

# Image Compression Using the One-Dimensional Discrete Pulse Transform

by

**Ernst Wilhelm Uys**

*Thesis presented in partial fulfillment of the requirements  
for the degree of Master of Science  
(Physical and Mathematical Analysis)*



*at*

*Stellenbosch University*

Department of Physics  
Faculty of Science

Supervisor: Dr. Carl Heinrich Rohwer  
Co-supervisor: Prof. Hans Christoph Eggers

Date: March 2011



## **Declaration**

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 3 February 2011

Copyright © 2011 Stellenbosch University

All rights reserved

## Abstract

The nonlinear *LULU* smoothers excel at removing impulsive noise from sequences and possess a variety of theoretical properties that make it possible to perform a so-called Discrete Pulse Transform, which is a novel multiresolution analysis technique that decomposes a sequence into resolution levels with a large amount of structure, analogous to a Discrete Wavelet Transform.

We explore the use of a one-dimensional Discrete Pulse Transform as the central element in a digital image compressor. We depend crucially on the ability of space-filling scanning orders to map the two-dimensional image data to one dimension, sacrificing as little image structure as possible. Both lossless and lossy image compression are considered, leading to five new image compression schemes that give promising results when compared to state-of-the-art image compressors.

## Opsomming

Die nie-lineêre *LULU* gladstrykers verwyder impulsiewe geraas baie goed uit rye en besit verskeie teoretiese eienskappe wat dit moontlik maak om 'n sogenoemde Diskrete Puls Transform uit te voer; 'n nuwe multiresolusie analise tegniek wat 'n ry opbreek in 'n versameling resolusie vlakke wat 'n groot hoeveelheid struktuur bevat, soortgelyk tot 'n Diskrete Golfie Transform.

Ons ondersoek of 'n eendimensionele Diskrete Puls Transform as die sentrale element in 'n digitale beeld kompressor gebruik kan word. Ons is afhanklik van ruimte-vullende skandeur ordes om die tweedimensionele beeld data om te skakel na een dimensie, sonder om te veel beeld struktuur te verloor. Vyf nuwe beeld kompressie skemas word bespreek. Hierdie skemas lewer belowende resultate wanneer dit met die beste hedendaagse beeld kompressors vergelyk word.

## Acknowledgements

This thesis is really the product of two interests that were sparked in lectures given by my supervisors in my honours year: Dr. Carl Rohwer introduced me to *LULU* theory, and Prof. Hans Eggers introduced me to information theory. I want to thank them for these introductions, without which this thesis would not have been possible.

I wish to express my sincere thanks to Dr. Carl Rohwer for his patience and wisdom in guiding me. I could not have asked for a better supervisor, and I value the knowledge — both technical and non-technical — that I have gained from him.

Special thanks is due to the NRF for providing a post-graduate bursary, and also to Prof. Eggers for arranging it.

I also want to thank Marc-Allen Johnson and Indutech (Pty) Ltd. for being instrumental in arranging much-needed employment during various stages of this project. It is greatly appreciated.

And last, but certainly not least, to my parents: thank you for your unfaltering love and support throughout this endeavour.

# Contents

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 The Discrete Pulse Transform</b>	<b>3</b>
2.1 Introduction	3
2.2 Smoothing	3
2.2.1 Purpose and Definition	3
2.2.2 Operation	4
2.2.3 Smoother Design	6
2.2.3.1 Effectiveness	6
2.2.3.2 Robustness	7
2.2.3.3 Consistency	7
2.2.3.4 Efficiency	7
2.2.4 Smoother Definitions	7
2.2.5 Smoother Axioms	9
2.2.6 Linearity and Nonlinearity	10
2.2.7 Idempotence and Co-idempotence	10
2.3 <i>LULU</i> Smoothers	12
2.3.1 Definitions	12
2.3.2 Properties	14
2.3.2.1 Ordered Semigroup Structure	14
2.3.2.2 Local Monotonicity	17
2.3.2.3 Idempotence and Co-idempotence	19
2.3.2.4 Variation Reduction and Preservation	19
2.3.2.5 Shape Preservation	19
2.3.2.6 Syntonicity	20
2.3.2.7 Signal/Noise Ambiguity	20
2.3.2.8 Duality	20
2.4 The Discrete Pulse Transform	21
2.4.1 Multiresolution Analysis	21
2.4.2 Definitions	22
2.4.3 Properties	22
2.4.3.1 Structure	22
2.4.3.2 Consistency	22

2.4.3.3	Preservation and Reduction of Total Variation . . . . .	25
2.4.3.4	Total Pulse Count . . . . .	26
2.4.3.5	Shape Preservation . . . . .	26
2.4.3.6	Commutation with Quantization and Thresholding Operators . . . . .	27
2.4.4	Extension to Higher Dimensions . . . . .	27
2.5	Conclusion . . . . .	28
<b>3</b>	<b>Data Compression</b> . . . . .	<b>29</b>
3.1	Introduction . . . . .	29
3.2	Information theory . . . . .	29
3.2.1	Information content . . . . .	30
3.2.2	Entropy . . . . .	31
3.2.3	Relative entropy . . . . .	31
3.3	Data modelling . . . . .	32
3.3.1	Lossless and lossy data compression . . . . .	32
3.4	Coding . . . . .	33
3.4.1	Overview . . . . .	33
3.4.2	Symbol codes . . . . .	33
3.4.2.1	Prefix-free codes . . . . .	34
3.4.2.2	The Kraft inequality . . . . .	34
3.4.2.3	Characterising the average codeword length . . . . .	35
3.4.3	Huffman coding . . . . .	36
3.4.3.1	Weaknesses of Huffman coding . . . . .	38
3.4.4	Golomb coding . . . . .	39
3.4.4.1	Adaptive and Generalized Golomb coding . . . . .	40
3.5	Stream codes . . . . .	41
3.5.1	Arithmetic coding . . . . .	41
3.5.1.1	Binary arithmetic codes . . . . .	43
3.5.1.2	Probabilistic modelling and arithmetic coding . . . . .	43
3.6	Quantization . . . . .	43
3.6.1	Scalar quantization . . . . .	44
3.6.1.1	Uniform quantization . . . . .	46
3.6.1.2	Nonuniform quantization . . . . .	47
3.6.2	Properties of Probabilistically Optimized Scalar Quantizers . . . . .	49
3.6.3	Entropy Coding of Quantizer Output . . . . .	50
3.7	Conclusion . . . . .	50
<b>4</b>	<b>Image Compression using the DPT</b> . . . . .	<b>51</b>
4.1	Introduction . . . . .	51
4.2	Compression Scheme Structure . . . . .	51
4.3	Sequencing . . . . .	52
4.3.1	Input . . . . .	52
4.3.1.1	Image Test Set . . . . .	53
4.3.2	Colour Translation . . . . .	53
4.3.3	Image Scan . . . . .	55
4.3.3.1	Goal . . . . .	55
4.3.3.2	Scanning Orders . . . . .	55
4.3.3.3	Scanning Order Total Variation . . . . .	59



4.3.3.4	Scanning Order Isotropy . . . . .	67
4.3.3.5	Image Anisotropy . . . . .	67
4.3.4	Image Tiling . . . . .	69
4.4	Lossless Compression . . . . .	74
4.4.1	PulseTrain Scheme . . . . .	74
4.4.1.1	Modelling . . . . .	74
4.4.1.2	Coding . . . . .	81
4.4.2	PulseTree Scheme . . . . .	84
4.4.2.1	Modelling . . . . .	84
4.4.2.2	Coding . . . . .	84
4.5	Lossy Compression . . . . .	90
4.5.1	Fidelity Criteria . . . . .	90
4.5.1.1	Peak Signal-to-Noise Ratio (PSNR) . . . . .	90
4.5.1.2	Structural Similarity Index (SSIM) . . . . .	91
4.5.2	LossyPulseTree scheme . . . . .	92
4.5.2.1	Modelling . . . . .	92
4.5.2.2	Coding . . . . .	96
4.6	Conclusion . . . . .	96
<b>5</b>	<b>Conclusion</b>	<b>100</b>
	<b>Bibliography</b>	<b>103</b>

## List of Figures

2.1	Smoothing two noise-corrupted signals with linear and nonlinear smoothers. Note that the underlying data is discrete; linear interpolation is used only for clarity of presentation. . . . .	5
2.2	Comparison between the probability distributions of impulsive and non-impulsive noise sources. . . . .	6
2.3	Illustration of the partial order on $\mathcal{S}$ . . . . .	9
2.4	A two-stage separator cascade. . . . .	11
2.5	The general operation of $L_n$ and $U_n$ . . . . .	14
2.6	Smoothing a signal contaminated with nonimpulsive noise with the $F_n$ operator. . . . .	15
2.7	Smoothing a signal contaminated with impulsive noise with the $C_n$ operator. . . . .	16
2.8	Mutual order relations between the key <i>LULU</i> operators (adapted from [67]). . . . .	18
2.9	Nested subsets of $\mathcal{M}$ . . . . .	18
2.10	Signal/noise ambiguity with $L_n U_n$ and $U_n L_n$ . . . . .	20
2.11	Multiresolution analysis of a sequence $x$ , where $s_i$ represents a smoothed level and $r_i$ represents a resolution level. . . . .	21
2.12	Demonstration of a DPT using the $C_n$ operator. Note that only non-empty levels are shown. . . . .	23
2.13	DPT output monotonicities. . . . .	24
2.14	Variation spectrum of the DPT of a typical random sequence of length 100. . . . .	25
2.15	Presence spectrum of the DPT of a typical random sequence of length 100. . . . .	26
3.1	Binary tree corresponding to the Huffman code in Table 3.1 . . . . .	38
3.2	Discrete geometric probability distribution with parameter $g = 0.25$ . . . . .	40
3.3	Probability distribution function and cumulative distribution function of a random variable. . . . .	41
3.4	Using arithmetic coding to code the sequence $a_2 a_3 a_1$ from the ensemble $\{ \{a_1, 0.3\}, \{a_2, 0.6\}, \{a_3, 0.1\} \}$ . . . . .	42
3.5	The two types of quantization. . . . .	44
3.6	Generic scalar quantization. . . . .	45
3.7	Midrise and midtread quantizers. . . . .	46
3.8	Non-uniform quantization. . . . .	47
4.1	Compression scheme structure. . . . .	51
4.2	Images from the Image Test Set (see Section 4.3.1.1) . . . . .	54

4.3	RGB and $YC_bC_r$ colour planes of an image depicting a landscape. . .	56
4.4	RGB and $YC_bC_r$ colour planes of an image depicting a galaxy. . . .	57
4.5	Using the Hilbert scanning order to scan a $4 \times 4$ grid. . . . .	58
4.6	GP scanning order. . . . .	60
4.7	Serpentine scanning order. . . . .	60
4.8	Luxburg Variation 2 scanning order. . . . .	60
4.9	Meurthe scanning order. . . . .	61
4.10	Coil scanning order. . . . .	61
4.11	R scanning order. . . . .	61
4.12	Kochel scanning order. . . . .	62
4.13	H scanning order. . . . .	62
4.14	Hilbert scanning order. . . . .	62
4.15	Z scanning order. . . . .	63
4.16	$\beta\Omega$ scanning order. . . . .	63
4.17	$AR^2W^2$ scanning order. . . . .	63
4.18	DPT anisotropy for the Landscape image. . . . .	70
4.19	DPT anisotropy for the Portrait image. . . . .	71
4.20	DPT anisotropy for the Clouds image. . . . .	72
4.21	DPT anisotropy for the Noise image. . . . .	73
4.22	Tiling an image with progressively smaller tiles. . . . .	74
4.23	The three <code>PulseTrain</code> models. . . . .	75
4.24	Basic lossless model compression factors for various tile sizes and scanning orders with DPT operator $L$ . . . . .	76
4.25	Basic lossless model compression factors for various tile sizes and scanning orders with DPT operator $C$ . . . . .	77
4.26	Typical DPT presence spectrum. . . . .	78
4.27	Comparison between compression factors of the basic lossless model and the pulse count prediction lossless model. In both cases the $\beta\Omega$ scanning order and the $F$ DPT operator was used. . . . .	79
4.28	Offset entropy per resolution level for tile size 512 (first 100 levels). . . . .	80
4.29	Fraction of total pulses contained in first $n$ resolution levels (determined experimentally). . . . .	80
4.30	Threshold model compression factors for various tile sizes. Scanning order $\beta\Omega$ , DPT operator $F$ . . . . .	81
4.31	Sample worst-case images for lossless compression. . . . .	85
4.32	Model for the <code>PulseTree</code> scheme. . . . .	86
4.33	Space tracking with the <code>PulseTree</code> scheme. . . . .	89
4.34	Comparing the PSNR and SSIM image quality models. (b) and (c) have the same PSNR, but the SSIM indices correctly indicate that (b) has higher fidelity than (c). . . . .	92
4.35	Effects of quantization with isotropic and anisotropic scanning orders. With both images the first 10 resolution levels were left out in a partial DPT reconstruction. . . . .	94
4.36	Blocking artifacts in the <code>LossyPulseTree</code> scheme when the tile size is too small. . . . .	97
4.37	Edge deterioration artifacts in the <code>LossyPulseTree</code> scheme. . . . .	98

## List of Tables

2.1	Multiplication table of the <i>LULLU</i> semigroup. . . . .	17
3.1	Huffman code example. . . . .	37
3.2	Golomb code with $m = 4$ . . . . .	39
3.3	Golomb code with $m = 5$ . . . . .	40
3.4	Signal-to-noise ratios (in dB) of optimal uniform and nonuniform quantizers for Gaussian and Laplacian distributions (adapted from [70]). . . . .	49
3.5	Output entropies (in bits) of optimal uniform and nonuniform quantizers for Gaussian and Laplacian distributions (adapted from [70]). . . . .	49
4.1	Locality measures of various scanning orders (adapted from [34]). . . . .	64
4.2	Total variation per pixel for scanning orders of atomic size 2. . . . .	65
4.3	Total variation per pixel for scanning orders of atomic size 3. . . . .	65
4.4	Scanning orders of atomic size 2 with lowest average variation. . . . .	66
4.5	Scanning orders of atomic size 3 with lowest average variation. . . . .	67
4.6	Anisotropy for scanning orders of atomic size 2. . . . .	68
4.7	Anisotropy for scanning orders of atomic size 3. . . . .	68
4.8	Basic lossless model compression factors for various DPT operators with tile size $512 \times 512$ and scanned with the H order. . . . .	77
4.9	Performance of Rice coding with the lossless PulseTrain scheme. . . . .	82
4.10	Performance of arithmetic coding with the lossless PulseTrain scheme. . . . .	83
4.11	Compression results for the simple PulseTree scheme using scanning order $\beta\Omega$ , DPT operator $F$ , using arithmetic coding. . . . .	88
4.12	Compression results for the PulseTree scheme using scanning order $\beta\Omega$ , DPT operator $L$ , using arithmetic coding. . . . .	88
4.13	Compression results for the verbatim PulseTree scheme using scanning order $\beta\Omega$ , DPT operator $L$ , using arithmetic coding. . . . .	88
4.14	Results of LossyPulseTree compression scheme with quantization and no Weber culling. . . . .	96
4.15	Results of LossyPulseTree compression scheme with quantization and Weber culling. . . . .	99

## Chapter 1 - Introduction

In a time that is being called the dawn of the “information age”, we expect quite naturally that the digital artifact of arguably our most important and information-rich sense, vision, will become increasingly important. Digital images are becoming ubiquitous and this has been helped on by the appearance of the internet and, more importantly, the availability of digital cameras — the widespread use of cheap digital cameras on cellphones being a notable example. An intimately related phenomenon is that of digital video, which is also becoming widespread, and advances in digital imaging usually leads quite straightforwardly to advances in digital video.

Why is image compression important? Surely technology advances quickly enough these days so that there is enough storage capacity for information rich media such as digital images and video? The furious pace at which technology is progressing is unfortunately also a double-edged sword, since it is not only storage technology that is improving constantly: sensor and image acquisition technologies are improving at a comparable pace, if not faster. In any case, there is a demand for ever higher image and video resolutions and it seems that, at least for the time being, storage capacity will not be able to comfortably supply the demand for information rich media.

Besides storage capacity there is also the need for these digital images and videos to be transmitted from point to point, typically around the globe via the internet. While it is true that the bandwidth of the average internet connection is increasing, once again we can argue that demand is outstripping supply, especially in non-first world countries where even basic internet access is still a problem.

Ultimately then, we view image compression and data compression in general as a trade-off between the fundamental constraints of space and time. With no data compression a large amount of storage space is needed, but no time is wasted on the decompression of the data at hand. With data compression the situation is reversed: less storage space is needed, but more time is needed to decompress the data. Adding the time needed for data transmission introduces a new constraint favouring the application of data compression. Data compression allows processing power to act as a mitigator of the limitation introduced by storage and transmission technology.

In this thesis we will investigate whether a one-dimensional Discrete Pulse Transform can be used to compress digital images. A Discrete Pulse Transform is a transform similar in some ways to a Discrete Wavelet Transform, which forms the basis of the JPEG2000 image compression standard which is the next iteration in the popular and widely successful image compression standard. Both of the transforms are examples of multiresolution analyses. Inspired by the success of using wavelets to compress digital images, we have naturally

asked ourselves the question as to what extent a Discrete Pulse Transform can be used to achieve the same goal.

The differences between the two transforms are numerous enough so that it is not immediately apparent whether using a Discrete Pulse Transform to compress a digital image will be successful. For example, a Discrete Pulse Transform is nonlinear and decomposes a sequence into pulses, whereas a Discrete Wavelet Transform is linear and decomposes a sequence dyadically into wavelet coefficients.

There is at least one property of a Discrete Pulse Transform that makes us optimistic about its use in image compression: shape preservation. Shape preservation roughly translates to edge preservation under quantization, among other things. This is in contrast to image compression schemes using linear multiresolution analysis techniques, like the Discrete Cosine Transform and the Discrete Wavelet Transform, where Gibbs phenomena appear as an undesired property, blurring edges and becoming the most visible compression artifact.

The plan of this thesis is as follows: in the second chapter we introduce the theory behind the Discrete Pulse Transform; in the third chapter we explore the topic of data compression. The promised synthesis of the title occurs in the fourth and final chapter.

## Chapter 2 - The Discrete Pulse Transform

### 2.1 Introduction

A *Discrete Pulse Transform* (hereafter abbreviated as *DPT*) is a nonlinear transform that decomposes a sequence of real numbers into a set of resolution sequences. Conceptually it is similar to the well-known and widely used Discrete Wavelet Transform.

The DPT is part of the so-called *LULU* theory<sup>1</sup>, which is a novel theory dealing with a set of nonlinear operators and their various properties with applications in a variety of areas. It provides a theory of nonlinear smoothing based on the wisdom of median smoothers, on which the *LULU* theory also casts some theoretical light [57, 61, 66]. The key theoretical properties of the DPT were proven during the 1990's and early 2000's [57, 65, 58, 59, 61, 66, 60, 62, 63, 67], based on some earlier industrial applications.

### 2.2 Smoothing

To understand the DPT it is necessary to understand how the *LULU* smoothers that form its foundations actually work. Before we move on to *LULU* smoothers proper, it will be useful to look at smoothers in general.

#### 2.2.1 Purpose and Definition

We consider smoothers as operators on sequences of real numbers. The types of data these sequences represent are varied and can be, for instance, time series, electronic signals or probability densities. The goal of smoothing is to remove as much of the noise as possible to achieve an estimation of the underlying signal. Smoothing makes two fundamental assumptions:

- The data represents a signal that has been contaminated by additive noise, i.e.  $data = signal + noise$ .
- The underlying signal is smooth.

The first assumption makes use of the concepts *signal* and *noise*. How do we distinguish between them? In some applications the characteristics of the signal are known beforehand, for example, in a time series that is known to represent seasonal fluctuations, the signal is known to be periodic; possibly sinusoidal. In other cases the true characteristics of the signal are unknown

---

<sup>1</sup>The name *LULU* is an abbreviation of *Lower Upper Lower Upper* which is a reference to the operators that form the basis of the theory.

and it is indeed the job of the smoother to reveal those characteristics. In these cases assumptions must be made about the noise component of the data, for instance, it might be assumed that the noise is sampled independently and identically from a Gaussian probability distribution. Knowing the character of the noise might enable us to extract or approximate the signal. The assumption that the underlying signal is “smooth” is, in a sense, an assumption about the character of the noise that has contaminated the data, and therefore makes it possible to extract the signal. It is often said that noise should contain as little structure as possible [24].

Smoothing can also be described as being a bridge between nonparametric and parametric methods, that is, methods that make no assumptions about the data, and methods that make strong assumptions about the data, i.e. that the data has been generated by some kind of mathematical model [74]. Smoothing sometimes tries to minimize the assumptions necessary to make sense of data. With no assumptions, no further analysis is possible, but by making the assumption that the underlying signal is “smooth”, it becomes possible to recover it approximately, given that the assumption is correct.

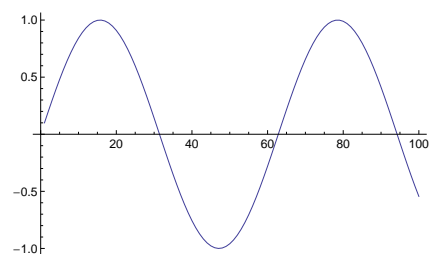
Smoothing is similar to curve-fitting and sometimes the line between the two is blurred, for example in regression-based smoothers. Fitting a curve to data implies an underlying model for the signal. Smoothing, on the other hand, need not make assumptions about an underlying model for the signal, but does sometimes assume a model for the noise contaminating it, allowing the data to “speak for itself”. This is probably the most important distinction between smoothing and other data analysis techniques: with smoothing the emphasis is on the assumptions about the noise contaminating a supposed signal, whereas with other data analysis techniques the emphasis is on the assumptions about the signal itself. Consequently it is possible for smoothing techniques to reveal unexpected structure, which is essential in data analysis [77, 24].

A possible caveat must be mentioned regarding the statistical soundness of smoothing procedures: a statistical model that takes the influence of noise into account may be able to extract more information from a noise-contaminated signal than the procedure of smoothing the data first and subsequently fitting a statistical model. Bayesian statistics might be better suited to this task; see [9] for a Bayesian point of view on smoothing.

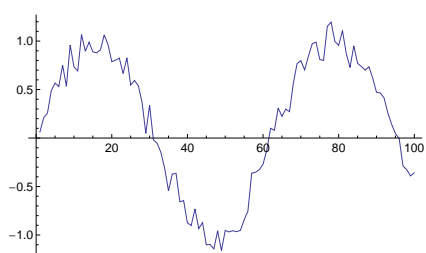
### 2.2.2 Operation

Noise can be characterized using probability distributions: for some noise sources the probability distribution of the output values of the source is known beforehand theoretically, whereas for other noise sources the probability distribution has to be inferred using histograms to approximate the underlying probability distribution [12]. The probability distribution of a noise source can then be used to classify it. In general there are two main types of noise: impulsive noise and non-impulsive noise. With impulsive noise the probability distribution of the noise source is heavy-tailed, so that noise that is rare and large in amplitude is encountered. The probability distributions of non-impulsive noise sources, on the other hand, are not heavy-tailed and drop off quite rapidly. Figure 2.2 illustrates the difference between the probability distributions of impulsive and non-impulsive noise sources.

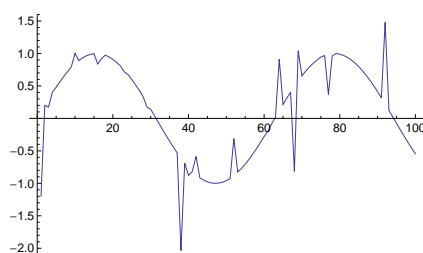




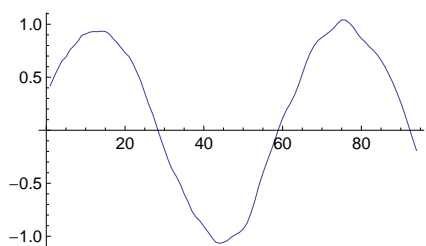
(a) Original signal.



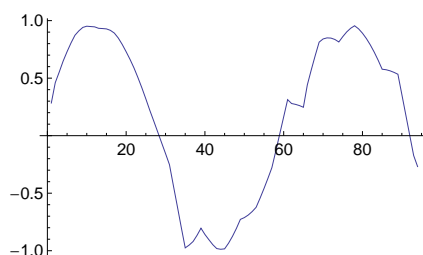
(b) Signal corrupted with non-impulsive noise.



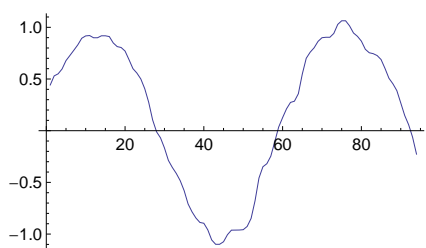
(c) Signal corrupted with impulsive noise.



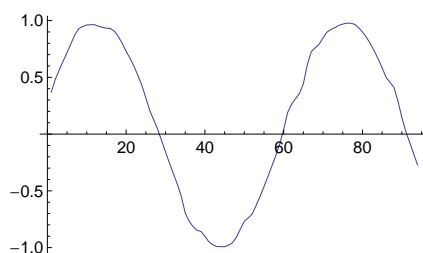
(d) Signal corrupted with non-impulsive noise smoothed with linear smoother.



(e) Signal corrupted with impulsive noise smoothed with linear smoother.



(f) Signal corrupted with non-impulsive noise smoothed with nonlinear smoother.



(g) Signal corrupted with impulsive noise smoothed with nonlinear smoother.

Figure 2.1: Smoothing two noise-corrupted signals with linear and nonlinear smoothers. Note that the underlying data is discrete; linear interpolation is used only for clarity of presentation.

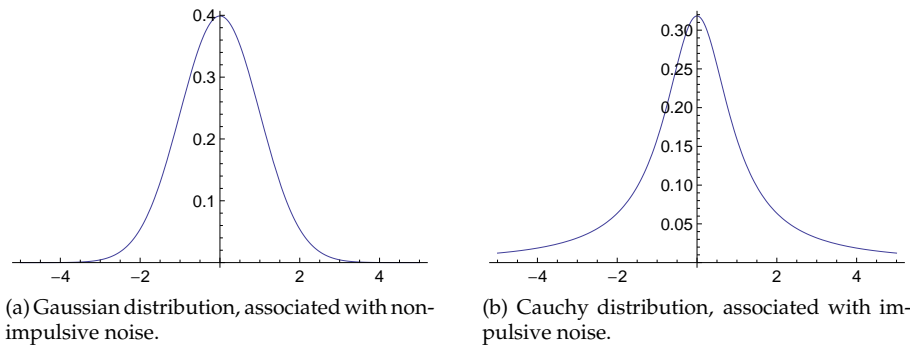


Figure 2.2: Comparison between the probability distributions of impulsive and non-impulsive noise sources.

To demonstrate, consider a vinyl record: the signal is the waveform of sound as it was recorded in the studio and the constant hiss in the background is non-impulsive noise; the occasional popping sound is impulsive noise. Smoothing can help us recover the original waveform, or at least an approximation thereof.

Smoothers usually operate by visiting each element in an input sequence in turn and considering the neighbours of that element to decide how that element should be modified, or smoothed. This is often called a “sliding window” approach, and the size of the window can vary, i.e. the distance to the element in question for the element to be considered a neighbour can vary. With the running average smoother, for example, each element is replaced by the average of the elements in the sliding window, whereas with the running median smoother each element is replaced by the median value of all the elements in the sliding window. In cases where the modified value is actually selected without modification from the elements of the sliding window the smoother is called an *order selector* [63].

### 2.2.3 Smoother Design

There are a few general criteria that are useful when it comes to the design and evaluation of a smoother. These criteria define what makes a “good” smoother — unfortunately some of them are difficult to measure, but remain useful as guiding principles.

#### 2.2.3.1 Effectiveness

A smoother is *effective* when it approximates the underlying signal very well for a given sequence [63].

This may be an impossible ideal in general, but it remains useful as a guiding principle, and embodies the goal of smoothing. One way to test the effectiveness of a smoother is to apply it to simulated data: known functions to which noise from known probability distributions have been added.

### 2.2.3.2 Robustness

A smoother is *robust*<sup>2</sup> if it can handle both impulsive and non-impulsive noise [78, 65, 63].

A robust smoother will remove impulsive noise and will not let it influence other data values in the sequence, or at least, try to minimize the effect.

Linear smoothers are not robust since an impulse is “smeared out” in the resulting sequence, leading to bias in the estimation of the signal. Nonlinear smoothers do not suffer from this restriction and it is possible for nonlinear smoothers to remove some cases of impulsive noise effectively. With the median smoother, for example, it is in some cases possible for a single impulse to propagate in such a way that it can influence values arbitrarily far away in the worst case. Therefore stability is an issue with both linear and nonlinear smoothers — the only difference being that with linear smoothers there is no remedy.

### 2.2.3.3 Consistency

A smoother is *consistent* if it recognises its own output as signal, and if it recognises its own residual as noise [63].

In a sense a consistent smoother defines signal to be the set of all its outputs, and it defines noise as the set of all its residuals.

With a consistent smoother it is redundant to apply the smoother more than once to the same sequence. Consistency is intimately related to idempotence and co-idempotence; these concepts are discussed in Section 2.2.7.

### 2.2.3.4 Efficiency

A smoother is *efficient* if it is economical to compute [63].

Economy, of course, is a relative concept: computations that are economical in one area might be prohibitively expensive in another. Consider the contrast between the capabilities of a personal computer and an integrated circuit that has to respond to changes in its environment in real-time. Therefore the area of application should be kept in mind when designing for economy in a smoother.

It might be argued that given the computational power of hardware today, and the speed at which it is growing, efficiency is actually no longer a requirement. There are two points in favour of efficiency though. The first is that the size of the data that must be routinely analyzed is growing just as fast as the computing capabilities, if not faster [35, 28] (see also Section 3.1). The second is energy efficiency: inefficient computations waste energy, which is becoming an expensive resource [35].

In addition to the considerations above, the benefits of being able to parallelize a smoothing algorithm should also be kept in mind.

## 2.2.4 Smoother Definitions

**Definition 2.1** (Sequence). *A sequence is a bi-infinite, ordered list of real numbers. Let  $S$  be the set of all sequences:*

---

<sup>2</sup>Also called *resistant* or *stable* [63].

$$\mathcal{S} = \langle x_i \rangle, \quad x_i \in \mathbb{R}, \quad i \in \mathbb{Z}$$

All sequences are assumed to be in  $\ell_1$ , which is defined as the set of all sequences  $\langle x_i \rangle$  such that:

$$\sum_{i=-\infty}^{\infty} |x_i| < \infty$$

Virtually all sequences found in practice are finite and can be made to be  $\ell_1$  sequences by appending and prepending zeros to the sequence. The above requirement therefore merely excludes pathological cases. For the purpose of this text we shall assume that a smoother is an operator that maps sequences onto sequences.

Addition of sequences and multiplication by scalars is defined as follows:

**Definition 2.2** (Addition of sequences).

$$x + y = \langle x_i + y_i \rangle, \quad x, y \in \mathcal{S}, \quad i \in \mathbb{Z}$$

**Definition 2.3** (Multiplication by scalar).

$$\alpha x = \langle \alpha x_i \rangle, \quad x \in \mathcal{S}, \quad \alpha \in \mathbb{R}, \quad i \in \mathbb{Z}$$

With the addition of these operations  $\mathcal{S}$  becomes a vector space [3].

It will be useful to introduce an order on the set of sequences  $\mathcal{S}$  in the following way:

**Definition 2.4** (Order on  $\mathcal{S}$ ).

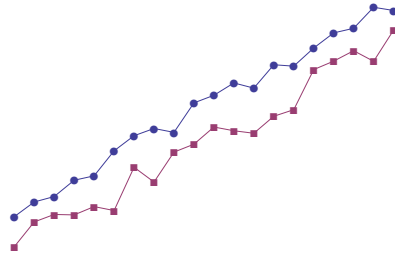
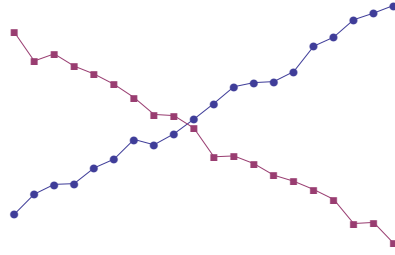
$$x \leq y \Leftrightarrow x_i \leq y_i, \quad i \in \mathbb{Z}$$

This introduces a partial order on  $\mathcal{S}$  since the axioms of reflexivity, anti-symmetry and transitivity are satisfied [6]. Note that this is not a total order since not all sequences can be compared to each other in this way; see Figure 2.3 for an illustration.

We find it convenient to add some definitions and axioms concerning operators on sequences.

**Definition 2.5.** Let  $\mathcal{O}$  be the set of operators on  $\mathcal{S}$ ;  $x \in \mathcal{S}$  and  $A, B \in \mathcal{O}$ . Then:

1.  $(A + B)x = Ax + Bx$
2.  $Ix = x$  ( $I$  is the identity operator)
3.  $\mathbf{0}x = \langle y_i \rangle, \quad y_i = 0, \quad i \in \mathbb{Z}$  ( $\mathbf{0}$  is the null operator)
4.  $(\alpha A)x = \alpha(Ax), \quad \alpha \in \mathbb{R}$
5.  $(AB)x = A(Bx)$
6.  $(Ex)_i = x_{i+1}, \quad i \in \mathbb{Z}$  ( $E$  is the shift-left operator)
7.  $Nx = -x$  ( $N$  is the negation operator)

(a) A case where  $x \leq y$ .(b) A case where the two sequences  $x$  and  $y$  cannot be compared using Definition 2.4.Figure 2.3: Illustration of the partial order on  $\mathcal{S}$ .

An important attribute of an operator is that of order preservation between two sequences. Operators with this attribute are called *syntone operators*<sup>3</sup> and are defined as follows [48, 63]:

**Definition 2.6.** *An operator  $A$  is syntone if*

$$x \leq y \Rightarrow Ax \leq Ay$$

Compositions of syntone operators are clearly also syntone:

$$x \leq y \Rightarrow ABx \leq ABy \quad \text{if } A, B \text{ are syntone}$$

### 2.2.5 Smoother Axioms

Most of the design criteria for smoothers (as defined in Section 2.2.3) can be replaced (or formalized) by the following axioms which appear in [63] and which have appeared previously in slightly different form<sup>4</sup> in [48].

**Definition 2.7** (Smoother axioms). *An operator  $S$  is a smoother if:*

1.  $SE = ES$  (Horizontal translation invariance)
2.  $S(x + c) = S(x) + c$  (Vertical translation invariance)
3.  $S(\alpha x) = \alpha S(x)$ ,  $\alpha \in \mathbb{R}$ ,  $\alpha \geq 0$  (Scale independence)

<sup>3</sup>These operators are also called isotone, monotone or order-preserving [63].

