Ridgelets: A promising new wavelet-like transform to represent objects with linear singularities

Maria Beatriz Teruel

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation

Teruel, Maria Beatriz, "Ridgelets: A promising new wavelet-like transform to represent objects with linear singularities" (2003). UNLV Retrospective Theses & Dissertations. 1513.

https://digitalscholarship.unlv.edu/rtds/1513

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
RIDGELETS: A PROMISING NEW WAVELET-LIKE TRANSFORM TO
REPRESENT OBJECTS WITH LINEAR SINGULARITIES

by

María Beatriz Teruel
Licenciada en Matematica
Universidad Nacional del Sur,
Bahía Blanca, Argentina
1999

A thesis submitted in partial fulfillment
of the requirement for the

Master of Science Degree
Mathematical Sciences Department
College of Sciences

Graduate College
University of Nevada, Las Vegas
May 2003

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The Thesis prepared by

Maria Beatriz Teruel

Entitled

Ridgelets: A Promising New Wavelet-Like Transform to Represent Objects with Linear Singularities

is approved in partial fulfillment of the requirements for the degree of

Master of Science

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

Ridgelets: A Promising New Wavelet-Like Transform to Represent Objects with Linear Singularities

by

Maria Beatriz Teruel

Dr. David G. Costa, Examination Committee Chair
Professor of Mathematics
University of Nevada, Las Vegas

In the last two decades plenty of research has been carried out in the field of Wavelet theory and it is well known that wavelets can efficiently deal with point-like singularities. Unfortunately, such is not the case for higher dimensions singularities. To overcome this weakness of the Wavelet transform E. Candès and D. Donoho [4] introduced a new wavelet-like transform that can effectively deal with linear singularities in two dimensions, namely the Ridgelet transform. This new representation tool exploits the ability of wavelets to deal with point singularities. In fact, the Ridgelet transform is equivalent to a one-dimensional wavelet transform in the Radon domain. By doing so, a line singularity is transformed into a point singularity (by means of the Radon transform) which can then be efficiently analyzed by the wavelet transform.
This thesis presents the Ridgelet transform, its properties and connections to the Radon and Wavelet transform. Also, the reader is presented with practical results that allow us to see how the Ridgelet transform is much better suited than the Wavelet transform for representing images with straight edges (linear singularities).
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER 1  INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2  THE CONTINUOUS RADON TRANSFORM</td>
<td>4</td>
</tr>
<tr>
<td>2.1 The Slant Stack Transform</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER 3  THE DISCRETE RADON TRANSFORM</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Inversion of the Discrete Slant Stack Transform</td>
<td>21</td>
</tr>
<tr>
<td>CHAPTER 4  FRAMES</td>
<td>24</td>
</tr>
<tr>
<td>4.1 Bounded Unconditional Bases (Riesz Bases) and Frames</td>
<td>31</td>
</tr>
<tr>
<td>CHAPTER 5  THE WAVELET TRANSFORM</td>
<td>40</td>
</tr>
<tr>
<td>5.1 The Continuous Wavelet Transform</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Meyer Wavelets</td>
<td>42</td>
</tr>
<tr>
<td>5.3 The Discrete Wavelet Transform</td>
<td>45</td>
</tr>
<tr>
<td>CHAPTER 6  THE CONTINUOUS RIDGELET TRANSFORM</td>
<td>48</td>
</tr>
<tr>
<td>CHAPTER 7  THE DISCRETE RIDGELET TRANSFORM</td>
<td>60</td>
</tr>
<tr>
<td>CHAPTER 8  IMPLEMENTATION REMARKS AND ALGORITHM</td>
<td>63</td>
</tr>
<tr>
<td>8.1 Source Code for the Algorithm</td>
<td>66</td>
</tr>
<tr>
<td>CHAPTER 9  CONCLUSIONS</td>
<td>97</td>
</tr>
<tr>
<td>9.1 Practical Results</td>
<td>97</td>
</tr>
<tr>
<td>9.2 Final Remarks</td>
<td>108</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>110</td>
</tr>
<tr>
<td>Appendix A</td>
<td>110</td>
</tr>
<tr>
<td>Appendix B</td>
<td>113</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF SYMBOLS

\( S \)  
Schwarz space of rapidly decreasing \( C^\infty \) functions.

\( Rf \)  
Radon Transform of \( f \).

\( S^{*-1} \)  
Unit sphere in \( \mathbb{R}^n \).

\( \hat{f} \)  
Fourier Transform of \( f \).

\( F_n \)  
n-dimensional Fourier Transform.

\( H \)  
Hilbert Transform.

\( \langle \cdot, \cdot \rangle \)  
Inner product.

\( Sf \)  
Slant Stack Transform of \( f \).

\( C^\infty_0 \)  
Space of \( C^\infty \) functions with compact support.

\( P_{T_i}, i=1,2 \)  
Pseudopolar Fourier Transform.

\( \mathcal{H} \)  
Hilbert space.

\( R_u f \)  
Radon Transform of \( f \) in the direction of \( u \).

\( R \)  
Ridgelet Transform.

\( F^\alpha \)  
Fractional Fourier Transform of factor \( \alpha \).
LIST OF FIGURES

Figure 2.1: Slant Stack Transform of a point................................. 13
Figure 2.2: Slant Stack Transform of a line segment.................. 14
Figure 3.1: Set of basically vertical lines................................. 18
Figure 3.2: Set of basically vertical lines................................. 19
Figure 3.3: Pseudopolar grid.................................................. 20
Figure 5.2: Modulus of the Meyer wavelet associated to the auxiliary function $\nu(x) = x^4(35-84x + 70x^2 - 20x^3)$ on $[0, 1]$ ................. 43
Figure 5.2: Time domain representation of the mother Meyer wavelet $\psi$ associated to $\nu$.......................................................... 44
Figure 6.1: A typical ridgelet function....................................... 49
Figure 7.1: Map of ridgelet coefficients..................................... 62
Figure 9.1.1: Half Dome......................................................... 98
Figure 9.1.2: Reconstruction with 8 Ridgelet coefficients .......... 98
Figure 9.1.3: Reconstruction with 8 Meyer wavelet coefficients .... 98
Figure 9.1.4: Reconstruction with 8 Coif – 6 wavelet coefficients 99
Figure 9.1.5: Reconstruction with 8 Dau – 4 wavelet coefficients ... 99
Figure 9.1.6: Reconstruction with 64 Ridgelet coefficients ........ 100
Figure 9.1.7: Reconstruction with 64 Meyer wavelet coefficients .. 100
Figure 9.1.8: Reconstruction with 64 Coif – 6 wavelet coefficients .. 100
Figure 9.1.9: Reconstruction with 64 Dau – 4 wavelet coefficients .. 100
Figure 9.1.10: Original image (prism)....................................... 101
Figure 9.1.11: Reconstruction with 64 Ridgelet coefficients ........ 101
Figure 9.1.12: Reconstruction with 64 Meyer wavelet coefficients 101
Figure 9.1.13: Reconstruction with 64 Coif – 6 wavelet coefficients 102
Figure 9.1.14: Reconstruction with 64 Dau – 4 wavelet coefficients 102
Figure 9.1.15: Reconstruction with 256 Ridgelet coefficients ...... 103
Figure 9.1.16: Reconstruction with 256 Meyer wavelet coefficients 103
Figure 9.1.17: Reconstruction with 1024 Ridgelet coefficients ...... 103
Figure 9.1.18: Reconstruction with 1024 Meyer wavelet coefficients 103
Figure 9.1.19: Original image (cube)........................................ 104
Figure 9.1.20: Reconstruction with 64 Ridgelet coefficients ......... 105
Figure 9.1.21: Reconstruction with 64 Meyer wavelet coefficients . 105
Figure 9.1.22: Reconstruction with 256 Ridgelet coefficients ....... 105
Figure 9.1.23: Reconstruction with 256 Meyer wavelet coefficients 105
Figure 9.1.24: Reconstruction with 1024 Ridgelet coefficients ....... 106
Figure 9.1.25: Reconstruction with 1024 Meyer wavelet coefficients 106
Figure 9.1.26: Original image (lined up cubes) ......................... 106
Figure 9.1.27: Reconstruction with 1024 Ridgelet coefficients ..... 107
ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor, Dr. David G. Costa, for his guidance, unfailing patience and mentorship during this course of study. He has been an ever-present force in helping me to mature as a student and as a researcher.

I am especially grateful to Dr. Evangelos Yfantis whose help and support made it possible for me to be here today. His guidance and consistence advice helped me get through the first stages of my graduate student life at UNLV.

I am very grateful to Dr. Corran Webster and Dr. Xi Li for taking time out of their schedules to be members of my committee.

Special gratitude is devoted to my dear friend Theodores Lazarakis, who kindly and patiently helped me with all of the practical details in this thesis. I will always be grateful to him for putting up with me and for providing such a rich source of conversation, education and entertainment. I also appreciate his competent and meticulous review of this manuscript.
Last, but not least, I would like to thank my family. I owe my own interest in individual development to the belief they have demonstrated in me from day one. I want to thank my parents who gave their best so that their children will prosper. To my father who keeps believing in his “little girl” and never ceases to encourage me and demonstrate his love. To my brother and his wife for putting up with me and listen to my complaints. Their love, support, patience and constant back-up helped me get through this experience in one piece. To my cousins whose belief in me made most of this possible. I am indebted to them for all their support and unconditional help along all this years, and for always being there for me and my family. I owe everything I am and everything I have accomplished to my wonderful family.
INTRODUCTION

Wavelet theory is a relatively recent development in Applied Mathematics. Most of its success lies on the fact that wavelets are a fairly simple mathematical tool with a great variety of possible applications. One of the most appealing features of wavelet bases is that they provide efficient representation of functions which are smooth away from zero-dimensional (point-like) singularities. Unfortunately, wavelet bases are seriously challenged by n-dimensional singularities ($n \geq 1$). It was motivated by these limitations that Emmanuel Candès and David Donoho [4] introduced a new wavelet-like transform called the Ridgelet Transform.

The idea underlying the definition of this new transform is that of representing functions by superpositions of “Ridge Functions”, that is, by superpositions of functions of the form $\rho(x_1, \ldots, x_n) = r(a_1 x_1 + \ldots + a_n x_n)$ where $r(\cdot)$ is a scalar function. Thus, the function $r(\cdot)$ is the profile of the ridge function as the ridge is traversed orthogonally to its level sets. The terminology "Ridge Function" arose first in tomography and Ridgelet analysis makes use of a key tomographic concept, the Radon Transform.
The goal of this thesis is not only to introduce the Ridgelet transform but to show, by means of practical results in $\mathbb{R}^2$, the superiority of this new transform over the Wavelet transform when dealing with functions with singularities along lines.

Given all the connections that the Ridgelet transform has with the Radon transform and the theory of wavelets, the outline of this thesis is as follows. In Chapter 2 we introduce the Continuous Radon Transform along with its most important properties as well as its connection to the Fourier transform. Section 2.1 presents the reader with a re-definition of the Radon transform, the Slant Stack Radon transform, which will play a key role later in the definition of the Discrete Ridgelet transform.

Chapter 3 deals with the discretization of the Radon transform based on the Slant Stack definition given in Chapter 2.

Given that Frames play a very important role in the process of discretizing both the Wavelet and the Ridgelet transforms, Chapter 4 presents the reader with some of the most important results of the theory of Frames and its connections to Riesz bases. The importance of frames lies in the fact that in order to obtain numerically stable reconstruction for a function $f$ from its coefficients $\langle f, \phi_n \rangle$ it is required that the discrete collection $\{\phi_n\}$ constitute a frame.

Chapter 5 serves as an overview of the Wavelet transform (the continuous and discrete versions) as it also presents the definition of the
family of Meyer wavelets. This family of wavelets will be one of the pillars in the construction of the Discrete Ridgelet transform.

Chapter 6 and 7 deal with the Ridgelet transform itself. In Chapter 6 we introduce the definition and main properties of the Continuous Ridgelet Transform while in Chapter 7 a definition for the Discrete Ridgelet Transform is given. As we mentioned before, this discretization process involves the Discrete Slant Stack (Radon) transform and the discrete Meyer wavelet transform.

Finally, in Chapter 8 we present the reader with some facts about the theory involved in the algorithmic implementation of the Ridgelet transform. Section 8.2 contains the source code written in Microsoft Foundation Classes (MFC) of the most important functions in the algorithm which are used later to compare the performance of both transformations.

In Chapter 9 we compare the performance of the Ridgelet and Wavelet transform at representing objects with linear singularities. Such comparison is done by first applying (separately) both transforms to different images with linear singularities and then reconstructing them with a certain number of significant coefficients (defined by the user). Finally, section 9.2 contains some remarks on the very latest work done in Ridgelet analysis.
CHAPTER 2

THE RADON TRANSFORM

Let \( f \in \mathcal{S}(\mathbb{R}^n) \). The Radon Transform of \( f \) is a function \( Rf \) defined on hyperplanes; the value of \( Rf \) at a given hyperplane is the integral of \( f \) over that hyperplane.

Definition

Let \( s \in \mathbb{R} \), \( \omega \in S^{n-1} \). Then

\[
R[f](s, \omega) = \int_{s=0}^{s} f(x) dS_x = \int_{\mathbb{R}^n} f(x) \delta(s-x \cdot \omega) \, dx,
\]

i.e., \( R[f](s, \omega) \) is the integral of \( f \) over the hyperplane perpendicular to \( \omega \) at a signed distance \( s \) from the origin. Therefore, the Radon Transform of a function in \( \mathbb{R}^n \) gives the totality of all integrals of \( f \) over all hyperplanes in \( \mathbb{R}^n \). Note that

\[
\int_{-\infty}^{\infty} Rf(s, \omega) \, ds = \int_{\mathbb{R}^n} f(x) \, dx
\]

Let us state, without proofs, some of the most important properties of the Radon Transform [8, 16].
Properties

1- The Radon Transform is a linear transformation from $S(\mathbb{R}^n)$ to $S(\mathbb{R} \times S^{n-1})$.

2- $Rf(-s,-\omega) = Rf(s,\omega)$, i.e., $Rf$ is an even function of $(s,\omega)$.

3- $R[f(x-a)](s,\omega) = Rf(s-a\cdot\omega,\omega)$, $a \in \mathbb{R}^n$

4- $Rf$ can be extended to $\mathbb{R} \times \mathbb{R}^n / \{0\}$ as an even homogeneous function of degree -1, namely,

$$Rf(ks,k\omega) = |k|^{-1} Rf(s,\omega), \ k \neq 0.$$

Next, we study the relation between the Radon Transform and the Fourier Transform [13, 6]. We will denote the Fourier Transform of $f$ by $\hat{f}$, i.e.,

$$\hat{f}(\xi) = (2\pi)^{-\frac{n}{2}} \int f(x) e^{-ix \cdot \xi} dx.$$

By using polar coordinates we can write $\xi = \rho \omega$, $\omega \in S^{n-1}$, so that

$$\hat{f}(\rho \omega) = (2\pi)^{-\frac{n}{2}} \int f(x) e^{-i\rho (x \cdot \omega)} dx = (2\pi)^{-\frac{n}{2}} \int_{-\infty}^{0} \int f(x) e^{-i\rho (x \cdot \omega)} dS_x \, ds,$$

i.e.,

$$\hat{f}(\rho \omega) = (2\pi)^{-\frac{n}{2}} \int_{-\infty}^{0} e^{-i\rho s} Rf(s,\omega) \, ds.$$

In other words, we have

$$\hat{f}(\rho \omega) = (2\pi)^{-(n-1)} 2 \mathcal{F}_r [Rf(\cdot,\omega)](\rho) \quad \text{(I)}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
where $\mathcal{F}_r$ denotes the one-dimensional Fourier Transform along the radial direction. This result is called the Fourier Slice Theorem.

Graphically,

On the other hand, from the inversion formula of the Fourier Transform it follows that

$$Rf(s, \omega) = (2\pi)^{n/2} \mathcal{F}_r^{-1} \left[ \hat{f}(\rho \omega) \right](s, \omega) = (2\pi)^{n/2} \int_{-\infty}^{\infty} e^{i\rho \omega} \hat{f}(\rho \omega) d\rho \quad (II)$$

Finally, we state (again without proof) the basic results on the Radon Transform.

Theorem 2.1: [Inversion Formula]

For $f \in S(\mathbb{R}^n)$ one has that

$$f(x) = \int_{|t|=1} h(x \cdot \omega, \omega) d\omega$$

where

$$h(s, \omega) = \begin{cases} \frac{(2\pi)^{n+1}}{2} \left( \frac{1}{i} \frac{\partial}{\partial s} \right)^{n-1} Rf(s, \omega), & n \text{ odd.} \\ i \left( \frac{2\pi}{2} \right)^{n+1} H \left( \frac{1}{i} \frac{\partial}{\partial s} \right)^{n-1} Rf(s, \omega)(t), & n \text{ even.} \end{cases} \quad (III)$$
with $H$ being the Hilbert transform with respect to $s$, [see Appendix A1].

To simplify the notation let us denote $h = \mathcal{K}f$, where the definition of the operator $\mathcal{K}$ is given in (III). Thus,

$$f(x) = \int_{|\omega|=1} \mathcal{K}f(x, \omega, \omega) \, d\omega.$$ 

This inversion formula is in the form derived by Radon and although is very concise is not easy to implement on a discrete form (e.g., the Hilbert transform has a singularity which makes the discrete implementation very difficult). Later in this chapter we will introduce a different version of the Radon transform that will lead to an easy discrete implementation of the transform and its inverse.

Before presenting our next theorem let us compute the Adjoint Radon Transform. To do so, let $\mathcal{K}g(s, \omega)$ be defined as above and $\varphi \in \mathcal{S}$. Then,

$$\langle \varphi, R[f] \rangle = \int_{-\infty}^{\infty} \int_{|\omega|=1} \varphi(s, \omega) \overline{Rf(s, \omega)} \, ds \, d\omega$$

$$= \int_{-\infty}^{\infty} \int_{|\omega|=1} \varphi(s, \omega) \overline{f(x)} \, \delta(s-x, \omega) \, dx \, ds \, d\omega$$

$$= \int f(x) \int_{|\omega|=1} \varphi(x, \omega, \omega) \, d\omega \, dx = \int \overline{f(x)} \left( R^* \varphi \right)(x) \, dx = \langle R^* \varphi, f \rangle.$$ 

Therefore,

$$\left[ R^* \varphi \right](x) = \int_{|\omega|=1} \varphi(x, \omega, \omega) \, d\omega,$$

(IV)
so that the adjoint Radon transform corresponds to integration of \( \varphi \) over all hyperplanes passing through a given point. In the literature of computed tomography this operator is called the "Backprojection Operator" [8].

In view of the above, Theorem 2.1 can be rephrased as

Theorem 2.1:

If \( f \in \mathcal{S} \) then \( f = R'K[Rf] \), where \( R \) is the Radon transform, \( R' \) its adjoint and \( K \) the operator defined in (III).

Finally we state a Parseval relation for the Radon transform.

Theorem 2.2:

If \( f \in \mathcal{S}(\mathbb{R}^*) \) then

\[
\int |f|^2 \, dx = \int \int |\sqrt{K}Rf|^2 \, ds \, d\omega ,
\]

where the operator \( \sqrt{K} \) is defined through the formula

\[
\mathcal{F}_i(\sqrt{K}g)(\rho) = \left[ \frac{(2\pi)^{n+1}}{2} \right]^{1/2} |\rho|^{n+1} \hat{g}(\rho).
\]

Moreover, \( \sqrt{K}R \) is a unitary mapping of \( L^2(\mathbb{R}^n) \) onto \( L^2(\mathbb{R} \times S^{n-1}) \).

Proof:

From Theorem 2.1,
\[ \int |f|^2 \, dx = \int |f|^2 \, dx = \int \mathcal{F}(R \mathcal{K} R f)(x) \, dx = \int \int_{|\omega| = 1} |\mathcal{F}(R \mathcal{K} R f)(\omega)|^2 \, d\omega \]

= \int \int_{|\omega| = 1} |\mathcal{K} R f|^2 \, d\omega \, d\omega.

In other words, \( \sqrt{\mathcal{K}} R : S(\mathbb{R}^n) \subset L^2(\mathbb{R}^n) \rightarrow L^2(\mathbb{R} \times S^{n-1}) \) is an isometry. Hence, given that \( S(\mathbb{R}^n) \) is dense in \( L^2(\mathbb{R}^n) \), \( \sqrt{\mathcal{K}} R \) can be extended (uniquely, by continuity) to an isometry from \( L^2(\mathbb{R}^n) \) into \( L^2(\mathbb{R} \times S^{n-1}) \).

To prove that \( \sqrt{\mathcal{K}} R \) is onto it is sufficient to show that \( (\sqrt{\mathcal{K}} R)(S(\mathbb{R}^n)) \) contains some subset \( \mathcal{D} \) dense in \( L^2(\mathbb{R} \times S^{n-1}) \).

Let

\[ \mathcal{D} = \{ \varphi \in S(\mathbb{R} \times S^{n-1}) : F \varphi(\rho, \omega) = 0 \text{ for } \rho \text{ in some neighborhood of the origin} \} \]

Clearly \( \mathcal{D} \) is dense in \( L^2(\mathbb{R} \times S^{n-1}) \).

Now, given \( \varphi \in \mathcal{D} \) let \( g(\tau) = (2\pi)^{-n} \left[ T \varphi \right](\tau) \left[ \tau, T \varphi(\tau) \right] \). Then, \( g(\tau) \in S(\mathbb{R}^n) \) and there exists \( f \in S(\mathbb{R}^n) \) such that \( g = \hat{f} \). Let us show that \( \sqrt{\mathcal{K}} R f = \varphi \).

Since

\[ \mathcal{F}(\sqrt{\mathcal{K}} R f)(\rho, \omega) = \left[ \frac{2\pi}{2} |\rho|^{n-1} \right]^{1/2} \mathcal{F} f(\rho, \omega) = \left[ \frac{|\rho|^{n-1}}{2} \right]^{1/2} \hat{f}(\rho \omega) \]

applying the inverse Fourier transform we obtain that \( \sqrt{\mathcal{K}} R f = \varphi \). We have shown that \( \left( \sqrt{\mathcal{K}} R \right)(L^2(\mathbb{R}^n)) \supset \mathcal{D} \). Therefore, given the density of \( \mathcal{D} \)
in $L^2(\mathbb{R} \times S^{n-1})$, we conclude that $\sqrt{\mathcal{K}} R$ is a unitary mapping from $L^2(\mathbb{R}^n)$ onto $L^2(\mathbb{R} \times S^{n-1})$.

2.1 The Slant Stack Transform

The definition of the Radon Transform given in the previous section is usually referred to as the normal Radon transform due to its use of the normal equation of the hyperplane in its definition.

Another way of defining the Radon transform in $\mathbb{R}^2$ (useful in many applications) is through the use of the slope $p$ and line offset $\tau$ as the parameters. Thus the transformation is given by

$$R[f](p, \tau) = \int_{-\infty}^{\infty} f(x, px + \tau) \, dx.$$ 

This version of the Radon transform is referred to as the “Slant Stack” [1, 14]. In what follows we will always refer to the two dimensional case and we will denote the Slant Stack of $f$ as $Sf(p, \tau)$ to distinguish the two definitions.

As it is expected the normal Radon transform is related to slant stacking. In fact, if $\omega = (\cos \theta, \sin \theta)$ for some $0 \leq \theta < 2\pi$, we have that
\[ Rf(s, \omega) = \int \int f(x, y) \delta(s - x \cos \theta - y \sin \theta) \, dx \, dy = \]
\[ = \frac{1}{|\sin \theta|} \int \int f(x, y) \left[ y \left( \frac{s}{\sin \theta} - x \cot \theta \right) \right] \, dx \, dy = \]
\[ = \frac{1}{|\sin \theta|} \int f(x, \frac{s}{\sin \theta} - x \cot \theta) \, dx \]

so that, according to the previous definition,

\[ Rf(s, \omega) = \frac{1}{|\sin \theta|} \int f\left( x, \frac{s}{\sin \theta} - x \cot \theta \right) \, dx = \frac{1}{|\sin \theta|} Sf\left( -\cot \theta, \frac{s}{\sin \theta} \right) , \quad (I) \]

provided \( \theta \neq 0, \pi \).

