APPLYING MULTIMEDIA COMPRESSION TECHNIQUES TO GROUND PENETRATING RADAR DATA

Michael Jeffords*
M.S. Program in Computer Science, Rivier College

Abstract

Raw uncompressed Ground Penetrating Radar (GPR) data is approaching Tb sizes for applications that could be tied to Geographical Information Systems (GIS). As more and more requests are made to tie this information to GIS systems an efficient method of compression for network transfer will be necessary to provide it to end users. GPR data is typically highly oversampled and therefore should be a good candidate for compression. The oversampling is needed during collection as you can easily adjust data during processing to recover information that may otherwise be hidden in the noise. However, it is the premise of this research that after removing (or separately transferring) constant and periodic background information and applying gain to overcome attenuation the remaining information will be readily compressible using a Fourier series without any significant data loss. Additionally, an attempt will be made to determine the feasibility of progressively transferring increasing resolution by transferring the Fourier series in groups of most important to least important frequency components.

1. Introduction

Fourier analysis in various forms is used throughout many disciplines (signal processing, networking, audio, video, image processing, engineering, etc.). Google "Fourier related transforms" and you will find numerous transforms that allow you to mathematically model and convert information between the sampled domain and the frequency domain. Many compression techniques rely on this concept.

Basic Premise

The basic premise is that any signal can be represented by a series of sine waves of varying frequencies, amplitudes, and phases. Paul Falstad created an excellent interactive demonstration can be found at http://www.falstad.com/fourier/ [1]. You can visually see how the differing frequency components are added to create various signals.

It works on Anything

Fourier based transforms work regardless of the actual frequency of the signals (as long as the highest frequency being modeled is less than ½ the sampling frequency). The Fourier series uses multiples of the fundamental frequency and adds them together to model the signal. We care nothing of where the data comes from just that it is sampled in even intervals in time or space. The Nyquist-Shannon sampling theorem indicates that the highest frequency component in a sampled signal that can be reconstructed is less than ½ the number of samples taken per second. [2]

$$\text{MaxFrequency} < \tfrac{1}{2} * (\text{SamplesPerSecond})$$

ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

Oversampling Explained

Let's say that we have a signal which we know should contain information between 30Hz and 80Hz but we have enough samples to define a range of 10Hz to 500Hz.

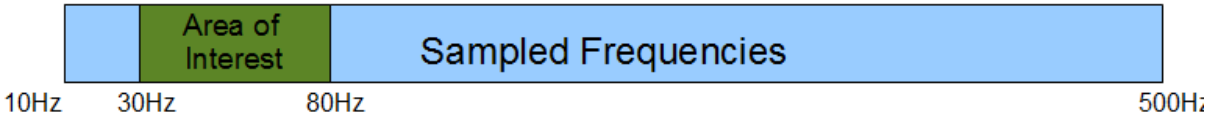

Figure 1: Oversampling Data

The Area of Interest in Figure 1 is much smaller than what we obtained samples for. In a perfect world the data oversampling makes the data highly redundant. However, in the presence of noise this oversampling helps you extract the area of interest. And yet, once you extract the information you want the extra information is wasted space.

Oversampling gives more information for the user to process visually. This visual benefit of oversampling can still be achieved by oversampling, compressing and then decompressing back to the original number of samples.

GPR example

Ground Penetrating Radar pushes ultra wide band signals (many frequencies) into solid objects. The signal reflects back to a nearby receiver whenever the dielectric properties of the material changes. The change could be a void, a bedrock layer, rebar in concrete, pipes in the ground etc. Due to the types of materials encountered the signal has a tendency to attenuate very quickly. Additionally, since the signal attenuates quickly noise plays a significant role. Therefore in GPR oversampling is commonly used to obtain extra information from which to extract the area of interest. Often we obtain between 256 to 1024 samples in ranges of 10 ns to 200 ns. Often the maximum frequency recorded is approximately 25GHz but the frequency range of the antenna might be from 500MHz to 3GHz (see Fig. 2).

