ROADMAKERS PAVAGE, PULSE REFORMATION FRAMEWORK AND IMAGE SEGMENTATION IN THE DISCRETE PULSE TRANSFORM

by

George Gene Stoltz

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Electronic Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology
UNIVERSITY OF PRETORIA

April 2014

Hierdie verhandeling word opgedra aan my dierbare familie.

My vriende, vir al die goeie tye wat ek gemis het.

My studieleiers, vir die menigde geselsies.

Dankie.

Die pad na wysheid is lank en eensaam, laat God jou vergesel.

SUMMARY

ROADMAKERS PAVAGE, PULSE REFORMATION FRAMEWORK AND IMAGE SEGMENTATION IN THE DISCRETE PULSE TRANSFORM

by

George Gene Stoltz

Supervisor(s):	Dr I. Fabris-Rotelli, Prof L.P. Linde
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Electronic Engineering)
Keywords:	LULU operators, DPT decomposition, Discrete Pulse Transform, graph algorithm, connected operator leakage, image segmentation, Pulse Reformation, BSD database, spot detection, DPT Library

The Discrete Pulse Transform (DPT) is a hierarchical decomposition of a signal in n-dimensions, built by iteratively applying the LULU operators. The DPT is a fairly new mathematical framework, with few applications, and is prone to leakage within the domain, as are most other connected operators. Leakage is the unwanted union of two sets after the DPT is applied. Leakage thus provides false information regarding the data. A solution to the leakage is proposed. Implementing the DPT in n-dimensions is not a trivial task and a platform to aid the research effort was required. The search for applications of the DPT is extended to image segmentation, where the potential was measured in a quantitative way.

The DPT was implemented by presenting a new algorithm, the Roadmakers Pavage based on the Roadmakers algorithm. The algorithm utilizes graph theory as a basis and is packaged in the DPT Library, created to assist other researchers. The Roadmaker's Pavage is currently the fastest available algorithm and presents the extracted pulses in a a more suitable manner.

The Pulse Reformation framework was developed to address the leakage problem within the

DPT. It was specifically tested with circular probes and showed successful object extraction of red blood cells. Additionally, by utilising the LULU scale-space, similar performance to the Difference of Gaussians method in detecting mRNA in fluorescence microscopy was demonstrated.

The DPT was also utilized in image segmentation. Using Iterated Conditional Modes and k-means, the DPT segmentation was compared to the other segmentation methods, such as the Gaussian scale-space. The DPT showed potential in image segmentation and it is recommended that further research be conducted with the DPT in image segmentation.

LIST OF ABBREVIATIONS

BCE	Bidirectional Consistency Error
BSD	Berkley Segmentation Dataset
CSA	Cost Scaling Algorithm
CSR	Compressed Sparse Row
DoG	Difference of Gaussian
DPT	Discrete Pulse Transform
GCE	Global Consistency Error
ICM	Iterated Conditional Modes
LCE	Local Consistency Error
mRNA	messenger Ribonucleic acid
PRI	Probabilistic Rand Index
ROC	Receiver Operating Characteristic
VOI	Variation Of Index

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Discrete Pulse Transform (DPT) and LULU scale-space are modern mathematical tools and only recently in 2011 [1] has the mathematics been proven. This opens a whole new research arena with multiple research opportunities which includes a required computer implementation, solutions to fundamental mathematical problems and some applications and building on already investigated DPT applications.

The research objective is easily defined by the requirement of expanding the DPT research domain. This may be achieved firstly by developing a computer platform implementation of the DPT for easy use; then addressing the fundamental problem of leakage; and, finally, providing some application of the solved problem which decreases the rawness of the DPT environment. Lastly, by building on previous research efforts and applications, the literature can be enhanced as a whole.

The development of a computer platform for the DPT is an obvious step, but addressing a fundamental mathematical problem is not. The DPT is formed by iteratively applying connected operators on a data set. Obvious problems with data sets are the discretisation of elements. An obvious problem with transforms is the risk of hiding or creating false information. The mechanism of creating or hiding information is specific to every transform, in connected operators one such mechanism is called leakage [2]. Solving leakage and applying the solution to some real-world problems forms part of the research objective to expand the DPT research domain.

Most application-driven research effort in the DPT domain has been focused on image segmentation [3], image compression [4] and data smoothing [5]. The bulk of the work has

been performed in image segmentation but no quantitative analysis exists to merge the research with other image segmentation algorithms. Building on previous image segmentation algorithms, the DPT effort can be enhanced.

With these objectives in mind three hypotheses may be formed:

- It is possible to create a d-dimensional DPT implementation which executes in a reasonable time and can be used by other researchers to enhance the research effort on the DPT.
- The leakage problem found in the DPT can be solved to such an extent that the solution can be applied to real-world problems, for example counting overlapping red blood cells within an image.
- The capability of the DPT in image segmentation can be evaluated quantitatively and reduce the literature gap within image segmentation literature.

Before any DPT applications can be approached, a computer implementation of the DPT is required. There exist a few implementations of the DPT, but their computational speed can be improved and they are dimension specific. A good library is required for the DPT providing easy access to multi-dimensional extracted data, fast execution times and cross-platform availability.

With good implementation of the DPT a fundamental mathematical problem can be addressed. The LULU operators, which are used in the DPT and form the LULU scale-space, operate on connected sets. Connected operators result in leakage within the transformation, creating false information in connecting two or more sets. This creates uncertainty within the transformation data. A way to combat leakage within the DPT would create a more predictable transform.

Following the solution to the leakage problem some applications can be investigated. The DPT is easily applied to multi-dimensional data and is inherently discrete. Consequently, discrete data sets are a good fit for the transform. A good starting point to understand how the DPT acts within higher dimensional data would be to look at two-dimensional data, for example images. A basic image processing technique is to apply segmentation

algorithms to images to extract information about specific regions within an image. To produce DPT research, a definition of some guidelines to evaluate the viability of the DPT in image segmentation is a good starting point.

This dissertation aims to provide the following:

- an introduction to the DPT,
- the development of a new graph-based implementation algorithm,
- a solution to the leakage problem within the DPT, and
- the evaluation of the DPT for image segmentation.

The available literature, on the DPT, is presented in the relevant chapters throughout the dissertation. Chapter 2 provides an overview of the mathematical development of the LULU operators, followed by the DPT and LULU scale-space. Chapter 3 implements the DPT by providing a new graph-based algorithm called the Roadmaker's Pavage. Chapter 4 provides a solution to the leakage problem of the DPT and applies the solution to real world problems such as biomedical engineering, focusing on cell segmentation and mRNA detection. Chapter 5 applies the DPT to image segmentation in order to evaluate the performance of the DPT. Chapter 6 contains the conclusion and is followed by a complete list of the references cited in the dissertation. One appendix is included which contains the user guide for the DPT Library developed in the dissertation.

The following paper has been published in the Proceedings of the 23nd Annual Symposium of the Pattern Recognition Association of South Africa: *On the leakage problem with the Discrete Pulse Transform decomposition*[6].

The following paper has been submitted for journal publication: *Pulse Reformation algorithm for leakage of connected operators.*

The following papers will be submitted for journal publication:

- Roadmaker's Pavage: A graph-based Discrete Pulse Transform implementation with application in data communication.

- Applications and memory-efficient implementation of the two-dimensional Discrete Pulse Transform.

CHAPTER 2

THE DISCRETE PULSE TRANSFORM

The Discrete Pulse Transform started with the development of the LULU theory by Carl Rohwer, in 1983 at the Institute for Maritime Technology in Simonstown, South Africa. The collection of LULU theory and the Discrete Pulse Transform (DPT), previously called multiresolution analysis, is discussed in depth in the book *Nonlinear Smoothers and Multiresolution Analysis* [7]. The book left the property of basic consistency unproven as well as the Highlight Conjecture. The basic consistency was partially completed in one dimension by Rohwer and Wild in 2007 [8] and finally completed in 2010 by Anguelov and Fabris-Rotelli [3]. The Highlight Conjecture, referring to the strong consistency of the DPT, was proven by Laurie [1] in 2011 and also by Fabris-Rotelli [9] using a different method in 2012.

This chapter discusses the main aspects and concepts of the mathematical building blocks of the DPT. We refer the reader interested in mathematical proofs and a more complete development of the DPT to the sources referenced in this chapter, with emphasis on the book by Carl Rohwer [7] .

2.1 THE FRAMEWORK

This section provides a mathematical framework in which the LULU operators and the DPT will be explained. The framework is provided in one dimension. The LULU operators, followed by the DPT will first be discussed in one dimension. The LULU operators and DPT will then be discussed in multi-dimensions, followed by a brief overview of the LULU scale-space.

The LULU operators were developed from the concept of a separator which follows from a

smoother. A smoother is an operator which removes noise from a signal by removing the noise elements from the contaminated signal. A separator further divides the signal into noise and true signal. A sequence in a vector space is most informative as a signal, as it is easier to relate to the natural world, such as a sequence of temperature values over a period of time. Although real world sequences are finite, their boundaries can be padded with zeros to create bi-infinite sequences. Let $\mathcal{X}$ be the vector space of bi-infinite sequences $x = \{x_i\}$ of real numbers with norms, such as

$$||x||_p = \left(\sum_{i=-\infty}^{\infty} |x_i|^p \right)^{\frac{1}{p}}, \text{for } p = 1, 2, ... \tag{2.1}$$

where the inner product between two bi-infinite sequences $x = \{x_i\}$ and $y = \{y_i\}$ is given by

$$< x, y >= \sum_{i=-\infty}^{\infty} (x_i y_i)^{\frac{1}{2}}. \tag{2.2}$$

Other norms are also a possibility, but $||x||_1$ will mostly be used. All sequences are assumed to have finite energy, such that $||x||_1 < \infty$ [7]. We can more formally define the type of sets with which we will be working, namely partially ordered sets [10].

Definition 1. *For set S, the relation B is said to be a partial order of S if:*

- *B is reflexive: $(x_i, x_i) \in B$, $\forall\ x_i \in S$.*
- *B is anti-symmetric: if $(x_i, x_j) \in B$ and $(x_j, x_i) \in B$ it implies that $x_i = x_j$.*
- *B is transitive: if $(x_i, x_j) \in B$ and $(x_j, x_k) \in B$ then $(x_i, x_k) \in B$.*

A partial order B does not consist of all the pairs in S, thus B is said to be a total order if for all $x_i, x_j \in S$ there exists either a $(x_i, x_j) \in B$ or a $(x_j, x_i) \in B$. This property is also well known as the Trichotomy Law [11].

To create a more usable set for real world problems, we can define a vector lattice $\mathcal{A}(\Omega)$ of all real functions defined on the Abelian group Ω. An Abelian group has axioms such as closure, associativity, identity element, inverse element and commutativity and is also called a commutativity group [10].

Definition 2. *A partially ordered set $\mathcal{L}$ is a lattice if any $s_1, s_2 \in S$ admit a least upper bound $l_1 \bigvee l_2$ and a largest lower bound $l_1 \bigwedge l_2$. For a vector lattice we have that for two sequences $x = \{x_n\}, y = \{y_n\}$ that $x \leq y \iff x_n \leq y_n \forall\ n \in \Omega$.*

A lattice can be seen as complete when every subset of $\mathcal{L}$ has a least upper bound and a largest lower bound. Thus, the sets we will be considering will all be vector lattices. We also need to perform operations on these sets where operators can be given in terms of element-wise operations such as $\{Ax\}_i = a_i$, which take the sequence $\{x_i, ..., x_j\}$ and transform it into $\{a_i, ...a_j\}$.

Definition 3. *Let $F(\mathcal{X})$ be the set of all operators on $\mathcal{X}$,*

$$F(\mathcal{X}) = \{A : \mathcal{X} \to \mathcal{X}\}. \tag{2.3}$$

From here the notation of function composition will be used as $g \circ f = gf$, thus if the operator $f : X \to Y$ and operator $g : Y \to Z$ exist then $g \circ f : X \to Z$, which can also be written as $gf : X \to Z$.

Definition 4. *An operator $A \in F(\mathcal{X})$ is linear if $A(x + y) = Ax + Ay$ and $A(\lambda x) = \lambda Ax \quad \forall x, y \in X$ and $\lambda \in \mathbb{R}$.*

Definition 5. *For every $A, B \in F(\mathcal{X})$ and $x \in \mathcal{X}$*

1. $(A + B)x = Ax + Bx$ — *Sum of operators*
2. $Ix = x$ — *Identity operator*
3. $(0x)_i = 0$ — *Zero operator*
4. $(\alpha A)x = \alpha(Ax), \alpha \in \mathbb{R}$ — *Scalar associativity*
5. $(AB)x = A(Bx)$ — *Operator composition*
6. $(Ex)_i = x_{i+1}, \forall i$ — *Shift operator*
7. $Nx = -x$ — *Negative operator*
8. $(A + B)C = AC + BC$ — *Right distributivity*
9. $A^0 = I, A^{n+1} = A^n A, n \in \mathbb{Z}$ — *Operator powers/ compositions*

In our space $\mathcal{X}$, the obvious relation to use is the $\leq$ and $\geq$ operators. With this we can define a syntone operator, which preserves the partial order relation on sequences. A syntone operator is also called increasing [12] or monotone [13].

Definition 6. *An operator P on $\mathcal{X}$ is syntone if and only if, for all sequences $x, y \in \mathcal{X}, x \geq y \Rightarrow (Px) \geq (Py)$.*

We will be looking at a specific type of operator called a separator, which functions as a complete smoother. A complete smoother should separate a signal into its noise component and its true signal component, while changes to the signal must transform linearly to the extracted true signal and noise component. Bearing these requirements in mind, the following axioms were developed by Mallows [14] and restricted to only non-negative scaling factors [15]:

Definition 7. *Smoother Axioms: An operator P on $\mathcal{X}$ is a smoother if:*

1. *$PE = EP$ where E is the zero element of the space.*
2. *$P(x + c) = P(x) + c$ for each x where $c \in \mathcal{X}$ is a constant sequence.*
3. *$P(\alpha x) = \alpha P(x)$ for each x and scalar $\alpha \geq 0$.*

With these axioms, a separator can be defined [16].

Definition 8. *An operator P is a separator if it satisfies the smoother axioms in Definition 7 as well as the following two axioms:*

1. $P^2 = P$ *Idempotence*
2. $(I - P)^2 = (I - P)$ *Co-Idempotence*

Evaluating these axioms for a separator, one notes that the original signal is presented by $Ix = x$, the true signal is presented by Px and the noise is presented by $(I - P)x$. A smoother P applied to a signal x consisting of the true signal s and noise n would imply that $Ix = s + n = Px + (I - Px)$. If P is a separator, reapplying P to the newly extracted signal would yield $PPx = Px = s$, $P(I - Px) = 0$ and $(I - P)(I - P)x = n$.

To further aid in the concept of smoothers and separators four different criteria can be used to evaluate smoothers and separators. For evaluation purposes the idea of true signal and noise will be dependent on the application and usage of the operators. The following criteria could be used [16]:

1. **Effectiveness** in terms of how well the operator separates the signal and noise component, that is how close Px is to the true signal.

2. **Consistency** in terms of how well the smoother operates as a separator, how consistently the operator decomposes the signal into Px and $(I - P)x$. This provides the ability to predict the outcome of a non-linear operator.

3. **Stability** in which small changes in terms of noise must not result in large changes in either the true signal or the noise signal.

4. **Efficiency** in terms of computational economy. A method that requires infinite time to compute is unrealisable.

A good smoother will adhere to these criteria where the smoothness of the sequence on which the smoother is applied also needs some kind of measurement. The total variation of a sequence can be used as a measurement for smoothness [17].

Definition 9. *The total variation $T(x)$ of a sequence $x \in \mathcal{X}$ is given by:*

$$T(x) = \sum_{i=-\infty}^{\infty} |x_{i+1} - x_i|. \tag{2.4}$$

Total variation, as defined in Definition 9, is a semi-norm as $T(x) \leq 0$ for, all $x \neq 0$, $T(\alpha x) = |\alpha| T(x)$ and $T(x + y) \leq T(x) + T(y)$. With the LULU operators introduced in Section 2.2, the total variation also becomes a natural norm, since $T(x) \leq 2\|x\|_1$ [18].

Operators can have specific effects on a sequence such as preserving variation in the sequence or preserving the shape (trend) in the sequence. Operators are thus said to be variation preserving, neighbour trend preserving, difference reducing or fully trend preserving [17].

Definition 10. *An operator P is variation preserving if*

$$T(x) = T(Px) + T(x - Px). \tag{2.5}$$

Definition 11. *An operator P is neighbour trend preserving if for each x,*

$$x_{i+1} \leq x_i \Leftrightarrow (Px)_{i+1} \leq (Px)_i \tag{2.6}$$

and

$$x_{j+1} \geq x_j \Leftrightarrow (Px)_{j+1} \geq (Px)_j. \tag{2.7}$$

Definition 12. *An operator P is difference reducing if for each x and subscript i,*

$$|(Px)_{i+1} - (Px)_i| \leq |x_{i+1} - x_i|. \tag{2.8}$$

Definition 13. *An operator P is fully trend preserving if it is neighbour trend preserving and difference reducing.*

A fully trend preserving operator P is also total variation preserving. Fully trend preserving operators produce some important results. Let P_1 and P_2 be fully trend preserving operators then [7]:

1. The results of $P_1 \circ P_2$ and $P_2 \circ P_1$ are fully trend preserving.

2. $\alpha P_1 + (1-\alpha)P_2$ is fully trend preserving for $\alpha \in [0, 1]$.

3. $I - P_1$ and $I - P_2$ are fully trend preserving.

4. A fully trend preserving operator also preserves n-monotone sequences.

A monotone sequence can be extended to an n-monotone sequence which then refers to a monotone set with a cardinality equal to n [19].

Definition 14. *A sequence $x \in \mathcal{X}$ is n-monotone if and only if the sequence $\{x_i, x_{i+1}, ..., x_{i+n+1}\}$ is monotone for every i.*

2.2 LULU OPERATORS

The LULU operators exist in n dimensions but for a facile explanation only one dimension will be treated, followed by the n-dimensional LULU in a later section. The LULU operators are built on two fundamental operators known as selectors. These selectors have the fundamental property that only the elements appearing in their inputs can appear in their outputs [7].

Definition 15. *The erosion $\bigwedge$ and dilation $\bigvee$ operators on $\mathcal{X}$ are:*

1. $(\bigwedge x)_i = \min\{x_{i-1}, x_i\}$

2. $(\bigvee x)_i = \max\{x_i, x_{i+1}\}$

These two operators can then be extended to multiple elements such that $(\bigwedge^n x)_i = \min\{x_{i-n-1}, ..., x_i\}$ and $(\bigvee^n x)_i = \max\{x_i, ..., x_{i+n+1}\}$. When applying these operators to a finite sequence the edges are padded with zeros.

In general the erosion operator can be visualized as increasing the size of a concavity or decreasing the size of a convexity. Conversely, the dilation operator can be seen as increasing the size of a convexity and reducing the size of a concavity. These operators act equivalently to the erosion and dilation operators defined in mathematical morphology [13].

The erosion and dilation operators in Definition 15 have some logical properties which can be deduced, but are also proven. The properties are as follows [7]:

1. $\bigwedge$ and $\bigvee$ are syntone.
2. $\bigwedge^m \leq I \leq \bigvee^m$ for all $m \geq 0$
3. $\bigvee^m \bigwedge^m \leq ... \leq \bigvee \bigwedge \leq \bigwedge \bigvee \leq \bigwedge^m \bigvee^m$ for all $m > 0$
4. $\bigvee^m \bigwedge^m \bigvee^m = \bigvee^m$ and $\bigwedge^m \bigvee^m \bigwedge^m = \bigwedge^m$
5. $(\bigvee^m \bigwedge^m)^2 = \bigvee^m \bigwedge^m$ and $(\bigwedge^m \bigvee^m)^2 = \bigwedge^m \bigvee^m$

Property 5 shows that the composition operators are idempotent.

The dilation and erosion operators can be combined to become smoothers. This finally brings us to the LULU operators which are the combination of erosion and dilation operators [7]:

Definition 16. *The LULU operators are the finite composition of the operators:*

$$L_n = \bigvee^n \bigwedge^n \text{ and } U_n = \bigwedge^n \bigvee^n \tag{2.9}$$

where $(\bigwedge^n x)_i = \min\{x_{i-n-1}, ..., x_i\}$ *and* $(\bigvee^n x)_i = \max\{x_i, ..., x_{i+n+1}\}$.

Applying U_n and L_n each iteratively but separately on a sequence, one will observe that these operators are biased towards one direction. The sequence on which U_n is applied will be smoothed from below while L_n will smooth a sequence from above. These operators thus need to be applied in composition to create unbiased smoothers. From the above properties of the dilation and erosion operators we can create some basic properties for the LULU

operators to show the inner workings better:

1. $L_{n+1} \leq L_n \leq L_0 = I = U_0 \leq U_n \leq U_{n+1}$ (Syntone)
2. $L_n L_m = L_m$ and $U_n U_m = U_m$ for all $m \geq n$
3. $L_n^2 = L_n$ and $U_n^2 = U_n$ (Idempotent)
4. $(L_n U_n)^2 = L_n U_n$ and $(U_n L_n)^2 = U_n L_n$
5. $L_n \leq L_n U_n L_n \leq L_n U_n$ and $U_n L_n \leq U_n L_n U_n \leq U_n$
6. $L_n U_n x$ and $U_n L_n x$ is n-monotone.

With these properties we can see that the LULU operators form a 4-element semigroup [20]. A semi-group is a non-empty set where a binary operation $(a, b) \rightarrow ab$ has been defined which forms a closure and is associative.

Definition 17. *Let S be a set with a binary operator $\cdot$. $(S, \cdot)$ is a semi-group if $\cdot$ is associative, i.e. $(a \cdot b) \cdot c = a \cdot (b \cdot c) \quad \forall\ a, b, c \in S$.*

Smoothing is a subjective evaluation of your objective. Thus to smooth a sequence to a certain degree in one step is not always desired. The amount of smoothing of a sequence can be controlled by using systematical smoothing by utilising C_n and F_n also known as the Ceiling and the Floor operators. The sequential application of the smoothers provides better smoothing results by removing the ambiguity that's present when using only L_n or U_n [18].

Definition 18. *We define operator C_n,*

$$C_{n+1} = L_{n+1} \circ U_{n+1} \circ C_n \quad \textit{with} \quad C_1 = L_1 \circ U_1, \tag{2.10}$$

and the operator F_n,

$$F_{n+1} = U_{n+1} \circ L_{n+1} \circ F_n \quad \textit{with} \quad F_1 = U_1 \circ L_1. \tag{2.11}$$

The LULU operators can be summarized as being effective separators which are fully trend-preserving. The operators are stable as they have a Lipschitz constant and provide consistency as they are idempotent and co-idempotent [7] . The LULU smoothers was also shown to

be computational efficient via the Roadmakers Algorithm [1]. The LULU operators are good smoothers by the criteria presented for evaluating smoothers and in addition they also function as separators [7].

2.3 THE DISCRETE PULSE TRANSFORM

The smoothing of a sequence has an inherent problem, as smoothing a sequence destroys information unless the information removed is stored. The Discrete Pulse Transform (DPT) provides a way to keep all information of the sequence while smoothing it. The DPT is a hierarchical decomposition of a sequence and also forms a scale-space [9] and can be defined by [21]:

Definition 19. *The Discrete Pulse Transform is a mapping of a sequence x into a vector:*

$$DPT(x) = [D_1(x), D_2(x), ..., D_{N-1}(x)]. \tag{2.12}$$

The DPT is a decomposition of x obtained using either $U_n \circ L_n$ or $L_n \circ U_n$, thus

$$D_n = (C_{n-1} - C_n) = (I - L_n \circ U_n)C_{n-1} \tag{2.13}$$

or

$$D_n = (F_{n-1} - F_n) = (I - U_n \circ L_n)F_{n-1}. \tag{2.14}$$

The DPT decomposition then relates to a sequence such that:

$$x = \sum_{n=1}^{N-1} D_n(x). \tag{2.15}$$

Each of the sequences in $DPT(x)$ can be described with a base function called a pulse. Each of these pulses consist of different heights so that:

$$x = \sum_{n=1}^{N-1} \sum_{s=1}^{\gamma(n)} \psi_{ns} \tag{2.16}$$

where ψ_{ns} is called a pulse which consists of a sequence of n non-zero constant values and elsewhere zero. The pulse will have relative height of α_{ns} which can be negative or positive. The cardinality of ψ_{ns} is equal to n.

The Highlight Conjecture state that each of the heights of the pulses ψ_{ns} can be scaled with a positive non-zero integer before reconstructing the complete sequence. The sequence will then decompose consistently into the new scaled base pulses from the first time it was decomposed. The Highlight conjecture now known as the Highlight theorem was proved using graph theory [1]. The DPT is a consistent hierarchical non-linear decomposition of a sequence. It provides the ability to have a non-negative linear combination of a decomposition which provides the same components when decomposed [21].

The DPT has another important property, namely total variation preservation for a hierarchical decomposition of a sequence with the iterative application of separators it is important for the decomposition to be total variation preserving [3]. The total variation using the DPT decomposition is given by:

$$T(x) = \sum_{n=1}^{N-1} \sum_{s=1}^{\gamma(n)} T(\psi_{ns}) \tag{2.17}$$

2.4 THE DPT IN MULTI-DIMENSIONS

Up until now we have only treated the LULU operators in one dimension. To have a complete theoretical understanding we discuss the LULU operators in multi-dimensional space as well as the Discrete Pulse Transform. The Discrete Pulse Transform also provides the LULU scale-space. The multi-dimensional development including proofs can be found in [3] and [9].

An n-monotone sequence is part of a connectivity class and is thus a connected set. The concept of an n-monotone sequence can be extended to higher dimensions with the introduction of connectivity classes [13].

Definition 20. *Let A be an arbitrary nonempty set. A family $\mathcal{C} \in \mathcal{P}(A)$ is called a connectivity class if the following axioms hold:*

1. $\emptyset \in \mathcal{C}$
2. $\{x_i\} \in \mathcal{C}$ *for every* i *such that* $x_i \in A$
3. *For each* $C_j \in \mathcal{C}$ *and* $\bigcap_{j \in I} C_j \neq \emptyset$, *then* $\bigcup_{j \in I} C_i \in \mathcal{C}$.

Any element of $\mathcal{C}$ is called a connected set where we say $\mathcal{C}$ defines a connectivity on A. We can now extend our sequence to d-dimensions such that $x \in \mathbb{Z}^d, d \in \{1, 2, 3, ...\}$ with the additional constraint of a bounded space. If $W \subset \mathbb{Z}^d$ then $card(W) < \infty$ where $card$ is the cardinality of the set. The LULU operators operate on n-monotone sequences in one dimension which translate to connected sets in d-dimensions. Thus we require a discrete space which is sufficiently rich in connected sets. The connectivity within the LULU framework will be sufficient if it meets the following conditions [3]:

- $\mathbb{Z}^d \in \mathcal{C}$
- Let $C \in \mathcal{C}$ then for any $a \in \mathbb{Z}^d$, $E_a(C) \in \mathcal{C}$ where E_a is the translation operator.
- If $V \subset W$ and $V, W \in \mathcal{C}$ then there exist $x \in W \backslash V$ such that $V \cup \{x\} \in \mathcal{C}$.

For any set of cardinality or size of $n+1$ we can now define within the d-dimensional space equivalent n-monotone sets which contain the point $x \in \mathbb{Z}^d$.

$$\mathcal{N}_n(x) = \{V \in \mathcal{C} : x \in V, \texttt{card}(V) = n + 1\}. \tag{2.18}$$

The LULU operators can be redefined on an abelian group $\mathcal{A}(\mathbb{Z}^d)$ such that commutativity always holds within the lattice.

Definition 21. *Let $f \in \mathcal{A}(\mathbb{Z}^d)$ and $n \in \mathbb{N}$. Then for $x \in \mathbb{Z}^d$:*

$$L_n(f)(x) = \max_{V \in \mathcal{N}_n(x)} \left\{ \min_{y \in V} \{f(y)\} \right\}, \tag{2.19}$$

$$U_n(f)(x) = \min_{V \in \mathcal{N}_n(x)} \left\{ \max_{y \in V} \{f(y)\} \right\}. \tag{2.20}$$

In order to fully utilise the the partially ordered space $\mathcal{A}(\mathbb{Z}^d)$ we need to be able to define local maximums and minimums in the space. In order to determine maximums and minimums we need to know when an element is adjacent to another element.

Definition 22. *Let $V \in \mathcal{C}$ then a point $x \notin V$ is called adjacent to V if $V \cup \{x\} \in \mathcal{C}$. The set of all adjacent points of V are then:*

$$\texttt{adj}(V) = \{x \in \mathbb{Z}^d : x \notin V, V \cup \{x\} \in \mathcal{C}\}. \tag{2.21}$$

With the concept of adjacency we can classify a connected set as a local minimum or a local maximum.

Definition 23. *Let $V \in \mathcal{C}$ and $f \in \mathcal{A}(\mathbb{Z}^d)$ then V is called a local maximum set if:*

$$\texttt{max}_{y \in \texttt{adj}(V)}\{f(y)\} < \texttt{min}_{x \in V}\{f(x)\}. \tag{2.22}$$

Or V is called a local minimum set if

$$\texttt{min}_{y \in \texttt{adj}(V)}\{f(y)\} > \texttt{max}_{x \in V}\{f(x)\}. \tag{2.23}$$

The L_n operator removes local maximums of size smaller and equal to n while U_n removes local minimums of size smaller and equal to n. The two operators can't create new local maxima or minima but they may enlarge the cardinality of a connected set attributed as a local maxima or minima. The LULU operators maintain their properties from the 1-dimensional case such as being a separator, being fully trend preserving and preserving total variation. The total variation can be redefined for $f \in \mathcal{A}(\mathbb{Z}^d)$.

$$T(f) = \sum_{p \in \mathbb{Z}^d} \sum_{i=1}^{d} |f(p + (e_k)_i) - f(p)| \tag{2.24}$$

where $e_k \in \mathbb{Z}^d$ with $(e_k)_i = \begin{cases} 0, & \text{if } i \neq k \\ 1, & \text{if } i = k \end{cases}$ for all $k = 1, 2, ..., d$.

The Discrete Pulse Transform in multi-dimensions is represented the same way as in the one dimensional case:

$$DPT(f) = [D_1(f), D_2(f), ..., D_{N-1}(f)]. \tag{2.25}$$

Each component D_n is calculated as:

$$D_1(f) = (I - P_1)(f) \tag{2.26}$$

$$D_n(f) = (I - P_n) \circ Q_{n-1}(f), \quad n = 2, ..., N-1. \tag{2.27}$$

where $P_n = L_n \circ U_n$ or $P_n = U_n \circ L_n$ and $Q_n = P_n \circ ... \circ P_1, n \in \mathbb{N}$. Each different scale D_n can be represented by:

$$D_n(f) = \sum_{s=1}^{\gamma(n)} \psi_{ns} \tag{2.28}$$

and

$$f = \sum_{n=1}^{N-1} \sum_{s=1}^{\gamma(n)} \psi_{ns} \tag{2.29}$$

where $\gamma(n)$ is the total number of local maximum and local minima of size n and ψ is a pulse.