As we mentioned in the abstract, the Radon Transform and, therefore, the Slant Stack Transform map line singularities into point singularities. Moreover, it can also be shown that a point singularity is transformed into a line singularity. To show this let us compute the Slant Stack transform of delta-type functions (distributions). In that case we need to extend the definition of the Slant Stack Transform (SST) to distribution spaces. This can be done by using the adjoint \( S' \) of the SST, given by

\[ \langle Sf, h \rangle = \int \int (Sf)(p, \tau) h(p, \tau) \, dp \, d\tau = \int \int f(x, y) (S'h)(x, y) \, dx \, dy = \langle f, S'h \rangle , \quad (II) \]

for all \( f, h \in C_0^\infty(\mathbb{R}^2) \).

So let us calculate \( S'h \), \( h \in C_0^\infty(\mathbb{R}^2) \). From the definition of the SST we have that,

\[ S[f](p, \tau) = \int_{-\infty}^{\infty} f(x, px + \tau) \, dx , \]
that is, the SST of \( f \) at the point \((p, \tau)\), amounts to integrating \( f \) over the line \( L = \{(x, px + \tau) : x \in \mathbb{R}\} \). Thus, for \( h \in C_0^\infty(\mathbb{R}^2) \) one obtains that

\[
[Sf, h] = \int \left( \int \left( \int f(x, px + \tau) \, dx \right) h(p, \tau) \, dp \, d\tau \right)
\]

\[
= \int \left( \int \left( \int f(x, px + \tau) \, dx \right) h(p, \tau) \, dp \, d\tau \right)
\]

\[
= \int \left( \int f(x, px + \tau) \, dx \right) h(p, \tau) \, dp \, d\tau
\]

\[
= \int \left( \int f(x, px + \tau) \, dx \right) h(p, \tau) \, dp \, d\tau
\]

\[
= \int f(x, px + \tau) \, dx \, dp \, d\tau
\]

so that

\[
[Sf, h] = \int f(x, px + \tau) \, dx \, dp \, d\tau
\]

\[
= \int f(x, px + \tau) \, dx \, dp \, d\tau
\]

Note: \( S'h \) does not have compact support even if \( h \) does. In fact, since \( S'h \) amounts to integrating \( h \) over all the lines of the form \( L = \{(p, px + \tau) : p \in \mathbb{R}\} \), it is clear that the support of \( S'h \) extends over all the \( x- \) and \( y- \)-axis; therefore, it only holds that \( S'h \in C^\infty \).

Let us now consider \( h \) to be the Dirac delta distribution concentrated on a point, i.e., \( h = \delta_{(x_0, y)} \). Then, using (II) and for every \( \varphi \in C_0^\infty \) we define:

\[
\langle S\delta_{(x_0, y)}, \varphi \rangle = \langle \delta_{(x_0, y)}, S'\varphi \rangle.
\]

Thus, in view of (IV)

\[
\langle S\delta_{(x_0, y)}, \varphi \rangle = (S'\varphi)(x_0, y) = \int_{-\infty}^{\infty} \varphi(x_0, px + y) \, dp
\]
which shows that the support, $\text{supp}(S\delta(x_0,y_0))$, of $S\delta(x_0,y_0)$ is the line $\tau = -px_0 + y_0$ in the $p-\tau$ plane. We can say that "the point $(x_0,y_0)$ is transformed into the line $\tau = -px_0 + y_0$ under the SST".

We present the reader with a schematic interpretation of what we have just shown.

![Diagram](image)

**Figure 2.1 – Slant Stack Transform**

of the point $(x_0,y_0)$.

Next let us compute the SST of a line segment, say $l = \{(x,p'x + \tau) : x_0 \leq x \leq x_i\}$. That is, let us compute the SST of the $\delta$-

distribution concentrated on $l$ which is given by:

$$\langle \delta_l, \psi \rangle = \int_{x_0}^{x_i} \psi(x, p'x + \tau) \, dx \quad \text{for all } \psi \in C^\infty(\mathbb{R}^2). \quad \text{(V)}$$

Thus, using (II) and (V) we have that, for every $\varphi \in C^\infty_c(\mathbb{R}^2)$,
\[
(S\delta, \varphi) = \delta \ast S\varphi = \int_{x_0}^{x_1} (S\varphi)(x, p'x + \tau') \, dx = \int_{-\infty}^{\infty} \phi(p, -px + p'x + \tau') \, dx \, dp
\]

\[
= \int_{-\infty}^{\infty} \phi(p, t) \, dt \, dp.
\]

It then follows that \((S\delta, \varphi)\) amounts to integration of \(\varphi\) over all the lines in the \(p-\tau\) plane passing through the point \((p', \tau')\) and with slopes ranging in the interval \([-x_i, -x_0]\). Schematically,

![Figure 2.2 - Slant Stack Transform](image)

of the line segment \(l = \{(x, p'x + \tau') : x_0 \leq x \leq x_1\}\).

In the next chapter we will discuss a discretization of the Radon Transform based on the Slant Stack definition given in this section and later we will introduce the Discrete Ridgelet transform, Chapter 7, which will also be given in terms of the Discrete Slant Stack.
Before that, we observe that in (1) the definition of the Slant Stack is restricted to values of $\theta$ different from 0 and $\pi$. To overcome this mishap we can restate the definition of the Slant Stack Transform. In fact, given that we are considering lines of the form $y = p\cdot x + \tau$ we will divide the set of all those lines into two different subsets. We will call a "basically horizontal line" a line of the form $y = p\cdot x + \tau$ with $|p| \leq 1$. Analogously, a "basically vertical line" will be that of the form $x = p\cdot y + \tau$ with $|p| \leq 1$. If we use angles to represent the slope of such lines, the equations will be given by

"basically horizontal lines" \hspace{1cm} $y = (\tan \theta) x + \tau$ \hspace{1cm} for $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ \hspace{1cm}

"basically vertical lines" \hspace{1cm} $x = (\cot \theta) y + \tau$ \hspace{1cm} for $\theta \in [\frac{\pi}{4}, \frac{3\pi}{4}]$.

It only remains to restate the Slant Stack definition.

Definition: [Slant Stack]

The Slant Stack operator is defined by

$$(Sf)(r, \theta) = \begin{cases} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(y - (\tan \theta)x - \tau) \, dy \, dx \quad & \text{for } \theta \in [-\frac{\pi}{4}, \frac{\pi}{4}] \\ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta((\cot \theta)y - x - \tau) \, dx \, dy \quad & \text{for } \theta \in [\frac{\pi}{4}, \frac{3\pi}{4}] \end{cases}$$

where $r \in \mathbb{R}$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER 3

THE DISCRETE RADON TRANSFORM

Let us now present the discretization of the two dimensional Radon Transform [14] based on the Slant Stack definition. We want to define the transformation for data in an $n$ by $n$ grid.

Let $I(u,v), -\frac{n}{2} \leq u, v < \frac{n}{2}$, be an $n$ by $n$ array, $p$ a slope and $\tau$ an offset. We define the Slant Stack Radon transform of $I(u,v)$ as follows:

$$
(S_{1}I)(\tau, p) = \text{Radon}\{y = px + \tau\}, I)
$$

$$
(S_{2}I)(\tau, p) = \text{Radon}\{x = py + \tau\}, I)
$$

where

$$
(S_{1}I)(\tau, p) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}-1} \tilde{I}_{y}(u, pu + \tau)
$$

and

$$
(S_{2}I)(\tau, p) = \sum_{v=-\frac{n}{2}}^{\frac{n}{2}-1} \tilde{I}_{x}(pv + \tau, v).
$$

The operators $\tilde{I}_{y}$ and $\tilde{I}_{x}$ are interpolants in the $y$ - variable and the $x$ - variable, respectively, for summing along the slanted lines may yield values not present in the original array $I$. The interpolation is accomplished by means of the Dirichlet kernel
\[ D_m(t) = \frac{\sin(\pi t)}{m \sin(\pi t/m)} \]

where \( m = 2n \). This choice of \( m \) guarantees that there is no wrap-around effects when calculating the sums along slanted lines [see Appendix A2].

More specifically, the interpolants \( \tilde{I}^1 \) and \( \tilde{I}^2 \) are given by

\[ \tilde{I}^1(u,v) = \sum_{n=-\gamma_2}^{\gamma_2} I(u,v) D_m(v-n) \]

and

\[ \tilde{I}^2(x,v) = \sum_{n=-\gamma_2}^{\gamma_2} I(u,v) D_m(x-u) \]

If \( \theta \) is the angle associated to the slope \( p \), the definition reads

\[
(\mathcal{SI})(\tau, \theta) = \begin{cases} 
(\mathcal{S}_1 I)(\tau, \theta) = \text{Radon}(\{y = (\tan \theta) x + \tau, I\}) & \text{for } \theta \in \left[ -\frac{\pi}{4}, \frac{\pi}{4} \right] \\
(\mathcal{S}_2 I)(\tau, \theta) = \text{Radon}(\{x = (\cot \theta) y + \tau, I\}) & \text{for } \theta \in \left[ \frac{\pi}{4}, \frac{3\pi}{4} \right] 
\end{cases}
\]

To have a complete discretization of the Radon transform it remains to suitably discretize the parameters \( \theta \) and \( \tau \). Since we are summing along lines we want to choose \( \tau \) so that the line \( y = (\tan \theta) x + \tau \) intersects the bounding of the box \( I(u,v) \). It is easy to see that such range of intercepts \( \tau \) must be given by \( T_n = \{ \tau \in \mathbb{Z} : -n \leq \tau < n \} \).

The next step is to discretize the set of angles \( \theta \). Let \( \Theta \) be the set of angles given by \( \Theta = \Theta_1 \cup \Theta_2 \) where

\[ \Theta_1 = \left\{ \theta_{i}^{l} : \theta_{i}^{l} = \arctan \left( \frac{2l}{n} \right), I \in \mathbb{Z} \cup \left[ -\frac{n}{2} \leq l < \frac{n}{2} \right] \right\} \]
and

\[ \Theta_2 = \left\{ \theta_{l,n}^2 : \theta_{l,n}^2 = \frac{\pi}{2} - \arctan \left( \frac{2l}{n} \right), l \in \mathbb{Z}, -\frac{n}{2} \leq l < \frac{n}{2} \right\}. \]

Note that if \( \theta_{l,n}^1, \theta_{n+1,n}^1 \in \Theta_1 \), the slopes associated to those angles are

\[ s_1 = \tan(\theta_{l,n}^1) = \frac{2l}{n} \quad \text{and} \quad s_2 = \tan(\theta_{n+1,n}^1) = \frac{2(l+1)}{n} \]

and their difference is

\[ s_2 - s_1 = \frac{2(l+1)}{n} - \frac{2l}{n} = \frac{2}{n}, \]

i.e., the angles in \( \Theta_1 \) define a set of equispaced slopes. The same argument applies for \( \Theta_2 \), so that \( \Theta \) is the set of angles induced by lines with equally spaced slopes. Figures 3.1 and 3.2 show those sets of lines.

![Figure 3.1 - Set of basically vertical lines](image-url)
We can now define the Discrete Slant Stack (Radon) Transform (DSS).

Definition: [Two dimensional Discrete Slant Stack Transform]

Let $\theta \in \Theta$, $r \in T_s$. We define

$$\{S_I\}(t,\beta) = \text{Radon}\{y = \left(\tan \theta\right)x + \tau\}, I$$

for $\theta \in \Theta_1$.

$$\{S_I\}(r,\beta) = \text{Radon}\{x = \left(\cot \theta\right)y + \tau\}, I$$

for $\theta \in \Theta_2$.

With this definition we convert an $n \times n$ array $I(u,v)$ into a $2n \times 2n$ array $(SI)(r,\theta)$.

As we had in the continuum case, there is a relationship between the Slant Stack (Radon) Transform and the Fourier transform which is given by the Fourier Slice Theorem.
Theorem 3.1: [Fourier Slice Theorem]

Given an $n$ by $n$ array $I$,

$$T_i(S_o I)(k) = \begin{cases} \hat{i}(-(\tan \theta) k, k) & \text{for } \theta \in \Theta_1 \\ \hat{i}(k,-(\cot \theta) k) & \text{for } \theta \in \Theta_2 \end{cases}$$

where $k \epsilon \mathbb{Z}, n \leq k < n$, $T_i$ is the one dimensional Fourier transform and

$$\hat{i}(\xi_1, \xi_2) = \sum_{u=-n/2}^{n/2-1} \sum_{v=-n/2}^{n/2-1} I(u,v) e^{-\frac{2\pi i (\xi_1 u + \xi_2 v)}{n}} \quad \text{(II)}$$

NOTE: For a complete and detailed proof of Theorem 3.1 refer to [1,14].

The set of points

$$\left\{ \left(-\frac{2l}{n}, k\right), \left(k,-\frac{2l}{n}\right) : l, k \epsilon \mathbb{Z}, \ -\frac{n}{2} \leq l < \frac{n}{2}, \ -n \leq k < n \right\}$$

is called the "pseudo-polar grid" [Figure 3.3]. The resampling of $\hat{i}$ at the new grid is referred to as the "Pseudopolar Fourier Transform".

Figure 3.3 - Pseudopolar grid.
Definition: [Pseudopolar Fourier Transform]

The Pseudopolar Fourier transform $P\mathcal{F}_i, i=1,2,$ is the linear transformation given by

$$P\mathcal{F}_1(k,l) = \hat{i} \left( -\frac{2l}{n}, k \right)$$

$$P\mathcal{F}_2(k,l) = \hat{i} \left( k, -\frac{2l}{n} \right)$$ \hspace{1cm} (III)

for $-\frac{n}{2} \leq l < \frac{n}{2}$, $-n \leq k < n$ where $\hat{i}$ is given by (II).

Combining the above definition and Theorem 3.1 it follows that

$$\left( S_{\theta} \right)(\cdot) = \begin{cases} 
\mathcal{F}_1^{-1} \cdot P\mathcal{F}_1(\cdot, l) & \text{for } \theta \in \Theta_1 \\
\mathcal{F}_1^{-1} \cdot P\mathcal{F}_2(\cdot, l) & \text{for } \theta \in \Theta_2 
\end{cases}$$ \hspace{1cm} (IV)

Thus, to obtain a row in $(SI)(k,\theta)$ we need to fix $\theta$ and compute the inverse one dimensional Discrete Fourier Transform (DFT) of the corresponding row (for that $\theta$) in the matrix $P\mathcal{F}_i, i=1,2$ (the choice of $i$ will also depend on $\theta$).

3.1 Inversion of the Discrete Slant Stack Transform

According to (IV) $(SI)(k,\theta)$ is invertible if $\mathcal{F}_i^{-1} \cdot P\mathcal{F}_i, i=1,2,$ is invertible. Since $\mathcal{F}_i^{-1}$ is invertible, it remains to check the invertibility of the operator $P\mathcal{F}_i, i=1,2$ [14].

Let us first consider $P\mathcal{F}_1$. According to (II) and (III) we have that
$$P T_{i} (k,l) = \hat{I} \left( - \frac{2l}{n}, k, l \right) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}-1} \sum_{v=-\frac{n}{2}}^{\frac{n}{2}-1} I(u,v) e^{\frac{2\pi i k v}{n}} e^{\frac{-2\pi i k l}{n}} = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}-1} c_{u} (u) e^{-\frac{2\pi i k u}{n}},$$

where

$$c_{u} (u) = \sum_{v=-\frac{n}{2}}^{\frac{n}{2}-1} I(u,v) e^{\frac{2\pi i k v}{n}} .$$

Now, for each fixed $k_{o} \neq 0$ (i.e., for a fixed column of $P T_{i}$) let

$$T_{n} (x) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}-1} c_{u} (u) e^{-\frac{2\pi i k_{o} u}{n}} .$$

Then $T_{n}$ is a trigonometric polynomial. Since we know the values $T_{n} (\frac{-2l k_{o}}{n}) = \hat{I} \left( \frac{-2l k_{o}}{n}, k_{o} \right)$ for $-\frac{n}{2} \leq l < \frac{n}{2}$, i.e., we know the value of $T_{n}$ at $n$ distinct points, we can uniquely determine the coefficients $c_{u} (u), u \in \mathbb{Z}, -\frac{n}{2} \leq u < \frac{n}{2}$. Thus, we can compute the values of $T_{n}$ at any given point. In particular, we can calculate its values at any integer point $j$ obtaining

$$T_{n} (j) = \hat{I} (j, k_{o}), k_{o} \neq 0 .$$

(V)

Hence, by means of the 2D-Inverse Discrete Fourier Transform (IDFT), we can recover the values $I(j, k_{o}), k_{o} \neq 0, j \in \mathbb{Z}$.

The next step will be to recover the samples $I(j, 0)$. To do so let us consider the matrix $P T_{i}$. As before, for each $k_{o} \neq 0$ consider the column vector
Similar to before we consider the trigonometric polynomial
\[ \tilde{t}_k(x) = \sum_{v=-\gamma/2}^{\gamma-1} d_k(v) e^{\frac{2\pi j xv}{\gamma}}. \]
Then, knowing the values \( \tilde{t}_k(-2l/n k_o) \) for \(-\gamma/2 \leq l < \gamma/2\) allows us to uniquely determine the coefficients \( d_k(v) \) and therefore the polynomial \( \tilde{t}_k \) itself. Hence we can compute the values \( \tilde{t}_k(j) = \hat{I}(k_o, j), -n \leq j < n \). In particular,
\[ \tilde{t}_k(0) = \hat{I}(k_o, 0) \text{ for all } -n \leq k_o < n, k_o \neq 0, \] (VI)
so that, by means of the 2D-IDFT, we can recover \( I(k_o, 0) \), \( k_o \neq 0 \).

From (V) and (VI) it follows that we can recover the values \( I(k, j), k \neq 0, j \neq 0 \). It remains to obtain \( I(0, 0) \). But \( P\mathcal{F}_z I(0, 0) = \hat{I}(0, 0) \), so that \( I(0, 0) = \mathcal{F}_z^{-1} P\mathcal{F}_z I(0, 0) = \mathcal{F}_z^{-1} \hat{I}(0, 0) \). Thus, we have shown that we can recover \( \hat{I} \) from \( P\mathcal{F}_z I, i = 1, 2 \), so that applying the 2D-IDFT yields the original image \( I(u, v) \). Hence, the Discrete Slant Stack given in (IV) is invertible.
CHAPTER 4

FRAMES

The theory of frames provides an algorithm to express a function as a linear combination of the frame elements. Namely, if $\mathcal{H}$ is a Hilbert space and $\{\phi_j\}_{j\in J}$ a frame, an element $f \in \mathcal{H}$ is completely characterized by its coefficients $\{\langle f, \phi_j \rangle\}_{j \in J}$ and can be reconstructed from them via a simple and numerically stable algorithm.

Definition

A family of functions $\{\phi_j\}_{j \in J}$ in a Hilbert space $\mathcal{H}$ is called a "frame" if there exist constants $A > 0$ and $B < \infty$ so that for all $f \in \mathcal{H}$,

$$A\|f\|^2 \leq \sum_{j \in J} |\langle f, \phi_j \rangle|^2 \leq B\|f\|^2 \quad (I)$$

We call the best constants $A$ and $B$ the frame bounds.

If $A = B$, then we say the frame is "tight". In such a case we have that for all $f \in \mathcal{H}$,

$$\sum_{j \in J} |\langle f, \phi_j \rangle|^2 = A\|f\|^2$$

which implies, by the polarization identity, [see Appendix B1],

24
\[ A \langle f, g \rangle = \sum_{j \in J} \langle f, \varphi_j \rangle \langle \varphi_j, g \rangle \]
or,

\[ f = A^{-1} \sum_{j \in J} \langle f, \varphi_j \rangle \varphi_j \quad (\text{II}) \]
at least in the weak sense.

Although formula (II) looks like that of the expansion of \( f \) into an orthonormal basis, it is important to realize that frames, even tight frames, are NOT orthonormal bases.

Proposition 4.1:

If \( \{\varphi_j\}_{j \in J} \) is a tight frame for \( \mathcal{H} \), with frame bound \( A = 1 \), and if \( \|\varphi_j\| = 1 \) for all \( j \in J \), \( \{\varphi_j\}_{j \in J} \) constitutes an orthonormal basis.

Proof:

By hypothesis, \( \sum_{j \in J} |\langle f, \varphi_j \rangle|^2 = \|f\|^2 \) for all \( f \in \mathcal{H} \).

Hence, if \( \langle f, \varphi_j \rangle = 0 \) for all \( j \in J \), then \( \|f\|^2 = 0 \) so that \( f = 0 \).

Therefore, \( \overline{\text{span}} \{\varphi_j\}_{j \in J} = \mathcal{H} \). It remains to show that \( \{\varphi_j\}_{j \in J} \) is an orthonormal set.

For any \( i \in J \),

\[ \|\varphi_i\|^2 = \sum_{j \in J} |\langle \varphi_i, \varphi_j \rangle|^2 = \|\varphi_i\|^2 + \sum_{j \neq i} |\langle \varphi_i, \varphi_j \rangle|^2 = 1 + \sum_{j \neq i} |\langle \varphi_i, \varphi_j \rangle|^2. \]

Since \( \|\varphi_i\|^2 = 1 \) we get that \( \sum_{j \neq i} |\langle \varphi_i, \varphi_j \rangle|^2 = 0 \), i.e., \( \langle \varphi_i, \varphi_j \rangle = 0 \), \( j \neq i \), \( j \in J \).
Therefore, \( \{ \varphi_j \}_{j \in J} \) is an orthonormal set that spans \( \mathcal{H} \) and it constitutes an orthonormal basis for \( \mathcal{H} \).

Let us now introduce the "pre-frame operator."

Definition:

If \( \{ \varphi_j \}_{j \in J} \) is a frame in \( \mathcal{H} \), then its associated **pre-frame operator** \( F \) is the linear operator from \( \mathcal{H} \) to \( l^2(J) \) defined by

\[
(Ff)_j = \langle f, \varphi_j \rangle \quad j \in J.
\]

It follows from (I) that

\[
\| Ff \|^2 = \sum_{j \in J} |(Ff)_j|^2 = \sum_{j \in J} |\langle f, \varphi_j \rangle|^2 \leq B \| f \|^2.
\]

Therefore, \( F \) is bounded. The adjoint \( F^* \) of \( F \) is easy to compute. In fact,

\[
\langle F^* c, f \rangle = \langle c, Ff \rangle = \sum_{j \in J} c_j \langle \varphi_j, f \rangle = \sum_{j \in J} c_j \langle f, \varphi_j \rangle = \sum_{j \in J} c_j \langle \varphi_j, f \rangle = \left\langle \sum_{j \in J} c_j \varphi_j, f \right\rangle
\]

for all \( f \in \mathcal{H} \). Hence,

\[
F^* c = \sum_{j \in J} c_j \varphi_j \tag{III}
\]

at least in the weak sense (in fact, the series in (III) converges in norm, [see Appendix B2]).

Since \( \| F^* \| = \| F \| \) we have that \( \| F^* c \| \leq B^2 \| c \| \). And, in view of
the frame condition (I) can therefore be rewritten as

\[ A \text{Id} \leq F^*F \leq B \text{Id}. \]  

(IV)

In particular, this implies that \( F^*F \) is invertible, in view of the following elementary lemma, whose proof we omit.

Lemma 4.2:

If a positive bounded linear operator \( S \) on \( \mathcal{H} \) is bounded below by a strictly positive constant \( \alpha \), then \( S \) is invertible and its inverse \( S^{-1} \) is bounded by \( \alpha^{-1} \).

Now, from (IV) and the above lemma we have that \( \| (F^*F)^{-1} \| \leq A^{-1} \).