<sup>4</sup>Mallows [48] omits the restriction that  $\alpha \geq 0$  in the scale independence part of the axiom.

These axioms ensure that a smoother is translation invariant and scale independent. For a more restricted type of smoother, called a *separator*, the following axioms can be added:

**Definition 2.8** (Separator axioms). *A smoother  $S$  is a separator if:*

1.  $S^2 = S$  (*idempotence*)
2.  $(I - S)^2 = I - S$  (*co-idempotence*)

A separator can be thought of as a very consistent smoother since it removes noise and preserves signal without distortion. If a smoother is not a separator then the output from a smoother could be passed through the smoother again to possibly yield a better result; the result can also turn out to be worse.

### 2.2.6 Linearity and Nonlinearity

The operators in  $\mathcal{O}$  were defined to be right-distributive. It is important to note, however, that in general an operator  $A \in \mathcal{O}$  is not left-distributive. Only linear operators are left- and right-distributive in general, since  $A$  is linear, by definition, if:

$$A(\alpha x + \beta y) = \alpha Ax + \beta Ay$$

where  $\alpha, \beta \in \mathbb{R}$  and  $x, y \in \mathcal{S}$ .

### 2.2.7 Idempotence and Co-idempotence

The concepts of idempotence and co-idempotence were used to axiomatize the requirements for a separator (Section 2.2.5). It will be useful to discuss them here.

**Definition 2.9** (Idempotence). *An operator  $A \in \mathcal{O}$  is idempotent if and only if:*

$$A^2 = A$$

**Definition 2.10** (Co-idempotence). *An operator  $A \in \mathcal{O}$  is co-idempotent if and only if:*

$$(I - A)^2 = I - A$$

For a smoother to be very consistent, it must be able to confirm its own outputs; it has to be idempotent and co-idempotent. The distinction between idempotence and co-idempotence has been largely overlooked in the literature because of the emphasis on linear operators, for which idempotence and co-idempotence coincide. For nonlinear operators, however, this is not true in general. For example, the operator  $A$  defined as  $Ax = |x| = \{|x_i|\}$  is idempotent, but not co-idempotent, since  $(I - A)x = \{x_i - |x_i|\}$  but  $(I - A)^2x = 2(I - A)x \neq (I - A)x$ .

The following is a test for the co-idempotence of an operator:

**Theorem 2.1** (Test for co-idempotence). *An operator  $A$  is co-idempotent if and only if  $A(I - A) = \mathbf{0}$ .*

*Proof.*

$$I - A = (I - A)^2 = I - A - A + A^2 = I - A - A(I - A)$$

This is true if and only if  $A(I - A) = \mathbf{0}$ .

□

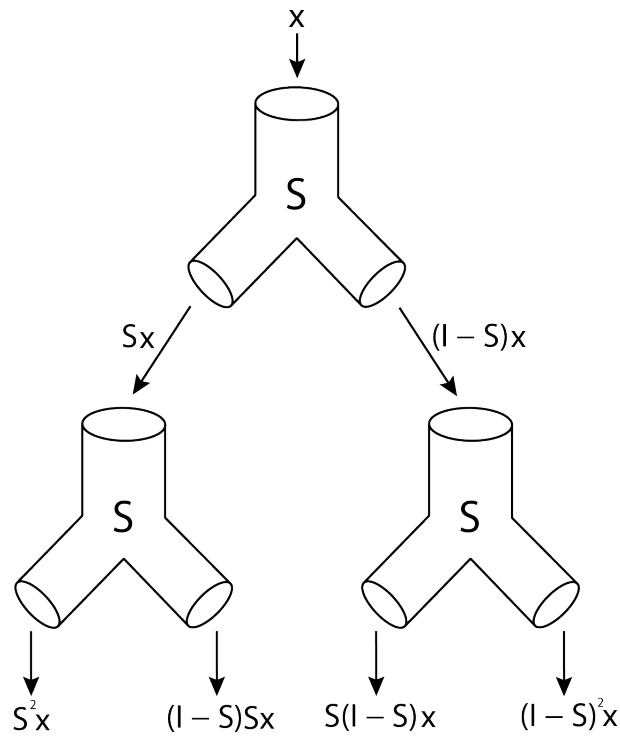


Figure 2.4: A two-stage separator cascade.

The importance of idempotence and co-idempotence can be illustrated with the following example (Figure 2.4). The original sequence  $x$  is smoothed by the smoother  $S$  and is separated into two sequences:  $s = Sx$ , the signal, and  $n = (I - S)x$ , the noise. These two sequences are then smoothed by  $S$  again. The whole process ultimately yields four outputs:

1.  $Ss = S^2x$
2.  $Sn = S(I - S)x$
3.  $(I - S)s = (I - S)Sx$
4.  $(I - S)n = (I - S)^2x$

If the smoother is idempotent, then  $S^2x = Sx$ , so that the smoother is *signal-consistent*; its definition of signal remains fixed. If the smoother is not idempotent, then it can be considered to be a deficient noise extractor. To yield

a better approximation of signal in this case, the result of  $Sx$  can be smoothed again, so that  $S^2x$  is the signal and  $(I - S)x + (I - S)Sx$  is the noise.

If the smoother is co-idempotent, then  $(I - S)^2x = (I - S)x$ , so that the smoother is *noise-consistent*; its definition of noise remains fixed. If the smoother is not co-idempotent, then it can be considered a deficient signal extractor. To yield a better approximation of noise in this case, the result of  $(I - S)x$  can be smoothed again, so that  $(I - P)^2x$  is the noise and  $Sx + S(I - S)x$  is the signal.

If the smoother is neither idempotent nor co-idempotent then a better approximation to the true signal and noise could be achieved by smoothing both the signal output  $Sx$  and the noise output  $(I - S)x$  of the smoother again. In this case the signal will be approximated by  $S^2x + S(I - S)x$  and the noise by  $(I - S)Sx + (I - S)^2x$ .

## 2.3 LULU Smoothers

The *LULU* smoothers form a class of nonlinear operators with a range of mathematical properties that make them predictable and useful as smoothers. Its closest relative is arguably the class of median smoothers; the two classes share some similarities, but in general a solid theory is lacking for median smoothers [78, 57, 66].

### 2.3.1 Definitions

The concept of a *pulse* will be useful throughout. A pulse is essentially just a finite constant region in a sequence surrounded by zeros. But first the concept of a *constant region* is needed:

**Definition 2.11** (Constant region). *A constant region in a sequence  $x \in \mathcal{S}$  is a finite sub-section  $\langle y_i \rangle$  of  $n$  identical real numbers with value  $v$  so that:*

$$x = \langle \dots, x_s, y_1, y_2, y_3, \dots, y_n, x_e, \dots \rangle$$

with  $y_i = v$ ,  $v \in \mathbb{R}$ ,  $v \neq x_s$ ,  $v \neq x_e$ .

A pulse can now be defined as a special case of a constant region as follows.

**Definition 2.12** (Pulse). *A pulse is a constant region with a value  $v \neq 0$  embedded into the zero sequence.*

*A pulse with a value  $v > 0$  is called an upward pulse, and a pulse with a value  $v < 0$  is called a downward pulse.*

**Definition 2.13** (Up/Down operators). *Let  $x \in \mathcal{S}$ . Then:*

$$\bigvee x = y, \quad y_i = \max\{x_i, x_{i+1}\}$$

$$\bigwedge x = y, \quad y_i = \min\{x_{i-1}, x_i\}$$

The action of the  $\bigvee$  and  $\bigwedge$  operators can be illustrated by considering how they act on a single pulse. The  $\bigvee$  operator widens an upward pulse of unit width to the left, whereas the  $\bigwedge$  operator removes the upward pulse completely. With an upward pulse of width 2 the action of the  $\bigvee$  operator



is the same as before; the  $\wedge$  operator, on the other hand, shrinks the original pulse down to a pulse of width 1, towards the right. Applying the  $\wedge$  operator again will remove the pulse completely, as before. In general then, the  $\vee^n$  operator widens a pulse by  $n$  towards the left and the  $\wedge^n$  operator shrinks a pulse by  $n$  towards the right.

Suppose now that we have an upwards pulse of width  $m$ . Applying the  $\vee^n$  operator with  $n < m$  will widen the pulse to a width of  $n + m$ . Subsequently applying the  $\wedge^n$  operator will reverse the effect of the  $\vee^n$  operator and return the pulse to its original width. The same applies to the situation where the order of the operators are reversed, as long as  $n < m$ . Note, however, that when  $n \geq m$ , the  $\wedge^n$  will destroy the pulse entirely so that there is no way for  $\vee^n$  to recreate it. The compositions  $\vee^n \wedge^n$  and  $\wedge^n \vee^n$  can thus be said to remove pulses with a width of at most  $n$ . The following definition is then expedient:

**Definition 2.14** ( $L_n$  and  $U_n$  operators). *Let  $n \in \mathbb{Z}$ ,  $n \geq 0$ . Then:*

$$L_n = \vee^n \wedge^n$$

$$U_n = \wedge^n \vee^n$$

**Definition 2.15** (Basic LULU operators). *The operators  $L_n$ ,  $U_n$ ,  $L_n U_n$  and  $U_n L_n$  will be called the basic LULU operators.*

The  $L_n$  operator will remove upward pulses of length at most  $n$ , and the  $U_n$  operator will remove downward blockpulses of length at most  $n$ . The  $L_n U_n$  and  $U_n L_n$  operators will remove both upward and downward pulses of length at most  $n$ . Note, however, that  $L_n U_n \neq U_n L_n$ .

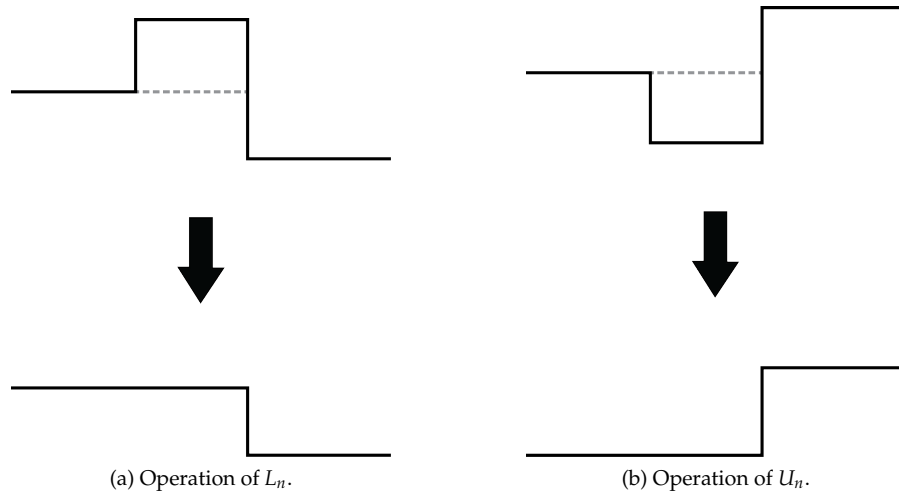
The actions of the basic LULU operators are actually more general than just the removal of upward or downward pulses: it can be shown how the basic LULU operators in fact act on *constant regions*, of which pulses are just a special case [43]. In general,  $L_n$  operates as follows:

- If a constant region is of length at most  $n$ , and;
- The value of the constant region is greater than the values of both its neighbouring constant regions, then;
- The value of the constant region is replaced with the greatest of the values of its neighbouring constant regions.

In the same vein,  $U_n$  operates as follows:

- If a constant region is of length at most  $n$ , and;
- The value of the constant region is less than the values of both its neighbouring constant regions, then;
- The value of the constant region is replaced with the lesser of the values of its neighbouring constant regions.

Figure 2.5 illustrates this operation of  $L_n$  and  $U_n$ .

Figure 2.5: The general operation of  $L_n$  and  $U_n$ .

**Definition 2.16** (*LULU operator*). Let  $n \in \mathbb{Z}$ ,  $n \geq 0$ . Any finite composition of the operators  $L_n$  and  $U_n$  will be called a LULU operator.

**Definition 2.17** ( $C_n$  and  $F_n$  operators).

$$C_n = L_n U_n C_{n-1} \quad \text{with} \quad C_0 = I$$

$$F_n = U_n L_n F_{n-1} \quad \text{with} \quad F_0 = I$$

The  $C_n$  and  $F_n$  operators systematically remove pulses of size 1 up to size  $n$ . These operators can be used as smoothers, in addition to the basic LULU operators. Figures 2.6 and 2.7 shows how a signal contaminated respectively with nonimpulsive and impulsive noise is smoothed by the  $F_n$  and  $C_n$  operators. Note the excellent performance of the  $C_n$  operator on the signal contaminated with impulsive noise (Figure 2.7).

## 2.3.2 Properties

### 2.3.2.1 Ordered Semigroup Structure

A fundamental structure in abstract algebra is the *group*, which is essentially a set on which a binary operation satisfying certain requirements has been defined [5]. A more general structure is the *semigroup*, which is defined as follows [38]:

**Definition 2.18** (Semigroup). A semigroup is a non-empty set  $G$  on which a binary operation  $(a, b) \rightarrow ab$  has been defined, which also satisfies the following properties:

**Closure** If  $a, b \in G$  then  $ab \in G$ .

**Associativity** If  $a, b, c \in G$  then  $(ab)c = a(bc)$ .

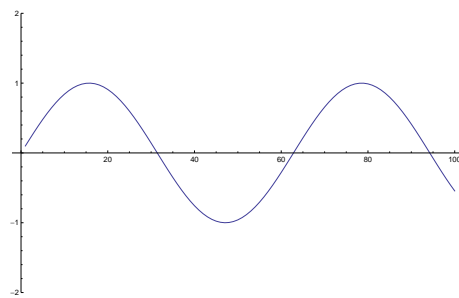
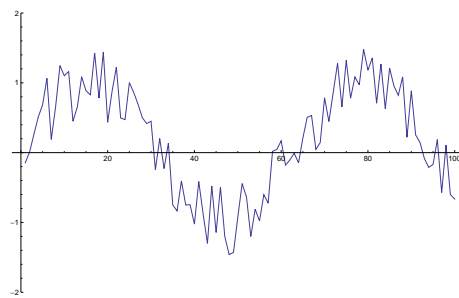
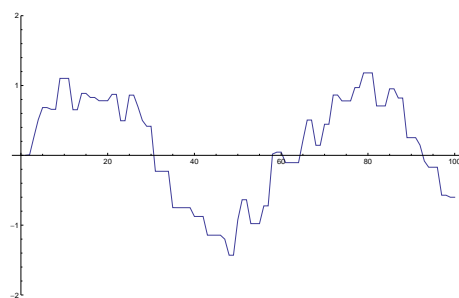
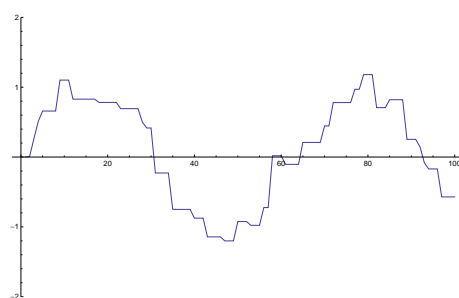
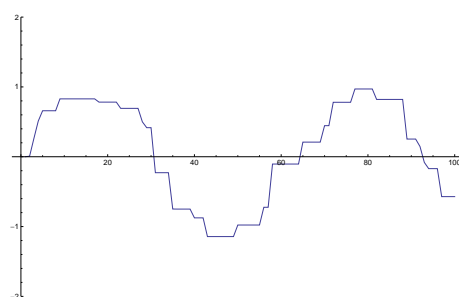
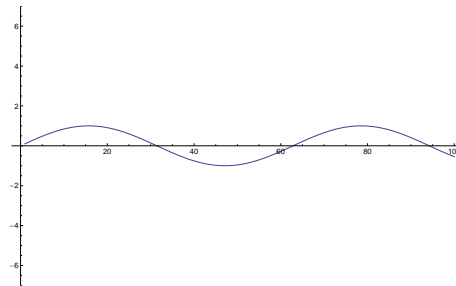
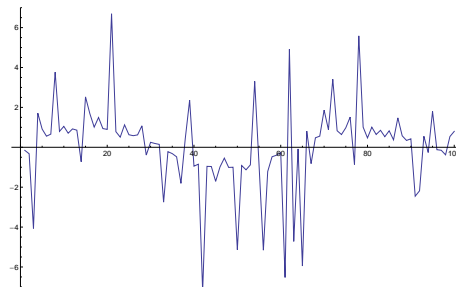
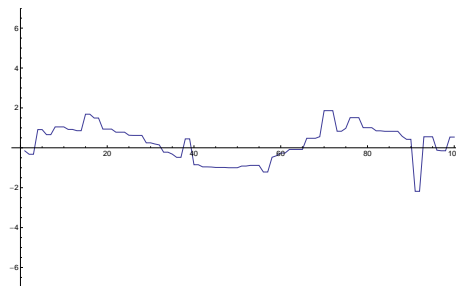
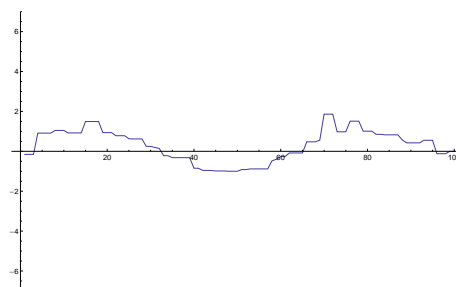
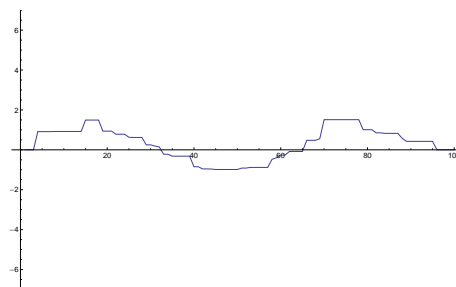
(a) Original signal,  $x$ .(b)  $y$ :  $x$  contaminated with nonimpulsive noise.(c)  $F_1 y$ .(d)  $F_2 y$ .(e)  $F_3 y$ .

Figure 2.6: Smoothing a signal contaminated with nonimpulsive noise with the  $F_n$  operator.

(a) Original signal,  $x$ .(b)  $y$ :  $x$  contaminated with impulsive noise.(c)  $C_1 y$ (d)  $C_2 y$ (e)  $C_3 y$ Figure 2.7: Smoothing a signal contaminated with impulsive noise with the  $C_n$  operator.

The basic *LULU* operators form a semigroup under operator multiplication<sup>5</sup> [63]. The multiplication table is shown in Table 2.1.

	$\mathbf{L}_n$	$\mathbf{U}_n$	$\mathbf{U}_n\mathbf{L}_n$	$\mathbf{L}_n\mathbf{U}_n$
$\mathbf{L}_n$	$L_n$	$L_nU_n$	$U_nL_n$	$L_nU_n$
$\mathbf{U}_n$	$U_nL_n$	$U_n$	$U_nL_n$	$L_nU_n$
$\mathbf{U}_n\mathbf{L}_n$	$U_nL_n$	$L_nU_n$	$U_nL_n$	$L_nU_n$
$\mathbf{L}_n\mathbf{U}_n$	$U_nL_n$	$L_nU_n$	$U_nL_n$	$L_nU_n$

Table 2.1: Multiplication table of the *LULU* semigroup.

There is also an order among the *LULU* operators for a given  $n$ , so that the *LULU* semigroup becomes an ordered semigroup. The order of the *LULU* operators is based fully on the following [63]:

$$L_n \leq U_nL_n \leq M_n \leq L_nU_n \leq U_n$$

where  $M_n$  is the well-known median smoother [77], defined as:

$$(M_n x)_i = \text{median}\{x_{i-n}, \dots, x_i, \dots, x_{i+n}\}$$

The composition of *LULU* operators  $\{L_n, U_n, L_k, U_k\}$ , where  $n \neq k$ , is not fully ordered; a useful partial ordering exists, however, based on the following [67]:

$$\dots \leq L_2 \leq L_1 \leq L_0 = I = U_0 \leq U_1 \leq U_2 \leq \dots$$

But, for instance,  $L_1U_1$  and  $L_2U_2$  are not comparable, so that they are not in a partial order. Figure 2.8 shows the complete order relations between the key *LULU* operators. It is complete in the sense that no further order relations between the *LULU* operators exist.

### 2.3.2.2 Local Monotonicity

The concept of smoothness has been formalized mathematically in many ways and in many settings. In real analysis, for example, a natural concept with which to gauge the smoothness of a function is the continuity of its derivatives [68]. For the discrete case of sequences there is a measure of smoothness called *n-monotonicity* [63].

**Definition 2.19** (*n-monotonicity*). A sequence  $x \in \mathcal{S}$  is *n-monotone* if

$$\{x_j, x_{j+1}, \dots, x_{j+n+1}\}$$

is monotone for every  $j \in \mathbb{Z}$ .

**Definition 2.20** (Set of *n-monotone* sequences). Let  $\mathcal{M}_n$  denote the set of all *n-monotone* sequences in  $\mathcal{S}$ .

According to this definition the set of all sequences is equivalent to the set of all 0-monotone sequences, since any two consecutive elements are automatically monotone, so that  $\mathcal{S} = \mathcal{M}_0 = \mathcal{M}$ .

<sup>5</sup>By trivially including the identity operator  $I$  as an *identity element* the basic *LULU* operators actually form a *monoid* [55].

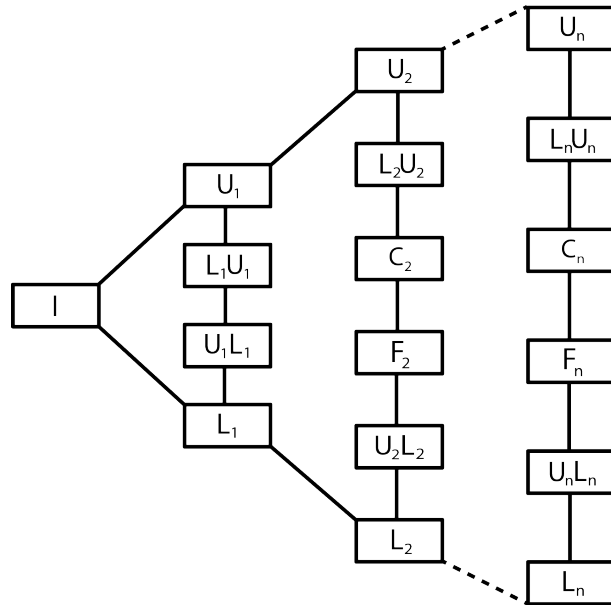


Figure 2.8: Mutual order relations between the key *LULU* operators (adapted from [67]).

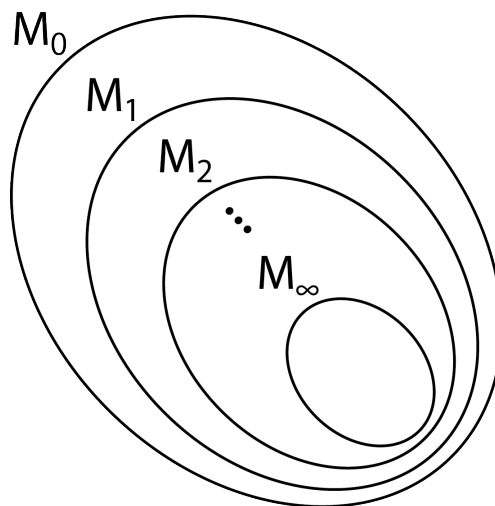


Figure 2.9: Nested subsets of  $\mathcal{M}$ .

### 2.3.2.3 Idempotence and Co-idempotence

Idempotence and co-idempotence formalize the notion of the consistency of a smoother, as discussed in Section 2.2.7.

The basic *LULU* operators are idempotent and co-idempotent. In fact polynomial time tests exist to determine whether so-called *stack filters* are idempotent and co-idempotent, the basic *LULU* operators being examples of such stack filters [67].

### 2.3.2.4 Variation Reduction and Preservation

The total variation of a sequence is another way to define the smoothness of sequences and is linked to the idea of local monotonicity.

**Definition 2.21** (Total variation). *The total variation  $T(x)$  of a sequence  $x \in \mathcal{S}$  is given by:*

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

The fact that the basic *LULU* operators map sequences into sets that are progressively more  $n$ -monotone suggests that the basic *LULU* operators must be variation reducing in some way. In fact, all compositions of the operators  $\vee$  and  $\wedge$  are variation reducing, so that as a special case all *LULU* operators are variation reducing [61, 63].

The basic *LULU* operators share a stronger variation-reducing property: they are all *variation preserving* in the following sense:

**Definition 2.22** (Variation preservation). *An operator  $A \in \mathcal{O}$  is variation preserving if:*

$$T(x) = T(Ax) + T(x - Ax)$$

### 2.3.2.5 Shape Preservation

**Definition 2.23** (Neighbour trend preservation). *An operator  $A$  is neighbour trend preserving if:*

$$x_i \leq x_{i+1} \Rightarrow A(x_i) \leq A(x_{i+1}) \quad \text{and} \quad x_i \geq x_{i+1} \Rightarrow A(x_i) \geq A(x_{i+1})$$

The operators  $U_n, L_n$  and all their compositions<sup>6</sup> are neighbour trend preserving (NTP), which means that these operators will never change the order between neighbours in a sequence.

The basic *LULU* operators also satisfy a stronger type of shape preservation called *full trend preservation*.

**Definition 2.24** (Full trend preservation). *An operator  $A$  is fully trend preserving if both  $A$  and  $(I - A)$  are neighbour trend preserving.*

---

<sup>6</sup>Compositions of NTP operators are also NTP [63].

### 2.3.2.6 Syntonecess

The concept of syntonecess was introduced in Section 2.2.4. The fundamental LULU operators  $\vee^n$  and  $\wedge^n$  are syntone, so that it follows that all of the LULU operators are syntone as well.

### 2.3.2.7 Signal/Noise Ambiguity

The smoothers  $L_n U_n$  and  $U_n L_n$  have differing definitions of noise. This is despite the fact that both map any sequence into  $\mathcal{M}_n$ . Both of these smoothers preserve sequence in  $\mathcal{M}_n$ , so that they agree on what class of functions are “signals”. But, given a sequence  $x$ , they may map  $x$  onto different signals. The component removed by smoothing with  $L_n U_n$ , which is  $(I - L_n U_n)x$ , is considered to be “noise” by  $L_n U_n$ .  $U_n L_n$  would remove  $(I - U_n L_n)x$  so that this is considered to be noise by  $U_n L_n$ .

By the co-idempotence of both operators,  $L_n U_n(I - L_n U_n)x = 0$  and  $U_n L_n(I - U_n L_n)x = 0$ .  $U_n L_n(I - L_n U_n)x$  is not necessarily equal to the zero sequence, however, nor is  $L_n U_n(I - U_n L_n)x$ . Thus these two operators have different interpretations of “noise”, even though  $L_n U_n$  agrees that  $U_n L_n x$  is pure “signal”, since  $L_n U_n(U_n L_n x) = U_n L_n x$ . Similarly  $U_n L_n(L_n U_n x) = L_n U_n x$ , so that  $U_n L_n$  agrees that  $L_n U_n x$  is a “signal”.

Consider Figure 2.10: the input sequence can be interpreted as a large pulse with a single smaller downwards pulse as noise, or as the zero sequence with two upward pulses as noise. With no other information these two interpretations are equivalent in the sense that both  $L_n U_n$  and  $U_n L_n$  separate the sequence into signal and noise consistently. This constitutes a *signal/noise ambiguity* between the two fundamental operators  $L_n U_n$  and  $U_n L_n$ , which form the basis of the  $C_n$  and  $F_n$  operators respectively.

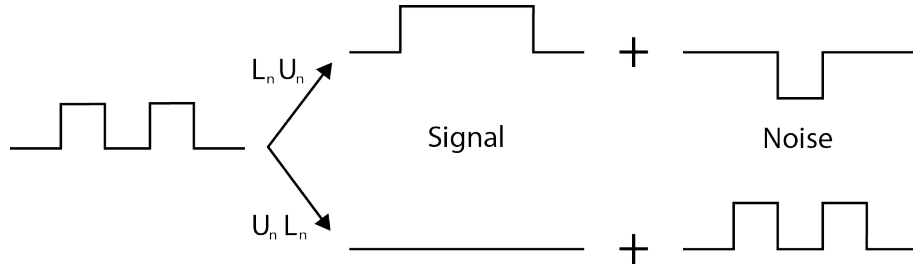


Figure 2.10: Signal/noise ambiguity with  $L_n U_n$  and  $U_n L_n$ .

### 2.3.2.8 Duality

**Definition 2.25** (Dual operators). *Let  $A, B \in \mathcal{O}$ . Then  $A$  is the dual of  $B$  if:*

$$AN = NB$$

The operators  $L_n$  and  $U_n$  are duals of each other [63] so that:

$$L_n(-x) = -U_n(x)$$



The chief importance of this duality is that it simplifies many proofs since if a property is proven for one of the two operators, it follows easily as a corollary that it holds for the other operator as well.

## 2.4 The Discrete Pulse Transform

### 2.4.1 Multiresolution Analysis

The overarching idea of decomposing a sequence into so-called “resolution levels” will be called Multiresolution Analysis (MRA)<sup>7</sup>. MRA has applications in, for example, signal analysis, image processing and image compression.

The key idea behind Multiresolution Analysis is contained in the word “resolution”. What does it mean? Resolution is associated with sharpness and detail; the greater the resolution of something, the more information it contains. It will be more instructive to focus on the concept of smoothness instead of resolution, since smoothness can be considered the opposite of resolution in a certain sense. The more resolution something has, the less smooth it is, and vice versa. MRA operates by successively smoothing a sequence and subtracting the output of a specific smoothing step from the input to that step, creating a *resolution level* that contains components of the input sequence that were not “smooth enough” according to the definition of smoothness in use. Each resolution layer and each smoothed layer is stored separately for analysis, creating a hierarchy of resolution levels and smoothed sequences. The MRA process is illustrated in Figure 2.13.

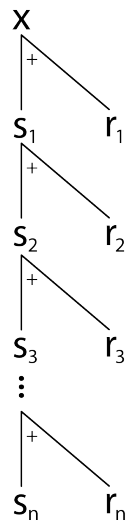


Figure 2.11: Multiresolution analysis of a sequence  $x$ , where  $s_i$  represents a smoothed level and  $r_i$  represents a resolution level.

<sup>7</sup>The term Multiresolution Analysis is used extensively in wavelet theory where it refers to a very specific construction consisting of a scaling function and a set of nested subspaces with certain special properties [25]. We think the meaning of the term too broad to be used solely for that purpose, and follow Bijaoui *et al.* [8] and Rohwer [63] in using it in a more general sense.

MRA processes are defined by the operator that is used to successively “smooth” the input. This operator can be either linear or nonlinear: a Discrete Wavelet Transform, for example, is a linear MRA technique, whereas a Discrete Pulse Transform is nonlinear.

## 2.4.2 Definitions

**Definition 2.26** (Discrete Pulse Transform). *Let  $x \in S$  and  $i \in \mathbb{Z}$ ,  $i \geq 1$ . The Discrete Pulse Transform transforms a sequence into two sets of sequences, the smooth set and the resolution set, using either the  $C_n$  or  $F_n$  operators. The smooth set consists of progressively smoothed sequences  $s_i$ . The resolution set consists of sequences  $r_i$  with progressively less resolution. The sequences are formed as follows:*

$$s_i = C_i x \quad \text{or} \quad s_i = F_i x$$

$$r_i = C_{i-1} x - C_i x \quad \text{or} \quad r_i = F_{i-1} x - F_i x$$

If a sequence consists only of non-negative numbers then it is also possible to use the  $L_n$  operator as a DPT operator in a way similar to that of the  $C_n$  and  $F_n$  operators [64]. We will make use of this fact in Chapter 4.

## 2.4.3 Properties

### 2.4.3.1 Structure

The DPT has a very rich structure that is amenable to concise description. Firstly, there is an upper bound on the number of levels in a DPT, given that the sequences under consideration have finite support.

**Theorem 2.2** (Upper bound on number of DPT levels). *Let  $N$  be the size of the support of a sequence  $x$ . Then there are a maximum of  $N$  DPT levels that are nonzero.*

Henceforth it will be assumed that there are  $N$  levels in a DPT decomposition, of which some may be zero.

The smoothed sequences  $s_i$  are all  $i$ -monotone, so that  $s_i \in \mathcal{M}_i$ . The resolution sequences  $r_i$  are very structured. These sequences are  $i-1$ -monotone so that  $r_i \in \mathcal{M}_{i-1}$  and they differ from the smoothed sequences in that they consist of pulses of width  $i$  that vary in magnitude; some are negative pulses and some are positive pulses.

Each resolution level can be thought of to consist out of two separate “signed” sequences: a positive resolution sequence  $r_i^+$  and a negative resolution sequence  $r_i^-$ , so that  $r_i = r_i^+ + r_i^-$ . Pulses in either the negative or positive resolution sequence are separated by at least  $i$  values, so that the distance between any two pulses in the signed resolution sequences is at least  $i$ . Moreover no pulses overlap in either  $r_i^-$ ,  $r_i^+$  or  $r_i$ .

### 2.4.3.2 Consistency

The most important property of the DPT is its *consistency*<sup>8</sup>, which is defined as follows. Suppose the resolution sequences are modified in some way, and

<sup>8</sup>Not to be confused with the consistency of a smoother, as discussed in Section 2.2.3.3.