Definition 24. *A function $\psi \in \mathcal{A}(\mathbb{Z}^d)$ is called a pulse if there exist a connected set V and a nonzero real number α such that*

$$\psi(x) = \begin{cases} \alpha, & if \quad x \in V \\ 0, & if \quad x \in \mathbb{Z}^d \backslash V. \end{cases} \tag{2.30}$$

The DPT decomposition forms a scale-space which can be formally defined as [9]:

Definition 25. *Let $f \in \mathcal{A}(\mathbb{Z}^d)$. The set*

$$\mathcal{S}_{f,\Lambda} = \{(\lambda, \mathcal{L}_f(\lambda)) : \lambda \in \Lambda\} \tag{2.31}$$

is called a scale-space of f generated by the operator $\mathcal{L}$ with respect to scale parameter set Λ and measure of smoothness $S \in \mathcal{A}(\mathbb{Z}^d)$.

S is a function called the measure of smoothness which is dependant on the requirement of the specific task. Overall a very smooth signal yields a smoothness measure of 0 where rougher signals yield higher values. In case of the DPT the measure of smoothness determines how close the current sequence is to its local monotonicity. The interested reader can find more information in *Discrete Pulse Transform of images and applications* [9].

The Discrete Pulse Transform forms the scale-space

$$S_{f,LULU} = \{(n, P_n(f)) : n \in \Lambda_0 = \{0, 1, 2, ..., N\}\} \tag{2.32}$$

called the LULU scale-space. A scale-space allows the tracking of structures within a domain through different scales ranging from fine to coarse.

2.5 CONCLUSION

In this chapter we discussed a mathematical framework on which the LULU operators and the Discrete Pulse Transform where built. Some interesting properties such as idempotence and

co-idempotence of the LULU operators where discussed. The LULU operators and DPT were also discussed in multi-dimensions followed naturally by the creation of the LULU scale-space. The consistent decomposition of the DPT was also discussed.

In the next chapter, the more practical side of implementing and representing the DPT will be discussed. An example of the DPT decomposition is included with the development of a software library for the DPT called *The DPT Library* [22] in Chapter 3.

CHAPTER 3

THE DPT IMPLEMENTATION

The computation of any set of mathematics is important. If a set of mathematics can not be computed by computers, the application of the mathematics in real-world problems reduces drastically. The Discrete Pulse Transform (DPT) can be implemented by using the operators as they were described in Chapter 2. This implementation method is however very inefficient and execution time is slow. A general concept for implementing the DPT efficiently is called the Roadmaker's algorithm [23].

The Roadmaker's algorithm showed, in 2006, that the DPT can be executed in $O(n)$ time [23]. The Roadmaker's algorithm uses the analogy of building a perfectly straight road in executing the DPT. You start with filling all the holes of size one in the road, followed by removing all the bumps in the road of size one. By continuously increasing the size of the holes and bumps to be removed, the road will become perfectly smooth. The removed holes and bumps represents the pulses extracted by the DPT. A Python implementation, specific in two dimensions, for image processing of the Roadmaker's algorithm was done in 2009 by Van der Walt [24]. The Roadmakers algorithm was extended to graph theory and used to prove the Highlight Conjecture in 2011. Utilizing graph theory, the implementation of the DPT in d-dimensions was shown to be possible and this was also implemented in Python [1].

The implementation done by Van der Walt [24] is based on a line scan method to find the pulses and a Compressed Sparse Row [25] format to store them. The Compressed Sparse Row format is used as each individual pulse can be represented by a sparse binary matrix. The relation of each pulse is then stored as a tree where the nodes in the tree have various properties such as, the height of the pulse and the Compressed Sparse Row matrix.

In this chapter a new algorithm and a library, based on the Roadmaker's algorithm, is developed called the Roadmaker's Pavage and the DPT Library. An in-depth discussion of the Roadmaker's Pavage is given with some comparison to the Roadmakers algorithm. An execution time analysis is applied to the Roadmakers Pavage to show linear complexity. Implementation specific details are also provided with a user guide for the DPT Library.

3.1 THE ROADMAKERS ALGORITHM

The Roadmaker's algorithm in [1] is the first to use graphs to do the DPT decomposition. The Roadmakers algorithm concepts are discussed with complementary pseudo code.

The d-dimensional sequence $\mathbf{x}$ with N elements can be represented as a real-valued function with domain defined on a connected undirected graph. Each element is assigned a unique tag such that no two elements have the same tag. An element tagged x_j represents the node j where the node's value x_j can also be referred to as the height h_j of the node.

Each node has a set of neighbours $N(j)$ associated with it where each neighbour is represented by an edge such that if node j has a neighbour node i the edge is denoted by $E(j,i)$. If node j is a neighbour of node i then node i is a neighbour of node j thus the edges connecting them is the same edge $E(j,i) = E(i,j)$, that is the edges are undirected.

A cluster of size n is a subgraph consisting of n nodes. The neighbours of a cluster are those nodes which are a neighbour of any node in the subgraph of n nodes but which is not contained in the subgraph of n nodes. A cluster in which the value of every node is equal, is called a constant-valued cluster. A constant-valued cluster can be contracted into a single node of *size* n, with the same neighbours as the cluster of size n.

A *pit* is defined as a local minimum, thus if a node's value is strictly lower than all of its neighbours, it is called a *pit*. Conversely a node which is a local maximum is called a *bump*. A node which is either a *pit* or a *bump* is called a *feature*. A node i is said to be the nearest neighbour of node j if $\min\limits_{k \in N(j)} |x_j - x_k|$ is attained for $k = i$ where $N(j)$ are the neighbours of node j.

The Roadmakers algorithm proposes two operators P_n and Q_n which operate on the nodes created in the graph. P_n is equivalent to L_n where Q_n has is equivalent to U_n.

- $P_n\mathbf{x} = \mathbf{y}$, where $y_j = x_m$ if node j is a *pit* of size n with nearest neighbour node m, otherwise $y_j = x_j$.

- $Q_n\mathbf{x} = \mathbf{y}$, where $y_j = x_m$ if node j is a *bump* of size n with nearest neighbour node m, otherwise $y_j = x_j$.

A *feature* is *flattened* whenever P_n or Q_n have operated on it so that the *feature* node has been modified to the value of its nearest neighbour, that is x_m. The *pit*-removal operator P_n and *bump*-removal operator Q_n operate on the whole sequence instantaneously but separately. Fortunately when executing P_n the *pits* of size n can be *flattened* in any order. Q_n also have this property.

To implement a mathematically sound DPT decomposition with this algorithm, a cluster of infinite size and zero value must be part of the graph, this cluster is also called the *zero node*. It is required to create edges between the *zero node* and the data elements which are on the boundaries of the sequence.

The DPT can be executed by firstly creating a node for each data point in a graph where each node has an edge towards each one of its neighbours. Secondly all *pits* of *size* 1 are *flattened* which is followed with the *flattening* of all *bumps* of size 1. All the *pits* of *size* 2 are then *flattened* followed with the *flattening* of all the *bumps* of *size* 2. The *flattening* of *features* continue in this fashion until only one cluster of infinite size and 0 value is left. This state will be achieved after all the features of size N and smaller have been removed. The process can be executed by removing the *bumps* of size n before the *pits* of size n. The process is shown in Algorithm 1.

If a *feature* j of *size* n is *flattened* to its nearest neighbour m, a *pulse* p of *size* n is created and stored representing the *flattened feature*. The stored pulse forms part of the DPT decomposition. The stored *pulse* is also assigned a *height*. The *height* h_p of *pulse* p is equal to the difference between the values of *feature* j and its nearest neighbour m, thus $h_p = x_j - x_m$. Note that h_p can be negative.

The Roadmaker's algorithm can be completed in $O(m)$ complexity. An in-depth discussion on the proposed algorithm based on the Roadmaker's algorithm, called the Roadmaker's Pavage, will be given in Section 3.2 below. In the rest of the chapter, when referring to the Roadmakers

Algorithm 1 The Roadmaker's algorithm overview

1: Create N nodes from sequence $\mathbf{x}$
2: For every neighbour j of every node i create an edge $E(i, j)$.
3: **for** $n = 1$ to N **do**
4: Apply P_n
for every *pit* of *size* n in graph **do**
Flatten feature
Store *pulse* of *size* n representing *flatten feature*.
end for
5: Apply Q_n
for every *bump* of *size* n in graph **do**
Flatten feature
Store *pulse* of *size* n representing *flatten feature*.
end for
6: **end for**

Algorithm, the author specifically refer to the graph based algorithm given in [1].

3.2 THE ROADMAKERS PAVAGE

3.2.1 Overview

The Roadmakers Pavage is based on the Roadmakers Algorithm, named in such a manner that by paying a fee a better road can be built. The fee comes in as computational gradient where a more usable Discrete Pulse Transform decomposition is stored. The new algorithm increases the required execution cycles per edge, but still keeps the algorithm complexity linear. The Roamakers Algorithm provides a very dense representation of the DPT, thus making it very difficult to utilise the decomposition. The Roadmakers Pavage extract pulses in a more usable format as well as providing a way to easily visualise the DPT decomposition. An overview of the Roadmaker's Pavage is given in Algorithm 2 below.

Algorithm 2 The algorithm layout

1: $data \leftarrow$ vector of N $d-dimensional$ elements.
2: $C_k(i) \leftarrow$ define connectivity function for data
with $k = 1, 2..., p$ where p is the number of connections.
3: $[W_G, P_G]$ = CONSTRUCT GRAPH($data, C_k(i)$)
4: $FeatureTable$ = FEATURE TABLE(W_G)
5: P_G = DISCRETE PULSE TRANSFORM(W_G, P_G,$FeatureTable$)

The data sequence is converted into a vectorized set by defining the connectivity of the data sequence such as 4-connectivity for 2-dimensional data, see Definition 20 in Chapter 2. The data sequence is used to create two graphs, the *work graph* W_G and the *pulse graph* P_G. A *feature table* is created from analysing the *work graph* where the *feature table* is then traversed to perform the DPT. Each new pulse extracted from the *work graph* updates the *pulse graph*. The algorithm stops whenever the *feature table* becomes empty. The *pulse graph* then presents the extracted pulses of the DPT.

3.2.2 An Example

Before an in-depth discussion of each section, an example will be discussed to aid the visualization of the algorithm as well as providing a comparison between the Roadmakers Algorithm and the Roadmakers Pavage. For simplicity we use a 1-dimensional example. The sequence

$\mathbf{x} = [x_1 \ x_2 \ x_3 \ x_4] = [7 \ 3 \ 5 \ 5]$ consisting of four elements, each with a value as shown in Figure 3.1. This sequence will be used in the example that follows.

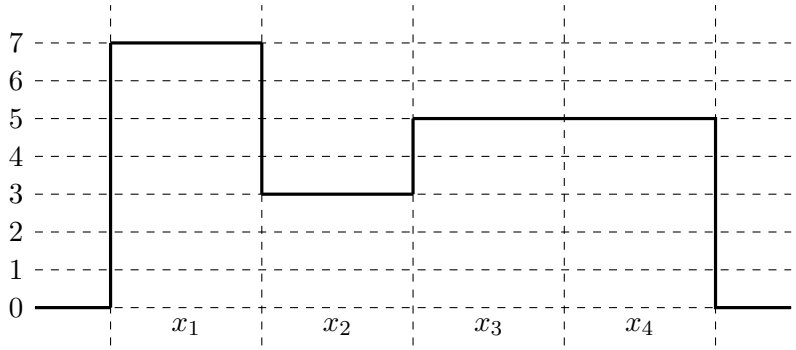


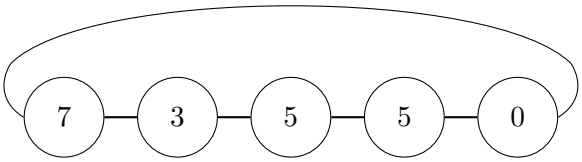
Figure 3.1: The Input sequence for the DPT algorithm example.

Before the DPT can be performed, the data sequence $\mathbf{x}$ must be converted to a graph. The converted data sequence is shown in Figure 3.2. Figure 3.2a shows the graph for the Roadmakers Algorithm where the additional zero padding node can be observed. The initialization of the Roadmakers Pavage is shown in Figure 3.2b with two graphs, the *pulse graph* and *work graph*. The *pulse graph* consist of arcs, where the *work graph* have edges. These two graphs are linked by *virtual edges* which are not part of either graphs but keep track of how the different nodes in each graph relate to the other.

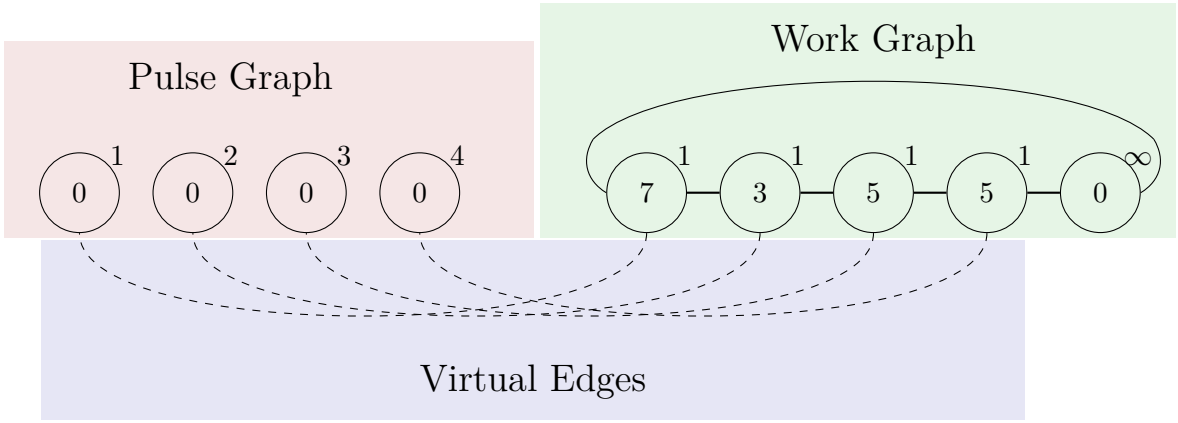
The *work graph* constitutes the data sequence and its connectivity. The *work graph* is initialised by creating a node for each data element in the data sequence. The nodes are then connected with edges to the neighbours of each data element. To conform to the DPT theory, all the boundary elements in the data sequence must be connected to an element with a 0 value which connect to an infinite number of other elements with 0 value. For this purpose we create a *zero node* in the *work graph*. The *zero node* is a node with *value* 0 and infinite *size*. For a node created from a boundary data element, an edge is added, connecting to the *zero node*. In Figure 3.2b, in the *work graph*, the number within the node represents the *value* of the data element the node was created from. The superscript of the node represents the number of elements the node contains. The Roadmakers Algorithm follows the same initialisation, but excludes the superscript.

The *pulse graph* comprises the pulses extracted with the DPT. An important aspect of storing

the pulses is to know which pulse belong to which elements in the data sequence. This is achieved when the *pulse graph* is initialised with base nodes of *height* zero and *size* equal to the index within the data sequence. For each element in the data sequence, one base node is created with the *size* equal to the index as the data element. Each base node is then connected with a *virtual edge* to its corresponding *work graph* node. In Figure 3.2b, in the *pulse graph*, the number within the node represents the *height* of the pulse the node represents where the superscript represent the *size* of the pulse. Although the *size* of the node can determined by traversing the graph for usability each node records its own size.



(a) Roadmakers Algorithm



(b) Roadmakers Pavage

Figure 3.2: Initialization of the DPT decomposition algorithms

After the graphs have been initialised they can be traversed to find nodes with neighbours of equivalent *value*. The *work graph* will also be traversed in the Roadmakers Pavage. The equal-valued nodes are merged into clusters.

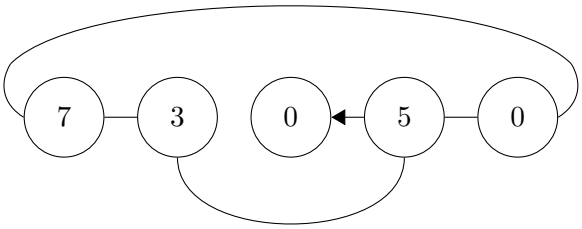
The Roadmakers Algorithm merges nodes by creating an arc between the equal-valued nodes, with all arcs directed away from one arbitrary chosen node to represent the cluster. The heights of the nodes not being used as a cluster representative is set to zero and all edges

connecting to them is moved to the cluster node. The cluster node then connects with edges to all of the clusters' neighbours.

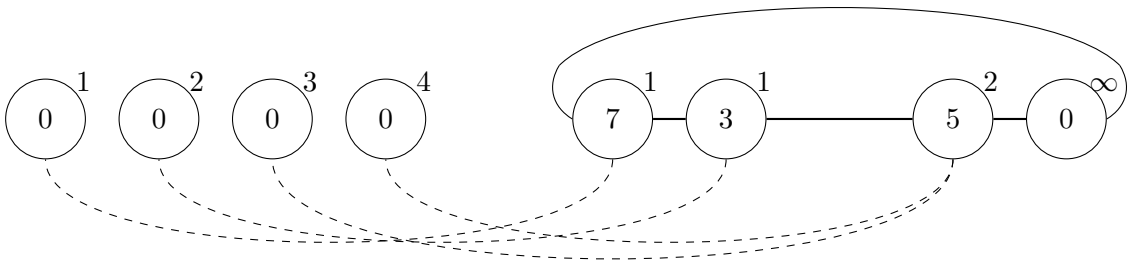
The Roadmakers Pavage merges nodes to create clusters by subsequently removing all nodes except one which will represent the cluster. A node is only deleted after all its edges and *virtual edges* have been moved to the cluster node and the *size* of the cluster node has been increased with the *size* of the node to be deleted.

While the graphs are traversed to find equal-valued nodes, each node is also checked to determine whether it's a *feature*. A *feature table* is created by adding every node identified as a feature to the table. The *feature table* only contains references to nodes identified as features within the graphs. This table is not sorted and needs to be traversed to find the required feature size and type.

The new updated graphs can be seen in Figure 3.3. The two nodes in the *work graph* with the *value* of 5 are merged to form a new node of *size* 2 which is shown in Figure 3.3b. All the nodes left in the *work graph* are *features*. In Figure 3.3a an arc is added from the node which will present the two merged nodes and the place holder node which becomes a zero. For both graphs the node of *value* 7 is a *bump* of *size* 1, the node of *value* 3 is a *pit* of *size* 1, the node of *value* 5 is a *bump* of *size* 2 and the node of *value* 0 is a *pit* of infinite *size*.



(a) Roadmakers Algorithm

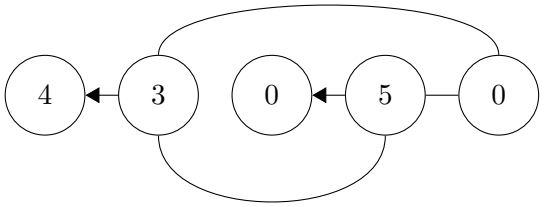


(b) Roadmakers Pavage

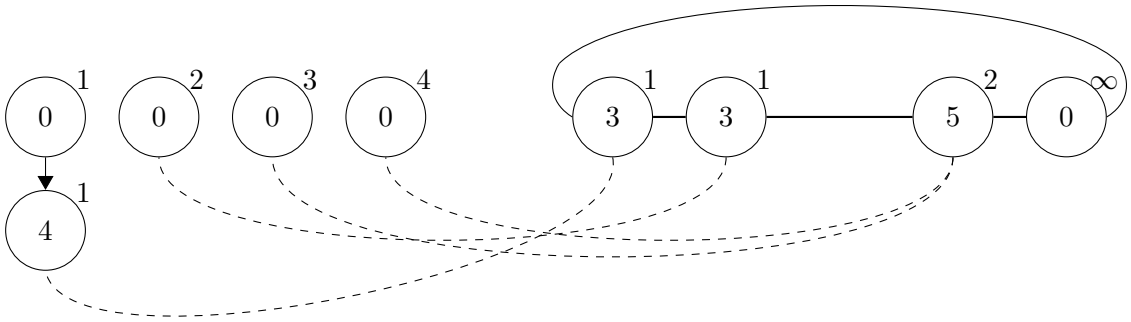
Figure 3.3: Merging nodes.

The DPT decomposition is now applied starting with $n = 1$ by first applying the P_n and then Q_n followed by increasing n until $n = N$. The algorithm can be varied by applying Q_n first followed by P_n. The first *feature* to be extracted is the *bump* of *value* 7. The feature's nearest neighbour is the node of *value* 3. To *flatten* the feature a new pulse must be created of *height* 4 consisting of the same elements as the *feature*.

The Roadmakers Algorithm achieves this by *flattening* the feature and setting the feature nodes' height to the height of the extracted pulse. An arc is created from the nearest neighbour towards the feature node while all the edges is shifted to the nearest neighbour. This step includes the pulse extraction and the creation of clusters and is shown in Figure 3.4a.



(a) Roadmakers Algorithm



(b) Roadmakers Pavage

Figure 3.4: Extracting the first pulse.

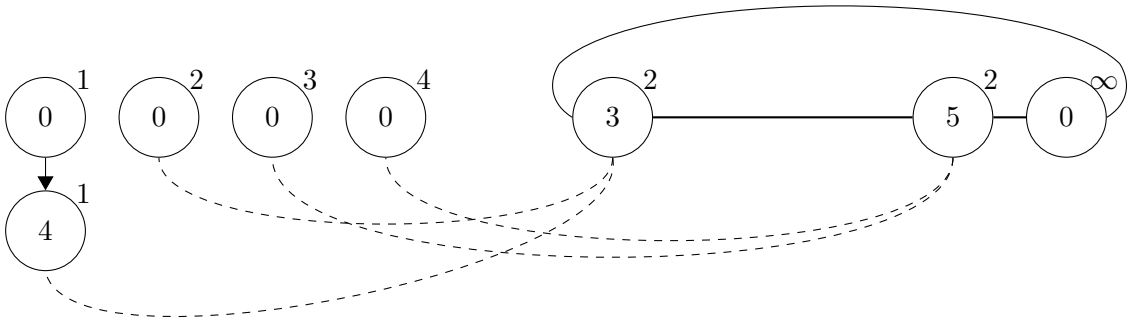
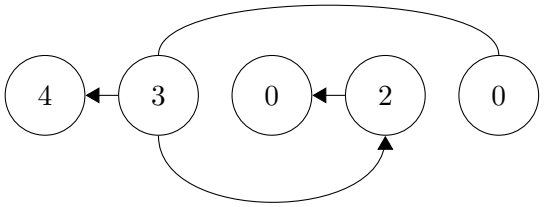


Figure 3.5: Merging two nodes.

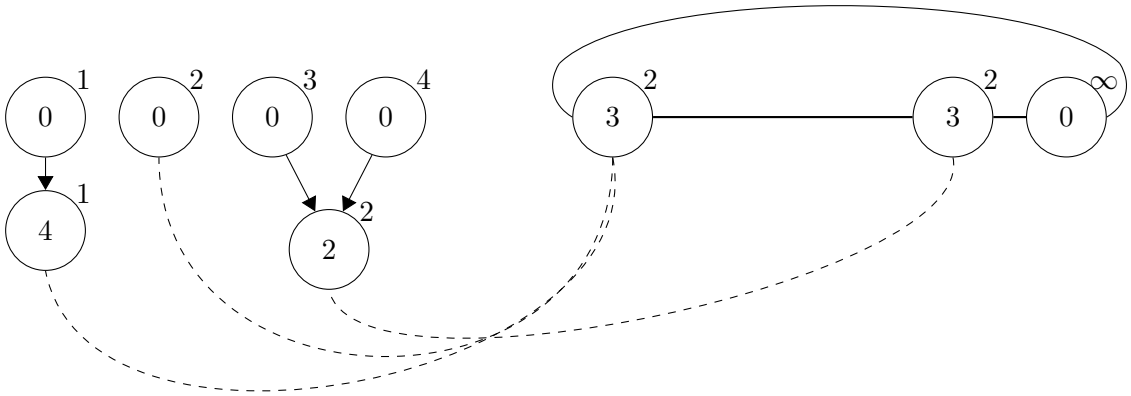
The Roadmakers Pavage creates a new node in the *pulse graph* with *height* 4 and *size* 1. The *feature* node in the *work graph* connects to a *base node* in the *pulse graph* through a *virtual edge*. An arc is created from this *base node* to the new node created in the *pulse graph*. The *virtual edge* is then moved to connect between the new node in the *pulse graph* and the *feature*

node in the *work graph*. The flattening of the *feature* node is completed by setting the *value* of the *feature* node to the *value* of the nearest neighbour, this is shown in Figure 3.4b. The merging of all equal-valued nodes is show in Figure 3.5.

Next a second pulse can be extracted. The same procedure is followed as described for the first pulse but now a pulse of *size* 2 and *height* 2 will be extracted. The extraction of the pulse for Roadmakers Algorithm is shown in Figure 3.6a. The extraction of the pulse by the Roadmakers Pavage is shown in Figure 3.6b and followed by the merging operation in in Figure 3.7.



(a) Roadmakers Algorithm



(b) Roadmakers Pavage

Figure 3.6: Extraction of second pulse.

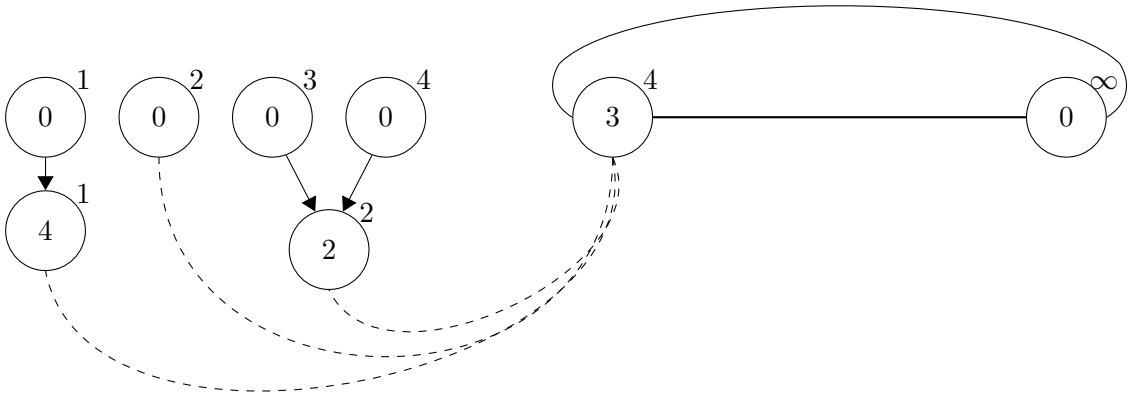
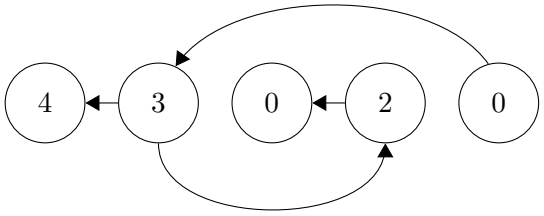
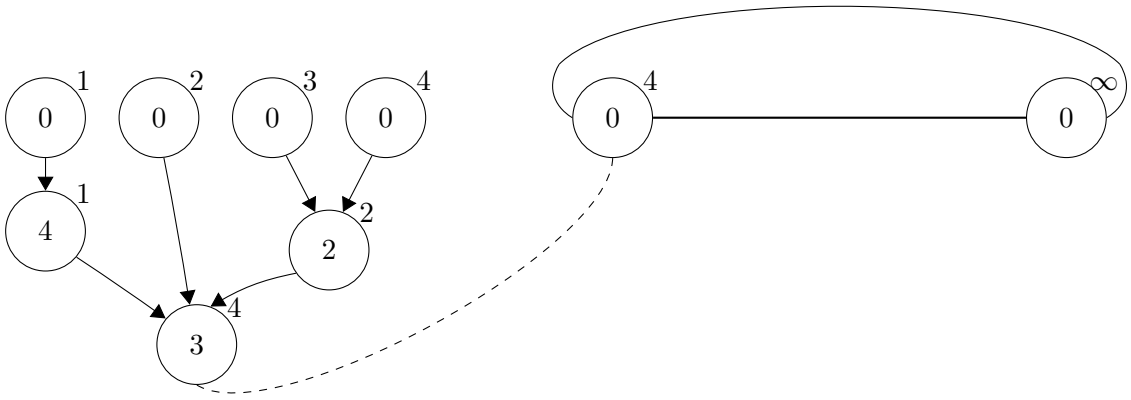


Figure 3.7: Merged node after second extracted pulse.



(a) Roadmakers Algorithm



(b) Roadmakers Pavage

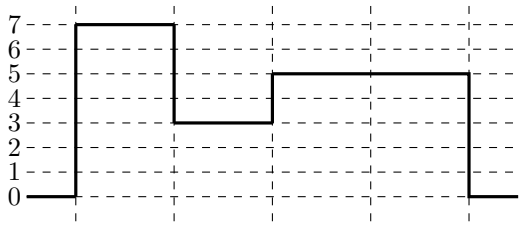
Figure 3.8: Extraction of last pulse.

A third pulse can be extracted from the graphs. There exist no features of *size* 3 thus the third pulse is of *size* 4 and *height* 3. The pulse extraction is shown in Figure 3.8. The extraction of this pulse shows the utility of using a tree structure to store the pulses. Although the extracted pulse is of *size* 4, thus consisting of 4 elements, it only has three children or references.

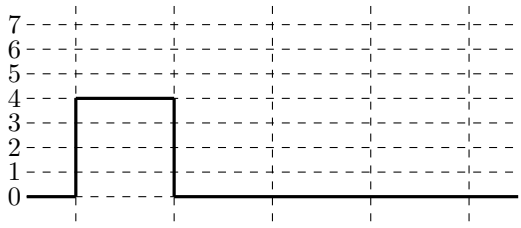
In Figure 3.8b it is evident that the *pulse graph* is in the form of a tree structure. This structure is a special case of a Directed Acyclic Graph. The nodes are connected with arcs. An arc is always directed towards the node's parent. A node is a child of another node if that node is its parent. It can then readily be said that a node in the *pulse graph* without any children is a *base node*.

Figure 3.9 shows the completed DPT decomposition for both algorithms, the input signal and the three reconstructed pulses. This figure concludes the example.

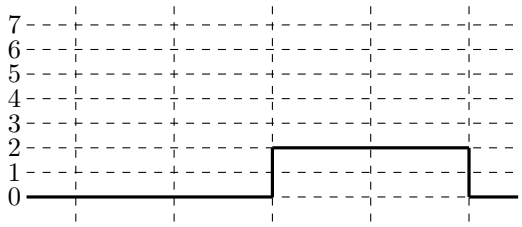
The example showed the complete process of the DPT decomposition as proposed by the Roadmakers Pavage with a comparison to the Roadmakers Algorithm. Each step shown in Algorithm 2 of the Roadmakers Pavage will now be discussed in more detail.



