FPGA Based Image Processing
With The Curvelet Transform

John Wisinger and Rabi Mahapatra
Department of Computer Science
Texas A&M University
College Station, TX 77840
979-845-5787
{jlw9843, rabi} @cs.tamu.edu

ABSTRACT
In the past few years, image processing has begun to make its way into many new areas, both academic and commercial. One of the most popular areas is computer generated graphics. This includes films, video games, medical imaging, and various other multimedia systems for both entertainment and more serious applications. A fairly recent development in this field is the curvelet transform. The curvelet transform was designed to overcome deficiencies in the wavelet and similar transforms.

Although impressive performance can be achieved with curvelets, the complexity of their implementation is quite a drain on standard microprocessors. It is for this reason that an FPGA implementation was developed. By offloading some of the processing work into a properly configured FPGA, speeds can be achieved in excess of one hundred times faster than current high-end servers.

This increase in processing speed and image representation ability combine to have some useful applications. Now, highly complex image processing can be done in small areas allowing for the design systems that were previously not feasible to develop. By using the concepts presented in this paper, ideas have come about for the development of a large-scale Boltzman equation solver, and a satellite hyper spectral imaging system. The Boltzman equation solver has been developed before, but only by using very costly and space consuming servers. Design of the satellite hyper spectral imaging system has been hindered by the low data transmission rate of the communication system. By processing some of the data on the system itself, this problem is removed.

Keywords
Curvelet, Radon, Ridgelet, FPGA

1. INTRODUCTION
Image processing has become a very popular field in the past few years. This is primarily due to the fast entrance of digital imaging into the entertainment industry (including movies and video games). There is often a need to store large amount of image data and process it very quickly. These tasks are very complex and require a large amount of computation to be completed. Creating specialized hardware would greatly reduce the time consumed by these processes. Also, the use of advanced techniques in image construction and decomposition would greatly increase the speed and effectiveness of the overall process. It is for this reason that an FPGA implementation of the curvelet transform is proposed and demonstrated.

It has long been known that the wavelet transform has many limitations when it comes to representing straight lines and edges in image processing. Not long ago, researchers at Stanford developed a solution to this problem [1, 4]. They created the curvelet transform, a transform that uses wavelets, but handles edges and lines much better.

As with every other application where large amounts of data are processed, speed is a very important issue. Modern processors are not designed for parallel algorithms or specialized mathematical operations. Digital signal processors (DSPs) could be used, but it has been shown numerous times in past research that the performance of DSPs can be beaten out by Field Programmable Gate Arrays (FPGAs).

The reprogrammability of FPGAs allows them to be used to implement any architecture a designer can develop. Another key advantage to FPGAs is their high-speed field reprogrammability. Just as one processor stores several pieces of software to do different tasks, an FPGA can keep several bitstreams and entirely change its hardware in a matter of milliseconds. In applications which have no need to take place concurrently (such as an image transform and inverse transform), there is no need to use several Application Specific Integrated Circuits (ASICs) when an FPGA could do multiple tasks in only one chip. This reduces costs and design effort with almost no loss of processing speed.

By implementing the curvelet transform as an FPGA coprocessor, these complex transforms could be done on large amounts of data faster than any current method. By using highly parallel hardware and the most advanced transforms, large amounts of image data can be processed in very small amounts of time.

2. MOTIVATION
2.1 Why use FPGAs?
One of the primary advantages to FPGAs is their reconfigurable architecture. This opens up a whole world of possibilities unavailable in microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), or any other chip with a specific architecture.

One of the most obvious possibilities is the reduction of the total number of chips (which reduces cost and board area). In many cases, it is useful to have a device that performs two different tasks. One example would be a device that implements several different transforms and their inverses. The hardwired
architecture of standard microprocessors and DSPs gives the
designer limited opportunities to change the way these things are
done. Obviously it is possible to store any number of transform
algorithms in memory and then run them on these processors, but
the hardware is unable to adapt to the different transforms, and
therefore the designer is forced to make due with what is given to
them.