Furthermore,

\[ B^{-1} \text{Id} \leq (F^*F)^{-1} \leq A^{-1} \text{Id} \]  

(V)

In fact, we already have \( \| (F^*F)^{-1} \| \leq A^{-1} \). On the other hand, since \( \| F^*F \| \leq B \),

\[ \| f \| = \| (F^*F)(F^*F)^{-1} f \| \leq \| F^*F \| \| (F^*F)^{-1} f \| \leq B \| (F^*F)^{-1} f \| \]

hence,

\[ B^{-1} \| f \| \leq \| (F^*F)^{-1} f \| \leq A^{-1} \| f \| . \]

Given that the inequality holds for all \( f \in \mathcal{H} \) we obtain the inequalities in (V).
By applying the operator \((F^*F)^{-1}\) to the vectors \(\varphi_j\) one obtains a new family of vectors, which we will denote by \(\tilde{\varphi}_j\):

\[
\tilde{\varphi}_j = (F^*F)^{-1}\varphi_j.
\]

The new family \(\{\tilde{\varphi}_j\}_{j \in J}\) turns out to be a frame as well.

Definition:
The operator \(F^*F\) is called the Frame operator associated to the frame \(\{\varphi_j\}_{j \in J}\).

Proposition 4.3:
The \(\{\tilde{\varphi}_j\}_{j \in J}\) constitute a frame with frame bounds \(B^{-1}\) and \(A^{-1}\),

\[
B^{-1}\|f\|^2 \leq \sum_{j \in J} \left| \langle f, \tilde{\varphi}_j \rangle \right|^2 \leq A^{-1}\|f\|^2. \tag{VI}
\]

Moreover, the associated pre-frame operator \(\tilde{F}: \mathcal{H} \to l^2(J)\),

\[
(\tilde{F}f)_j = \langle f, \tilde{\varphi}_j \rangle, \quad j \in J,
\]

is such that \(\tilde{F}^* \tilde{F} = \tilde{F}^* = \text{Id} \) and \(\tilde{F}^* \tilde{F} = \tilde{F} \tilde{F}^*\) is the orthogonal projection operator of \(l^2(J)\) onto \(\text{Range}(F) = \text{Range}(\tilde{F})\).

Proof:
We recall that if \(S\) is a bounded operator with inverse \(S^{-1}\) and if \(S^* = S\), then \((S^{-1})^* = S^{-1}\).

Therefore, from

\[
\langle f, \tilde{\varphi}_j \rangle = \langle f, (F^*F)^{-1}\varphi_j \rangle = \langle (F^*F)^{-1}f, \varphi_j \rangle
\]

we obtain
\[ \sum_{j \in J} \left| \langle f, \tilde{\varphi}_j \rangle \right|^2 = \sum_{j \in J} \left| \langle (F^*F)^{-1} f, \varphi_j \rangle \right|^2 = \| F(F^*F)^{-1} f \|^2 \]

\[ = \langle (F^*F)^{-1} f, F^*F(F^*F)^{-1} f \rangle = \langle (F^*F)^{-1} f, f \rangle \]  

(VII)

From (V) and (VII) it follows that \( B^{-1} \| f \|^2 \leq \sum_{j \in J} \left| \langle f, \tilde{\varphi}_j \rangle \right|^2 \leq A^{-1} \| f \|^2 \), so that \( \{ \tilde{\varphi}_j \}_{j \in J} \) constitutes a frame. Moreover, (VII) implies that \( \tilde{F}^* \tilde{F} = (F^*F)^{-1} \).

We omit the elementary (but technical) proof that \( \text{Range} (F) = \text{Range} (\tilde{F}) \) and that \( F \) (respectively \( \tilde{F} \)) is a one-to-one mapping from \( \mathcal{H} \) onto the \( \text{Range} (F) = \text{Range} (\tilde{F}) \) with inverse \( \tilde{F}^* \) (respectively \( F^* \)).

Definition:

The family \( \{ \tilde{\varphi}_j \}_{j \in J} \) is called the "dual frame" of \( \{ \varphi_j \}_{j \in J} \).

It can be easily checked that the dual frame of \( \{ \tilde{\varphi}_j \}_{j \in J} \) is the original frame \( \{ \varphi_j \}_{j \in J} \).

The previous conclusions can be expressed in a less abstract form.

Namely, \( \tilde{F}^* F = F^* \tilde{F} = \text{Id} \) is equivalent to saying that

\[ f = \sum_{j \in J} \langle f, \varphi_j \rangle \tilde{\varphi}_j = \sum_{j \in J} \langle f, \tilde{\varphi}_j \rangle \varphi_j \]  

(VIII)

In fact,

\[ f = F^* \tilde{F} f = \sum_{j \in J} \langle \tilde{F} f, \varphi_j \rangle \varphi_j = \sum_{j \in J} \langle f, \varphi_j \rangle \varphi_j \]

and, on the other hand,

\[ f = \tilde{F}^* F f = \sum_{j \in J} \langle \tilde{F} f, \tilde{\varphi}_j \rangle \tilde{\varphi}_j = \sum_{j \in J} \langle f, \varphi_j \rangle \tilde{\varphi}_j , \]

hence (VIII) follows.
This means that \( f \) can be reconstructed from the \( \langle f, \varphi \rangle \)'s and its reconstruction formula is given by (VIII). Furthermore, (VIII) gives us a formula for writing \( f \) as a superposition of the \( \varphi_j \)'s.

Now, since frames, even tight frames, are generally NOT bases (the \( \varphi_j \)'s are typically not linearly independent), given a function \( f \) there may exist many different expansions as superpositions of the \( \varphi_j \)'s. The importance of the frames lies in the fact that the coefficients \( \langle f, \varphi \rangle \) are the most economical. This result is given in detail in the following proposition.

Proposition 4.4:

If \( f = \sum_{j \in J} c_j \varphi_j \) for some \( c = (c_j)_{j \in J} \in l^2(J) \) and not all \( c_j \) are equal to \( \langle f, \varphi_j \rangle \) then

\[
\sum_{j \in J} |c_j|^2 > \sum_{j \in J} |\langle f, \varphi_j \rangle|^2.
\]

Moreover, if such \( c = (c_j)_{j \in J} \in l^2(J) \) exists, then

\[
\sum_{j \in J} |\langle f, \varphi_j \rangle|^2 + \sum_{j \in J} |\langle f, \varphi_j \rangle - c_j|^2 = \sum_{j \in J} |c_j|^2.
\]

Proof: [see Appendix B3]

Proposition 4.4 yields another important result. In fact, if we focus our attention on the \( \varphi_j \)'s in the first half of (VIII), we see that we may have
non-uniqueness there as well. That is, there may exist other families
\((u_j)_{j \in J}\) such that \(f = \sum_{j \in J} \langle f, \varphi_j \rangle u_j\). Then, by Proposition 4.4,
\[
\sum_{j \in J} |\langle u_j, g \rangle|^2 \geq \sum_{j \in J} |\langle \varphi_j, g \rangle|^2 \quad \text{for all } g \in \mathcal{H}.
\]

Finally, keeping in mind that \(\tilde{\varphi}_j = (F^*F)^{-1} \varphi_j\), \((\text{VIII})\) gives us a
reconstruction formula for \(f\) using the \(\langle f, \varphi_j \rangle\)'s. Thus, we only need to
compute the \(\tilde{\varphi}_j\), which involves inverting the frame operator \(F^*F\), \([7]\).

4.1 Bounded Unconditional Bases (Riesz Bases) And Frames

Let \(\{\varphi_n\}_{n \in \mathbb{N}}\) be a frame. From this point on \(S\) will denote the frame
operator \(F^*F\) associated with \(\{\varphi_n\}_{n \in \mathbb{N}}\).

Proposition 4.1.1:
The removal of a vector from a frame leaves either a frame or an
incomplete set. In particular,
\[
\langle \varphi_m, S^{-1} \varphi_m \rangle = 1 \quad \Rightarrow \quad \{\varphi_n\}_{n \neq m} \text{ is a frame.}
\]
\[
\langle \varphi_m, S^{-1} \varphi_m \rangle = 1 \quad \Rightarrow \quad \{\varphi_n\}_{n \neq m} \text{ is incomplete.}
\]

Proof: [see Appendix B4]

Definition:
A frame \(\{\varphi_n\}_{n \in \mathbb{N}}\) is exact if it ceases to be a frame whenever any single
element is deleted from the sequence.
Corollary 4.1.2:

If \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is an exact frame then, \( \{ \varphi_n \}_{n \in \mathbb{N}} \) and \( \{ S^{-i} \varphi_n \}_{n \in \mathbb{N}} \) are biorthogonal, i.e., \( \langle \varphi_n, S^{-i} \varphi_m \rangle = \delta_{mn} \)

Proof:

If \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is an exact frame Proposition 4.1.1 implies that \( \langle \varphi_m, S^{-i} \varphi_m \rangle = 1 \) for all \( m \) and, according to the proof of Proposition 4.1.1, \( \langle \varphi_n, S^{-i} \varphi_m \rangle = 0 \) for all \( n \neq m \). Then, \( \langle \varphi_n, S^{-i} \varphi_m \rangle = \delta_{mn} \).

A family \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is a “Riesz Basis” or a “bounded unconditional basis” for \( \mathcal{H} \) if there is bounded invertible operator \( U: \mathcal{H} \rightarrow \mathcal{H} \) and an orthonormal basis \( \{ e_n \}_{n \in \mathbb{N}} \) for \( \mathcal{H} \) such that \( U \varphi_n = e_n \).

Equivalently, \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is a Riesz basis for \( \mathcal{H} \) if and only if \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is complete in \( \mathcal{H} \) and there exist constants \( A', B' > 0 \) such that

\[
A' \left\| \sum_{n \in \mathbb{N}} |c_n|^2 \right\| \leq \left\| \sum_{n \in \mathbb{N}} c_n \varphi_n \right\|^2 \leq B' \left\| \sum_{n \in \mathbb{N}} |c_n|^2 \right\| \text{ for all finite sequences of scalars } \{ c_n \}.
\]

Also, \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is a Riesz basis if \( 0 < \inf_{n \in \mathbb{N}} \| \varphi_n \| \leq \sup_{n \in \mathbb{N}} \| \varphi_n \| < \infty \) and the series \( f = \sum_{n \in \mathbb{N}} c_n \varphi_n \) converges unconditionally for all \( f \), i.e., every permutation of the series converges (hence the expression “bounded unconditional basis”).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
It is easy to show that if \( \{\varphi_n\}_{n \in \mathbb{N}} \) is a Riesz basis then, \( \{\varphi_n\}_{n \in \mathbb{N}} \) is a frame. In fact, by definition of Riesz basis there exist a bounded invertible operator \( U: \mathcal{H} \to \mathcal{H} \) (i.e., \( U \) is linear, bijective, continuous and \( U^{-1} \) is continuous) and an orthonormal basis \( \{e_n\}_{n \in \mathbb{N}} \) for \( \mathcal{H} \) such that \( U e_n = \varphi_n \). Therefore, for all \( f \in \mathcal{H} \) we have that

\[
\sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2 = \sum_{n \in \mathbb{N}} |\langle f, U e_n \rangle|^2 = \sum_{n \in \mathbb{N}} \left| \langle U^* f, e_n \rangle \right|^2 = \sum_{n \in \mathbb{N}} \left| \langle f, U e_n \rangle \right|^2 = \|U^* f\|^2.
\]

On the other hand, since \( U \) is a topological isomorphism we have that

\[
\|U^{-1}\| \|f\| \leq \|U^* f\| \leq \|U\| \|f\|. 
\]

Then, by setting \( A = \left(\|U^{-1}\|^2\right)^{1/2} \) and \( B = \|U^*\|^2 \), one obtains that

\[
A \|f\|^2 \leq \sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2 \leq B \|f\|^2
\]

and it follows that \( \{\varphi_n\}_{n \in \mathbb{N}} \) is a frame.

**Theorem 4.1.3:**

Let \( \{\varphi_n\}_{n \in \mathbb{N}} \) be a frame. The following are equivalent:

1. \( \{\varphi_n\}_{n \in \mathbb{N}} \) is a Riesz basis for \( \mathcal{H} \).
2. \( \{\varphi_n\}_{n \in \mathbb{N}} \) is an exact frame.
3. \( \{\varphi_n\}_{n \in \mathbb{N}} \) is minimal, i.e., for all \( m, \varphi_m \notin \overline{\text{span}} \{\varphi_n\} \).
4. \( \{\varphi_n\}_{n \in \mathbb{N}} \) has a unique biorthogonal sequence in \( \mathcal{H} \), namely, 
   \[ \{S^{-1}\varphi_n\}_{n \in \mathbb{N}} \].
5- If $\sum_{n=1}^N c_n \varphi_n = 0$ in $\mathcal{H}$ for a sequence of scalars $(c_n) \in l^2$ then $c_n = 0$ for all $n$.

Proof:

Let $\{\varphi_n\}_{n \in \mathbb{N}}$ be a frame.

$[(1) \Rightarrow (2)]$

If $\{\varphi_n\}_{n \in \mathbb{N}}$ is a Riesz basis, there exist an orthonormal basis $\{e_n\}_{n \in \mathbb{N}}$ and a topological isomorphism $U: \mathcal{H} \to \mathcal{H}$ such that $U \varphi_n = e_n$; therefore, the removal of any vector from the frame is equivalent to the removal of a vector from $\{e_n\}_{n \in \mathbb{N}}$ which leaves an incomplete set. Therefore, $\{\varphi_n\}_{n \in \mathbb{N}}$ is an exact frame.

$[(2) \Rightarrow (1)]$

Let us show $\{\varphi_n\}_{n \in \mathbb{N}}$ is a bounded unconditional basis for $\mathcal{H}$.

Assume $\{\varphi_n\}_{n \in \mathbb{N}}$ is an exact frame with bounds $A$, $B$. From Corollary 4.1.2, $\{\varphi_n\}_{n \in \mathbb{N}}$ and $\{S^{-1} \varphi_n\}_{n \in \mathbb{N}}$ are biorthogonal. So for every $m$ fixed,

$$A \|S^{-1} \varphi_m\|^2 \leq \sum_{n \in \mathbb{N}} |\langle S^{-1} \varphi_m, \varphi_n \rangle|^2 = |\langle S^{-1} \varphi_m, \varphi_m \rangle|^2 \leq \|S^{-1} \varphi_m\|^2 \|\varphi_m\|^2$$

so that $A \leq \|\varphi_m\|^2$.

On the other hand,

$$\|\varphi_m\|^2 = |\langle \varphi_m, \varphi_m \rangle|^2 \leq \sum_{n \in \mathbb{N}} |\langle \varphi_m, \varphi_n \rangle|^2 \leq B \|\varphi_m\|^2$$

and $\|\varphi_m\|^2 \leq B$.

Hence, $A \leq \|\varphi_m\|^2 \leq B$ and $\{\varphi_n\}_{n \in \mathbb{N}}$ is bounded in norm.
It remains to show that $\{\varphi_n\}_{n \in \mathbb{N}}$ is an unconditional basis. For that we only need to prove that for all $f \in \mathcal{H}$ the representation formula

$$f = \sum_{n \in \mathbb{N}} \langle f, S^{-1} \varphi_n \rangle \varphi_n$$

is unique.

Suppose $f = \sum_{n \in \mathbb{N}} c_n \varphi_n$ for some $\{c_n\}_{n \in \mathbb{N}} \in \ell^2(\mathbb{N})$ then,

$$\langle f, S^{-1} \varphi_m \rangle = \sum_{n \in \mathbb{N}} c_n \langle \varphi_n, S^{-1} \varphi_m \rangle = \sum_{n \in \mathbb{N}} c_n \langle \varphi_n, S^{-1} \varphi_m \rangle = c_m \quad \text{for all } m.$$

From the above it follows that $\{\varphi_n\}_{n \in \mathbb{N}}$ is an unconditional basis for $\mathcal{H}$.

$[(2) \Rightarrow (3)]$

Suppose by contradiction that $\{\varphi_n\}_{n \in \mathbb{N}}$ is not minimal then there exists $m_0$ such that $\varphi_{m_0} \in \text{span} \{\varphi_n\}$, i.e., there exists a sequence $\{\Phi_j\}_{j \in \mathbb{N}} \subset \text{span} \{\varphi_n\}$ such that $\lim_{j \to \infty} \Phi_j = \varphi_{m_0}$. Moreover,

$$\Phi_j = \sum_{k=-N_j}^{N_j} a_k^j \varphi_k, \quad \text{for each } j \quad (\text{X})$$

Now, since $\{\varphi_n\}_{n \in \mathbb{N}}$ is a frame $f = \sum_{n \in \mathbb{N}} \langle f, S^{-1} \varphi_n \rangle \varphi_n$ for all $f \in \mathcal{H}$. In particular, from Proposition 4.1.1 one obtains that

$$\varphi_{m_0} = \sum_{n \in \mathbb{N}} \langle \varphi_{m_0}, S^{-1} \varphi_n \rangle \varphi_n = \langle \varphi_{m_0}, S^{-1} \varphi_{m_0} \rangle \varphi_{m_0} + \sum_{n \neq m_0} \langle \varphi_{m_0}, S^{-1} \varphi_n \rangle \varphi_n = \varphi_{m_0} + \sum_{n \neq m_0} \langle \varphi_{m_0}, S^{-1} \varphi_n \rangle \varphi_n.$$
Hence,

\[ \sum_{n\in\mathbb{N}} \langle \varphi_m, S^{-1} \varphi_n \rangle \varphi_n = 0 \]  \hfill (XI)

On the other hand,

\[
\langle \Phi_j, S^{-1} \varphi_n \rangle \varphi_n = \left( \sum_{k=N_j}^{N_j} \sum_{k=m_0}^{N_j} a_k' \varphi_k, S^{-1} \varphi_n \right) \varphi_n = \sum_{k=N_j}^{N_j} \sum_{k=m_0}^{N_j} a_k' \langle \varphi_k, S^{-1} \varphi_n \rangle \varphi_n
\]

so that

\[
\sum_{m=-M}^{M} \langle \Phi_j, S^{-1} \varphi_n \rangle \varphi_n = \sum_{m=-M}^{M} \sum_{k=N_j}^{N_j} \sum_{k=m_0}^{N_j} a_k' \langle \varphi_k, S^{-1} \varphi_n \rangle \varphi_n = \sum_{m=-M}^{M} \sum_{k=N_j}^{N_j} a_k' \langle \varphi_k, S^{-1} \varphi_n \rangle \varphi_n.
\]

Now, if we take the limit as \( M \to \infty \), from \((X)\) and the fact that \( \{f_n\} \) is an exact frame, it follows that

\[
\sum_{m=-M}^{M} \langle \Phi_j, S^{-1} \varphi_n \rangle \varphi_n = \sum_{k=N_j}^{N_j} \sum_{m=m_0}^{m_0} \sum_{m=m_0}^{m_0} a_k' \langle \varphi_k, S^{-1} \varphi_n \rangle \varphi_n =
\]

\[
= \sum_{k=N_j}^{N_j} a_k' \langle \varphi_k, S^{-1} \varphi_n \rangle \varphi_n =
\]

\[
= \Phi_j - \sum_{k=N_j}^{N_j} a_k' \langle \varphi_k, S^{-1} \varphi_m \rangle \varphi_m = \Phi_j.
\]

then, taking the limit as \( j \to \infty \) yields

\[
\sum_{m=m_0}^{m_0} \langle \varphi_m, S^{-1} \varphi_n \rangle \varphi_n = \varphi_m
\]

and from \((XI)\) it follows that \( \varphi_m = 0 \) so that \( \langle \varphi_m, S^{-1} \varphi_m \rangle = 0 \neq 1 \) which is a contradiction. Therefore, \( \{\varphi_n\}_{n\in\mathbb{N}} \) is minimal.
Since \( \{ \phi_n \}_{n \in \mathbb{N}} \) is minimal, Proposition 4.1.1 implies that \( \langle \phi_m, S^{-1} \phi_n \rangle = 1 \) (otherwise for all \( m \), \( \{ \phi_n \}_{n \in \mathbb{N}} \) is a frame and, therefore, complete). Then, from the proof of Proposition 4.1.1 we see that \( \langle \phi_n, S^{-1} \phi_m \rangle = 0 \) for all \( n \neq m \)

so that

\[ \langle \phi_n, S^{-1} \phi_m \rangle = \delta_{nm} \text{ for all } n,m. \]

It only remains to show that, under the above condition, \( \{ S^{-1} \phi_n \}_{n \in \mathbb{N}} \) is unique in \( \mathcal{H} \).

Assume to the contrary there exists a sequence \( \{ g_n \}_{n \in \mathbb{N}} \) in \( \mathcal{H} \) such that

\[ \langle \phi_n, g_m \rangle = \delta_{nm} \text{ and } g_{n_0} = S^{-1} \phi_{n_0} \text{ for some index } n_0. \]

Let

\[ h = g_{n_0} - S^{-1} \phi_{n_0}. \tag{XII} \]

Clearly \( h \neq 0 \) and,

\[ \langle h, \varphi_n \rangle = \langle g_{n_0} - S^{-1} \phi_{n_0}, \varphi_n \rangle = \langle g_{n_0}, \varphi_n \rangle - \langle S^{-1} \phi_{n_0}, \varphi_n \rangle = \delta_{n_0,n} - \delta_{n_0,n} = 0 \]

for all \( n \). That is, \( h \) is orthogonal to the set \( \{ \varphi_n \}_{n \in \mathbb{N}} \). Since \( \{ \varphi_n \}_{n \in \mathbb{N}} \) is a frame it is complete so that \( h = 0 \) which contradicts (XII). Therefore, \( \{ S^{-1} \phi_n \}_{n \in \mathbb{N}} \) is unique.
Let \((c_n)_{n \in \mathbb{N}} \in l^2\) be such that \(\sum_{n \in \mathbb{N}} c_n \varphi_n = 0\). Given that \(S\) is a topological isomorphism we have that \(\sum_{n \in \mathbb{N}} c_n S^{-1} \varphi_n = 0\). Therefore, one obtains that for all \(m\)

\[
0 = \left\langle \sum_{n \in \mathbb{N}} c_n S^{-1} \varphi_n, \varphi_m \right\rangle = \sum_{n \in \mathbb{N}} c_n \left\langle S^{-1} \varphi_n, \varphi_m \right\rangle = \sum_{n \in \mathbb{N}} c_n \delta_{nm} = c_m .
\]

From Proposition 4.1.1 we know that the removal of a vector from a frame leaves either a frame or an incomplete set. We must show that for all \(m\), \(\{\varphi_n\}_{n \neq m}\) is an incomplete set.

Assume to the contrary that there exists \(m\) such that \(\{\varphi_n\}_{n \neq m}\) is a frame. Then, for all \(f \in \mathcal{H}\),

\[
f = \sum_{n \neq m} \left\langle f, S^{-1} \varphi_n \right\rangle \varphi_n .
\]

In particular, \(\varphi_m = \sum_{n \neq m} \left\langle \varphi_m, S^{-1} \varphi_n \right\rangle \varphi_n \) or \(\varphi_m - \sum_{n \neq m} \left\langle \varphi_m, S^{-1} \varphi_n \right\rangle \varphi_n = 0\).

Letting \(c_n = \begin{cases} 1 & n = m \\ -\left\langle \varphi_m, S^{-1} \varphi_n \right\rangle & n \neq m \end{cases}\) it is clear that \((c_n)_{n \in \mathbb{N}} \in l^2\) and \(\sum_{n \in \mathbb{N}} c_n \varphi_n = 0\). Then, by hypothesis, \(c_n = 0\), for all \(n\) which is a contradiction. Therefore, \(\{\varphi_n\}_{n \neq m}\) is incomplete for all \(m\) and \(\{\varphi_n\}_{n \in \mathbb{N}}\) is an exact frame.

\[\blacksquare\]
The theory of Riesz bases and frames goes much further than what we have presented here. We conclude this chapter with a couple of remarks.

1- As a consequence of Theorem 4.1.3, we note that if \( \{\varphi_n\}_{n \in \mathbb{N}} \) is a frame but NOT a Riesz basis, there exist non-zero sequences \( (c_n)_{n \in \mathbb{N}} \in l^2 \) such that \( \sum_{n} c_n \varphi_n = 0 \). Therefore any function \( f \in L^2(\mathbb{R}) \) can be written as \( f = \sum_{n} (\langle f, S^{-1} \varphi_n \rangle + c_n) \varphi_n \), i.e., \( f \) has many representations as superpositions of the frame elements.