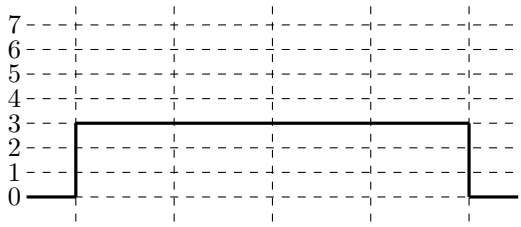
(a) Input data sequence



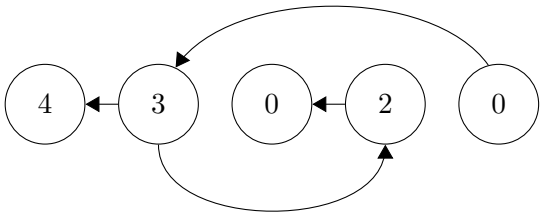
(b) Pulse of $size = 1$



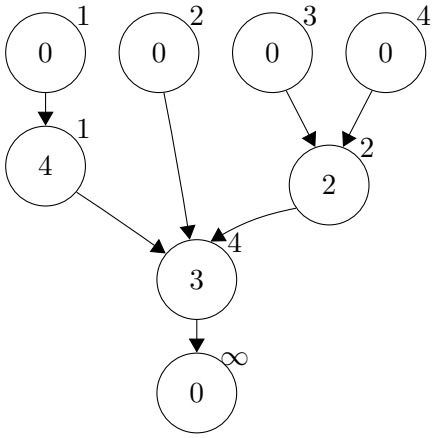
(c) Pulse of $size = 2$



(d) Pulse of $size = 4$



(e) Roadmakers Algorithm



(f) Roadmakers Pavage

Figure 3.9: DPT decomposition with extracted pulses.

3.2.3 Data Sequence Vectorization

The original data set must be digitised and placed into a workable format so we can work with high dimensional data. Thus we use a lexicographic ordering.

The lexicographic ordering of the example given in the previous section is then:

$$\mathbf{x} = \mathcal{O}(data) = \mathcal{O}\left(\begin{bmatrix} 7 \\ 3 \\ 5 \\ 5 \end{bmatrix}\right) = \begin{bmatrix} 7, & 3, & 5, & 5 \end{bmatrix}. \tag{3.1}$$

Any d-dimensional data sequence S can be ordered lexicographically. We create a formula to convert the coordinates of the d-dimensional data to the index $i = 1, 2, ..., N$ of a vector $\mathbf{x}$. Each element in the data sequence S can be addressed by a d-tuple such that the tuple is presented by $(s_1, s_2, ..., s_d)$ where $s_j \in \{1, 2, ..., max_j\}$ and $j \in \{1, 2, ..., d\}$. Then $x_i = S(s_1, s_2, ..., s_d)$ and the index i is given by:

$$\begin{aligned} i = s_1 + & \\ & (s_2 - 1)(\max_1) + \\ & (s_3 - 1)(\max_2.\max_1) + ..+ \\ & (s_d - 1)(\max_{d-1}.\max_{d-2}..\max_1) \end{aligned} \tag{3.2}$$

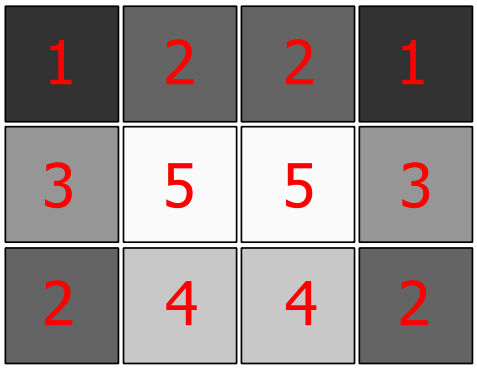


Figure 3.10: 2-Dimensional data sequence(image)

An image of a 2-dimensional sequence is shown Figure 3.10. The numbers in the image indicate the value of each element, although it is also presented by grey levels. The sequence in Figure 3.10 has a cardinality of $N = 12$, $\max_1 = 4$ and $\max_2 = 3$. The vectorized data is then given by

$$\mathbf{x} = \mathcal{O}\left(\begin{bmatrix} 1 & 2 & 2 & 1 \\ 3 & 5 & 5 & 3 \\ 2 & 4 & 4 & 2 \end{bmatrix}\right) = \begin{bmatrix} 1, & 2, & 2, & 1, & 3, & 5, & 5, & 3, & 2, & 4, & 4, & 2 \end{bmatrix}. \tag{3.3}$$

3.2.4 Define Connectivity Functions

To define neighbours and edges of different nodes in a graph, the connectivity of the data set must be defined. The connectivity specifies the neighbours of a data point which then gets translated to the index of the vectorized data. A node for each element in $\mathbf{x}$ will be created. To find the neighbours of each node a function C is defined for each neighbour k. The function is dependant on the index i of the current data point x_i in the vector $\mathbf{x}$. The function $C_k(i)$ outputs a new index j_k which is used to allocate the neighbouring data point x_{j_k}. The set of functions is then given by

$$j_k = C_k(i) \qquad \forall k = 1, 2, ..p. \tag{3.4}$$

Looking at a 1-dimensional data sequence, a logical 2-connectivity can be defined. The neighbours of a data point in the 1-dimensional data sequence are defined as the data point left and the data point right of the current data point position. The functions to support this definition is then

$$C_1(i) = i - 1,$$
$$C_2(i) = i + 1.$$

Looking at a typical 4-connectivity in a 2-dimensional data sequence such as an image, the neighbours can be above, below, left and right of the current data point. The connectivity functions supporting such configuration are then

$$\begin{aligned} \text{Top neighbour} \qquad & C_1(i) = i - \max_1, \\ \text{Bottom neighbour} \qquad & C_2(i) = i + \max_1, \\ \text{Left neighbour} \qquad & C_3(i) = i - 1, \\ \text{Right neighbour} \qquad & C_4(i) = i + 1. \end{aligned}$$

The $\max_1$ is the maximum of the 1^{st} dimension of the 2 dimensions as used in the vectorization of the data set in Section 3.2.3. By utilizing functions in this way the indexes are assigned with regards to each data point. More complicated configuration and graphs can be constructed. The connectivity function for 8-connectivity in a 2-dimensional space is then

$$\begin{aligned}
\text{Top left neighbour} \quad & C_1(i) = i - \max_1 - 1, \\
\text{Top neighbour} \quad & C_2(i) = i - \max_1, \\
\text{Top right neighbour} \quad & C_3(i) = i - \max_1 + 1, \\
\text{Left neighbour} \quad & C_4(i) = i - 1, \\
\text{Right neighbour} \quad & C_5(i) = i + 1, \\
\text{Bottom left neighbour} \quad & C_6(i) = i + \max_1 - 1, \\
\text{Bottom neighbour} \quad & C_7(i) = i + \max_1, \\
\text{Bottom right neighbour} \quad & C_8(i) = i + \max_1 + 1.
\end{aligned}$$

3.2.5 Constructing the graphs

The algorithm is more formally started by initializing two graphs namely a *pulse graph* P_G and a *work graph* W_G. The *pulse graph* will consist of the finished decomposition containing all the extracted pulses of the DPT. The *work graph* is constructed by utilising the data points and connectivity functions and is reduced to a single node when the DPT decomposition is complete. The creation of these two graphs can be seen in Algorithm 3.

Algorithm 3 shows how to construct each node within the *work graph* and *pulse graph* by assigning specific values to each node. The $V_{W,0}$ is the *zero* node which connects to all the boundary nodes in the *work graph*. The virtual edges are connected to each pair of nodes considered in both graphs. The edges within the *work graph* are connected by using the connectivity functions defined earlier and is shown in Algorithm 4.

Algorithm 3 Construct Graph

procedure CONSTRUCT GRAPH($data, C_k(i)$)
  Create the empty work graph $W_G = (V_W, E_W)$
  Create the empty pulse graph $P_G = (V_P, A_P)$
  Create node $V_{W,0}$ in work graph W_G with ▷ The *Zero* node.
    $size \leftarrow \infty$
    $height \leftarrow 0$
  for every i in $data$ **do**
    Create node $V_{W,i}$ in work graph W_G with
      $size \leftarrow 1$ ▷ Nodes have multiple properties
      $height \leftarrow$value of data point
    Create node $V_{P,i}$ in pulse graph P_G with
      $size \leftarrow 0$ ▷ Show it present position data
      $height \leftarrow i$ ▷ i denotes index in data
    Create *VirtualEdge* $(V_{W,i}, V_{P,i})$
  end for
  SET CONNECTIVITY($W_G, C_k(i)$)
end procedure

Algorithm 4 Set Connectivity

procedure SET CONNECTIVITY($W_G, C_k(i)$)
  for every node $V_{W,i}$ in work graph W_G **do**
    for every k in $C_k(i)$ **do**
      $j_k \leftarrow C_k(i)$ ▷ Calculate relative index for node connection
      if j out of bounds **then**
        $j_k \leftarrow 0$ ▷ Connect to *Zero* node
      end if
      Create edge $(V_{W,i}, V_{W,j_k})$ in W_G
    end for
  end for
end procedure

3.2.6 The Feature Table

After creating the two graphs a feature table must be constructed from the *work graph*. This feature table is the crux of the algorithm and keeps a list of all the features currently in the *work graph*. This helps in reducing the requirement to traverse through the *work graph* in the search for features of the correct size. The creation of the feature table is shown in Algorithm 5.

Algorithm 5 Feature Table

```
1:  procedure FEATURE TABLE(W_G)
2:      Create empty FeatureTable
3:      for every node V_{W,i} in work graph W_G do
4:          if height of V_{W,i} = any height of neighbor V_{W,j} then
5:              size of V_{W,j} ← size of V_{W,j} + size of V_{W,i}
6:              Neighbor V_{W,j} inherit all neighbors of V_{W,i}
7:              Neighbor V_{W,j} inherit all VirtualEdges connected to V_{W,i}
8:              Delete V_{W,i} from work graph W_G
9:          else if (height of V_{W,i} > height of all neighbors)
                OR (height of V_{W,i} < height of all neighbors) then
10:             Add node V_{W,i} to FeatureTable
11:         end if
12:     end for
13: end procedure
```

While the *feature table* is created the nodes with equal-value neighbours are combined into single nodes presenting the clusters. Whenever a node has a neighbour with the same value they get combined. To combine two nodes, firstly the edge connecting them is removed after which all the other edges and virtual edges are transferred to one node while the other node gets removed from the graph.

An advantage to finding all the features in the beginning is that since new features cannot be created with the DPT, no further search for features are required in the algorithm. A current feature can only be amended to present a larger feature after it was flattened. The maximum number of possible features that can exist in the *feature table* at one instance is thus exposed in the beginning of the algorithm.

3.2.7 The DPT Decomposition

The *work graph* and *pulse graph* have been initialized, the *work graph* has been optimized by merging all the nodes with equal-valued neighbours forming clusters and a *feature table* has been created. The Discrete Pulse Transform can now be applied on the *work graph* which will then output a tree like structure in the *pulse graph*. The decomposition is seen in Algorithm 6.

Algorithm 6 demonstrates the execution of a Q_nP_n DPT decomposition which can be modified to a P_nQ_n DPT decomposition by applying Q_n first. The DPT decomposition is started by searching the feature table for any bumps of size 1. After all the bumps of size 1 in the feature table have been removed the pits of size 1 are removed. Each time a feature is removed the resulting node is checked to establish whether it is a feature or not. If the resulting node is a feature, the old feature is updated in the feature table. Otherwise, the old feature is deleted from the feature table.

The features are removed by sequentially increasing the scale. The scale can be increased if and only if there exist no features in the feature table of the current scale. The scale is increased until no features are left in the feature table.

Before removing any feature, it must be reaffirmed as a feature. By the nature of removing features arbitrarily, one can not be certain that a previously removed feature did not change the current node's status as a feature. When removing a feature, a node must be added into the *pulse graph*, which is achieved by using the virtual edges. After the removal of a feature two nodes must always be merged as the node which was a feature will now have an equivalent neighbour.

To reaffirm a feature as a feature, all the neighbours must be revisited to make sure that the feature is still a local minimum or a local maximum. If a neighbour has changed to an equal value, the current node must be merged with its neighbour and the *feature table* must be updated. This process is shown in Algorithm 7.

Algorithm 6 Discrete Pulse Transform

procedure DISCRETE PULSE TRANSFORM(W_G, P_G,*FeatureTable*)
  $scale \leftarrow 1$
  CNode $\leftarrow$ first node $V_{W,i}$ in *FeatureTable*
  while *FeatureTable* $\neq$ *empty* **do**
    if *CNode size* $=$ *scale* **then**
      if CHECK FEATURE(W_G,*FeatureTable*,*CNode*) **then**
        NodeIsPulse $\leftarrow$ *false*
        if *CNode* is a `min` feature **then** ▷ Apply P_n first
          if `max` feature with *size* $=$ *scale* $\nexists$ in *FeatureTable* **then**
            NodeIsPulse $\leftarrow$ *true*
          end if
        else if *CNode* is a `max` feature **then**
          NodeIsPulse $\leftarrow$ *true*
        end if
      end if
    end if
    if *NodeIsPulse* $=$ *true* **then**
      ADD PULSE(W_G, P_G,*FeatureTable*,*CNode*)
    end if
    if *FeatureTable* contains no features which *size* $=$ *scale* **then**
      Increase *scale*
    end if
    CNode $\leftarrow$ next node in *FeatureTable*
  end while
end procedure

Algorithm 7 Check Feature

1: **function** CHECK FEATURE(W_G,*FeatureTable*,*CNode*)
2: **if** *height* of *CNode* = any *height* of neighbor $V_{W,j}$ **then**
3: *size* of $V_{W,j} \leftarrow$ *size* of $V_{W,j}$ + *size* of *CNode*
4: Neighbour $V_{W,j}$ inherit all neighbors of *CNode*
5: Neighbour $V_{W,j}$ inherit all *VirtualEdges* connected to *CNode*
6: Delete *CNode* from *FeatureTable*
7: Delete *CNode* from work graph W_G
8: Add $V_{W,j}$ to *FeatureTable*
9: **return** *false*
10: **else if** *height* of *CNode* > *height* of all neighbors **then**
11: *CNode* is `max` feature
12: **return** *true*
13: **else if** *height* of *CNode* < *height* of all neighbors **then**
14: *CNode* is `min` feature
15: **return** *true*
16: **else**
17: Delete *CNode* from *FeatureTable*
18: **return** *false*
19: **end if**
20: **end function**

3.2.8 The Pulse Graph

The aim of the Discrete Pulse Transform is to extract pulses from the set of data. These pulses are represented in a graph structure which can be simplified to be a tree representation with the leaves as the *base nodes* and the last extracted pulse, the zero infinite node (the root). The leaves of the tree are initialised in the beginning and present the *position nodes* or *base nodes* of the data points. The branches of the tree are then created while performing the DPT decomposition. When the decomposition is completed the root of the tree is created. The root node represents all the data points including the zero padding nodes. A node is added to the tree whenever a feature is flattened within the *work graph*. To add the new node in the correct position, virtual edges are used to connect the *pulse graph* and the *work graph*. Algorithm 8 shows how an extracted pulse, thus a flattened feature in the *work graph*, is added to the *pulse graph* as a new node.

Algorithm 8 Add Pulse

1: **procedure** ADD PULSE($W_G, P_G, FeatureTable, CNode$)
2: $i, j, k \leftarrow$ arbitrary node indexes
3: $V_{W,j} \leftarrow$ neighbor of $CNode$ with nearest $height$
4: Create node $V_{P,k}$ in work graph P_G
$size \leftarrow size$ of $V_{W,j}$
$height \leftarrow Cnode\ height$ minus $V_{W,j}\ height$
5: **for** every $VirtualEdge\ (CNode, V_{P,p})$ connected to $CNode$ **do**
6: Create Arc $(V_{P,p}, V_{P,k})$
7: Delete $VirtualEdge\ (CNode, V_{P,p})$
8: **end for**
9: Add $VirtualEdge\ (V_{W,j}, V_{P,k})$
10: Delete $CNode$ from $FeatureTable$
11: Delete $CNode$ from work graph W_G
12: Add $V_{W,j}$ to $FeatureTable$
13: **end procedure**

3.2.9 Pulse Reconstruction

The biggest advantage of storing the pulses as described previously is the ease of reconstructing the original signal from the pulses. The original data can be reconstructed by two means. The first is to start the reconstruction from the leaves of the tree, in other words each data point gets reconstructed separately. We can call this reconstruction the *point reconstruction* of the DPT. The second method of reconstruction is to start at the root of the tree and reconstruct all the data points in a recursive way called *root reconstruction*. Pulse reconstruction does not only refer to reconstructing the original data but also to extract specific pulses in the DPT and present it in the format of the original data.

Starting at the leaves of the tree, Algorithm 9 can be used to reconstruct the original data by evaluating the value of each data point separately.

Algorithm 9 Point reconstruction of the DPT.

1: **function** PointReconstruction(P_G)
2: Let j be an arbitrary index and $data$ an array
3: **for** every $V_{P,i} \rightarrow size = 0$ **do**
4: Let $CNode = V_{P,i}$, $height = 0$
5: **while** Arc($CNode, V_{P,j}$) exist **do**
6: $height = height + CNode \rightarrow height$
7: $CNode = V_{P,i}$
8: **end while**
9: $data[V_{P,i} \rightarrow height] = height$
10: **end for**
11: **return** $data$
12: **end function**

Algorithm 9 is not the most effective way to reconstruct the original data from the DPT starting at the base nodes, but provides a way to integrate more advanced functions as shown in Algorithm 10. In the algorithm we use the properties of a base node to reconstruct the data. A base node in the *pulse graph* is denoted by having a *size* of zero and the *height* then denotes the index to which the node corresponds in the original data.

Algorithm 9 can be extended to support various additional functions. One such function is to

reconstruct pulses of a certain size or reconstruct only specific data points. In Algorithm 10 we show the reconstruction of a specific sized pulse, which can easily be adapted to reconstruct a range of pulses.

Algorithm 10 Conditional point reconstruction of pulses in the DPT.

function POINTRECONSTRUCTION(P_G, $RequiredSize$)
  Let j be an arbitrary index and $data$ an array
  for every $V_{P,i} \rightarrow size = 0$ **do**
    Let $CNode = V_{P,i}$, $height = 0$, $StopWhileLoop = false$
    while (Arc($CNode, V_{P,j}$) exist) **and** ($StopWhileLoop = false$) **do**
      if $CNode \rightarrow size = RequiredSize$ **then**
        $height = height + CNode \rightarrow height$
      else if $CNode \rightarrow size > RequiredSize$ **then**
        $StopWhileLoop = true$
      else
        $CNode = V_{P,i}$
      end if
    end while
    $data[V_{P,i} \rightarrow height] = height$
  end for
  return $data$
end function

A recursive way to reconstruct the original data can be used by starting at the root of the tree. This algorithm relies on using the computers stack to store the various variables. First we need to define the recursive function which is shown in Algorithm 11.

The initialization of recursive reconstruction of DPT is shown in Algorithm 12.

Opposed to the *point reconstruction*, the *root reconstruction* is not that easily modifiable to provide different reconstruction functions. One function that is suitable for the *root reconstruction* is the reconstruction of single pulses. Within that concept various queries can be made on a specific pulse using *root reconstruction*. Queries such as how many other individual pulses does a specific pulse contain, what is the spatial content of a specific pulse, and many other queries can be resolved with the *root reconstruction*.

Algorithm 11 Root reconstruction of DPT.

1: **function** RootReconstruction(P_G,$CNode$,$height$,$data$)
2: $height = height + CNode \rightarrow height$
3: **for** all j such that Arc($V_{P,j}$,$CNode$) exist **do**
4: **if** $V_{P,j} \rightarrow size = 0$ **then**
5: $data[V_{P,i} \rightarrow height] = height$
6: **return** $data$
7: **else**
8: $data$ = RecursiveFromRoot(P_G,$V_{P,j}$,$height$,$data$)
9: **end if**
10: **end for**
11: **return** $data$
12: **end function**

Algorithm 12 Root Reconstruct from the DPT root.

1: **function** RootReconstruction(P_G)
2: Let $data$ be an array
3: Let $height = 0$
4: Let $root = V_{P,k}$ for k such that $V_{P,k} \rightarrow size = \infty$
5: $data$ =RecursiveFromRoot(P_G,$root$,$height$,$data$)
6: **return** $data$
7: **end function**

3.3 PERFORMANCE EVALUATION

The development of a new algorithm requires a performance evaluation. Firstly, as stated before, the DPT can be computed in $O(n)$ time, thus meaning it can be done in linear time with regards to a set of input points. This was the case for the Roadmakers Algorithm which in theory should then also be the case for the Roadmakers Pavage.

To be able to test the algorithms performance, suitable data must be generated, easy interpretable data would make the evaluation easier such as utilising images. Images are currently one of the main application areas of the DPT. The generated data must closely approximate real-world images, as the DPT will mostly be applied to real-world images. A solution would be to use real-world images of different sizes, but a large image database is required containing the range of image sizes over which the algorithm is to be tested. Such a database is not available, thus we need to generate data which closely approximate the execution times of a real image. To find a suitable way to generate data, we use four different generation methods in a 2-dimensional space with 4-connectivity. Images usually uses 4-connectivity and a 2-dimensional space can be visualised as an image. We will generate data for an 8-bit image, meaning we only use values from 0 to 255. Firstly, we use a random number generator using a uniform distribution over 0 to 255 for every pixel or element. Secondly, we use the images from the Berkley Segmentation Dataset [26]. Thirdly, we use a uniform distribution for every element, but limit the distribution within a range of a previous generated values. The third method is seen in Algorithm 13. Lastly, we count through all possible values by increasing the value of the previous pixel value by 1. For method four to work, the image's length or width must not be multiples of 256. The fourth method can be seen in Algorithm 14.

Algorithm 13 The third method

1: **for** every pixel i **do**
2: $pixel(i)$ = Number from Uniform Distribution $[pixel(i-1)-5, pixel(i-1)+5]$.
3: **if** $pixel(i) > 255$ or $pixel(i) < 0$ **then**
4: $pixel(i)$ = Number from Uniform Distribution [0, 255]
5: **end if**
6: **end for**

Generating data with these four methods we can see how the execution time differ. 50 images where generated with each method. The Roadmakers Pavage was executed on each image,

Algorithm 14 The fourth method

```
1: for every pixel i do
2:     pixel(i) = pixel(i − 1) + 1
3:     if pixel(i) > 255 then
4:         pixel(i) = 0
5:     end if
6: end for
```

and an average execution time for every method were calculated. The average times with example images of the generation method are shown in Figure 3.11.

(a) Image for Method 1 with an average execution time of 286 ms

(b) Image for Method 2 with an average execution time of 960 ms

(c) Image for Method 3 with an average execution time of 1280 ms

(d) Image for Method 4 with an average execution time of 2700 ms

Figure 3.11: Examples of generated data with the average DPT execution time

Evaluating the average execution times of each method we observe that method three in Figure 3.11c gives the best execution time approximation to real world images given in Figure 3.11b. Generating data with method four provides possibly worst case results for the execution time of the DPT, but other methods must be investigated. Method one gives the fastest execution time which is expected when every data element differ by large amounts as only a few larger pulses is formed. For the rest of the evaluation tests we will use generation method three as it approximate real world images the closest.

The Roadmakers Pavage and the Roadmakers algorithm are compared in Figure 3.12. The algorithm used for the Roadmakers algorithm is the original algorithm implemented in Python [1]. Figure 3.12 show that the current Roadmakers Pavage implementation is signific-

antly faster than the original Roadmakers Algorithm implementation.

We will now look at the computational time of multi-dimensional data with different number of connections per node, to demonstrate linearity with regards to the number of nodes (data points) and to show how the chosen connectivity influence computational time. Different data sets are generated contain different number of data elements starting at 1000 data points up to one million data points. Ten images where created for every selected number of data points producing a maximum, minimum and average execution time.

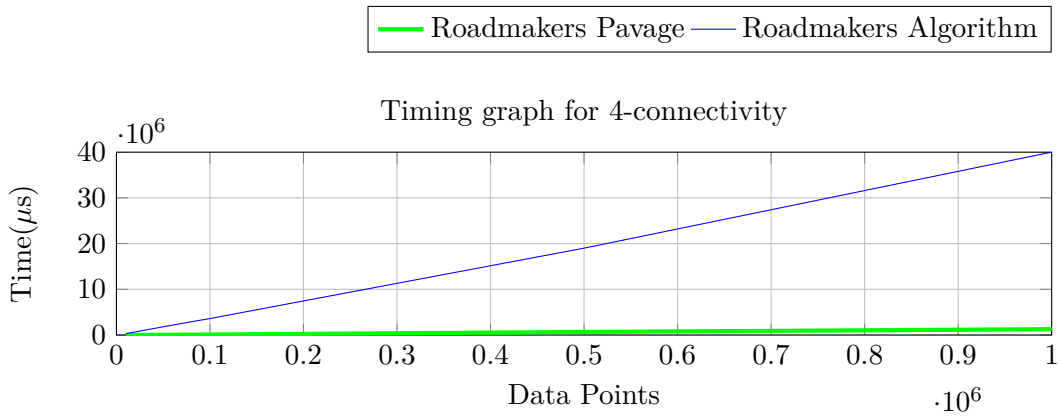


Figure 3.12: Comparison between Roadmakers Algorithm and Roadmakers Pavage

The number of connections per node is selected by doubling the previous amount thus 2, 4, 8 and 16. Using two connections per node can be interpreted as choosing 2-connectivity, or choosing eight connections per node can be interpreted as an 8-connectivity. Although sixteen connections per node can not be related directly to some connectivity, it provides insight in how the computation time scales with the change in connections per node.

It is important to show the minimum, maximum and average computation times to provide better insight into the execution times. These graphs are not to compare the algorithm in speed to other algorithms as hardware architecture, processor speed and operating system has a influence in the algorithms execution time. The graphs are specifically to show the Roadmakers Pavage linearity. The separate graphs for the four different number of connections per node are shown in Figure 3.13, where the number of data points is plotted against the computation time.

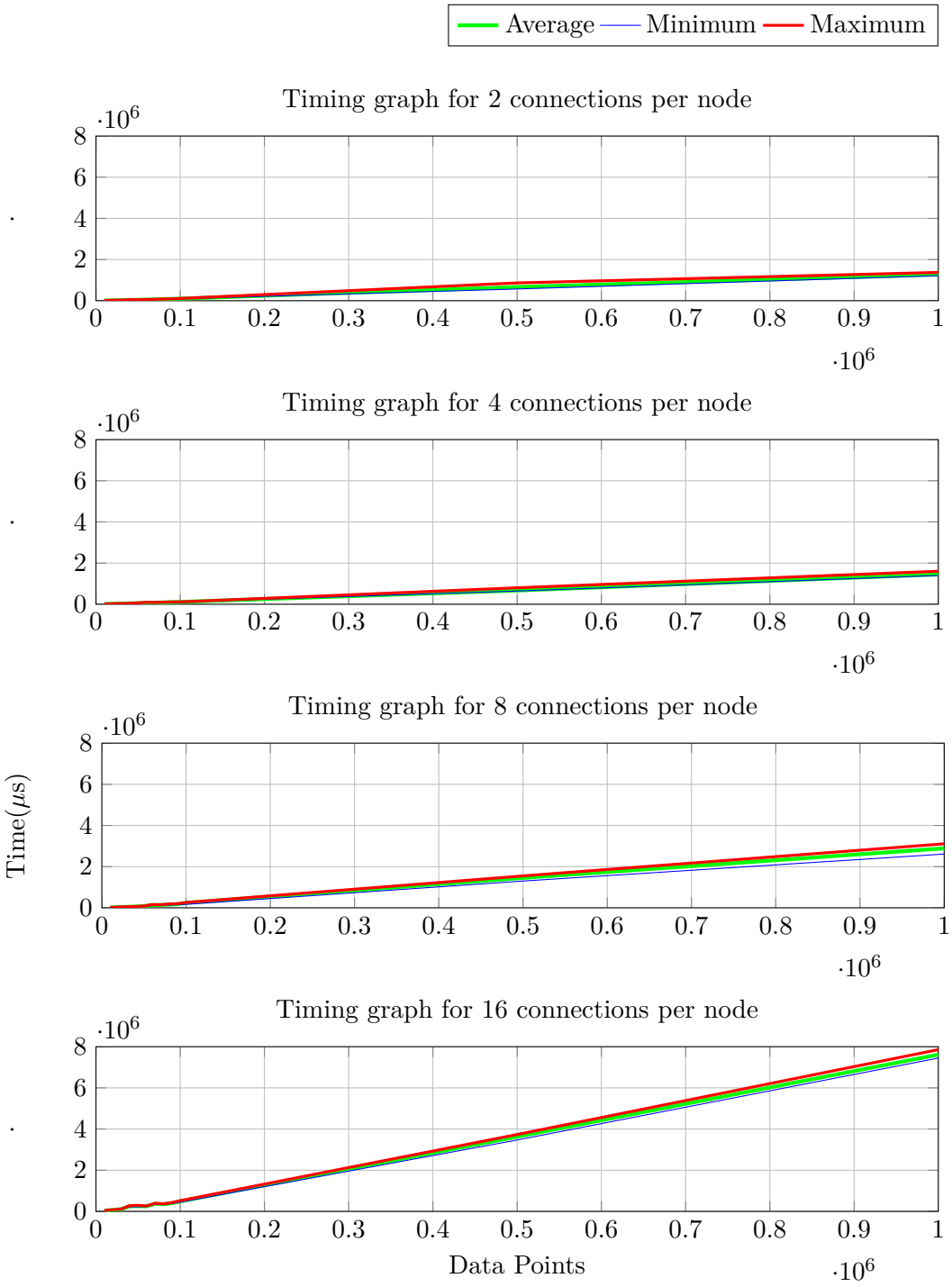


Figure 3.13: Timing diagram using number of data points

In Figure 3.13 the computation time increases linearly with the increase in the number of data points. It is clear that in changing the number of connections per node, the computation time differ significantly. These graphs verify the $O(n)$ complexity of the algorithm and show that changing the number of connections per node changes the execution time exponentially.

3.4 IMPLEMENTATION DETAIL

Although the algorithm was fully explained in Section 3.2 the implementation of the algorithm still has a lot of detaill. The complete code can be found online in the GitHub repository with full descriptions of the whole algorithm as implemented [22]. The DPT library guide is attached in Appendix A.

3.5 CONCLUSION

In this chapter the author developed a new algorithm for the Discrete Pulse Transform called the Roadmakers Pavage. The Roadmakers Pavage is implemented to provide a more intuitive way of working with pulses and also provide faster execution times for the DPT in n-dimensional space. The algorithm is provided in the form of a library, the DPT Library [22], which is there to facilitate future research in the Discrete Pulse Transform domain.

In the next chapters we consider applications of the DPT. We first look at resolving an issued called *leakage* within the DPT domain. This is followed by looking at the potential of the DPT in image segmentation. All the following chapters utilises the DPT library.