One solution to this would be an ASIC. This would give the
designer the freedom to modify the hardware in the most optimal
way for each transform. Unfortunately, a separate ASIC would be
required for every transform that was desired. This is the
advantage of an FPGA. Several transforms can be designed and
stored in ROM on the board. A single FPGA can be
reprogrammed on the fly to perform any of these stored transforms when needed. Take the example of designing a board
that needs to perform five different transforms. One could design
a board with ten parallel DSPs or microprocessors that can be
used in any way to calculate the transforms. Instead, that same
area and money can be used for ten FPGAs to achieve an increase
in speed. A board could be filled with ten different ASICs (two
for each transform), which would give an even higher speed than
FPGAs. The advantage to FPGAs is that if the application needed
ten of one transform and none of the others, eight ASICs would
sit idle, while other data would be waiting. The FPGAs could be
reconfigured very quickly to handle this problem.

Another major advantage to FPGAs is the fact that they can be
configured as dedicated processors. That means that there is no
overhead for an operating system. This is a large time advantage.
The other speedup is in stages of the data path within the
processor. Many instructions do not need to go through each
stage of the data path. However, in order to keep the pipeline
running smoothly, they must sit in that stage and wait. With an
FPGA design, this is unnecessary since those other instructions
would not be included.

2.2 Improvement Over Current Work
One possible application of the curvelet in an FPGA would be in
the development of a hyperspectral imager. This device would
orbit the earth and take pictures of the sky over time. What would
then be left are three-dimensional pictures of stars that become
represented as lines (since the star moves over time). This data
then needs to be sent back to stations on the earth for processing.
The problem is that the slow communication system from space to
earth, does not allow this much data to be sent fast enough.
Obviously the best solution would be to compress this data.

This is where the curvelet transform becomes useful. The curvelet
is the best at compressing straight lines in images, so it would be
the optimum choice. The next problem is speed. There needs to
be a way to compress this data as fast as the camera can take
pictures. Other features that the hyperspectral imaging device
might need are navigation, power management and
communication control. These features will not constantly be
needed, so using dedicated processing power for them would be
somewhat wasteful. By having FPGAs in the system, they could
be reconfigured to control the device’s movement, power, etc.
when needed, or be used to do transforms when the device is
located in the proper spot. This means that the FPGAs that are in
the system to do image processing can also be used for totally
different purposes. This would be totally impossible if using an
ASIC. Since DSPs are designed specifically for signal
processing, tasks that require different capabilities (such as bus
communication or power electronics control) could not be done
by them, while an FPGA would be perfect.

There is some overhead in using an FPGA in a system. In order
for an FPGA to be reconfigured, the configuration must be stored
in memory chips in the system. Every time an FPGA is
reconfigured, it takes a certain amount of time to do this. This
time can range from 1.2 ms for the smallest Virtex chip (XCV50)
to 31ms for the largest Virtex-E (XCV3200E). For some FPGAs,
partial reconfiguration is also an option if the entire chip does not
need to be modified. This can happen in as little as 4 µs to change
a tiny fraction of the chip. FPGAs also consume more power than
most ASICs and are usually less dense then ASICs. These things
may point against FPGAs, but the advantages definitely outweigh
the disadvantages.

Another advantage of FPGAs is the ability to find a radhard
version. Radhardness is a very important quality for electronics
deVICES that will be used in space (as well as some other areas).
FPGAs are widely used as radhard devices, and it is also common
to see DSPs implemented in an FPGA in order to gain
radhardness. Radhard DSPs are not easy to find on the market
(something which would make their cost very high). FPGAs are
also known to consume less power than DSPs.

3. BACKGROUND
3.1 Radon Transform
3.1.1 Continuous Radon Transform
The continuous radon transform is defined by the following
formula [11]:

\[ R_f(\theta, t) = \sum \sum f(x_1, x_2) \delta(x_1 \cos \theta + x_2 \sin \theta - t) \]

In the digital domain, this is simply the addition of all (x,y)
coordinates that lie along a particular line defined by intercept
point t and slope \( \theta \). This obviously gives a very easy digital
implementation of the transform for digital images. Unfortunately, due to the natural continuous nature of the
transform, the obvious digital implementation is not easy to
invert.