2- It is immediate that an inexact frame cannot be a basis for, by definition, there is an \( m \) such that \( \{\varphi_n\}_{n=m} \) is a frame, and hence complete, while no subset of a basis can be complete.
CHAPTER 5

THE WAVELET TRANSFORM

The theory of wavelets allows us to decompose a function as a linear combination of some basic functions of elementary form, namely “wavelets”, which are generated by translations and dilations of one single function, the “mother wavelet”. It turns out that the wavelet transform decomposes the function into different scales with different levels of resolution.

5.1 Continuous Wavelet Transform

Definition:

Let \( \psi \in L^2(\mathbb{R}) \). The function \( \psi \) is said to be a mother wavelet or an analyzing wavelet provided it satisfies

\[
C_\psi = 2\pi \int_{-\infty}^{\infty} \frac{\hat{\psi}(\xi)^2}{|\xi|} d\xi < \infty.
\]

The above condition is called the admissibility condition and is equivalent to saying that \( \psi \) has zero average, i.e., \( \int_{-\infty}^{\infty} \psi(t) dt = 0 \).
Then, for all real numbers $a$ and $b$ with $a \neq 0$, the continuous wavelet transform of a function $f$ with respect to $\psi$ is defined as

$$\mathcal{W}_\psi[f](a, b) = \int_{-\infty}^{\infty} f(x) \overline{\psi_{a,b}(x)} \, dx,$$

where,

$$\psi_{a,b}(x) = \frac{1}{\sqrt{|a|}} \psi\left(\frac{x-b}{a}\right).$$

Thus, the wavelet transform of $f$ at scale $a$ and position $b$ is computed by convolving $f$ with $\psi_a(x) = |a|^{-\frac{1}{2}} \psi\left(\frac{x}{a}\right)$ at $x=b$. Large values of $|a|$ correspond to small frequencies, small values of $|a|$ correspond to high frequencies. By changing the parameter $b$ we move the location center of $\psi_{a,b}$. Given that $\psi$ has zero average, a wavelet coefficient $\mathcal{W}_\psi[f](a, b)$ measures the correlation of $f$ with respect to $\psi_{a,b}$ in a neighborhood of $b$ whose size is proportional to $a$, hence giving a time-frequency description of $f$. Therefore, wavelets can be used to "zoom-in" on very short lived high frequency phenomena such as singularities.

Theorem 5.1.1: [Inversion formula]

If $f \in L^2(\mathbb{R})$ and $\psi$ is an analyzing wavelet then,

$$f(x) = C_\psi^{-1} \int_{-\infty}^{\infty} \mathcal{W}_\psi[f](a, b) \psi_{a,b}(x) \frac{da \, db}{a^2}$$

(III)
It is easy to see that the continuous wavelet transform is a continuous transformation from $L^2(\mathbb{R})$ into $L^2(\mathbb{R}^2, \frac{da \, db}{a^2})$ [7], preserving the inner product up to a constant. If $\psi$ is normalized so that $C_\psi = 1$ the transformation becomes an isometry.

It is important to note that, in most practical applications, the mother wavelet is assumed to vanish at $\pm \infty$; therefore, the domain of the continuous wavelet transform is extended to a larger class of functions other than $L^2(\mathbb{R})$. For future reference we will introduce the family of Meyer wavelets.

5.2 Meyer Wavelets

A Meyer wavelet is a frequency band-limited function whose Fourier transform is smooth. This wavelet is defined by combining positive and negative frequencies and a few phase factors. Namely,

$$\hat{\psi}(\xi) = \begin{cases} 
(2\pi)^{-\frac{3}{2}} e^{\frac{i}{2}\xi} \sin\left(\frac{\pi}{2} \nu\left(\frac{1}{2\pi|\xi|} - 1\right)\right) & \text{for } \frac{2\pi}{3} \leq |\xi| \leq \frac{4\pi}{3} \\
(2\pi)^{-\frac{3}{2}} e^{\frac{i}{2}\xi} \cos\left(\frac{\pi}{2} \nu\left(\frac{1}{2\pi|\xi|} - 1\right)\right) & \text{for } \frac{4\pi}{3} \leq |\xi| \leq \frac{8\pi}{3} \\
0 & \text{otherwise}
\end{cases}$$

where $\nu$ is a $C^p$ or $C^\infty$ function satisfying

$$\nu(x) = \begin{cases} 
0 & \text{if } x \leq 0 \\
1 & \text{if } x \geq 1
\end{cases}$$
and

\[ \nu(x) + \nu(1-x) = 1 \]

The regularity of \( \hat{\varphi} \) is the same as that of \( \nu \).

Figures 5.1 and 5.2 show the graphs of \( |\hat{\varphi}| \) and \( \varphi \), respectively, for the choice \( \nu(x) = x^4(35 - 84x + 70x^2 - 20x^3) \) on \([0, 1]\).

Figure 5.1 - Modulus of the Meyer wavelet associated to the auxiliary function \( \nu \) shown above

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Remark: If \( v_p \in C^p(\mathbb{R}) \) is chosen so that \( v_p \) restricted to \([0,1]\) is a polynomial then it can be shown that the polynomial must be of odd degree, namely \( 2p+1 \), and for unique coefficients \( \{ b_0, b_1, \ldots, b_p \} \) we have that

\[
v_p(x) = x^{p+1} \sum_{k=0}^{p} b_k x^k.
\]

Some examples [10] are:

\[
\begin{align*}
v_0(x) &= x, \\
v_1(x) &= x^2(3 - 2x), \\
v_2(x) &= x^3(10 - 15x + 6x^3), \\
v_3(x) &= x^4(35 - 84x + 70x^2 - 20x^3),
\end{align*}
\]

for \( 0 \leq x \leq 1 \).
5.3 The Discrete Wavelet Transform

The next step is to discretize the wavelet transform so that the resulting set constitutes a frame. Therefore, a function \( f \in L^2(\mathbb{R}) \) will be completely determined by its discrete wavelet coefficients.

For convenience, the scale parameter \( a \) is considered to be positive. The discretization of the dilation parameter is given by \( a = a_0^j \) where \( j \in \mathbb{Z} \) and \( a_0 > 1 \) is the fixed dilation step. Moreover, the discretization of the location parameter is accomplished by uniformly sampling \( b \) at intervals proportional to the scale \( a_0^j \) as high frequency wavelets need to be translated by small steps in order to cover the whole time range, while lower frequency wavelets need to be translated by larger steps.

Then the discrete collection of wavelets is given by

\[
\psi_{j,k}(x) = a_0^{-j/2} \psi(a_0^{-j} x - kb_0), \quad j, k \in \mathbb{Z}
\]

Suitable choices of \( \psi, a_0 \) and \( b_0 \) [7] will lead to frames of wavelets. Therefore in order to reconstruct \( f \) from its wavelets coefficients, we need to obtain the dual frame \( \{ \tilde{\psi}_{j,k} \} \) [see Chapter 4]. According to the definition, \( \{ \tilde{\psi}_{j,k} \}_{j,k} = \left( (F^*F)^{-1} \right) \psi_{j,k} \) where \( F^*F \) is the frame operator associated to \( \{ \psi_{j,k} \} \).
Although an infinite number of $\psi_{j,k}$ needs to be calculated, the computation can be reduced. In fact, let $D^n$ and $T^n$ be the operators defined by

$$D^n f(x) = a_0^{-\frac{n!}{2}} f(a_0^{-m} x) \quad \text{and} \quad T^n f(x) = f(x + n b_0),$$

respectively. Since for every $f$ we have that

$$\left( F^* F \right) f = \sum_{j,k} \langle f, \psi_{j,k} \rangle \psi_{j,k},$$

it follows that

$$\left( F^* F \right) [D^n f] = \sum_{j,k} \langle D^n f, \psi_{j,k} \rangle \psi_{j,k} \quad \text{(IV)}$$

where

$$\langle D^n f, \psi_{j,k} \rangle = \left[ \int (D^n f)(t) \psi_{j,k}(t) \, dt = \int a_0^{-\frac{n!}{2}} f(a_0^{-m} t) \psi_{j,k}(t) \, dt = \int f(u) a_0^{-\frac{n!}{2}} \psi_{j,k}(a_0^{-m} u) \, du \right.$$

$$= \int f(u) a_0^{-\frac{n!}{2}} a_0^{-\frac{j!}{2}} \psi(a_0^{-m} u - k b_0) \, du = \int f(u) a_0^{-\frac{(j-m)!}{2}} \psi(a_0^{-m} u - k b_0) \, du$$

$$= \int f(u) \psi_{(j-m),k}(u) \, du = \langle f, \psi_{(j-m),k} \rangle$$

so that (IV) becomes

$$\left( F^* F \right) [D^n f] = \sum_{j,k} \langle f, \psi_{(j-m),k} \rangle \psi_{j,k}. \quad \text{(V)}$$

On the other hand, we have that

$$D^n \left[ \left( F^* F \right) f \right] = \sum_{j,k} \langle f, \psi_{j,k} \rangle D^n \psi_{j,k} = \sum_{j,k} \langle f, \psi_{(j-m),k} \rangle D^n \psi_{(j-m),k}. \quad \text{(VI)}$$

But

$$D^n \psi_{(j-m),k}(x) = a_0^{-\frac{n!}{2}} \psi_{(j-m),k}(a_0^{-m} x) = a_0^{-\frac{n!}{2}} a_0^{-\frac{(j-m)!}{2}} \psi(a_0^{-m} a_0^{-j-m} x - k b_0)$$

$$= a_0^{-\frac{j!}{2}} \psi(a_0^{-j} x - k b_0) = \psi_{j,k}(x)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
so that from (V) and (VI) it follows that

\[ D^m [(F^*F)f] \sum_{j,k} (f, \psi_{(j-m,k)}) D^m \psi_{(j-m,k)} = \sum_{j,k} (f, \psi_{(j-m,k)}) \psi_{j,k} = (F^*F)[D^m f]. \]

Therefore, since \((F^*F)\) commutes with the dilation operator \(D^m\), so does \((F^*F)^{-1}\). In particular, since

\[ \psi_{j,k}(x) = a^{-j/2}_0 \psi(a^{-k} - kb_0) = D'T^k \psi(x), \]

we have that

\[ \tilde{\psi}_{j,k} = \left[(F^*F)^{-1}\right][D'T^k \psi] = D'T^k (F^*F)^{-1} \psi, \]

so that

\[ \tilde{\psi}_{j,k}(x) = a^{-j/2}_0 \tilde{\psi}_{0,k}(a^{-k} x). \]

That is, computing the dual frame \(\{\tilde{\psi}_{j,k}\}\) comes down to computing the family \(\{\tilde{\psi}_{0,k}\}\).

*Note:* Since it can be verified that the translation operator \(T^m\) does not commute with the frame operator, the computation of the dual frame (in the general case) cannot be simplified any further.

*Remark:* In practice the scaling step is chosen to be \(a_0 = 2\), so we go from one scale to the next by doubling or halving the translation step; and \(b_0 = 1\) so that \(\psi_{j,k}(x) = 2^{-j/2} \psi(2^{-j} x - k), j, k \in \mathbb{Z}\).  

Let \( \Gamma = \{ \gamma = (a, u, b) : a, b \in \mathbb{R}, a > 0, u \in S^{d-1} \} \) be a parameter space, where \( a > 0 \) denotes a scaling factor, \( u \in S^{d-1} \) an orientation and \( b \in \mathbb{R} \) a location. Given a function \( \psi : \mathbb{R} \to \mathbb{R} \) we write \( \psi_{\gamma}(x) = a^{-\frac{d}{2}} \psi\left(\frac{a^{d}x + b}{a}\right) \) for each \( \gamma = (a, u, b) \in \Gamma \) and consider the measure \( \mu(dy) = \frac{da}{a^{d+1}} \sigma_d du db \), where \( \sigma_d \) is the surface area of the unit sphere \( S^{d-1} \) in dimension \( d \) and \( du \) is the uniform probability measure on \( S^{d-1} \). We will always assume that \( \psi : \mathbb{R} \to \mathbb{R} \) belongs to the Schwarz space \( S(\mathbb{R}) \). The continuous Ridgelet transform is defined as follows [4].

Definition:

A function \( \psi : \mathbb{R} \to \mathbb{R} \) is called an admissible function if it satisfies the condition

\[
K_\psi = \int \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi < \infty \quad (I)
\]

A "ridge" function \( \psi_\gamma \) generated by an admissible \( \psi \) is called a Ridgelet

[Figure 6.1].
Note that if $\psi$ is concentrated in the interval $[-R,R]$, the ridgelet
$\psi_r(x) = a^{-\frac{1}{2}} \psi \left( \frac{ux - b}{a} \right)$ is supported in the strip \( \{ x \in \mathbb{R}^d : |u \cdot x - b| \leq Ra \} \).

So now we can define the *Ridgelet Transform*.

**Definition:**

Given $f \in L^1(\mathbb{R}^d)$ the *Ridgelet Transform* of $f$ at $\gamma \in \Gamma$ is defined as

$$\mathcal{R}(f)(\gamma) = \left( f, \psi_\gamma \right)$$

**Remark:** It is not hard to show that, for $\psi \in \mathcal{S}(\mathbb{R})$, the admissibility condition (I) is equivalent to the requirement of vanishing moments:

$$\int t^k \psi(t) \, dt = 0 \quad k \in \{0, 1, \ldots, \left[ \frac{d+1}{2} \right] - 1 \}$$

[see Appendix C1]

Note that the number of vanishing moments grows linearly with the dimension $d$.

---

**Figure 6.1 - A typical ridgelet function.**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Theorem 6.1: [Inversion formula]

Suppose that \( f \) and \( \hat{f} \) are in \( L^1(\mathbb{R}^d) \). If \( \psi \) is admissible then

\[
f = c_\psi \int \langle f, \psi_x \rangle \psi_x \, \mu(d\gamma),
\]

(II)

where \( c_\psi = \pi (2\pi)^{-d} K_\psi^{-1} \).

Proof:

Let \( R_u f \) be the Radon transform of \( f \) in the direction \( u \in S^{d-1} \), i.e.,

\[
R_u f(t) = \int_{x \cdot u = t} f(x) \, dS_x, \quad u \in S^{d-1}, \quad t \in \mathbb{R}.
\]

With a slight change in notation we now let \( \psi_u(x) = a^{-\frac{d}{2}} \psi(\gamma_u) \) and
\( \tilde{\psi}(x) = \psi(-x) \). We also let \( \omega_{a,u}(b) = \left[ \tilde{\psi} \ast R_u f \right](b) \) and

\[
I = \int \langle f, \psi_x \rangle \psi_x(x) \, \mu(d\gamma).
\]

By using Fubini's theorem, we have

\[
\int_{\mathbb{R}^d} f(x) \psi_x(x) \, dx = \int_{\mathbb{R}^d} f(x) \psi_x(u \cdot x - b) \, dx
\]
\[
= \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} f(x) \psi_x(s) \delta(s - (u \cdot x - b)) \, ds \, dx
\]
\[
= \int_{\mathbb{R}^d} \left[ \int_{\mathbb{R}^d} f(x) \delta((s + b) - u \cdot x) \, dx \right] \psi_x(s) \, ds
\]
\[
= \int_{\mathbb{R}} \int_{\mathbb{R}^d} f(x) \delta((s + b) - u \cdot x) \, dx \psi_x(s) \, ds
\]
\[
= \int_{\mathbb{R}} R_u f(s + b) \psi_x(s) \, ds = \left[ \tilde{\psi} \ast R_u f \right](b) = \omega_{a,u}(b),
\]

so that,
\[ I = \int \psi_o (u \cdot x - b) \omega_o (b) \sigma_{d+1} du \: db. \]

Since \( \hat{f} \in L^1(\mathbb{R}^d) \) and \( R_{x} f = \hat{f}(\xi u) \), [see Appendix C2], it follows that 
\[ R_{x} f \in L^1(\mathbb{R}^d). \]

Therefore,
\[ I = \int \psi_o * [\hat{\psi}_o * R_{x} f](u \cdot x) \sigma_{d+1} du. \]

Now, since \( \psi_o * [\hat{\psi}_o * R_{x} f] \in L^1(\mathbb{R}) \) and its one-dimensional Fourier transform is given by
\[ a |\hat{\psi}(a \xi)|^2 \hat{f}(\xi u), \] [see Appendix C3],
the inverse Fourier transform yields
\[ I = \frac{1}{2\pi} \int e^{i\xi x} \hat{f}(\xi u) a \left| \hat{\psi}(a \xi) \right|^2 \frac{d \xi}{\sigma_{d+1}} du \: d\xi. \]

But \( \psi \) is real valued, so that \( \hat{\psi}(-\xi) = \hat{\psi}(\xi) \) and
\[ I = \frac{1}{\pi} \int e^{i\xi x} \hat{f}(\xi u) a \left| \hat{\psi}(a \xi) \right|^2 1_{\{\xi > 0\}} \frac{d \xi}{\sigma_{d+1}} du \: d\xi. \]

Thus, by Fubini and (I) one obtains
\[ I = \frac{1}{\pi} \int e^{i\xi x} \hat{f}(\xi u) \left\{ \int |\hat{\psi}(a \xi)|^2 \frac{d \xi}{\sigma_{d+1}} \right\} 1_{\{\xi > 0\}} \sigma_d du \: d\xi \]
\[ = \frac{1}{\pi} \int e^{i\xi x} \hat{f}(\xi u) K_{\psi} |\xi|^{d+1} 1_{\{\xi > 0\}} \sigma_d du \: d\xi \]
\[ = \frac{1}{\pi} K_{\psi} \int_{\mathbb{R}^d} e^{i\xi x} \hat{f}(k) \: dk = \frac{1}{\pi} K_{\psi} (2\pi)^d f(x), \]
for all \( x \in \mathbb{R}^d \). Hence,
\[ f = \left[ \pi (2\pi)^d K_{\psi}^{-1} \right] \left\{ \langle f, \psi \rangle \psi, \mu(d\gamma) \right\} = c_{\psi} \int \langle f, \psi \rangle \psi, \mu(d\gamma), \]
as we wanted to show.

Remark: From the above proof we can see that Ridgelet analysis becomes a one dimensional wavelet analysis in the Radon domain. This characteristic makes Ridgelets well-suited for dealing with functions smooth away from linear singularities.

Theorem 6.2: [Parseval relation]

Assume that \( f \in L^1 \cap L^2(\mathbb{R}^d) \) and \( \psi \) admissible. Then,

\[
\|f\|_2^2 = c_\psi \int \langle f, \psi_\gamma \rangle^2 \mu(d\gamma)
\]

where \( c_\psi = \pi (2\pi)^{-d} K_\psi^{-1} \).

Proof:

As in the proof of Theorem 6.1 let \( \omega_{a,u}(b) = [\tilde{\psi} * R_a f](b) \). Then one obtains that

\[
J = \int \langle f, \psi_\gamma \rangle^2 \mu(d\gamma) = \int \|\omega_{a,u}(b)\|^2 \mu(d\gamma) = \int \|\omega_{a,u}(b)\|^2 \frac{da}{d^{d+1}} \sigma_d du db
\]

and by Fubini's theorem it follows that

\[
\int \|\omega_{a,u}(b)\|^2 \frac{da}{d^{d+1}} \sigma_d du db = \int \|\omega_{a,u}\|_2^2 \frac{da}{d^{d+1}} \sigma_d du . \tag{III}
\]

Since \( \omega_{a,u} \) is the convolution between two integrable functions, it is itself integrable. Moreover, \( \omega_{a,u} \in L^2(\mathbb{R}) \) with \( \|\omega_{a,u}\|_2 \leq \|f\|_1 \|\psi\|_2 \) [see Appendix]
Therefore, its Fourier transform is well defined and
\[ \hat{\omega}_{a,x}(\xi) = \overline{\psi(a\xi)} \hat{f}(\xi u). \]

Now, applying Plancherel's theorem along with Fubini's and the admissibility condition (I) we have that
\[
J = \left( \int \left( \int \|a_{a,x}(b)\|_2^2 \, db \right) \frac{\sigma_d}{a_{a,x}^d} \right) \int \left( \int \|\hat{\omega}_{a,x}(\xi)\|^2 \frac{\sigma_d}{a_{a,x}^d} \sigma_d \, du \, d\xi \right) \\
= \frac{1}{2\pi} \int \left( \int \left| \hat{\psi}_a(\xi) \right|^2 \left| \hat{f}(\xi u) \right|^2 \frac{\sigma_d}{a_{a,x}^d} \sigma_d \, du \, d\xi \right) \\
= \frac{2}{2\pi} \int \left( \int \left| \hat{\psi}_a(\xi) \right|^2 \left| \hat{f}(\xi u) \right|^2 \frac{\sigma_d}{a_{a,x}^d} \sigma_d \, du \, d\xi \right) \\
= \frac{1}{\pi} K_{\nu} \int_{\{\xi \in \mathbb{R}^d : \xi \cdot u = 0\}} \left| \hat{f}(\xi u) \right|^2 \xi \cdot \xi^{-1} \, du \, d\xi
\]

We have shown that
\[
J = \frac{1}{\pi} K_{\nu} (2\pi)^d \|f\|_2^2,
\]
so that
\[
\|f\|_2^2 = \pi K_{\nu}^{-1} (2\pi)^{-d} \int \langle f, \psi \rangle^2 \mu(d\gamma)
\]

Remark: By normalizing \( \psi \) so that \( c_\nu = 1 \), the above result shows that there is a linear transformation \( \mathcal{R} : L^2(\mathbb{R}^d) \to L^2(\Gamma, \mu(d\gamma)) \), the Ridgelet Transform, which is an \( L^2 \)-isometry and whose restriction to \( L^1 \cap L^2(\mathbb{R}^d) \) satisfies:
\[
\mathcal{R}(f)(\gamma) = \langle f, \psi_\gamma \rangle.
\]
Proposition 6.3: [Generalized Parseval relation]

Let \( f, g \in L^2(\mathbb{R}^d) \). Then

\[
\langle f, g \rangle = c_\nu \int \mathcal{R}(f)(\gamma) \mathcal{R}(g)(\gamma) \, \mu(\gamma) . \tag{V}
\]

Proof:

It is sufficient to prove the property for \( L^1 \cap L^2(\mathbb{R}^d) \), for it is a dense subspace of \( L^2(\mathbb{R}^d) \).

Let \( f, g \in L^1 \cap L^2(\mathbb{R}^d) \). By using Fubini's theorem, (I) and Plancherel we have that

\[
\begin{align*}
\int \mathcal{R}(f)(\gamma) \mathcal{R}(g)(\gamma) \, \mu(\gamma) &= \int \langle f, \psi_\gamma \rangle \langle g, \psi_\gamma \rangle \, \mu(\gamma) \\
&= \int \left[ \hat{\psi}_{a \nu} \ast R_{a \nu} f \right] (b) \left[ \hat{\psi}_{\nu} \ast R_{\nu} g \right] (b) \frac{da \, d\nu}{a^{d-1} \sigma_d} \, db \\
&= \frac{1}{2\pi} \int \langle \hat{\psi}_{a \nu} \ast R_{a \nu} f, \hat{\psi}_{a \nu} \ast R_{a \nu} g \rangle \frac{da \, d\nu}{a^{d-1} \sigma_d} \, du \\
&= \frac{1}{2\pi} \int \hat{f}(\xi u) \hat{g}(\xi u) \left| a \psi(a\xi) \right|^2 \frac{da \, d\nu}{a^{d-1} \sigma_d} \, du \, d\xi \\
&= \frac{1}{\pi} K_\nu \int \hat{f}(\xi u) \hat{g}(\xi u) \xi^{d-1} \, d\xi \, du = c_\nu \langle f, g \rangle.
\end{align*}
\]

To delve deeper into the significance of the inversion formula, let us now consider some important results on the convergence of truncated ridgelet expansions. To that effect let \( \varepsilon > 0 \) and consider

\[
\Gamma_\varepsilon = \left\{ \gamma = (a, u, b) : \varepsilon \leq a \leq \varepsilon^{-1}, u \in \mathbb{S}^{d-1}, b \in \mathbb{R} \right\} \subset \Gamma,
\]

then we have the following result.
Proposition 6.4:

Let $f \in L^1(\mathbb{R}^d)$ and $\{\alpha_r\} = \{\langle f, \psi_r \rangle\}_{r \in \Gamma}$. Then, for all $\varepsilon > 0$,

$$\alpha_{\varepsilon} \in L^1(\Gamma, \mu(d\gamma)).$$

Proof:

Given that $\alpha_r = \langle f, \psi_r \rangle = \left[ \int \omega_{a,u}(b) \frac{da}{a^{d+1}} \sigma_d \, du \right]$, it follows by Fubini that

$$\int_{\Gamma} \alpha_r \mu(d\gamma) = \int_{\Gamma} \omega_{a,u}(b) \frac{da}{a^{d+1}} \sigma_d \, du \, db = \int_{\Gamma} \omega_{a,u} \frac{da}{a^{d+1}} \sigma_d \, du \quad (VI)$$

but $\|\omega_{a,u}\| \leq \|\psi\| \|f\| = a^{\frac{d}{2}} \|\psi\| \|f\|$, [see Appendix C5] so that (VI) yields

$$\int_{\Gamma} \alpha_r \mu(d\gamma) \leq \sigma_d \|f\| \|\psi\| \|f\| \int_{\Gamma} \frac{da}{a^{d+1}} \leq \sigma_d \|f\| \|\psi\| \int_{\Gamma} \frac{da}{a^{d+1}} < \infty$$

From the proposition above it follows that for any $f \in L^1(\mathbb{R}^d)$, the definition

$$f_\varepsilon := c_{\psi} \int_{\Gamma} \langle f, \psi_r \rangle \psi_r \mu(d\gamma)$$

makes sense due to the fact that $\{\psi_r\}_{r \in \Gamma}$ is uniformly $L^\infty$ bounded on $\Gamma_r$.