CHAPTER 4

PULSE REFORMATION

Pulse Reformation is the proposed framework in which we solve the leakage problem within the Discrete Pulse Transform. We revisit the concept of connectivity in more depth in which we discuss what the leakage problem is and how others have attempted to solve it. We also give some experimental results using the Pulse Reformation framework.

Mathematical Morphology is the manipulation of geometrical structures in set theory and lattice theory. The concept of connectivity was introduced by Serra and Matheron [13]. Following a more general concept of connectivity one can say that a topological set is disconnected if it is equal to the union of two disjoint nonempty open sets [27]. In discrete spaces every element is an open set, thus creates a dilemma in conjunction to the fact that elements in a lattice of higher order than 1 forms only a partially ordered set. Let $x_i \in \mathbb{Z}$ then it is easy to say that $[x_i, x_j]$ for some $i < j$ forms a connected set as all $x_k \in [x_i, x_j]$ for $i \leq k \leq j$ because $\mathbb{Z}$ is a total order. In $\mathbb{Z}$ it is evident that for $x_i \in \mathbb{Z}$ has two neighbours x_{i-1} and x_{i+1}. If we extend the space to $\mathbb{Z}^d$ it becomes difficult to specify a connected set if each element's neighbours are not identifiable as an ordered space.

Given a universal set E the collection of subsets of E is denoted by $\mathcal{P}(E)$. We can then say that $X \in \mathcal{P}(E)$ or $X \subseteq E$. We use the axiomatic connectivity defined in Definition 20. In image analysis the most obvious connectivity to make use of is 4- or 8-connectivity which are graph connectivities with neighbourhood relations defined.

4.1 THE LEAKAGE PROBLEM

The leakage problem informally is when a space has for example two disconnected sets. After a certain operation on the space, these two sets become connected as a by-product of the operator. Usually one of the by-products of the operator is the creation of a new connected set which is connected to the two previously disconnected sets, which now form a connected set. This phenomena can be seen in Figure 4.1, illustrating leakage between the ball and the seal where actually their are two different objects but appear as one connected set.

Figure 4.1: Leakage illustrated between the seal and the ball.

In image processing leakage can result from various occurrences. A good philosophy of why leakage occur in images is given by O'Callaghan and Bull [28]. According to them leakage occurs due to the existence of weak points in the gradient of object boundaries. These weak boundaries does not stop the segments from flowing into the background or other significant partitions.

Leakage is also interpreted as over segmentation. Li and Wilson [29] proposed the usage of multi-resolution techniques in conjunction with Markov random processes when doing texture segmentation to stop leakage. Image segmentation with partitioning of connected components based on openings is prone to over segmentation. A proposed solution is to treat all singletons generated by the operator as elements from larger connected components [30]. The larger connected components refer to connectivity classes in higher dimensional space which are extensions of the normally defined connectivity classes in mathematical morphology [31].

Other attempts at creating solutions to the leakage problem are by redefining existing con-

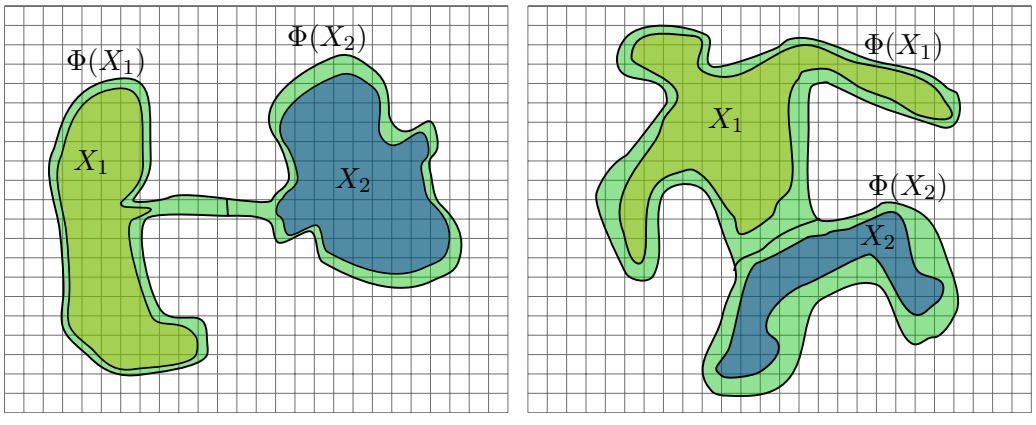
nectivities. Salembier and Oliveras [32] defined a pseudo-connectivity which either increases or decreases the number of neighbourhood relations which allow them to solve leakage in certain cases. A more successful adaptation of the connectivity definition was done by Tzafestas and Maragos [33] by using a multiscale connectivity and a generalized connectivity measure with which they determine to what degree the leaking set must be connected. Some second generation connectivity was also defined for general leakage reduction on any filter that operate on connected components called connected filters [34]. Another attempt at reducing leakage is the definition of stopping criteria in morphological opening and opening by reconstruction [35].

Leakage exist in other image processing spaces such as the active-contour model where possible reduction in leakage is to estimate the position of possible edges in the image by minimal weighted local variance [36]. Graham et al [37] used adaptive parameters within the active-contour model to possibly stop estimated leakage.

Leakage can then more formally be defined as:

Definition 26. *Let* $\Phi : \mathbb{Z}^d \rightarrow \mathbb{Z}^d$ *be an arbitrary operator. If there exists* $\left\{X_i \in \mathbb{Z}^d, i \in I, X_i \in \mathcal{C}\right\}$ *with* $\cap_{i\in I} X_i = \emptyset$ *such that* $\cup_{i\in I} \Phi\left(X_i\right) \in \mathcal{C}$ *then* Φ *is said to have caused leakage.*

An operator Φ may of course satisfy Definition 26 in different mannerisms. Consider Figure 4.2 for images.



(a) **(b)**

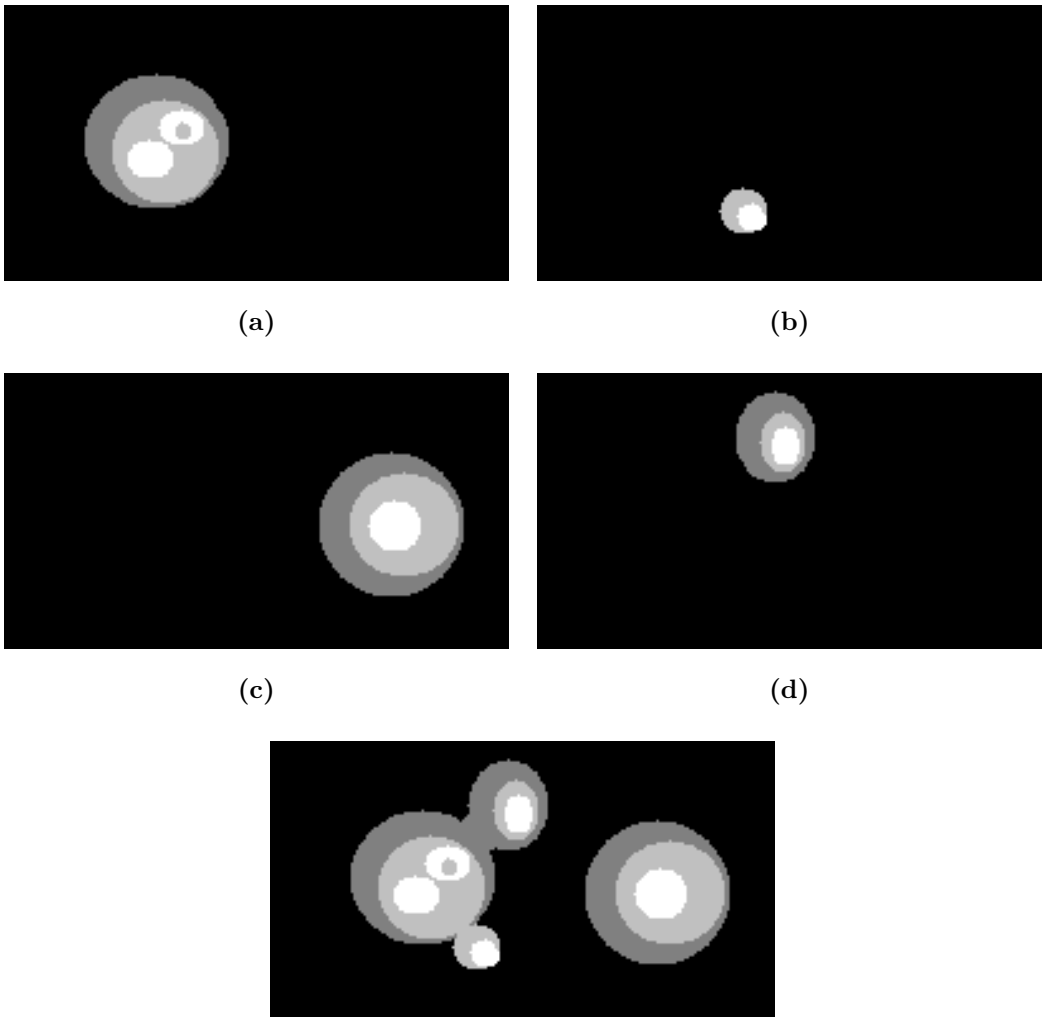
Figure 4.2: Different kinds of leakage.

In Figure 4.2a the leakage is clearly undesired as X_1 and X_2 should be two separate objects. However, in Figure 4.2b the operator has likely connected two objects which should be one in cases such when smoothing is applied to an image. We thus define a measure of leakage as follows.

Definition 27. *The strength of a leakage is measured as* $\texttt{card}(A_{Strength})$ *where:*

$$A_{Strength} = \{\{x_i, x_j\} : \{x_i, x_j\} \in \mathcal{C}, x_i \in \Phi(X_i), x_j \in \Phi(X_j), i \neq j, ij \in I\} \tag{4.1}$$

The larger the strength of the leakage the larger the undesired effect is on the image. It is clear than in the case of a good smoother desired leakage will occur.



(a) **(b)**

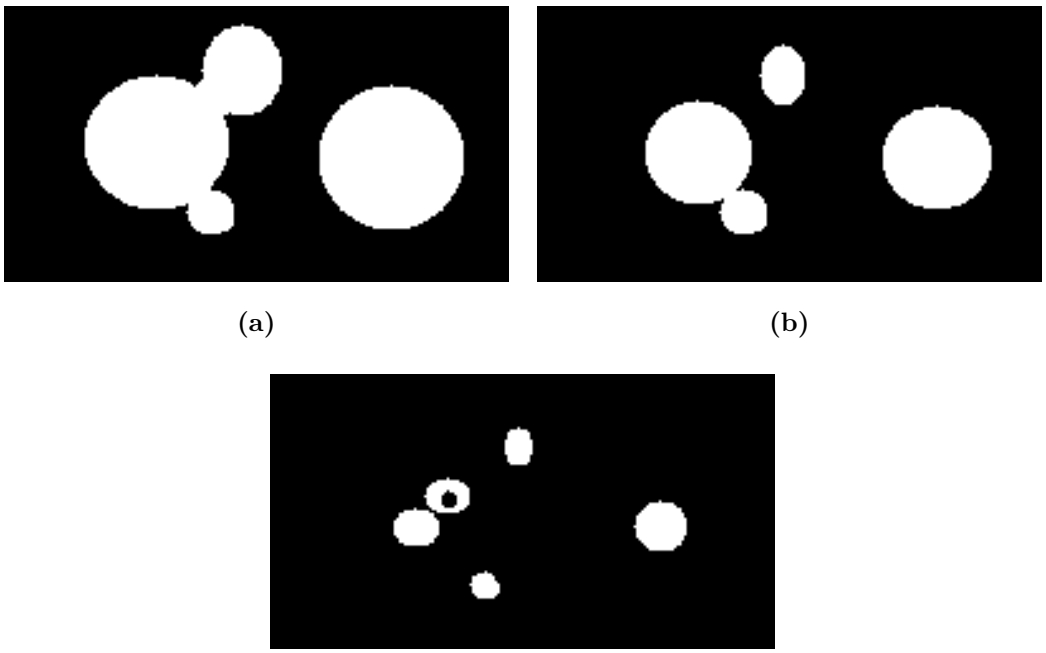
(c) **(d)**

(e) Combined objects

Figure 4.3: Synthetic Objects in an image

A synthetic example can be created to clearly demonstrate the leakage problem. We use four separate objects with varying internal intensities shown in Figure 4.3 and combine them into one image. We would like to extract the four objects.

To extract the different objects we can use connected components to indicate individual objects. A set consisting of connected components will then denote one object. Two simplistic methods can be used. The first is the use of thresholding. The synthetic image can be thresholded at three different levels as the image only consist of four discrete grey levels. The thresholded images are shown in Figure 4.4.

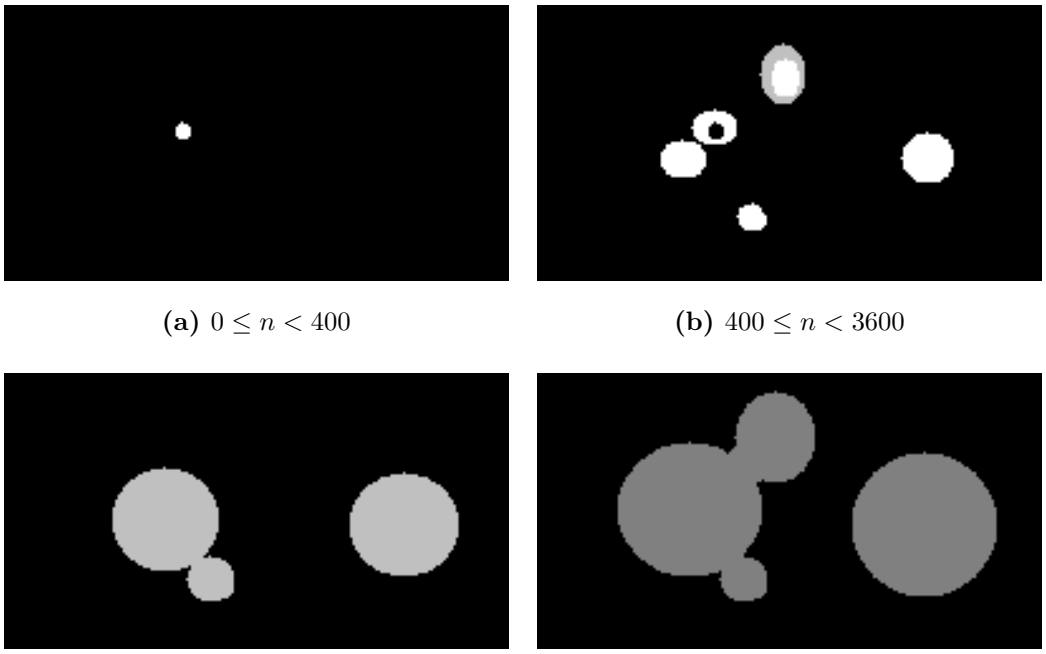


(a) **(b)**

(c)

Figure 4.4: Synthetic image thresholded at 3 different levels.

In Figure 4.4 it is clear that no threshold will yield 4 connected sets which will cohere with the four original objects. Another way, is to use the LULU scale-space and threshold different pulse sizes. We choose four different pulse ranges which are shown in Figure 4.5.



(a) $0 \leq n < 400$ **(b)** $400 \leq n < 3600$

(c) $3600 \leq n < 6400$ **(d)** $6400 \leq n < 21800$

Figure 4.5: Synthetic images thresholded for different pulse ranges.

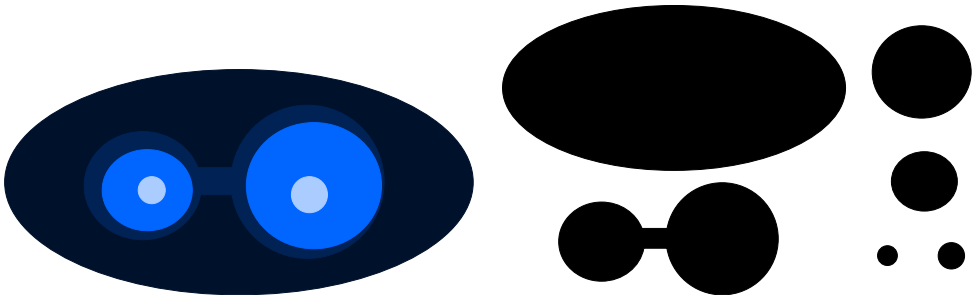
Even by using the DPT scale space it is not possible to extract four connected components that will yield the required connected sets. In Figure 4.4 and Figure 4.5 leakage is evident in most of the thresholded images. Although technically there exist many other ways to possibly extract the objects, this problem was only used to illustrate leakage in simple connected component and DPT framework. In the next section we describe a proposed method to eliminate leakage within the DPT framework.

4.2 THE PROPOSED FRAMEWORK

Leakage occurs in the Discrete Pulse Transform domain. To explain the proposed framework we can visualise the leakage problem as a box of brittle magnets. The task is to successfully remove all the magnets from the box and place the individuals in a row. The problem then lies in identifying individual magnets. Two or more individual magnets can be stuck together which must be pulled apart. However an individual magnet can only be separated by breaking it. By looking at structural cues we can separate these magnets, such as if two balls are stuck together they most probably must be separated. If two cubes, unaligned, are stuck together

we can assume they must be separate. If a sphere and a pyramid are stuck together they must probably be separated. We can thus continue in this fashion for all kinds of known shapes and say with high probability that these different structures do not fit together.

The DPT extracts pulses at various scales (refer to Section 2.2.3). In Figure 4.6 we demonstrate the decomposition of a simple image into its various pulses.



(a) DPT input image **(b)** DPT output pulse

Figure 4.6: A DPT example decomposition. The pulses do not show any variation in height. The difference between the various luminosities in (a) are identical.

The scales from the DPT can be stacked from largest to smallest scale forming the LULU scale-space. A visual demonstration is shown in Figure 4.7. The pulses in Figure 4.7 are said to form a stack defined by the scale-space neighbourhood relation given below in Definition 28 and Definition 29.

Definition 28. *Two arbitrary DPT pulses, ψ_{ns_2} and ψ_{ms_1} with $n < m$, are called scale-space neighbours if $\psi_{ns_2} \subset \psi_{ms_1}$ and for any other DPT pulse ψ_{ps_3}, $n < p < m$ we have $\psi_{ps_3} \cap \psi_{ns_2} = \emptyset$.*

Where the strength of the scale-space neighbour relation is measured as the inverse of the difference in cardinality of the two pulses ϕ_{ns_2} and ϕ_{ms_1}, naturally $\frac{1}{m-n}$.

Definition 29. *A collection of DPT pulses are said to form a stack if they are each scale-space neighbours of at least one other pulse in the collection.*

In Figure 4.7 the pulses illustrated form a stack. The Pulse Reformation algorithm will obtain the true pulses $\mathcal{R}_{ns_{21}}$, $\mathcal{R}_{ns_{22}}$ and $\mathcal{R}_{ns_{23}}$.

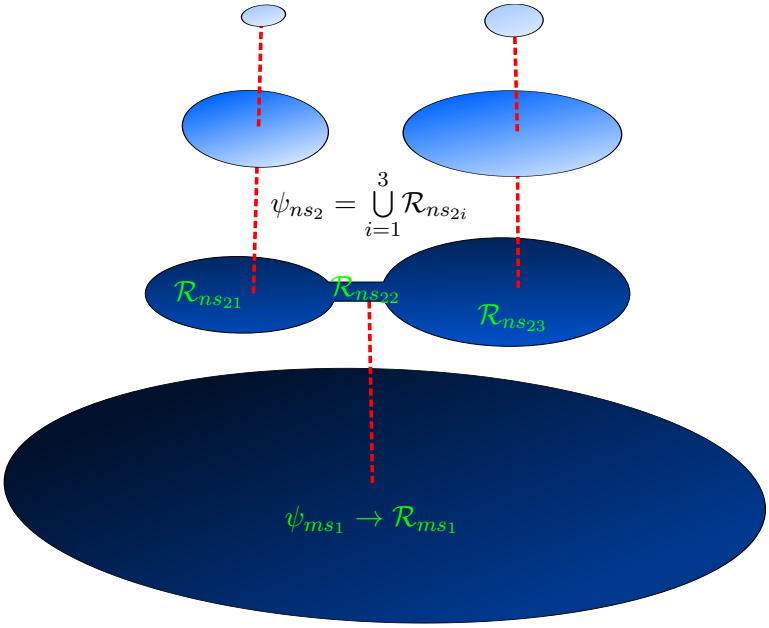


Figure 4.7: A DPT pulses stacked in different scale

By using this definition we can say that every pulse consists of regions $\psi_{ns} = \bigcup_{i=1}^{p} \mathcal{R}_{ns_k}$. This is demonstrated in Figure 4.8. Assume that Figure 4.8 is an image of two separate balls, thus two individual objects. Inspecting Figure 4.8, only one object is observed, the full pulse ψ_{ns_2}. The two objects are linked by a third region. The third object is then referred to as a leakage region, which on its own can possibly also be an object or only noise.

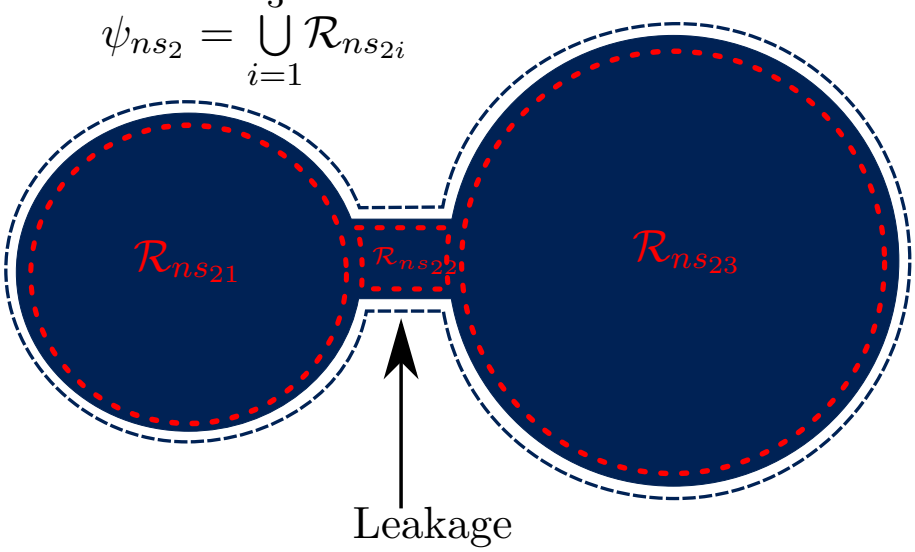


Figure 4.8: A pulse extracted by the DPT showing the possible regions which the pulse consists of.

If we want to eliminate leakage we need to estimate the true regions $\mathcal{R}_{ns_{21}}$, $\mathcal{R}_{ns_{22}}$ and $\mathcal{R}_{ns_{23}}$.

Although we have defined leakage more formally above it is largely a subjective matter in the literature. We aim to, in the LULU scale-space, objectively eliminate leakage. In case of the magnet box we aim at finding all the rigid shapes with the most probable shape having the least amount of edges. We can then objectively eliminate leakage.

The proposed framework will be developed using circular probes, although other shapes can also used within the framework. The LULU scale-space will be transformed in such a way that circular objects will form the strongest joined stack. A *joined stack* is formed when a group of scale-space neighbours forming a stack also cohere to an additional requirement. The additional requirement involves a *principle point* $\tilde{\mathcal{R}}_{ns_k}$ for each region $\mathcal{R}_{ns_k}$. For development of the framework we will define our aim as the identification of circular objects in the scale-space.

The principle point needs to somehow capture the core structure of the region. In case of circular objects, the center point at the arithmetic mean in terms of the x-position and the y-position, is always surrounded by edges and lies within itself. By using this reasoning we can assume that the center of a circle will capture the core purpose of circular objects. The circular object can also be reconstructed from the principle point by iteratively increasing the radius of the circle centred at the principle point. In general if using any shaped probe, the principle point $\tilde{\mathcal{R}}_{ns_k}$ should be:

- Scale Invariant
- Translation invariant
- Rotation invariant
- Affine invariant

A few examples are shown in Figure 4.9. The red dot shows the principle point and the dark blue shows elements part of the geometrical set. The circle's principle point has been discussed above. The principle point of the doughnut shape in Figure 4.9 can be defined as the center of a circle which is not contained within the set but is surrounded by a continuous

edge. The principle point of a concave mirror shape should lie at the focal point of the concave side of the set. The principle point of a triangle should lie in the middle of the shortest edge. From all of these principle points the objects can be reconstructed by knowing one extra parameter such as the radius or distance of a corner or edge in the set.

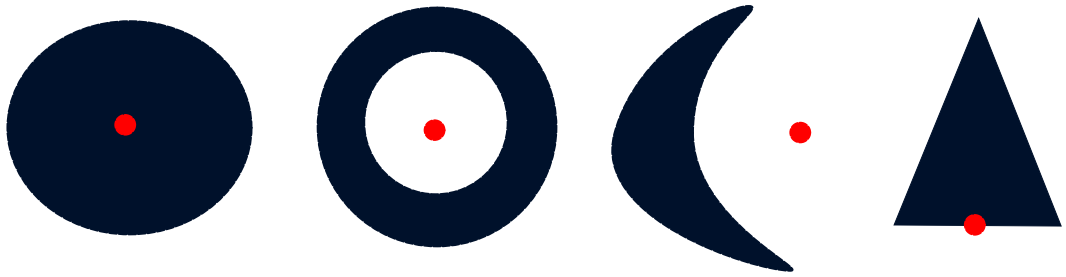


Figure 4.9: Examples of possible principle points denoted by the red dot.

To estimate a region's principle point, iteratively erode the pulse until the next erosion yields an empty set. Figure 4.10 illustrates this. The last non-empty erosion will represent the principle point. The region can then be reconstructed by dilating the principle point until the defined energy function E_{ns} below is minimised.

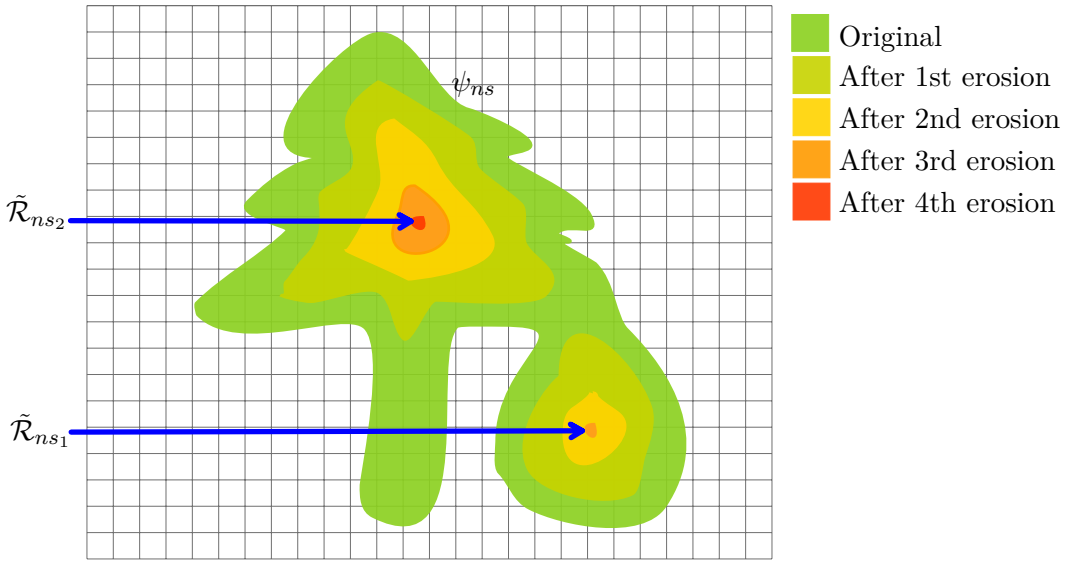


Figure 4.10: Example of finding the principle points in a pulse.

Each region $\mathcal{R}_{ns_k}$ will have a principle point. Therefore each pulse can contain multiple principle points. The regions must adhere to the boundary conditions of the DPT scale-

space thus:

$$\mathcal{R}_{ns_k} = (\mathcal{R}_{ns_k} \cap \psi_{ns}) \,/\, \bigcup_{k \neq i} (\mathcal{R}_{ns_i}). \tag{4.2}$$

The regions within a pulse can only consist of unique elements thus every element in a pulse can only be assigned to one region. From the principle point each region is reconstructed by minimizing the energy function E_{ns}. For the circular probes we can define an energy function:

$$E_{ns} = \sum_{k=1}^{p} \sum_{t=1}^{\kappa(s_k)} \frac{\texttt{card}\{\zeta(t-1, \tilde{\mathcal{R}}_{ns_k})\}}{|\texttt{card}\{\zeta(t, \tilde{\mathcal{R}}_{ns_k})\} - \texttt{card}\{\zeta(t-1, \tilde{\mathcal{R}}_{ns_k})\}|} \tag{4.3}$$

where $\zeta(t, \tilde{\mathcal{R}}_{ns_k})$ denotes a set containing all the elements within a circle of radius t centred at $\tilde{\mathcal{R}}_{ns_k}$, the t^{th} dilation of the circular set centred at the principle point, while staying within the boundary of the pulse; where p is the number of regions; and $\kappa(s_k)$ the appropriate t number of dilations where t is a geometrical parameter. The energy function determines the best circles centred at the principle points associating each pulse element to a region, restricted to the boundary conditions in Equation 4.2.

We have created regions within each pulse and founded a principle point for each region. Next we need to relate regions at different scales to one another. Each region has the apparent same defining principle point which should then be positioned at a known position between scales. Specifically for circular objects all the principle points should be at the same geometrical position. We can thus say that two or more regions form a joined stack if the principle points are joined and the regions form a stack defined in Definition 29. The definition of joined principle points is given in Definition 30 where the definition of a joined stack is given in Definition 31

Definition 30. *Two scale-space neighbours ψ_{ns} and ψ_{ms} with $n < m$ containing regions $\mathcal{R}_{ns_j}$ and $\mathcal{R}_{ms_i}$ with geometrical parameters r_j, r_i and principle points $\tilde{\mathcal{R}}_{ns_j}$, $\tilde{\mathcal{R}}_{ms_i}$. The principle points are said to be joined if $J(\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}) < \epsilon = \mathcal{N}(\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}, r_i, r_j)$ where J is the joining function and $\mathcal{N}$ is a noise function.*

The joining function J provides a relation between the two principle points and can be any type of polygon or line. The noise function $\mathcal{N}$ provides a measure of how similar the two regions are based on the expected relative position of the principle points. For the

circular case we define $r_j = \kappa(s_j)$, $r_i = \kappa(s_i)$ and $J(\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}) = \|\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}\|$ with $\|\cdot,\cdot\|$ giving the euclidean distance between the two points. We also define our noise function $\mathcal{N}(\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}, \kappa(s_i), \kappa(s_j)) = c * (\texttt{card}\{\mathcal{R}_{ns_i}\}/\texttt{card}\{\mathcal{R}_{ms_j}\}) * (\kappa(s_i)/\kappa(s_j))$.