3.1.2 Finite Radon Transform (FRAT)
Fortunately, a variation on the radon transform that works well in
finite spaces has been proposed [8]:

\[ r(k,l) = FRAT_f(k,l) = \frac{1}{\sqrt{P_{(i,j)kL_k,j}}} \sum f(i,j). \]

Where,

\[ L_{k,j} = \{ (i,j) : j = ki + l(\text{mod } p), \; i \in \{0,1,\ldots, p - 1\} \} \]

\[ L_{p,j} = \{ (l,j) : j \in Z_p \}. \]
The FRAT works under the same principle as the continuous transform, however it “wraps” the lines around the edges of the image to make sure that all lines are of equal length.

Its inversion, known as the finite back projection operator is given in similar fashion [8].

\[ FBP_x(i, j) = \frac{1}{\sqrt{p}} \sum_{(k,l) \in P_{i,j}} r(k,l) \]

Where,

\[ P_{i,j} = \{(k,l): l = j - ki (mod \, p), k \in \{0,1,..., p-1\}\} \]

The finite radon transform does require \( p \) to be prime. This means that all images must be square, and the number of pixels along each side must be a prime number.

### 3.2 Wavelet Transform

There are many variations of the wavelet transform each with its own strengths and weaknesses. For this application, the Haar wavelet was chosen [10], as it is fairly simple and appears to give good performance.

#### 3.2.1 Haar Wavelet

The Haar wavelet consists of replacing every two consecutive points with the average and detail coefficients (distance from the average). The process repeats until there is only one average coefficient and all the rest are detail. This can best be shown with an example.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Average</th>
<th>Detail</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[3 7 9 5]</td>
<td></td>
<td>[3 7 9 5]</td>
</tr>
<tr>
<td>1</td>
<td>[5 7]</td>
<td>[-2 2]</td>
<td>[5 7 -2 2]</td>
</tr>
<tr>
<td>2</td>
<td>[6]</td>
<td>[-1]</td>
<td>[6 -1 -2 2]</td>
</tr>
</tbody>
</table>

#### 3.2.2 Inverse Haar Wavelet

The inverse of the one dimensional Haar wavelet is just the above process in reverse. Beginning with the average and first detail coefficient, the two numbers are added to get the first value and subtracted to get the second value. This repeats until all detail coefficients have been transformed.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Average</th>
<th>Detail</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[3 7 9 5]</td>
<td></td>
<td>[3 7 9 5]</td>
</tr>
<tr>
<td>1</td>
<td>[5 7]</td>
<td>[-2 2]</td>
<td>[5 7 -2 2]</td>
</tr>
<tr>
<td>2</td>
<td>[3 7 9 5]</td>
<td></td>
<td>[3 7 9 5]</td>
</tr>
</tbody>
</table>

#### 3.2.3 The “á trous” Wavelet

This is a more complex version of the wavelet transform [6]. It still replaces each pair of pixels with an average and detail coefficient, but it uses a more complex filter. For this implementation, the Lagrange interpolation filter was chosen. It is given by:

\[ h = \frac{1}{2} * \frac{1}{\sqrt{2}} (1,1) * \frac{1}{\sqrt{2}} (1,1) = \left( \frac{1}{4}, \frac{1}{2}, \frac{1}{4} \right) \]

The “á trous” algorithm is then given by the following C code

```c
for(j=1; j<=MAX_LEVEL; j++) {
    for(l=0; l<YRANGE; l++) {
        for(k=0; k<XRANGE; k++) {
            c[j][k][l]=0;
            for(y=0; y<FILTER_DEGREE; y++) {
                for(x=0; x<FILTER_DEGREE; x++) {
                    c[j][k][l]+=c[j-1][(k-offx)%X][(l-offy)%Y]/filter_mask(x,y);
                }
            }
            w[j][k][l]=c[j-1][k][l]-c[j][k][l];
        }
    }
}
```