Theorem 6.5:

Assume $f, g \in L^1 \cap L^2(\mathbb{R}^d)$ and $\psi$ is admissible. Then

1. $f_\varepsilon \in L^2(\mathbb{R}^d)$ and
2. $\|f - f_\varepsilon\| \to 0$ as $\varepsilon \to 0$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Proof:

Let \( \phi_\lambda(x) = (2\pi \lambda)^{-\frac{d}{2}} e^{-\text{d}i\lambda^T x} \) and define \( f^\lambda_\varepsilon \) as follows:

\[
f^\lambda_\varepsilon = c_\varepsilon \int_{\mathbb{R}^d} \left( f \ast \phi_\lambda, \psi_\gamma \right) \psi_\gamma \mu(d\gamma)
\] (VII)

Let us show that \( f^\lambda_\varepsilon \in L^2(\mathbb{R}^d) \).

From the properties of the Radon Transform it is clear that

\[
R_u(f \ast \phi_\lambda) = R_u f \ast R_u \phi_\lambda \quad \text{and} \quad R_u \phi_\lambda(t) = (2\pi \lambda)^{-\frac{d}{2}} e^{-\frac{\text{d}i\lambda^T}{\lambda}}
\] [see Appendix C6]. On the other hand, from the properties of the Fourier transform and the results obtained in Appendix C2 we have that

\[
\mathcal{F}\left[R_u f \ast R_u \phi_\lambda \right](\xi) = \left[R_u f \ast R_u \phi_\lambda \right](\xi) = \hat{f}(\xi u) e^{-\frac{i}{\lambda^T}}.
\]

Using the same argument used for the proof of Theorem 6.1, we get

\[
f^\lambda_\varepsilon(x) = c_\varepsilon \int_{\mathbb{R}^d} \left[ \tilde{\psi}_\lambda \ast R_u f \ast R_u \phi_\lambda \right](b) \frac{du}{a^d \sigma_d} \mu du db
\]

\[
= c_\varepsilon \int_{\mathbb{R}^d} \left[ \tilde{\psi}_\lambda \ast R_u f \ast R_u \phi_\lambda \right](u \cdot x) \frac{du}{a^d \sigma_d} \mu du.
\]

Now by using the one-dimensional Inverse Fourier transform, the results of Appendices [C2] and [C6] and Fubini's theorem we obtain

\[
f^\lambda_\varepsilon(x) = \frac{c_\varepsilon}{2\pi} \int_{\mathbb{R}^d} e^{i (u \cdot x)} \hat{\psi}_\lambda(\xi) \hat{\phi}_\lambda(\xi) \hat{f}(\xi) R_u^\lambda(\xi) \frac{du}{a^d \sigma_d} \mu d\xi
\]

\[
= \frac{c_\varepsilon}{2\pi} \int_{\mathbb{R}^d} e^{i (u \cdot x)} |\hat{\psi}_\lambda(\xi)|^2 \hat{f}(\xi u) e^{-\frac{i}{\lambda^T}} \frac{du}{a^d \sigma_d} \mu d\xi
\]

i.e.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[ f_\epsilon^t(x) = \frac{c_\psi}{\pi} \int_{\mathbb{R}^d \times \mathbb{R}^+} e^{i \langle u, x - \xi \rangle} |\hat{\psi}(\xi)|^2 \hat{f}(\xi u) \frac{du}{u} \sigma_\epsilon \, du \, d\xi \]

Next if \( \xi \neq 0 \),

\[
\int_{\mathbb{R}^d} |\hat{\psi}(a\xi)|^2 \frac{da}{a^d} = |\xi|^{d-1} \int_{\mathbb{R}^d} |\hat{\psi}(t)|^2 \frac{dt}{t} = |\xi|^{d-1} K_\nu c_\epsilon(|\xi|)
\]

where \( c_\epsilon(|\xi|) = K_\nu^{-1} \int_{|\xi|}^{|\xi|} |\hat{\psi}(t)|^2 \frac{dt}{t^d} \), then it is clear that \( c_\epsilon(|\xi|) \to \frac{1}{\epsilon} \) as \( \epsilon \to 0 \). Now, letting \( k = |\xi|\mu \) we obtain,

\[ f_\epsilon^t(x) = \frac{c_\psi}{\pi} \int_{|k| \leq |\xi|} e^{i \langle k, x \rangle} K_\nu c_\epsilon(k) \hat{f}(k) \, dk = \frac{1}{\pi} c_\psi K_\nu \int_{|k| \leq |\xi|} e^{i \langle k, x \rangle} c_\epsilon(k) \hat{f}(k) \, dk
\]

Hence, \( f_\epsilon^t \) is the inverse Fourier transform of an \( L^2 \) element and then \( f_\epsilon^t \in L^2(\mathbb{R}^d) \).

Now let us prove that \( f_\epsilon^t \to f_\epsilon \) pointwise and in \( L^1(\mathbb{R}^d) \). Let \( x \in \mathbb{R}^d \). Then

\[
|f_\epsilon^t(x) - f_\epsilon(x)| = c_\psi \left| \int_{\mathbb{R}^d} \langle f \ast \phi_\epsilon - f, \psi_\gamma \rangle \psi_\gamma \, d\gamma \right|
\]

so that

\[
|f_\epsilon^t(x) - f_\epsilon(x)| \leq c_\psi \int_{\mathbb{R}^d} \left| \langle f \ast \phi_\epsilon - f, \psi_\gamma \rangle \right| \psi_\gamma \, d\gamma
\]

\[
\leq c_\psi \sup_{\gamma \in \mathbb{R}^d} \int_{\mathbb{R}^d} \left| \langle \psi_\gamma \ast \left( R_\epsilon f \ast R_\epsilon \phi_\epsilon - R_\epsilon f \right) \rangle (b) \right| \mu(d\gamma)
\]

and, by Fubini’s theorem and the fact that \( |\psi_\gamma(x)| = a^{-\frac{1}{2}} \| \psi \|_a \leq a^{-\frac{1}{2}} \| \psi \|_a \),

it follows that
\[
\left| f_{\varepsilon}^\lambda(x) - f_{\varepsilon}(x) \right| \leq c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \left\{ \left| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right| (b) \right\|_1 \mu(d\gamma)
\]

\[
= c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\{ \left| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right| (b) \right\|_1 \frac{ds}{\sigma_d} \sigma_d du
\]

\[
= c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\{ \left| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right|_1 \frac{ds}{\sigma_d} \sigma_d du
\]

\[
\leq c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right\|_1 \frac{ds}{\sigma_d} \sigma_d du,
\]

that is,

\[
\left| f_{\varepsilon}^\lambda(x) - f_{\varepsilon}(x) \right| \leq c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right\|_1 \frac{ds}{\sigma_d} \sigma_d du
\]

\[
= c_\psi e^{-\lambda \varepsilon} \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right\|_1 \frac{ds}{\sigma_d} \sigma_d du \quad \text{(VIII)}
\]

\[
= \delta(\varepsilon) \left\| \varphi \right\|_1 \int_{\varepsilon}^1 \int_{s^{d+1}} \left\| \mathcal{W}_\lambda \left( R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right) \right\|_1 \frac{ds}{\sigma_d} \sigma_d du.
\]

Since for each \( u \) fixed the set \( \{ R_{\lambda} \phi_{\lambda} \} \) is an approximate identity in \( L^1(\mathbb{R}) \), [see Appendix C7] it follows that \( \left\| R_{\lambda} f \ast R_{\lambda} \phi_{\lambda} - R_{\lambda} f \right\|_1 \rightarrow 0 \) as \( \lambda \rightarrow 0 \). Moreover, from Appendix C4 and (II) we have that

\[
\left\| R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right\|_1 \leq \left\| R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} \right\|_1 + \left\| R_{\varepsilon} f \right\|_1 \leq 2 \left\| R_{\varepsilon} f \right\|_1 \leq 2 \left\| f \right\|_1,
\]

so that the dominated convergence theorem yields

\[
\int_{s^{d+1}} \left\| R_{\varepsilon} f \ast R_{\lambda} \phi_{\lambda} - R_{\varepsilon} f \right\|_1 \frac{ds}{\sigma_d} \sigma_d du \rightarrow 0 \quad \text{as} \quad \lambda \rightarrow 0.
\]

Now from (VIII) we obtain

\[
\left\| f_{\varepsilon}^\lambda - f_{\varepsilon} \right\|_1 \rightarrow 0 \quad \text{as} \quad \lambda \rightarrow 0. \quad \text{(IX)}
\]
On the other hand, since
\[
\left| e^{-\frac{1}{2}k^2}\kappa(k) \right| \leq \kappa(k) |\widehat{f}(k)| \text{ for all } \lambda
\]
and,
\[
e^{-\frac{1}{2}k^2}\kappa(k) \widehat{f}(k) \to \kappa(k) \widehat{f}(k) \text{ pointwise as } \lambda \to 0,
\]
by the dominated convergence theorem one obtains that
\[
e^{-\frac{1}{2}k^2}\kappa(k) \widehat{f}(k) \to \kappa(k) \widehat{f}(k) \text{ in } L^2(\mathbb{R}^d) \text{ as } \lambda \to 0.
\]
Therefore, given that the Fourier Transform is an isometry it follows that
\[
f_\varepsilon \to (2\pi)^{-d} \mathcal{F}[\kappa \widehat{f}] \text{ in } L^2(\mathbb{R}^d)
\]
so that from (IX) we get that \( f_\varepsilon = (2\pi)^{-d} \mathcal{F}[\kappa \widehat{f}] \). And, by completeness,
\( f_\varepsilon \in L^2(\mathbb{R}^d) \). Finally, since
\[
\left\| \widehat{f}_\varepsilon - \widehat{f} \right\|_2^2 = \int |\widehat{f}(k)|^2 \left(1 - \kappa(k) \right)^2 dk
\]
and \( 0 \leq \kappa \leq 1 \) is such that \( \kappa(\xi) \to 1 \) as \( \varepsilon \to 0 \), it follows that
\[
\left\| \widehat{f}_\varepsilon - \widehat{f} \right\|_2 \to 0 \text{ as } \varepsilon \to 0
\]
so that
\[
\left\| f_\varepsilon - f \right\|_2 \to 0 \text{ as } \varepsilon \to 0
\]
CHAPTER 7

THE DISCRETE RIDGELET TRANSFORM

From the previous chapter it is clear that the ridgelet coefficient $\mathcal{R}(f)(\gamma)$ is the one-dimensional wavelet coefficient of $R_{\gamma}f$ (the Radon Transform of $f$). Therefore the theory of ridgelets is closely related to the Radon transform theory as well as to the theory of rotation and scaling of functions. These are some of the reasons as to why there is no simple definition of the transformation in the discrete case. Nevertheless, the definition of the discrete Ridgelet transform [9] that we are about to present bears a strong analogy with the continuum case.

From this point, and given the nature of our application, we will be referring to the results of the two dimensional case. Thus, the input for the discrete ridgelet transform will be an $n$ by $n$ array $I(u,v)$ where $-\frac{n}{2} \leq u,v < \frac{n}{2}$.

Let $\psi_{j,k}(t)$ be the discrete Meyer wavelet where $-\frac{m}{2} \leq t < \frac{m}{2}$, $m=2n$, $0 \leq j < \log_2(m)$ and $0 \leq k < 2^j$. Following the definitions in Chapter 3 let $\Theta = \Theta_1 \cup \Theta_2$ where

$$\Theta_i = \{\theta_{l,n}^i : \theta_{l,n}^i = \arctan \left( \frac{2l}{n} \right), l \in \mathbb{Z}, -\frac{n}{2} \leq l < \frac{n}{2} \}$$
and

$$\Theta_2 = \left\{ \theta_{i,n}^2 : \theta_{i,n}^2 = \frac{\pi}{2} - \arctan\left(\frac{2l}{n}\right), l \in \mathbb{Z}, -\frac{n}{2} < l < \frac{n}{2} \right\}$$

Then, assuming \( n \) is given we define the digital ridgelets as follows.

**Definition:**

A digital ridgelet \( \rho_{j,k,l} \) is an \( n \) by \( n \) array defined as

$$\rho_{j,k,l}(u,v) = \psi_{j,k}(u + \tan(\theta_{i,n}^1),v)$$

and

$$\rho_{j,k,2l}(u,v) = \psi_{j,k}(v + \tan(\theta_{i,n}^2),u)$$

(1)

where \( 0 \leq j < \log_2(n), 0 \leq k < 2^j \) and \(-\frac{n}{2} < l < \frac{n}{2}\).

To simplify the notation let us denote \( \lambda = (j,k,s,l) \) and \( \Lambda \) the set of all those quads.

From (1) the relationship between the Discrete Ridgelet transform and the Discrete Slant Stack (Radon) transform becomes evident. As in the definition of the DSS, the choice of \( m = 2n \) is made to accomplish geometric faithfulness and avoid wrap-around artifacts.

We will now present the definition of the Discrete Ridgelet transform.

**Definition:**

The Discrete Ridgelet transform is the one dimensional Meyer wavelet transform in \( t \) of the Slant Stack transform, namely,

$$\mathcal{R}_1(j,k,s,l) = \langle SI(\cdot; s,l), \psi_{j,k}(\cdot) \rangle$$

(II)
where \(\langle \cdot, \cdot \rangle\) denotes the inner product in the \(t\)-variable.

That is, we go from the image domain to the Radon domain by means of the Discrete Slant Stack transform and there we apply the one dimensional Meyer wavelet transform to each column of \(SI(t; s, l)\). Hence, the map of Ridgelet coefficients looks as follows [Figure 7.1].

![Figure 7.1 - Map of ridgelet coefficients.](image)

We observe that the Discrete Ridgelet transform takes an input array of size \(n\) by \(n\) to produce an output of size \(2n\) by \(2n\).
CHAPTER 8

IMPLEMENTATION REMARKS AND ALGORITHM

In this chapter we will present an outline of the algorithm implementing the Discrete Ridgelet Transform (DRT). Furthermore, the actual source code implementation of the most important functions involved in the computation will also be provided.

As defined in Chapter 7 [(II)] given an \( n \) by \( n \) array \( I(u,v) \), the Discrete Ridgelet transform of \( I(u,v) \) is the one dimensional wavelet transform of the columns in \( SI(k;s,l) \) (its discrete Slant Stack transform).

The first step towards the computation of the DRT is the computation of the discrete Slant Stack of the image.

According to equation (IV) in Chapter 3

\[
(S_p I)(\cdot) = \begin{cases} 
F_i^{-1} \cdot P F_i I(\cdot, l) & \text{for } \theta \in \Theta_1 \\
F_i^{-1} \cdot P F_i I(\cdot, l) & \text{for } \theta \in \Theta_2
\end{cases}
\]

To unify the notation let us denote

\[
SI(\cdot ; s, l) = \begin{cases} 
F_i^{-1} \cdot P F_i I(\cdot, l) & \text{for } s = 1 \text{ so that } \tan(\theta) = 2l/n \\
F_i^{-1} \cdot P F_i I(\cdot, l) & \text{for } s = 2 \text{ so that } \cot(\theta) = 2l/n
\end{cases}
\]

where \(-\eta/2 \leq l < \eta/2\).

Thus, to compute the Slant Stack we need to compute the Pseudo-polar

63
Fourier transform. Underlying the computation of the Pseudo-polar Fourier transform lies a variation of the DFT, namely, the Fractional Fourier transform whose definition will now be given.

Definition: [(Unaliased) Fractional Fourier Transform] [3]

Given a vector \( Y = (Y_j), -\frac{n}{2} \leq j < \frac{n}{2} \) and \( \alpha \in \mathbb{R} \), the Fractional Fourier transform of \( Y \) is given by

\[
(F^\alpha Y)(\omega) = \sum_{j=-\frac{n}{2}}^{\frac{n}{2}-1} Y_j e^{\frac{2\pi i \alpha j \omega}{n}}, \quad \omega \in \mathbb{Z}, \quad -\frac{n}{2} \leq \omega < \frac{n}{2}
\]

This transform gives a version of the DFT based on fractional roots of unity \( e^{-2\pi i/\alpha} \) instead of the usual integral roots \( e^{-2\pi i/n} \).

We will now show the steps involved in the computation of the Pseudo-polar Fourier transform of \( I(u,v) \).

According to the definition in Chapter 3,

\[
P\mathcal{F}_I I(k,l) = \hat{I}\left(-\frac{2l}{n}, k\right)
\]

\[
P\mathcal{F}_I I(k,l) = \hat{I}\left(k, -\frac{2l}{n}\right)
\]

for \( -\frac{n}{2} \leq l < \frac{n}{2}, -n \leq k < n \) where \( \hat{I} \) is given by

\[
\hat{I}(\xi_1, \xi_2) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2}-1} \sum_{v=-\frac{n}{2}}^{\frac{n}{2}-1} I(u,v) e^{-\frac{2\pi i}{n}(\xi_1 u + \xi_2 v)}
\]

For convenience, let us consider \( P\mathcal{F}_I I(k,-l) = \hat{I}\left(\frac{2l}{n}, k\right) \). It is clear that we can obtain \( P\mathcal{F}_I I(k,l) \) from \( P\mathcal{F}_I I(k,-l) \) by just flipping the latter vector.
with respect to its center.

Then, for each fixed \( k \), we have that

\[
\mathcal{P} \mathcal{F}_l I(k,-l) = \hat{I} \left( \frac{2l}{n} k, k \right) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2} - 1} \sum_{v=-\frac{n}{2}}^{\frac{n}{2} - 1} I(u,v) e^{-2\pi i (\frac{2l}{n} u) e^{-2\pi i \frac{2l}{n} v} =}
\]

\[
= \sum_{u=-\frac{n}{2}}^{\frac{n}{2} - 1} \left( \sum_{v=-\frac{n}{2}}^{\frac{n}{2} - 1} I(u,v) e^{-2\pi i \frac{2l}{n} v} \right) e^{-2\pi i \frac{2l}{n} m}. \tag{I}
\]

Let

\[
c_i(u) = \sum_{v=-\frac{n}{2}}^{\frac{n}{2} - 1} I(u,v) e^{-2\pi i \frac{2l}{n} v} \tag{II}
\]

Clearly, (II) corresponds to symmetrically padding \( I(u,v) \) with zeros in the y-direction and then performing the one dimensional DFT to each column (where now each column has \( 2n \) elements). Now combining (I) and (II) we have that

\[
\mathcal{P} \mathcal{F}_l I(k,-l) = \hat{I} \left( \frac{2l}{n} k, k \right) = \sum_{u=-\frac{n}{2}}^{\frac{n}{2} - 1} c_i(u) e^{-2\pi i \frac{2l}{n} u} \tag{III}
\]

Since \( \{c_i(u)\}_u \) has length \( n \) and \(-\frac{n}{2} \leq l < \frac{n}{2}\) (III) entails symmetrically padding the vector \( \{c_i(u)\}_u \) to length \( m=2n \), then calculating its Fractional Fourier transform with factor \( \alpha = \frac{2l}{n} \) and finally returning only the \( n \) central elements of its transform.

Repeating this procedure for every \( k \), \(-n \leq k < n\), we obtain all the columns of \( \mathcal{P} \mathcal{F}_l I(k,-l) \), therefore, we obtain \( \mathcal{P} \mathcal{F}_l I(k,l) \).
It is now clear that calculating the Pseudopolar Fourier transform involves a series of applications of the conventional DFT along with the Fractional Fourier transform.

In order to obtain $P\mathcal{F}_j I(k,l)$ we start with $P\mathcal{F}_j I(k,-l)$ and repeat the steps above except for the fact that the padding of $I(u,v)$ in (II) has to be performed in the x-direction so that the 1-D DFT is applied to each row.

Finally, and according to the definition in Chapter 3, in order to obtain the Discrete Slant Stack of $I(u,v)$ we need to compute the 1-D DFT to each row of $P\mathcal{F}_j I(k,l), i = 1, 2$.

The last step towards the computation of the Discrete Ridgelet transform of $I(u,v)$ is to apply the one dimensional Meyer wavelet transform to each column of the $2n \times 2n$ matrix $(SI)(k,s,l)$ just obtained.

8.1 Source Code For The Algorithm

In this section we provide the most significant components of the source code (in MFC) that implements the algorithm just described. The program will be used later in Chapter 9 to compare the performance of both the Ridgelet and Wavelet transforms at representing images containing straight edges. A complete version of the program can be obtained from the authors.
The algorithm for calculating the Ridgelet transform consists of two computational stages. Namely,

1. The computation of the forward and inverse Slant Stack Radon transform is performed in MATLAB 6.5 (R 13) by means of the algorithm Fast Slant Stack which is part of the BeamLab 200 toolbox (2002).

   Note: Beamlab software is available at NO CHARGE. It is available for download at


2. Once the Slant Stack is computed by MATLAB, the output is dumped onto the disk so it can be used by the application “Ridgelets.exe” to compute the forward and inverse 1-D Meyer wavelet transform and, therefore, the Ridgelet transform.