Definition 31. *Two regions $\mathcal{R}_{ns_j}$ and $\mathcal{R}_{ms_i}$ with $n < m$ form a joined stack if their principle points are joined and $\mathcal{R}_{ns_j} \subseteq \mathcal{R}_{ms_i}$.*

This definition is visually shown in Figure 4.11 where $J(\tilde{\mathcal{R}}_{ns_j}, \tilde{\mathcal{R}}_{ms_i}) = a$. The ϵ can be interpreted as a noise canceller within the LULU scale-space. If a perfect scale-space was constructed all the pulses forming joinings belong to the same object and will have aligned principle points. In Figure 4.11 the large coloured dots denote the principle point in each region.

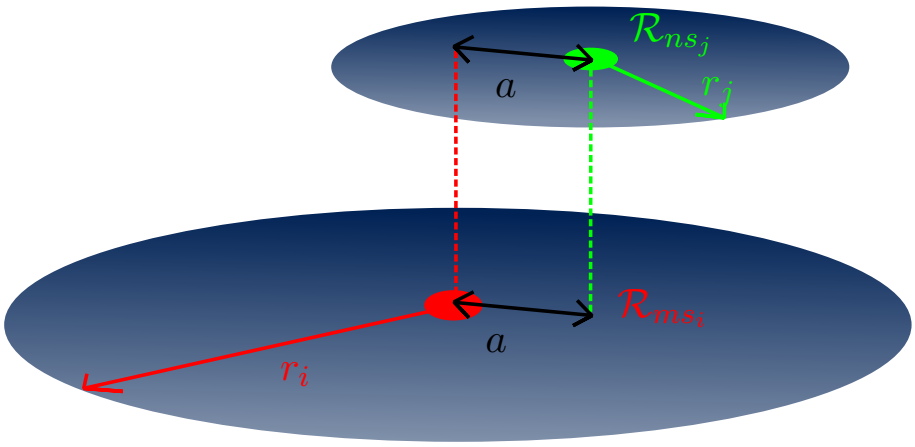


Figure 4.11: The joining of two arbitrary pulses

The strength of the joining of the two regions can be measured as the strength of a scale-space neighbour. The smaller the difference in cardinality of the two regions, the stronger the joining becomes. An additional strength measure can be added such as the variation of the principle points from the expected distance.

We have now shown that we can estimate regions from pulses. We estimate a region from the first estimated principle point and then re-estimate the principle point using the region. This process can be repeated until the principle point moves less than the noise function $\mathcal{N}$ and E_{ns} is minimized.

The estimation of the regions within pulses can be represented by a four corner model. The

four nodes are the *Principle Point* node, the *Pulse* node, the *Joining* node and the *Regions* node. The nodes are shown in Figure 4.12. Each node's name is self-evident of the represented data at the node. The arrows present the process that needs to be executed to transform one set of data to another.

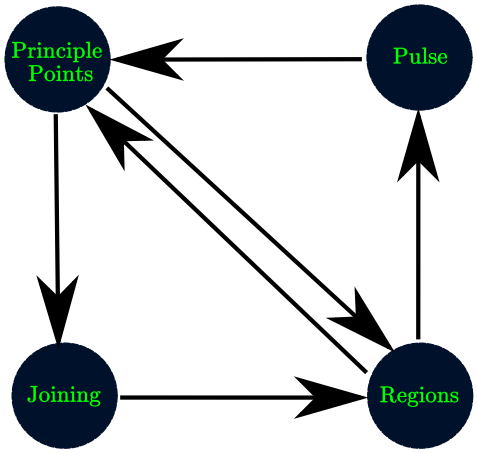


Figure 4.12: Region based Pulse Reformation Model.

The *Pulse* node can be used to estimate the *Principle Points* node. The *Principle Points* node is used to estimate the *Regions* and to determine joining with other scale-space neighbour regions. The *Joining* node can be used to also estimate the *Regions* node. The *Regions* node can estimate the *Principle Points* as well as the *Pulse* node. This whole process follows an iterative nature.

To create a more robust estimation of the regions we need to estimate each region by including the structure of the complete LULU scale-space. The Pulse Reformation framework uses an iterative approach to first estimate an initial principle point which is used to estimate the regions within each pulse. The principle points that form a joined stack can be used to adjust principle points and re-estimate the regions until all energy functions E_{ns} and all joining functions $J(\cdot)$ have been minimized. This combined iterative model is represented in Figure 4.13.

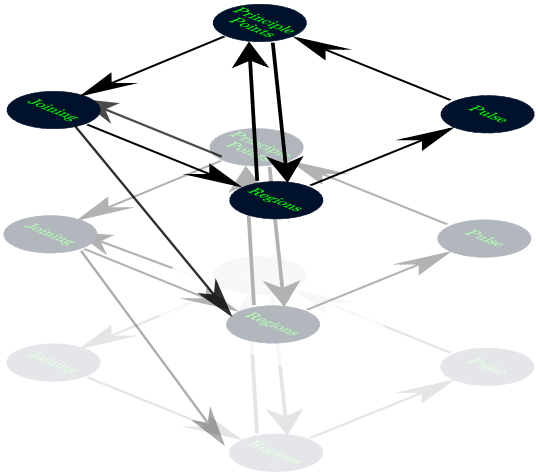


Figure 4.13: Region based Pulse Reformation Model.

After the Pulse Reformation framework has been applied the LULU scale-space consists of various joined stacks. Each joined stack will have a *joined stack strength* which is calculated by the sum of the strength of the scale-space neighbours within the joined stack divided by the number of regions contained in the joined stack. The strongest joined stacks will then present the most salient objects.

To be able to use the Pulse Reformation framework four main things need to be defined:

1. The Principle Point Estimator $\tilde{\mathcal{R}}_{ns_k}$.

2. The Energy function E_{ns} which needs to be minimized.

3. The joining function $J(\cdot)$.

4. The noise function or principle point alignment $\epsilon = f(\cdot)$.

Each of these functions have already been defined for the case of circular objects. We use the Pulse Reformation framework to develop Algorithm 15 for circular objects.

Algorithm 15 Pulse Reformation for circular object

Estimate principle points and regions for each pulse in the DPT scale-space.
repeat
    from the smallest region **upto** the largest region.
    if region's principle point forms a joining **then**
        Move the principle point such that $J(\cdot) = 0$.
        if the energy function is not reduced **then**
            Move the principle point back.
        if region is joined to a region on a larger scale **then**
            Recalculate the energy function.
            - Restrict current region to region on larger scale.
    else
        for every principle point in pulse of current region **do**
            Move the principle point towards another principle point.
            - Chosen principle points must be part of a scale-space neighbour.
            - Move distance is equal to the equivalent noise function.
            if the energy function is not reduced **then**
                Move principle point back.
        end for
    endif
until no principle points can move.

We can now apply the Pulse Reformation framework to the synthetic image created in Figure 4.3. In Figure 4.14 the output of the framework is shown. Four different objects were extracted each coinciding with a synthetic object. Each extracted object shows protrusion where leakage occurred. This is where the framework has an uncertainty as to which principle point the region elements belong.

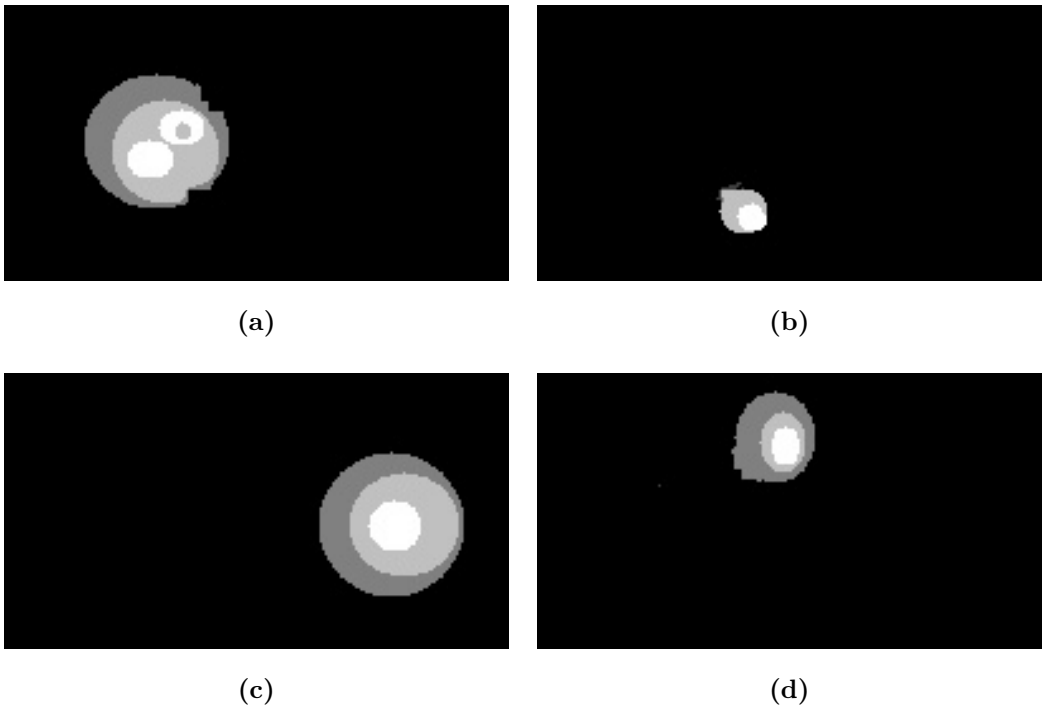


Figure 4.14: The Pulse Reformation of the synthetic image.

The extracted objects can easily be thresholded just above zero to provide four discrete connected sets which then coincide with the original objects. The Pulse Reformation framework can now be tested on a real world image.

4.3 EXPERIMENTAL EVALUATION

4.3.1 Leakage reduction

The results of this section are published in the paper [6].

The Pulse Reformation framework will be applied to the LULU scale-space and the strongest joined stacks will be extracted. The experimental image to be used is shown in Figure 4.15. It contains 6 distinct blood cells. A blood cell image has a large amount of object leakage, such that certain blood cells may touch one another.

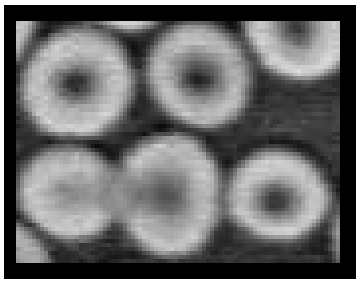


Figure 4.15: Test Image of blood cells.

The 6 strongest objects extracted with the Pulse Reformation algorithm is shown in Figure 4.16 including the original image. As we can see all 6 blood cells are extracted as separate objects.

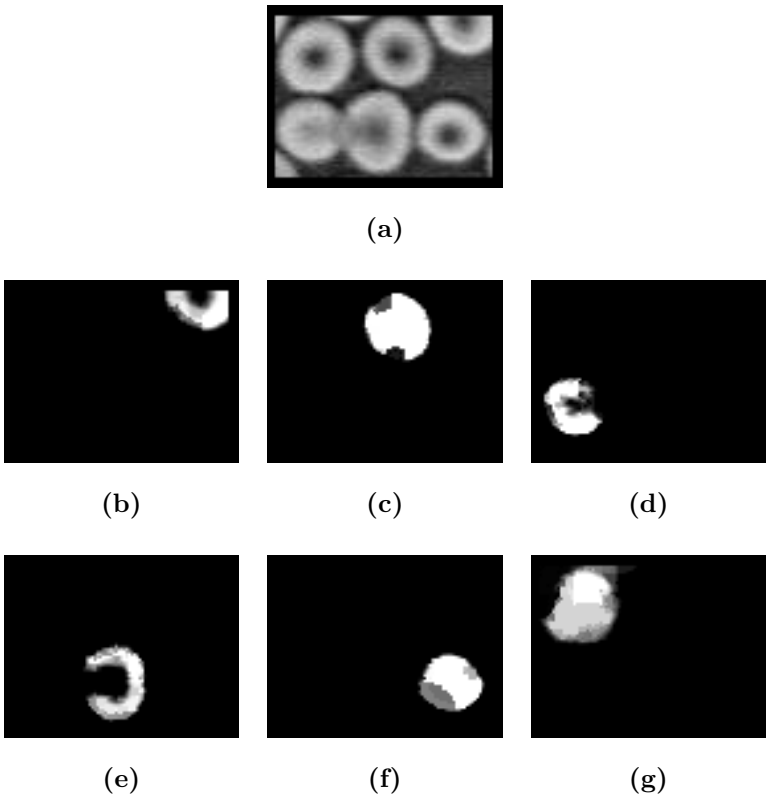
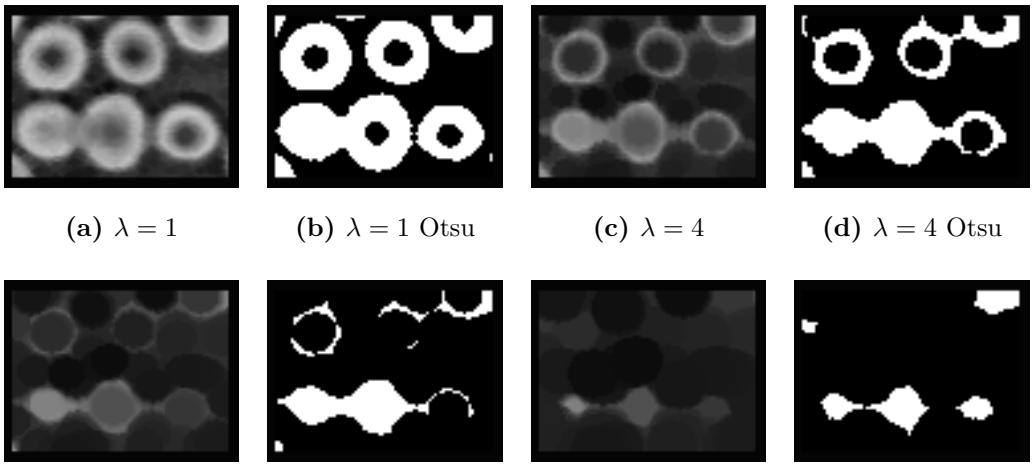


Figure 4.16: The Pulse Reformation of a blood cell image.

Before we discuss the results of the Pulse Reformation framework we can compare the results to another technique which is used to combat leakage in an equivalent case. The λ-

connected components method [38] will be used in conjunction with a thresholding method, namely Otsu's method [39]. The λ-connected components are those in which the center of a disk structuring element of radius λ can be moved along a continuous path throughout the connected component such that the entire disk stays within the domain of the connected component. The results of the λ-connected components are shown in Figure 4.17.



(a) $\lambda = 1$ **(b)** $\lambda = 1$ Otsu **(c)** $\lambda = 4$ **(d)** $\lambda = 4$ Otsu

(e) $\lambda = 6$ **(f)** $\lambda = 6$ Otsu **(g)** $\lambda = 9$ **(h)** $\lambda = 9$ Otsu

Figure 4.17: λ-connected components with and without thresholding.

As we can see only $\lambda = 1$ with thresholding almost provides six discrete connected sets which coincide with the 6 blood cells. Figure 4.17 (c)-(h) don't extract any meaningful objects.

As in the tested synthetic image the blood cells in Figure 4.16 show the effect of leakage resolved. The resulting protrusions and cavities indicate positions of leakage. In some of the objects the cavities are severe.

The implementation of the Pulse Reformation framework can be implemented with more complicated energy functions, noise functions and shape probes to increase accuracy. The experiment does supply valid information that the new proposed framework, Pulse Reformation, does carry merit and can be implemented to combat leakage.

4.3.2 Object Detection

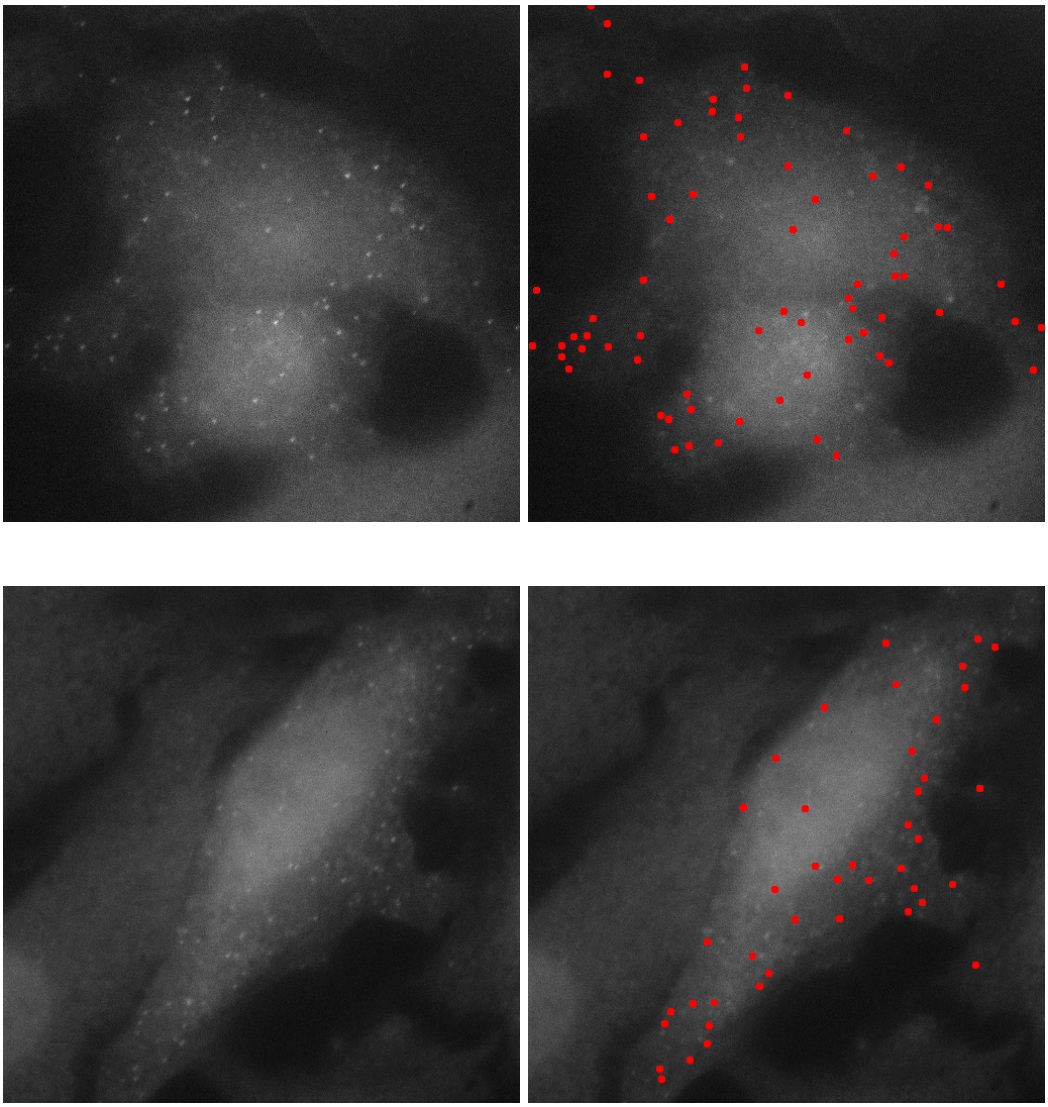
The Pulse Reformation framework can also be used for spot detection. We can not only extract the strongest joined stacks but only the joined stack who's bottom region or top region is within a certain range. For spot detection the bottom region of the joined stack must be smaller or equal to the largest expected spot.

Fixed cell imaging of individual mRNA molecules is accomplished by using 48 or more singly labelled oligonucleotide probes [40]. By utilizing fluorescent microscopy the mRNA becomes computationally identifiable fluorescent spots. There can be hundreds of mRNA in a cell. An effective spot detector and spot counter is thus required.

A current spot detector exists which uses thresholding of a difference of two Gaussians [40]. The Difference of Gaussian (DoG) is a method where the DoG image is created by taking the difference of two images where each image was convolved with a different Gaussian function. The DoG is known to be important in biological visual processing [41]. The DoG spot detector has three different parameters which can be modified: the range of the Gaussian window, the variance of the Gaussian and the specific threshold. In depth the method actually has 5 parameters as both the Gaussians must be selected, each with a window and a variance. Usually one Gaussian is assumed to be the original image.

Our spot detector will only use the joined stacks with a bottom pulse cardinality smaller than a set size. These joined stacks are then thresholded by the joined stack strength to remove the very weak joined stacks. The Pulse Reformation spot detector then only uses 2 parameters: the expected size of the objects and the threshold value.

For the experiment we created our own ground truth. The ground truth was not evaluated by an expert in mRNA thus we can not measure the algorithms true performance but only the relative performance. The DoG spot detector was confirmed to be an accurate detector but was not quantified [40]. The two test images and their ground truths are shown in Figure 4.18.



(a) Images of mRNA [40] **(b)** Ground truths: Detected mRNA

Figure 4.18: Fluorescent microscopy images of mRNA.

The two algorithms will be evaluated by using precision-recall graphs discussed in Section 5.5.2. The precision and recall is calculated with:

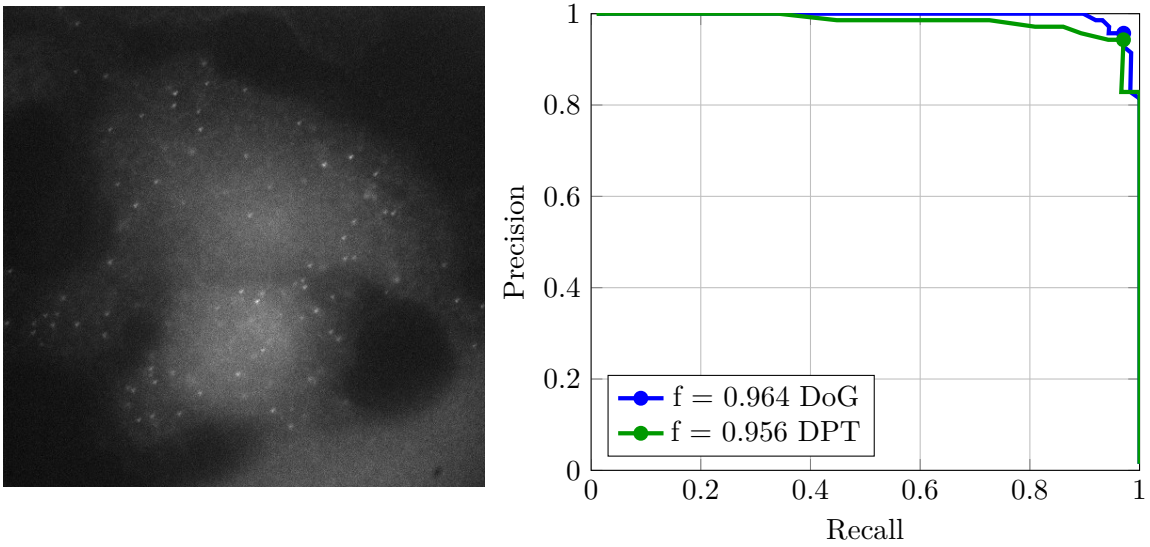
$$\text{Precision} = \frac{tp}{tp + fp} \tag{4.4}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{4.5}$$

where the f-measure is calculated with:

$$F\text{-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.6}$$

The true positives, tp, are measured by taking the number of correctly detected spots. Thus the detected spots that coincide with the ground truth. The false positives, fp, are measured by all the spots that do not coincide with ground truth. If two spots are detected close together and both coincide with one spot on the ground truth, one is taken as correct and the other is then a false positive. The false negatives, fn, are calculated as all the spots in the ground truth that were not detected by the spot detection algorithm.

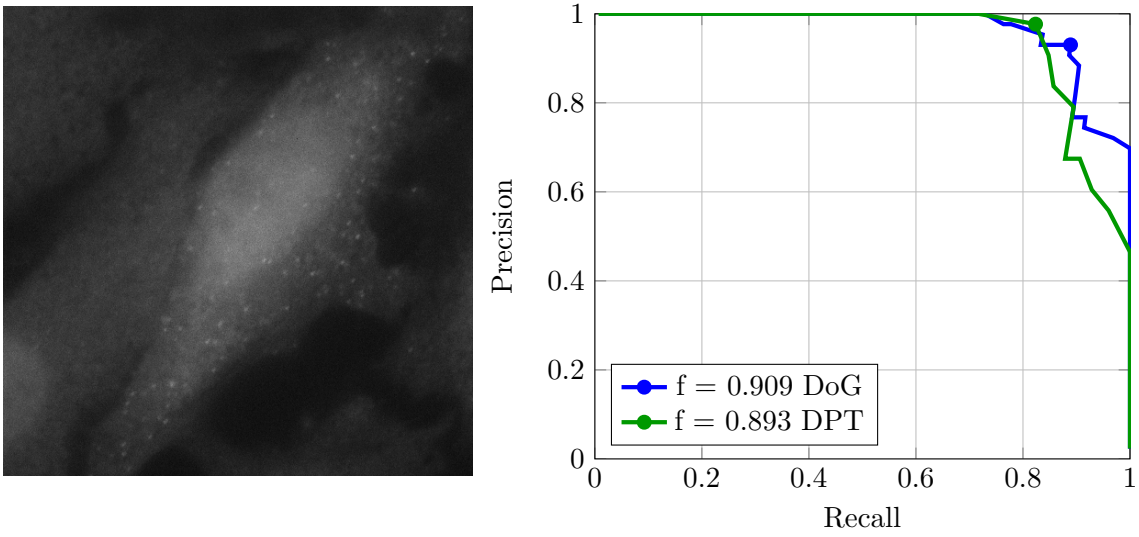


(a) Test Image **(b)** Test Results

Figure 4.19: The first Precision-Recall graph for the DPT and DoG spot detectors.

In Figure 4.19(b) and Figure 4.20(b) the f-measures are plotted for increasing threshold values. The maximum f-measures are indicated on the graphs.

Both the algorithms perform very well. The Pulse Reformation spot detector using the DPT is only slightly less accurate than the DoG method. Taking into account that the DPT method has 3 less parameters to tune than the DoG method and is less sensitive to its parameters, the DPT method is superior to the DoG method. The sensitivity of the DoG method to the chosen variance parameter is very high, as the chosen variance propagates to each pixel in the image. Choosing the correct variance for a Gaussian function is also not a trivial matter. The DPT method is not very sensitive to the expected size of the spot as long as the expected size is greater than the actual size of the spot. The expected size of the spot is easily measured by looking at the number of pixels within the spot.



(a) Test Image **(b)** Test Results

Figure 4.20: The second Precision-Recall graph for the DPT and DoG spot detectors.

4.4 CONCLUSION

We have provided an effective algorithm to deal with leakage in images and have applied the technique to salient object detection and a more specific application in spot detection. In the next chapter we investigate the potential of the LULU scale-space for image segmentation.

CHAPTER 5

IMAGE SEGMENTATION

The division of an image into meaningful regions, also known as image segmentation is an essential step in image analysis. Image segmentation is the independent partitioning of an image into disjoint regions. These regions are visually different, homogeneous and meaningful with respect to certain specific characteristics. These characteristics or properties enable image analysis and can be grey levels, colour, texture or any other specific properties.

The Discrete Pulse Transform is a mathematical tool. Before much time can be invested in developing the mathematical tool for a specific purpose the tool must be able to show potential in the field. This chapter is aimed at determining whether the DPT has any potential in image segmentation.

5.1 SEGMENTATION TECHNIQUES

Image segmentation can be accomplished in various ways. A basic property accompanying all image segmentation techniques can be defined [42].

Definition 32. *Let $\mathcal{L}$ be a lattice. An image $f(I) \subset \mathcal{L}$ with grid $I \subset \mathbb{Z}^2$ is successfully segmented if I is divided into n unique subsets $S_i \subset I$ for $i = 1, 2, ..., n$ such that $I = \bigcup_{i=1}^{n} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. A group of subsets with specific characteristics form a class.*

For an image f segmented as $\mathcal{P}_1(f) = \{S_{1i}\}_i$ or $\mathcal{P}_2(f) = \{S_{2j}\}_j$ for $i = 1, 2, ..., n$ and $j = 1, 2, ..., m$ where $\mathcal{P}_1$ and $\mathcal{P}_2$ are two different segmentation algorithms, segment S_{1i} is more coarse than segment S_{2j} if $S_{2j} \subset S_{1i}$. A coarse segment can also be referred to as a segment at larger scale thus a finer segment is defined as a segment on a smaller scale.

Within the literature we can divide image segmentation into six main categories namely; threshold based segmentations, histogram based segmentations, edge detection based segmentation, region based segmentation, watershed transformation segmentation and graph partitioning segmentation.

Threshold based segmentation is one of the fundamental image segmentation techniques. The techniques usually have two different classes, namely the foreground and the background. A certain grey level is chosen where all pixels above the specific level are classified as foreground and all the others are classified as background. More complex threshold segmentation techniques can be created such as adaptive thresholding and mutli-level thresholding to obtain more classes [43].

Histogram-based techniques are focused on estimating different probability distributions within the histograms formed from color channels or grey scale levels. These estimated distributions are used to assign classes and segment the image [44].

A common approach to segmentation is using edge detection algorithms to create separate regions. The first and second order derivatives of an image are usually used for edge detection, which is followed with thresholding and algorithms that link broken edges. Edge detection algorithms such as the Canny edge detector [45] and Sobel edge detector [46] have had wide application in segmentation algorithms [47].

Region growing is also known as pixel based image segmentation. It is highly dependent on the selection of initial seeding points which are used to iteratively decide whether the pixel neighbours of a seed point should form part of the class [48]. This type of segmentation is good for interactive image segmentation [49]. Region growing also includes techniques which arbitrarily split an image into small regions and then recombine and split the regions based on specific criteria [50].

The watershed transformation is a well established image segmentation method which uses the concept of a topographic relief where basins are filled with water and each basin determines the different class. The typographic domain can be determined by intensity levels or grey levels [51]. A more advanced watershed concept is called active contour models. The active contour model uses an energy function defined on the typographic layout of the image. The energy function is minimized in accordance to the contour defined in the image [52].

The last category is graph partitioning segmentation. An image is modelled as a weighted undirected graph where each pixel is a node in a graph and the connectivity is presented by the edges between neighbouring pixels. The weights of the edges in the graph are calculated by using a similarity measure between pixels. A common method to partition the graph is to do a minimum cut of the graph [53]. A minimum graph cut is realized by removing the edges that give the lowest sum of weights to separate the graph into two new graphs. Each new graph can be cut an arbitrary number of times to create more refined segmentations.