Where FILTER DEGREE is the number of coefficients in the filter, X and Y are the size of the image, and offx and offy are given by:

\[ offx = 2^j * (fx - FILTER_{\_\_\_DEGREE}/2) \]

#### 3.2.4 The Inverse “á trous” Wavelet

The inverse is fairly simple to implement. It is just the addition of the various images that are outputted from the original transform. It is given formally by:

\[ I(x,y) = c_j(x,y) + \sum_{j=1}^{j} w_j(x,y) \]

### 3.3 Ridgelet Transform

The ridgelet transform is simply the application of a 1 dimensional wavelet transform to the slices of the radon transform. The finite ridgelet transform (FRIT) is therefore the application of the FRAT to the entire image followed by the wavelet transform on each row [5]. The inverse FRIT is the entire...
operation in reverse. Apply the inverse wavelet transform to the rows, and then use the inverse FRAT on the entire image.

3.4 Curvelet Transform
The curvelet transform begins with the application of the á trous wavelet, and then is followed by the repeated application of the ridgelet transform. The algorithm is most simply stated as:

1. apply the á trous algorithm with J scales
2. set $B_1 = B_{\text{min}}$
3. for $j = 1, \ldots, J$ do,
   a. partition the subband with a block size $B_j$ and apply the ridgelet transform
   b. if $j$ modulo 2 = 1 then $B_{j+1} = 2B_j$
   c. else $B_{j+1} = B_j$

4. RELATED WORK
Due to the relative newness of the ridgelet and curvelet transforms, there have been very few implementations of each [2, 3, 9]. The majority of the published work is from those who originally developed the transforms. All of these implementations have been done in software only. Most of them were Matlab implementations of the equations themselves. Both transforms were used several times to demonstrate their various abilities such as denoising and compression. To our knowledge, this is the only hardware implementation of the radon and ridgelet transforms as well as their inverses.

The wavelet transform has been implemented rather exhaustively. It has been done in hardware, software and every combination of the two. Because of this fact, implementing the wavelet transform will be the smallest part of this work. The radon transform has many software implementations, and even a one-sided hardware implementation, but we have not seen any complete implementations in any form of hardware.

The codesign methodology (the combination of hardware and software) is becoming increasingly more popular. Its use often involves FPGAs, and it has helped to show the superior performance that FPGAs can provide. There have been a few instances of using codesign techniques in image processing [7, 12]. These focus primarily on the wavelet transform. We have never seen any attempts at using codesign for radon, ridgelet or curvelet transforms.

5. RIDGELET ARCHITECTURE 1
This architecture focuses on parallelism. The time of the entire transform is equal to the time it takes to place the image onto the FPGA plus the time it takes to read it back. There is no time necessary for computation since this is done during the writing.

5.1 Generic Transform
In many transforms, each output pixel is simply the addition of a certain set of input pixels. Examples of include the Radon, Inverse Radon, and Hough transforms. Because of this, a generic architecture was developed that would take in a square image and return an identical sized image with the output pixels equaling additive combinations of the input pixels based on look up tables. These look up tables could then be loaded with the proper values for whichever transform was desired.
5.3 Inverse Wavelet
The inverse wavelet transform is the same as the above architecture, but reversed. The addition and subtraction are there, but the last bit is included. The hierarchy goes from smallest to largest as shown below.

![Inverse Wavelet Transform Architecture](image)

5.4 Ridgelet
Once the radon and wavelet transforms have been implemented, the ridgelet is straightforward. Each output row of the radon is simply passed through the wavelet transform before it reaches the final output multiplexer. The primary advantage to this method is that it does not require any extra clock cycles to perform the ridgelet than it does to perform the radon.

![Ridgelet Transform Architecture](image)

5.5 Inverse Ridgelet
The inverse ridgelet requires that the inverse wavelet happen before the inverse radon. In this case, each input row is passed through the inverse wavelet before it reaches the inverse radon. Since pixels enter one at a time, there is a need for registers to store an entire row before the inverse wavelet takes place. Unfortunately, this adds extra clock cycles (equal to the number of pixels in a row).