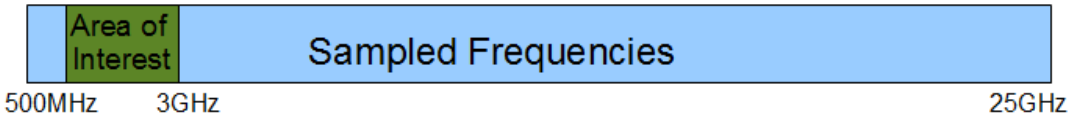

Figure 2: 2GHz Center Frequency Antenna with common parameters

Because Fourier related transforms are used in so many fields, they are common and their computational speed has been optimized. Many of the uses deal with images, audio and streaming media. It becomes practical to take the following general approach to combine filtering and compression:

- Encoding: allow the oversampled data to contribute to the frequency domain
- Compressing: remove the frequency components outside the area of interest
- Transmission: store/transmit the remaining component frequencies
- Decoding: reconstruct the signal at the end user location from the frequency components.

2. Problem Definition and Requirements

GPR data has evolved over the years and many advances have been made in

- number of channels – 2, 4, 8, 16 channel systems are now common.
- acquisition speeds - 4 channel data can be collected at speeds over 900scans/second
- scan density – 4 channel data can be collected at 512, 1024, 2048 and higher samples/scan
- bit depth – systems now collect 32 bit samples.

This means that systems can now collect 1Gb of data in 5 minutes. It is likely that 20Gb and larger files will be commonplace. Up through 2010 file sizes typically did not exceed 1Gb and often would have been comprised of many smaller profiles collected over a day or more.

Compression will be vital for seamless end user experiences. The largest files will be generated to monitor road and railway health by non-destructively testing pavement thicknesses and other conditions. These data sets will likely reach Terrabyte levels as entire interstates or road ways within municipalities are traversed. A rough estimate shows that 8 hours of data collection could produce 90Gb of data. This data will need to be integrated with GIS systems in various forms.

Using lossless techniques such as zip and Huffman encoding to get up to a 2:1 compression ratio and take significant amounts of time. For instance, on an Intel Core i5 processor with solid state drives, it took 3 hours to zip an 83Gb GPR data file and it only compressed the file to 60% of its original size. Zip techniques appear to processing the entire file followed by encoding the entire file. For efficiency we need to be able to compress/decompress scan by scan.

Frequency(MHz)				Oversampled X times At Samples/Scan		
Center	Low	High	Range(ns)	256	512	1024
200	50	340	200	2	4	8
400	100	680	50	4	8	15
1000	250	1700	20	4	8	15
1600	400	2720	10	5	9	19
2600	650	4420	8	4	7	14
1000	250	1700	20	4	8	15
2000	500	3400	12	3	6	13

Figure 3: Amount of Oversampling on typical GPR data

Since the data is oversampled we can use a lossy compression depending on what information is lost. The noise that is outside our area of interest is completely unnecessary and is commonly filtered out when processing data while keeping the same number of samples. Thus we are already using lossy techniques intentionally to filter out noise. Figure 3 shows that we might be able to achieve between 2:1 and 19:1 compression by storing our data in the frequency domain.

Raw 32 bit GPR data is really constrained to 24 bits with 8 bits of headroom to prevent clipping of data and to allow the stacking of up to 256 scans. Although 8 bit data has proven to be inadequate for many processes, once data is collected 24/32 bit data can be reduced to a 12 to 16 bit depth with minimal effect on processing operations. Storing the information as signed shorts would also achieve a 2:1 compression. This would theoretically allow for 4:1 to 38:1 compression based on Figure 3. For the road and rail applications a 2000MHz antenna would typically have a 12:1 compression. That would

reduce data collection from 90Gb per day to approximately 7.5Gb of data collection per day. The compressed data could then be transferred through wireless technologies back to the home office during idle times.