5.2 QUANTITATIVE EVALUATION METHOD

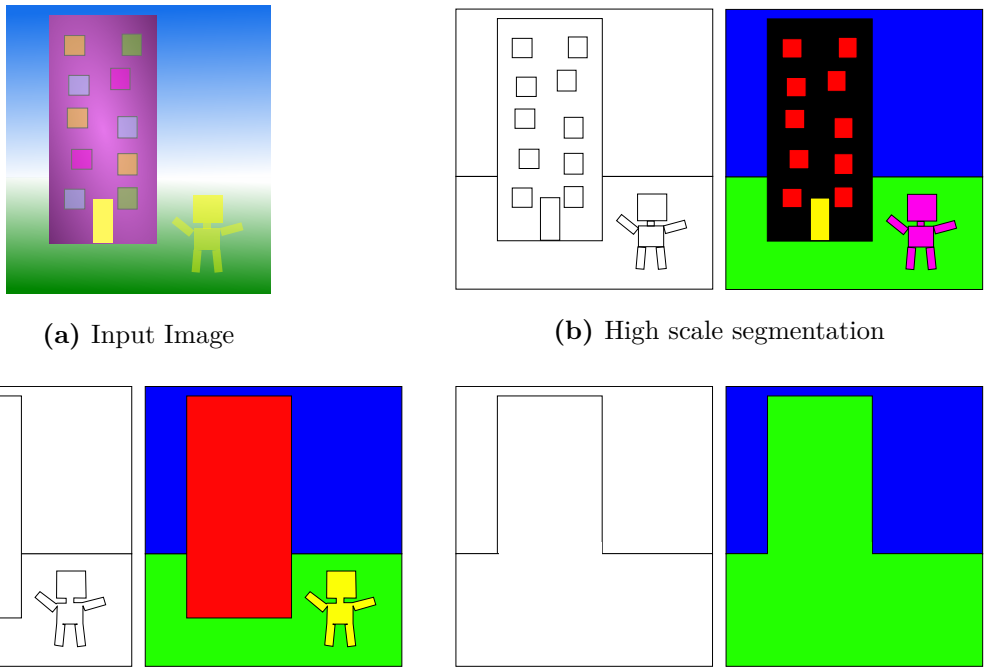
The evaluation of image processing tasks in a quantitative fashion are very important. It is necessary to be able to compare different concepts and algorithms and build on the most successful concepts. There are mainly two different philosophies in evaluating image processing tasks. One such philosophy is evaluating machine vision tasks in context of a particular task. The latter is in evaluating the algorithms with regards to a pre-defined ground truth with suitable metrics.

Evaluating an algorithm with regard to a specific task can be useful when tailoring a specific solution. Newly developed algorithms can be benchmarked on the task but the performance will be unknown for any other task. A philosophy for task specific evaluation is given by Borra and Sarkar [54]. They define performance measures for grouping objects based on object recognition performed in context of constrained search and indexing. Other more general performance measures for image segmentation focussing on task specific performance are the Probabilistic Rand Index (PRI) [55] and Variation Of Information (VOI) measure [56].

Image segmentation techniques can be evaluated by either detected contours or by clustered sets of pixels. Detected contours can be converted to pixel clusters by using pixel information enclosed by the contour from the original image and clustered pixels can be converted to contours. If all contours are detected correctly the corresponding pixels in the original image within each contour can be clustered to classify each region enclosed by a contour [57]. If contours extracted have different confidence intervals, clusters of different importance can be extracted and thus segmentations at different scales can be achieved.

Figure 5.1 shows a synthetic example of image segmentation using contour detection and clus-

tering at different segmentation scales. Figure 5.1b is a high scale segmentation and you can see every piece of information is segmented. Figure 5.1c is a medium scale segmentation showing you can see foreground, two different mid-grounds and a background. Figure 5.1d shows a low scale segmentation with only a background and a foreground. This example clearly shows the interchangeability of clustering and contour detection. After contour detection each set encircled can be classified to produce a clustered version of image segmentation.



(a) Input Image

(b) High scale segmentation

(c) Medium scale segmentation

(d) Low scale segmentation

Figure 5.1: Segmentation at different scales with contour detection and clustering.

Contour detection provides the core of image segmentation via preliminary image division. An evaluation method testing contour detection as a fundamental property of image segmentation would then help determine the potential of DPT in image segmentation. The use of precision-recall graphs with F-measures introduced by David et al. [58] can be used. The Berkley Segmentation Database (BSD) [26] is used for evaluating image segmentation algorithms utilizing the precision-recall graphs as introduced by David et al.

5.2.1 The Berkley Segmentation Database

The Berkley Segmentation Database [26] was originally developed to gather statistics on natural images and later to support the evaluation of image segmentation algorithms. The dataset consists of 300 color images with constant size of 481 x 321 pixels. The set is divided into 200 training images and 100 testing images on which the image segmentation algorithms performance can be measured. Some sample images can be seen in Figure 5.2.

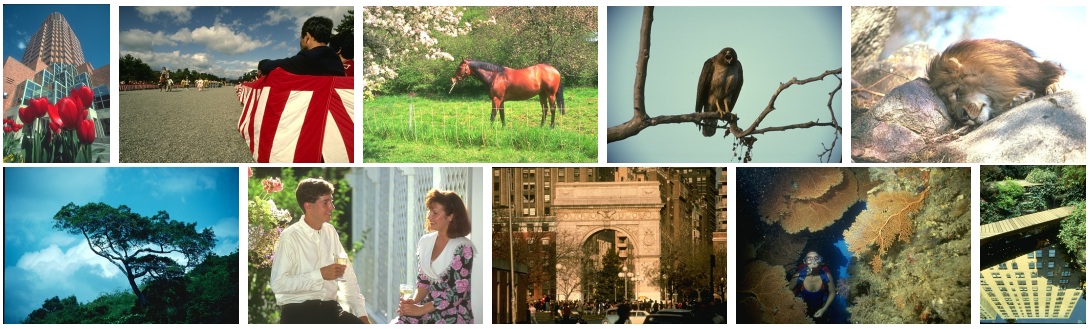
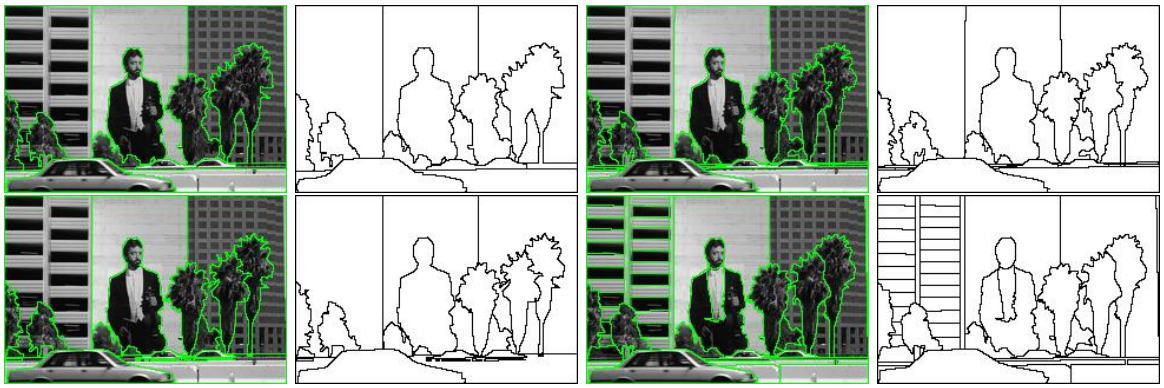


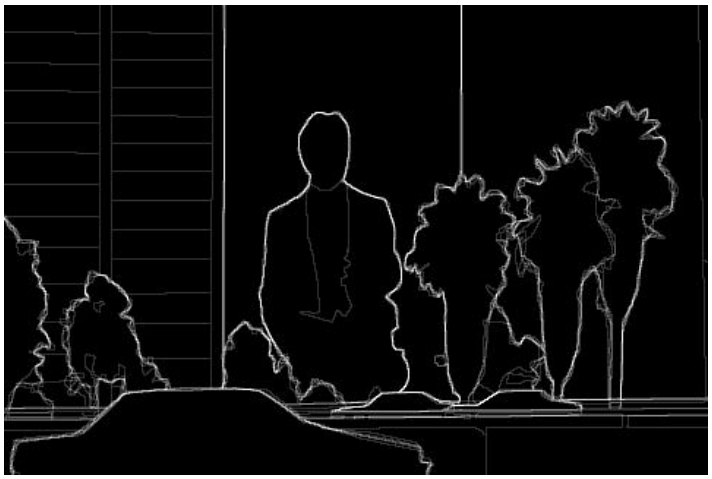
Figure 5.2: Berkley Segmentation Database sample images

The ground truth for image segmentation in this case is in the form of boundaries created from between 4 to 8 human segmentations. The human segmentations were acquired by a human candidate using a computer program which allows the user to trace boundaries. The human candidates were mostly undergraduates presented with random images from the database with no prior knowledge of the final outcome. The segmentations of the same image from different candidates were combined into a confidence map. A confidence map for a specific natural image is created by adding every human segmentation to a specific image and normalizing it to the number of images added. White will present the highest confidence and black will be the lowest confidence in the gray scale image. An example of a created ground truth with its segmentation is shown in Figure 5.3.

(a) Original image



(b) Human segmented images



(c) Confidence map

Figure 5.3: Creation of a ground truth image with human segmentations for the BSD

5.2.2 Evaluation Metrics

A way to evaluate a contour detection is to measure the mutual information in a specific segment S of the ground truth segmentation and a specific segment $\hat{S}$ of the algorithm segmentation [59].

To calculate the mutual information the ground truth and algorithm output can be modelled as a binary map such that $S_x \in \{0,1\}$ and $\hat{S}_y \in \{0,1\}$ where $S_x \in S$, $\hat{S}_y \in \hat{S}$. The joint probability distribution is given by $p(x,y) = P(S_x = x, \hat{S}_y = y)$ and the mutual information is given by

$$I(S,\hat{S}) = \sum_y \sum_x p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \tag{5.1}$$

The mutual information can also be calculated for a segmentation with a confidence map, that is $\hat{S}_y \in [0,1]$. The joint distribution $p(x,y)$ can be calculated by using the confidence interval to fill the bins. As an example, in the binary case, the number of elements which are 0 in S and 1 in $\hat{S}$ divided by the total number of elements in S will give $p(0,1)$. This can be repeated for $p(0,0)$, $p(1,0)$ and $p(1,1)$, by using joint distribution values $p(x)$ and $p(y)$ can be calculated. This measure is not very intuitive and it does not provide information about the quality of segmentation. It also does not account for small segmentation shifts within the image thus other measures are required.

The variation of image segmentation can be seen as an ill-posed problem. Image segmentation is ill-posed as there exists no unique solution and changing of initial conditions does not necessarily change the segmented image thus an image segmentation algorithm needs to be evaluated in different ways. In accordance with the BSD a strong regional based measure (GCE and LCE metric) and a strong statistical method (precision-recall graphs) will be used and is discussed below [26].

5.2.2.1 The GCE and LCE metric

Image segmentation as an ill-posed problem does seem to be solved consistently by human segmentation but with varying detail. The scale at which an image is observed differs for each human observer but the main boundaries for images stay the same. Martin et al. [26] use this consistency to propose two metrics to evaluate segmentations. The metrics are designed

to be tolerant to scale changes and thus manage region refinement at different scales. Let $S_1 = \{S_{1i}\}$ be the ground truth segmentation and $S_2 = \{S_{2j}\}$ be the segmentation produced by the algorithm. To measure the consistency, a measure for each pixel p_k is required and can be defined as:

$$E(S_1, S_2, p_k) = \frac{|R(S_1, p_k) \backslash R(S_2, p_k)|}{|R(S_1, p_k)|} \tag{5.2}$$

where $R(S, p_k)$ is the segmentation set $S_m \in S$ which contains the pixel p_k. The equation represents operations on sets thus $\backslash$ is set differences and $|\cdot|$ is the cardinality of the set within. Evaluating the measure one can see that when all the pixels in S_1 are contained in S_2 the measure outputs 0. The measure is not symmetrical thus for each pixel, both $E(S_1, S_2, p_k)$ and $E(S_2, S_1, p_k)$ must be calculated.

To get useful information regarding the segments, a summation of the errors within the segments must be calculated. Assuming that either all segments S_{2j} in S_2 are courser refinements of all segments in S_1 or vice versa the Global Consistency Error (GCE) can be defined as

$$GCE(S_1, S_2) = \frac{1}{n} \min \left\{ \sum_k E(S_1, S_2, p_k), \sum_k E(S_2, S_1, p_k) \right\} \tag{5.3}$$

where n is the total number of pixels. It's possible that refinements can occur in both directions thus another measure called the Local Consistency Error (LCE) is given as

$$LCE(S_1, S_2) = \frac{1}{n} \sum_k \min \{E(S_1, S_2, p_k), E(S_2, S_1, p_k)\}. \tag{5.4}$$

The GCE looks at the minimum consistency measured over the whole segment where the LCE does not make the same assumption as the GCE and chooses the minimum consistency per pixel thus calculate the minimum possible consistency of the segmentation. These two measures can be shown to be low for human segmentations of the same image and high between random segmentations of the same image. It is also evident that $LCE \leq GCE$.

These two measures can be further investigated by testing them at two extremes, over segmentation and no segmentation. Over segmentation is obtained when each pixel in an image receives its own label. This will result in LCE and GCE being equal to zero. No segmentation can be evaluated when all pixels in the image have the same label which will also result in LCE and GCE equal to zero. A measure which permit refinement to such a degree will not suffice for a benchmark.

The LCE metric can be made more strict by penalizing dissimilar segmentations with a

region overlap. This improved LCE is called the Bidirectional Consistency Error (BCE) and is defined as

$$BCE(S_1, S_2) = \frac{1}{n} \sum_k \max \{E(S_1, S_2, p_k), E(S_2, S_1, p_k)\}. \tag{5.5}$$

The BCE can now be seen as taking the maximum consistency of each pixel and will penalize a large amount of dissimilarity. The segmentation of an image is subjective to the viewer thus if multiple ground truths $\{S_1^{(b)}\}$ exist for an image, each created segment S_{2j} must be evaluated against each ground truth segment, thus the BCE can be extended to

$$\widetilde{BCE}(S_2) = \frac{1}{n} \sum_k \min_{\{S_1^{(b)}\}} \left\{ \max \left\{ E(S_2, S_1^{(b)}, p_k), E(S_1^{(b)}, S_2, p_k) \right\} \right\}. \tag{5.6}$$

The equation above shows that the sum of any segment in all the ground truths producing the minimum consistency is taken while still compensating for refinements. The $\widetilde{BCE}$ measure captures the consistency of the complete image segmentation. The measure can thus be used as a benchmark metric.

5.2.2.2 Precision and Recall Curves

Precision-Recall graphs and Receiver Operating Characteristic (ROC) graphs are standard evaluation methods in information retrieval. The Precision-Recall Graphs have two axes called precision and recall where the ROC graphs have two axes called fallout and recall [60].

Precision, the positive prediction value, is the percentage of relevant detected units which are correct, thus the ratio of correctly detected relevant units to the total number of relevant detected units. Recall, the true positive rate, is the percentage of how many of the relevant units that exist have been detected, thus the ratio of correctly detected relevant units to the true existing number of relevant units. Fallout, the false positive rate, is the percentage of non-relevant units which are correct, thus the ratio of correctly detected non-relevant units to the total number of detected units. Relevant pixels are the pixels which the algorithm is aiming to detect.

$$\text{precision} = \frac{\text{correctly detected relevant units}}{\text{total number of retrieved units}} \tag{5.7}$$

$$\text{recall} = \frac{\text{correctly detected relevant units}}{\text{total number of existing relevant units}} \tag{5.8}$$

These three performance measures can be evaluated in terms of boundary evaluation. Precision gives the percentage of boundary pixels in the segmentation that correspond to the boundary pixels in the ground truth. Recall gives the percentage of boundary pixels in the ground truth that were successfully detected. Fallout gives the percentage of pixels of non-boundary pixels in the segmentation that correspond to the non-boundary pixels in the ground truth.

As fallout uses all pixels in the image it is not suitable for segmentation evaluation as it's heavily dependant on the image size. The dependency of fallout on the image size can be explained by using boundary line thickness. If we increase the image size, the boundary will remain a thickness of 1 pixel but detection algorithms keep detecting a proportionate thick line. Increasing the image width proportional by n pixels, the total number of pixels in the image increases by n^2 but as the boundary has a thickness of 1 pixel the number of true positives will grow by a factor of n where the number of true negatives grow by n^2. By using this argument fallout will decline with the rate $1/n$. Precision on the other hand does not have this problem as it is normalized. ROC graphs can thus not be used for boundary evaluation and image segmentation as fallout is not suitable.

To get a better understanding of the precision-recall metrics we can use a binary classifier. A classification model using only two classes, thus a binary classifier, can assign two labels, positive or negative. There are four possible outcomes. If the classifier labelled a positive sample as positive it is called a true positive (tp). If a classifier labelled a negative sample as positive it is called a false positive (fp). If a classifier labelled a positive sample as negative it is called a false negative (fn). If a classifier labelled a negative sample as negative it is called a true negative (tn). Using these labelling methods we can readily define recall and precision [61]

$$\text{Precision} = \frac{tp}{tp + fp}, \tag{5.9}$$

$$\text{Recall} = \frac{tp}{tp + fn}. \tag{5.10}$$

It is difficult to evaluate two measures simultaneously thus we can use the harmonic mean of precision and recall. It is traditionally known as the balanced f-measure or the f_1 meas-

ure [61]

$$f\text{-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5.11}$$

To calculate precision and recall graphs for a segmented image we need to be able to match the algorithm generated segmentation and the ground truth. To achieve this we need to match each edge pixel to an edge pixel in the ground truth image. Each boundary pixel p_k in the segmented image has a position (x_k, y_k) within the image plane. Each pixel also has an orientation θ_k measured with regards to the positive x-axis. The orientation θ_k is given by the direction of the normal vector to the boundary line at the position of pixel p_k. A pixel p_k can then be described by (x_k, y_k, θ_k).

To match the boundary pixels a bipartite graph can be set up by creating two disjoint sets, the pixels of the ground truth S_1 and the pixels of the segmented image S_2. Each pixel p_{k_1} is then connected with an edge with weight $w_{p_{k_1} \to p_{k_2}}$ to every pixel p_{k_2}. The weight function provides a cost function to match two pixels, by measuring the similarity of the two pixels in the segmentation and ground truth respectively,

$$w_{p_{k_1} \to p_{k_2}} = \sqrt{(x_{k_1} - x_{k_2})^2 + (y_{k_1} - y_{k_2})^2} + \alpha \frac{|\theta_{k_1} - \theta_{k_2}|}{\pi/2}. \tag{5.12}$$

The cost function uses the euclidean distance and a scaled orientation function. The orientation function is added to aid in the matching of edges. The scaling parameter α is set to $\alpha = 1/d_{max}$ where d_{max} is 0.01 times the number of pixels in the diagonal of the image [58]. This weight function allows the construction of a matched bipartite graph between all pixels in the two segmentations being compared. The precision and recall graphs for boundary problems are then finding the minimum cost matching between two boundary segments.

One practical difficulty for this cost function is that the cardinality of the two sets in the bipartite graph will differ. This can be solved by padding the set with the lowest cardinality with outlier nodes thus creating high valued weights. Whenever a node is matched to an outlier node the match is assumed to be unmatched as the outlier node is only virtual.

Solving the bipartite graph becomes a minimum cost maximum flow problem. This problem can be readily solved by Andrew Goldberg's Cost Scaling Algorithm (CSA) package [62]. One threshold is added to the bipartite graph: if a node is matched to another node with euclidean distance greater than d_{max} the node is rematched to an outlier node and called unmatched.

The precision and recall can be calculated from the matched bipartite graph where S_2 is the segmented image and S_1 is the ground truth. The error is given by the nodes matched to outlier nodes, which are also called unmatched nodes. We will get two error measurements, one from S_2 matched to outliers and another from S_1 matched to outliers. Precision is the fraction that is equivalent to the ground truth, thus the percentage matched edges in S_2 to S_1. Recall is the percentage of ground truth within the segmented image, thus the percentage of edges matched in S_1 to S_2.

$$Precision = \frac{\texttt{card}\{matched(\hat{S})\}}{\texttt{card}\{\hat{S}\}} \tag{5.13}$$

$$Recall = \frac{\texttt{card}\{matched(S)\}}{\texttt{card}\{S\}} \tag{5.14}$$

One important factor of the precision-recall graphs are that the measure is not refinement tolerant. Two images can be exact refinements of one another but can have a f-measure of zero. It is important to have the exact same image sizes when executing a f-measure.

The precision-recall graphs can be interpreted by using the precision, recall, f-measure and the shape of the graph itself. A high precision and high recall will yield a high f-measure which is then a good apparent segmentation.

A precision-recall graph can be constructed by creating segmentations at different refinements with the same algorithm. Each refinement level will provide one point on the graph with a recall value, precision value and an f-measure. At a coarse refinement one should have high precision and low recall whereas at a fine refinement the precision should be low and the recall high. This will indicate a correctly operating segmentation algorithm. Precision quantifies the amount of noise in the output detector. A low precision then indicates over segmentation. Recall quantifies the amount of ground truth detected in the output thus a very low recall score indicates under segmentation. The f-measure can be computed for every point on the graph but the point on the graph providing the highest f-measure would provide the best trade-off between precision and recall and also show the best possible result for the algorithm.

5.3 SEGMENTATION USING THE DPT

The Discrete Pulse Transform forms the LULU-scale space and provides a large number of features per pixel within the image. Each pixel at location (x, y) can be described by a vector called the Discrete Pulse Vector [24],

$$\mathbf{p}_{(x,y)} = [h_1 \quad h_2 \quad h_3 \quad ... \quad h_N]^T \tag{5.15}$$

where h_n represents the height of the pulse of scale n which coincide with pixel (x, y). Note most of the h_n's will have a zero value as each scale is not present in every pixel. The human visual system is a large driving force for image segmentation algorithms and it has been shown that using clustering in a scale-space can simulate the human visual system [63]. Fabris-Rotelli [24] [9] has also shown with a qualitative approach that clustering in the LULU scale-space can be used for segmentation, which will be the basis from which we will build our basic image segmentation method. Therein using scale-space life time, that is, the number of non-zero h_n's, the longest living pixels are those more salient. The Gaussian scale-space also makes use of this concept [64]. For simplicity we will reduce the number of features by combining different ranges of features. It has been shown that image content can be well summarised with [24] [9]

$$f_b(x, y) = \sum_{n=b_1}^{b_2} |h_n|, b = 1, ..., d \tag{5.16}$$

where $f_b(x, y)$ is the feature range of pixel (x, y) starting at pulse scale b_1 and up to pulse scale b_2. The d-dimensional feature vector for pixel (x, y) is created by choosing d different feature ranges and is given by

$$\mathbf{f}_{x,y} = [f_1(x, y) \quad f_2(x, y) \quad ... \quad f_d(x, y)]. \tag{5.17}$$

We now require a clustering algorithm. Clustering algorithms usually need to be determined experimentally to find the most suitable algorithm, as no algorithm works well for all kinds of data [65]. The k-means clustering algorithm in conjunction with an Iterated Conditional Mode (ICM) algorithm has been shown to work well in image segmentation [66].

The k-means algorithm has an interesting property. It segments a data space into a Voronoi diagram [67]. A Voronoi diagram is a way to divide a space into a number of regions with pre-specified region points or seed points. The k-means algorithm acts by shifting these seed points to provide optimal regions. The regions created within the Voronoi diagram are

related to the distance metric used within the space. Although we can segment the features into regions using the k-means algorithm, we have not yet made use of the spatial information of the image.

To use the spatial information we require an initial labelling of the image, with the label pertaining to the region in which the k-means have divided it. We will first discuss the clustering algorithm using the k-means and ICM followed by the creation of a confidence map.

5.3.1 The Clustering Algorithm

Given an image with N pixels yields a set of N vectors in a d-dimensional space $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ where $\mathbf{x}_i$ is a d-tuple, each vector $\mathbf{x}_i$ relates to one pixel. The k-means algorithm minimizes the within-cluster sum-of-squares using k clusters $\mathbf{S} = \{S_1, S_2, ..., S_k\}$ where each cluster S_i has a mean centroid $\mathbf{c}_i$. The set of clusters that minimizes the within-cluster sum-of-squares $\mathbf{S}$ is obtained by the convergence of the algorithm and provide centroids for each cluster, that is,

$$arg \min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{(x,y)\in S_i} \|\mathbf{f}_{x,y} - \mathbf{c}_i\|^2. \tag{5.18}$$

The centroids obtained by the k-means algorithm are used as initial conditions for the Iterated Conditional Modes algorithm. Let the clusters for the initialization step $\alpha = 0$ be $\mathbf{c}_i^{(0)}$ for $i = 1, 2, ..., k$. Each pixel is assigned to a cluster using the ICM cluster criteria, the centroids are then recalculated and the pixels reassigned to the new clusters. The process is repeated until there is little or no change in the position of the centroids, thus the algorithm converges. The following two steps must be repeated until the algorithm converges:

1. For N number of pixels assign pixel (x, y) to cluster k for which the following equation is minimum:

$$(\mathbf{f}_{x,y} - \mathbf{c}_k^{(\alpha)})^T(\mathbf{f}_{x,y} - \mathbf{c}_k^{(\alpha)}) - \beta v^{(\alpha)} N_{x,y}^{(\alpha)}(k) \tag{5.19}$$

   where

   - β is spatial penalisation suggested as 1.5 [68],
   - $v^{(\alpha)} = \frac{1}{N}\sum_{k=1}^{N}\sum_{(x,y)\in S_k^{(\alpha)}}(\mathbf{f}_{x,y} - \mathbf{c}_k^{(\alpha)})^T(\mathbf{f}_{x,y} - \mathbf{c}_k^{(\alpha)})$ is the within cluster variance

- and $N_{x,y}^{(\alpha)}(k)$ is the number of neighbours of pixel (x, y) currently classified in cluster k at iteration α.

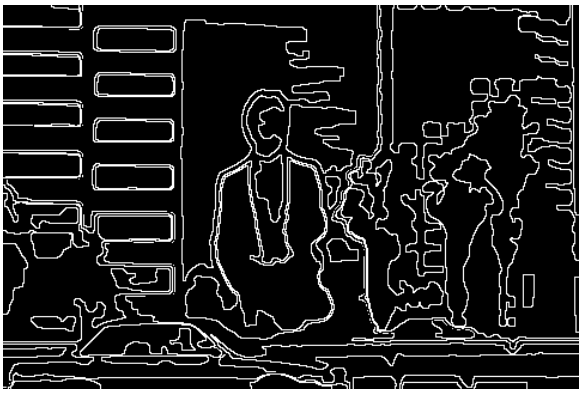
2. Recalculate centroids with

$$\mathbf{c}_k^{(\alpha)} = \frac{1}{N_k^{(\alpha)}} \sum_{(x,y)\in S_k^{(\alpha)}} \mathbf{f}_{x,y}, \tag{5.20}$$

where $N_k^{(\alpha)}$ is the number of elements within the class $\mathbf{c}_k^{(\alpha)}$. After the convergence of the ICM algorithm, all pixels are assigned to a specific segment which coincide with the centroid. A contour image can be created by using the boundaries of the different segments to which the pixels were assigned to. An example of the ICM segmented image is shown in Figure 5.4b. The final ICM segmented image is a binary image containing only the boundaries of the segments and is shown in Figure 5.4c.

(a) Original image

(b) ICM segmented image with 6 clusters



(c) Edges of ICM segmented image

Figure 5.4: ICM segmentation example

5.3.2 The Confidence Map

Using the ICM segmentation method directly creates a problem as the segmentation is heavily dependant on two parameter sets namely the number of clusters and the number of features with their related pulse ranges. Choosing these parameters incorrectly will segment the image into senseless segments as can be seen in Figure 5.5 where 3 clusters and 5 features with ranges $0-50$, $51-500$, $501-5000$, $5001-50000$ and $50001-100000$ were chosen.

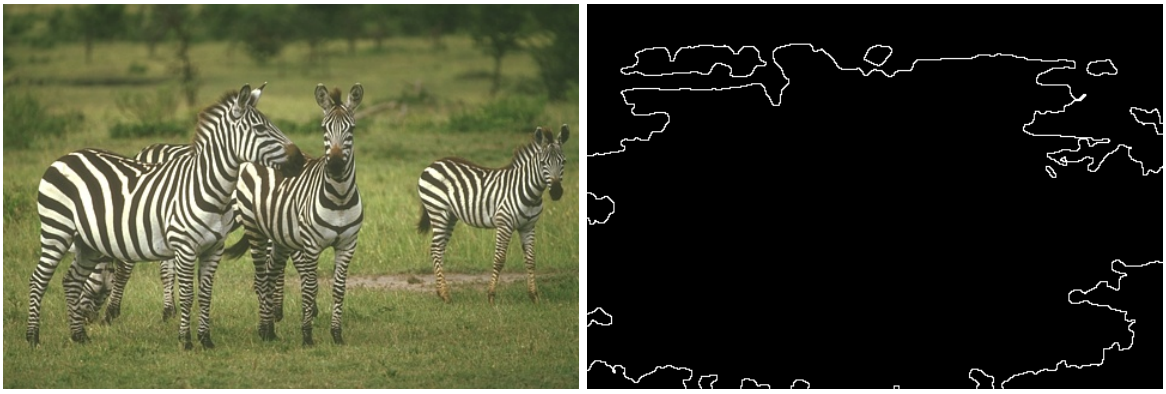
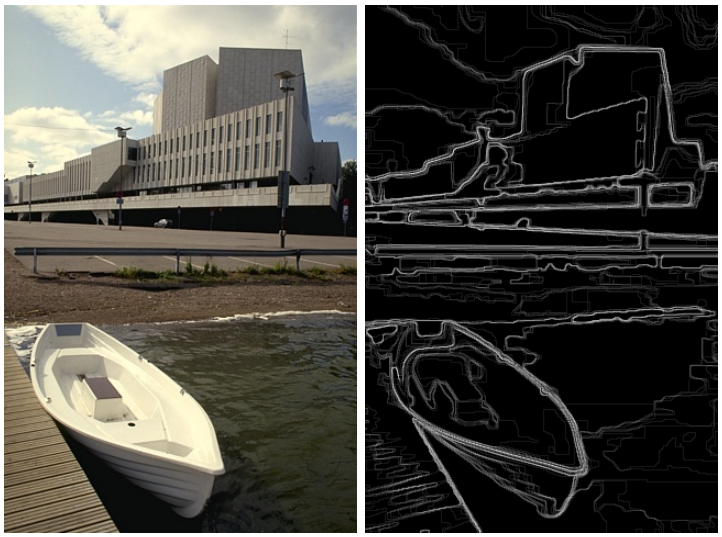


Figure 5.5: ICM segmentation resulting from badly chosen parameters.

To decrease the image segmentation sensitivity to these parameters a confidence map can be constructed using a range of discrete parameters within the algorithm. For every change in a parameter a segmentation image will be created. Each different parameter set will segment the image differently, with different refinements and focus on different textures. The main segmentation contours will appear in most of the images and will thus contribute high confidence to these contours. The confidence map adds another parameter to the algorithm. This parameter defines how the different images are combined to create the confidence map and is explained in detail later in this section. An example of segmentations and the combined confidence map are shown in Figure 5.6. The confidence map is created from the final segmented image with all parameters.