![Inverse Ridgelet Transform Architecture](image)

6. RIDGELET ARCHITECTURE 2
This architecture focuses on flexibility. Instead of the massive parallelism in architecture 1, a more compact and iterative approach is used. Each function is broken into its own module and a primary control block directs the flow of data. This allows for a higher clock rate, and blocks can be rearranged or added to increase parallelism as desired.

6.1 Read and Load Blocks
This architecture is based on the Block RAM inside the FPGA. The image is loaded into the RAM, and then processed. When the transform is complete, the image is then returned to the host PC. In order for this to work, modules had to be developed to load data into the Block RAM and read the data from the Block RAM.

The read and load modules are fairly straightforward. The read module, pictured in Figure 6, consists of a block that connects the output of the RAM to the parallel port as well as providing some control signals to the RAM. A tri-state buffer is used to control which module has access to the RAM's data and control busses.

![RAM Reading Module](image)

The load module is very similar. As can be seen in Figure 7, the only difference in the read and load modules is that the load module connects to the input data of the RAM instead of the output data (for obvious reasons). Another tri-state buffer is used, since all RAM inputs must have these to avoid signal contention.

![RAM Loading Module](image)

6.2 Radon Transform
The radon transform in this architecture is noticeably different than the one in the first. The radon transform block uses a block called “frat calculator” to generate the list of which points in the
input image affect which points in the output image. In the first
architecture this was stored on chip (which takes quite a bit of
storage space). After the point list is calculated, “address
generator” converts the pixel values into RAM locations and
switches the RAM address input. The accumulator is used to add
the entire group of pixels chosen by "frat calculator". The local
control block organizes the flow of this process with input from
the main controller. The diagram can be seen in Figure 8.

6.3 Wavelet Transform
The wavelet transform in this design has much in common with
that of the previous architecture. The usage of Block RAM makes
an obvious difference, but the values are still read into a block of
registers, transformed and then fed into a second block of registers
so that they can be transferred into RAM again. Just as in the first
architecture, this process is pipelined to increase speed. One
change to the Haar transform block itself is the addition of a clock
to aid pipelining. The flow is demonstrated in Figure 9.

6.4 Control Block
Now that all the pieces have been described, the overall
combination can be explained. Data comes in from the PC’s
parallel port. At this point it goes through the RAM loader and
into the RAM. The control block decides how much data it will
accept and uses the RAM loader to accomplish this. Next, the
control block runs the radon transform block, and then the Haar
transform (or their inverses). At the end, the data is transferred
back to the parallel port through the RAM reader.

The advantage of this design is the ease of adding more modules. To
double the speed of the radon transform, another radon block
can be added with no other changes (except making the control
block aware of the new addition). With the exception of the RAM
loader and reader (which are limited by the size of the FPGA
input bus), all modules can be duplicated to increase speed at the
cost of area. The design (with only one of each module) is
pictured in Figure 10.

7. RESULTS

7.1 Test Bench
7.1.1 Hardware Platform
In order to test the ridgelet architecture, it was programmed on a
Xilinx XCV1000E FPGA. The input image was sized at 17 x 17
to fill the prime number requirement of the FRAT. The FPGA
was connected to a PC through a parallel port. Since the parallel
port is limited to sending eight data bits and receiving four, a
separate control block was added in the FPGA to convert the
sixteen bits used in the transform to and from the size handled by
the parallel port. The FPGA can be quickly reprogrammed to
switch between the ridgelet transform and its inverse.

7.1.2 Software Platform
The PC used was a Dual AMD Athlon MP 1800 running at 1.5
GHz. A C program running on this PC reads in a 289 x 289
image file (in PNM format) and runs the “à trous” decomposition
on it. Next, it sends this image through the parallel port with the
proper control signals for the FPGA. It then reads the image back
and stores it in another PNM file. The PC side is identical for the
ridgelet transform and its inverse.