   /*===================================================================*/
   /*  This codes generates the half Dome of equation */
   /*  */
   /*  f(x, y) = \begin{cases} e^{-(x^2+y^2)} & \text{if } x^2+y^2 \leq 1 \\ 0 & \text{otherwise} \end{cases} */
   /*  */
   /*===================================================================*/

   // GenerateFunc.cpp: implementation of the CGenerateFunc class.
   static char THIS_FILE[]="_FILE__";

   void CGenerateFunc::HalfDome(int w, int h)
   {
      if(w != m_width || h != m_height)
      {
         m_width = w;
         m_height = h;
         if(m_Data != NULL)
            delete [] m_Data;

         m_Data = new double[w*h];
      }
ZeroMemory(m_Data, sizeof(double) * w * h);
double n_x1, n_x2;
int x1, x2;
double f_s = -2.5, f_e = 2.5;
for(x2 = 0; x2 < h; x2++)
for(x1 = 0; x1 < w; x1++)
{
    n_x2 = f_s + (double(x2)/double(w))*(f_e-f_s);
    n_x1 = f_s + (double(x1)/double(w))*(f_e-f_s);
    if(n_x2 < (0.5*n_x1-.5))
        m_Data[x1+(h-x2-1)*w] = (255.0*exp(-n_x1*n_x1)-n_x2*n_x2));
}
/* ----------------------------------------------------------------------------------------------------------
void CMainFrame::OnDumpImageDataForRT()
/* Dump data for MATLAB to compute the Forward Slant Stack
{
    CRidgeletsDoc *pTmpDoc = (CRidgeletsDoc*) GetActiveDocument();
    switch(WriteToFile("c:\ImglN.dat", pTmpDoc->GetImgData(),
                     pTmpDoc->GetImgWidth()))
    {
        case -1:
            MessageBox("Error while writing to output file!");
            return;
            break;
        case -2:
            MessageBox("No file opened!");
            return;
            break;
        default:
            break;
    }