(a) Original **(b)** Confidence map

Figure 5.6: Created confidence map example

The three parameters sets, namely the number of clusters in the k-means algorithm, the features used to create the feature space and the creation of the confidence map needs to be addressed and logically evaluated. The number of clusters will clearly define the refinement of the segmentation, thus more clusters in the k-means algorithm will provide a more refined segmentation.

The feature space is defined by the number of features chosen and how the features are extracted. As defined earlier we will linearly combine a range of pulse sizes by heights. We can choose a number of features d and calculate the pulse ranges by using a formula. Each feature is given by

$$f_i(x,y) = \sum_{n=b_{i-1(x,y)}}^{b_i(x,y)} |h_n| \tag{5.21}$$

where $i = 1, 2, ..., d$, where $b_d(x,y) = N$. By example we can choose four features $d = 4$ and use a linear formula such as $b_i(x,y) = b_{i-1}(x,y) + 9$ where $b_0(x,y) = 0$ to calculate the value of each feature. We will call this formula the feature formula. The first feature is given by $f_i(x,y) = \sum_{n=0}^{9} |h_n|$, the second feature is given by $f_i(x,y) = \sum_{n=10}^{18} |h_n|$ and so forth thus the feature vector is given by,

$$\mathbf{f}_{x,y} = \left[\sum_{n=0}^{9} |h_n| \quad \sum_{n=10}^{18} |h_n| \quad \sum_{n=19}^{27} |h_n| \quad \sum_{n=28}^{N} |h_n| \right]. \tag{5.22}$$

To choose the pulse ranges, we need an idea of in what scale range most of the information lies, thus we can create a few histograms containing the number of pulses in each scale. These histograms can be seen in Figure 5.7.

By observing these histograms we can see that most of the pulses extracted with the DPT lies within the first 100 scales. Although this is dependent on the size of the image for most images the first one hundred scales contains most of the pulses. Whenever the feature formula is constructed it must contain most of these pulses and the larger scales must not be excluded either as they also contain information [69].

We now have two main parameters which can be varied to obtain a set of segmented images. This is the number of classes and the number of features in each feature vector. We can also change the feature formula but it is not directly a parameter and will also be dependant on the number of features selected and the scale capture range. If we have p discrete parameter values, such as p different clusters or p different values for d, p different segmentation sets $\mathbf{S}_j$ will be created where $j = 1, 2, ..., p$. From each segmentation set a binary contour image L_j is created containing the outlines of the segments in $\mathbf{S}_j$.

The confidence map can be created using a set of line images and combining them to create a grey scale image C which represents the confidence map. The first way to combine the binary images into the confidence map is by cascading them linearly thus,

$$C(x,y) = \frac{I_{max}}{p} \sum_{j=1}^{p} L_j(x,y) \quad \forall\ pixels\ (x,y), \tag{5.23}$$

where I_{max} is the maximum grey level in the image, $C(x,y)$ the confidence map for pixel (x,y) and $L_j(x,y)$ the line image of segmentation $\mathbf{S}_j$ at pixel (x,y). This method shows that if a specific segment is always present in all the segmentations it will have the highest grey level thus I_{max}. If a pixel is never part of a line, that pixel will have a zero value in the confidence map.

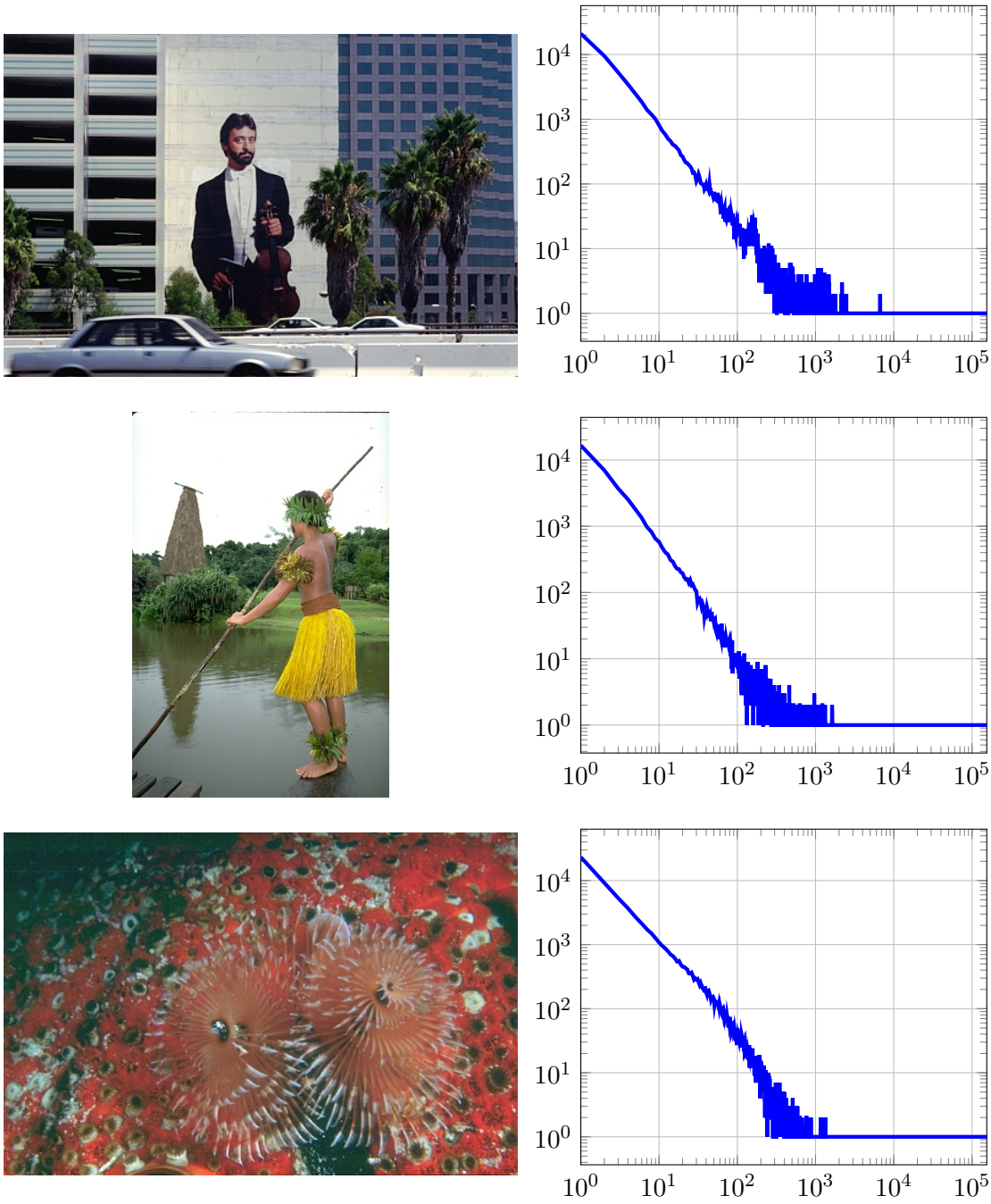


Figure 5.7: Histograms plotted on log-log plots with number of pulses (y-axis) against pulse scale (x-axis)

Another way to combine the various segmentations is by giving preference to one parameter, by example increasing the number of clusters result in more refined segmentations. Two clusters will provide the most significant segmentations followed by 3 clusters and so forth. The two kernel segmentation has priority over the other segmentations. Assuming that 2 clusters cohere with the first segmentation $j = 1$, the confidence map can be constructed as

$$C(x,y) = I_{max} \cdot \max_{j=1,\ldots,p} \left\{ \frac{p+1-j}{p} L_j(x,y) \right\}. \tag{5.24}$$

It is instinct to assume that choosing two clusters for the algorithm will provide a background and a foreground. This is not the case. The algorithm will create two distribution which best present all the pixels in the feature space. Choosing more clusters gives better distinction to dense feature space and also outliers. A high number of clusters will provide a highly segmented image but a low number of clusters will not specifically give a sparse segmentation.

5.3.3 The Image segmentation algorithm

The complete image segmentation algorithm using the DPT can be summarized as:

1. Start with iteration $t = 0$.

2. Obtain the Discrete Pulse Vector for each (x, y) by executing the DPT on the image,

$$\mathbf{p}_{(x,y)} = [h_1 \quad h_2 \quad h_3 \quad \ldots \quad h_N]^T. \tag{5.25}$$

3. Choose the number of features d.

4. Choose your feature formula to calculate the pulse size ranges, for example

$$b_i(x,y) = 5 * b_{i-1}(x,y) \quad \text{with } b_0 = 0, b_1 = 5, b_d = N \text{ and } i = 1, 2, \ldots, d. \tag{5.26}$$

5. Create a feature vector $\mathbf{f}_{xy}$ for each pixel (x, y) such that

$$\mathbf{f}_{xy} = [f_1(x,y) \quad f_2(x,y) \quad \ldots \quad f_d(x,y)]^T, \tag{5.27}$$

and where

$$f_i(x,y) = \sum_{n=b_{i-1}(x,y)}^{b_i(x,y)} |h_n| \quad \text{for } i = 1, 2, \ldots, d. \tag{5.28}$$

6. Choose the number of clusters k.

7. Execute the k-means algorithm to obtain k initial centroids $\mathbf{c}_j^{(0)}$ for $j = 1, 2, ..., k$.

8. Execute the ICM algorithm with centroids $\mathbf{c}_j^{(0)}$ for $j = 1, 2, ..., k$ as initialization to obtain segmentation.

9. Create a contour image L_t by using the segments obtained from the ICM algorithm, where t is the current number of iterations performed for the algorithm.

10. Increase the number of iterations t and repeat steps 3 to 10 p times while changing the variable parameters as required. The variable parameters are either the number of features d or the number of clusters k.

11. Create the final segmented image C, also called the confidence map by

$$C(x, y) = \frac{I_{max}}{p} \sum_{j=1}^{p} L_j(x, y), \tag{5.29}$$

or

$$C(x, y) = I_{max} \cdot \max_{j=1,...,p} \left\{ \frac{p + 1 - j}{p} L_j(x, y) \right\}. \tag{5.30}$$

5.4 IMAGE SEGMENTATION EVALUATION

5.4.1 Preparation

The image segmentation algorithm using the DPT needs to be evaluated to determine its potential. This can only be evaluated by comparing it to another set of image segmentation algorithms. An ideal comparison will be image segmentations that use scale-space and edges to segment the image. It is preferable that the image segmentation algorithm should not use any kind of training.

The Gaussian scale-space can be used for image segmentation and fits the criteria discussed above. It is a well known practice to take the difference of Gaussian to detect edges in an image. Lindeberg has shown that image segmentation can be done using the Gaussian scale-space [70]. A basic implementation of such image segmentation is the Canny edge detector [45].

There exist various other more complex usages of the Gaussian scale-space for image segmentation but we want to compare the basic concept of the Gaussian scale-space to the LULU scale-space. As both these methods can be classified as edge detectors we will also look at the direct usage of the Laplacian operator (second derivative) and Sobel operator (first derivative) [71] with a threshold to see whether more advanced edge detectors do have any benefit.

The Canny segmentation is summarized as:

- Apply a small Gaussian filter to reduce reduce noise in the image.
- Take the partial derivative, by utilizing the Sobel operator, of the image in the x and y direction with which a gradient vector at each pixel can be calculated.
- Apply non-maxima suppression for 8 directions.
- Apply hysteresis thresholding with edge tracing.
- Repeat the process for various thresholds to create a confidence map by adding them linearly together.

The Sobel segmentation is achieved by taking the first order partial derivative, by utilizing the Sobel operator, in the x and y direction with which the amplitude of the gradient vector is calculated for each pixel. These values are used directly in the confidence map. The confidence map for the Laplace segmentation is created in the same manner but utilizes the second derivative, the Laplace operator.

The precision-recall graph shown in Figure 5.8 contains the evaluation on the BSD of the Canny segmentation, Laplace segmentation and the Sobel segmentation. A fourth function containing the human segmentation for the test images from the BSD are also shown on the figure. Each function is provided with its best f-measure donated with a dot on the graph.

After an image is segmented by an algorithm a confidence map is calculated for where edges are most probable. To translate the confidence map into regions, a threshold must be selected to acquire a binary image with the regions then bounded by lines. For an ideal image

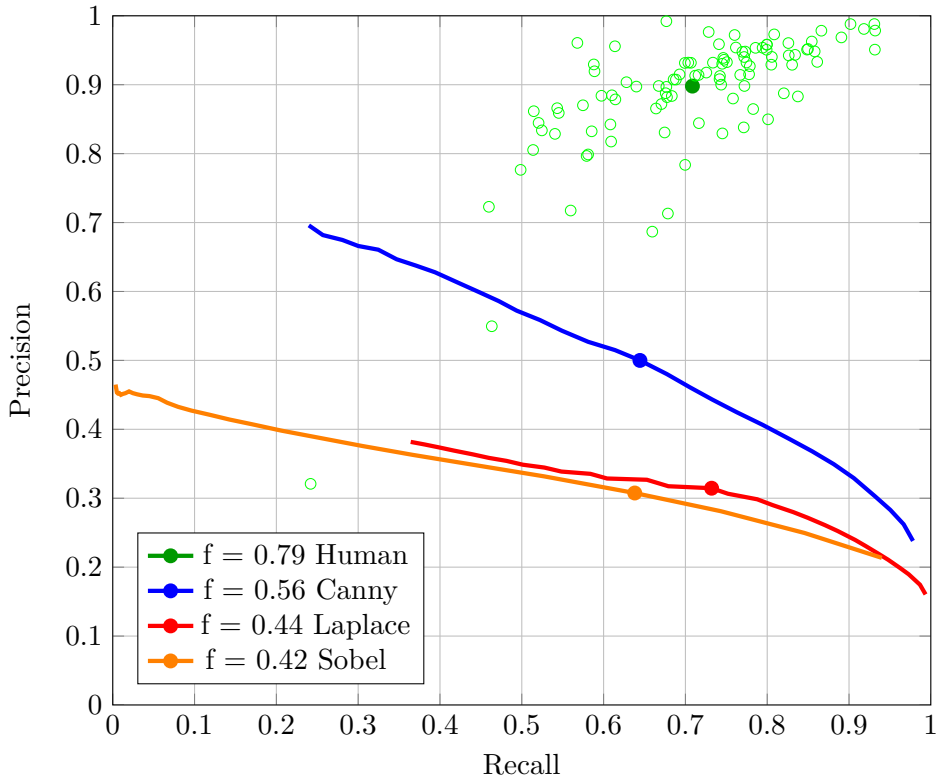


Figure 5.8: Precision-Recall graph for Canny edge detector, Laplace segmentation and Sobel segmentation

segmentation the threshold can be interpreted as selecting the amount of refinement for the regions. A large threshold should provide the boundaries for only the large objects, such as the foreground, middle-ground and background. Lowering the threshold increases the amount of detail thus including more specific objects. The continuous graphs in Figure 5.8 are generated by starting at the lowest threshold ending at the highest possible threshold which will then have no edges. The graphs do not indicate the thresholds used as it has no relevance in comparing the algorithms.

The human segmentations have been obtained in the same fashion, creating a confidence map from a set of images which are thresholded. The confidence map for the human segmentations were created by adding the different segmentations linearly together. Each human segmentation was tested on the created ground truth. As each human only segments the image with hundred percent certainty, each segment creates only a point on the precision-recall graph. The average f-measure of all human segmentations gives a 0.79 score. In Figure 5.8 it is clear that the Canny edge detector achieves the closest results to the human segmentations, thus

we will use the Canny detector to benchmark our own segmentation algorithm.

5.4.2 Experimentation

The three parameter sets still have unknown effects on how they influence the image segmentation. Firstly the number of features will be investigated. We require a simplistic feature formula and will use $b_i(x,y) = b_{i-1}(x,y) + range$ where $b_0(x,y) = 0$. After a few test runs using only two clusters, it was determined that creating pulse size ranges of 5 works in general the best, thus $range = 5$. This value might differ for using a different number of clusters. A more non-linear feature formula was also tested $b_i(x,y) = 2 * b_{i-1}(x,y)$ with $b_0(x,y) = 0$, $b_1(x,y) = 5$, $b_2(x,y) = 25$, $b_d(x,y) = N$ and $i = 3,4,...,d$. The number of features d were varied from 2 to 20 and both feature formulas are shown in Figure 5.9 using 5 clusters each.

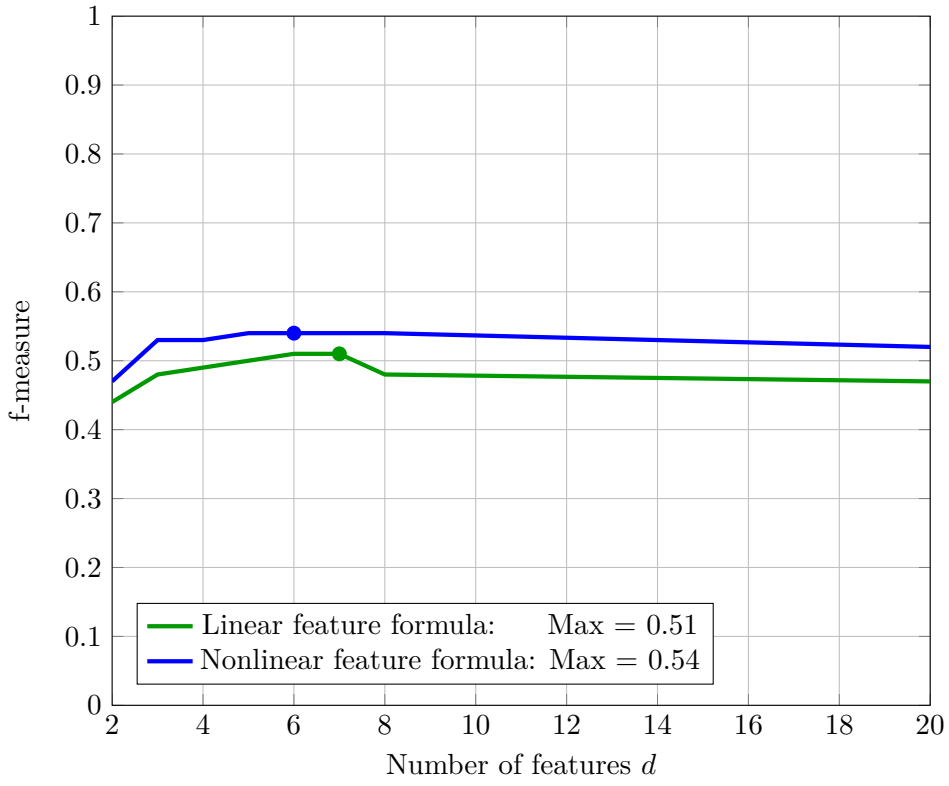


Figure 5.9: Image segmentation with variable features showing best f-measure against number of features used.

We can now perform the same experiment by varying the number of clusters and keeping

the number of features constant. The number of clusters was varied between 2 and 20. The confidence map was created by providing preference to the segmentation with the least number of clusters. The non-linear feature formula is used. The f-measure results of the variable clusters with constant features are shown in Figure 5.10.

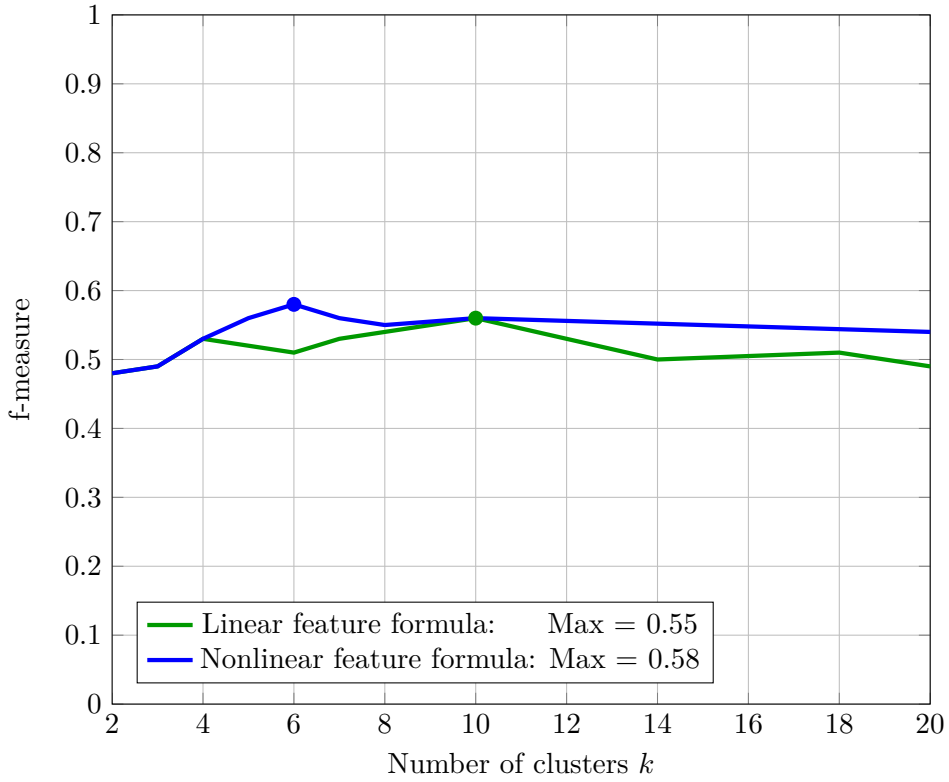


Figure 5.10: Image segmentation with variable clusters showing best f-measure against number of clusters used.

In Figure 5.10 we observe that using 6 clusters with the non-linear feature formula provide the best f-measure score. To compare the DPT ICM segmentation to the Canny detector we use the parameters giving the best results as determined by the experimentation. We use 6 clusters, 5 features and the non-linear feature formula. The overall performance is discussed in the next section.

5.4.3 Analysis

Analysing the precision-recall graph in Figure 5.11 we can see that both methods performs equally well. The DPT segmentation seems to oversegment the images fairly quickly which

is indicated by the movement of the graph towards a zero precision. While the DPT undersegments the image the precision is fairly high indicating that the segments that do get extracted are correct. It also has the maximum f-measure score between the two methods. The Canny segmentation does not have zero precision at an almost one recall, neither a high precision at low recall.

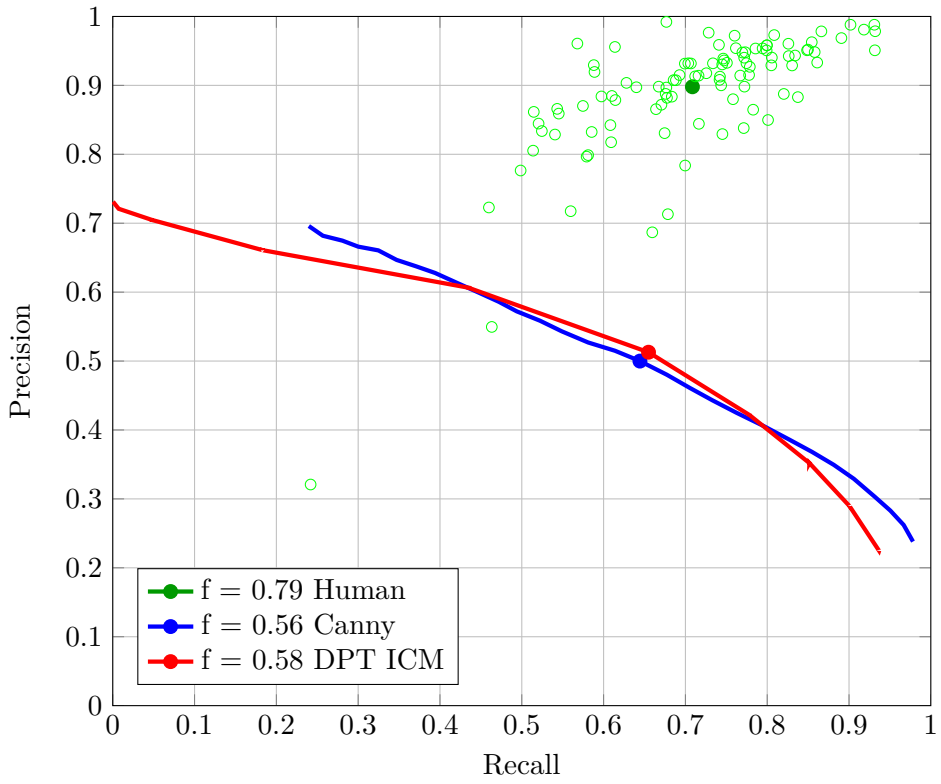


Figure 5.11: Precision-Recall graph for the Canny edge detector and DPT image segmentation

We can plot the average f-measure for each threshold value used to test the segmentations, which is shown in Figure 5.12. Looking at Figure 5.12 we can see that the Canny segmentation has a very good average f-measure and never drops below $f = 0.36$.

Figure 5.12 shows a step like function for the DPT algorithm. This step like function is due to the low number of clusters used which only introduce 20 different levels in the confidence map that can be thresholded. In contrast the Canny edge detector produces a confidence map with 255 different levels and thus plots more continuous.

The high peak of the DPT algorithm in Figure 5.12 gives you the ability to segment the

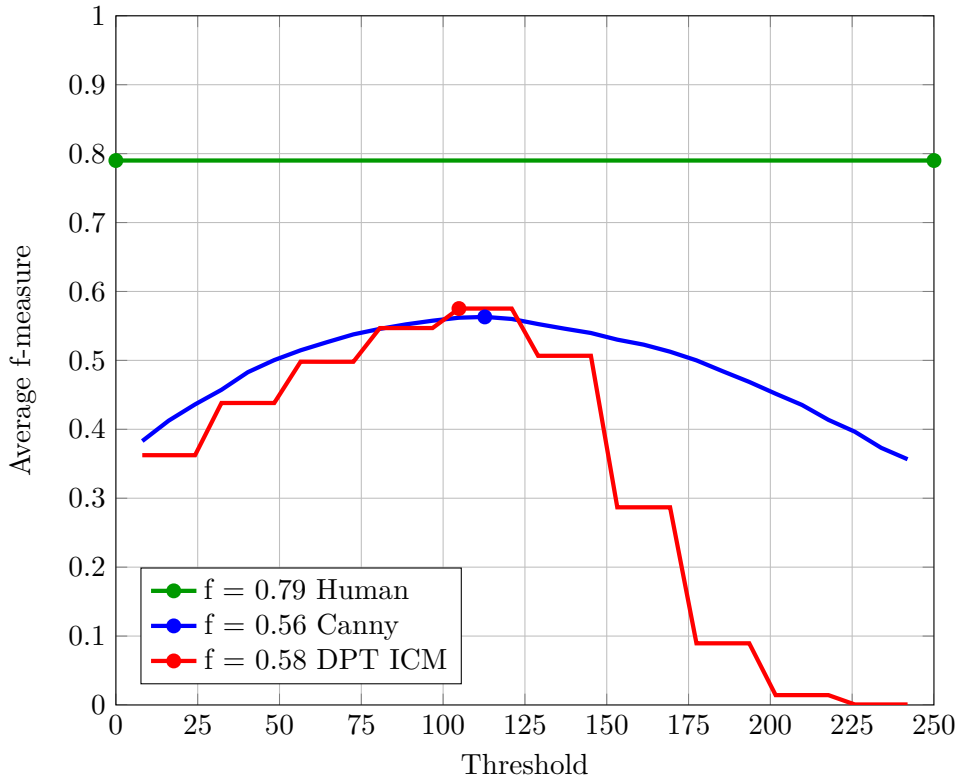


Figure 5.12: A plot showing the average f-measure for a specific threshold.

image with a specific purpose much easier. Selecting the incorrect threshold will immediately give you senseless information. Classification algorithms can then detect nothing at all but when selecting the correct threshold, all objects will be apparent. The Canny algorithm has a very good average which desensitizes the segmentation algorithm to the chosen threshold value. Although this sounds like a very good quality it creates difficulty in the sense that one might think the current threshold is the best as you are able to classify objects, where in reality it is not the best achievable and some objects might be missed.

The DPT ICM segmentation algorithm has a large amount of work left and a large number of studies can be done. A study on the different clustering algorithms which can be used and which are most suitable for the DPT features may reveal that the k-means and ICM method might not be perfect. A study on the influence of the pulse ranges chosen for features have on the performance of the segmentation algorithm and whether there exist a set of features that are invariant to the clustering algorithm can be investigated. A study on different methods on creating the confidence map and whether one method is invariant over the range of parameters should also be done. It should be possible by using the results of these three

studies to create a good stable DPT image segmentation algorithm.

The DPT image segmentation algorithm shows high potential for more advanced segmentation possibilities. One such possibility is to add different descriptors for the individual pulses such as eccentricity and convexity and add it to the feature vector used [72]. A neural network or machine learning method can also be employed to use all the features to segment the image. This image segmentation study opened up a large number of new research opportunities.

5.5 CONCLUSION

In this chapter we showed that the DPT can be used for image segmentation at a more elementary level. Therefore, a boundary detection algorithm was developed. The algorithm extracted features from the DPT for each pixel, clustering each pixel into a class using Iterated Conditional Modes initialized with a k-means algorithm. The boundaries of each class was traced in the image producing the segmentation. By varying the number of features, feature construction and number of clusters, different segmentations of the same image were created, which were used to create the confidence map. The confidence map consisted of grey scale boundaries presenting different confidences of each boundary.

We used precision-recall graphs as our quantitative measure in the subjective field of image segmentation. The precision-recall graphs were used to compare the Canny edge detector to the DPT boundary detection. The Canny edge detector was chosen as it uses a scale-space and form the basis for a large number of advanced image segmentation techniques. The precision-recall graphs showed that the DPT contour detector performed better than the Canny edge detector.

The chapter introduced new possibilities for researchers to pursue ideas such as using machine learning techniques for contour detection and clustering for image segmentation, as we showed the DPT has potential for image segmentation. We also added image segmentation to the possible applications for the DPT and the LULU scale-space. In the next chapter the thesis will be concluded as a whole, summarising what has been done.

CHAPTER 6

CONCLUSION

In the dissertation an introduction to the Discrete Pulse Transform (DPT) was presented. The DPT was extended as a mathematical tool, by developing a library containing an efficient implementation of the DPT in n-dimensions. The Pulse Reformation framework was proposed as a solution to the leakage problem, which arises for connected operators, such as the LULU operators. The Pulse Reformation framework was used in bioengineering applications, such as isolation of red blood cells and spot detection, specifically counting mRNA. Furthermore, it was established that the DPT shows potential in image segmentation. The DPT ICM image segmentation algorithm was compared to other relevant image segmentation algorithms by utilizing precision-recall graphs.

The following new research outputs have been contributed to the research domain of LULU operators and the Discrete Pulse Transform:

- A new graph-based algorithm for the DPT, called the Roadmaker's Pavage, was developed. The Roadmaker's Pavage stores the extracted pulses in an easily accessible manner while providing an intuitive presentation of the DPT decomposition. It is also valid for implementation in any dimension.

- A library, called the *DPT Library*, is available online [22]. The implemented Roadmaker's Pavage is currently the fastest available code of the DPT decomposition for n-dimensions.

- A framework to combat leakage, called Pulse Reformation, was developed. It was shown that the Pulse Reformation framework outperforms other proposed methods for

connected operators, such as λ-connected components. The framework was illustrated in biomedical engineering for isolating and counting red blood cells.