7.2 Circuit Performance

7.2.1 Ridgelet Architecture 1
The parallel port described in the design is just for the test setup.
The speed of that port is much lower than the speed of the FPGA.
The entire ridgelet transform takes approximately 1.6 seconds
with the parallel port. In a real implementation, something similar
to a PCI bus would be a much better communication system. The
Xilinx synthesis tools do give the performance of any compiled
design. Based on those numbers, we can determine the speed of
the transform when the PCI (or some other high speed bus) is
used. The FRIT architecture would take 289 x 289 send cycles
plus another 289 x 306 for reading. This would all be at a clock
rate of 33MHz. The IFRIT would have (306 + 17) x 289 send
cycles and 289 x 289 for the read. The maximum clock speed for the inverse is 18MHz.

By multiplying these numbers out, the total time for the FRIT is 5.2ms. Total for the inverse FRIT is 9.8ms. This means the entire process takes 15ms. For a comparison, this same process took 1.5s on a dual Athlon MP 1800. That gives a speed increase factor of 100.

Figure 11 shows the completion time of the first ridgelet architecture for various image sizes. The y-axis shows the amount of time (in milliseconds) for the entire transform to take place. The x-axis shows the number of pixels in the image. The graph shows the results for both the finite radon transform (FRIT) and its inverse (IFRIT). The maximum size of pixels that can be processed in 33 ms (30 frames per second) in the case of FRIT is 534285. For the IFRIT it is 283333. For reference, a short list of common image sizes and the number of pixels in them follows.

<table>
<thead>
<tr>
<th>Image Size</th>
<th># of Pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>640 x 480</td>
<td>307200</td>
</tr>
<tr>
<td>800 x 600</td>
<td>480000</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>786432</td>
</tr>
</tbody>
</table>

Table 3. Common Image Sizes and Number of Pixels

The second architecture has several more cycles, but the clock rate is higher. For the 289 x 289 image, there are 17 x 18 x 289 read and 17 x 18 x 289 write cycles. For the entire radon transform (or its inverse), 17² x 18 x 289 clock cycles are necessary. For the Haar transform, an additional (17 + 4) x 17 x 289 clock cycles are needed. This architecture is almost six times as slow as the first forward architecture and three times as slow as the first inverse architecture. However, this architecture is one-eighth the size of the first ridgelet architecture. Another advantage is the fact that some blocks can be repeated to increase speed. Due to the large amount of clock cycles used by the radon transform relative to the Haar, it turns out not to be very useful to increase the number of Haar blocks, but increasing the number of radon blocks has a noticeable effect (up to a point).

The reconfigurability advantages of the second ridgelet architecture are shown in Figure 12. More blocks of the radon or Haar type can be added to increase the speed of the transform at the cost of area. The y-axis shows the time (in milliseconds) that it takes to complete the entire FRIT or IFRIT. The x-axis shows the number of radon or Haar blocks used in the FPGA. The lines with darkened shapes represent an architecture with one radon block and the number of Haar blocks shown on the x-axis. The lines with white shapes represent an architecture with one Haar block and the number of radon blocks shown on the x-axis.

Table 4 shows the minimum amount of each of the four types of blocks needed to reach video quality imaging (30 frames/sec), for each of the given image sizes. The last column shows the amount of slices in the FPGA what would be taken up by that particular design.

Table 4. Video Quality Area Results for Ridgelet 2

<table>
<thead>
<tr>
<th></th>
<th>radon blocks</th>
<th>Haar blocks</th>
<th>load RAM</th>
<th>read RAM</th>
<th>total slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>289 x 289</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>828</td>
</tr>
<tr>
<td>800 x 600</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1968</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2736</td>
</tr>
</tbody>
</table>

Figure 12. Speed Data for Second Ridgelet

It can also be shown that the addition of radon blocks has a fairly small effect on the overall area taken within the FPGA. The addition of Haar blocks has a much larger effect. Combined with the previous speed results, this works out very well since the radon is the block that should be added anyway. The results of this experiment for all image sizes are shown in Figure 13.
From Figure 12, it can be seen that as more blocks are added, the time decreases asymptotically to a certain point. This point varies with the number of FPGAs used. The more FPGAs used, the lower the asymptotical point would become. This is shown in Figure 14. The y-axis shows the asymptotical minimum time, and the x-axis shows the number of FPGAs. The lines are the same as Figure 12. It can be seen that two FPGAs are twice as fast as one, three FPGAs are thrice as fast as one, etc.