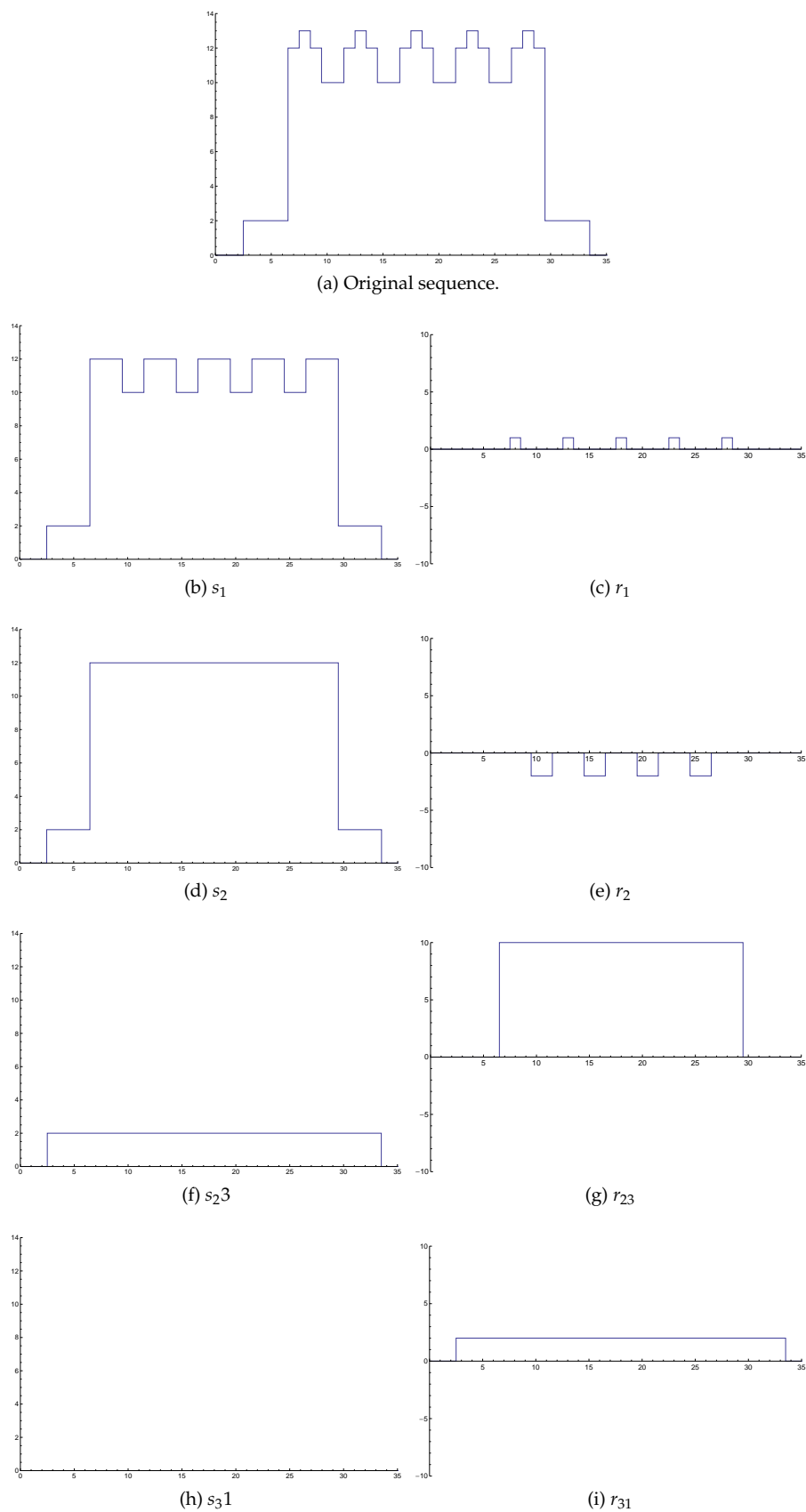


Figure 2.12: Demonstration of a DPT using the  $C_n$  operator. Note that only non-empty levels are shown.

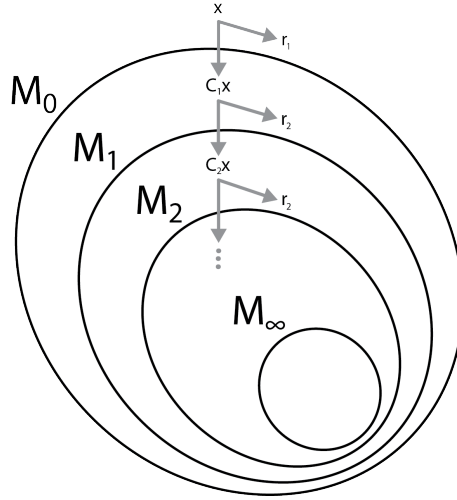


Figure 2.13: DPT output monotonicities.

that the DPT is taken of the sum of these modified resolution sequences. If the resolution sequences of this new DPT are the same as the modified resolution sequences, then the DPT is *consistent* for that particular modification.

The most basic consistency is where no modification has been made:

**Theorem 2.3** (Primary consistency). *Let  $r_i$  be the DPT resolution sequences of a sequence  $x$ . Then:*

$$\sum_{i=1}^{\infty} r_i = x$$

Historically there have been a succession of theorems proving stronger and stronger consistencies in the DPT, motivated by numerical evidence and culminating in the so-called *highlighting theorem* which subsumes all other consistency results.

**Theorem 2.4** (Highlighting theorem). *Let  $\pi_i$  be the individual pulses making up the DPT resolution sequences of a sequence  $x$ , and let  $\rho_i$  be the individual pulses making up the DPT resolution sequences of the following sequence:*

$$\sum_{i=1}^N \alpha_i \pi_i \quad \alpha_i \in \mathbb{R}, \alpha_i \geq 0$$

where  $N$  is the total number of pulses in the resolution sequences  $r_i$ . Then:

$$\rho_i = \alpha_i \pi_i$$

This theorem states that individual pulses can be scaled by non-negative real numbers and be recovered in a subsequent DPT. This theorem is general enough to include a large number of cases: for example, scaling an entire resolution level by a non-negative number will result in a consistent DPT.

### 2.4.3.3 Preservation and Reduction of Total Variation

The total variation of the resolution sequences of a DPT has two important attributes. The first is that the DPT is variation preserving in the following sense:

**Theorem 2.5** (DPT variation preservation). *Let  $r_i$  be the DPT resolution sequences of a sequence  $x$ . Then:*

$$\sum_{i=1}^{\infty} T(r_i) = T(x)$$

The second is that there is a power law relationship between the total variation of a smoothed sequence and the level of that smoothed sequence:

**Theorem 2.6** (Total variation decay [39]). *Let  $s_i$  be the smoothed sequences of a DPT of a sequence  $x$  where each  $x_i$  is independent and identically distributed according to some probability distribution with finite mean and variance. Then:*

$$T(s_i) \propto i^{-k} \quad k \in \mathbb{R}$$

*with  $k$  dependent on the probability distribution.*

For a sequence consisting of independent and identically distributed elements from the uniform distribution, it has been found empirically that  $k \approx 2.5$ .

An obvious and useful way to visualize the total variation decay is the *variation spectrum*, which plots the total variation of the resolution levels. Figure 2.14 shows a typical variation spectrum of a chosen random sequence.

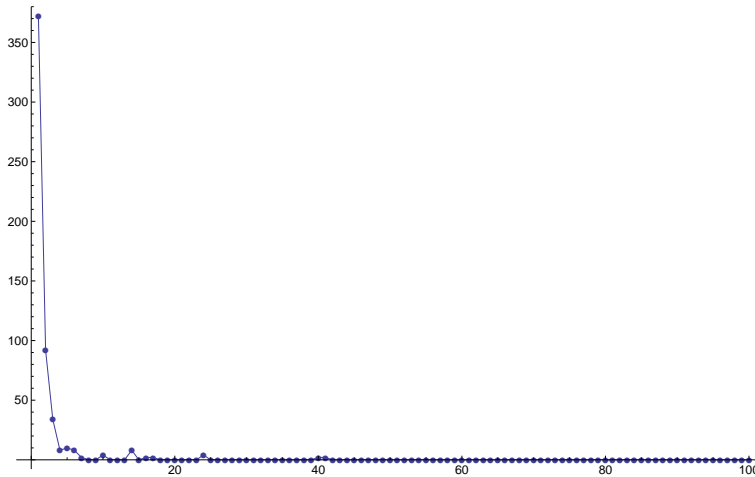


Figure 2.14: Variation spectrum of the DPT of a typical random sequence of length 100.

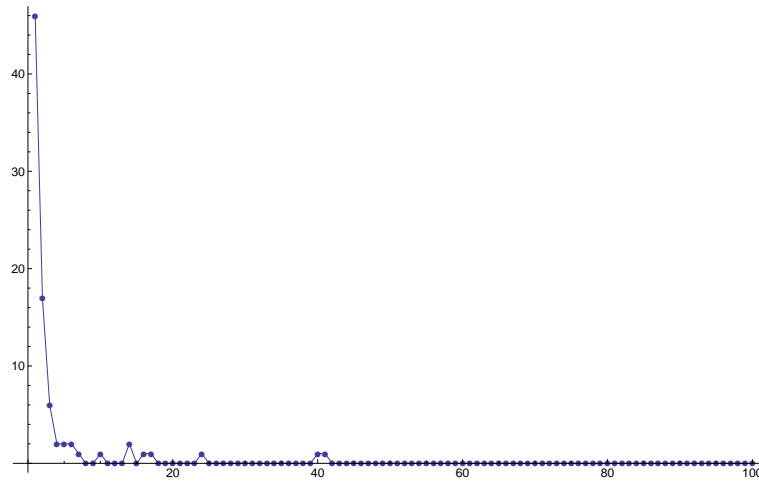


Figure 2.15: Presence spectrum of the DPT of a typical random sequence of length 100.

#### 2.4.3.4 Total Pulse Count

**Theorem 2.7** (Bounds on DPT pulse count). *Let  $\pi_T$  be the total number of pulses in all resolution levels of the DPT of a sequence with length  $N$ . Then:*

$$0 \leq \pi_T \leq N$$

Hence there can never be more pulses in a DPT than the length of the input sequence.

The *presence spectrum* is a plot of the number of pulses in a resolution level against the pulse width of that resolution level. The presence spectrum makes the allocation of the number of resolution pulses to different resolution levels explicit, and is useful to characterize the sequence at hand [20]. Figure 2.15 shows the presence spectrum of a typical random sequence. This shape is independent of the distribution

#### 2.4.3.5 Shape Preservation

The DPT inherits a number of shape preservation properties with regards to the smoothed sequences from the *LULU* smoothers used in its construction, the most important being *full trend preservation*, which is a stronger form of neighbour trend preservation, discussed in Section 2.3.2.5.

More relevant in the DPT is a shape preservation property in the resolution sequences, which makes it ideal for applications such as image processing. Any edge in the original sequence is reflected in the resolution sequences, and vice versa, so that edges in the resolution sequences are also preserved in a partial reconstruction of the original sequence.

To illustrate the importance of this fact, a comparison with the Discrete Wavelet Transform (DWT) is in order. Since the DWT is linear it does not respond to discontinuities very well, for example, a single impulse, or an edge,

is smeared and some of its energy is dispersed to its neighbours, destroying some of the signal and resulting in a loss of detail [22]. Since the DPT is nonlinear it does not suffer from this congenital defect and the property of shape preservation ensures that edges and impulses are preserved, even in partial reconstructions.

#### 2.4.3.6 Commutation with Quantization and Thresholding Operators

The quantization and thresholding operators frequently appear in signal processing. They are defined as follows:

**Definition 2.27** (Quantization operator).

$$Q_h(x)_i = h \operatorname{sgn}(x_i) \operatorname{int}\left(\frac{|x_i|}{h} + \frac{1}{2}\right)$$

**Definition 2.28** (Thresholding operator).

$$T_t(x)_i = \begin{cases} x_i & \text{if } |x_i| \leq t \\ \operatorname{sgn}(x_i)t & \text{if } |x_i| > t \end{cases}$$

Both the quantization and thresholding operator commute conveniently with all compositions of  $L_n$  and  $U_n$ .

#### 2.4.4 Extension to Higher Dimensions

It is possible to generalize the one-dimensional *LULU* operators to  $n$  dimensions [21, 2, 1]. Broadly speaking, this is done by using standard ideas in Mathematical Morphology. The concept of a pulse is generalized: in  $n$  dimensions a pulse becomes a more general connected region with a single value. Such a pulse is then removed by the generalized  $L_n$  operator if and only if it is greater in value than all of its neighbouring pulses and, conversely, pulses that are smaller in value than all of its neighbouring pulses are removed by the generalized  $U_n$  operator.

In one dimension some generalized *LULU* operators yield exactly the same results as the original *LULU* operators, and they also have a large number of their properties, for example: operator duality, an identical *LULU* semigroup structure, full trend preservation and total variation preservation.

A generalized  $n$ -dimensional Discrete Pulse Transform can also be defined using the generalized *LULU* operators. This generalized DPT also has some properties of its one-dimensional counterpart; of particular note is the fact that the decomposition is also consistent and total variation preserving. The Highlighting Theorem (Theorem 2.4) has also been proven to hold in the  $n$ -dimensional case [42].

The two-dimensional case of the generalized DPT holds considerable promise for image processing and compression. Research is ongoing regarding the applications. In this text, however, we restrict ourselves to the one-dimensional DPT, hoping to obtain some understanding of the benefits and limitations of using the one-dimensional DPT to compress digital images.

## 2.5 Conclusion

*LULU* theory provides a set of nonlinear operators that can be used as smoothers for use in data analysis. These smoothers are particularly good at removing impulsive noise and in many cases they can replace the median smoother, which is currently arguably the most popular smoother for removing impulsive noise.

More significantly for our purposes, however, *LULU* theory also provides the Discrete Pulse Transform, which is a nonlinear transform analogous to the Discrete Wavelet Transform. The DPT has a rich structure, despite being nonlinear in nature, and it is this rich structure that compels us to explore its potential use in the field of image compression — particularly appealing are its properties of consistency and shape preservation, which should be significant when it comes to designing a lossy image compressor.



## Chapter 3 - Data Compression

### 3.1 Introduction

Information is becoming more ubiquitous every day and the capability to store and deliver it in a timely manner is becoming essential. Since both storage and communication technology have limited capacities, it is necessary to use them in an efficient manner. This is where data compression plays an important role — it reduces the amount of data that has to be stored and transmitted without sacrificing the information contained in the original data, as with lossless compression, or, in the case of lossy compression, achieving an efficient compromise between data and information.

As an example of modern data storage, processing and transmission requirements, consider the Large Hadron Collider (LHC) computing grid operated by CERN [13]. During operation it produces 37 terabytes of data per day, where 27 terabytes of that data is raw data, and the remaining 10 terabytes is so-called “event-summary” data which is the output of processing done at the data center on-site at LHC. This data is then sent via dedicated networks operating at 10 gigabytes per second to institutions around the world for further analysis. In all, the LHC computing grid produces around 10 petabytes of data per year during operation.

Another example is that of the Mars Reconnaissance Orbiter (MRO), where data compression plays an explicit role [51]. The MRO is a satellite orbiting Mars at an altitude of 300 km equipped with a high-resolution imaging component called HiRISE (High Resolution Imaging Science Experiment) that takes photos of the Martian surface from orbit. The maximum size of an individual photo is  $20000 \times 63780$  pixels with a bit depth of 24 bits. Such a photo requires around 2 gigabytes of storage without any compression. The HiRISE system, however, contains an on-board dedicated image compression component that uses the FELICS image compression algorithm to compress the images losslessly and store them in the main memory of HiRISE which has a capacity of about 3.5 gigabytes. These compressed images are then transmitted to earth where they are decoded, processed and ultimately released to the scientific community and the public at large as losslessly compressed JPEG2000 images, typically achieving a 3:1 compression ratio.

### 3.2 Information theory

The concept of information can be defined in many ways, with varying degrees of accuracy, depending on the domain in which it is used. The theory of information we will be using in this text is based on a mathematical definition

of information, and is fundamentally probabilistic. It made its first appearance in a seminal paper [71] by Claude Shannon published in 1948.

Throughout this section we will make extensive use of the concept of an *ensemble*, defined as follows (adapted from [47]):

**Definition 3.1** (Ensemble [47]). *An ensemble  $\chi$  is a triple  $(x, \mathcal{A}_x, \mathcal{P}_x)$ , where the outcome  $x$  is the value of a random variable, which takes on one of a set of possible values from an alphabet  $\mathcal{A}_x = \{a_1, a_2, \dots, a_i, \dots, a_1\}$ , having probabilities  $\mathcal{P}_x = \{p_1, p_2, \dots, p_1\}$ , with  $p(x = a_i) = p_i$ ,  $p_i \geq 0$  and  $\sum_{a_i \in \mathcal{A}_x} p(x = a_i) = 1$ .*

### 3.2.1 Information content

**Definition 3.2** (Information content [47]). *Let  $e_i$  be events from some sample space  $E$ . Let  $p_i$  be the probability associated with event  $e_i$ . The information content of an event  $e_i$  is defined as:*

$$h(e_i) := \log_2 \frac{1}{p_i}$$

In the definition above the base of the logarithm is 2, and the resulting unit of information content is called the *bit*. Other bases can also be used, with a corresponding change in the unit of information, for example, when base  $e$  (Euler's constant) is used, the information content is measured in *nats*, and when the base is 10, the information content is measured in *bans*. Base 2 is customarily used because of the fact that binary numbers are used to store information (or rather, data) in modern computers, and because Shannon used it in the original paper on information theory.

While this is a purely mathematical definition, and may seem arbitrary, it does make sense intuitively. To see why, suppose that an event  $\alpha$  has a very high probability, say  $p(\alpha) = 0.999$ . What the information content definition tells us is that the amount of "information" that is received or transmitted when  $\alpha$  actually occurs, is very low:  $h(\alpha) = \log_2 \frac{1}{0.999} \approx 0.00144$  bits. Intuitively, this makes sense: if a high probability event occurs it doesn't tell us much; we have not really learnt something new about the state of affairs because we have been expecting that event — it did not surprise us.

In contrast, consider an event  $\beta$  with a very low probability of occurring:  $p(\beta) = 0.001$ . When this event actually occurs, it conveys the following amount of information according to Definition 3.2:  $h(\beta) = \log_2 \frac{1}{0.001} = 9.97$  bits, which eclipses the information content of event  $\alpha$ . Once again, this is in accord with the intuitive notion of information: a low probability event actually occurring tells us more about the state of affairs than a high probability event. We might even choose to revise our probabilities given the new evidence.

It seems as if the only requirement so far for the information content of an event has been that it is inversely proportional to the probability of the event. Why not use a simple inverse then — why specifically use a logarithm? The reason for this is that there is another aspect of information that we would like to capture mathematically. This additional aspect is the amount of information conveyed by independent events. Suppose  $\lambda$  and  $\xi$  are two independent events, i.e. that  $p(\lambda \xi) = p(\lambda)p(\xi)$ . It seems reasonable that the amount of information conveyed when these two events occur together is

simply the sum of the two separate information contents. A simple inverse will not be sufficient to model this aspect, since:

$$\frac{1}{p_1 p_2} \neq \frac{1}{p_1} + \frac{1}{p_2}$$

Using a logarithm, however, leads to the desired result:

$$h(\lambda \xi) = \log_2 \frac{1}{p(\lambda \xi)} = \log_2 \frac{1}{p(\lambda) p(\xi)} = \log_2 \frac{1}{p(\lambda)} + \log_2 \frac{1}{p(\xi)} = h(\lambda) + h(\xi)$$

### 3.2.2 Entropy

**Definition 3.3** (Entropy [47]). *The entropy  $H$  of a random variable  $X$  is defined as:*

$$H(X) := \sum_x p(x) \log_2 \frac{1}{p(x)}$$

The entropy of a random variable is simply the average information content of the outcomes of that random variable, so that entropy can also be seen as a measure of the amount of uncertainty about the outcomes of a random variable  $X$ . For example, suppose an event  $\lambda$  has probability  $p(\lambda) = 1$  of occurring so that it is certain that this event will occur. If the entropy for such a distribution is calculated, it will turn out to be zero. The entropy, in this case, can be interpreted as saying that there is no uncertainty concerning this random variable, since the outcome will always be  $\lambda$ . The entropy for any other random variable that does not attain unity for some event will be nonzero and positive. Intuitively, the state of maximal uncertainty is when all events are equiprobable, which corresponds to the uniform distribution. Mathematically, entropy does attain a maximum for any sample space when the probability distribution over that space is uniform. Entropy is in many ways the more fundamental information-theoretic concept, and forms the basis for many other subsequent theorems, definitions and concepts.

### 3.2.3 Relative entropy

**Definition 3.4** (Relative entropy [47]).

$$D_{KL}(P||Q) := \sum_x p(x) \log_2 \frac{p(x)}{q(x)}$$

Relative entropy<sup>1</sup> is a measure of how much two probability distributions “differ” from each other. Like ordinary entropy, relative entropy is measured in bits. Relative entropy is not symmetric, so that  $p \neq q \Rightarrow D_{KL}(p||q) \neq D_{KL}(q||p)$ , with the consequence that relative entropy cannot be interpreted as a distance metric between probability distributions.

Relative entropy can be understood as giving the amount of information that is gained when changing from one distribution  $q$  to another,  $p$ , where both distributions are attempts to model the same situation.

<sup>1</sup>Relative entropy is also known as *Kullback-Leibler divergence*.

**Theorem 3.1** (Gibbs' inequality [47]). *Relative entropy satisfies the following inequality:*

$$D_{KL}(p||q) \geq 0$$

*with equality if and only if  $p = q$ .*

### 3.3 Data modelling

Any real world data can be represented digitally, that is, its representation can be encoded as a binary string. The fidelity of the representation can be brought arbitrarily close to the original by merely increasing the amount of bits used to store the data. Consider a piece of music. Fundamentally it consists out of a continuous, time-varying signal. Continuity cannot be duplicated identically as a digital signal, but it is possible to come arbitrarily close to it. For example, the ubiquitous Compact Disc Digital Audio (CDDA) standard approximates a continuous sound signal by taking 44100, 16-bit stereo samples per second. The Digital Versatile Disc Audio (DVD-A) standard is capable of increasing the sample rate to a maximum of 192 KHz and the bit rate to 24-bit. The same is true for images. The resolution can be increased arbitrarily and the accuracy of the individual samples can also increase without limit. Of course, for inherently discrete data, no such approximation is necessary, as with, for example, textual data.

The process of choosing a representation for a data source is called *data modelling*. For a given data source, many different models are possible, and the best model will depend on the purpose or application of the data in question. If an exact representation of the data is not needed, then a model can be used where some of the data is thrown away — this is called *lossy compression*. Consider the MPEG-1 Audio Layer 3 (MP3) audio standard: exact fidelity (quantitative fidelity) is not desired, but qualitative fidelity is: to a human, there must be no noticeable difference between the original sound and the reconstructed sound. Such a model must then take into account how sound is perceived by humans.

These two modes of compression, lossless and lossy compression, will now be discussed in turn.

#### 3.3.1 Lossless and lossy data compression

Lossless compression preserves the original signal exactly — no information is lost in the compression process.

There are many areas where information loss cannot be tolerated, as an example, consider a piece of text. Any alteration of the text will most likely alter the meaning of the text — and this defeats the purpose of storing text. Another example is medical (or military) imaging, where the arbitrary omission or addition of data can be life-threatening. In addition, any data that undergoes further processing will not be amenable to lossy compression, because the distortion introduced by the lossy compression might be amplified to a significant extent.

Lossless compression cannot achieve the high compression ratios of lossy compression, so that the compression ratio is limited by the redundancy of the data itself.

It is easier to measure the efficacy of lossless compression schemes, since the only relevant measure is the compression ratio, which is unlike lossy compression, where human perception is often the final judge of quality, alongside the purely quantitative measure of compression ratio.

The way information is discarded in lossy compression will of course depend on the application. In general, though, the consumer of such lossy-compressed data is human, so certain facts about human perception must be taken into account when lossy schemes are designed.

## 3.4 Coding

### 3.4.1 Overview

Coding<sup>2</sup> is the process of mapping between the symbols of two alphabets, the source alphabet and the code alphabet. In the case of lossless compression, coding is applied to exploit the redundancies of the source string so that the size of the encoded string is smaller than the original string. The two most widely used coding techniques will be discussed here: symbol codes and stream codes.

### 3.4.2 Symbol codes

The simplest symbol codes assign one codeword to each symbol of a source string. More complex symbol codes are able to encode more than one source symbol with a single codeword — these are the so-called block encoders, and while they theoretically achieve greater compression, they are not practical enough to be of use in most problems.

Symbol codes are lossless codes, i.e. no information is lost in the encoding of a source string. This has two implications, the first being that any encoded string must be uniquely decodable, i.e. there must be a one-to-one mapping between a source symbol and a codeword for that symbol. The second implication is that not all source strings can be compressed. If a symbol code achieves any compression at all on some source strings, it must necessarily make the coded strings for other source strings longer. To achieve compression, the codewords of some source symbols have to be shorter than those symbols themselves. This implies that some codewords have to be longer than their source symbols. Consider a source string consisting out of a sequence of only one symbol: the symbol with the longest codeword. This string will not be compressed, but rather, it will be enlarged. The aim of lossless compression is to assign as short as possible codes to the source strings that will most likely appear. This is done through statistical modelling of the source.

Symbol codes achieve compression by assigning shorter codewords to symbols that have a high probability of occurring, and conversely, assigning longer codewords to symbols with a low probability of occurring. The goal is to minimize the *expected length* of a coded string.

---

<sup>2</sup>The word “coding” as it is used in this text should not be confused with channel coding, where the aim is to achieve reliable communication over a noisy channel.

**Definition 3.5** (Expected length of symbol code [47]). Let  $L_C(s)$  denote the length of an encoded source string  $s$  under a symbol coding scheme, or code,  $C$ . Then the expected length,  $\overline{L}_C$ , of such an encoded string is:

$$\overline{L}_C = \sum_{s \in S} p(s)L_C(s)$$

where  $p(s)$  is the probability of the symbol  $s$  appearing, and  $S$  is the source alphabet.

For a symbol code to be practical it should not be overly complex to encode and decode. A reasonable requirement for a symbol code is that it must be able to be decoded sequentially, with minimal context. The worst case here would be a string that cannot be decoded until the whole string has arrived or has been processed in some way. This requires extra memory and processing power, with a corresponding penalty in speed. One might be concerned as to the consequences of limiting our codes to these sequential codes — is it not possible that some non-sequential codes achieve greater compression? The answer here is, fortunately, no: there is a class of sequential symbol codes that are, in fact, optimal. They are called *prefix-free codes*.

### 3.4.2.1 Prefix-free codes

The codewords of prefix-free codes are easy to distinguish from one another since no codeword is a prefix for any other codeword. Consider the following case, where there is a code with two codewords, the one being 101 and the other 10. After receiving two code symbols 10, we are unsure of whether the codeword 10 was meant, or whether the codeword 101 is in fact being transmitted. It is only after receiving additional code symbols (one, in this case) that we can be sure of what the intended codeword was. This uncertainty is caused by the fact that the 10 codeword is a prefix for the 101 codeword. If no such prefixes existed, we could decode a codeword as soon as it can be matched to a codeword in the code. It is for this reason that prefix-free codes are also called self-punctuating codes<sup>3</sup>. We will henceforth restrict our attention to prefix-free codes.

### 3.4.2.2 The Kraft inequality

The requirement of unique decodability imposes a certain limit on the collection of codeword lengths,  $l_i$ . This seems intuitive — for example, if the codewords 0 and 1 have been chosen, then there can be no other codewords, since any other codeword will be confused with sequences of either one of the first two codewords. In this case the limit on the codeword lengths is codewords of length larger than one cannot exist. Consider another code, with codewords 0, 01, 10 and 11. In this case the string 0110 is not uniquely decodable. It will become uniquely decodable if one of the length two codewords are lengthened to have a size of 3 — thus unique decodability imposes a limit on the lengths of codewords in a uniquely decodable code.

The *Kraft inequality*<sup>4</sup> articulates this constraint on the codeword lengths of uniquely decodable codes as follows:

<sup>3</sup>Prefix-free codes are also confusingly called *prefix codes* in the literature.

<sup>4</sup>The Kraft inequality is also known as the *Kraft-McMillan inequality*.

**Theorem 3.2** (Uniquely decodable codes satisfy the Kraft inequality [47]). *For any uniquely decodable code over the binary alphabet  $\{0, 1\}$ , the codeword lengths  $l_i$  must satisfy:*

$$\sum_i 2^{-l_i} \leq 1$$

A code with codeword lengths that satisfy the Kraft inequality with equality is called a *complete code*.

The theorem above states necessary conditions for a uniquely decodable code. Another theorem guarantees the existence of a uniquely decodable code with codeword lengths  $l_i$  if those lengths satisfy the Kraft inequality:

**Theorem 3.3** (Uniquely decodable codes exist for codeword lengths satisfying the Kraft inequality [47]). *Given a set of positive integers  $l_i$  that satisfy the Kraft inequality*

$$\sum_i 2^{-l_i} \leq 1$$

*there exists a uniquely decodable code with codeword lengths equal to  $l_i$ .*

### 3.4.2.3 Characterising the average codeword length

The goal of symbol coding is to minimize the expected length  $\overline{L}_C$  of a codeword. What are the theoretical bounds on  $\overline{L}_C$ ?

**Theorem 3.4** (Bounds on compression with symbol codes [70]). *For an ensemble  $X$  there exists a prefix code  $C$  with average codeword length  $\overline{L}_C$  satisfying:*

$$H(X) \leq \overline{L}_C < H(X) + 1$$

The lower bound of compressibility for symbol codes is the entropy of the source,  $H(X)$ . This lower bound can only be achieved if the codeword lengths  $l_i$  are equal to the information content of a symbol:

$$l_i = \log_2 p_i^{-1}$$

Any set of codeword lengths that differ from the information contents as given above will be suboptimal, but in practice, unavoidable. For a given set of codeword lengths  $l_i$ , a set of *implicit probabilities* are defined — probabilities for which the code with those codeword lengths would have been optimal, according to the definition above.

**Definition 3.6** (Implicit probabilities of a code [47]). *Given a code  $C$  with codeword lengths  $l_i$ , the implicit probabilities of the source symbols induced by the code is given by:*

$$q_i = 2^{-l_i}$$

These implicit probabilities can now be used to determine sharper bounds on the average length of a codeword, and hence, on the compression achievable with symbol codes.

**Theorem 3.5** (Expression for the average codeword length [47]). *Given the probabilities  $p_i$  of an ensemble, and the implicit probabilities  $q_i$  induced by a code  $C$  encoding that ensemble, the expected length of a codeword is given by:*

$$\bar{L}_C = H(X) + \sum_i p_i \log \frac{p_i}{q_i}$$

**Theorem 3.6** (Sharper bounds on compression with symbol codes [70]). *Let  $C$  be a symbol code that encodes an ensemble  $X$ , having probabilities  $p_i$ . Let  $p_{max}$  be the largest such probability. If  $p_{max} \geq 0.5$  then  $H(X) \leq \bar{L}_C < H(X) + p_{max}$ . Otherwise  $H(X) \leq \bar{L}_C < H(X) + p_{max} + 0.086$ .*

The expression derived above for the expected length of a codeword can also be used to measure how much information is redundant in a code that is not ideal, i.e. a code whose expected length is not equal to the entropy of the source. According to the above expression, the average amount of “wasted” bits per codeword is equal to  $\sum_i p_i \log p_i/q_i$ , which is exactly the relative entropy between the distributions  $p$  and  $q$ .

### 3.4.3 Huffman coding

Huffman coding is an optimal prefix-free symbol coding scheme — no symbol code can achieve greater compression for a given ensemble [70]. The Huffman coding algorithm is as follows:

#### Huffman coding algorithm

1. Combine the two least probable symbols into a new symbol with probability equal to the sum of the probabilities of the constituent symbols. Keep track of how the symbols were combined. If a symbol is the result of such a combination, call it a composite symbol, otherwise call it an atomic symbol.
2. Repeat step 1 until only one symbol is left. Call this symbol the *parent symbol*.
3. The code for each of the two constituent symbols of the parent symbol is equal to the code for the parent symbol appended with a 0 or 1 respectively.
4. If a constituent symbol of the parent symbol is composite, make it the parent symbol and go to step 3. Otherwise the algorithm is complete.

As an example of how the Huffman coding algorithms works, consider the following ensemble:

Outcome	Probability
$a_1$	0.2
$a_2$	0.4
$a_3$	0.2
$a_4$	0.1
$a_5$	0.1



The two symbols with the lowest probabilities are  $a_4$  and  $a_5$ . We combine them into a new composite symbol  $c_1$  with probability equal to the sum of their original probabilities:

Outcome	Probability
$a_1$	0.2
$a_2$	0.4
$a_3$	0.2
$c_1$	0.2

At this combination step we combine the symbols  $c_1$  and  $a_3$  into a new composite symbol  $c_2$  with probability equal to 0.4:

Outcome	Probability
$a_1$	0.2
$a_2$	0.4
$c_2$	0.4

Here we combine the symbols  $c_2$  and  $a_1$  into a new composite symbol  $c_3$  with probability equal to 0.6:

Outcome	Probability
$a_2$	0.4
$c_3$	0.6

At the final combination step we combine the symbols  $a_2$  and  $c_3$  into a new composite symbol  $c_4$  with probability equal to 1. This is the first parent symbol. We now move to step 3 of the algorithm.

The parent symbol  $c_4$  at this stage does not have a code assigned to it, so that the intermediate codes for the symbols  $a_2$  and  $c_3$  are 0 and 1, respectively.

Since  $c_3$  is the composite symbol of the child symbols of the current parent symbol  $c_4$ , it becomes the new parent symbol (step 4 of the algorithm). Applying step 3 to this symbol, and taking into account that the code for the current parent symbol is 1, yields an intermediate code of 10 for  $a_1$  and 11 for  $c_2$ .

$c_2$ , with intermediate code 11, becomes the new parent symbol. Applying step 3 again yields an intermediate code of 110 for  $a_3$  and 111 for  $c_1$ .

At the final code assignment step,  $c_1$  with intermediate code 111 becomes the new parent symbol. Applying step 3 yields an intermediate code of 1110 for  $a_4$  and 1111 for  $a_5$ .

The final code assignment is then as shown in Table 3.1.

Outcome	Probability	Code
$a_1$	0.2	10
$a_2$	0.4	0
$a_3$	0.2	110
$a_4$	0.1	1110
$a_5$	0.1	1111

Table 3.1: Huffman code example.

How do we know that Huffman coding produces prefix-free codes? The codewords of any binary code can be attached to the nodes of a binary tree. Figure 3.1 shows the corresponding binary tree for the Huffman code shown in Table 3.1. The code for an outcome can be obtained by traversing the tree from its root (i.e. the node without any parents;  $c_4$  in the case of Figure 3.1) to the node corresponding to the outcome in question, appending a 0 or 1 to the code for that outcome depending on whether a left or right branch was followed at a junction in the tree. It can be seen that every outcome in the alphabet is a leaf node of the binary tree, i.e. a node with no children. This means that no code for any outcome is a prefix for any other code for any other outcome, so that the generated code is *prefix-free* in the sense of Section 3.4.2.1.