    //delete file...
    CFile my_DelFile;
    if( my_DelFile.Open("C:\ImgRT.dat", CFile::modeRead, NULL) )
    {
        my_DelFile.Close();
        my_DelFile.Remove("C:\ImgRT.dat");
    }
    CWnd * my_cwnd = NULL;
*/
```cpp
my_cwnd = FindWindow("SunAwtFrame", "MATLAB");
my_cwnd->SetActiveWindow();
my_cwnd->SetFocus();
my_cwnd->SetCapture();
my_cwnd->BringWindowToTop();
my_cwnd->EnableWindow(TRUE);

//Execute the MATLAB routines
my_cwnd->SendMessage(WM_KEYDOWN, 0x00000046, 0x00210001);
my_cwnd->SendMessage(WM_CHAR, 0x00000066, 0x00210001);
my_cwnd->SendMessage(WM_KEYUP, 0x00000046, 0xC0210001);

my_cwnd->SendMessage(WM_KEYDOWN, 0x00000052, 0x00130001);
my_cwnd->SendMessage(WM_CHAR, 0x00000072, 0x00130001);
my_cwnd->SendMessage(WM_KEYUP, 0x00000052, 0xC0130001);

my_cwnd->SendMessage(WM_KEYDOWN, 0x00000054, 0x00140001);
my_cwnd->SendMessage(WM_CHAR, 0x00000074, 0x00140001);
my_cwnd->SendMessage(WM_KEYUP, 0x00000054, 0xC0140001);

my_cwnd->SendMessage(WM_KEYDOWN, 0x0000000D, 0x00010001);
my_cwnd->SendMessage(WM_CHAR, 0x0000000D, 0x00010001);
my_cwnd->SendMessage(WM_KEYUP, 0x0000000D, 0x00010001);

for(;;)
{
    Sleep(1000);
    if(ReadFromFile("c:\lmgrt.dat") != -1)
        break;
}

pTmpDoc->InitImgData(m_pInputData, m_InputSize, m_InputSize);
ReleaseInputMemory();//Release memory allocated by ReadFromFile().

CRidgeletsView *pTmpView = (CRidgeletsView*) GetActiveView();
pTmpView->Invalidate();
pTmpView->UpdateWindow();
this->SetActiveWindow();
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
void CMainFrame::OnDumpImgDataForIRT()
/* Dump data for MATLAB to compute the Inverse Slant Stack */
{
    CRidgeletsDoc *pTmpDoc = (CRidgeletsDoc*) GetActiveDocument();
    switch(WriteToFile("c:\lmgRT.dat", pTmpDoc->GetImgData(),
                      pTmpDoc->GetImgWidth()))
    {
    case -1:
        MessageBox("Error while writing to output file!");
        return;
        break;
    case -2:
        MessageBox("No file opened!");
        return;
        break;
    default:
        break;
    }

    //delete file...
    CFile my_DelFile;
    if( my_DelFile.Open("C:\\ImgOUT.dat",CFile::modeRead, NULL) )
    {
        my_DelFile.Close();
        my_DelFile.Remove("C:\\ImgOUT.dat");
    }

    CWnd * my_cwnd = NULL;
    my_cwnd = FindWindow("SunAwtFrame", "MATLAB");
    my_cwnd->SetActiveWindow();
    my_cwnd->SetFocus();
    my_cwnd->SetCapture();
    my_cwnd->BringWindowToTop();
    my_cwnd->EnableWindow(TRUE);
    my_cwnd->SendMessage(WM_KEYDOWN, 0x00000049, 0x00170001);
    my_cwnd->SendMessage(WM_CHAR, 0x00000069, 0x00170001);
    my_cwnd->SendMessage(WM_KEYUP, 0x00000049, 0x00170001);
    my_cwnd->SendMessage(WM_KEYDOWN, 0x00000052, 0x00130001);
    my_cwnd->SendMessage(WM_CHAR, 0x00000072, 0x00130001);
    my_cwnd->SendMessage(WM_KEYUP, 0x00000052, 0x00130001);
    my_cwnd->SendMessage(WM_KEYDOWN, 0x00000054, 0x00140001);
    my_cwnd->SendMessage(WM_CHAR, 0x00000074, 0x00140001);
    my_cwnd->SendMessage(WM_KEYUP, 0x00000054, 0x00140001);

    //reproduced with permission of the copyright owner. further reproduction prohibited without permission.
my_cwnd->SendMessage(WM_KEYDOWN, 0x0000000D, 0x001C0001);
my_cwnd->SendMessage(WM_CHAR, 0x0000000D, 0x001C0001);
my_cwnd->SendMessage(WM_KEYUP, 0x0000000D, 0x001C0001);

for(;;)
{
    Sleep(1000);
    if(ReadFromFiles("c:\\ImgOUT.dat") != -1)
        break;
}
pTmpDoc->InitImgData(m_pInputData, m_InputSize, m_InputSize);

ReleaseInputMemory();

C RidgeletsView *pTmpView = (C RidgeletsView*) GetActiveView();
pTmpView->Invalidate();
pTmpView->UpdateWindow();
this->SetActiveWindow();

/* Write Data to file */
/*Write Data to file*/
int CMainFrame: :WriteToFile(CString fname, double *pData, int size)
/*
 * Writes the values pointed by pData to the file
 * specified by the name of "fname". The header information is as follows:
 * (int) size, (double) value1, value2, ..., value(size*size). (each row at a time!)
 */

Returns -1 if error occurred while opening file!
Returns -2 if wrong pointer to data to output.
Returns 0 if everything went ok.
 *
{
    FILE *pfile;
    pfile = fopen(fname, "wb");

    if(pfile == NULL)
        return -1;

    if(pData == NULL)
        return -2;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
fwrite( &size, sizeof(int), 1, pfile);

for(int y = 0; y<size; y++)
for(int x = 0; x<size; x++)
    fwrite( &(pData[x+y*size]), sizeof(double), 1, pfile);

fclose(pfile);

return 0;

/**
** Read Data from file
*/

int CMainFrame::ReadFromFile(CString fname)
{
    FILE *pfile;
    pfile = fopen(fname, "rb");

    if(pfile == NULL)
        return -1;

    Sleep(1000);
    fread(&m_InputSize, sizeof(int), 1, pfile);

    m_pInputData = new double[m_InputSize*m_InputSize];

    if(m_pInputData == NULL)
        return -2;

    int x, y;

    for(y = 0; y<m_InputSize; y++)
    for(x = 0; x<m_InputSize; x++)
        fread( &(m_pInputData[x+y*m_InputSize]), sizeof(double), 1, pfile);

    fclose(pfile);

    return 0;
}
Implementation of the “Modify Signal on the Wavelet domain”

NOTE: The rejection of coefficients is done using a two standard deviations threshold

modify_transformed_signal_dlg::modify_transformed_signal_dlg(CWnd* pParent /*=NULL*/, wavelet_analysis_class *wac, double *ptr_to_image, int width2, int height, int *pTmpTL) : CDialog(modify_transformed_signal_dlg::IDD, pParent)
{
    m_total_levels = pTmpTL;
    wac_ptr = wac;
    ptr_to_image_data = ptr_to_image;

    width = width2;
    height = height2;

    m_value = 0.0;
    m_transform_level = 1;
    m_sigma_multiplier = 2.0;
    m_number_of_MSC = 0;
}

void modify_transformed_signal_dlg::OnOK()
{
    if(ptr_to_image_data == NULL)
        CDialog::OnOK();

    UpdateData(true);

    // SELECTION = Selection of Most Significant Coefficients
    if(m_MSC_selection.GetCheck())
    {
        wac_ptr->WaveletMostSigCoef(ptr_to_image_data, width, height, m_number_of_MSC, (*m_total_levels));
    }

    CDialog::OnOK();
}
/* Calculation of the Wavelet transform on Radon Domain */
/* NOTE: 1-D WT on the columns of the Slant Stack matrix */

// RadonWaveletCoefSel.cpp : implementation file  
// RadonWaveletCoefSel dialog 
RadonWaveletCoefSel::RadonWaveletCoefSel(CWnd* pParent /*=NULL*/, 
wavelet_analysis_class *pWAC, double *pImgData, int width, int height, int TrsfLvl) : CDialog(RadonWaveletCoefSel::IDD, pParent) 
{  
    m_Sigma_multiple = 2.0;  
    m_wavelet_level_selection = 1;  
    m_wavelet_multiplier = 0.0;  
    m_NumberOfSigCoef = 0;  

    m_pImgData = pImgData;  
    m_ImgWidth = width;  
    m_ImgHeight = height;  
    m_pWAC = pWAC;  
    m_TrsfLvl = TrsfLvl;  
}

void RadonWaveletCoefSel::OnOK() 
{  
    UpdateData(TRUE);  

    if(m_SelectSigCoef_Radio_Button.GetCheck() == 1)  
        m_pWAC->WaveletMostSigCoef_RT(m_pImgData, m_ImgWidth, 
            m_ImgHeight, m_NumberOfSigCoef, m_TrsfLvl);  
    CDialog::OnOK();  
}

int wavelet_analysis_class::COMPUTE_OFFSET(int width, int height, int width_offset, 
    int height_offset, int width_limit, int height_limit)  
{  
    while(width_offset>width_limit-1)  
        width_offset -= width_limit;
while(width_offset < 0)
    width_offset += width_limit;
while(height_offset>height_limit-1)
    height_offset -= height_limit;
while(height_offset < 0)
    height_offset += height_limit;

    return width_offset+height_offset*width;
}

/*================================================================----------------*/

wavelet_analysis_class::wavelet_analysis_class()
{
    scaling_signal = wavelet_signal = NULL;
    scaling_function = wavelet_function = NULL;
}

/*================================================================----------------*/

void wavelet_analysis_class::SetNewCoefficients(double *scaling_function_in, double *
wavelet_function_in, int number_of_elements_in_function_in, int signal_shift_in)
{
    if(scaling_signal != NULL)
        delete [] scaling_signal;
    if(wavelet_signal != NULL)
        delete [] wavelet_signal;
    if(scaling_function != NULL)
        delete [] scaling_function;
    if(wavelet_function != NULL)
        delete [] wavelet_function;

    scaling_function = NULL;
    wavelet_function = NULL;

    scaling_function = new double[number_of_elements_in_function_in];
    wavelet_function = new double[number_of_elements_in_function_in];

    number_of_elements_in_function = number_of_elements_in_function_in;
    signal_shift = signal_shift_in;
for(int i = 0; i<number_of_elements_in_function; i++)
{
    scaling_function[i] = scaling_function_in[i];
    wavelet_function[i] = wavelet_function_in[i];
}

void wavelet_analysis_class::Forward_Wavelet_Transform(unsigned char
*image_data_in2, double *image_data_out, int image_width,
int image_height, int transform_level )
{
    if(wavelet_function == NULL || scaling_function == NULL)
        return;
    double *image_data_in = new double[image_width*image_height];
    for(int i = 0; i<image_height; i++)
        for(int j = 0; j<image_width; j++)
            image_data_in[i*image_width+j] =
                double(image_data_in2[i*image_width+j]);
    memset(image_data_out, 0, image_width*image_height*sizeof(double));
    int new_image_width;
    int new_image_height;
    int counter;
    new_image_width = 2*(image_width/2);
    new_image_height = 2*(image_height/2);
    for(int up_to_level = 1; up_to_level <= transform_level; up_to_level++)
    {
        for(int temp_h = 0; temp_h < new_image_height; temp_h++)
        {
            counter = 0;
            for(int temp_w = 0; temp_w < new_image_width-1; temp_w+=2)
            {
                image_data_out[COMPUTE_OFFSET(image_width,
                    image_height, counter, temp_h, image_width, image_height)] = 0;
                image_data_out[COMPUTE_OFFSET(image_width,
                    image_height, counter+new_image_width/2, temp_h,
                    image_width, image_height)] = 0;
            }
        }
    }
}
for(int filter_counter=0;
    filter_counter < number_of_elements_in_function;
    filter_counter++)
{
    image_data_out[COMPUTE_OFFSET(image_width, image_height, counter, temp_h, image_width, image_height)] +=
    image_data_in[COMPUTE_OFFSET(image_width, image_height, temp_w+signal_shift+filter_counter, temp_h, new_image_width, image_height)] *
    scaling_function[filter_counter];
    image_data_out[COMPUTE_OFFSET(image_width, image_height, counter+new_image_width/2, temp_h, image_width, image_height)] +=
    image_data_in[COMPUTE_OFFSET(image_width, image_height, temp_w+signal_shift+filter_counter, temp_h, new_image_width, image_height)] *
    wavelet_function[filter_counter];
}

memcpy(image_data_in, image_data_out, image_width*image_height*sizeof(double));

for(int temp_w = 0; temp_w < new_image_width; temp_w++)
{
    counter = 0;

    for(int temp_h = 0; temp_h < new_image_height-1; temp_h+=2)
    {
        image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, counter, image_width, image_height)] = 0;
        image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, (counter+new_image_height/2), image_width, image_height)] = 0;

        for(int filter_counter=0; filter_counter <
            number_of_elements_in_function; filter_counter++)
        {
            image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, counter, image_width, image_height)] +=
            image_data_in[COMPUTE_OFFSET(image_width, image_height, temp_w, new_image_width, image_height)] *
            image_data_in[COMPUTE_OFFSET(image_width, image_height, temp_w, (counter+new_image_height/2), image_width, image_height)] *
            wavelet_function[filter_counter];
        }
        counter++;}
}
(filter_counter+temp_h)+signal_shift, image_width,
new_image_height]) * scaling_function[filter_counter];

image_data_out[COMPUTE_OFFSET(image_width,
image_height, temp_w, (counter+new_image_height/2),
image_width, image_height)] +=
image_data_in[COMPUTE_OFFSET(image_width,
image_height, temp_w,
(filter_counter+temp_h)+signal_shift, image_width,
new_image_height)]) * wavelet_function[filter_counter];
}
counter++;
}
}

memcpy(image_data_in, image_data_out, image_width*image_height*sizeof(double));

new_image_width = image_width / (int)pow(2, up_to_level);
new_image_height = image_height / (int)pow(2, up_to_level);

new_image_width = 2*(new_image_width/2);
new_image_height = 2*(new_image_height/2);
}
delete [] image_data_in;
}

/*--------------------------------------------------------------------------------------------------------------------*/

void wavelet_analysis_class::Inverse_Wavelet_Transform(double *image_data_in,
unsigned char *image_data_out, int image_width,
int image_height, int transformed_level, int last_level )
{
    if(wavelet_function == NULL || scaling_function == NULL)
        return;
    double *image_data_out_temp = new double[image_width*image_height];
    memset(image_data_out, 0, image_width*image_height*sizeof(unsigned char));
    memset(image_data_out_temp, 0, image_width*image_height*sizeof(double));

    int new_image_width;
    int new_image_height;
    int counter = 0;
for(int up_to_level = transformed_level; up_to_level>last_level; up_to_level--)
{
    new_image_width = image_width / (int)pow(2,up_to_level-1);
    new_image_height = image_height / (int)pow(2,up_to_level-1);

    new_image_width = 2*(new_image_width/2);
    new_image_height = 2*(new_image_height/2);

    for(int temp_w = 0; temp_w < new_image_width; temp_w++)
    {
        counter = 0;

        for(int temp_h = 0; temp_h < new_image_height/2; temp_h++)
        {
            for(int filter_counter=0; filter_counter <
                number_of_elements_in_function; filter_counter++)
                image_data_out_temp[COMPUTE_OFFSET(image_width,
                                    image_height, temp_w,
                                    (counter+filter_counter)+signal_shift, image_width,
                                    new_image_height)] +=
                (image_data_in[COMPUTE_OFFSET(image_width,
                                image_height, temp_w, temp_h, image_width,
                                image_height)] * scaling_function[filter_counter] +
                image_data_in[COMPUTE_OFFSET(image_width,
                                image_height, temp_w, (temp_h+new_image_height/2),
                                image_width, image_height)] *
                wavelet_function[filter_counter]);

            counter+=2;
        }
    }
}

for(int i = 0; i<new_image_height; i++)
for(int j = 0; j<new_image_width; j++)
    image_data_in[i*image_width+j] = image_data_out_temp[i*image_width+j];

memset(image_data_out_temp, 0, image_width*image_height*sizeof(double));

for(int temp_h = 0; temp_h < new_image_height; temp_h++)
{
    counter = 0;

    for(int temp_w = 0; temp_w < new_image_width/2; temp_w++)
    {

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
for(int filter_counter=0; filter_counter <
    number_of_elements_in_function; filter_counter++)
    image_data_out_temp[COMPUTE_OFFSET(image_width,
        image_height, (counter+filter_counter)+signal_shift,
        temp_h, new_image_width, image_height)] +=
        (image_data_in[COMPUTE_OFFSET(image_width,
            image_height, temp_w, temp_h, image_width,
            image_height)] * scaling_function[filter_counter] +
            image_data_in[COMPUTE_OFFSET(image_width,
                image_height, temp_w+new_image_width/2, temp_h,
                image_width, image_height)] *
                wavelet_function[filter_counter]);

counter+=2;
}
}

for(i = 0; i<new_image_height; i++)
    for(int j = 0; j<new_image_width; j++)
        image_data_in[i*image_width+j] = image_data_out_temp[i*image_width+j];

if(up_to_level > last_level+1)
    memset(image_data_out_temp, 0,
        image_width*image_height*sizeof(double));

double result;

for(int i = 0; i<image_height; i++)
    for(int j = 0; j<image_width; j++)
    {
        result = image_data_out_temp[i*image_width+j];

        if(result > 255)
            result = 255;
        if(result < 0)
            result = 0;

        result += 0.5;

        image_data_out[i*image_width+j] = unsigned char( int(result) );
    }

delete[] image_data_out_temp;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
void wavelet_analysis_class::Forward_Wavelet_Transform_Double_Radon(double *image_data_in2, double *image_data_out, int image_width, int image_height, int transform_level)
/* One dimensional Wavelet transform in Radon Domain */
{
    if(wavelet_function == NULL || scaling_function == NULL)
        return;
    double *image_data_in = new double[image_width*image_height];
    for(int i = 0; i < image_height; i++)
        for(int j = 0; j < image_width; j++)
            image_data_in[i*image_width+j] = image_data_in2[i*image_width+j];
    memset(image_data_out, 0, image_width*image_height*sizeof(double));
    int new_image_width;
    int new_image_height;
    new_image_width = 2*(image_width/2);
    new_image_height = 2*(image_height/2);
    int counter;
    for(int up_to_level = 1; up_to_level <= transform_level; up_to_level++)
    {
        for(int temp_w = 0; temp_w < new_image_width; temp_w++)
        {
            counter = 0;
            for(int temp_h = 0; temp_h < new_image_height-1; temp_h+=2)
            {
                image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, counter, image_width, image_height)] = 0;
                image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, (counter+new_image_height/2),
                    image_width, image_height)] = 0;
                for(int filter_counter=0; filter_counter <
                    number_of_elements_in_function; filter_counter++)
                {
                    image_data_out[COMPUTE_OFFSET(image_width,
image_height, temp_w, counter, image_width,
image_height) +=
image_data_in[COMPUTE_OFFSET(image_width,
image_height, temp_w,
(filter_counter+temp_h)+signal_shift, image_width,
new_image_height)] * scaling_function[filter_counter];
image_data_out[COMPUTE_OFFSET(image_width,
image_height, temp_w, (counter+new_image_height/2),
image_width, image_height)] +=
image_data_in[COMPUTE_OFFSET(image_width,
image_height, temp_w,
(filter_counter+temp_h)+signal_shift, image_width,
new_image_height)] * wavelet_function[filter_counter];
}
counter++;}
}
memcpy(image_data_in, image_data_out, image_width*image_height*sizeof(double));
new_image_height = image_height / (int)pow(2,up_to_level);
new_image_height = 2*(new_image_height/2);
dele [image_data_in;

void wavelet_analysis_class::Inverse_Wavelet_Transform_Double.Radon(double
*image_data_in, double *image_data_out, int image_width, int
image_height, int transformed_level, int last_level )
/* Inverse one dimensional Wavelet transform of Radon data
{ if(wavelet_function == NULL || scaling_function == NULL)
return;

double *image_data_out_temp = new double[image_width*image_height];
memset(image_data_out, 0, image_width*image_height*sizeof(unsigned char));
memset(image_data_out_temp, 0, image_width*image_height*sizeof(double));

int new_image_width;
int new_image_height;
int counter = 0;
new_image_width = image_width;

for(int up_to_level = transformed_level; up_to_level>last_level; up_to_level--)
{
    new_image_height = image_height / (int)pow(2,up_to_level-1);
    new_image_height = 2*(new_image_height/2);

    for(int temp_w = 0; temp_w < new_image_width; temp_w++)
    {
        counter = 0;
        for(int temp_h = 0; temp_h < new_image_height/2; temp_h++)
        {
            for(int filter_counter=0; filter_counter <
                number_of_elements_in_function; filter_counter++)
                image_data_out_temp[COMPUTE_OFFSET(image_width,
                    image_height, temp_w,
                    (counter+filter_counter)+signal_shift, image_width,
                    new_image_height)] +=
                (image_data_in[COMPUTE_OFFSET(image_width,
                    image_height, temp_w, temp_h, image_width,
                    new_image_height)] * scaling_function[filter_counter] +
                image_data_in[COMPUTE_OFFSET(image_width,
                    image_height, temp_w, (temp_h+new_image_height/2),
                    image_width, new_image_height)] *
                wavelet_function[filter_counter]);

            counter+=2;
        }
    }

    for(int i = 0; i<new_image_height; i++)
        for(int j = 0; j<new_image_width; j++)
            image_data_in[i*image_width+j] = image_data_out_temp[i*image_width+j];

    if(up_to_level > last_level+1)
        memset(image_data_out_temp, 0,
            image_width*image_height*sizeof(double));
}

for(int i = 0; i<image_height; i++)
     for(int j = 0; j<image_width; j++)
         image_data_out[i*image_width+j] = image_data_out_temp[i*image_width+j];

delete [] image_data_out_temp;
struct my_sig_coef_list_struct
{
    int x;
    int y;
    double val; //normalized
    double original_val; //original
};

/*-----------------------------------------------*/

void wavelet_analysis::WaveletMostSigCoef(double *image_data_out, int
image_width, int image_height, int num_sig_coef, int trsf_lvl)
/* Obtain the most significant coefficients of the wavelet transform. The number of
/* coefficients is determined by the user.

{
    if(wavelet_function == NULL || scaling_function == NULL)
        return;
    if(num_sig_coef == 0)
    {
        int new_image_width;
        int new_image_height;
        int temp_w;
        int temp_h;

        new_image_width = image_width / (int)pow(2, trsf_lvl);
        new_image_height = image_height / (int)pow(2, trsf_lvl);

        new_image_width = 2*(new_image_width/2);
        new_image_height = 2*(new_image_height/2);

        for(temp_w = 0; temp_w < image_width; temp_w++)
            for(temp_h = 0; temp_h < image_height; temp_h++)
            {
                if( ! (temp_w < new_image_width && temp_h <
                        new_image_height) )
                    image_data_out[temp_w + temp_h*image_width] = 0.0;
            }
    }
    my_sig_coef_list_struct *my_new_sig_list =
        new my_sig_coef_list_struct[num_sig_coef];
    memset(my_new_sig_list, 0, sizeof(my_sig_coef_list_struct) * num_sig_coef);

    return;
}
for(int transform_level = 1; transform_level <= trsf_lvl; transform_level++)
{
    int new_image_width;
    int new_image_height;
    int temp_w;
    int temp_h;
    double mean_H = 0.0;
    double sigma_H = 0.0;

    double mean_V = 0.0;
    double sigma_V = 0.0;

    double mean_B = 0.0;
    double sigma_B = 0.0;

    new_image_width = image_width / (int)pow(2,transform_level-1);
    new_image_height = image_height / (int)pow(2,transform_level-1);

    new_image_width = 2*(new_image_width/2);
    new_image_height = 2*(new_image_height/2);

    for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
        for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
            {
                //for Horizontal
                mean_H += image_data_out[COMPUTE_OFFSET(image_width,
                image_height, temp_w, temp_h+new_image_height/2,
                image_width, image_height)];

                //for Vertical
                mean_V += image_data_out[COMPUTE_OFFSET(image_width,
                image_height, temp_w+new_image_width/2, temp_h,
                image_width, image_height)];

                //for Biodirectional
                mean_B += image_data_out[COMPUTE_OFFSET(image_width,
                image_height, temp_w+new_image_width/2,
                temp_h+new_image_height/2, image_width, image_height)];
            }

    mean_H /= double(new_image_width*new_image_height/4);
    mean_V /= double(new_image_width*new_image_height/4);
    mean_B /= double(new_image_width*new_image_height/4);

    }
for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
    for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
    {
        sigma_H += (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w, temp_h+new_image_height/2, image_width, image_height)] - mean_H) * (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h, image_width, image_height)] - mean_H);
        sigma_V += (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h, image_width, image_height)] - mean_V) * (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h, image_width, image_height)] - mean_V);
        sigma_B += (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h+new_image_height/2, image_width, image_height)] - mean_B) * (image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h+new_image_height/2, image_width, image_height)] - mean_B);
    }

    sigma_H /= double(new_image_width*new_image_height/4);
    sigma_H = sqrt(sigma_H);

    sigma_V /= double(new_image_width*new_image_height/4);
    sigma_V = sqrt(sigma_V);

    sigma_B /= double(new_image_width*new_image_height/4);
    sigma_B = sqrt(sigma_B);
for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
{

for(int i = 0; i < num_sig_coef; i++)
{
    if( fabs((image_data_out[COMPUTE_OFFSET(image_width,
        image_height, temp_w, temp_h+new_image_height/2,
        image_width, image_height)] - mean_H)/sigma_H) >
        fabs(my_new_sig_list[i].val) )
    {
        for(int j = num_sig_coef-l; j > i; j--)
        {
            my_new_sig_list[j].x = my_new_sig_list[j-1].x;
            my_new_sig_list[j].y = my_new_sig_list[j-1].y;
            my_new_sig_list[j].val = my_new_sig_list[j-1].val;
            my_new_sig_list[j].original_val =
                my_new_sig_list[j-1].original_val;
        }
        my_new_sig_list[i].x = temp_w;
        my_new_sig_list[i].y = temp_h+new_image_height/2;
        my_new_sig_list[i].val =
            ((image_data_out[COMPUTE_OFFSET(image_width,
                image_height, temp_w, temp_h+new_image_height/2,
                image_width, image_height)] - mean_H)/sigma_H);
        my_new_sig_list[i].original_val =
            image_data_out[COMPUTE_OFFSET(image_width,
                image_height, temp_w, temp_h+new_image_height/2,
                image_width, image_height)];
        break;
    }
}
for(i = 0; i < num_sig_coef; i++)
{
    if( fabs((image_data_out[COMPUTE_OFFSET(image_width,
        image_height, temp_w+new_image_width/2, temp_h,
        image_width, image_height)] - mean_V)/sigma_V) >
        fabs(my_new_sig_list[i].val) )
    {
        for(int j = num_sig_coef-l; j > i; j--)
        {
            my_new_sig_list[j].x = my_new_sig_list[j-1].x;
            my_new_sig_list[j].y = my_new_sig_list[j-1].y;
            my_new_sig_list[j].val = my_new_sig_list[j-1].val;
        }
        my_new_sig_list[i].x = temp_w;
        my_new_sig_list[i].y = temp_h+new_image_height/2;
        my_new_sig_list[i].val =
my_new_sig_list[j].original_val = my_new_sig_list[j-1].original_val;

my_new_sig_list[i].x = temp_w+new_image_width/2;
my_new_sig_list[i].y = temp_h;

my_new_sig_list[i].val =
((image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h, image_width, image_height)] - mean_V)/sigma_V);

my_new_sig_list[i].original_val =
image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h, image_width, image_height)];

break;
}

for(i = 0; i < num_sig_coef; i++)
{
    if(fabs((image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h+new_image_height/2, image_width, image_height)]) - mean_B)/sigma_B) > fabs(my_new_sig_list[i].val)
    {
        for(int j = num_sig_coef-1; j > i; j--)
        {
            my_new_sig_list[j].x = my_new_sig_list[j-1].x;
            my_new_sig_list[j].y = my_new_sig_list[j-1].y;
            my_new_sig_list[j].val = my_new_sig_list[j-1].val;
            my_new_sig_list[j].original_val =
                my_new_sig_list[j-1].original_val;
        }
        my_new_sig_list[i].x = temp_w+new_image_width/2;
        my_new_sig_list[i].y = temp_h+new_image_height/2;
        my_new_sig_list[i].val =
            ((image_data_out[COMPUTE_OFFSET(image_width, image_height, temp_w+new_image_width/2, temp_h+new_image_height/2, image_width, image_height)]) -
            mean_B)/sigma_B);
    }
my_new_sig_list[i].original_val =
image_data_out[COMPUTE_OFFSET(image_width,
image_height, temp_w+new_image_width/2,
temp_h+new_image_height/2, image_width, image_height)];
break;
}
}

// Reject Values using the user defined sigma value,
int new_image_width;
int new_image_height;
int temp_w;
int temp_h;

new_image_width = image_width / (int)pow(2,trsf_lvl);
new_image_height = image_height / (int)pow(2,trsf_lvl);
new_image_width = 2*(new_image_width/2);
new_image_height = 2*(new_image_height/2);

for(temp_w = 0; temp_w < image_width; temp_w++)
    for(temp_h = 0; temp_h < image_height; temp_h++)
    {
        if( !(temp_w < new_image_width && temp_h <
            new_image_height) )
            image_data_out[temp_w + temp_h*image_width] = 0.0;
    }
for(int i = 0; i < num_sig_coef; i++)
    image_data_out[my_new_sig_list[i].x +
    my_new_sig_list[i].y*image_width] = my_new_sig_list[i].original_val;

delete [] my_new_sig_list;

void wavelet_analysis_class::WaveletMostSigCoef_RT(double *image_data_out, int
image_width, int image_height, int num_sig_coef, int trsf_lvl)
/* Choice of the most significant coefficients of the Ridgelet transform */
{
    if(wavelet_function == NULL || scaling_function == NULL)
        return;

if(num_sig_coef == 0)
{
    int new_image_width;
    int new_image_height;
    int temp_w;
    int temp_h;
    new_image_width = image_width;
    new_image_height = image_height / (int)pow(2, trsf_lvl);

    new_image_width = 2*(new_image_width/2);
    new_image_height = 2*(new_image_height/2);

    for(temp_w = 0; temp_w < image_width; temp_w++)
        for(temp_h = 0; temp_h < image_height; temp_h++)
        {
            if( ! (temp_h < new_image_height) )
                image_data_out[temp_w + temp_h*image_width] = 0.0;
        }
    return;
}

my_sig_coef_list_struct *my_new_sig_list1 =
    new my_sig_coef_list_struct[num_sig_coef/2];
memset(my_new_sig_list1, 0, sizeof(my_sig_coef_list_struct) * num_sig_coef/2);
my_sig_coef_list_struct *my_new_sig_list2 =
    new my_sig_coef_list_struct[num_sig_coef/2];
memset(my_new_sig_list2, 0, sizeof(my_sig_coef_list_struct) * num_sig_coef/2);

for(int transform_level = 1; transform_level <= trsf_lvl; transform_level++)
{
    int new_image_width;
    int new_image_height;
    int temp_w;
    int temp_h;
    double mean_S1 = 0.0;
    double sigma_S1 = 0.0;
    double mean_S2 = 0.0;
    double sigma_S2 = 0.0;
    new_image_width = image_width;
    new_image_height = image_height / (int)pow(2, transform_level-1);

    new_image_width = 2*(new_image_width/2);
    new_image_height = 2*(new_image_height/2);
for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
    for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
    {
        //for S1 (i.e., transform level corresponding to the basically
        //horizontal lines)
        mean_S1 += image_data_out[COMPUTE_OFFSET(image_width,
                                  image_height, temp_w, temp_h+new_image_height/2,
                                  image_width, image_height)];

        //for S2 (i.e., transform level corresponding to the basically
        //vertical lines)
        mean_S2 += image_data_out[COMPUTE_OFFSET(image_width,
                                   image_height, temp_w+new_image_width/2,
                                   temp_h+new_image_height/2, image_width, image_height)];
    }

mean_S1 /= double(new_image_width*new_image_height/4);
mean_S2 /= double(new_image_width*new_image_height/4);

for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
    for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
    {
