for image analysis*, International Conference on Image Processing **1** (2002), I–765–I–768.

[41] George Marsaglia and Wai Wan Tsang, *The ziggurat method for generating random variables*, Journal for Statistical Software **5** (2000), no. 8.

[42] Amin Massad, Martin Babós, and Bärbel Mertsching, *Perceptual grouping in grey-level images by combination of gabor filtering and tensor voting*, 16th International Conference on Pattern Recognition (ICPR'02) **2** (2002), 20677.

[43] Gerard Medioni, Chi-Keung Tang, and Mi-Suen Lee, *Tensor voting: Theory and applications*, 12 Congres Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle (RFIA), February 2000.

[44] Changki Min and Gerard Medioni, *Tensor voting accelerated by graphics processing units (GPU)*, 18th International Conference on Pattern Recognition **3** (2006), 1103–1106.

[45] Changki Min and Gerard Medioni, *Inferring segmented dense motion layers using 5D tensor voting*, IEEE Transactions on Pattern Analysis and Machine Intelligence **30** (2008), no. 9, 1589–1602.

[46] Philippos Mordohai and Gerard Medioni, *Stereo using monocular cues within the tensor voting framework*, IEEE Transactions on Pattern Analysis and Machine Intelligence **28** (2006), no. 6, 968–982.

[47] M. Nicolescu, *A voting-based computational framework for visual motion analysis and interpretation*, Phd thesis, University of Southern California, August 2003.

[48] Mircea Nicolescu and Gerard Medioni, *4-D voting for matching, densification and segmentation into motion layers*, 16th International Conference on Pattern Recognition, vol. 3, August 2002, pp. 303–308.

[49] ———, *Layered 4D representation and voting for grouping from motion*, IEEE Transactions onPattern Analysis and Machine Intelligence **25** (2003), no. 4, 492–501.

[50] ———, *Motion segmentation with accurate boundaries — a tensor voting approach*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, June 2003, pp. I–382–I–389.

[51] Nvidia, *Nvidia CUDA Compute Unified Device Architecture — programming guide*, Nvidia, version 2.0 ed., July 2008.

[52] Shmuel Peleg, Benny Rousso, Alex Ray-Acha, and Assaf Zomet, *Mosaicing on adaptive manifolds*, IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000), no. 10, 1144–1154.

[53] Jan Poland, *Three different algorithms for generating uniformly distributed random points on the n-sphere*, unpublished note available on the internet., October 2000.

[54] Alex Rav-Acha, Yael Pritch, Dani Lischinski, and Shmuel Peleg, *Dynamosaicing: Mosaicing of dynamic scenes*, IEEE Transactions on Pattern Analysis and Machine Intelligence **29** (2007), no. 10, 1789–1801.

[55] P. A. Smith, *Edge-based motion segmentation*, Phd thesis, University of Cambridge, 2001.

[56] S. M. Smith, *Edge thinning used in the SUSAN edge detector*, Tech. Report TR95SMS5, Oxford Centre, 1995.

[57] S. M. Smith and J. M. Brady, *SUSAN — a new approach to low level image processing*, Tech. Report TR95SMS1, Oxford Centre, 1995.

[58] Chi-Keung Tang and Gerard Medioni, *Extremal feature extraction from 3-D vector and noisy scalar fields*, Proceedings Visualization '98 (1998), 95–102.

[59] ———, *Robust estimation of curvature information from noisy 3D data for shape description*, The Proceedings of the Seventh IEEE International Conference on Computer Vision, September 1999, pp. 426–433.

[60] ———, *Curvature-augmented tensor voting for shape inference from noisy 3D data*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 6, 858–864.

[61] Chi-Keung Tang, Gerard Medioni, and Mi-Suen Lee, *Epipolar geometry estimation by tensor voting in 8D*, The Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 1, September 1999, pp. 502–509.

[62] ———, *N-dimensional tensor voting and application to epipolar geometry estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001), no. 8, 829–844.

[63] ———, *N-dimensional tensor voting and application to epipolar geometry estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001), no. 8, 829–844.

[64] Linmi Tao, Vittorio Murino, and Gerard Medioni, *A tensor voting approach for the hierarchical segmentation of 3-D acoustic images*, First International Symposium on 3D Data Processing Visualization and Transmission (2002), 126.

[65] Wai-Shun Tong, Chi-Keung Tang, and Gerard Medioni, *First order tensor voting, and application to 3-D scale analysis*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, December 2001, pp. I–175–I–182.

[66] ———, *Simultaneous two-view epipolar geometry estimation and motion segmentation by 4D tensor voting*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004), no. 9, 1167–1184.

[67] Wai-Shun Tong, Chi-Keung Tang, Philippos Mordohai, and Gerard Medioni, *First order augmentation to tensor voting for boundary inference and multiscale analysis in 3D*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004), no. 5, 594–611.

[68] D. Tschumperle, *PDE based regularization of multivalued images and applications*, Phd thesis, Universite de Nice-Sophia Antipolis, December 2002.

[69] L. Vincent and P. Soille, *Watersheds in digital spaces: An efficient algorithm based on immersion simulations*, IEEE Transactions on Pattern Analysis and Machine Intelligence **13** (1991), no. 6, 583–598.

[70] Stella X. Yu, *Object-specific figure-ground segregation*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2** (2003), II– 39–45.

[71] X. Zhang, C. Ling, L. Qi, and E.Z. Wu, *The measure of diffusion skewness and kurtosis in magnetic resonance imaging*, to appear in Pacific Journal of Optimization (2009).