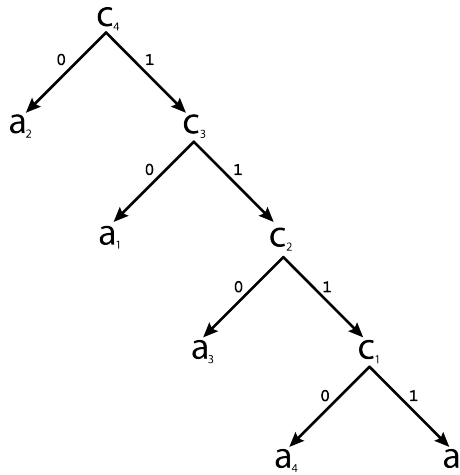


Figure 3.1: Binary tree corresponding to the Huffman code in Table 3.1

The average length of codewords generated by the Huffman coding algorithm can be characterized as follows [70]:

$$H(X) \leq \overline{L}_C < H(X) + 1 \quad (3.1)$$

where  $H(X)$  is the entropy of the source and  $\overline{L}_C$  is the average codeword length. This characterization follows from an argument involving the Kraft-McMillan inequality discussed earlier and is in fact not sharp. A sharper bound on the average length of Huffman-generated codewords is as follows [70]:

$$H(X) \leq \overline{L}_C < \begin{cases} H(X) + 0.086 + p_{max} & \text{if } p_{max} < 0.5 \\ H(X) + p_{max} & \text{if } p_{max} \geq 0.5 \end{cases} \quad (3.2)$$

### 3.4.3.1 Weaknesses of Huffman coding

A Huffman code uses at least one bit per symbol. The lower bound of compressibility is the entropy  $H(X)$  of the source, and in cases where this entropy is less than one bit, Huffman coding is not very efficient. As an example, consider a source with an entropy of  $H(X) = 0.023$  bits, with a source string length of 100. The lower bound on the compressed length is 2.3 bits, whereas

Huffman coding can at best achieve a compressed length of 100 bits — this is clearly not ideal.

### 3.4.4 Golomb coding

Golomb codes are used to encode integers, with the assumption that the probability of an integer appearing is inversely proportional to its size, i.e. smaller integers are more probable than larger ones [29, 70].

Golomb codes are parameterized by an integer  $m > 0$ . A Golomb code represents an integer  $n \geq 0$  using two numbers  $q$  and  $r$  in the following way:

$$q = \left\lfloor \frac{n}{m} \right\rfloor$$

$$r = n - qm$$

so that  $q$  is the quotient and  $r$  is the remainder when  $n$  is divided by  $m$ .

The number  $q$  is then encoded using the *unary code*, which is a binary code that encodes an integer  $k$  by  $k$  ones followed by a zero<sup>5</sup>, so that, for instance, the unary code for 5 is 111110.

The number  $r$  (with  $0 \leq r < m$ ) is encoded using the regular binary representation. Ordinarily this would require  $\lceil \log_2 m \rceil$  bits, but there is a way to reduce the average number of bits required to represent  $r$ . It works as follows. The first  $2^{\lceil \log_2 m \rceil} - m$  values are encoded using the  $\lceil \log_2 m \rceil$ -bit binary representation. The rest of the values are encoded as  $r + 2^{\lceil \log_2 m \rceil} - m$ , using  $\lceil \log_2 m \rceil$  bits. To decode this representation of  $r$  the actual number of bits used to encode it must somehow be known by the decoder. This is made possible by the fact that all the numbers encoded with  $\lceil \log_2 m \rceil$  bits have a common prefix, which is simply all the leftmost binary ones of the binary representation of  $2(2^{\lceil \log_2 m \rceil} - m)$ . For example, the prefix derived from the binary number 100 would be 1, and the prefix derived from 110 would be 11. The  $\lceil \log_2 m \rceil$ -bit encoded numbers do not have this prefix, so that upon encountering it the decoder knows that the number being decoded is  $\lceil \log_2 m \rceil$  bits long, and that it must subtract  $2^{\lceil \log_2 m \rceil} - m$  from that number to obtain  $r$ .

To illustrate the Golomb code, consider the two cases,  $m = 4$  and  $m = 5$ , shown in Tables 3.2 and 3.3 respectively.

n	q	r	Code	n	q	r	Code
0	0	0	000	8	2	0	11000
1	0	1	001	9	2	1	11001
2	0	2	010	10	2	2	11010
3	0	3	011	11	2	3	11011
4	1	0	1000	12	3	0	111000
5	1	1	1001	13	3	1	111001
6	1	2	1010	14	3	2	111010
7	1	3	1011	15	3	3	111011

Table 3.2: Golomb code with  $m = 4$ .

<sup>5</sup>Or, equivalently,  $k$  zeros followed by a one.

n	q	r	Code	n	q	r	Code
0	0	0	000	8	1	3	10110
1	0	1	001	9	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000

Table 3.3: Golomb code with  $m = 5$ .

The Golomb code is optimal for a source with a geometric probability distribution [27]:

$$p(k) = g(1 - g)^k$$

where  $g$  is the distribution parameter. This distribution is illustrated in Figure 3.2.

The optimal Golomb parameter  $m$  for this distribution is:

$$m = \left\lceil -\frac{1}{\log_2 g} \right\rceil$$

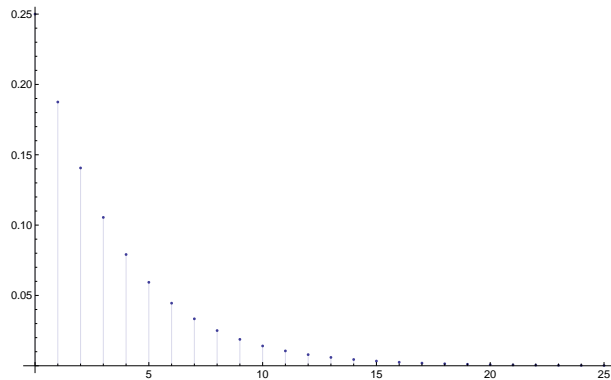


Figure 3.2: Discrete geometric probability distribution with parameter  $g = 0.25$ .

#### 3.4.4.1 Adaptive and Generalized Golomb coding

*Rice coding* can be described as an adaptive Golomb code: the input to the coder is divided into blocks of a certain length, and the block is encoded using the optimal Golomb parameter  $m$  for that block, using a brute-force approach. The optimal parameter is stored along with the encoded block [56, 40]. Rice coding is widely used in lossless image and audio compression.

### 3.5 Stream codes

*Stream codes* encode a sequence (or stream) of symbols with one codeword, in contrast to *symbol codes* that encode each symbol with a separate codeword, forcing the codewords of a symbol code to be at least one bit long. Stream codes do not have this limitation, so that we can expect that it will compress sources with a very low entropy very effectively. In fact, with some stream codes the average number of bits needed to encode an symbol is virtually equal to the entropy for that source, which, theoretically, is the best performance that can be hoped for [47]. One such stream code will be discussed in the next section.

#### 3.5.1 Arithmetic coding

*Arithmetic coding* is based on the idea that a sequence of symbols can be uniquely associated with a real number in the unit interval [70, 47]. To see how arithmetic coding accomplishes this, consider first how it encodes a single symbol. To uniquely associate a symbol with a real number an invertible function  $c : \mathcal{A} \rightarrow \mathbb{R}$  is needed. Without loss of generality we can let the function  $c$  map symbols from  $\mathcal{A}$  into the unit interval. For the function  $c$  to be invertible it has to be monotonic. It also has to take the probability of a symbol into account: less probable symbols should have longer representations (codewords) than symbols that are more probable. The *cumulative distribution function*  $F_S$  of the source satisfies both these requirements. The cumulative distribution function is used to completely subdivide the unit interval into nonoverlapping subintervals  $I_i$ , with the lower limit  $\ell(I)$  of each subinterval given by:

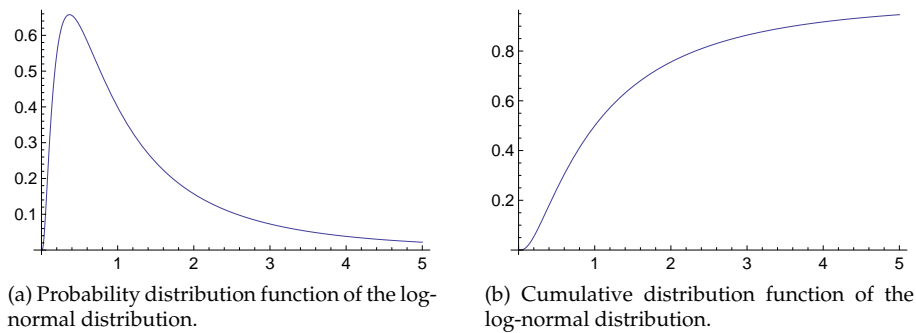


Figure 3.3: Probability distribution function and cumulative distribution function of a random variable.

$$\ell(I_i) = \sum_{j=1}^{i-1} p(a_j) \quad (3.3)$$

and the upper limit  $u(I)$  of each subinterval given by:

$$u(I_i) = \sum_{j=1}^i p(a_j) \quad (3.4)$$

The size of each subinterval is directly proportional to the probability of the symbol associated with that subinterval. To encode a symbol it is enough to know the corresponding interval for that symbol. To encode a sequence of symbols, recursion is used. After the first symbol is assigned to a subinterval, that subinterval is remapped to the unit interval, and the process is repeated for every symbol in the sequence. It is clear that the subintervals for any symbol in a given position do not overlap; they are unique, so that it is enough to know the extent of the subinterval in order to decode the sequence. Since the subintervals are disjoint using the upper and lower limits of a subinterval to identify it is redundant: any number in the subinterval will do, providing that the number is chosen the same way for every subinterval. We will use the midpoint of an subinterval to identify it.

To decode a sequence given a real number — the code — the same kind of process is followed. We assume that the original sequence length is known. The unit interval is subdivided just as before, and the symbol corresponding to the subinterval in which the code lies is the first symbol in the sequence. That subinterval is then subdivided again in the same way as the unit interval, and the symbol corresponding to the subinterval in which the code now lies is the second symbol. The decoding process continues in this manner, until all the symbols of the sequence are decoded. Figure 3.4 illustrates the whole process.

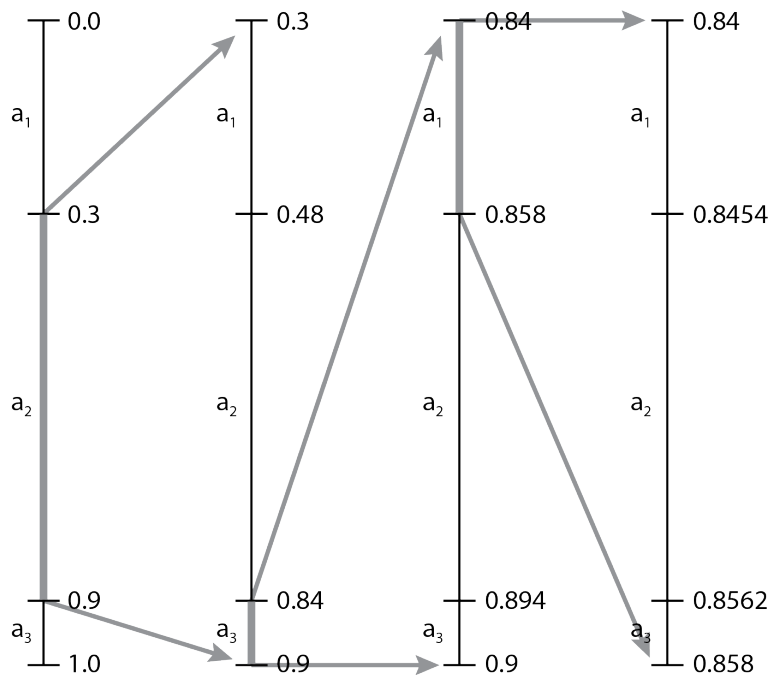


Figure 3.4: Using arithmetic coding to code the sequence  $a_2 a_3 a_1$  from the ensemble  $\{ \{a_1, 0.3\}, \{a_2, 0.6\}, \{a_3, 0.1\} \}$ .

### 3.5.1.1 Binary arithmetic codes

So far the codes generated by arithmetic coding have been real numbers, which is not very efficient since real numbers can potentially be irrational and therefore require an infinite amount of data to represent. To be practical arithmetic coding has to work with codes that have only a finite amount of precision, and preferably be amenable to easy representation with binary numbers. Fortunately this is possible. To identify a subinterval  $I_i$  with a binary number, the following strategy is used: the binary number 0 represents the lower half of an interval, and the binary number 1 represents the upper half. This strategy is applied recursively, analogously to the process previously. Starting with the unit interval, the binary number 00 will then represent the first quarter of the unit interval, while the binary number 10 will represent the third quarter of the unit interval. As soon as such a “binary” interval falls entirely within a subinterval  $I_i$ , that binary interval is taken to *identify* that subinterval. Both the encoding and decoding processes described above can subsequently be carried out using binary numbers to represent the code generated by the arithmetic coding process.

### 3.5.1.2 Probabilistic modelling and arithmetic coding

The probability model used during arithmetic coding need not be static: a different probability model can be used to encode every symbol, as long as both the encoder and decoder have access to it. This fact can be used to build a so-called *adaptive* coder that adapts to the statistics of the source automatically. This is useful in situations where the source statistics are unknown. Such an adaptive coder can be built by starting off with a uniform probability distribution over the source alphabet, and increasing the probability of a symbol as soon as it has been encoded. Over time the probability distribution will approximate the true probability distribution of the source.

More intelligent adaptive arithmetic coders can be designed that take into account more context than just the symbol at hand. For example, instead of using a one-dimensional probability distribution  $p(a_i)$  to encode a symbol, a multi-dimensional probability distribution  $p(a_i | a_{i-1}a_{i-2}a_{i-3} \dots)$  can be used to take into account the context of the symbol to be encoded. This can lead to probability distributions that are more accurate with a lower entropy, so that the coding and hence compression is more efficient.

## 3.6 Quantization

*Quantization* is the process of reducing the size of a signal by reducing the amount of numbers available to represent individual samples of the signal [70]. This operation is of course *lossy*, in the sense discussed in section 3.3.1 above, and as such is used only with lossy data compression. The word “quantization” implies the discretization of real numbers, but in practice it actually just amounts to reducing the number of discrete units available to represent a signal, given the finite precision of any kind of number in a computer system. Quantization ultimately has two conflicting goals: reducing the size of a signal while at the same time maximizing the fidelity of the resulting lossy signal.

There are basically two types of quantization, defined by the type of input and output of the quantization algorithm. One is called *scalar quantization*, where the entities being quantized are *scalars* (i.e. one-dimensional entities) and the other is called *vector quantization*, where the entities being quantized are *vectors*, which can be  $n$ -dimensional in general.

We will only explore scalar quantization in this section since we will be using scalar quantization to build lossy compression algorithms in section 4.5. The reason for this is that scalar quantization is adequate for the purpose at hand, as will become clear during the discussion below.

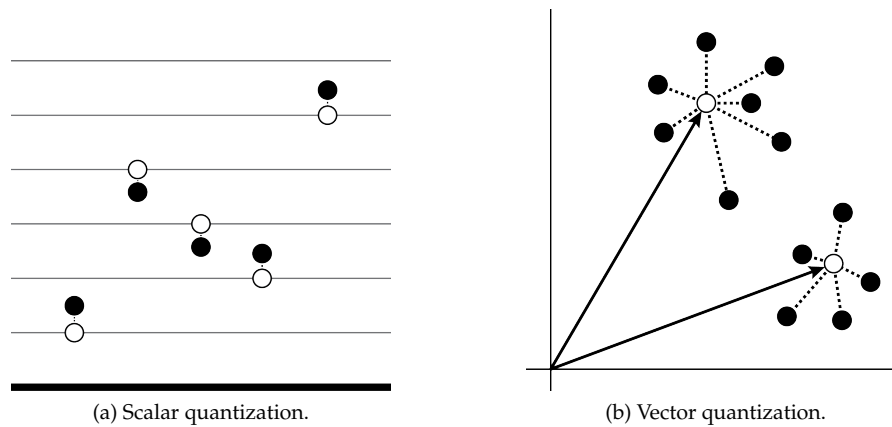


Figure 3.5: The two types of quantization.

### 3.6.1 Scalar quantization

With scalar quantization the entities being quantized are scalars — ordinary real numbers. In general quantization is achieved by selecting certain input intervals and for each such interval selecting a value that will represent all the values contained in that interval. The endpoints of the intervals are called the *decision boundaries*, and can be denoted by  $d_i$ . The representative values for each interval are called *reconstruction levels*, and can be denoted by  $y_i$ . Figure 3.6 illustrates a generic quantizer. If  $M$  intervals are specified then the resulting quantizer is called an  $M$ -level quantizer. Note that for an  $M$ -level quantizer there will always be  $M + 1$  decision boundaries; the first decision boundary may be designated by  $d_0$ . With these definitions we can now formalize the operation of a quantizer as a function  $Q(x)$ :

$$Q(x) = y_i \iff d_{i-1} < x \leq d_i$$

The difference between the original signal and the quantized signal,  $x - Q(x)$ , is called the *quantization error*. The quantization error can also be seen as a kind of additive “noise” that influences the original signal, producing the quantized signal. Modelling the quantization error as additive noise is sometimes beneficial conceptually.



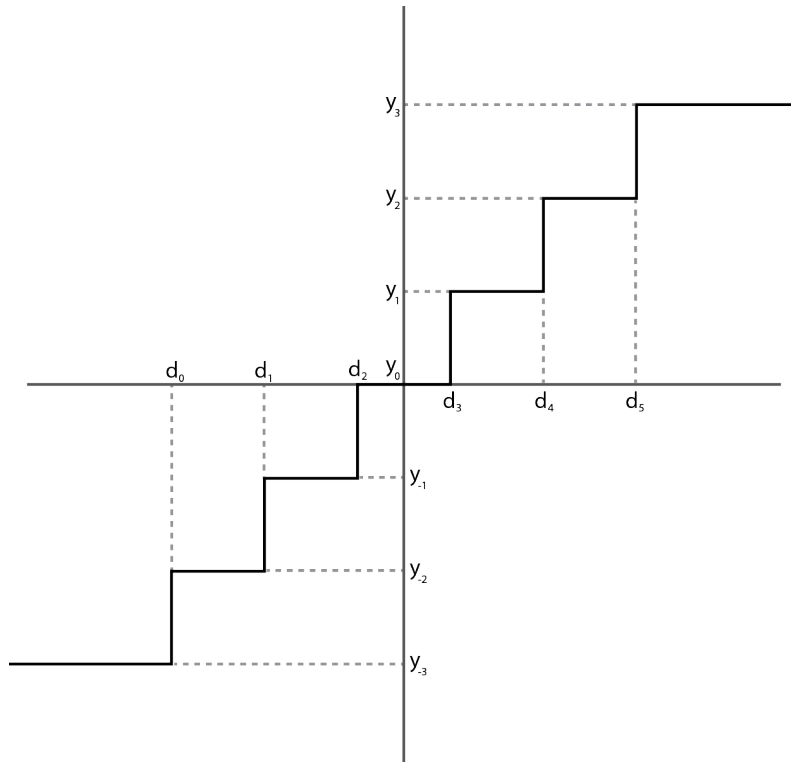


Figure 3.6: Generic scalar quantization.

The amount of quantization error can be characterized by the *mean squared error* between the original signal and the quantized signal, as follows:

$$\sigma_q^2 = \int_{-\infty}^{\infty} [x - Q(x)]^2 p(x) dx$$

where  $p(x)$  is the probability density function governing the behaviour of the input values  $x$ .

Using the definition of  $Q(x)$  the expression above can be written more explicitly as:

$$\sigma_q^2 = \sum_{i=1}^M \int_{d_{i-1}}^{d_i} (x - y_i)^2 p(x) dx \quad (3.5)$$

These definitions enable us to state the problem of quantizer design as follows [70]: given an input probability distribution function,  $p(x)$ , and the number of levels  $M$  in the quantizer, find the decision boundaries  $d_i$  and the reconstruction levels  $y_i$  so as to minimize the mean squared quantization error.

An important property of scalar quantizers is whether they include zero as one of their reconstruction levels. For many applications this is important, an example being lossy audio compression where it is important to be able to accurately represent silence, a zero value. Quantizers including zero as a reconstruction value are called *midtread quantizers* while those that do not are

called *midrise quantizers*. Figure 3.7 illustrates the difference between these two types of quantizers.

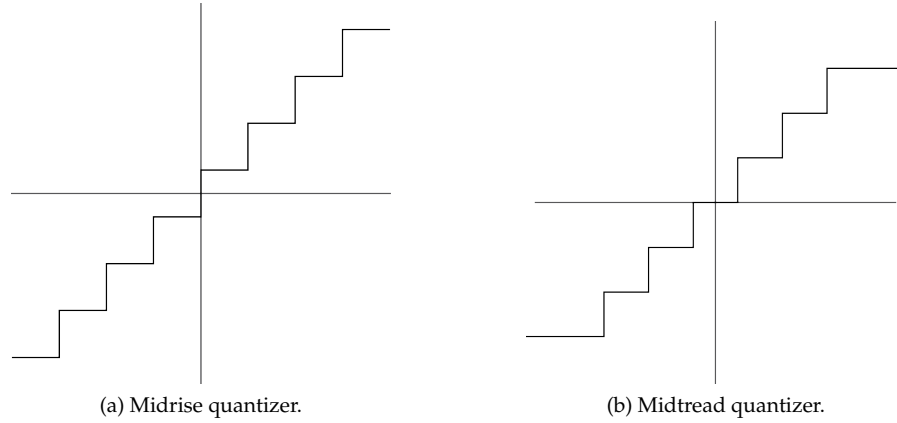


Figure 3.7: Midrise and midtread quantizers.

It should also be noted that the scalar quantization operators as defined here commute with all compositions of the basic *LULU* operators as mentioned in Section 2.4.3.6.

### 3.6.1.1 Uniform quantization

The *uniform quantizer* is a quantizer where both the decision boundaries and reconstruction levels are evenly spaced, so that they are all the same size, i.e. they are all uniform, except for possibly the two outer intervals which may be infinite in size, which might be the case when dealing with a large input range. The constant spacing of the decision boundaries and reconstruction levels is called the *step size*,  $\Delta$  [70].

The expression for the mean squared error of a uniform quantizer can be found by using equation 3.5 and the fact that the decision boundaries  $d_i$  and the reconstruction levels  $y_i$  are governed by the step size  $\Delta$ . After some algebra this becomes:

$$\begin{aligned} \sigma_q^2(\Delta) = & 2 \sum_{i=1}^{\frac{M}{2}-1} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right)^2 p(x) dx \\ & + 2 \int_{(\frac{M}{2}-1)\Delta}^{\infty} \left(x - \frac{M-1}{2}\Delta\right)^2 p(x) dx \end{aligned} \quad (3.6)$$

Designing an optimal uniform quantizer for an input with a probability distribution  $p(x)$  amounts to finding the value of  $\Delta$  that minimizes equation 3.6. This can be done by taking the derivative of equation 3.6, setting it to zero, and solving for  $\Delta$ .

$$\begin{aligned} \frac{d\sigma_q^2}{d\Delta} = & - \sum_{i=1}^{\frac{M}{2}-1} (2i-1) \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right) p(x) dx \\ & - (M-1) \int_{(\frac{M}{2}-1)\Delta}^{\infty} \left(x - \frac{M-1}{2}\Delta\right) p(x) dx = 0 \end{aligned} \quad (3.7)$$

Due to the complexity of this equation and the presence of a possibly non-analytic probability distribution function  $p(x)$  it is usually solved numerically.

### 3.6.1.2 Nonuniform quantization

Like its name suggests, a *nonuniform quantizer* is a quantizer where the decision boundaries and reconstruction levels are not evenly spaced. This is motivated by the observation that the distortion caused by quantization can be minimized if we represent values that have a high probability of occurring more accurately than other less probable values. This amounts to decreasing the spacing between the decision boundaries for sections of input that have a high probability of occurring [70]. The effect of this scheme on the quantization error can be seen directly when looking at Equation 3.5.

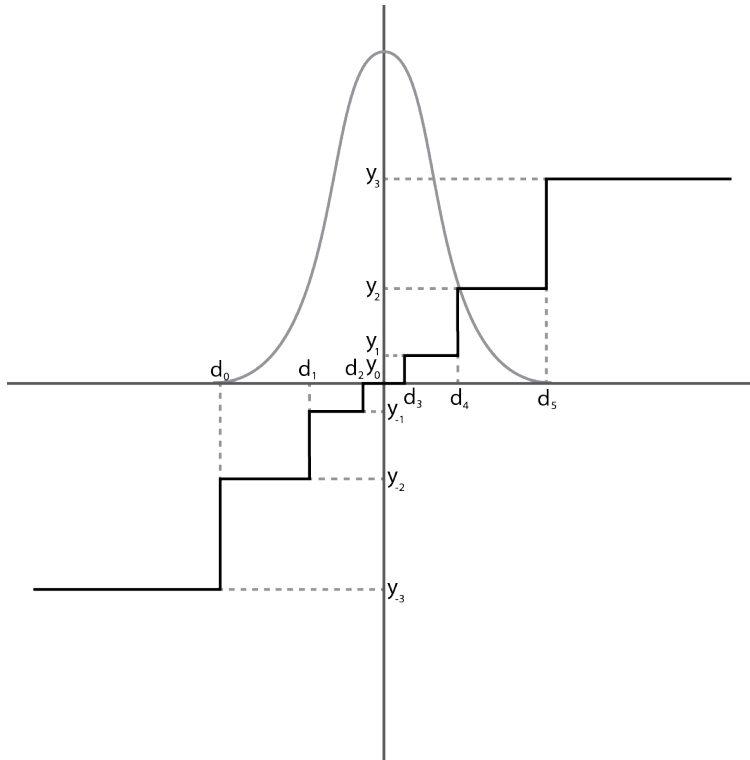


Figure 3.8: Non-uniform quantization.

While the basic problem of nonuniform quantization can be approached in the same way as uniform quantization, that is, finding the decision bound-

aries and reconstruction levels that will minimize the mean squared error, the solution is much more complex than that of uniform quantization. Figure 3.8 illustrates a generic nonuniform quantizer.

Designing an optimal  $M$ -level nonuniform quantizer for a source with a known probability distribution function amounts to finding decision boundaries  $d_i$  and reconstruction levels  $y_i$  that minimizes the mean squared quantization error given in Equation 3.5. To find an expression for the reconstruction levels  $y_i$  we take the partial derivative of Equation 3.5 with respect to  $y_i$ , and set it to zero. The resulting expression is:

$$y_j = \frac{\int_{d_{j-1}}^{d_j} xp(x)dx}{\int_{d_{j-1}}^{d_j} p(x)dx} \quad (3.8)$$

The decision boundaries are defined as the midpoints of neighbouring reconstruction levels:

$$d_j = \frac{y_{j+1} + y_j}{2} \quad (3.9)$$

These two equations are interdependent so to solve them simultaneously an iterative numerical procedure is needed. This iterative numerical procedure is called the *Lloyd-Max algorithm*, and was discovered independently at least three times [46, 49, 45]. The algorithm works as follows:

1. Choose the initial reconstruction levels  $y_j$  arbitrarily.
2. For  $1 \leq j \leq M - 1$ , set  $d_j = \frac{1}{2}(y_{j+1} + y_j)$ .
3. For  $1 \leq j \leq M$ , set  $y_j$  equal to the mean of  $p(x)$  in the interval  $x \in (d_{j-1}, d_j]$ .
4. Repeat steps (2) and (3) until there is no more significant improvement in the mean squared error  $\sigma_q^2$ .

As the Lloyd-Max algorithm is executed, the mean squared error  $\sigma_q^2$  decreases monotonically, and since the mean squared error is non-negative it approaches some limit. Consequently, if the algorithm terminates if the improvement in mean squared error is less than some  $\varepsilon > 0$ , then the algorithm must terminate in a finite number of iterations.

Optimized nonuniform quantizers give more accurate results than optimized uniform quantizers for a given number of reconstruction levels, as can be seen in Table 3.4, where the signal-to-noise ratios of optimized uniform and nonuniform quantizers for Gaussian and Laplacian sources are compared. The difference in accuracy is more apparent with the Laplacian distribution since it is more peaked than the Gaussian distribution. In general, the more peaked the probability distribution of a source is<sup>6</sup>, the bigger the accuracy gap between an optimized uniform quantizer and an optimized nonuniform quantizer will be, with the nonuniform quantizer always outperforming the uniform quantizer.

---

<sup>6</sup>Or equivalently, the lower its entropy is.

	<i>Gaussian</i>		<i>Laplacian</i>	
<i>M</i>	<b>Uniform</b>	<b>Nonuniform</b>	<b>Uniform</b>	<b>Nonuniform</b>
4	9.24	9.3	7.05	7.54
6	12.18	12.41	9.56	10.51
8	14.27	14.62	11.39	12.64

Table 3.4: Signal-to-noise ratios (in dB) of optimal uniform and nonuniform quantizers for Gaussian and Laplacian distributions (adapted from [70]).

	<i>Gaussian</i>		<i>Laplacian</i>	
<i>M</i>	<b>Uniform</b>	<b>Nonuniform</b>	<b>Uniform</b>	<b>Nonuniform</b>
4	1.904	1.911	1.751	1.728
6	2.409	2.442	2.127	2.207
8	2.759	2.824	2.394	2.479
16	3.602	3.765	3.063	3.473
32	4.449	4.730	3.779	4.427

Table 3.5: Output entropies (in bits) of optimal uniform and nonuniform quantizers for Gaussian and Laplacian distributions (adapted from [70]).

### 3.6.2 Properties of Probabilistically Optimized Scalar Quantizers

Both uniform and nonuniform scalar quantizers have certain properties when they have been probabilistically optimized — that is, when the probability density function  $p(x)$  of the source is known and when the reconstruction levels  $y_j$  and the decision boundaries  $d_j$  have been chosen so as to minimize Equation 3.5. The properties are as follows [70]:

1. The mean value of the input to the quantizer is equal to the mean value of the output of the quantizer.
2. For a given probabilistically optimized quantizer, the variance of its output is always less than or equal to the variance of its input.
3. The mean squared quantization error  $\sigma_q^2$  is given by:

$$\sigma_q^2 = \sigma_x^2 - \sum_{j=1}^M y_j^2 \int_{d_{j-1}}^{d_j} p(x) dx$$

where  $\sigma_x^2$  is the variance of the source and  $p(x)$  is the probability density function of the source.

4. Let  $X$  be the random variable representing the source and  $Z$  be the random variable representing the quantization error. Then:

$$E[XZ] = -\sigma_q^2$$

5. For a given probabilistically optimized quantizer, the output of the quantizer is orthogonal to the quantization noise in the following sense:

$$E[Q(X) Z | d_0, d_1, d_2, \dots, d_M] = 0$$

### 3.6.3 Entropy Coding of Quantizer Output

The simplest way to encode the output of an  $M$ -level quantizer is to assign a unique code of length  $\lceil \log_2 M \rceil$  bits to each reconstruction level. If the probability distribution of the outputs is not uniform, however, then *entropy coding* the output is a more efficient solution, as discussed in Section 3.4.2.

Empirically it has been found that the output of a uniform quantizer is entropy coded more efficiently than the output of a nonuniform coder [70]. This can be explained by the fact that nonuniform quantizers have smaller quantization intervals in areas of high probability and larger quantization intervals in areas of low probability, which has an equalizing effect on the input probabilities of the quantization intervals. Consequently the output probabilities are affected in the same way, so that the output probability distribution becomes more uniform, increasing its entropy and making it more expensive to code. Table 3.5 shows the difference between the coding rates of optimal uniform and nonuniform quantizers for a sources with a Gaussian and Laplacian distribution respectively. From the table it can be seen that the difference in coding efficiency between the uniform and nonuniform quantizers become more pronounced as more reconstruction levels are used. The difference is also dependent upon the entropy of the source distribution, with the uniform quantizer outperforming the nonuniform quantizer by a bigger margin when the entropy of the source is low, as is the case with the Laplacian distribution.

## 3.7 Conclusion

In this chapter the basic theory behind data compression was introduced and it was seen that the theory of data compression is actually a part of information theory. Two basic lossless data compression categories were introduced: symbol coding and stream coding. Huffman and Golomb coding were presented as two prototypical symbol coding techniques, and arithmetic coding was presented as the prototypical stream coding technique, achieving the best possible performance of any coding technique.

Quantization was introduced as an example of a lossy compression technique. We focused on scalar quantization and saw how to build an optimum uniform and nonuniform quantizer for a particular data source.

These data compression techniques will now be applied to compress images in the next chapter.

## Chapter 4 - Image Compression using the DPT

### 4.1 Introduction

In this chapter the concepts explored in the previous two chapters will be used to construct two types of image compression schemes: one type of scheme will be lossless and the other lossy. The lossy compression schemes will be based on the best lossless compression scheme, since the path from the one to the other is clear and natural.

### 4.2 Compression Scheme Structure

All the compression schemes developed in this text will follow a common structure, shown in Figure 4.1.

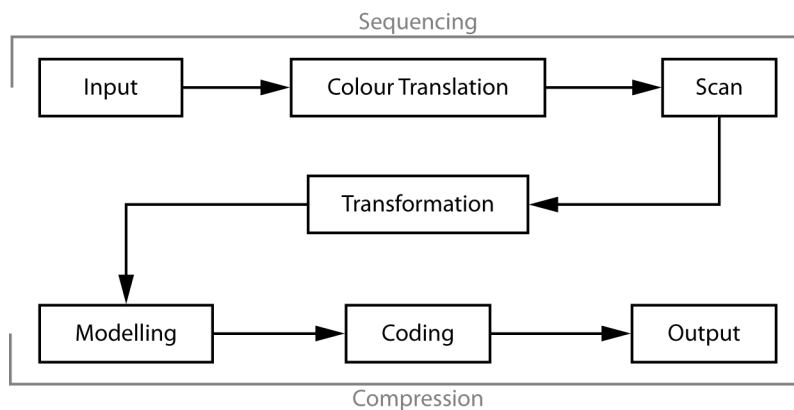


Figure 4.1: Compression scheme structure.

As indicated, the structure consists of three components. The first component, called *sequencing*, is the process of forming a sequence<sup>1</sup> (or sequences) from the two-dimensional data of the input image so that it can be fed into the DPT.

The second component is called *transformation*, and it is the process of applying a DPT transform to the sequence using a suitable DPT operator.

The third component, *compression*, is the actual process of compressing the sequence via its DPT representation, thereby compressing the original image because of the consistency of the DPT transform, resulting in an equivalence

<sup>1</sup>In the sense of Section 2.2.4.

between a sequence and its DPT transform (see Section 2.4.3.2). This component consists of three subcomponents, the first two of which, *modelling* and *coding*, were discussed in Chapter 3. These two subcomponents will form the bulk of the work presented in this chapter: finding suitable representations of the DPT structure and coding them efficiently. The last subcomponent, *output*, is the actual output at the file level on a computer system. There are many properties that are desirable in an image compression scheme that can be said to fall under this heading, for example: progressive transmission of resolution levels, region-of-interest coding, bit-error robustness, image metadata and image security [32, 14, 17, 41]. For this text, however, this subcomponent is considered to be out of scope. Mention will nevertheless be made of it if it is evident that some feature can be implemented in a straightforward manner during the modelling and coding phases.

The sequencing component will be shared across all the compression schemes developed here since it is essentially independent from the transformation and compression components, and a discussion about this component follows.