        sigma_S1 += ( (image_data_out[COMPUTE_OFFSET(image_width,
                                  image_height, temp_w, temp_h+new_image_height/2,
                                  image_width, image_height)] - mean_S1) *
                    (image_data_out[COMPUTE_OFFSET(image_width,
                                  image_height, temp_w, temp_h+new_image_height/2,
                                  image_width, image_height)] - mean_S1) );

        sigma_S2 += ( (image_data_out[COMPUTE_OFFSET(image_width,
                                  image_height, temp_w+new_image_width/2,
                                  temp_h+new_image_height/2, image_width, image_height)] -
                      mean_S2) *
                    (image_data_out[COMPUTE_OFFSET(image_width,
                                  image_height, temp_w+new_image_width/2,
                                  temp_h+new_image_height/2, image_width, image_height)] -
                      mean_S2) );
    }

sigma_S1 /= double(new_image_width*new_image_height/4);
sigma_S1 = sqrt(sigma_S1);

sigma_S2 /= double(new_image_width*new_image_height/4);
sigma_S2 = sqrt(sigma_S2);
for(temp_w = 0; temp_w < new_image_width/2; temp_w++)
for(temp_h = 0; temp_h < new_image_height/2; temp_h++)
{
    for(int i = 0; i < num_sig_coef/2; i++)
    {
        if( fabs((image_data_out[COMPUTE_OFFSET(image_width,
            image_height, temp_w, temp_h+new_image_height/2,
            image_width, image_height)] - mean_S1)/sigma_S1) >
            fabs(my_new_sig_list1[i].val) )
        {
            for(int j = num_sig_coef/2-1; j > i; j--)
            {
                my_new_sig_list1[j].x = my_new_sig_list1[j-1].x;
                my_new_sig_list1[j].y = my_new_sig_list1[j-1].y;
                my_new_sig_list1[j].val = my_new_sig_list1[j-1].val;
                my_new_sig_list1[j].original_val =
                    my_new_sig_list1[j-1].original_val;
            }
            my_new_sig_list1[i].x = temp_w;
            my_new_sig_list1[i].y = temp_h+new_image_height/2;
            my_new_sig_list1[i].val =
                ((image_data_out[COMPUTE_OFFSET(image_width,
                    image_height, temp_w, temp_h+new_image_height/2,
                    image_width, image_height)] - mean_S1)/sigma_S1);
            my_new_sig_list1[i].original_val =
                image_data_out[COMPUTE_OFFSET(image_width,
                    image_height, temp_w, temp_h+new_image_height/2,
                    image_width, image_height)];
            break;
        }
    }
}
for(i = 0; i < num_sig_coef/2; i++)
{
    if( fabs((image_data_out[COMPUTE_OFFSET(image_width,
                    image_height, temp_w+new_image_width/2,
                    temp_h+new_image_height/2, image_width, image_height)] -
                    mean_S2)/sigma_S2) > fabs(my_new_sig_list2[i].val) )

for(int j = num_sig_coeff-1; j > i; j--)
{
    my_new_sig_list2[j].x = my_new_sig_list2[j-1].x;
    my_new_sig_list2[j].y = my_new_sig_list2[j-1].y;
    my_new_sig_list2[j].val = my_new_sig_list2[j-1].val;
    my_new_sig_list2[j].original_val =
        my_new_sig_list2[j-1].original_val;
}

my_new_sig_list2[i].x = temp_w+new_image_width/2;
my_new_sig_list2[i].y = temp_h+new_image_height/2;
my_new_sig_list2[i].val =
    ((image_data_out[COMPUTE_OFFSET(image_width,
        image_height, temp_w+new_image_width/2,
        temp_h+new_image_height/2, image_width,
        image_height)] - mean_S2)/sigma_S2);
my_new_sig_list2[i].original_val =
    image_data_out[COMPUTE_OFFSET(image_width,
        image_height, temp_w+new_image_width/2,
        temp_h+new_image_height/2, image_width,
        image_height)];
break;
}
}

// Reject Values using the sigma chosen by the user...

int new_image_height;
int temp_w;
int temp_h;

new_image_height = image_height / (int)pow(2,trans_lvl);

new_image_height = 2*(new_image_height/2);
for(temp_w = 0; temp_w < image_width; temp_w++)
    for(temp_h = 0; temp_h < image_height; temp_h++)
    {
        if(! (temp_h < new_image_height) )
            image_data_out[temp_w + temp_h*image_width] = 0.0;
    }
for(int i = 0; i < num_sig_coef/2; i++)
{
    image_data_out[my_new_sig_list1[i].x +
    my_new_sig_list1[i].y*image_width] = my_new_sig_list1[i].original_val;
    image_data_out[my_new_sig_list2[i].x +
    my_new_sig_list2[i].y*image_width] =
    my_new_sig_list2[i].original_val;
}

delete [] my_new_sig_list1;
delete [] my_new_sig_list2;

]*)
/* Set up the wavelet transform filters */
/*/gui_wavelet_setup_dlg.cpp : implementation file
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
void wavelet_setup_dlg::OnSelchangeWaveletsListBox()
{
    // TODO: Add your control notification handler code here
    CString name_cs;

    m_wavelet_list_box.GetText( m_wavelet_list_box.GetCurSel(), name_cs );

    for(int i = 0; i<number_of_wavelet_data_structure_elements; i++)
    if(name_cs == ptr_to_wavelet_list[i].wavelet_name)
    {
        m_number_of_elements =
        ptr_to_wavelet_list[i].number_of_elements;
        m_vector_shift = ptr_to_wavelet_list[i].shift;
        m_scaling_coefficients = ptr_to_wavelet_list[i].elements_list;
        m_name_of_wavelet = name_cs;
    }

    CString *temp_coef = new CString[m_number_of_elements];
    m_scaling_coefficients.TrimLeft();
    m_scaling_coefficients.TrimRight();

    int counter = 0;
    int length = 0;
    char temp_char;

    length = m_scaling_coefficients.GetLength();
CString cs_temp;
for(int i = 0; i<m_number_of_elements; i++)
{
    cs_temp.Empty();

    while(counter<length && m_scaling_coefficients[counter] != ' ')
    {
        temp_char = m_scaling_coefficients.GetAt(counter);
        cs_temp += temp_char;
        counter++;
    }
    counter++;
    temp_coef[i] = cs_temp;
}

counter = -1;

for(i=0; i<m_number_of_elements; i++)
{
    if(counter == -1)
    {
        if(temp_coef[i].GetAt(0) == '.')
            temp_coef[i].SetAt(0, '.');
        else
        {
            CString temp;
            temp = ",";
            temp+=temp_coef[i];
            temp_coef[i] = temp;
        }
    }
    temp_coef[i].TrimLeft();
}

counter *= -1;

m_wavelet_coefficients.Empty();
for(i=m_number_of_elements-1; i>=0; i--)
{
    m_wavelet_coefficients += temp_coef[i];
    if(i>0)
        m_wavelet_coefficients += " ";
}
delete [] temp_coef;
}
}

UpdateData(false);
OnPlotgraph();
}
CHAPTER 9

CONCLUSIONS

9.1 Practical Results

The purpose of this section is to show how, in comparison to Wavelets, Ridgelets are better suited in representing functions with linear singularities. To demonstrate this we have implemented the algorithm mentioned in Chapter 8 and have considered images having resolutions 256 x 256 pixels. Thus, when computing the ridgelet and wavelet transforms we will have 262,144 and 65,536 coefficients, respectively.

Once the transformations have been computed we will reconstruct the image by choosing a certain number of significant coefficients. To choose the coefficients for the reconstruction stage the threshold applied was $T = \pm 2\sigma$ (i.e., two standard deviations). Equal number of coefficients was taken from each level of the respective transformations.

In our first set of figures we show different representations of the mutilated Gaussian given by the equation

$$f(x,y) = 1_{[y : 2y < x]} e^{-(x^2+y^2)}$$
Figure 9.1.1 shows the original image, whereas figures 9.1.2 through 9.1.5 show the reconstruction obtained with 8 coefficients using Ridgelets, Meyer wavelets (see Section 5.2), Coifman-6 and Daubechies-4, respectively.

Figure 9.1.1 - Half Dome.

Figure 9.1.2 - Reconstruction with 8 Ridgelet coefficients.

Figure 9.1.3 - Reconstruction with 8 Meyer wavelet coefficients.
In this example it is clear how with just a few coefficients the Ridgelet transform is able to preserve the edge more precisely than the wavelet transform.

In the next set of figures we can see how that difference becomes more evident when considering just a few additional coefficients.

First let us see the improvement in the reconstruction of the Half Dome with 64 coefficients.
Figure 9.1.6 – Reconstruction with
64 Ridgelet coefficients.

Figure 9.1.7 – Reconstruction with
64 Meyer wavelet coefficients.

Figure 9.1.8 – Reconstruction with
64 Coif - 6 wavelet coefficients.

Figure 9.1.9 – Reconstruction with
64 Dau - 4 wavelet coefficients.

So we see that the linear singularity of the Gaussian is more
clearly noticeable in the Ridgelet transform reconstruction.

Now let us consider a more interesting picture.
Figure 9.1.10 – Original image.

Figure 9.1.11 – Reconstruction with 64 Ridgelet coefficients.

Figure 9.1.12 – Reconstruction with 64 Meyer wavelet coefficients.
We observe that the most significant Ridgelet coefficients have preserved the most important features of the object, namely the edges. We also observe that, when considering just a few coefficients, the Daubechies 4 and Coifman 6 wavelets in comparison with the Meyer wavelet, perform worse at representing this type of images. Therefore, from this point on consideration will be given only to images that were analyzed and reconstructed using the Meyer wavelets.

Now, if we use Figure 9.1.10 as we have in the past but with an image reconstruction of 256 and 1024 significant coefficients, respectively, we obtain the following results:
Clearly, the Ridgelet transform has recognized and stored all the edges rather perfectly, whereas the Wavelet transform yields an image with nearly all of the edges. Moreover, the edges reconstructed by the
wavelet transform appear jagged. In the case of the Ridgelet reconstruction with 1024 significant coefficients [Figure 9.1.17] we observe a sharper appearance of edges, while the Wavelet reconstruction with an equal amount of significant coefficients, yields an image having fainter edge appearance [Figure 9.1.18].

Finally, the last two sets of figures show pictures with a number of straight edges having multiple directions. Once more, the differences between the abilities of the two transforms at recognizing and storing edges is clearly noticeable.

Figure 9.1.19 – Original image.
Figure 9.1.20 – Reconstruction with 64 Ridgelet coefficients.

Figure 9.1.21 – Reconstruction with 64 Meyer wavelet coefficients.

Figure 9.1.22 – Reconstruction with 256 Ridgelet coefficients.

Figure 9.1.23 – Reconstruction with 256 Meyer wavelet coefficients.
Figure 9.1.24 – Reconstruction with 1024 Ridgelet coefficients.

Figure 9.1.25 – Reconstruction with 1024 Meyer wavelet coefficients.

In this last example we will only consider reconstructions with 1024 and 2048 coefficients.

Figure 9.1.26 – Original image.
Figure 9.1.27 – Reconstruction with 1024 Ridgelet coefficients.

Figure 9.1.28 – Reconstruction with 1024 Meyer wavelet coefficients.

Figure 9.1.29 – Reconstruction with 2048 Ridgelet coefficients.

Figure 9.1.30 – Reconstruction with 2048 Meyer wavelet coefficients.
Again, the ability of reconstructing a visually superior image via the use of the Ridgelet transform is evident; furthermore, the image quality is also increased by increasing the number of significant coefficients. The inability of the wavelet transform for obtaining similar results, even with a much larger significant coefficient set, is also apparent.

Thus, the examples just presented produce a visual statement that allow us to see the superiority of the Ridgelet transform over the Wavelet transform for representing objects with linear singularities.

9.2 Final Remarks

The complexity of the algorithm that implements the Discrete Ridgelet Transform (DRT) may be seen as a disadvantage due to the fact that the DRT is a much slower algorithm than the Discrete Wavelet Transform (DWT). Yet, we must keep in mind that Ridgelet analysis is very recent and therefore, there is ample room for further improvement.

In 1998 David Donoho introduced a modification to the notion of Ridgelets, the "Orthonormal Ridgelets", which provide a complete orthonormal system in $L^2(\mathbb{R}^2)$. Among other things, this orthonormal approach allows for simple and effective nonlinear approximations.

Although Ridgelets are efficient in dealing with linear singularities they are not efficient at representing objects with curved singularities. To overcome this weakness E. Candès and D. Donoho have used
Ridgelets to construct a system of so-called "Curvelets" which are indeed very efficient in dealing with these type of singularities. It has also been shown that Curvelets are superior to the classical systems. For more information on the evolution of these new tools derived from the Ridgelet transform we refer the reader to Dr. David Donoho's web page (www-stat.stanford.edu/~donoho) where an extensive list of papers and references on the subject can be found.
APPENDIX A

[A1] – The Hilbert Transform is a basic example of a so-called singular integral operator.

Definition:

The Hilbert transform of $f$ is defined as

$$H[f](x) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(t)}{t-x} dt$$

where the integral is a Cauchy principal value.

The next theorem gives us an explicit formula for the Hilbert transform in terms of the Fourier transform.

Theorem:

If $f \in L^1(\mathbb{R})$, then $Hf \in L^1(\mathbb{R})$ and

$$\mathcal{F}[Hf(x)](\omega) = -i \text{sign}(\omega) \mathcal{F}[f(x)](\omega)$$

Proof:[Sketch]

$$\mathcal{F}[Hf(x)](\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-i\omega x} H[f](x) \, dx = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-i\omega y} \frac{f(t)}{t-x} \, dx \, dt$$

$$= \frac{1}{\pi} \int_{-\infty}^{\infty} f(t) \left( \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-i\omega y} \, dy \right) \, dt = \frac{1}{\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{i\omega y} \, dy \, dt$$

$$= \frac{1}{\pi} \int_{-\infty}^{\infty} f(t) e^{i\omega y} \left( -i \sqrt{\frac{\pi}{2}} \text{sign}(\omega) \right) \, dt = \frac{1}{\pi} \int_{-\infty}^{\infty} f(t) e^{i\omega y} \left( -i \sqrt{\frac{\pi}{2}} \text{sign}(\omega) \right) \, dt$$

$$= -i \text{sign}(\omega) \int_{-\infty}^{\infty} e^{-i\omega t} f(t) \, dt = -i \text{sign}(\omega) \mathcal{F}[f(x)](\omega).$$
Remark: From the above theorem and Parseval's relation it follows that $\mathbf{H} : L^2(\mathbb{R}) \rightarrow L^2(\mathbb{R})$ is a unitary mapping.

[A2] – Since

$$D_n(t) = \frac{\sin(\pi t)}{m \sin\left(\frac{\pi t}{m}\right)}$$

and $m = 2n$ we have that

$$D_n(t + m) = \frac{\sin[\pi(t + m)]}{m \sin\left(\frac{\pi(t + m)}{m}\right)} = \frac{-\sin[\pi t + 2\pi n]}{m \sin\left(\frac{\pi t + 2\pi n}{m}\right)} = \frac{-\sin(\pi t)}{m \sin\left(\frac{\pi t}{m}\right)} = -D_n(t)$$

that is, $|D_n(t + m)| = |D_n(t)|$ so that the "periodicity" of the kernel will result in wrap-around of the data. Yet, the choice $m = 2n$ assures the wrap-around effect will not affect the result of the summation.

In fact, by choosing $D_n(t)$ we are interpolating a vector of $m = 2n$ samples which, applied to our input array $I(u, v)$, $-\frac{n}{2} \leq u, v < \frac{n}{2}$, is equivalent to symmetrically padding $I$ with zeros. Suppose we want to calculate $I$ at a point $(x_i, y_i)$ for a line $y = sx + t$ and say $\frac{n}{2} \leq y_i < \frac{3n}{2}$, i.e., although $y_i$ does not belong to the original input array $I$ it belongs to the padded version of $I$. Since $m = 2n$, given the "periodicity" of $D_n(t)$, the points of the form $y_2 = y_i - 2n$ will be considered. But then, since $|s| \leq 1$ $-n \leq t < n$, it follows that $-\frac{3n}{2} \leq y_2 < -\frac{n}{2}$ which means that the interpolated samples...
at $\frac{n}{2} \leq y \leq \frac{3n}{2}$ do not overlap with the samples of $I$ at $-\frac{n}{2} \leq y < \frac{n}{2}$.

Therefore, the wrap-around occurs over zero samples and does not affect the summation.

Graphically,
APPENDIX B

[B1] - The polarization identity recovers $\langle f, g \rangle$ from $\|f \pm g\|$, $\|f \pm ig\|$, in fact,

$$\langle f, g \rangle = \frac{1}{4} \left( \|f + g\|^2 - \|f - g\|^2 + i\|f + ig\|^2 - i\|f - ig\|^2 \right)$$

Hence,

$$A(f, g) = \frac{4}{4} \left( \|f + g\|^2 - \|f - g\|^2 + i\|f + ig\|^2 - i\|f - ig\|^2 \right)$$

$$= \frac{1}{4} \left[ \sum_{j \in J} |\langle f + g, \varphi_j \rangle|^2 - \sum_{j \in J} |\langle f - g, \varphi_j \rangle|^2 + i \sum_{j \in J} |\langle f + ig, \varphi_j \rangle|^2 - i \sum_{j \in J} |\langle f - ig, \varphi_j \rangle|^2 \right]$$

$$= \sum_{j \in J} \langle f, \varphi_j \rangle \langle \overline{g}, \varphi_j \rangle = \sum_{j \in J} \langle f, \varphi_j \rangle \langle \varphi_j, g \rangle.$$ 

[B2] - To prove that $\sum_{j \in J} c_j \varphi_j$ converges to $F'c$ in norm we must show that for any $\{J_n\}_{n \in \mathbb{N}}$ such that $J_n \subseteq J_m$ for all $n \leq m$ and $\bigcup_{n \in \mathbb{N}} J_n = J$ it follows that

$$\left\| F'c - \sum_{j \in J_n} c_j \varphi_j \right\| \to 0.$$ 

First, let $n_2 \geq n_1 \geq n_0$. Then, since

$$\left\| \sum_{j \in J_{n_2}} c_j \varphi_j - \sum_{j \in J_{n_1}} c_j \varphi_j \right\| = \sup_{\|f\|=1} \left| \langle \sum_{j \in J_{n_2} \setminus J_{n_1}} c_j \varphi_j, f \rangle \right|$$

it follows that
\[
\left\| \sum_{j \in J_2} c_j \varphi_j - \sum_{j \in J_1} c_j \varphi_j \right\| \leq \sup_{\|f\|_2} \left( \sum_{j \in J_2 \cup J_1} |c_j|^2 \right)^{1/2} \left( \sum_{j \in J} \langle \varphi_j, f \rangle^2 \right)^{1/2}
\]

\[
\leq B^2 \left( \sum_{j \in J_2 \cup J_1} |c_j|^2 \right)^{1/2} \rightarrow 0
\]

Hence, \( \tau_n = \sum_{j \in J_2} c_j \varphi_j \) constitutes a Cauchy sequence with limit \( \tau \in L^2(\mathbb{R}) \).

Now, for this \( \tau \in L^2(\mathbb{R}) \) we have that:

\[
\langle \tau, f \rangle = \lim_{n \to \infty} \langle \tau_n, f \rangle = \lim_{n \to \infty} \sum_{j \in J_2} c_j \langle \varphi_j, f \rangle = \sum_{j \in J} c_j \langle \varphi_j, f \rangle = \sum_{j \in J} \langle (\mathcal{F}_c), f \rangle = \langle c, Ff \rangle
\]

for all \( f \in L^2(\mathbb{R}) \). Therefore, \( \tau = F^* c \).

\[\text{[B3]} \quad \text{Saying that } f = \sum_{j \in J} c_j \varphi_j \text{ is equivalent to saying that } F^* c = f.\]

Now, let \( a \in \text{Range}(F) = \text{Range}(\mathcal{F}) \), \( b \in \text{Range}(F)^\perp \) and \( c = a + b \).

In particular \( a \perp b \), hence \( \|c\|^2 = \|a\|^2 + \|b\|^2 \).

Since \( a \in \text{Range}(F) \), there exists \( g \in \mathcal{H} \) such that \( a = \mathcal{F} g \) and \( c = \mathcal{F} g + b \).

Hence, \( f = F^* c = F^* \mathcal{F} g + F^* b \). But \( b \in \text{Range}(F)^\perp = \ker(F^*) \), so that \( f = F^* \mathcal{F} g \). But \( F^* \mathcal{F} = \text{Id} \) so \( f = g \) and \( c = \mathcal{F} f + b \). Therefore,

\[
\sum_{j \in J} |c_j|^2 = \|c\|^2 = \|\mathcal{F} f\|^2 + \|b\|^2 = \sum_{j \in J} \left| \langle f, \varphi_j \rangle \right|^2 + \|b\|^2 > \sum_{j \in J} \left| \langle f, \varphi_j \rangle \right|^2.
\]

It remains to show that \( \sum_{j \in J} \left| \langle f, \varphi_j \rangle \right|^2 + \sum_{j \in J} \left| \langle f, \varphi_j \rangle \right|^2 - c_j^2 = \sum_{j \in J} |c_j|^2 \).

To simplify the notation let \( a_j = \langle f, \varphi_j \rangle \).

Note that,
\[ \langle f_j, (F^*F)^{-1}f \rangle = \langle (F^*F)^{-1}f, f \rangle = \langle \phi_j, f \rangle = a_j. \]

Since \( f = \sum_{j \in J} c_j \phi_j \) and \( f = \sum_{j \in J} a_j \phi_j \) we have that
\[
\langle f, (F^*F)^{-1}f \rangle = \sum_{j \in J} a_j \langle \phi_j, (F^*F)^{-1}f \rangle = \sum_{j \in J} a_j \langle \phi_j, (F^*F)^{-1}f \rangle = \sum_{j \in J} a_j \overline{a_j} = \sum_{j \in J} |a_j|^2.
\]

On the other hand,
\[
\langle f, (F^*F)^{-1}f \rangle = \sum_{j \in J} c_j \langle \phi_j, (F^*F)^{-1}f \rangle = \sum_{j \in J} c_j \langle \phi_j, (F^*F)^{-1}f \rangle = \sum_{j \in J} c_j \overline{a_j}.
\]

Hence,
\[
\sum_{j \in J} |a_j|^2 + \sum_{j \in J} |a_j - c_j|^2 = \sum_{j \in J} |a_j|^2 + \sum_{j \in J} \left( |a_j|^2 - a_j \overline{c_j} - c_j \overline{a_j} + |c_j|^2 \right) = \sum_{j \in J} |c_j|^2.
\]

\[ \text{[B4]} \text{ - Let } m_0 \text{ be fixed and } \alpha_n \text{ such that } a_n = \left\langle \phi_{m_0}, S^{-1} \phi_n \right\rangle = \left\langle S^{-1} \phi_{m_0}, \phi_n \right\rangle \text{.} \]

We know that \( \phi_{m_0} = \sum_{n \in \mathbb{N}} a_n \phi_n \) but also \( \phi_{m_0} = \sum_{n \in \mathbb{N}} c_n \phi_n \) where \( c_n = \delta_{nm_0} \) . Then, by Proposition 4.4, it follows that
\[
1 = \sum_{j \in J} |c_j|^2 = \sum_{j \in J} |a_j|^2 + \sum_{j \in J} |a_j - c_j|^2 = |a_{m_0}|^2 + \sum_{n \neq m_0} |a_n|^2 + |a_{m_0} - 1|^2 + \sum_{n \neq m_0} |a_n|^2.
\]

Hence
\[ 1 = |a_{m_0}|^2 + |a_{m_0} - 1|^2 + 2 \sum_{n \neq m_0} |a_n|^2. \]  \hspace{1cm} (IX)

Now, if \( a_{m_0} = \langle \varphi_{m_0}, S^{-1} \varphi_{m_0} \rangle = 1 \) then (IX) yields

\[ 1 = 1 + 2 \sum_{n \neq m_0} |a_n|^2, \]

so that

\[ |a_n|^2 = 0, \text{ for all } n \neq m_0. \]

Therefore, \( \langle \varphi_{m_0}, S^{-1} \varphi_n \rangle = 0 \) for all \( n \neq m_0 \), i.e., \( S^{-1} \varphi_{m_0} \) is orthogonal to all \( \varphi_n \), \( n \neq m_0 \). But \( \langle \varphi_{m_0}, S^{-1} \varphi_{m_0} \rangle = 1 \), so that \( S^{-1} \varphi_{m_0} \neq 0 \) and \( \{ \varphi_n \}_{n \neq m_0} \) is not complete.

On the other hand, if \( a_{m_0} = \langle \varphi_{m_0}, S^{-1} \varphi_{m_0} \rangle \neq 1 \), then \( \varphi_{m_0} = \sum_{n \in N} a_n \varphi_n \) yields

\[ \varphi_{m_0} = \frac{1}{1-a_{m_0}} \sum_{n \in N} a_n \varphi_n. \]

Thus, for all \( f \in \mathcal{H} \) we have

\[ \left| \langle f, \varphi_{m_0} \rangle \right|^2 = \left| \frac{1}{1-a_{m_0}} \sum_{n \in N} a_n \langle f, \varphi_n \rangle \right|^2 \leq \left[ \frac{1}{1-a_{m_0}} \sum_{n \in N} |a_n|^2 \right] \left[ \sum_{n \in N} |\langle f, \varphi_n \rangle|^2 \right], \]

say,

\[ \left| \langle f, \varphi_{m_0} \rangle \right|^2 \leq C \sum_{n \neq m_0} \left| \langle f, \varphi_n \rangle \right|^2. \]
Therefore,

$$\sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2 = |\langle f, \varphi_{n_0} \rangle|^2 + \sum_{n \neq n_0} |\langle f, \varphi_n \rangle|^2 \leq (1+C) \sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2$$

and

$$A \| f \|^2 \leq \sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2 \leq (1+C) \sum_{n \neq n_0} |\langle f, \varphi_n \rangle|^2 \leq (1+C) \sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2.$$ 

In other words,

$$\frac{A}{(1+C)} \| f \|^2 \leq \sum_{n \in \mathbb{N}} |\langle f, \varphi_n \rangle|^2 \leq B \| f \|^2$$

and \{\varphi_n\}_{n=n_0} is a frame.
APPENDIX C

[C1] - We have to show that if \( \psi \in s(\mathbb{R}) \) the admissibility condition

\[
K_{\psi} = \int \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi < \infty \tag{I}
\]

is equivalent to the requirement of vanishing moments:

\[
\int t^k \psi(t) dt = 0 \quad k \in \{0, 1, \ldots, \left[ \frac{d+1}{2} \right] - 1 \}.
\]

Assume \( \psi \in s(\mathbb{R}) \) and (I) holds. Then \( \hat{\psi}(0) = 0 \).

In fact, if \( \hat{\psi}(0) \neq 0 \), by continuity there exist \( \varepsilon > 0 \) and a neighborhood \( I \) about the origin such that \( |\hat{\psi}(\xi)| > \varepsilon \) for all \( \xi \in I \). Then,

\[
\int \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi \geq \int \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi > \varepsilon^2 \int \frac{d\xi}{|\xi|^d} = \infty
\]

which contradicts (I). Therefore \( \hat{\psi}(0) = 0 \). Now, since \( \psi \in C^\infty(\mathbb{R}) \), it follows that for each \( m \in \mathbb{N} \) there exists a neighborhood \( I \) about the origin such that for all \( \xi \in I \) we can write

\[
\hat{\psi}(\xi) = \sum_{n=0}^{\infty} \hat{\psi}^{(n)}(0) \frac{\xi^n}{n!} + o(\xi^n) \tag{II}
\]

Assume to the contrary that there exists an index \( m, 0 \leq m \leq \left[ \frac{d+1}{2} \right] - 1 \),

such that \( \hat{\psi}^{(k)}(0) = 0 \) for all \( 0 \leq k < m \) and \( \hat{\psi}^{(m)}(0) \neq 0 \). Then, (II) becomes
\[ \hat{\psi}(\xi) = \psi^{(m)}(0) \frac{\xi^m}{m!} + o(\xi^m). \]

Since \(\psi^{(m)}(0) \neq 0\), there exists a neighborhood \(I\) about the origin such that for all \(\xi \in I\),

\[ |o(\xi^m)| \leq \left| \psi^{(m)}(0) \right| \frac{|\xi|^m}{2m!} \]

and, then,

\[ |\hat{\psi}(\xi)| = \left| \psi^{(m)}(0) \frac{\xi^m}{m!} + o(\xi^m) \right| \geq \left| \psi^{(m)}(0) \right| \frac{|\xi|^m}{m!} - |o(\xi^m)| \geq \left| \psi^{(m)}(0) \right| \frac{|\xi|^m}{2m!}. \]

Hence,

\[ \int_{|\xi| = \delta} \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi \geq \int_{|\xi| = \delta} \frac{|\hat{\psi}(\xi)|^2}{|\xi|^d} d\xi \geq \left| \psi^{(m)}(0) \right|^2 \frac{1}{2m!} \int_{|\xi| = \delta} \frac{d\xi}{|\xi|^{d-2m}}. \]  \(\text{(III)}\)

Now, since \(0 < m \leq \left\lfloor \frac{d+1}{2} \right\rfloor - 1\) we obtain that \(d - 2m \geq 1\) and (III) yields a contradiction. Therefore,

\[ \hat{\psi}^{(k)}(0) = 0 \text{ for all } k \in \{0,1,\ldots,\left\lfloor \frac{d+1}{2} \right\rfloor - 1\} \]

and, by the properties of the Fourier Transform, we obtain

\[ \int_{t^k}^\psi(t) dt = 0, \ k \in \{0,1,\ldots,\left\lfloor \frac{d+1}{2} \right\rfloor - 1\} \]

\[ [C2] - R^n f(\xi) = \hat{f}(\xi u) \]

Recall that \(R^n f(s) = R f(s,u) = \int f(x) dS_x = \int f(x) \delta(s - u \cdot x) dx\) so that
\[
\left[ R_u f(s) \right](\xi) = \int_{-\infty}^{\infty} e^{-i\xi t} R_u f(s) ds = \int_{-\infty}^{\infty} e^{-i\xi t} \int_{\mathbb{R}^d} f(x) \delta(s - u \cdot x) dx ds
\]

and, by Fubini's theorem it follows that
\[
\left[ R_u f(s) \right](\xi) = \int_{\mathbb{R}^d} f(x) \int_{-\infty}^{\infty} e^{-i\xi t} \delta(s - u \cdot x) ds \ dx = \int_{\mathbb{R}^d} f(x) e^{-i\xi (u \cdot x)} dx = \hat{f}(\xi u)
\]

[C3] - From the properties of the Fourier Transform we know that \( \mathcal{F}(f \ast g) = \mathcal{F}(f) \cdot \mathcal{F}(g) \)

Hence,
\[
\mathcal{F} \left[ \psi_a \ast \left( \hat{\psi}_a \ast R_u f \right) \right](\xi) = \left[ \hat{\psi}_a \cdot \hat{\psi}_a \cdot \hat{R_u f} \right](\xi).
\]

But,
\[
\left( \hat{\psi}_a \cdot \hat{\psi}_a \right)(\xi) = \left[ a^{\frac{1}{2}} \hat{\psi}(\xi a) \right] \left[ a^{\frac{1}{2}} \overline{\hat{\psi}(\xi a)} \right] = a \left| \hat{\psi}(\xi a) \right|^2.
\]

In fact,
\[
\left[ \hat{\psi}_a(t) \right](\xi) = \int_{-\infty}^{\infty} e^{-i\xi t} a^{\frac{1}{2}} \psi \left( \frac{t}{a} \right) dt = \int_{-\infty}^{\infty} e^{-i\xi x} a^{\frac{1}{2}} \psi (s) ds = a^{\frac{1}{2}} \hat{\psi}(\xi a)
\]

and since \( \tilde{\psi}(x) = \psi(-x) \) it is clear that \( \hat{\psi}_a(\xi) = a^{\frac{1}{2}} \overline{\hat{\psi}(\xi a)} \).

[C4] - Recall that \( f \in L^1 \cap L^2(\mathbb{R}^d) \).

If \( R_u f \in L^1(\mathbb{R}) \) and \( \psi_a \in L^2(\mathbb{R}) \) we have to show that \( \left\| \omega_{a,u} \right\|_2 \leq \left\| \psi_a \right\|_2 \left\| f \right\|_1 \).
As before let $\omega_{a,u}(b) = \left[\tilde{\psi}_a * R_u f\right](b)$. Given that $\psi(x) = \psi(-x)$ one obtains that

$$\left\| \omega_{a,u}(b) \right\|_2 = \left\| \tilde{\psi}_a * R_u f \right\|_2 \leq \left\| \tilde{\psi}_a \right\|_2 \left\| R_u f \right\|_1 = \left\| \psi_a \right\|_2 \left\| R_u f \right\|_1$$

(I)

On the other hand,

$$\left\| R_u f \right\|_1 = \int_{-\infty}^{\infty} \left| R_u f(s) \right| ds = \int_{-\infty}^{\infty} \int f(x) \delta(s-u \cdot x) dx \left| ds \right| \leq \int_{-\infty}^{\infty} \int f(x) \delta(s-u \cdot x) \left| dx \right| ds$$

$$= \int_{-\infty}^{\infty} \int f(x) ds \int_{-\infty}^{\infty} f(x) dx = \left\| f \right\|_1$$

(II)

Hence, from (I) it follows that $\left\| \omega_{a,u} \right\|_2 \leq \left\| \psi_a \right\|_2 \left\| f \right\|_1$, as we wanted to show.

\[\text{[C5]} \quad \left\| \omega_{a,u} \right\|_1 \leq a^{\frac{1}{2}} \left\| \psi \right\|_1 \left\| f \right\|_1\]

In fact,

$$\left\| \omega_{a,u} \right\|_1 = \int_{-\infty}^{\infty} \left| \omega_{a,u}(b) \right| db = \int_{-\infty}^{\infty} \left| \tilde{\psi}_a * R_u f \right|(b) db = \int_{-\infty}^{\infty} \int \tilde{\psi}_a(t) R_u f(t-b) dt db$$

so that from Fubini's theorem and [APPENDIX C4, (II)] it follows that

$$\left\| \omega_{a,u} \right\|_1 \leq \int_{-\infty}^{\infty} \int \tilde{\psi}_a(t) \left| R_u f(t-b) \right| dt db = \int_{-\infty}^{\infty} \left| \tilde{\psi}_a(t) \right| \left\{ \int_{-\infty}^{\infty} \left| R_u f(t-b) \right| db \right\} dt$$

$$= \left\| R_u f \right\|_1 \int_{-\infty}^{\infty} \left| \tilde{\psi}_a(t) \right| dt = a^{\frac{1}{2}} \left\| R_u f \right\|_1 \left\| \psi \right\|_1 \leq a^{\frac{1}{2}} \left\| f \right\|_1 \left\| \psi \right\|_1.$$
If \( \phi_\lambda(x) = (2\pi \lambda)^{-\frac{d}{2}} e^{-\frac{|x|^2}{4\lambda}} \), \( x \in \mathbb{R}^d \) then, \( R_\lambda \phi_\lambda(s) = (2\pi \lambda)^{-\frac{d}{2}} e^{-\frac{|s|^2}{4\lambda}} \).

Let \( U \) be the unitary linear transformation satisfying \( x'_1 = (Ux)_1 = u \cdot x \).

Then applying \( U \) and Fubini's theorem,

\[
\left[ R_\lambda e^{-\frac{|x|^2}{4\lambda}} \right](s) = \int_{\mathbb{R}^d} e^{-\frac{|x|^2}{4\lambda}} \delta(s - u \cdot x) \, dx = \int_{\mathbb{R}^d} e^{-\frac{|s|^2}{4\lambda}} \delta(s - x'_1) \, dx' = \int_{\mathbb{R}^{d+1}} e^{-\frac{|y|^2}{4\lambda}} \delta(s - x'_1) \, dy = \pi^{-\frac{d}{2}} e^{-\frac{|s|^2}{4\lambda}}.
\]

For example, if \( d = 2 \), the change of variables is given by the unitary linear transformation

\[
\begin{pmatrix} y \\ w \end{pmatrix} = \begin{pmatrix} u_1 & u_2 \\ -u_2 & u_1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]

where \( u = (u_1, u_2) \in S^{d-1} \) is the direction vector. Then,

\[
\left[ R_\lambda e^{-\frac{|x|^2}{4\lambda}} \right](s) = \int_{\mathbb{R}^2} e^{-\frac{|x|^2}{4\lambda}} \delta(s - u_1 x + u_2 y) \, dx dy = \int_{\mathbb{R}^2} e^{-\frac{|s|^2}{4\lambda}} \delta(s - v) \, dv dw = \int_{\mathbb{R}} e^{-\frac{|w|^2}{4\lambda}} \, dw = \sqrt{\pi} e^{-\frac{|s|^2}{4\lambda}}.
\]

Now, going back to our original problem, we obtain that

\[
R_\lambda \phi_\lambda(s) = (2\pi \lambda)^{-\frac{d}{2}} \int e^{-\frac{|x|^2}{4\lambda}} \delta(s - u \cdot x) \, dx = (2\pi \lambda)^{-\frac{d}{2}} (2\lambda)^{\frac{d}{2}} \int e^{-|t|^2} \delta(s - u \cdot (\sqrt{2\lambda}t)) \, dt = \pi^{-\frac{d}{2}} (2\lambda)^{\frac{d}{2}} \left[ R_\lambda e^{-\frac{|t|^2}{4\lambda}} \right](s).
\]

and the result follows.
[C7] - \( \{ R \phi \}_k \) is an approximate identity in \( L^1(\mathbb{R}) \).

For simplicity let \( \phi_\lambda = R \phi \).

If \( \phi(t) = (2\pi)^{-\frac{1}{2}} e^{-\frac{t^2}{2}} \), it is clear that \( \phi \) is nonnegative and \( \int_{\mathbb{R}} (2\pi)^{-\frac{1}{2}} e^{-\frac{t^2}{2}} \, dt = 1 \).

Then, for any \( \lambda > 0 \), \( \phi_\lambda(t) = (2\pi\lambda)^{-\frac{1}{2}} e^{-\frac{t^2}{2\lambda}} \) and for all \( f \in L^1(\mathbb{R}) \) we have

\[
\left\| f \ast \phi_\lambda - f \right\|_1 \to 0 \quad \lambda \to 0
\]
REFERENCES


VITA

Graduate College
University of Nevada, Las Vegas

Maria Beatriz Teruel

Local Address:
3875 Cambridge Street Apt. 216
Las Vegas, Nevada 89119

Home Address:
Mitre 1252 Dto. 14
Bahia Blanca, Buenos Aires 8000
Argentina

Degrees:
    Licenciatura en Matematica, 1999
    Universidad Nacional del Sur, Bahia Blanca
    Buenos Aires, Argentina

Thesis Title: Ridgelets: A Promising New Wavelet-Like Transform to Represent Objects with Linear Singularities.

Thesis Examination Committee:
    Chairperson, Dr. David G. Costa, Ph. D.
    Committee Member, Dr. Xi Li, Ph. D.
    Committee Member, Dr. Corran Webster, Ph. D.
    Graduate Faculty Representative, Dr. Evangelos Yfantis, Ph. D.