3. Finding the Right Transform for the Job

GPR data is often converted to images in order to allow the user to interpret the data visually. Often users convert these images to JPEG formats through various means. Although JPEGs have an excellent compression ratio with an ability to select varying levels of compression there are drawbacks when it comes to GPR data:

- Within the standard for JPEG compression the bit depth is dropped to 8 bits for intensity which in our case can be related to amplitude. Previously we discussed the need for a minimum of 12 to 16 bits for advanced GPR processing.
- It performs a 2 dimensional Discrete Cosine Transform (DCT) on 8x8 pixel blocks which means that
  - Our data would no longer be stored in logical units representing individual scans.
  - Every 8 samples vertically would have a boundary artifact as the DCT does not guarantee continuity between the end sample of one block and the start sample of a new block.
- Even though JPEG is common I find it unlikely that 5Gb or greater JPEG files would be easily viewable by other programs yet end users would expect it to work.
- Directly using JPEG would cause difficulties in the saving of header information as GPR data has very different needs than JPEG.

Thus using JPEG has drawbacks which result in the necessity for modifying the implementation.

Curvelet and Wavelet Transforms

The Curvelet transform is a relatively new type of transform used for recognizing curved features in 2 and 3 dimensional data. It is similar to Wavelet transforms with an attempt to overcome some of the deficiencies of wavelets. Although both are related to Fourier transforms they are more geared toward computer vision and reconstruction of features in sparse data [3]. Since they are both geared toward 2 or more dimensions their use is outside the scope of this research.

Any continuous Fourier based transform has the disadvantage of adding artifacts at the boundary conditions (start and end of scan) since a periodic function must wrap (start and end value must be equal). It will additionally result in extra frequency components to fit the data at boundary conditions [4].

For these reasons we decided to examine discrete Fourier Transforms:

- FFT – Fast Fourier Transforms – typically the basis of all the other transforms. Most transforms are implemented using an FFT of some sort.
- DFT – Discrete Fourier Transform – This transform can be computed directly from the sample values and is considered periodic for the number of samples given as input. The drawback is that it needs to be periodic.
- DST and DCT – Discrete Sine Transform and Discrete Cosine Transforms – These are very similar to each other but are different in the symmetry that they exhibit. They are based on the DFT. [5]

4. Reasons for Selecting DCT Type II transform

We selected the Discrete Cosine Transform DCT II for the following reasons:

- DCT only uses real numbers – GPR amplitudes are real numbers.
- DCT does not require end sample and start sample to be matching values and thus results in minimal artifacts at the boundary conditions. DST and DFT would introduce artifacts at the boundaries. This is overcome by the DCT; note how the first red dot and last red dot are at different amplitudes in the DCT (see Fig. 4). [4]

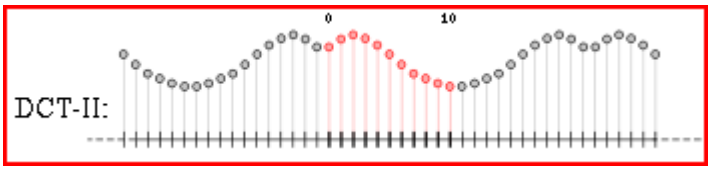

Figure 4: Symmetry of DCT-II [4]

- DCTs of Type II and Type IV are the transforms used in most audio and image compression schemes including JPEG, MPEG, MP3, and Vorbis. It is used because cosines are efficient at compressing the important frequency components into a small cluster at the low frequency end of the spectrum. The DFT on the other hand tends to place components at both ends of the frequency spectrum. [4]
- Due to its common use many open source implementations exist for both Java and C/C++. The package selected for this project is called JTransforms version 2.3 (a multithreaded multicore Java package by Piotr Wendykier) [6]. Its computational speeds are on the same order of magnitude as FFTW a C based library with JTransforms outperforming FFTW in some instances and FFTW outperforming JTransforms in others. [6][7]

5. Examining the JPEG compression algorithm

Since JPEG/MPEG uses the DCT in its codec, reviewing the overall method used to transform an image can teach us a lot about our own implementation. Please remember that JPEG alone does not satisfy the needs of GPR data compression for reasons outlined previously.