### 4.3 Sequencing

#### 4.3.1 Input

A digital image  $I_a$  can be formalized as a collection of finite, integer-valued, two-dimensional functions with discrete independent variables as follows [37]:

$$I_a := \{f_i\} \quad f_i(x, y) \in \mathbb{Z} \quad x, y \in \mathbb{Z} \quad 0 \leq f_i, x, y < \infty$$

where each function  $f_i$  constitutes a *colour plane* or *colour channel*. Each colour plane consists of a finite number of elements called *pixels*<sup>2</sup>. The value of the function  $f_i$  at a certain pixel's location is called that pixel's *intensity*.

The most important attributes of a digital image is its *resolution* and its *colour depth*. The resolution of an image is basically the number of pixels in an image, although in practice it usually refers to the actual dimensions of an image from which the pixel count can be subsequently calculated. The colour depth of an image is a measure of how many colours a pixel in an image can assume and it is usually measured in bits, referring to the exponent  $k$  in the following equation:  $C = 2^k$ , where  $C$  is the number of distinct colours. A greyscale image usually has 256 intensity levels so that its colour depth is  $\log_2 256 = 8$  bits. Since any colour perceivable by the human eye is a combination of red, green and blue, a colour image is divided into so-called *colour planes*, with each plane assigned to a primary colour [10]. When these colour planes are viewed independently in this way they can be thought of as greyscale images, so that a colour image is basically a combination of three greyscale images.

Digital images can be formed from a variety of sources, the most well-known source being the visible part of the electromagnetic spectrum. There are many other sources of digital images, for example: other bands in the electromagnetic spectrum, like gamma rays, X-rays, ultraviolet, infrared, microwaves and radio waves; acoustic energy, used in ultrasound; and electronic energy, used, for example, in electron microscopy [30].

---

<sup>2</sup>From *picture element*.



### 4.3.1.1 Image Test Set

A standard set of images will be needed to test the compression algorithms developed in this text.

The Image Test Set is a set of 1000 images of varying resolution, gathered from various sources. Most of these images are colour images, with a standard colour depth of 24 bits, with 8 bits per colour channel. Depending on the desired resolution for the application at hand (for example, testing scanning orders; see Section 4.3.3) these images were processed as follows to yield 1000 colour images with dimensions  $n \times n$ :

1. The images were all resized (preserving aspect ratio and using resampling for more accurate results [76]) so that their shortest sides were equal to  $n$  pixels.
2. The images were then cropped so that only the  $n \times n$  sized area at the bottom-left of the resized image remained. The bottom part of the image was chosen conservatively because in practice images have more variation in their bottom areas than in their top areas (for example, in an image of a landscape the top area contains mostly sky, which has lower variation than the bottom area containing the ground).

### 4.3.2 Colour Translation

As mentioned earlier, colour images consist of three colour planes that together provide a colour coordinate in some colour space. The colour space that is used most often is the simplest one: the RGB colour space. The RGB colour space records the intensity of each primary colour at each pixel.

There are many other colour spaces available, for instance HSL, HSV, CIE  $L^*u^*v^*$ , CIE  $L^*a^*b^*$  and CMYK [10]. The most important factor in selecting a colour space for image compression is component independence: the three colour planes of the colour space transformed image should contain as little correlation as possible; in other words, they should contain as little mutual information as possible in order to avoid encoding the same information twice [80, 15].

The  $Y'C_bC_r$  colour space is generally viewed as having this property and it is used in both the original JPEG and the new JPEG2000 image compression standards for this reason [80, 15]. The  $Y'C_bC_r$  colour space uses one plane to encode the intensity information (i.e. how light or dark a pixel is) of an image and the two remaining planes encode the colour information. The transformation from RGB to  $Y'C_bC_r$  can be calculated using matrix algebra as follows:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad (4.1)$$

Figures 4.3 and 4.4 show the RGB and  $Y'C_bC_r$  colour planes for two images. In both examples it can be seen that all the RGB colour planes contain almost all the detail of the original image, as far as intensity is concerned. This is not the

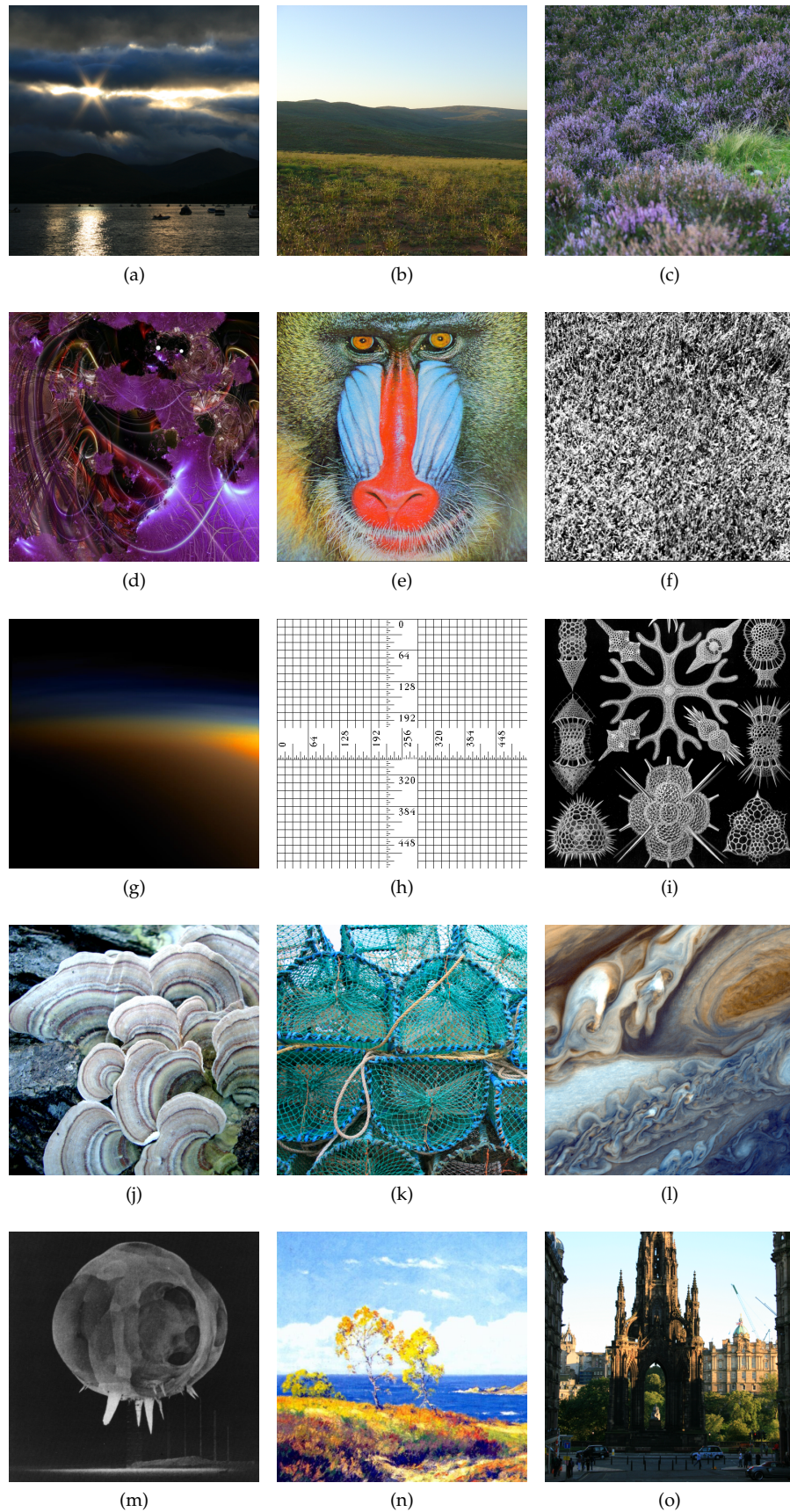


Figure 4.2: Images from the Image Test Set (see Section 4.3.1.1)

case with the  $YC_bC_r$  colour planes: typically it is only the  $Y$  plane that contains the same amount of detail (and thus information) as the original image. The  $C_b$  and  $C_r$  planes that encode colour information are much smoother, with lower total variation, and generally their intensities are also not correlated very strongly. The efficiency of the  $YC_bC_r$  colour space can be seen in Figure 4.4 especially, where the  $Y$  plane contains virtually *all* the detail with the  $C_b$  and  $C_r$  planes having an almost uniform intensity.

For all of the image compressors developed in this chapter we will subsequently simplify the problem and focus only on compressing greyscale images, because of the fact that an effective colour image compressor can be built from an effective greyscale image compressor using the  $YC_bC_r$  colour space.

### 4.3.3 Image Scan

#### 4.3.3.1 Goal

Since we will be using the one-dimensional DPT in the transformation stage of our image compressors it is necessary to somehow convert the two-dimensional data of an image to a one-dimensional sequence that can be transformed by the DPT. What is needed is an invertible map of the following type:  $f : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ .

Information will of course be lost in this process of dimensionality reduction; therefore this map must satisfy an additional requirement: it must minimize the amount of information lost during the reduction process, or, equivalently, it must capture as much of the two-dimensional image structure as possible.

#### 4.3.3.2 Scanning Orders

A *scanning order* is simply a way to order the points in a discrete space: it is a bijective mapping from  $\mathbb{Z} \rightarrow \mathbb{Z}^n$ , so that it fulfills one of the requirements stated in the previous section. The discrete space, in our case, is the two-dimensional image plane. A scanning order can be derived quite naturally from a space-filling curve, and vice versa; in fact, we will use the two terms interchangeably. A *space-filling curve* is a continuous curve that visits every point in an  $n$ -dimensional space. More formally, it is a continuous, surjective mapping from  $\mathbb{R} \rightarrow \mathbb{R}^n$  [34, 69]. These curves are related to (and can be used to prove) a result of Cantor showing that any two smooth finite-dimensional manifolds contain the same number of points, i.e. that they can be brought to a one-to-one correspondence, so that for instance the unit interval and the unit square have the same cardinality [11]. Many space-filling curves are also fractal in nature. The Sierpiński-Knopp curve is an example of such a fractal curve [52].

Without loss of generality we will assume that the target of our scanning orders is a regular subdivision of the unit square, analogous to the image plane. It is important to note that all the scanning orders we will be considering have a so-called *atomic size* which is the square root of the area of the recursive structure used to define the scanning order (see [34] for more details). Practically, this means that a scanning order with atomic size  $s_a$  can only scan grids with side-lengths that are powers of  $s_a$ . Henceforth we will assume that our input images have dimensions that are compatible with the scanning order we are using. There are various techniques available to deal with images that have



(a) Landscape image



(b) Red plane



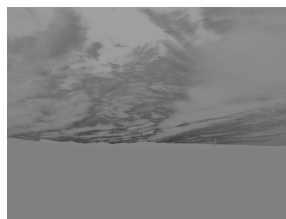
(c) Green plane



(d) Blue plane

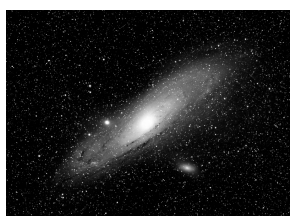


(e) Y plane

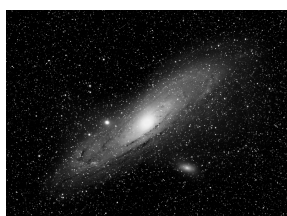
(f)  $C_b$  plane(g)  $C_r$  planeFigure 4.3:  $RGB$  and  $YC_bC_r$  colour planes of an image depicting a landscape.



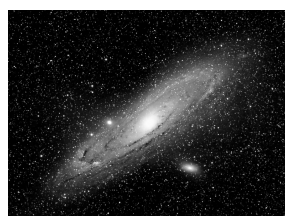
(a) Galaxy image



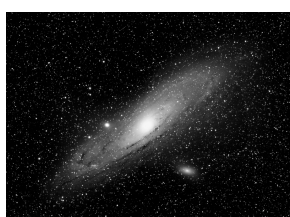
(b) Red plane



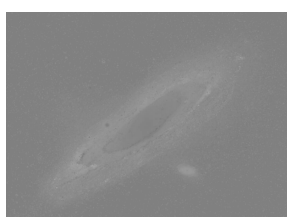
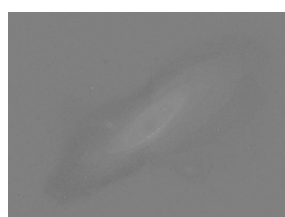
(c) Green plane



(d) Blue plane



(e) Y plane

(f)  $C_b$  plane(g)  $C_r$  planeFigure 4.4: RGB and  $YC_bC_r$  colour planes of an image depicting a galaxy.

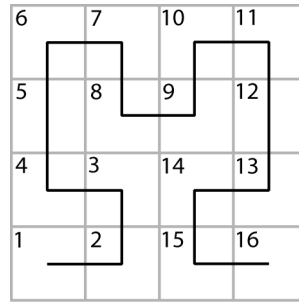


Figure 4.5: Using the Hilbert scanning order to scan a  $4 \times 4$  grid.

incompatible dimensions [15]. These techniques will have a negligible impact on the efficiency of our compression algorithms, so that we will not discuss them here.

As stated earlier, the main goal of the sequencing component is to minimize the amount of information lost during the process of moving from a two-dimensional image to a one-dimensional sequence and this can be done by capturing as much of the image structure as possible. To do this, it has to exploit the two-dimensionality of the source in some way. Intuitively, a scanning order can be called local if points that are close to each other in the two-dimensional image are also close to each other in the one-dimensional scanned sequence in some sense. The formalization of this concept is called the *locality* of a space curve [34].

Locality in scanning orders can be measured in a variety of ways [50, 54]. We will use the locality measure defined by Gotsman et. al. in [31], called *worst-case locality*,  $WL_r$ :

$$WL_r = \lim_{m \rightarrow \infty} \max_{1 \leq i < j \leq m} \frac{d_r(S(i), S(j))^2}{(j - i)/m} \quad i, j, m \in \mathbb{Z} \quad (4.2)$$

where  $S(i)$  is the centerpoint of the  $i$ th square of the subdivision of the unit square and  $d_r$  is the distance between two points in the plane according to the  $r$ -norm. Worst-case locality essentially measures how far away two points might be in the plane if they are close together in the resulting scanned sequence. We will be using the three usual norms  $L_1$ ,  $L_2$  and  $L_\infty$ , so that the three locality measures are  $WL_1$ ,  $WL_2$  and  $WL_\infty$ .

Haverkort [34, 33] has compiled a collection of scanning orders that have been constructed from various space-filling curves, principally “traditional” space-filling curves like the Hilbert curve and the Peano curve, but also from more exotic curves like the  $\beta\Omega$  curve and the  $AR^2W^2$  curve, depicted in Figure 4.16 and Figure 4.17 respectively. A complete list of the scanning orders we will be considering follows:

- *GP order*. This is actually the so-called “Peano curve”, mentioned above, but we follow Haverkort in calling it the “GP order” so that it is not confused with other space-filling curves in the literature that are also generically called “Peano curves” (for example, in [79]). Atomic size 3.

- *Serpentine order*. A variation of the GP order. Also called the *Meander order*. Atomic size 3.
- *Luxburg Variation 2*. A variation of the GP-order. Originally defined by Luxburg [79]. Atomic size 3.
- *Meurthe order*. Originally defined by Haverkort [34]. Atomic size 3.
- *Coil order*. Also known as Luxburg Variation 2. Atomic size 3.
- *R order*. A variation of the GP-order. Atomic size 3.
- *Kochel order*. Originally defined by Haverkort [33]. Atomic size 3.
- *H order*. Constructed from the Sierpiński-Knopp curve. Since the Sierpiński-Knopp curve is defined on a triangular lattice, some modifications are necessary to adapt it to a square lattice. The theoretical properties, however, remain unchanged [52, 34]. Atomic size 2.
- *Hilbert order*. Constructed from the well-known Hilbert curve [36]. Atomic size 2.
- *Z order*. Constructed from a space-filling curve originally defined by Lebesgue [44], and widely used in computer science [26, 4]. Atomic size 2.
- $\beta\Omega$  *order*. Originally defined by Wierum [83], this order has the peculiar property of not starting or ending in any corner of the unit square, as most other orders do. Atomic size 2.
- $AR^2W^2$  *order*. Originally defined by Asano [4]. Atomic size 2.
- *Raster order*. This order is basically a horizontal line-by-line scan of the image grid, similar to the way a CRT monitor operates. This scanning order has no atomic size since it can be used to scan images with arbitrary dimensions.
- *Snake order*. This order is the same as the raster order except that instead of only scanning in one direction, this order alternates line by line between scanning horizontally from the right and horizontally from the left. Unlike the raster order it forms a continuous curve. The technical name for this order is the *baustrophedonic order* [30]. Just like the raster order, this scanning order has no atomic size and can be used to scan images with arbitrary dimensions.

#### 4.3.3.3 Scanning Order Total Variation

The total variation of data is a reliable indicator of the compressibility of that data: low total variation correlates with higher compressibility, and vice versa [23, 18]. Therefore it is important to characterize scanning orders by the total variation of the sequences they produce. We will use images from the Image Test Set, described in section 4.3.1.1, as a source of images.

It will also be interesting to see to what extent the theoretical locality measures  $WL_r$  of the scanning orders under consideration are correlated with

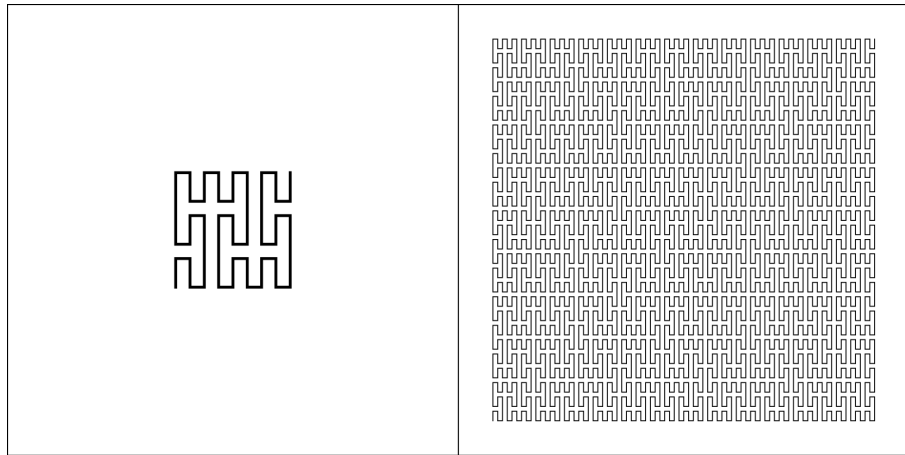


Figure 4.6: GP scanning order.

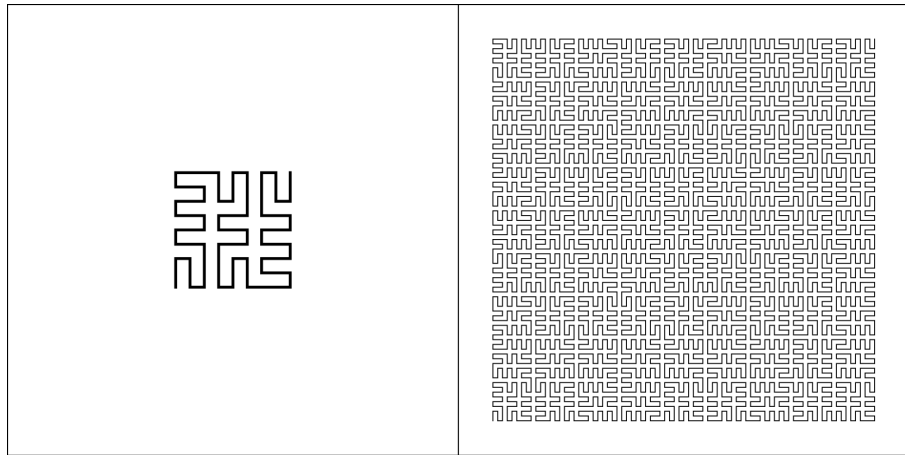


Figure 4.7: Serpentine scanning order.

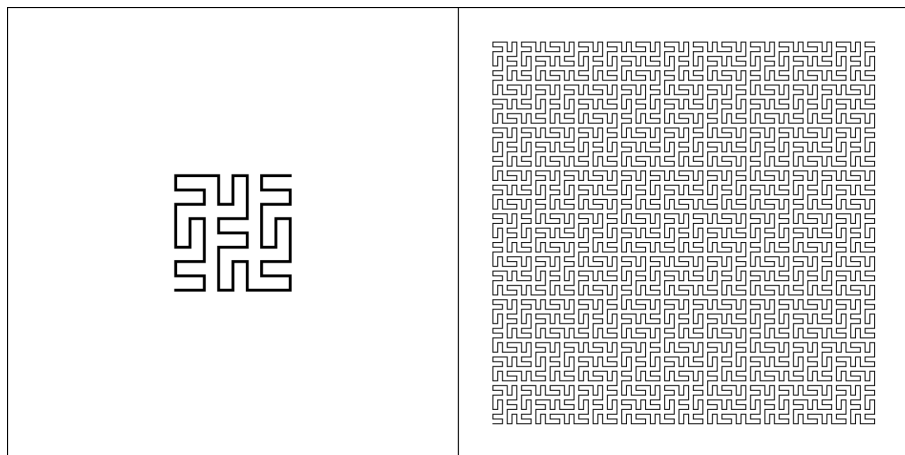


Figure 4.8: Luxburg Variation 2 scanning order.



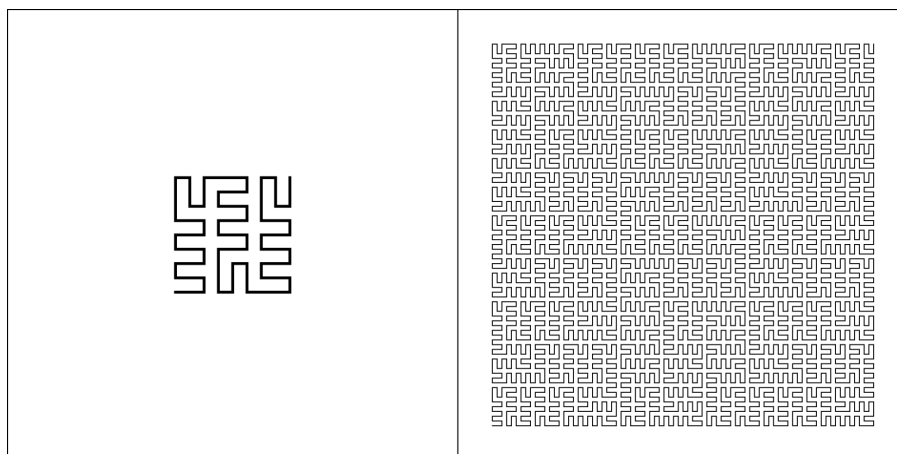


Figure 4.9: Meurthe scanning order.

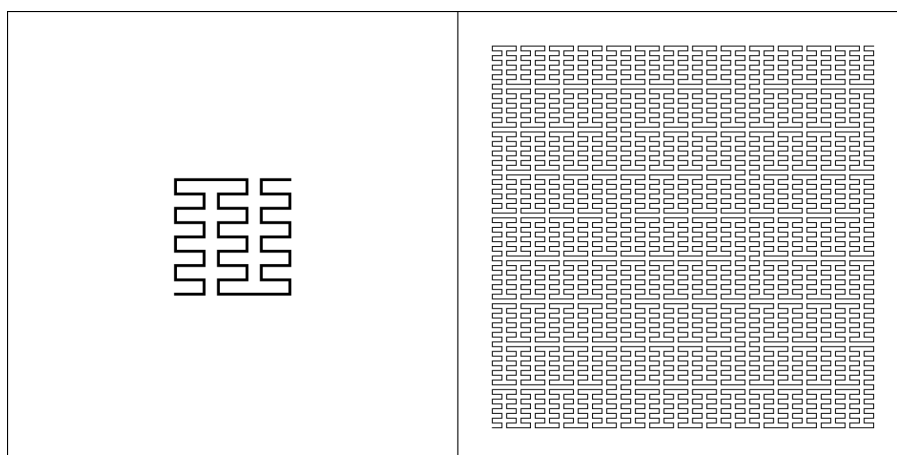


Figure 4.10: Coil scanning order.

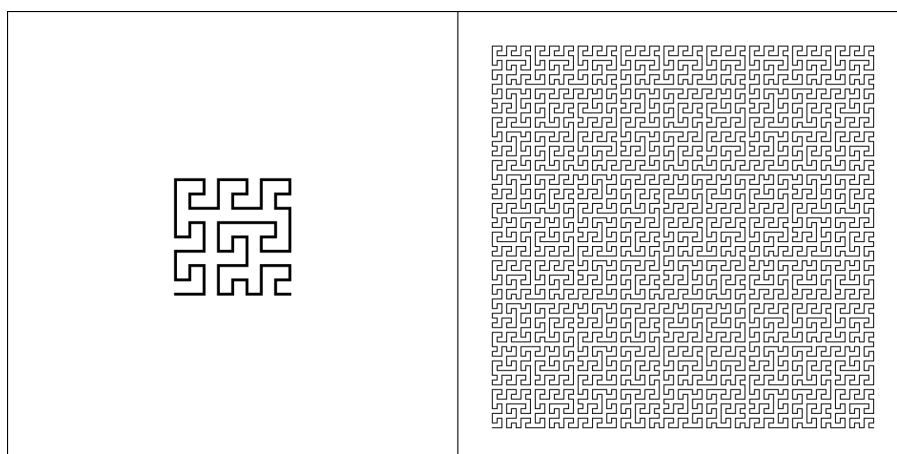


Figure 4.11: R scanning order.

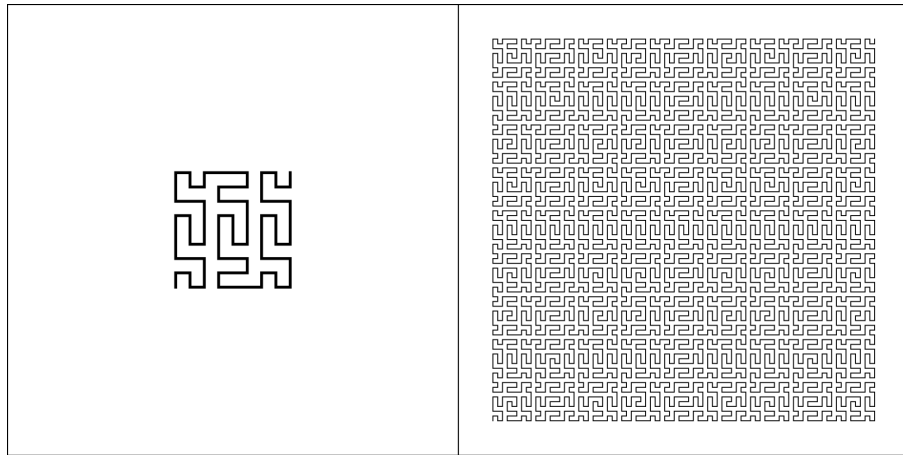


Figure 4.12: Kochel scanning order.

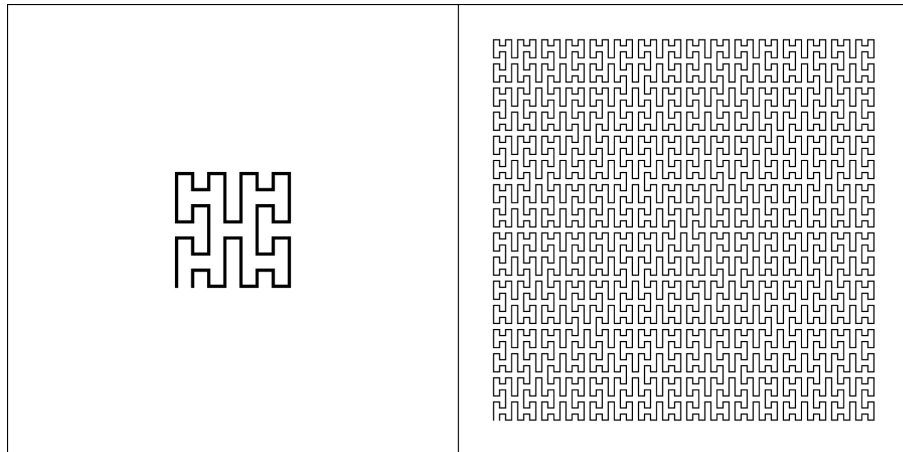


Figure 4.13: H scanning order.

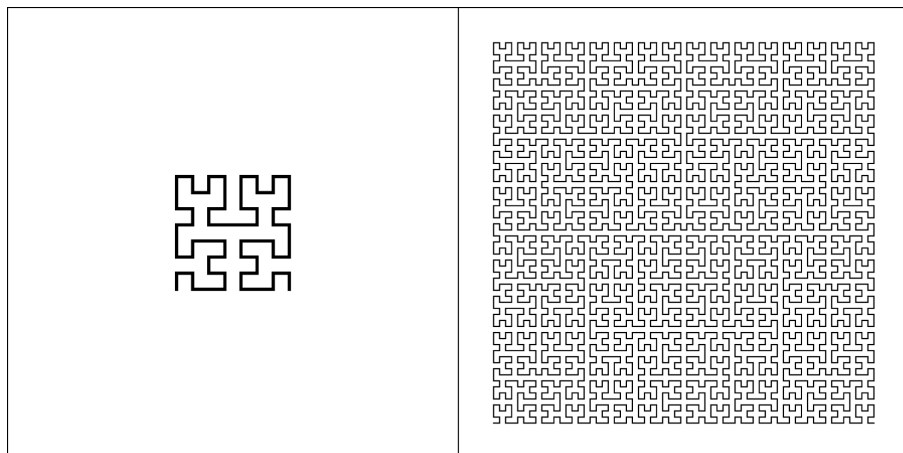


Figure 4.14: Hilbert scanning order.

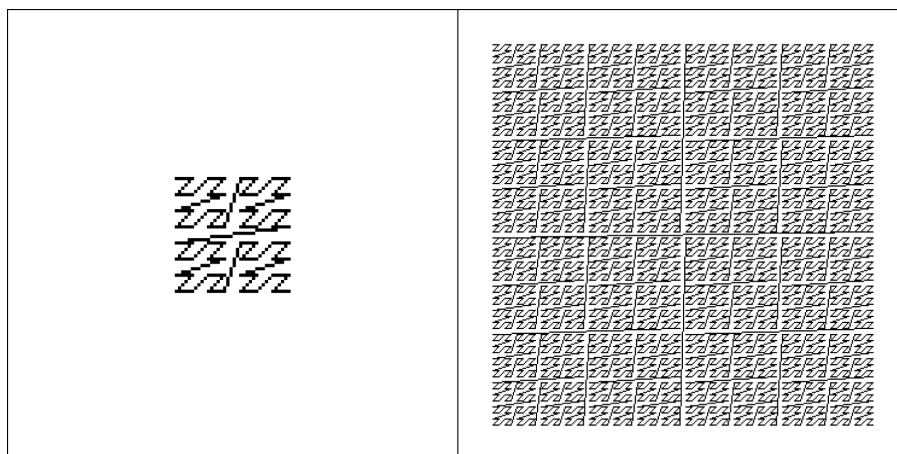


Figure 4.15: Z scanning order.

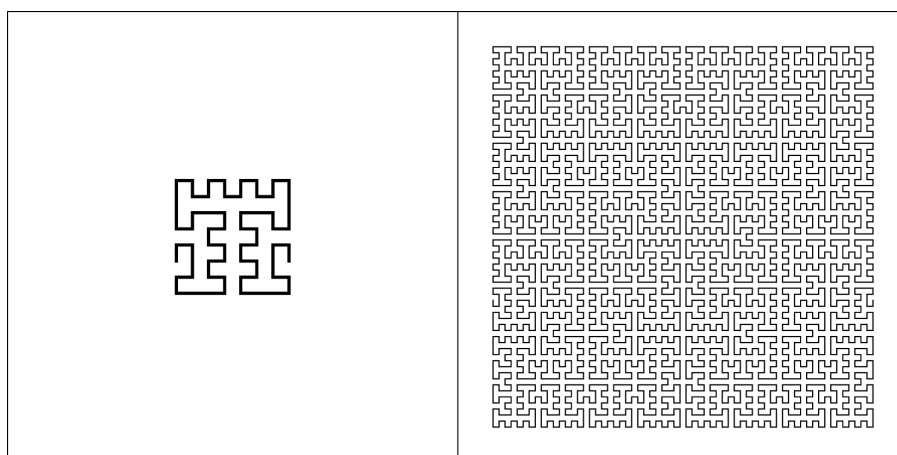


Figure 4.16:  $\beta\Omega$  scanning order.

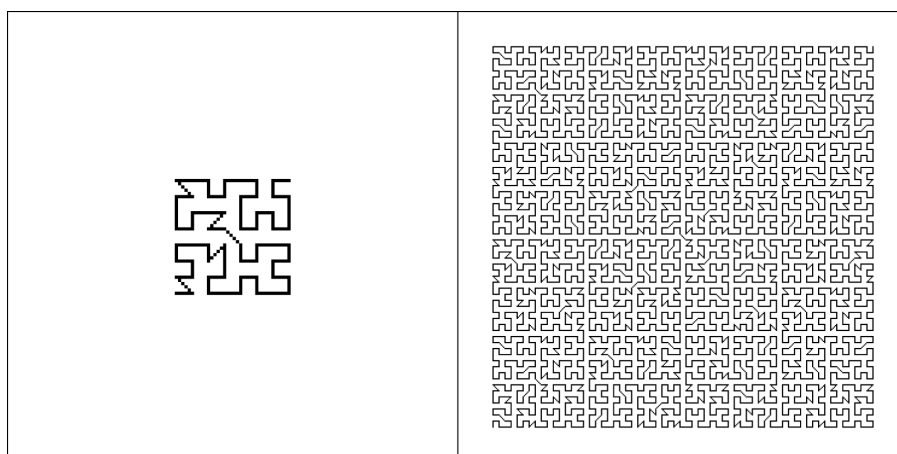


Figure 4.17:  $AR^2W^2$  scanning order.

their empirical total variations. The locality measures  $WL_r$  of some<sup>3</sup> of the scanning orders we will be testing appear in [34]; they are duplicated in Table 4.1. The values of some of these locality measures were found analytically whereas others were found numerically, and a value with a decimal expansion indicates that that particular measure was found numerically. Note that the H order has the lowest  $WL_r$  values indicating that it has the best theoretical locality. This is because of the fact that the H order is optimally locality-preserving in two dimensions [52].

Scanning Order	$WL_1$	$WL_2$	$WL_\infty$
GP	$10\frac{2}{3}$	8	8
Serpentine	10.000	6.250	5.625
Luxburg 2	10	$6\frac{1}{4}$	$5\frac{5}{8}$
Meurthe	10.667	5.667	5.333
Coil	$10\frac{2}{3}$	$6\frac{2}{3}$	$6\frac{2}{3}$
H	8	4	4
Hilbert	9	6	6
Z	$\infty$	$\infty$	$\infty$
$\beta\Omega$	9.000	5.000	5.000
$AR^2W^2$	12.000	6.046	5.400

Table 4.1: Locality measures of various scanning orders (adapted from [34]).