- The Pulse Reformation framework was applied to counting mRNA in fluorescence microscopy images. It was shown that Pulse Reformation can be used for spot detection. The spot detection capability is comparable to other spot detectors and uses fewer tuning parameters.

- The potential of the DPT in image segmentation was shown by means of a quantified evaluation. For the first time, the DPT's performance was compared to other image segmentation algorithms and was shown to be proficient.

Some limitations exist in the research conducted within the dissertation. The Pulse Reformation framework was implemented only for circular probes and gives the probability of relative circular objects within the domain. However, the circular probe can easily be replaced with various other basic shapes to extract a set of probable shapes within the domain. This set can then be used in generic object detection algorithms. The red blood cell extraction examples don't show obvious limitations, but the algorithm is expected to fail when two cells are overlapping and when one cell is highly rotated in the z-plane touching another red blood cell.

The image segmentation algorithm, using the DPT, is highly limited in its direct application to object segmentation. The algorithm provides an image which consists of probability borders, when segmenting objects there are no uncertainty in the outlines of the segmented objects. The algorithm thus provides fuzziness in the outlines of objects where a more advanced algorithm must be used to determine the real borders of the objects. The generated probability map can thus be used as a pre-segmentation for more complex algorithms.

Future work for the DPT includes continual algorithm improvement, the continual development of the Pulse Reformation framework, image segmentation and application in data communication. The implementation of the DPT can be optimised and also developed for parallel processing. With the Roadmaker's Pavage, the DPT can be applied to video sequences where further applications can be developed. Different probes for the Pulse Reformation framework can be implemented, and testing can be done in other application areas. The theory and application of the Pulse Reformation framework can be extended to general

connected operators. Image segmentation with the DPT can be further investigated by using more advanced classifiers and different feature vectors. The DPT lends itself to applications in data communication as it is inherently discrete, where comparing the DPT with noise reduction methods such as higher order statistics will be a good start.

REFERENCES

[1] D. Laurie, "The Roadmaker's Algorithm for the Discrete Pulse Transform," *Image Processing, IEEE Transactions on*, vol. 20, no. 2, pp. 361 –371, Feb 2011. [Online]. Available: http://dip.sun.ac.za/~laurie/DPT/

[2] P. Salembier and A. Oliveras, "Practical extensions of connected operators," in *Mathematical Morphology and its applications to image and signal processing*. Springer, 1996, pp. 97–110.

[3] R. Anguelov and I. Fabris-Rotelli, "LULU operators and Discrete Pulse Transform for multidimensional arrays," *Image Processing, IEEE Transactions on*, vol. 19, no. 11, pp. 3012–3023, 2010.

[4] E. W. Uys, "Image compression using the one-dimensional discrete pulse transform," Ph.D. dissertation, Stellenbosch: University of Stellenbosch, 2011.

[5] R. Rahmat, A. S. Malik, and N. Kamel, "3-d content generation using optical passive reflective techniques," in *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*. IEEE, 2011, pp. 639–642.

[6] I. Fabris-Rotelli and G. Stoltz, "On the leakage problem with the Discrete Pulse Transform decomposition," in *Proceedings of the 23rd Annual Symposium of the Pattern Recognition Association of South Africa*, A. de Waal, Ed., 29-30 November 2012, pp. 179–186.

[7] C. Rohwer, *Nonlinear Smoothers and Multiresolution Analysis*. Birkhauser, 2005.

[8] M. Rohwer, C.H.; Wild, "LULU theory, idempotent stack filters and the mathematics of vision of Marr," *Advances in Imaging and Electron physics*, vol. 146, pp. 57–162, 2007.

[9] I. N. Fabris-Rotelli, "Discrete Pulse Transform of images and applications," Ph.D. dissertation, University of Pretoria, 2012.

[10] G. Grätzer, *General lattice theory*. Springer, 2003.

[11] G. Hellman, "Mathematical constructivism in spacetime," *The British Journal for the Philosophy of Science*, vol. 49, no. 3, pp. 425–450, 1998.

[12] C. Rohwer and M. Wild, "Natural alternatives for one dimensional median filtering," *Quaestiones Mathematicae*, vol. 25, no. 2, pp. 135–162, 2002.

[13] J. Serra, "Image Analysis and Mathematical Morphology. Vol I, and Image Analysis and Mathematical Morphology. Vol II: Theoretical Advances," 1982.

[14] C. L. Mallows, "Some theory of nonlinear smoothers," *The Annals of Statistics*, vol. 8, no. 4, pp. 695–715, 1980.

[15] C. Rohwer, "Idempotent one-sided approximation of median smoothers," *Journal of Approximation Theory*, vol. 58, no. 2, pp. 151–163, 1989.

[16] C. Rohwer, "Projections and separators," *Quaestiones Mathematicae*, vol. 22, no. 2, pp. 219–230, 1999.

[17] C. Rohwer, "Variation reduction and LULU-smoothing," *Quaestiones Mathematicae*, vol. 25, no. 2, pp. 163–176, 2002.

[18] C. Rohwer and M. Wild, "LULU theory, idempotent stack filters, and the mathematics of vision of marr," *Advances in Imaging and Electron Physics*, vol. 146, pp. 57–162, 2007.

[19] C. Rohwer and L. Toerien, "Locally monotone robust approximation of sequences," *Journal of Computational and Applied Mathematics*, vol. 36, no. 3, pp. 399–408, 1991.

[20] E. Malkowsky and C. Rohwer, "The LULU-semigroup for envelopes of functions," *Quaestiones Mathematicae*, vol. 27, no. 1, pp. 89–97, 2004.

[21] C. Rohwer and D. Laurie, "The Discrete Pulse Transform," *SIAM Journal on Mathematical Analysis*, vol. 38, no. 3, pp. 1012–1034, 2006.

[22] G. Stoltz, "The DPT Library," https://github.com/genetica/DPT_Library, 2013.

[23] D. P. Laurie and C. H. Rohwer, "Fast implementation of the discrete pulse transform," in *Proc. Int. Conf. Numer. Anal. Appl. Math.* Weinheim, Germany, 2006, pp. 15–19.

[24] I. Fabris-Rotelli and S. J. Van der Walt, "The Discrete Pulse Transform in two dimensions," in *Proceedings of the 20th Annual Symposium of the Pattern Recognition Association of South Africa*, 2009. [Online]. Available: http://dip.sun.ac.za/~stefan/dpt/

[25] R. Barrett, *Templates for the solution of linear systems: building blocks for iterative methods.* Siam, 1994, no. 43.

[26] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Computer Vision, 2001. Proceedings of Eighth IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 416–423.

[27] G. Simmons, *Introduction to Topology and Modern Analysis*, ser. International Series in Pure and Applied Mathematics. Krieger Publishing Company, 1963.

[28] R. J. O'Callaghan and D. R. Bull, "Combined morphological-spectral unsupervised image segmentation," *Image Processing, IEEE Transactions on*, vol. 14, no. 1, pp. 49–62, 2005.

[29] C.-T. Li and R. Wilson, "Image segmentation based on a multiresolution bayesian framework," in *Image Processing, 1998. Proceedings of International Conference on.* IEEE, 1998, pp. 761–765.

[30] G. K. Ouzounis and M. H. Wilkinson, "Countering oversegmentation in partitioning-based connectivities," in *Image Processing, 2005. IEEE International Conference on*, vol. 3. IEEE, 2005, pp. III–844.

[31] M. Wilkinson, "Attribute-space connected filters," in *Mathematical Morphology: 40 Years On*, ser. Computational Imaging and Vision, C. Ronse, L. Najman, and E. DecenciÃĺre, Eds. Springer Netherlands, 2005, vol. 30, pp. 85–94.

[32] J. Goutsias, L. Vincent, and D. S. Bloomberg, *Mathematical morphology and its applications to image and signal processing*. Springer, 2000, vol. 18, ch. Practical extensions of connected operators, pp. 97–110.

[33] C. S. Tzafestas and P. Maragos, "Shape connectivity: multiscale analysis and application to generalized granulometries," *Journal of Mathematical Imaging and Vision*, vol. 17, no. 2, pp. 109–129, 2002.

[34] M. H. Wilkinson, "Connected filtering by reconstruction: Basis and new advances," in *Image Processing, 2008. 15th IEEE International Conference on*. IEEE, 2008, pp. 2180–2183.

[35] I. R. Terol-Villalobos, J. D. Mendiola-Santibáñez, and S. L. Canchola-Magdaleno, "Image segmentation and filtering based on transformations with reconstruction criteria," *Journal of Visual Communication and Image Representation*, vol. 17, no. 1, pp. 107–130, 2006.

[36] W. Law and A. C. Chung, "Minimal weighted local variance as edge detector for active contour models," in *Computer Vision–ACCV 2006*. Springer, 2006, pp. 622–632.

[37] M. W. Graham, J. D. Gibbs, and W. E. Higgins, "Robust system for human airway-tree segmentation," in *Medical Imaging*. International Society for Optics and Photonics, 2008, pp. 69 141J–69 141J.

[38] I. Santillán, A. M. Herrera-Navarro, J. D. Mendiola-Santibáñez, and I. R. Terol-Villalobos, "Morphological connected filtering on viscous lattices," *Journal of Mathematical Imaging and Vision*, vol. 36, no. 3, pp. 254–269, 2010.

[39] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[40] A. Raj, P. van den Bogaard, S. A. Rifkin, A. van Oudenaarden, and S. Tyagi, "Imaging individual mrna molecules using multiple singly labeled probes," *Nature Methods*, vol. 5, no. 10, pp. 877–879, 2008.

[41] D. Marr and A. Vision, "Vision: A computational investigation into the human representation and processing of visual information," *WH San Francisco: Freeman and Company*, 1982.

[42] J. Serra, "A lattice approach to image segmentation," *Journal of Mathematical Imaging and Vision*, vol. 24, no. 1, pp. 83–130, 2006.

[43] R. C. Gonzalez and E. Richard, "Digital image processing," *ed: Prentice Hall Press, ISBN 0-201-18075-8*, 2002.

[44] J. R. Beveridge, J. Griffith, R. R. Kohler, A. R. Hanson, and E. M. Riseman, "Segmenting images using localized histograms and region merging," *International Journal of Computer Vision*, vol. 2, no. 3, pp. 311–347, 1989.

[45] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[46] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, no. 9, pp. 1277–1294, 1993.

[47] W.-Y. Ma and B. Manjunath, "Edge flow: a framework of boundary detection and image segmentation," in *Computer Vision and Pattern Recognition, 1997. Proceedings of IEEE Computer Society Conference on.* IEEE, 1997, pp. 744–749.

[48] C. Revol and M. Jourlin, "A new minimum variance region growing algorithm for image segmentation," *Pattern Recognition Letters*, vol. 18, no. 3, pp. 249–258, 1997.

[49] R. Adams and L. Bischof, "Seeded region growing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 6, pp. 641–647, 1994.

[50] R. Ohlander, K. Price, and D. R. Reddy, "Picture segmentation using a recursive region splitting method," *Computer Graphics and Image Processing*, vol. 8, no. 3, pp. 313–333, 1978.

[51] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 6, pp. 583–598, 1991.

[52] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[53] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000.

[54] S. Borra and S. Sarkar, "A framework for performance characterization of intermediate-level grouping modules," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 11, pp. 1306–1312, 1997.

[55] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[56] M. Meilă, "Comparing clusterings: an axiomatic view," in *Proceedings of the 22nd International Conference on Machine Learning*. ACM, 2005, pp. 577–584.

[57] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions: An empirical evaluation," in *Computer Vision and Pattern Recognition, 2009. IEEE Conference on*. IEEE, 2009, pp. 2294–2301.

[58] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 5, pp. 530–549, 2004.

[59] C. Fowlkes, D. Martin, and J. Malik, "Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches," in *Computer Vision and Pattern Recognition, 2003. Proceedings of the IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–54.

[60] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine learning*. ACM, 2006, pp. 233–240.

[61] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[62] A. V. Goldberg and R. Kennedy, "An efficient cost scaling algorithm for the assignment problem," *Mathematical Programming*, vol. 71, no. 2, pp. 153–177, 1995.

[63] M. G. Ramos, S. S. Hemami, and M. A. Tamburro, "Psychovisually-based multiresolution image segmentation," in *Image Processing, Proceedings of the International Conference on*, vol. 3. IEEE, 1997, pp. 66–69.

[64] T. Lindeberg, "Scale-space for discrete signals," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 3, pp. 234–254, 1990.

[65] V. Estivill-Castro, "Why so many clustering algorithms: a position paper," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 1, pp. 65–75, 2002.

[66] P. Debba, A. Stein, F. van der Merwe, E. Carranza, and A. Lucieer, "Field sampling from a segmented image," in *Proceedings of the International Conference on Computational Science and its Applications, Part I Annual Symposium on Computational Geometry*. Springer-Verlag, 2008, pp. 756–768.

[67] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering," in *Proceedings of the 10th Annual Symposium on Computational Geometry*. ACM, 1994, pp. 332–339.

[68] J. Besag, "On the statistical analysis of dirty pictures," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 259–302, 1986.

[69] S. J. Van der Walt, "Super-resolution imaging," Ph.D. dissertation, Stellenbosch: University of Stellenbosch, 2010.

[70] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 117–156, 1998.

[71] L. S. Davis, "A survey of edge detection techniques," *Computer Graphics and Image Processing*, vol. 4, no. 3, pp. 248–270, 1975.

[72] I. Fabris-Rotelli, "The discrete pulse transform for images with entropy-based feature detection," in *Proceedings of the 22nd Annual Symposium of the Pattern Recognition Association of South Africa*, 2011, pp. 22–25.

[73] G. G. P. License, "Mingw - minimalist gnu for windows," this is an electronic webpage. Date of publication: [Date unavailable]. Date retrieved: September 7, 2013. Date last modified: May 25, 2012. [Online]. Available: http://www.mingw.org/

[74] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

APPENDIX A

THE DPT LIBRARY GUIDE

The work done on creating a sufficient algorithm does not add to the body of knowledge if other researchers can not use it. A library called The DPT Library was created which other researchers can access and utilize to facilitate research on the Discrete Pulse Transform. A code library requires the easy use of the functions it contain. This sections contains the library guide on how to use the DPT Library Code. The library is provided on Github [22] . The guide is given in full:

A.1 DPT LIBRARY GUIDE INDEX

The DPT Library Guide contains the following topics:

Overview
How to use `DPT.h`
`DPT2PGraph`
`ReconstructGraph`
`data_structure`
`connectivity`
`DPT_Graph` - The Integer Structure
The Future
Example

A.2 OVERVIEW

This library implements the Discrete Pulse Transform in n-dimensions. It is written in C++ but can easily be changed to a C library by exchanging all the reference-pointers and the "new" commands with relative C commands (a step required for the library). It consist of two main commands, the transform and the reconstruction.

A.3 HOW TO USE DPT.H

The header file `DPT.h` must be included into your programming code while `DPT.cpp` must be available in the compiler directory. The code must be compiled with the -O3 flag. The library was tested with Mingw32 [73]. There exist two main functions `DPT2PGraph` and `ReconstructGraph`.

- The `DPT2PGraph` function are used to perform the DPT with the supplied input data.
- The `ReconstructGraph` function is used to reconstruct the dataset using specified criteria.

A.4 DPT2GRAPH

The `DPT2Graph` is the main library function which applies the DPT to the supplied data. The function requires three inputs which consists of the required connectivity for the data, the data on which the DPT must be performed and a place to store the DPT graph. The function then saves the DPT in the supplied memory block and also gives an indication on how many pulses were extracted throughout the algorithm.

```
int DPT2Graph(   int *&connectivity,
                 int *&data_structure,
                 PGraphNode *&DPT_Graph );
```

Inputs:

connectivity	This is an integer pointer to the connectivity vector
data_structure	This is an integer pointer to the data vector on which the DPT must be performed
DPT_Graph	This is a pointer to the special data structure *PGraphNode* which will contain the DPT.

Output: The function outputs an integer which denotes the number of pulses which were created in the DPT graph.

A.5 RECONSTRUCTGRAPH

The `ReconstructGraph` is a basic function which can be used to manipulate the various pulses extracted from user supplied data. The function have five inputs which includes the *Pulse Graph* and the range of pulses which needs to be extracted. Some advance features includes specifying a variable range of pulses and also a possible offset on the output for presentation purposes. Pulses can have negative values thus if your output data structure can not handle negative values offset can be added to all the answers.

```
int *ReconstructGraph(  int n_nodes,
                        PGraphNode *&DPT_Graph,
                        int *pulseRange,
                        int size_pulseRange,
                        int offset);
```

Inputs:

n_nodes	This is an integer which provides the number of pulses in the DPT graph
DPT_Graph	This is a pointer to the special data structure *PGraphNode* which contains the DPT.
pulseRange	This is a vector containing the ranges of pulses which needs to be used for reconstruction.
size_pulseRange	This is the number of elements within the *pulseRange* vector.
offset	This is the default value for all data points in the data set.

Output: The function outputs the reconstructed data structure where each element relates to a specific data point in the original data set.

The input variable *pulseRange* must have an even value. The first element is a lower bound where the second element is an upper bound, the third element is then again a lower bound etc. By example:

1. Reconstruct all the pulses which have size from 5 to 10 and a size from 20 to 30.

```
size_pulseRange = 4;

pulseRange[0] = 5;

pulseRange[1] = 10;

pulseRange[2] = 20;

pulseRange[3] = 30;
```

2. Reconstruct all the pulses which have a size of 12.

```
size_pulseRange = 2;

pulseRange[0] = 12;

pulseRange[1] = 12;
```

sdata_structure

The `data_structure` variable is used to pass the data points itself and must consist of an element for each d-tuple thus using n-dimensions where each dimension d_k has a range $[0, d_{k_{max}})$ then the vector would have the following format:

data_structure

$[0]$	= data point at $(0, 0, ..., 0)$
$[1]$	= data point at $(1, 0, ..., 0)$
$\vdots$	
$[d_{1_{max}} - 1]$	= data point at $(d_{1_{max}} - 1, 0, ..., 0)$
$[d_{1_{max}} - 1 + 1]$	= data point at $(0, 1, ..., 0)$
$[d_{1_{max}} - 1 + 2]$	= data point at $(0, 2, ..., 0)$
$\vdots$	
$[d_{1_{max}} * d_{2_{max}} * .. * d_{n_{max}} - 1]$	= data point at $(d_{1_{max}} - 1, d_{2_{max}} - 1, .., d_{n_{max}} - 1)$.

The `data_structure` relates the position(n-tuple) of the data point $x = (d_1, d_2, ..., d_n)$ to an index, this index i in the vector `data_structure[i]` can be calculated with:

$$i = d_1 + d_2 * d_{1_{max}} + d_3 * (d_{2_{max}} * d_{1_{max}}) + ..+ \\ d_n * (d_{(n-1)_{max}} * d_{(n-2)_{max}} * .. * d_{1_{max}}) \tag{A.1}$$

This can be illustrated by using an example:

Assuming a 3-dimensional structure of size $3 \times 2 \times 4$, thus dimension one ranges from $[0, 3]$, dimension two ranges from $[0, 2]$ and dimension three ranges from $[0, 4]$. The data set has a cardinality of 24. The `data_structure` that needs to be created for the DPT decomposition will then require 24 elements in the vector and will look as follow:

$$
\begin{aligned}
\texttt{data\_structure}[0] &= \text{element } (0,0,0)\\
\texttt{data\_structure}[1] &= \text{element } (1,0,0)\\
\texttt{data\_structure}[2] &= \text{element } (2,0,0)\\
\texttt{data\_structure}[3] &= \text{element } (0,1,0)\\
\texttt{data\_structure}[4] &= \text{element } (1,1,0)\\
\texttt{data\_structure}[5] &= \text{element } (2,1,0)\\
\texttt{data\_structure}[6] &= \text{element } (0,0,1)\\
\texttt{data\_structure}[7] &= \text{element } (1,0,1)\\
\texttt{data\_structure}[8] &= \text{element } (2,0,1)\\
\texttt{data\_structure}[9] &= \text{element } (0,1,1)\\
\texttt{data\_structure}[10] &= \text{element } (1,1,1)\\
\texttt{data\_structure}[11] &= \text{element } (2,1,1)\\
\texttt{data\_structure}[12] &= \text{element } (0,0,2)\\
\texttt{data\_structure}[13] &= \text{element } (1,0,2)\\
\texttt{data\_structure}[14] &= \text{element } (2,0,2)\\
\texttt{data\_structure}[15] &= \text{element } (0,1,2)\\
\texttt{data\_structure}[16] &= \text{element } (1,1,2)\\
\texttt{data\_structure}[17] &= \text{element } (2,1,2)\\
\texttt{data\_structure}[18] &= \text{element } (0,0,3)\\
\texttt{data\_structure}[19] &= \text{element } (1,0,3)\\
\texttt{data\_structure}[20] &= \text{element } (2,0,3)\\
\texttt{data\_structure}[21] &= \text{element } (0,1,3)\\
\texttt{data\_structure}[22] &= \text{element } (1,1,3)\\
\texttt{data\_structure}[23] &= \text{element } (2,1,3)
\end{aligned}
$$

A.6 CONNECTIVITY

The `connectivity` variable is used to pass information regarding the `data_structure`, such as the dimension n of the data, each dimensions range and the connectivity of the data. The connectivity can either be structured such as 4-connectivity, k-connectivity or each data point can have its own unique connection. The connection is specified by taking the change in the current position to the connected data point. The connectivity vector first contains the number of dimensions followed with each dimension's range, the number of connections per data point and then the change in dimension to specify the connected data point. The structure is as follow:

connectivity

$[0]$	$= n$	Amount of dimensions in data structure
$[1]$	$= d_{1_{max}}$	d_1 ranges from $[0, d_{1_{max}}]$
$\vdots$		
$[n]$	$= d_{n_{max}}$	d_n ranges from $[0, d_{n_{max}}]$
$[n+1]$	$= k$	Amount of connections per node.
$[n+1+0*n+1]$	$= \delta_{d_1,1}$	Change in dimension 1 for connection 1.
$[n+1+0*n+2]$	$= \delta_{d_2,1}$	Change in dimension 2 for connection 1.
$\vdots$		
$[n+1+0*n+n]$	$= \delta_{d_n,1}$	Change in dimension n for connection 1.
$[n+1+1*n+1]$	$= \delta_{d_1,2}$	Change in dimension 1 for connection 2.
$\vdots$		
$[n+1+(k-1)*n+n]$	$= \delta_{d_n,k}$	Change in dimension n for connection k.

This can be illustrated by using an example. Assume an image of two dimensions of size 121×153 and 4 connectivity then:

$$
\begin{aligned}
\texttt{connectivity}[0] &= 2 \\
\texttt{connectivity}[1] &= 121 \\
\texttt{connectivity}[2] &= 153 \\
\texttt{connectivity}[3] &= 4 \\
\texttt{connectivity}[4] &= 1 \\
\texttt{connectivity}[5] &= 0 \\
\texttt{connectivity}[6] &= -1 \\
\texttt{connectivity}[7] &= 0 \\
\texttt{connectivity}[8] &= 0 \\
\texttt{connectivity}[9] &= 1 \\
\texttt{connectivity}[10] &= 0 \\
\texttt{connectivity}[11] &= -1
\end{aligned}
$$

A.7 DPT_GRAPH - THE INTSTRUCT

The `DPT_Graph` is output into a type of tree represented by the `PGraphNode` *struct* which can be found in `DPT.h`. This *struct* can be converted into an array of vectors which uses the index of the array to represent edges between various nodes. Each node is represented by a vector.

The following notation is equivalent:

$$
\begin{aligned}
\text{vector}[i] = [v_1 \quad v_2 \quad v_3 \quad v_4] \iff & \text{vector}[i][0] = v_1 \\
& \text{vector}[i][1] = v_2 \\
& \text{vector}[i][2] = v_3 \\
& \text{vector}[i][3] = v_4
\end{aligned}
$$

Each vector at index i has the following structure:

IntStruct

$[i][0]$	The cardinality of the vector.
$[i][1]$	The size of the represented node.
$[i][2]$	The height of the represented node.
$[i][3]$	The index of the parent node.
$[i][4]$	The number of children c the represented node have.
$[i][4+1]$	The index of the first child.
$\vdots$	
$[i][4+c]$	
The index of child c.	

Every node which has a height equal to zero, has no children and the index of this node relates to the n-tuple of the data point as discussed in `data_structure` section. The node with a connected pulse equal to minus one has no parent and is thus the root of the tree.

Assuming we use a 1-dimensional signal with 4 data points such that $x = [5 \quad 8 \quad 4 \quad 4]$. The DPT will then yield effectively four pulses where the fourth pulse is the zero pulse which will form the root of the tree:

$$\begin{aligned}
&\text{Pulse 1} \quad p_1 = [0 \quad 3 \quad 0 \quad 0] \\
&\text{Pulse 2} \quad p_2 = [1 \quad 1 \quad 0 \quad 0] \\
&\text{Pulse 3} \quad p_3 = [4 \quad 4 \quad 4 \quad 4] \\
&\text{Pulse 4} \quad p_4 = [0 \quad 0 \quad 0 \quad 0]
\end{aligned}$$

These pulses can then effectively be stored in the `IntStruct` format which will represent these pulses as a tree:

$$
\begin{aligned}
\texttt{IntStruct}[0] &= [5 \quad 1 \quad 0 \quad 5 \quad 0] \\
\texttt{IntStruct}[1] &= [5 \quad 1 \quad 0 \quad 4 \quad 0] \\
\texttt{IntStruct}[2] &= [5 \quad 1 \quad 0 \quad 6 \quad 0] \\
\texttt{IntStruct}[3] &= [5 \quad 1 \quad 0 \quad 6 \quad 0] \\
\texttt{IntStruct}[4] &= [6 \quad 1 \quad 3 \quad 5 \quad 1 \quad 1] \\
\texttt{IntStruct}[5] &= [7 \quad 2 \quad 1 \quad 6 \quad 2 \quad 4 \quad 0] \\
\texttt{IntStruct}[6] &= [8 \quad 4 \quad 4 \quad 7 \quad 3 \quad 2 \quad 3 \quad 5] \\
\texttt{IntStruct}[7] &= [6 \quad 4 \quad 0 \quad -1 \quad 1 \quad 6]
\end{aligned}
$$

A.8 THE FUTURE

No library is ever complete thus there is still a lot of work that can be done on the library. Currently the library is doing the basic functions concerning the DPT, more advanced functions can still be developed. One important aspect is that the code must be ported to be ANSI C compliant.

A very important aspect of the library is to increase the ease of use thus supplying direct interfaces to commonly used data sets such as images, video and hyper spectral data.

A.9 EXAMPLE

A simple example in using the library for image processing is given in this section while utilizing OpenCV [74] as the image processing toolbox. Firstly two functions needs to be created to convert between the `IplImage` and the `IntStruct`. `IplImage` is OpenCV's specified struct for storing images. The conversion shown in Listing A.1 converts `IplImage` to an array of integer and returns an integer pointer to the beginning of the array.

Listing A.1: Convert from IplImage to IntStruct

```
int *IplImage2struct(IplImage *&img) {
    int *img_struct = 0;
    img_struct = new int[img->width*img->height];

    for (int y = 0; y < img->height; ++y)
    {
        uchar *ptr = (uchar *)(img->imageData + y*img->widthStep);
```

```
        for (int x = 0; x < img->width; ++x)
        {
            img_struct[y*img->width + x] = ptr[x];
        }
    }
    return img_struct;
}
```

Listing A.2 show how to convert the `IntStruct` to an `IplImage`. This is required if we want to reconstruct the DPT with specified pulses and display the image created. The function returns a pointer to the beginning of the image data.

Listing A.2: Convert from IntStruct to IplImage

```
IplImage *struct2IplImage(int *&data_struct,
                          int width, int height, int depth) {
    IplImage *img = 0;
    img = cvCreateImage(cvSize(width,height),depth,1);

    for (int y = 0; y < img->height; ++y)
    {
        uchar *ptr = (uchar *)(img->imageData + y*img->widthStep);
        for (int x = 0; x < img->width; ++x)
        {
            ptr[x] = data_struct[y*img->width + x];
        }
    }
    return img;
}
```

Before we can perform the DPT we still need to define the `connectivity` vector which is shown in Listing A.3. We will be using 4-connectivity and count the number of data points in the data set. We directly add the file name of the image we are loading.

Listing A.3: Create the connectivity vector array

```
int CreateDataStructureOpenCV(int *&connectivity,
                              int *&data_structure,
                              char *filename) {
    int n_nodes = 0;
```

```
    IplImage *img = cvLoadImage(filename,0);

    int width = img->width;
    int height = img->height;

    // count number of data points
    n_nodes = width*height;

    data_structure = IplImage2struct(img);

    // use four connectivity
    connectivity = new int [12];
    connectivity[0] = 2;            //dimensions
    connectivity[1] = img->width;   // x - dim[1] - size
    connectivity[2] = img->height; //y - dim[2] - size
    connectivity[3] = 4;            //graph connections
    //right
    connectivity[4] =  1;      //connection[1] change in dim[1]
    connectivity[5] =  0;      //connection[1] change in dim[2]
    //left
    connectivity[6] = -1;      //connection[1] change in dim[1]
    connectivity[7] =  0;      //connection[1] change in dim[2]
        //bottom
    connectivity[8] =  0;      //etc.
    connectivity[9] =  1;
        //top
    connectivity[10] =  0;
    connectivity[11] = -1;

        return n_nodes;
}
```

The complete process can then be completed in the final function. We will reconstruct only all the pulses of size 25 and give the final result an offset of 128. The amount of offset is related to the 8-bit grayscale images that we use. We need to add the offset as we are working with unsigned 8-bit integers. Listing A.4 first creates the integer structure followed with the connectivity vector, the DPT decomposition and also a reconstruction.

Listing A.4: DPT Library Example

```
void DPTexample(char *data_filename) {
    int *connectivity;
    int *data_structure;
    // the total number of nodes
    int n_nodes = 0;
    int n_pulses;
    // The DPT decomposition variable
    PGraphNode *DPT_Graph;

    // Create Data Structure, OpenCV specific
    n_nodes =  CreateDataStructureOpenCV(connectivity,
                                         data_structure,
                                         data_filename);

    // DPT decomposition
    n_pulses = DPT2PGraph(connectivity,
                          data_structure,
                          DPT_Graph);

    // Get original image sizes
    IplImage *img = cvLoadImage(data_filename,0);

        int *intStruct;
    int offset = 0;
    int size_pulseRange = 2;
    int *pulseRange;
    pulseRange = new int[size_pulseRange];
    pulseRange[0] = 25;
    pulseRange[1] = 25;

    // Get data_structure reconstruction
    intStruct = ReconstructGraph(n_nodes, DPT_Graph, pulseRange,
                                 size_pulseRange, offset);

    // convert intStruct to an IplImage of OPENCV
    IplImage *image = struct2IplImage(intStruct,
                                      img->width, img->height);
    cvShowImage("All pulses of size 50",image);
    cvWaitKey(0);
}
```