![Figure 14. Multiple FPGAs with Second Ridgelet](image)

7.2.3 Software
The part of the curvelet transform that was implemented in software was the “á trous” algorithm. On the dual Athlon machine, the transform took 770ms, and its inverse took 40ms.

7.3 Visual Performance
A fingerprint image was used to demonstrate the curvelet transform. Figure 15 shows the original image and its three subbands after the “á trous” decomposition.

![Figure 15. A Fingerprint Image and its Subbands](image)

Since the FRIT produces no visual result, the images after the FRIT are not shown. Figure 16 shows the subbands after the FRIT and then subsequent IFRIT. It then shows the three transformed and inverse transformed subbands recombined into one image by the inverse “á trous”.

![Figure 16. Subbands After FRIT and IFRIT Recombined](image)

7.4 Energy Compression
One of the primary purposes of the ridgelet transform is compression. Using the curvelet transform creates more images, so compression in itself is not useful, but the ridgelet compression can be used for denoising [9] among other things. Because of this, several images were run through the curvelet transform to see what compression the transform offered.

To begin with, a measurement of image size must be defined. Especially in denoising, it is useful to save the large pixel values, but remove the small values (this process is known as thresholding). For this reason, we will define a measurement called the “energy” of the image. Energy will be the sum of all the pixel values in the image. The term “X percent of the energy of the image” is the minimum number of pixels required to sum to X percent of the total energy.

Eight images were chosen for this experiment. 80 percent of the image’s energy was determined before transform. Next, the curvelet transform was performed leaving three separate transformed images. 80 percent of the energy of each of these images was determined, and the three values were averaged to determine compression. In all cases except one, the energy was significantly compressed in the image (compressed meaning put into less total pixel values). Table 5 shows the results of this experiment.
<table>
<thead>
<tr>
<th>Image</th>
<th>Original</th>
<th>Transformed</th>
<th>Size Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>brain</td>
<td>32546</td>
<td>28368</td>
<td>14.7%</td>
</tr>
<tr>
<td>cartoon</td>
<td>38635</td>
<td>40305</td>
<td>-4.1%</td>
</tr>
<tr>
<td>circle</td>
<td>23696</td>
<td>10280</td>
<td>130.5%</td>
</tr>
<tr>
<td>drawing</td>
<td>64951</td>
<td>33613</td>
<td>93.2%</td>
</tr>
<tr>
<td>fingerprint</td>
<td>50275</td>
<td>38097</td>
<td>32.0%</td>
</tr>
<tr>
<td>grid</td>
<td>73441</td>
<td>33889</td>
<td>116.7%</td>
</tr>
<tr>
<td>mac</td>
<td>46667</td>
<td>30202</td>
<td>54.5%</td>
</tr>
<tr>
<td>painting</td>
<td>60789</td>
<td>40793</td>
<td>49.0%</td>
</tr>
</tbody>
</table>

8. CONCLUSIONS AND FUTURE WORK

Based on the results presented in this thesis it is quite obvious that the FPGA is far more efficient (in speed and area) at implementing the various transforms. It is also clear that the curvelet transform is one of the most powerful methods of processing images. The use of these transforms is shown to make a powerful tool that can be used to accomplish tasks that have previously been unimplementable.

There are several future possibilities that can be based on this work. One improvement could involve replacing the Haar wavelet with one more suited to certain types of images. Another could involve replacing the finite radon transform with the digital radon transform involving the fast Fourier transform and its inverse.

Besides improvements to the design, other developments can be built on top of this. The primary ones are the hyperspectral imager and equation solver mentioned in the introduction. Both of these devices are things that could never have been done or were extremely costly in money, human time and processing time. With the new techniques discussed in this thesis the possibility of creating these systems and similar ones can be seen.

9. REFERENCES