The procedure is as follows. Since scanning orders with different atomic sizes are being considered, two sets of image dimensions have to be found that are comparable in size and each compatible with either the scanning orders of atomic size 2 or 3. The first and only pair of numbers from the series  $2^i$  and  $3^i$  that are reasonably close to each other is  $2^8 = 256$  and  $3^5 = 243$ . These two numbers are within 5% of each other and any number pair that comes closer is too large to be practical as the dimensions of an image.

Two image sets were subsequently formed by resizing (using resampling [76] for greater accuracy) the images from the Image Test Set: the first set consisted of images of size  $256 \times 256$  and the other consisted of images of size  $243 \times 243$ .

The images were scanned using the various scanning orders mentioned earlier. Scanning orders with atomic size 2 were used to scan the  $256 \times 256$  images and scanning orders with atomic size 3 were used to scan the  $243 \times 243$  images. For each scanning order, two images were scanned: the first was the regular unrotated image and the second was the image rotated clockwise by 90 degrees. The total variation of each resulting sequence was divided by the pixel count of the source image as an attempt to normalize the result so that the results of the two sets could be compared to each other. The Raster and Snake orders (which have no atomic sizes, so that they are compatible with any image dimensions) were run on both image sets to test the theory that normalizing the results should negate the effect of the different image sizes. Tables 4.2 and 4.3 show the results.

<sup>3</sup>The scanning orders for which we do not have locality measures are: the R order, Kochel order, Snake order and Raster order.

Scanning Order	Average	Minimum	Maximum
H	9.418	0.159	54.528
H (rotated)	9.344	0.161	54.697
Hilbert	9.381	0.158	54.648
Hilbert (rotated)	9.381	0.161	54.447
Z	11.672	0.251	66.058
Z (rotated)	11.885	0.247	66.190
$\beta\Omega$	9.386	0.161	54.470
$\beta\Omega$ (rotated)	9.384	0.160	54.482
$AR^2W^2$	9.582	0.162	55.759
$AR^2W^2$ (rotated)	9.584	0.164	55.793
Raster	9.417	0.211	60.798
Raster (rotated)	9.7202	0.159	54.178
Snake	9.268	0.161	60.616
Snake (rotated)	9.505	0.158	53.689

Table 4.2: Total variation per pixel for scanning orders of atomic size 2.

Scanning Order	Average	Minimum	Maximum
GP	8.7833	0.1638	52.9205
GP (rotated)	8.6505	0.1651	53.5537
Serpentine	8.7240	0.1673	53.3043
Serpentine (rotated)	8.7229	0.1647	53.2892
Luxburg Variation 2	8.7176	0.1656	53.2152
Luxburg Variation 2 (rotated)	8.7152	0.1642	53.2567
Meurthe	8.7224	0.1653	53.2839
Meurthe (rotated)	8.7183	0.1648	53.2557
Coil	8.7713	0.1646	53.0111
Coil (rotated)	8.6635	0.1653	53.5117
R	8.7166	0.1637	53.3199
R (rotated)	8.7148	0.1645	53.1732
Kochel	8.7126	0.1635	53.2890
Kochel (rotated)	8.7128	0.1637	53.2959
Raster	8.7374	0.2187	55.4347
Raster (rotated)	9.0645	0.1651	52.7610
Snake	8.5797	0.1657	55.2420
Snake (rotated)	8.8384	0.1634	52.3346

Table 4.3: Total variation per pixel for scanning orders of atomic size 3.

From the results it is clear that normalizing the variations does not produce the desired effect: the average relative difference between the values of the Raster and Snake order for the  $256 \times 256$  images are about 7%. This discrepancy, while small, prohibits us from directly comparing the efficiency of scanning orders of atomic size 2 and 3, and is most likely an artifact of the resizing and resampling operations applied to the original images. For now we will consider the results of the two image sets separately.

Table 4.4 and Table 4.5 shows the scanning orders that have the lowest average total variation for the  $256 \times 256$  and  $243 \times 243$  image sets respectively. With both image sets the Snake scanning order has the lowest average total variation. Unfortunately the maximum total variation of the Snake scanning order is among the worst of both image sets. Worst-case performance is important when it comes to image compression, so that we will have to discount the Snake scanning order.

For scanning orders of atomic size 2 — and disregarding the Snake scanning order — the H scanning order is the best scanning order, with the Hilbert and  $\beta\Omega$  scanning orders coming second and third respectively, although it has to be mentioned that these two orders have practically the same performance. All of these scanning orders have some of the best maximum total variations as well.

For scanning orders of atomic size 3 — again disregarding the Snake scanning order — the GP order is the best, with the Coil and Kochel scanning orders coming second and third respectively. These scanning orders have some of the best worst-case performance as well.

Looking at the order of the best performing scanning orders of atomic size 2 and 3 and comparing this order to the theoretical locality measures of Table 4.1, one notes an interesting pattern: the  $WL_1$  locality measure is the only measure with an order that approximately correlates with the total variation performance, so that one might conjecture that the  $WL_1$  locality measure is a good indicator of total variation performance. The lower the  $WL_1$  locality of a scanning order, the lower the average total variation that it produces will be. Supposing that this observation is correct, we now have a way to compare scanning orders of differing atomic sizes: compare their  $WL_1$  locality values. According to the data in Table 4.1 the scanning orders of atomic size 2 are all more efficient than the scanning orders of atomic size 3; in particular, the H order is the most efficient. Incidentally the H scanning order is also optimally local in two dimensions, as mentioned previously.

Scanning Order	Avg. Total Var.
Snake	9.268
H (rotated)	9.344
Hilbert (rotated)	9.381
Hilbert	9.382
$\beta\Omega$ (rotated)	9.384

Table 4.4: Scanning orders of atomic size 2 with lowest average variation.

Scanning Order	Avg. Total Var.
Snake	8.579
GP (rotated)	8.650
Coil (rotated)	8.663
Kochel	8.712
Kochel (rotated)	8.713

Table 4.5: Scanning orders of atomic size 3 with lowest average variation.

#### 4.3.3.4 Scanning Order Isotropy

The *isotropy* of a scanning order will also be important for our purposes. A scanning order is isotropic if it is not directionally biased. There are two reasons why we prefer isotropic scanning orders over anisotropic scanning orders. The first reason is that we do not want our image compressors to have directional bias, i.e. we want them to be as general as possible. The second reason is that the degradation of image quality under quantization is less severe with isotropic scanning orders than with anisotropic scanning orders. This will be discussed when we come to designing a lossy compressor in Section 4.5.

Operationally it is easier to define and measure the complementary concept of *anisotropy*. Anisotropy can be measured as follows:

$$\Lambda := \left| 1 - \frac{v_u}{v_r} \right| \quad (4.3)$$

where  $v_u$  is the total variation of the unrotated image and  $v_r$  is the total variation of the rotated image. A scanning order that is perfectly isotropic will have an anisotropy value of  $\Lambda = 0$ .

The anisotropy values for the best scanning orders from the previous section were calculated empirically and are given in Table 4.6 and Table 4.7. The results seem to agree with the intuition formed when merely looking at the depictions of the scanning orders. For scanning orders of atomic size 2 the Hilbert order is the most isotropic<sup>4</sup>, with the H order and Z order being the least isotropic. For scanning orders of atomic size 3 the Kochel order has the best isotropy, with the Coil and GP orders having the worst isotropy.

Taking both total variation and isotropy into account, we conclude that the Hilbert and  $\beta\Omega$  scanning orders are both tied for first place as our scanning order of choice. We also have to mention that if our choice was restricted to only scanning orders of atomic size 3 then the Kochel scanning order would have been our first choice, given its good total variation performance and its extremely high isotropy, in contrast to the GP and Coil scanning orders.

#### 4.3.3.5 Image Anisotropy

The majority of images intended for human consumption have a “preferred orientation”, i.e. a specific rotation of the image that just seems natural. Pho-

<sup>4</sup>In fact it is optimally isotropic for a scanning order with atomic size 2 [69]. The number of horizontal steps it takes will always be one less than the number of vertical steps it takes so that in the limit it becomes perfectly isotropic.

Scanning Order	Anisotropy
Hilbert	0.0000271
$\beta\Omega$	0.000207
$AR^2W^2$	0.000276
H	0.00793
Z	0.0179

Table 4.6: Anisotropy for scanning orders of atomic size 2.

Scanning Order	Anisotropy
Kochel	0.0000248
Serpentine	0.000127
R	0.000199
Luxburg Variation 2	0.000280
Meurthe	0.000474
Coil	0.0124
GP	0.0153

Table 4.7: Anisotropy for scanning orders of atomic size 3.

tographs of a landscape, for instance, are always presented with the horizon of the landscape in a horizontal orientation; a landscape photograph with a vertical horizon would look out of place, even though the image theoretically contains the same amount of information as the photograph with the horizontal orientation.

A DPT might be able to detect this “preferred orientation”. The theory is that the pulse count of a DPT of a specific orientation is indicative of the “complexity” of that orientation: a lower pulse count indicates a lower complexity, and vice versa. Furthermore, we identify low complexity images with “natural” images. To test this theory, we performed the following experiment.

The procedure was as follows: test images were all cropped circularly so that all of the image content would stay within the image frame upon rotation. Then, for each image, the image was rotated successively through 360 degrees in increments of one degree. For each rotation the image was scanned with the Hilbert scanning order and the Snake scanning order respectively. A DPT was then performed on each of these sequences, and the pulse count (i.e. the total number of pulses in all of the DPT resolution levels) were recorded for that angle, and that scanning order. The C operator was used for the DPT decomposition.

Four images were selected to test this theory. Two of the images have a definite (subjective) preferred direction; these two are the Landscape image (Figure 4.18) and the Portrait image (Figure 4.19). The preferred orientation of the third image, the Clouds image (Figure 4.20), is debatable. The last image is an image consisting of random, uniform, black and white noise (Figure 4.21), with no preferred orientation.

The results of the experiments are shown in Figures 4.18, 4.19, 4.20, 4.21. Intuitively, if the theory explained above is correct, all of the pulse count per angle of rotation graphs should essentially be flat, albeit noisy, since the



Hilbert scanning order has no preferred direction, so that images with a definite directional bias would not appear different than images with no such bias. This is not exactly the case however; the two images with a preferred direction (Figures 4.18 and 4.19) show a strong periodicity in their graphs. There might be two reasons for this: the first is that it is an artifact of the rotation algorithm, since a single pixel that is rotated by, say, 45 degrees will not map to exactly one pixel, and this artifact pushes up the pulse count for *all* images, resulting in a kind of sinusoidal noise. This theory is supported by the fact that the graph peaks at 45 degrees, exactly the angle at which this effect would be at a maximum. The second possible reason is that image bias might in fact play a role with the highly isotropic Hilbert scanning order, albeit a very small role. To support this observation, it will suffice to point out that the range of the pulse counts of the Hilbert scanning order graph is about 3000 in Figure 4.18, whereas the range of the Snake scanning order graph is 20000, almost ten times larger.

Another pattern in the results is a consistent spike with a magnitude of around 1000 occurring at angles that are a multiple of 90 degrees in all of the graphs. The constant nature of the spike and the fact that it occurs at multiples of 90 degrees leads us to hypothesize that it is once again an artifact of the rotation algorithm; specifically the effect of anti-aliasing (i.e. smoothing of edges) occurring at the edge of the image circle as it transitions into the black background. At angles that are a multiple of 90 degrees there is no anti-aliasing effect because of the nature of the rotation algorithm, whereas at other angles the upsampling stage of the rotation algorithm causes anti-aliasing at the edges of the image circle, causing a drop in the contribution to the total variation of that circular edge, leading to the aforementioned spike.

For both the images with a preferred orientation the pulse count per angle of rotation of the Snake scan reveals that the subjective orientation (i.e. 0 degrees) has the lowest DPT pulse count, supporting the theory. The Clouds image, for which the preferred direction is superficially unclear, actually also shows a preferred direction at 0 degrees, which could make sense since the photo was taken horizontally, and the clouds have a slight horizontal bias. The two graphs of the Noise image are almost exactly the same, reflecting the fact that this image has no preferred direction.

From this cursory investigation we can draw the tentative conclusion that the pulse count of a DPT is in fact an indication of the complexity and “naturalness” of an image, in the sense of indicating a preferred direction.

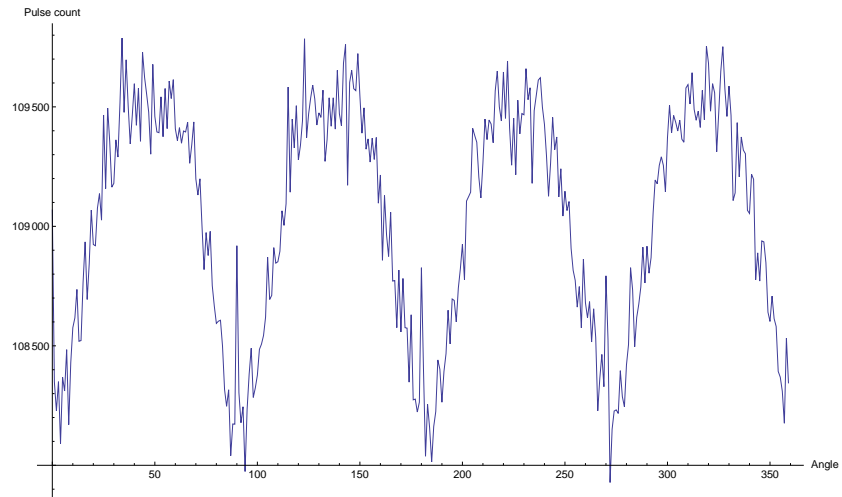
#### 4.3.4 Image Tiling

All of the image compression schemes developed here will make use of *image tiling*. With image tiling the image is divided into smaller sub-images, or *tiles*, that are then compressed independently of each other and only re-assembled at the decompression stage. The primary reason for using image tiling is to reduce the amount of computational resources<sup>5</sup> required to implement the compression scheme. Image tiling is a fairly common technique to mitigate computational load; both the JPEG standards use it [80, 15], for instance. Image tiling is illustrated in Figure 4.22.

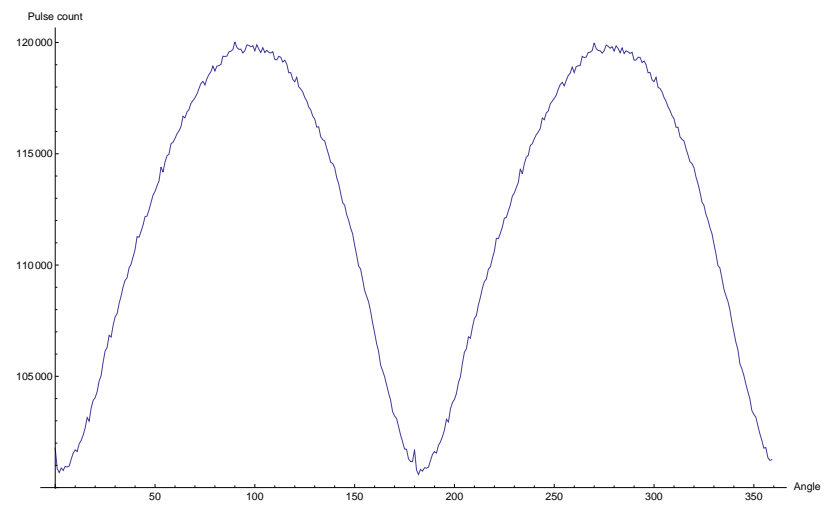
<sup>5</sup>This includes both time (processor speed) and space (memory capacity).



(a) Input image.



(b) DPT pulse counts per angle of rotation for image scanned with Hilbert scanning order.

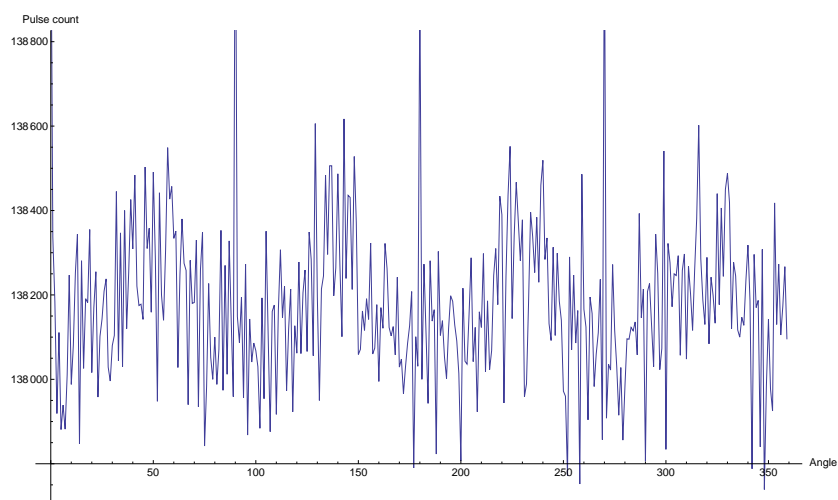


(c) DPT pulse counts per angle of rotation for image scanned with Snake scanning order.

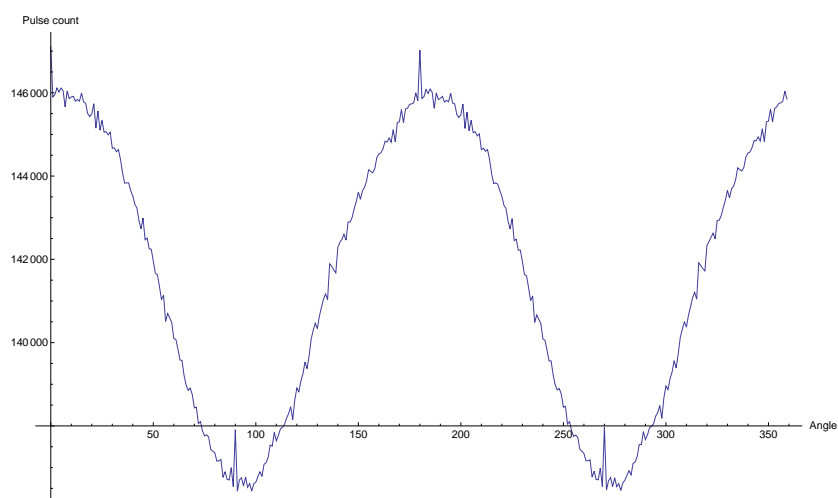
Figure 4.18: DPT anisotropy for the Landscape image.



(a) Input image.



(b) DPT pulse counts per angle of rotation for image scanned with Hilbert scanning order.

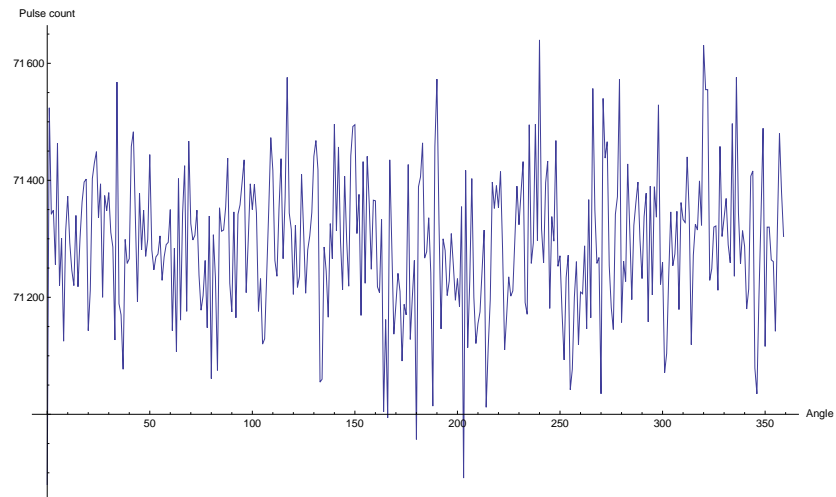


(c) DPT pulse counts per angle of rotation for image scanned with Snake scanning order.

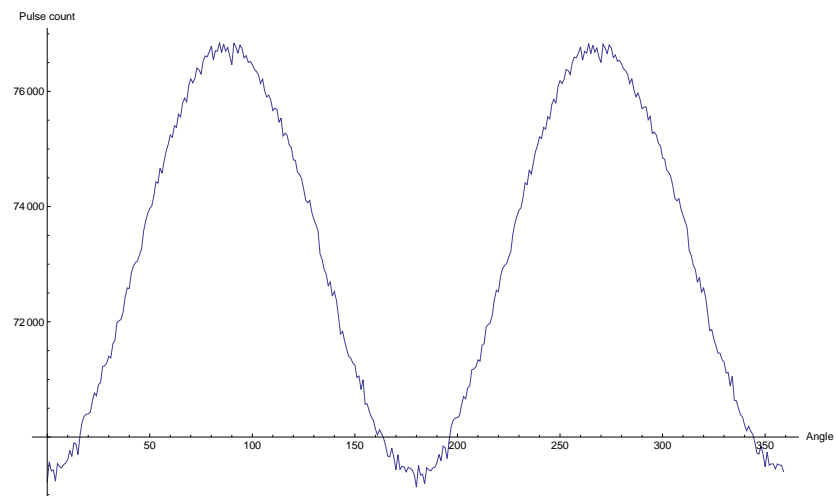
Figure 4.19: DPT anisotropy for the Portrait image.



(a) Input image.

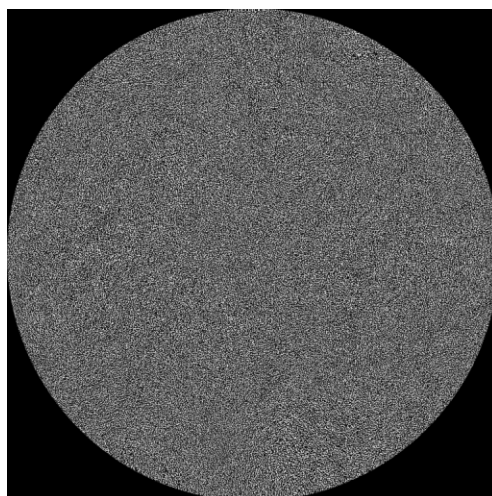


(b) DPT pulse counts per angle of rotation for image scanned with Hilbert scanning order.

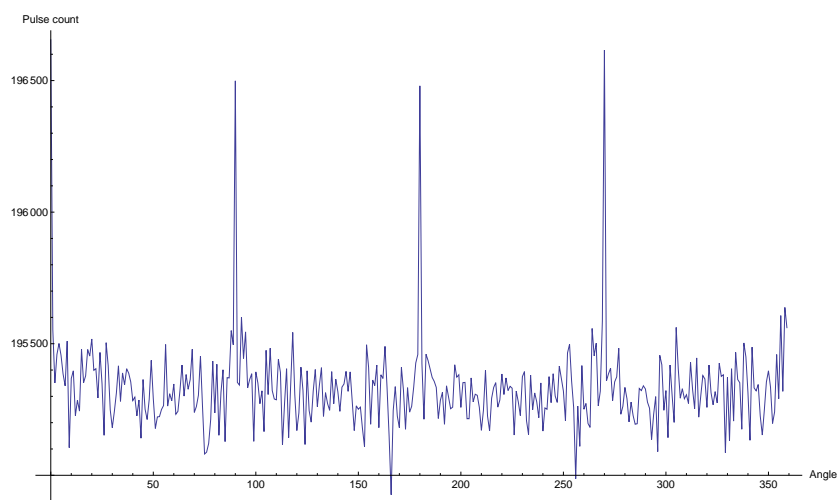


(c) DPT pulse counts per angle of rotation for image scanned with Snake scanning order.

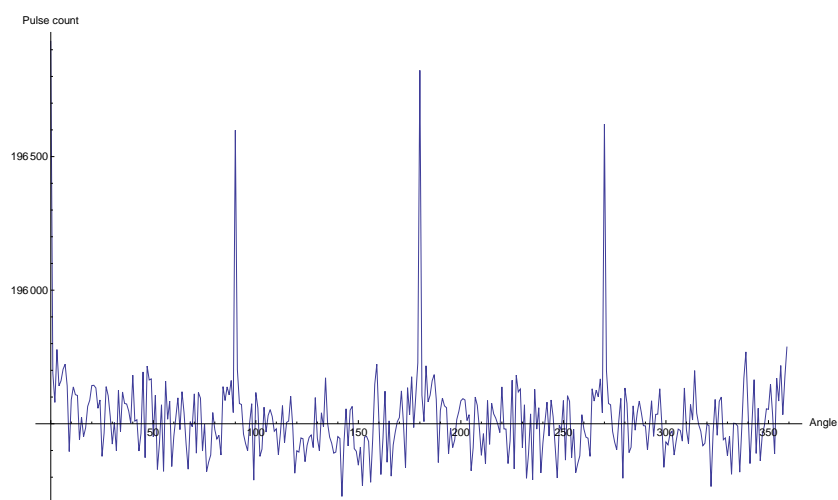
Figure 4.20: DPT anisotropy for the Clouds image.



(a) Input image.



(b) DPT pulse counts per angle of rotation for image scanned with Hilbert scanning order.



(c) DPT pulse counts per angle of rotation for image scanned with Snake scanning order.

Figure 4.21: DPT anisotropy for the Noise image.

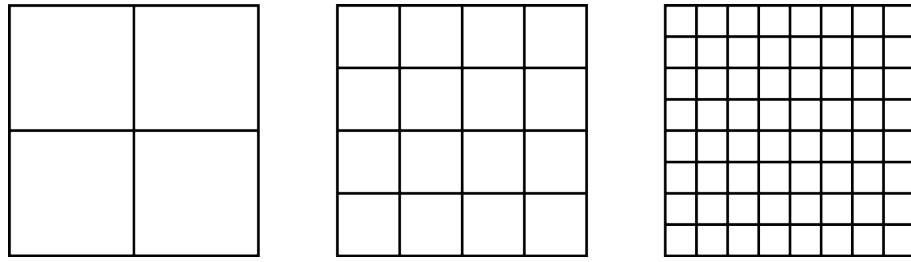


Figure 4.22: Tiling an image with progressively smaller tiles.

## 4.4 Lossless Compression

### 4.4.1 PulseTrain Scheme

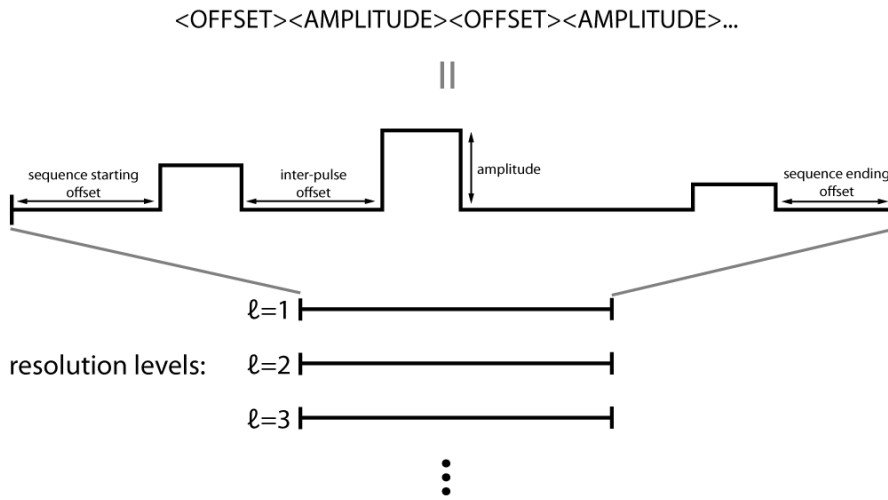
The PulseTrain scheme is a simple compression scheme that aims to compress the resolution levels produced by the DPT directly, that is, without any further processing.

#### 4.4.1.1 Modelling

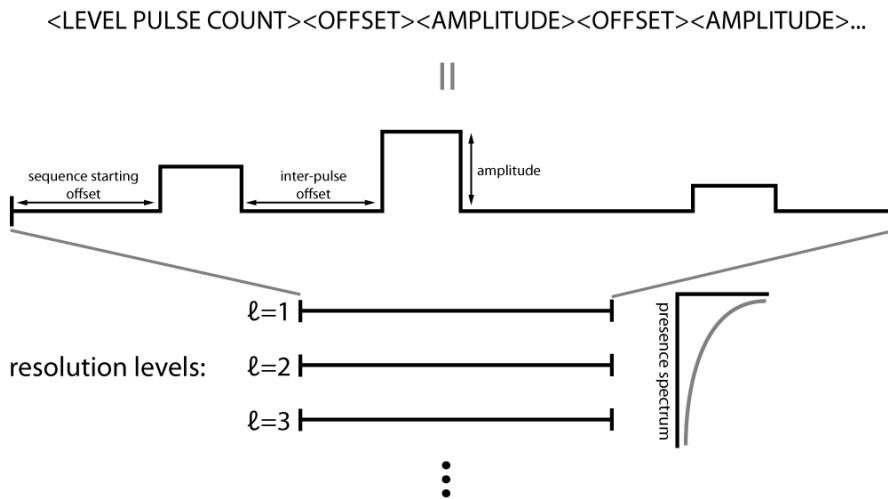
This section will discuss the basic model and variations thereupon for the PulseTrain compression scheme. For each model the theoretical lower bounds (i.e. the best-case) on the average compression ratio will be estimated by calculating the average entropy of its output when the model is run on the Image Test Set (see section 4.3.1.1). Actual compression ratios will be calculated using various coding schemes as discussed in Section 3.4. The three PulseTrain models are illustrated in Figure 4.23.

**Basic Model** The basic model for the PulseTrain scheme is as follows. Every resolution pulse has an *amplitude* and an *offset*, which is the distance between the end of the previous pulse and the start of the current pulse. If the pulse happens to be the first pulse, then the offset is the distance between the start of the sequence and the start of the pulse. These values are encoded directly, with the provision that for the last pulse in a resolution level, another offset is added, equal to the distance between the end of the last pulse and the end of the resolution level. This is necessary because otherwise the decoder will not know when it is finished with a resolution level. Note that the pulse width is not encoded since the decoder will always know with which resolution level it is working, so that it automatically knows the pulse widths of all the pulses on that resolution level.

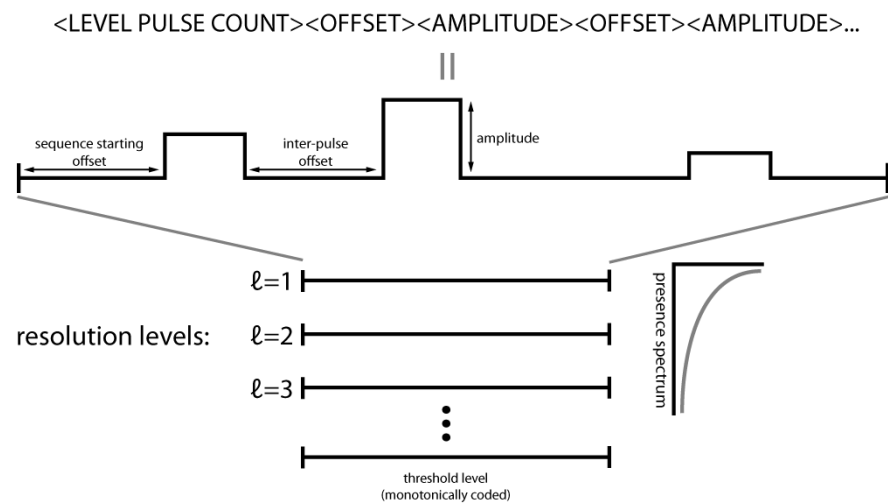
The basic model compression algorithm was ran on the Image Test Set with various tile sizes, scanning orders and DPT operators. Figures 4.24 and 4.25 show detailed results for the cases where the *L* and *C* DPT operator were used respectively. From the results it is clear that for both operators, a larger tile size is better, except when using the Snake scanning order, which actually worsens as the tile size increases with the *C* operator. Both operators have a marked peak of inefficiency at tile size  $8 \times 8$ . It can also be seen that the best scanning orders to use are the H order, Hilbert order and  $\beta\Omega$  order. These



(a) PulseTrain basic model.



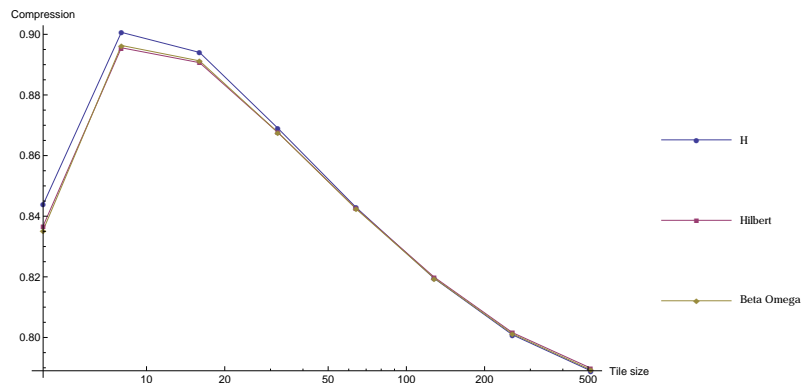
(b) PulseTrain pulse prediction model.



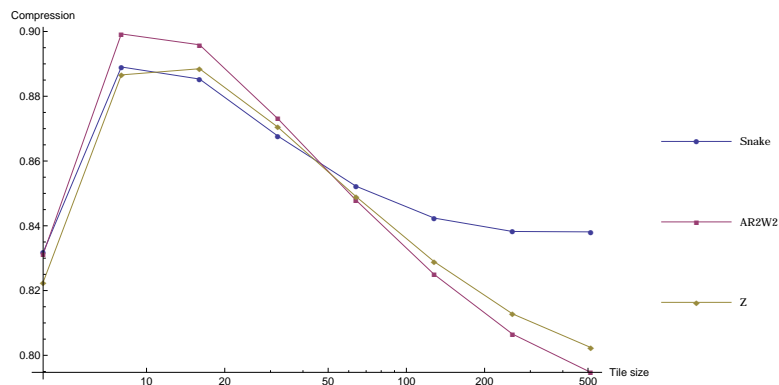
(c) PulseTrain threshold model.

Figure 4.23: The three PulseTrain models.

three scanning orders produce virtually the same amount of compression, but strictly speaking the  $\beta\Omega$  order is optimal for tile sizes  $32 \times 32$  to  $128 \times 128$ , with the H order producing only marginally better results for tile sizes  $256 \times 256$  and  $512 \times 512$ . Table 4.8 shows the compression factors for various DPT operators scanned using the H order with tile size  $512 \times 512$ . It is clear that the basic model is not very sensitive to the choice of DPT operator. The  $F$  operator performs the best, however, even if its improvement over the other operators is only marginal.



(a)



(b)

Figure 4.24: Basic lossless model compression factors for various tile sizes and scanning orders with DPT operator  $L$ .

**Pulse Count Prediction** Recall that in the basic model an extra offset is added to the end of a resolution level to signal to the decoder that the current resolution level is finished. If the pulse count for a resolution level were known, this would be unnecessary — the decoder would then keep a count of decoded pulses in a resolution level, and when this count is equal to the pulse count for that resolution level, it would know that there are no more pulses for that level. This saves one offset per resolution level, but it also adds the cost of



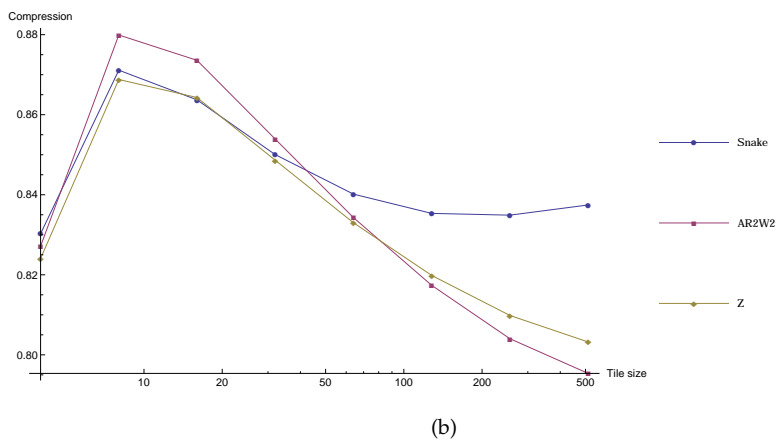
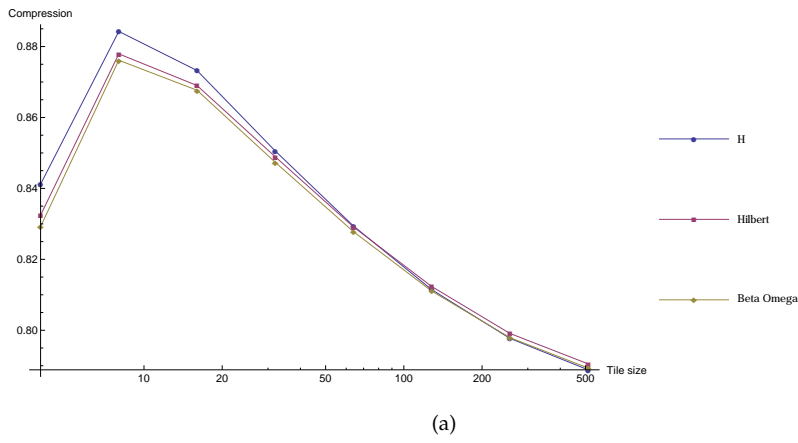


Figure 4.25: Basic lossless model compression factors for various tile sizes and scanning orders with DPT operator  $C$

DPT operator	Compression Factor
$C$	0.789
$F$	0.788
$L$	0.789

Table 4.8: Basic lossless model compression factors for various DPT operators with tile size  $512 \times 512$  and scanned with the H order.

encoding the pulse count at the start of each resolution level. The success of this scheme will depend on how economically the resolution level pulse counts can be encoded.

All pulse count information appears in the *presence spectrum* (see Section 2.4.3), which fortunately contains a fair amount of structure, increasing the chances of economical coding.

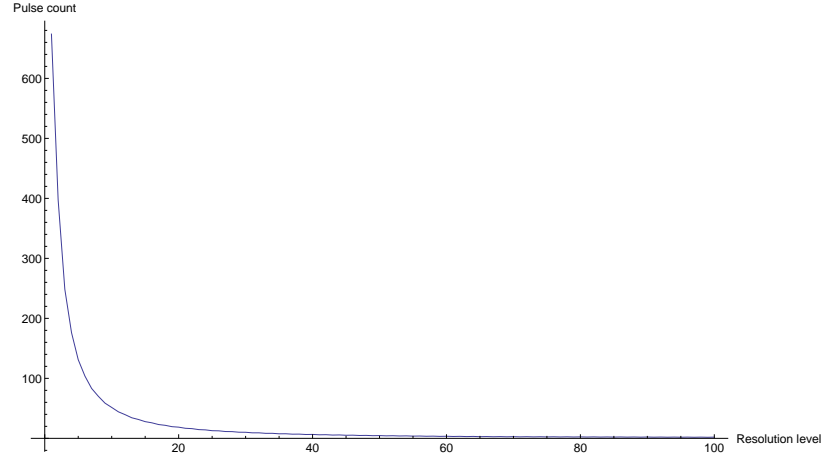


Figure 4.26: Typical DPT presence spectrum.

Ideally we would like to be able to predict the presence spectrum for any sequence of length  $N$  obtained from an image source with a fair amount of accuracy; the more accurate the prediction, the more economical the coding of the pulse counts will be. Thus we are looking for a function  $\hat{\pi}_\ell$ , predicting the pulse count for level  $\ell$  for a sequence of length  $N$ .

To obtain this prediction, average presence spectra were generated from the Image Test Set, using tile sizes that ranged from  $8 \times 8$  to  $512 \times 512$ , in powers of two. The tile sizes determined the sequence length  $N$ , which is equal to the square of the tile size. For each tile size a representative presence spectrum was formed, averaging over all the tiles of all 1000 test images. The following function was then fit to this representative presence spectrum:

$$\hat{\pi}_\ell = a\ell^{-1} + b\ell^{-2} + c\ell^{-3} + d\ell^{-4} + e\ell^{-5} + f\ell^{-6} + g\ell^{-7} \quad (4.4)$$

Using more than seven orders in this rational function causes the accuracy of the prediction to worsen in both the  $L_2$  and  $L_\infty$  norms.

We would like a function that will predict the pulse counts for any sequence length  $N$ ; we introduce this variable by factoring it out as follows:

$$\hat{\pi}_\ell = N(a'\ell^{-1} + b'\ell^{-2} + c'\ell^{-3} + d'\ell^{-4} + e'\ell^{-5} + f'\ell^{-6} + g'\ell^{-7}) \quad (4.5)$$

Experimentally it was determined that, in general, the parameters of the fit are dependent on two things: the scanning order used to scan the image and the DPT operator used in the decomposition. The parameters are, however, independent of the length of the sequence, so that one function can be used to predict the presence spectra of sequences produced by a particular combination of scanning order and DPT operator.

The accuracy of the fit is quite remarkable. For sequences produced by the  $\beta\Omega$  scanning order and the  $F$  DPT operator the maximum relative error and the average relative error is consistently around 40% and 10% respectively, for all tile sizes. Other combinations of scanning orders and DPT operators gave comparable results.

It must be emphasized that these fitted functions are not universal, since they were fitted to the particular data of the Image Test Set. We can expect, however, that these functions will be universal for the subset of “natural” images, insofar as the Image Test Set represents a faithful sampling of such images.

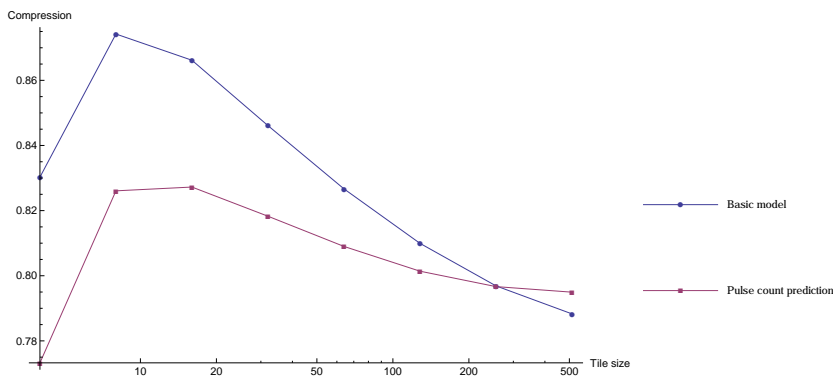


Figure 4.27: Comparison between compression factors of the basic lossless model and the pulse count prediction lossless model. In both cases the  $\beta\Omega$  scanning order and the  $F$  DPT operator was used.

Figure 4.27 show the results of the pulse count prediction model applied to images scanned with the  $\beta\Omega$  scanning order and decomposed with the  $F$  DPT operator, compared to the basic model which was ran with the same parameters. There is a significant reduction in the compression factor for tile sizes between  $4 \times 4$  and  $16 \times 16$ , whereafter the difference diminishes, until ultimately the two models converge at a tile size of  $256 \times 256$ . At tile size  $512 \times 512$  the basic model still offers the best compression, albeit by a narrow margin. Other combinations of scanning orders and DPT operators gave similar results.

A possible explanation for why the basic model offers better compression factors for big tile sizes than the pulse prediction model is that for big tile sizes the errors in the pulse count prediction start to accumulate, since there are so many of them. For instance, for tile size  $512 \times 512$  there are 262144 resolution levels, whereas for  $256 \times 256$  there are only 65536 resolution levels — 4 times less. Thus pulse count prediction is efficient for the smaller tile sizes, and is in fact optimal for the smallest tile size,  $4 \times 4$ .

**Threshold Model** The average DPT presence spectrum follows a power law (see Section 2.4.3.3), so that typically the first 3 resolution levels contain 50% of all the pulses in a DPT and the first 7 resolution levels contain 66% of all pulses [67]. This raises the question of whether it is worth the overhead to encode

all the resolution levels, as in the basic model. Figure 4.28 shows how the offset entropy increases logarithmically with resolution level, meaning that it becomes progressively more expensive to encode an offset in higher resolution levels with the basic model.

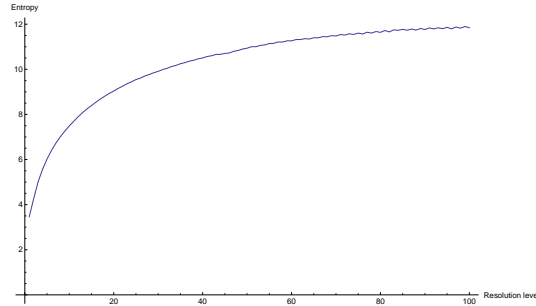


Figure 4.28: Offset entropy per resolution level for tile size 512 (first 100 levels).

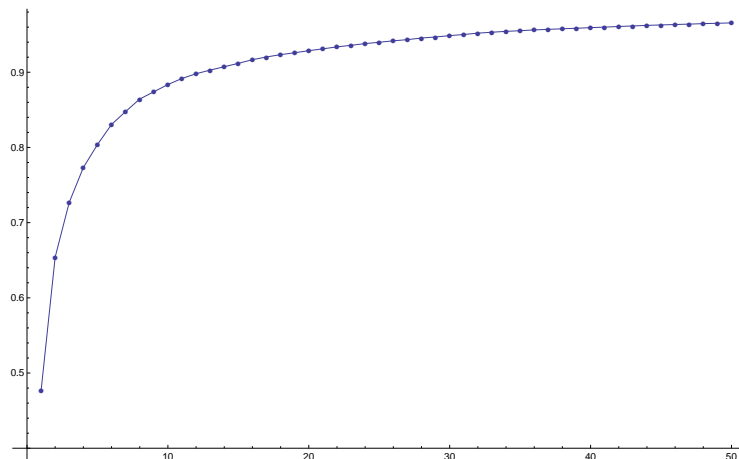


Figure 4.29: Fraction of total pulses contained in first  $n$  resolution levels (determined experimentally).

The threshold model aims to solve this problem by introducing a *threshold level* with index  $\ell = t$  beyond which no more resolution pulses are encoded; the threshold level is encoded instead. To regain the original sequence the resolution pulses of all resolution levels up to and including the threshold level are simply added to the base level. The base level contains certain regularities that can be exploited, the most important of which is the fact that it is  $t$ -monotone (see Section 2.4.3), so that it is well-suited to run-length encoding.

Figure 4.30 shows the results of the threshold model, using the  $\beta\Omega$  scanning order and the  $F$  DPT operator. Four threshold levels were tested: 1, 2, 3 and 13, with each instance encoding a total of 48%, 65%, 73% and 90% of the total number of pulses respectively. Unlike the basic and pulse count prediction models, the compression ratio versus tile size graph of the threshold model is

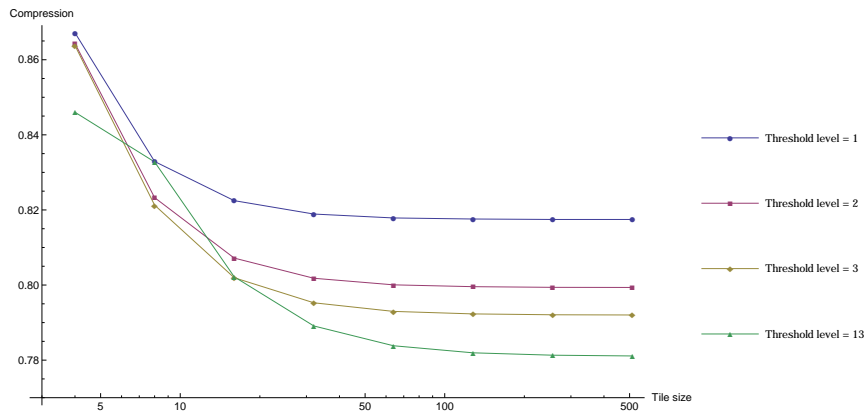


Figure 4.30: Threshold model compression factors for various tile sizes. Scanning order  $\beta\Omega$ , DPT operator  $F$ .

monotone. The performance of the threshold model seems to increase as the threshold level is raised, and with a threshold level of 3 or greater it produces better results than both the basic and pulse count prediction models.

#### 4.4.1.2 Coding

As mentioned at the beginning of the previous section, the results that have been reported thus far have been experimental measures of the *entropy* of the models as they were run images from the Image Test Set. As such they only represent *best-case* compression factors, since no coding algorithm can compress below the entropy of a source, as discussed in Section 3.4.2. It remains to see how these models perform when their results are actually coded using a coding algorithm, and it is only then that we can make our final conclusions about their performance.

**Rice Coding** Golomb coding and Rice coding were discussed in section 3.4.4. Golomb coding is essentially a code for geometrically distributed integers, and Rice coding is a scheme to improve on Golomb coding by dividing the data into blocks and finding the optimum Golomb parameter for that block.

For the PulseTrain scheme all the data to be coded can be represented by integers, but unfortunately the distributions of these integers are only approximately geometric. In general these distributions have heavier tails. For this reason we will use Rice coding instead and see if the adaptivity of this method can negate the effect of the mismatch in probability distributions.

The Rice coding algorithm was used to encode the PulseTrain basic, pulse count prediction and threshold models. The Rice blocksize used was 16, which means that the optimal Golomb parameter was chosen for every block of 16 samples. To keep computational costs under control the Rice coder chose the best Golomb parameter from a range of  $m = 1$  to  $m = 10$  — higher Golomb parameters were not evaluated. For a given block two extra options were available to the Rice coder, other than encoding it with a Golomb code. The

first option was to signal that the block was a constant value, and the second option was to use no code at all, i.e. to send the samples of the block unencoded. These “out-of-band” options for Rice coding are standard, and many implementations add many more such options [40]. In all cases, the  $\beta\Omega$  scanning order and the  $F$  DPT operator was used. The performance of the code was measured by the average redundancy and the average maximum compression factor. Redundancy measures how much larger the empirical code size is than the theoretical, optimal code size, as calculated using the entropy. The average maximum compression factor measures worst-case behaviour, something which is important for any compressor. The results are shown in Table 4.9, and it can be seen that neither the redundancy nor the average maximum compression factor is very good; the results for the Rice-coded threshold PulseTrain model is particularly bad.

Model	Redundancy	Avg. max. factor
Basic PulseTrain	55 %	2.0
Pulse count prediction PulseTrain	20 %	1.6
Threshold PulseTrain	246 %	3.6

Table 4.9: Performance of Rice coding with the lossless PulseTrain scheme.

**Arithmetic Coding** The arithmetic coding technique was discussed in section 3.5.1. The most important attribute of this technique is that it is optimal in the sense that it is theoretically possible to come arbitrarily close to the entropy of the data to be coded, with the entropy being the lower bound on compressibility. In practice, however, there are some considerations that have to be taken into account that might affect the attainability of this ideal limit.

Arithmetic coding was used to code the data produced by the basic model and the pulse prediction model. It will be recalled that arithmetic coding needs a probability distribution to operate from (see Section 3.5.1.2). There are two ways to obtain such a probability distribution. The first is to create a probability distribution from the statistics of a representative set of images, “training” the arithmetic coder, as it were, on one set of images. The second way is to update the probability as the arithmetic coder is coding; this is called *adaptive* coding. The advantage of adaptive coding over “static” coding is that the coder can adapt to the local statistics of the image (or part of the image) it is encoding. With static coding no new learning takes place, so that it can never be as fine-grained as adaptive coding.

The results of using arithmetic coding to code the various models are shown in Table 4.10. The average compression factor of the adaptive arithmetic coding consistently came within 1% of the theoretical value obtained in the previous section, confirming the theoretical optimality of the arithmetic coder. The maximum compression factors, however, are unfortunately very high, consistently being about 60% bigger than the average compression factors, attaining values as high as 1.46, which is of course unacceptable for an image compressor. Unsurprisingly the images that produced these high maximum values were found to be very complex, with high total variation. Some of these images are shown in Figure 4.31.

Several variations on the basic adaptive arithmetic coding theme were tried. Supplying the adaptive arithmetic coder with initial statistics from a representative image set (aiming to get the best of both the adaptive and static worlds), instead of starting from a uniform distribution did not make any noticeable difference to the compression factors, except for tile sizes  $256 \times 256$  and up. The reason for this is most likely that the local statistics quickly overwhelm the supplied initial statistics, except when there are too few tiles to adaptively train on, i.e. when the tile size is very big, resulting in a low tile count — with a tile size of  $256 \times 256$ , for example, there are only four tiles. In these cases of low tile count the adaptive coder essentially becomes a static coder, with the concomitant performance penalty.

The adaptivity of the adaptive arithmetic coder was also varied. The adaptivity is a measure of how responsive the coder is to changes in the underlying probability distribution. Theoretically, the adaptivity should be inversely proportional to the entropy of the underlying probability distribution; this was confirmed in experiment. Changing the adaptivity, however, did not change the result significantly. It should be possible to make the adaptivity itself adaptive. The best way to do this would probably be to estimate the entropy of the data as it is being decoded, and to adjust the adaptivity using this estimated entropy using some empirically derived formula.

Two other strategies linked to coder adaptivity was tried. When coder adaptivity comes into play, the order in which the tiles are processed can potentially make a difference. We can call the order in which the tiles are scanned the *inter-tile scanning order*, as opposed to the *intra-tile scanning order* which scans the points within the tiles themselves, as discussed in section 4.3.3. To see why the inter-tile scanning order can make a difference, consider a square image that is black in the top half and white in the bottom half. Supposing that the image is scanned from the top, using the horizontal snake scan will allow the coder to adapt optimally adapt to the black part of the image first, completely scanning it before moving on to the white part of the image and optimally adapting to that part as well. When using the vertical snake scan will result in the scan alternating between the black and white regions of the image, resulting in suboptimal adaptation of the coder. Both isotropic and anisotropic scanning orders were used for the inter-tile scanning order, but it made no significant difference.

The second strategy was to periodically reset the coder statistics, so as to erase the influence of past processed tiles, in case the statistics of the previous tiles are not really correlated with those of the new tiles; or, to give the statistics of the new tiles the opportunity to actually influence the coder's statistics. Several reset intervals were tested, for example, to reset the coder statistics every 10, 20, 100 tiles, but once again no significant difference was detected.

Model	Redundancy	Avg. max. factor
Basic PulseTrain	1 %	1.46
Pulse count prediction PulseTrain	1 %	1.10
Threshold PulseTrain	1 %	1.22

Table 4.10: Performance of arithmetic coding with the lossless PulseTrain scheme.

## 4.4.2 PulseTree Scheme

### 4.4.2.1 Modelling

The PulseTree compression scheme is designed to exploit all of the structure in the DPT, in contrast to the PulseTrain compression scheme. Among other things, the PulseTree compression scheme takes into account the key fact that the resolution pulses in the DPT do not overlap.

Whereas the PulseTrain compression scheme essentially arranged the pulses to be encoded as a list, the PulseTree compression scheme arranges the pulses as a tree<sup>6</sup>, with the widest possible resolution pulse at the root of the tree. The tree is then built recursively as follows: for every resolution pulse, the pulse appearing directly below it is considered its parent and is added as such to the tree. The resulting tree makes it easy to find the support of any resolution pulse: it is simply the parent pulse of the pulse in question. This representation should yield superior economization to the PulseTrain scheme. Figure 4.32 illustrates the basic construction of the PulseTree scheme.

In general there are four parameters that have to be encoded for every pulse in the pulse tree. They are:

- $w$ , pulse width.
- $p$ , pulse position relative to the parent pulse (or equivalently,  $\sigma$ , pulse offset relative to previous pulse).
- $a$ , pulse amplitude.
- $c$ , number of pulse children.

### 4.4.2.2 Coding

The PulseTree scheme makes it easy to incorporate a large amount of context in the coding stage. Such context is easily handled by arithmetic coding (see Section 3.5.1.2). This fact, along with the superior coding performance of arithmetic coding (as established in Section 4.4.1.2) leads us to choose it as our sole coding method for the PulseTree scheme.

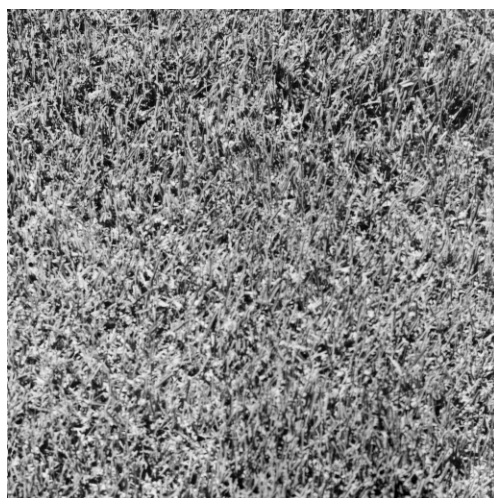
For each pulse in the pulse tree, the following context is available:

- $w_p$ , parent pulse width.
- $\sigma_p$ , parent pulse offset.
- $a_p$ , parent pulse amplitude.
- $c_p$ , number of child pulses of parent pulse (equivalently, number of siblings for current pulse).
- $a_c$ , cumulative amplitude; the sum of the amplitudes of the given pulse's ancestors.
- $\ell$ , layer the number of ancestors of the given pulse.

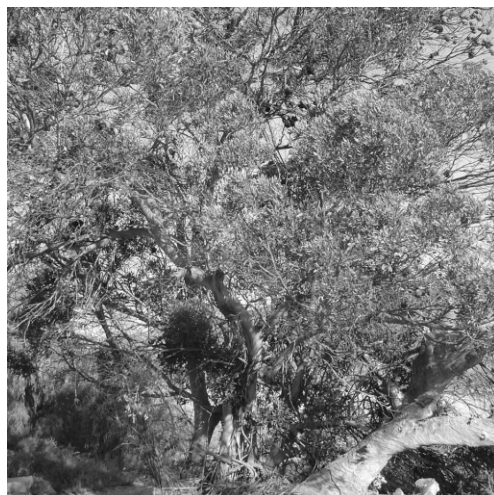
---

<sup>6</sup>As the concept is used in computer science [16].

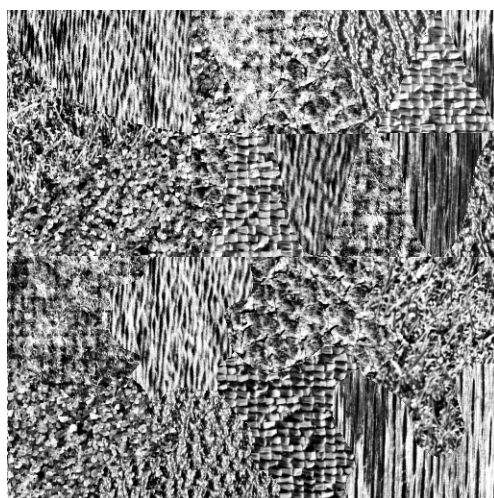




(a)



(b)



(c)

Figure 4.31: Sample worst-case images for lossless compression.

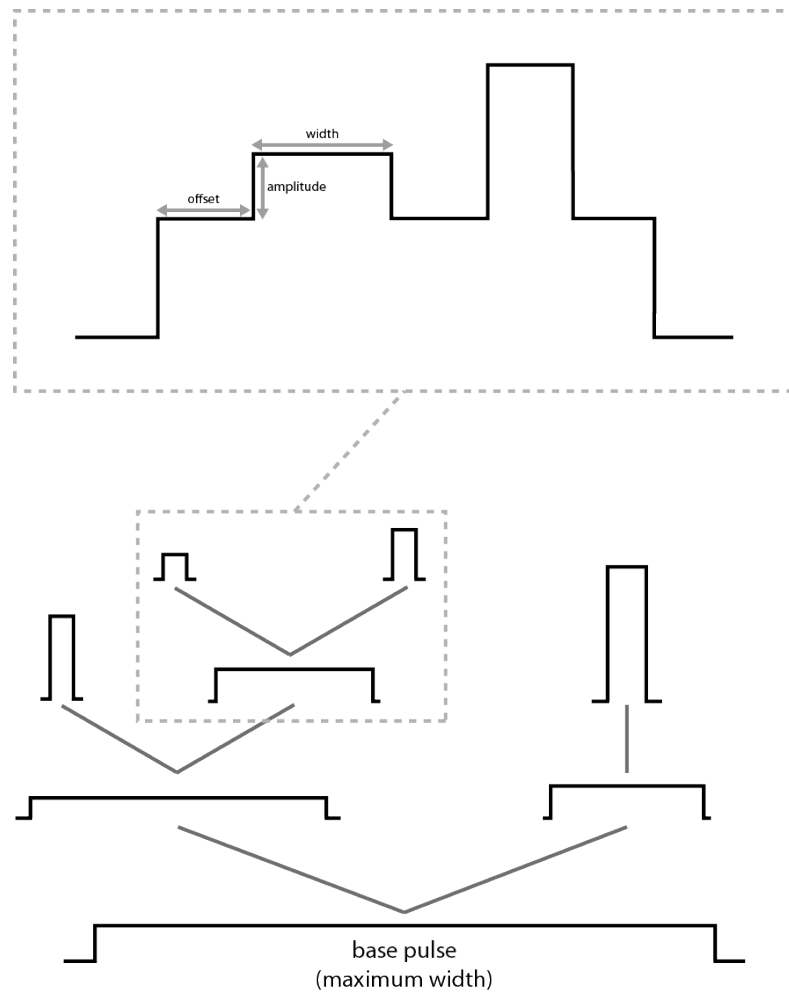


Figure 4.32: Model for the PulseTree scheme.

Along with the abovementioned context, the pulse's own parameters are also available as context, depending on the sequence in which they are decoded. For example, coding the width  $w$  first enables one to use it as context to code the amplitude using the width as context,  $p(a|w)$ .

A number of different coding schemes were tested, and we only report the two most interesting and practical schemes here. The two schemes can broadly be described to be simple and complex, respectively.

The simple scheme is to directly encode a pulse's parameters, with the exception of the amplitude, which was encoded using the cumulative amplitude  $a_c$  as context:  $p(a|a_c)$ . This was done because it was found that the amplitude  $a$  was the most expensive parameter to encode: it accounted for more than 60% of the eventual encoded size of a pulse. The space allocated to amplitude also proved very difficult to minimize, as it seemed to have very little correlation with any of the abovementioned parameters; of all the parameters, it was correlated most strongly with the cumulative amplitude  $a_c$ , and only

because the cumulative amplitude cannot exceed 255, which is the limit of the intensity of a pixel. This means that, for example, the probability that a new amplitude is 10 when the cumulative amplitude is already 250 is 0, making the coding of that parameter more efficient. The parameter that did almost as well in reducing the cost of coding an amplitude was the layer  $\ell$ , most likely for the same reason as the cumulative amplitude, but in a more indirect way, hence the loss of efficiency: a pulse in a high layer is more likely to have a correspondingly high cumulative amplitude.

There was one other optimization. The width  $w$  was encoded by encoding the error between the actual value and a prediction for the width. The prediction  $\hat{w}$  was computed as follows:  $\hat{w} = \frac{w_p}{w_c}$ . Interestingly this prediction gave better results than the more theoretically sound prediction that is based on the idea of the average width of a child pulse,  $\bar{w} = \frac{w_p}{2w_c}$ , which takes into account the offset, which is theoretically of the same order as the width.

The simple scheme was implemented using adaptive arithmetic coding without any prior statistics. Each parameter had its own arithmetic coder which was initialized only once, at the start of the compression procedure. This means that the arithmetic coder was allowed to learn from all the tiles, i.e. the whole image. The results of this scheme using the  $F$  DPT operator and the  $L$  DPT operator are shown in Table 4.11 and Table 4.12 respectively.

Unfortunately only the results for the first three tile sizes are available, since the compression algorithm took an unreasonable amount of time for larger tile sizes because of the fact that an unoptimized arithmetic coding algorithm was used — speed should not be a problem in a real-world application. This lack of experimental data raises the concern that the PulseTree scheme might actually display the same behaviour as the PulseTrain scheme, that is, having a peak in the tile size vs. compression factor graph, which for the PulseTrain scheme was consistently around a tile size of  $8 \times 8$  (see for example Figure 4.24). For the PulseTree scheme this peak might be shifted towards larger tile size values. Several (long running) experiments showed that this was not the case: the upward trend in compression factor continues all the way to the largest possible tile size of  $512 \times 512$ , at which a tile is as big as the original image.

Surprisingly the  $L$  DPT operator gave better results than the  $F$  operator. This is because of the fact that with the  $F$  operator amplitudes are more expensive to encode since they range from -255 to 255, in contrast to the  $L$  operator where the amplitude only ranges from 0 to 255. One might ask why this mechanism did not appear to play a role in the various PulseTrain schemes, where the  $F$  and  $L$  operators produced comparable results. An explanation might be that with the PulseTree scheme the two operators produce basically the same tree structure, barring amplitudes, and that this is an intrinsic consequence of the PulseTree scheme. The PulseTrain scheme is probably more sensitive to the nature of the resolution levels produced by different DPT operators.

The second scheme is the more complex one: it tries to exploit the amount of context that is available to the coder. The results obtained through this scheme were surprisingly not significantly better than those of the simple scheme: there was virtually no difference between the average compression factors of the two schemes. The maximum compression factor of the complex scheme did show an improvement, however, of about 5%. In addition to this

Tile size	Avg.	Max.	Min.
$4 \times 4$	0.855	1.30	0.0237
$8 \times 8$	0.914	1.41	0.00612
$16 \times 16$	0.936	1.44	0.00172

Table 4.11: Compression results for the simple PulseTree scheme using scanning order  $\beta\Omega$ , DPT operator  $F$ , using arithmetic coding.

Tile size	Avg.	Max.	Min.
$4 \times 4$	0.774	1.17	0.0236
$8 \times 8$	0.811	1.24	0.00603
$16 \times 16$	0.829	1.27	0.00166

Table 4.12: Compression results for the PulseTree scheme using scanning order  $\beta\Omega$ , DPT operator  $L$ , using arithmetic coding.

lack of performance, the scheme also turned out to be impractical. To see why, consider the details of the implementation. A pulse's parameters were encoded in the following order and using these contexts:

- $p(w | w_p, c_p)$
- $p(\sigma | w, w_p, c_p)$
- $p(a | w, a_c, \ell)$
- $p(c | w, a, a_c)$

Two problems were encountered with this implementation. The first problem was that it was impractical to initialize the arithmetic coder with prior statistics because of the size of the sample space. This was a minor problem, and could be overcome by using adaptive arithmetic coding starting from a blank slate. Upon switching to adaptive arithmetic coding the second problem was encountered: a very slow learning rate, once again due to the extremely large size of the sample space. There simply weren't enough data (training instances) for the adaptive arithmetic coder to form an accurate probability model.

To see the scope of the problem, consider the following typical case. Suppose that a modest tile size is being used, say,  $t = 8$ , so that each tile resolves into a sequence of length  $8 \times 8 = 64$ . The sizes of the sample spaces of the respective pulse parameters are then as follows:

Tile size	Avg.	Max.	Min.
$4 \times 4$	0.797	1.01	0.0339
$8 \times 8$	0.841	1.00	0.0450
$16 \times 16$	0.913	1.00	0.0714

Table 4.13: Compression results for the verbatim PulseTree scheme using scanning order  $\beta\Omega$ , DPT operator  $L$ , using arithmetic coding.

- $w: 64 \times 64 \times 64 = 64^3 = 262144$
- $\sigma: 64 \times 64 \times 64 \times 64 = 64^4 = 16777216$
- $a: 255 \times 64 \times 255 \times 64 = 255^2 \times 64^2 = 266342400$
- $c: 64 \times 64 \times 255 \times 255 = 255^2 \times 64^2 = 266342400$

This approach clearly suffers from the so-called curse of dimensionality [19]. Apart from the slow learning rate, there is also the more practical matter of storing such a model in computer memory as it is being encoded or decoded. Supposing that every outcome in the respective sample spaces is encoded using a single precision floating-point number, the total amount of memory needed to store all four sample spaces is a staggering 2 gigabytes. Of course, a more efficient storage strategy could be devised, possibly taking into account the sparsity of the sample spaces, but the problem would remain impractical even if such a strategy yielded an order of magnitude improvement. It is also the case that the problem worsens exponentially with increasing tile size.

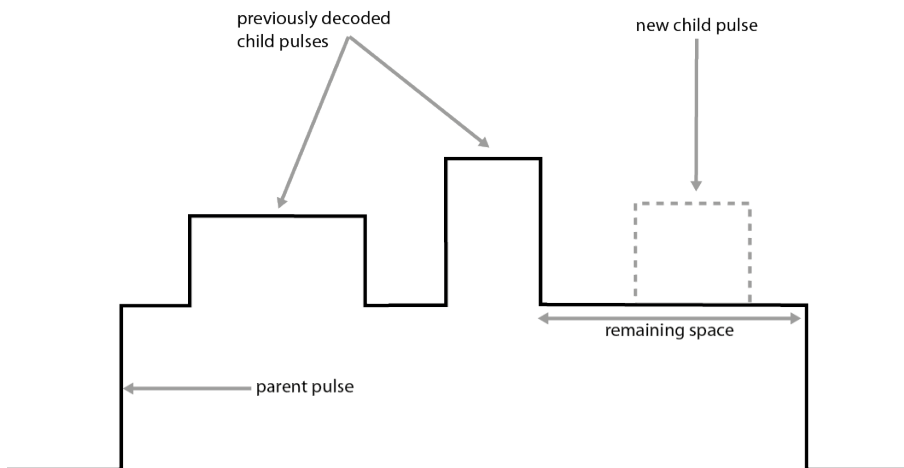


Figure 4.33: Space tracking with the PulseTree scheme.

Another approach that was found to be impractical in hindsight because of dimensionality is the *space tracking* scheme. With space tracking the remaining space left on a parent pulse as its child pulses were being decoded is kept track of, and used as context for the width and offset parameters, as shown in Figure 4.33. Once again this scheme did not change the average compression factor significantly, affecting only the maximum compression factor by reducing it by around 3%.

As with the PulseTrain scheme, several ancillary strategies were tried; in particular an inter-tile scanning order and a coder reset interval were employed, but as before, these strategies had no significant effect on the performance of either the simple or complex PulseTree compression schemes.

## 4.5 Lossy Compression

### 4.5.1 Fidelity Criteria

With any lossy compression procedure there is a loss of information and hence the introduction of some kind of *distortion* when the compressed signal is compared to the original signal; equivalently it loses a degree of *fidelity* [73]. In general the goal in lossy compression is to maximize the amount of compression while minimizing the distortion caused by throwing away some of the information in the original signal; *rate-distortion theory* is concerned with this trade-off between size and quality. With lossy image compression the situation is no different. With lossy image compression it is easy to measure the amount of compression but unfortunately it is not so easy to measure the quality of the resulting image, or equivalently, the amount of distortion it contains.

The type and amount of distortion that is tolerable with a lossy compression procedure depends on the application, and is called the *fidelity criteria* [30]. For example, the fidelity criteria of an image of an artwork that is destined for archiving is different from that of an image of an X-ray used in a medical diagnosis, and both of these are definitely different from an image of a book cover in an online bookshop. When designing a new lossy image compression scheme for a specific area the ideal would be to have the opinion of the end user on hand to guide the design of the scheme; or equivalently some kind of metric if the end user is a machine, performing some image processing task for example. This is not practical, however, for a number of reasons, the biggest being that it is intractable to continually survey a large enough sample of end users in order to get an idea of the fidelity of a certain lossy compression scheme. The solution to this problem is to approximate the ideal subjective evaluation with an objective evaluation. In general it is not easy to design an accurate objective evaluation function, especially for areas where the potential range of application is very broad, like the JPEG compression scheme [80, 15]. For certain other applications, like medical radiography, it is easier to design an accurate objective evaluation function [53].

The image compression algorithms we are designing in this text are not targeted to any one area of application, and this puts us in the same situation as that of image compression algorithms such as JPEG [80, 15]. For these algorithms that have a strong emphasis on generality two image fidelity models stand out: the one is simple and inaccurate, and the other is more complex, but also more accurate.

#### 4.5.1.1 Peak Signal-to-Noise Ratio (PSNR)

A simple and popular assessment of image fidelity is the *peak signal-to-noise ratio* (PSNR) [70]. It is essentially based on the *mean square error* (MSE), which is defined as follows

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N (x_n - y_n)^2 \quad (4.6)$$

where  $x$  is the original signal and  $y$  is the modified signal.

To incorporate the strength of the original signal relative to the error incurred between the original and the modified image, the *signal-to-noise ratio* (SNR) can be used:

$$SNR = 10 \log_{10} \frac{\overline{x^2}}{\sigma^2} \quad (4.7)$$

where  $\overline{x^2}$  is the average squared value of the original signal:  $\overline{x^2} = \frac{1}{N} \sum_{n=1}^N x_n^2$ , and where the ratio is measured on a logarithmic scale, the units of which are *decibels*.

The peak signal-to-noise ratio emerges when instead of measuring the error relative to the average squared value of the original signal, it is measured relative to the square of the peak value of the original signal, as follows:

$$PSNR = 10 \log_{10} \frac{x_{peak}^2}{\sigma^2} \quad (4.8)$$

where the ratio is again on a logarithmic scale.

While PSNR is relatively easy to compute and optimize for, it is unfortunately not very accurate, as can be seen in Figure 4.34. The next section will discuss the nature of this inaccuracy.

#### 4.5.1.2 Structural Similarity Index (SSIM)

The Structural Similarity Index (SSIM) [82] was developed to address the deficiencies of the *error sensitivity* approach, which assumes that perceptual quality is best estimated by the quantification of the visibility of errors. The biggest deficiency of the error sensitivity approach is that it does not take the functioning of the human visual system into account. PSNR, discussed above, is an example of such an error sensitivity approach.

SSIM, on the other hand, is based on the assumption that the human visual system is very sensitive to structural information, as opposed to factors like absolute pixel intensities. The SSIM similarity index has the following properties (where  $x$  and  $y$  are two image signals):

- Symmetry:  $S(x, y) = S(y, x)$
- Boundedness:  $S(x, y) \leq 1$
- Unique maximum:  $S(x, y) = 1 \iff x = y$

With SSIM, the more similar two images are to each other the closer the SSIM index will be to unity.

The SSIM similarity index was found to outperform other quality assessment models, such as PSNR, UQI [81] and the Sarnoff model [72], in an experimental setting using actual subjective evaluation. The power of SSIM over that of PSNR is illustrated in Figure 4.34 where an image has been contrast stretched and blurred, respectively. Both of these modifications have the same PSNR, but the SSIM index of the contrast stretched image is higher than that of the blurred image, reflecting the fact that the contrast stretched image has less perceptual distortion than the blurred image.



Figure 4.34: Comparing the PSNR and SSIM image quality models. (b) and (c) have the same PSNR, but the SSIM indices correctly indicate that (b) has higher fidelity than (c).

## 4.5.2 LossyPulseTree scheme

### 4.5.2.1 Modelling

The LossyPulseTree scheme is based on the PulseTree scheme discussed in Section 4.4.2. The most important modification is that with the LossyPulseTree scheme the pulse amplitudes are quantized before they are coded.

Quantization was discussed in Section 3.6. From that discussion it is clear that there are two main criteria when it comes to choosing a quantizer: the amount of *distortion* the quantizer introduces and the *rate* of the quantizer, i.e. how expensive it is to code the output of the quantizer. Results from rate-distortion theory tell us that it is impossible to have both [7]. From Section 3.6 we know that an optimal uniform quantizer has the best rate, whereas an optimal nonuniform quantizer has the least amount of distortion. We introduce a third criterion to choose between the two: practicality. It is much easier to design an optimal uniform quantizer than a nonuniform one, because of the fact that solving Equation 3.7 for  $\Delta$  is easier than using the Lloyd-Max algorithm to find the optimal boundaries and reconstruction levels for the same probability distribution. Therefore we will use uniform quantization in this text.

**Scanning Orders and Quantization** Since scanning orders are being used to map two-dimensional images to one-dimensional sequences, and since lossy compression involves the elimination of information, the scanning order of our lossy compression scheme will determine to a large extent the nature of the compression artifacts associated with that scheme. Intuitively, it seems obvious that an isotropic scanning order is preferable to an anisotropic scanning order, since with the latter the compression artifacts will be biased in a certain direction, and will therefore be more noticeable<sup>7</sup>.

<sup>7</sup>Assuming that the image itself has no directional bias, an assumption that has to be made if the compression scheme is to be general.



The difference between the quantization effects of isotropic and anisotropic scanning orders is illustrated in Figure 4.35. The distortion caused by leaving out the first 10 resolution levels in a partial DPT reconstruction is more noticeable with the anisotropic Snake scanning order than with the isotropic Hilbert scanning order — the Snake scanning order has a clear horizontal bias.

Since the Hilbert scanning order is the most isotropic scanning order available and it results in some of the best compression factors with the lossless PulseTree compression scheme, it will be our scanning order of choice for the LossyPulseTree scheme.

**Quantifying Lossy Image Compression Performance** With lossless image compression it is simple to measure the performance of a compression scheme: the smaller the size of the output, the better the compression scheme. With lossy coding the situation is different since we are also concerned about the *fidelity* of the result, in addition to the size of the output. Since we will be using the SSIM index introduced in Section 4.5.1.2 to quantify image fidelity, it seems expedient to define the following overall performance metric for our lossy image compression schemes:

$$\lambda := s - c \quad (4.9)$$

where  $s$  is the SSIM index and  $c$  is the compression factor,  $c := \frac{\text{compressed size}}{\text{original size}}$ . The minimum value for  $\lambda$  is  $-1$ , corresponding to the situation where  $s = 0$  and  $c = 1$ , which is the worst kind of performance for a lossy image compressor. The maximum value of  $\lambda$  is  $1$ , corresponding to the ideal but impossible situation where  $s = 1$  and  $c = 0$ . A  $\lambda$  value of  $0$  means that  $s = c$ , and consequently we will treat any  $\lambda < 0$  as unacceptable.

**Model Details** The simplest way to quantize would be to choose a fixed number of quantizer reconstruction levels and apply it to the pulses of *all* resolution levels. This strategy leads to unacceptable distortion, as can be seen in Figure 4.36. The distortion is because of the fact that quantization errors become more visible as pulse widths grow larger. Therefore we introduce a *quantization vector*  $\mathbf{q}$  of dimension equal to the number of resolution levels in a DPT. Each entry in this quantization vector corresponds to the number of reconstruction levels  $M$  of the quantizer for that resolution level.

The optimal values of the quantization vector were determined manually by first setting all the entries of the quantization vector to full quality, which is  $M = 256$ . Then, for each level, starting at resolution level  $r = 1$ , the number of reconstruction levels were set to  $M = 0$  and increased until no obvious compression artifacts were left in the resulting image. This was of course a subjective procedure, and the resulting image fidelity will reflect that. Also note that the image fidelity aimed for in this procedure was fairly strict, since no obvious compression artifacts were allowed in the resulting compressed image. It must be kept in mind that the level of tolerable image fidelity will depend on the application.

The resulting quantization vector is as follows:

$$\mathbf{q} = [2, 8, 32, 32, 32, 256, 256, 256, 256, 256, 256, \dots] \quad (4.10)$$



(a) Original image.



(b) Partial DPT reconstruction using the Snake scanning order (a highly anisotropic scanning order).



(c) Partial DPT reconstruction using the Hilbert scanning order (a highly isotropic scanning order).

Figure 4.35: Effects of quantization with isotropic and anisotropic scanning orders. With both images the first 10 resolution levels were left out in a partial DPT reconstruction.

Notice that for resolution levels 6 and up the number of reconstruction levels is equal to 256, which is full quality. This is because it was found empirically that quantizing pulses of width greater than 5 leads to noticeable compression artifacts, even on a relatively high quality setting of  $M = 128$  reconstruction levels. Therefore the concept of a *quantization threshold level* can be introduced, which in this case is equal to 5.

**Weber culling** In the field of visual psychophysics *Weber's Law* [75] aims to quantify the phenomenon of *just noticeable difference*, which can be explained as follows. Suppose that there is a large screen with a uniform illumination of intensity  $I$ , with a spot in the middle of the screen with an intensity  $I + \Delta I$ , with  $\Delta I > 0$ . Given  $I$ , what is the smallest value of  $\Delta I$  so that the human eye can see the spot? Weber's law quantifies the empirical observation that usually  $\frac{\Delta I}{I} = c$ , so that the sensitivity of the eye to illumination is described by a logarithmic function. Small differences in illumination are more perceptible against a dark background, whereas for bright backgrounds very large differences in illumination are necessary in order for the eye to be able to notice a difference. In practice, for a very large range of situations, the constant  $c$  in Weber's Law has been found to be about 0.02 [70].

It would be advantageous for us if Weber's Law could somehow be used to cull those resolution pulses in the DPT that are imperceptible. Unfortunately, the conditions are not equivalent: Weber's Law describes the perception two-dimensional objects (i.e. a spot on a screen), whereas resolution pulses are one-dimensional objects that are wrapped up in some way to approximate a two-dimensional object using a scanning order. But a "naive" approach can be tried in any case to see if it yields any useful results.

The naive approach works as follows. A pulse is discarded in accord with Weber's Law if its amplitude  $a$  satisfies the following condition:

$$a < c a_{cp} \quad (4.11)$$

where  $a_{cp}$  is the *cumulative* amplitude of the pulse's parent. This one-dimensional situation is completely analogous to the two-dimensional one for which Weber's Law holds.

Remarkably, the naive approach works, and even more remarkably, the optimal value for the constant  $c$  was found to be 0.02 which is exactly the same as for the two-dimensional case. Any value of  $c$  larger than about 0.02 causes noticeable distortion to appear. Consequently Weber culling can now be used to throw away imperceptible pulses before they are quantized. The results of this procedure are shown in Section 4.5.2.2.

**Compression artifacts** There are basically two kinds of distortion — or compression artifacts — associated with the LossyPulseTree scheme. The first kind of compression artifact is called *blocking* and is associated with excessive quantization in high resolution levels. Figure 4.36 illustrates the effect of the artifact: smooth sections of the image become blocky. The pulses of high resolution levels occupy a larger area in the image space, so that the effects of quantization are much more easily visible with them in contrast to pulses of small resolution levels. This compression artifact can be solved by either decreasing the amount of quantization in higher levels or by just choosing a

smaller tile size, thereby eliminating those high resolution levels altogether. It is of course desirable to use a tile size that is as small as possible in any case since smaller tile sizes correspond to better compression, as was seen in Section 4.4.2. Empirically it has been found that for the quantization vector  $\mathbf{q}$  described in Equation 4.10, the smallest tile size with which this blocking artifact is no longer a problem is 16.

The second compression artifact is called *edge deterioration* and is illustrated in Figure 4.37. This artifact is a result of the large amount of quantization that is applied to the lower resolution levels, specifically the first resolution level. The artifact appears when details that are relatively dark appear on a very bright background, as is the case with 4.37. Since the  $L$  operator is being used to perform the DPT transform, the bright background is transformed into actual *pulses* of varying widths, which would not be the case if the background was black and the details were white. Some of these bright “background pulses” are then of a very small width because of the intervening black details and the scanning order that was used. These small background pulses are then quantized and the result is usually that they lose a large amount of their brightness because of the nature of the quantizer. Hence the distortion, where it seems that the dark details seem to “bleed” into the white background.

#### 4.5.2.2 Coding

Arithmetic coding was used to code the results of the model and the details of the coding procedure is the same as that of the PulseTree scheme discussed in Section 4.4.2.

The coding was performed with and without Weber culling. The results are reported in Tables 4.14 and 4.15. Most of the resulting images — with and without Weber culling — contained no perceptible compression artifacts; this is reflected by the high average SSIM index.

Regarding Weber culling, it is clear that the technique is not as efficient as was hoped, as it reduces the average compression factor by a mere 0.01, leaving the SSIM index virtually unchanged. The most likely reason for this is that the amplitudes of those pulses that are discarded by Weber culling are very small amplitudes in the majority of cases, so that they would have been quantized to an amplitude of zero by the quantizer in any case, producing the same net effect.

	<b>Avg.</b>	<b>Max.</b>	<b>Min.</b>
Compression factor	0.45	0.93	0.0017
SSIM index	0.98	1.00	0.92
$\lambda$ factor	0.53	0.99	0.07

Table 4.14: Results of LossyPulseTree compression scheme with quantization and no Weber culling.

## 4.6 Conclusion

In this chapter the principles and methods of the previous two chapters have been combined to yield three new image compression schemes: two of them

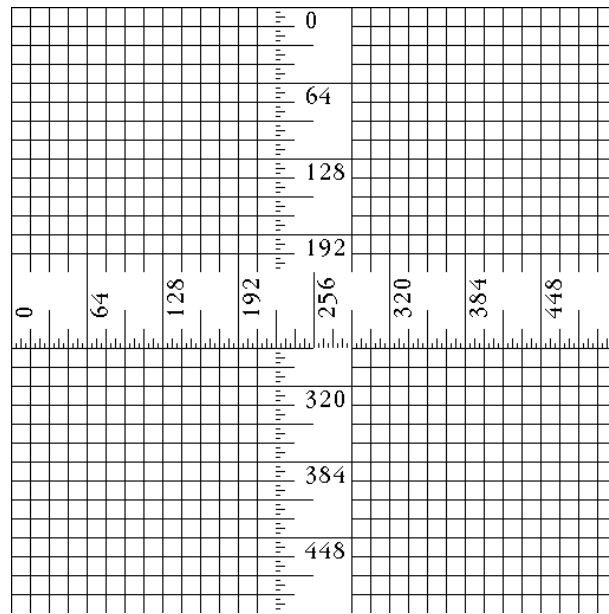


(a) Original image.

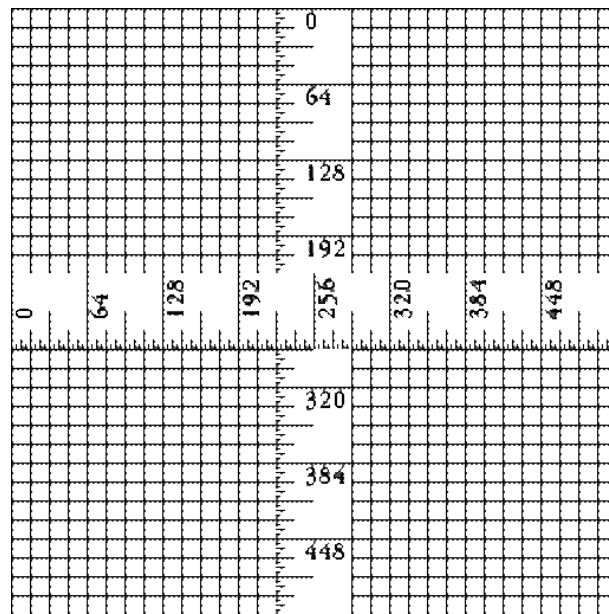


(b) Highly compressed image with blocking artifacts.

Figure 4.36: Blocking artifacts in the LossyPulseTree scheme when the tile size is too small.



(a) Original image.



(b) Edge deterioration.

Figure 4.37: Edge deterioration artifacts in the LossyPulseTree scheme.

	<b>Avg.</b>	<b>Max.</b>	<b>Min.</b>
Compression factor	0.44	0.92	0.0017
SSIM index	0.98	1.00	0.92
$\lambda$ factor	0.53	0.99	0.08

Table 4.15: Results of LossyPulseTree compression scheme with quantization and Weber culling.

lossless, and one lossy, based on the best lossless compression scheme.

All of the image compression schemes make use of space-filling scanning orders to map the two-dimensional image data to one-dimensional sequences ready for decomposition by a DPT. The scanning orders that produced the lowest total variation in the resulting one-dimensional sequences were chosen for use in the image compressors, since lower total variation is associated with higher compressibility. It was also important to use isotropic scanning orders so as not to introduce a directional bias into the image compressors.

On the lossless side, the PulseTree lossless compression scheme performs the best of the two compression schemes, achieving an average compression factor of 0.8, with a worst-case compression factor of 1.0 for images without a lot of structure (see Figure 4.31 for examples).

On the lossy side, the LossyPulseTree lossy compression scheme is based on the lossless PulseTree scheme and achieves an average compression factor of 0.45 without any perceptible quality degradation as corroborated by an average SSIM index of 0.98. The worst-case compression factor for this scheme is 0.93. It was seen that the choice of scanning order plays determines to a large extent the kind of distortion that emerges because of lossy compression. Once again an isotropic scanning order was called for to minimize the perceptual severity of the resulting distortions.

## Chapter 5 - Conclusion

The goal of this thesis was to investigate whether the one-dimensional Discrete Pulse Transform could be used to build an image compressor, a type of technology that is becoming ever more important in this digital age. The aim was not to build a complete image compressor that would be ready for everyday use, but rather to demonstrate that it would be possible to build such an image compressor by showing that essentially an image transformed by a one-dimensional Discrete Pulse Transform could be compressed efficiently.

In Chapter 2 *LULU* theory, which forms the basis of the Discrete Pulse Transform, was introduced. We listed some of its most important properties, with shape preservation being the most attractive for the application at hand.

In Chapter 3 we discussed data compression and noted some of the most efficient general data compression techniques, for both lossless and lossy compression.

In Chapter 4 the synthesis of the material of the previous chapters was developed, leading to four new image compression schemes based on the Discrete Pulse Transform.

Subsequently we can list the most important contributions of this thesis as follows:

- The results regarding the performance of all the scanning orders and image compression schemes in this text, based on experiments run on the Image Test Set, which is a collection containing 1000 diverse images of resolution  $512 \times 512$ , described in Section 4.3.1.1.
- Demonstration of the viability of using space-filling scanning orders to map two-dimensional image data to one dimension suitable for transformation by a one-dimensional Discrete Pulse Transform. A variety of such scanning orders were tested for their structure-preserving properties, as measured by the total variation of the one-dimensional sequence they produced. We required that the scanning orders be isotropic so that they would perform well on any kind of image, not just those images with a specific directional bias.

It was found that the Hilbert, H and  $\beta\Omega$  scanning orders performed the best overall, as discussed in Section 4.3.3.2.

- The development of four lossless image compression schemes based on the Discrete Pulse Transform. The best of these schemes, the PulseTree scheme of Section 4.4.2, achieves an average compression factor of 0.80 with a maximum compression factor of 1.0.



For comparison, the Portable Network Graphics (PNG) compression scheme, which is a widely used and efficient lossless image compression scheme, achieves an average compression factor of 0.68 with a worst-case compression factor of 0.99.

- The development of a lossy image compression scheme, the LossyPulseTree scheme of Section 4.5.2, based on the Discrete Pulse Transform. This scheme achieves an average compression factor of 0.45 and a worst-case compression factor of 0.93 with compressed images that have an average SSIM index<sup>1</sup> of 0.98 and a minimum SSIM index of 0.92 when compared to the original images. The scheme used a source-optimized uniform quantizer.

For comparison, the state-of-the-art JPEG2000 lossy image compression scheme achieves an average compression factor of 0.20 with a worst-case compression factor of 0.91. The average SSIM index is 0.97 and the worst-case SSIM index is 0.93.

- Demonstration that a psychovisual approach to lossy image compression based on Weber’s Law could be viable (the so-called “Weber culling” approach of Section 4.5.2.1). The effect of the approach was found to be negligible, however, when used in conjunction with a uniform quantization scheme. The approach was a one-dimensional approximation to the standard two-dimensional case as found in the literature, but remarkably the so-called empirical *Weber factor* in the one-dimensional case was almost exactly the same as the factor found in the standard two-dimensional case.

We consider this text to be an exploratory effort and we can list some possible avenues for future research here:

- Investigate more intelligent quantization strategies with the resolution levels of a DPT. The quantization method used in Section 4.5.2 with the LossyPulseTree scheme leads to noticeable quantization artifacts even on low quantization settings, because of the fact that space-filling scanning orders are being used to map the two-dimensional image data to one-dimensional sequences and back. The effects of quantization is thus more difficult to predict than the one-dimensional case. This difficulty in predicting the two-dimensional effect of one-dimensional quantization is, in a sense, partially cancelling out the shape preservation properties of the DPT, one of the key advantages of using a DPT.
- Use the two-dimensional Discrete Pulse Transform to do image compression. The  $n$ -dimensional generalization of a DPT was discussed in Section 2.4.4, where it was mentioned that most of the important properties of the one-dimensional DPT generalize to the  $n$ -dimensional case. Using a two-dimensional DPT will remove the need for scanning orders and provide a mapping of the problem that is much more natural, with the important benefit that image structure (i.e. information) will not be lost during the mapping process from two dimensions to one dimension.

---

<sup>1</sup>The SSIM index was discussed in Section 4.5.1.2.

It will also be much easier to design an efficient quantizer with a two-dimensional DPT, since the cancellation effect mentioned previously will not be present. Working with two-dimensional pulses will also provide a chance to design true psychovisual quantization schemes based on the direct application of Weber's law, improving on the one-dimensional approximation explored here.

- Use the DPT to do video compression. Video compression is fundamentally similar to image compression, since a video can be considered to be merely a sequence of images. A so-called *intra-frame* video compression scheme, where every frame in the video is modelled as being independent of all other frames, reduces to designing an efficient image compressor. For an *inter-frame* video compression scheme there are two options. The first is to apply the one-dimensional DPT to the sequence formed by tracking a single pixel over time, exploiting the fact that typically most video pixels remain stationary, resulting in easily compressible transforms with low pulse counts. This approach is also compatible with motion compensation, a standard video compression technique. The second option is to model the dependencies between image frames as three-dimensional pulses in a three-dimensional DPT.
- Explore the applicability of the *LULU* operators to the physiological problem of human vision. In Section 4.3.3.5 the topic of the anisotropy of images transformed with the DPT was briefly touched upon. This brief investigation seemed to suggest that the DPT might be an indicator of a natural "preferred orientation" for scenes. The photoreceptors in the retina are packed hexagonally, and at this level the eye could be performing a DPT-like decomposition along each of the three hexagonal axes to perform some early processing and/or encode the visual information so that it can be sent to the visual cortex for further processing.

Further research on this question will involve both mathematics and cognitive science.

## Bibliography

- [1] R. Anguelov. Discrete pulse transform of images: Algorithm and applications. In *Proceedings of the ICPR 2008: 19th International Conference on Pattern Recognition*, 2008.
- [2] R. Anguelov and I.N. Fabris-Rotelli. Discrete pulse transform of images. In *Proceedings of the ICISP 2008: International Conference on Image and Signal Processing*, 2008.
- [3] H. Anton and C. Rorres. *Elementary Linear Algebra with Applications*. Wiley, 9th edition, 2005.
- [4] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theor. Comput. Sci.*, 181(1):3–15, 1997.
- [5] R.B. Ash. *Basic Abstract Algebra*. Dover Publications, 2006.
- [6] V.K. Balakrishnan. *Introductory Discrete Mathematics*. Dover Publications, 2010.
- [7] T. Berger. *Rate distortion theory: A mathematical basis for data compression*. Prentice-Hall, 1971.
- [8] A. Bijaoui, F. Murtagh, and J.L. Starck. *Image Processing and Data Analysis — The Multiscale Approach*. Cambridge University Press, 1998.
- [9] G.L. Bretthorst. *Bayesian Spectrum Analysis and Parameter Estimation*. Springer-Verlag, 1988.
- [10] W. Burger and M.J. Burge. *Digital Image Processing*. Springer, 2007.
- [11] G. Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers*. Dover Publications, 1915.
- [12] B.P. Carlin and T.A. Louis. *Bayesian Methods for Data Analysis*. CRC Press, 3rd edition, 2009.
- [13] CERN. Worldwide LHC Computing Grid. <http://lcg.web.cern.ch/LCG/public/>, 2010.
- [14] C. Christopoulos, J. Askelöf, and M. Larsson. Efficient Region of Interest Coding Techniques in the Upcoming JPEG2000 Still Image Coding Standard. In *Proceedings of the 2000 International Conference on Image Processing (ICIP 2000)*, 2000.

- [15] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, 2000.
- [16] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.
- [17] C.D. Creusere. Robust image coding using the embedded zerotree wavelet algorithm. In *Proceedings of the 1996 Data Compression Conference*, 1996.
- [18] R.A. DeVore, B. Jawerth, and B.J. Lucier. Image compression through wavelet transform coding. *IEEE Trans. on Information Theory*, 38(2):719–746, 1992.
- [19] D. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. In *Math Challenges of the 21st Century (The American Math. Society)*, 2000.
- [20] J.P. du Toit. The discrete pulse transform and applications. Master’s thesis, University of Stellenbosch, 2007.
- [21] I.N. Fabris-Rotelli. LULU operators on multidimensional arrays and applications. Master’s thesis, University of Pretoria, 2009.
- [22] R. Fattal. Edge-avoiding wavelets and their applications. *ACM Trans. Graph.*, 28(3):1–10, 2009.
- [23] P.J.S.G Ferreira and A.J. Pinho. Histogram packing, total variation, and lossless image compression. In *Proceedings of EUSIPCO-2002, XI European Signal Processing Conference*, 2002.
- [24] J. Fox and J.S. Long. *Modern methods of data analysis*. Sage Publications, 1990.
- [25] M.W. Frazier. *An Introduction to Wavelets Through Linear Algebra*. Springer, 1999.
- [26] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.
- [27] R.G. Gallager and D.C. van Voorhis. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21(2):228–230, 1975.
- [28] J.F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva. The diverse and exploding digital universe. White paper. IDC, sponsored by EMC, 2008.
- [29] S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12:399–401, 1966.
- [30] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, 3rd edition, 2007.

- [31] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Trans. Image Processing*, 5(5):794–797, 1996.
- [32] C. Harrison, A.K. Dey, and S.E. Hudson. Evaluation of progressive image loading schemes. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2010.
- [33] H. Haverkort. Recursive tilings and space-filling curves with little fragmentation. In *EuroCG '10: Proceedings of the 26th European Workshop on Computational Geometry*, pages 185–188, 2010.
- [34] H. Haverkort and F. van Walderveen. Locality and bounding-box quality of two-dimensional space-filling curves. *Comput. Geom. Theory Appl.*, 43(2):131–147, 2010.
- [35] T. Higgs. Energy efficient computing. In *Proceedings of the 2007 IEEE International Symposium on Electronics and the Environment*, pages 210–215, 2007.
- [36] D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Math. Annln.*, 38:459–460, 1891.
- [37] S.G. Hoggar. *Mathematics of Digital Images*. Cambridge University Press, 2006.
- [38] J.M. Howie. *Fundamentals of Semigroup Theory*. Oxford University Press, 2006.
- [39] M.D. Jankowitz. *Some Statistical Aspects of LULU Smoothers*. PhD thesis, Stellenbosch University, 2007.
- [40] A. Kiely. Selecting the Golomb Parameter in Rice Coding. In *IPN Progress Report*, 2004.
- [41] D. Kundur and D. Hatzinakos. Digital Watermarking for Telltale Tamper Proofing and Authentication. *Proceedings of the IEEE*, 87(7), 1999.
- [42] D.P. Laurie. The Roadmaker’s Algorithm for the Discrete Pulse Transform. *IEEE Transactions on Image Processing*, 20:361–371, 2011.
- [43] D.P. Laurie and C.H. Rohwer. Fast implementation of the discrete pulse transform. In *Proceedings of ICAAM 2006: International Conference of Numerical Analysis and Applied Mathematics*, 2006.
- [44] H.L. Lebesgue. Leçons sur l’intégration et le recherche des fonctions primitives. *Gauthier-Villars*, pages 44–45, 1904.
- [45] S.P. Lloyd. Least Squares Optimization in PCM. *IEEE Transactions on Information Theory*, 28:127–135, 1982.
- [46] J. Lukaszewicz and H. Steinhaus. On measuring by comparison. *Zastos. Mat.*, pages 225–231, 1955.
- [47] D.J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2006.

- [48] C.L. Mallows. Some theory of nonlinear smoothers. *Ann. Stat.*, 8(4):695–715, 1980.
- [49] J. Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6:7–12, 1960.
- [50] G. Mitchison and R. Durbin. Optimal numberings of an  $n \times n$  array. *SIAM Journal on Discrete and Algebraic Methods*, 7(4):571–582, 1986.
- [51] NASA. Mars Reconnaissance Orbiter. <http://www.nasa.gov/mro>, 2010.
- [52] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. In *FCT '97: Proceedings of the 11th International Symposium on Fundamentals of Computation Theory*, pages 364–375, London, UK, 1997. Springer-Verlag.
- [53] The Effect of Image Data Compression on the Clinical Information Quality of Compressed Computed Tomography Images for Teleradiology Applications. *European Journal of Scientific Research*, 23(1), 2008.
- [54] A. Perez, S. Kamata, and E. Kawaguchi. Peano scanning of arbitrary size images. In *Proceedings of the International Conference on Pattern Recognition*, pages 565–568, 1992.
- [55] L.E. Renner. *Linear Algebraic Monoids*. Springer Berlin Heidelberg, 2009.
- [56] R.F. Rice. Some practical universal noiseless coding techniques. *JPL Publication*, 79(22), 1979.
- [57] C.H. Rohwer. Idempotent one-sided approximation of median smoothers. *Journal of Approximation Theory*, 58(2):151–163, 1989.
- [58] C.H. Rohwer. Projections and separators. *Quaestiones Mathematicae*, 22:219–230, 1999.
- [59] C.H. Rohwer. Fast approximation with locally monotone sequences. In *Proceedings of the 4th FAAT Conference, Maratea*, 2002.
- [60] C.H. Rohwer. *Multiresolution Analysis with Pulses*, volume 142 of *International Series of Numerical Mathematics*. Birkhäuser Verlag Basel, 2002.
- [61] C.H. Rohwer. Variation reduction and *lulu*-smoothing. *Quaestiones Mathematicae*, 25:163–176, 2002.
- [62] C.H. Rohwer. Fully trend preserving operators. *Quaestiones Mathematicae*, 27:217–229, 2004.
- [63] C.H. Rohwer. *Nonlinear Smoothing and Multiresolution Analysis*, volume 150 of *International Series of Numerical Mathematics*. Birkhäuser, 2005.
- [64] C.H. Rohwer. Personal communication, 2010.
- [65] C.H. Rohwer and L.M. Toerien. Locally monotone robust approximation of sequences. *Journal of Computational and Applied Mathematics*, 36:399–408, 1991.

- [66] C.H. Rohwer and M. Wild. Natural alternative for one dimensional median filtering. *Quaestiones Mathematicae*, 25:135–162, 2002.
- [67] C.H. Rohwer and M. Wild. LULU theory, Idempotent Stack Filters, and the Mathematics of Vision of Marr. *Advances in Imaging and Electron Physics*, 146:57–162, 2007.
- [68] H.L. Royden. *Real Analysis*. Macmillan, 1969.
- [69] H. Sagan. *Space-Filling Curves*. Springer, 1st edition, September 1994.
- [70] K. Sayood. *Introduction to Data Compression (3rd Edition)*. Morgan Kaufmann, 2006.
- [71] C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.
- [72] H.R. Sheikh, Z. Wang, A.C. Bovik, and L.K. Cormack. Image and video quality assessment research at LIVE. <http://live.ece.utexas.edu/research/quality/>.
- [73] D.A. Silverstein and J.E. Farrell. The relationship between image fidelity and image quality. In *Proceedings of the IEEE International Conference on Image Processing*, 1996.
- [74] J.S. Simonoff. *Smoothing methods in statistics*. Springer, 1996.
- [75] J.B.J Smeets and E. Brenner. Grasping weber’s law. *Current Biology*, 18(23), 2008.
- [76] H. J. Trussell. *Digital Image Processing*. Cambridge University Press, 2008.
- [77] J.W. Tukey. *Exploratory data analysis*. Addison-Wesley, 1977.
- [78] P.F. Velleman. Definition and comparison of robust nonlinear data smoothing algorithms. *Journal of the American Statistical Association*, 75(371):609–615, 1980.
- [79] U. von Luxburg. Lokaliitätsmaße von peanokurven. Technical report, Wilhelm-Shickard-Institut für Informatik, Universität Tübingen, 1998.
- [80] G.K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1), 1992.
- [81] Z. Wang and A.C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9:81–84, 2002.
- [82] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004.
- [83] J. Wierum. Definition of a new circular space-filling curve  $\beta\omega$ -indexing. Technical Report TR-001-02, Paderborn Center for Parallel Computing, 2002.