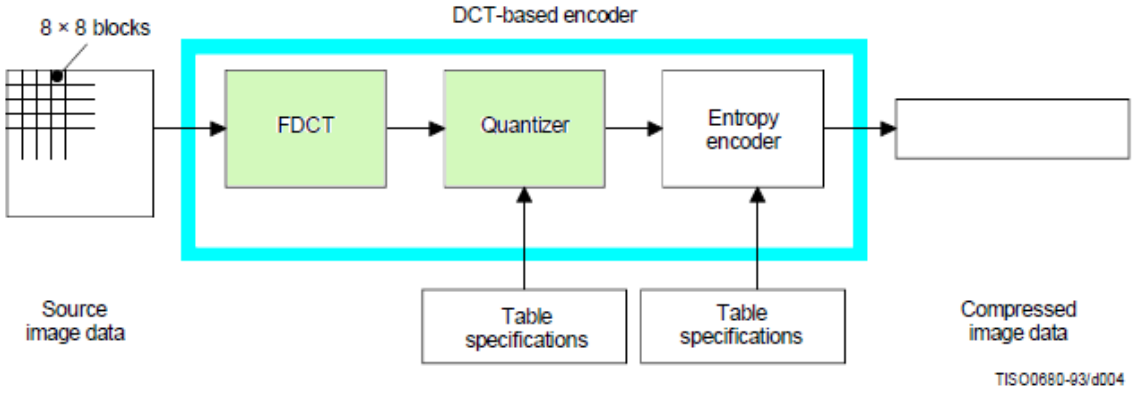

Figure 5: JPEG Standard – DCT based encoder Simplified Diagram [8]

JPEG-JFIF encoding consists of the following steps with steps similar to GPR needs bolded.

1. RGB information is transferred to YCbCr where Y is intensity. – This is unnecessary for GPR since the data only contains real amplitudes.
2. Color/Chroma information CbCr is reduced – again unnecessary for GPR.
3. Pixels are broken out into 8x8 arrays and a two dimensional DCT is performed after centering the amplitudes around zero. To center the values around zero 128 is subtracted from every amplitude. Y Cb and Cr components are all calculated separately. With GPR, to keep the basic building block of a scan a 1 dimensional DCT should be performed.
4. Amplitudes of the DCT are quantized to help maintain precision and the resulting matrix has the data clustered in the upper left of the 2d array (see Fig. 5). At this point the high frequency data toward the bottom right of the array may be discarded to increase compression. With GPR data the data will be quantized to preserve as much information from the float or double values within a 16 signed short.
5. The remaining data is put into a pattern that clusters the data around toward the beginning of a 1 dimensional array and then performs entropy coding such as Huffman encoding. With GPR data the Huffman encoding does not appear to be as useful since there is greater variation in codes due to the 16 bit nature. The noise outside the area of interest can just as easily be truncated from the saved frequency scan. [8][9]

6. Modeling an Encoder for GPR data compression

GPR Data will and can be encoded and compressed as follows (see Figure 6):

1. Scan pre-processing: “Compression works best if the source signals are a very high quality.”[10] When transferring the scan into an arrray of 256, 512 or 1024 scans perform the following preprocessing steps.
    - In one pass through each scan (an array of samples) -
        - Obtain Min Max ScanWidth and average amplitude values for the top and bottom of the scan.
        - Calculate the slope step and starting offset required to center the scan at 0.

- In the second pass through the array –
  - Subtract offset and slope step on a sample by sample basis
  - Apply gain and convert to Float in the range of (-128.000 to 127.000) on a sample by sample basis

2. Perform forward DCT: This will create a set of frequency values. See the red line in Figure 7.
3. Convert from float to short after multiplying all values by 64. This preserves 6 bits of resolution from this data which would otherwise be approximately 9 bits if converted directly. This gives 15 bits of resolution in the short. It meets the requirements from earlier.
4. Store/Transmit the area of interest only. See the next section for examples.

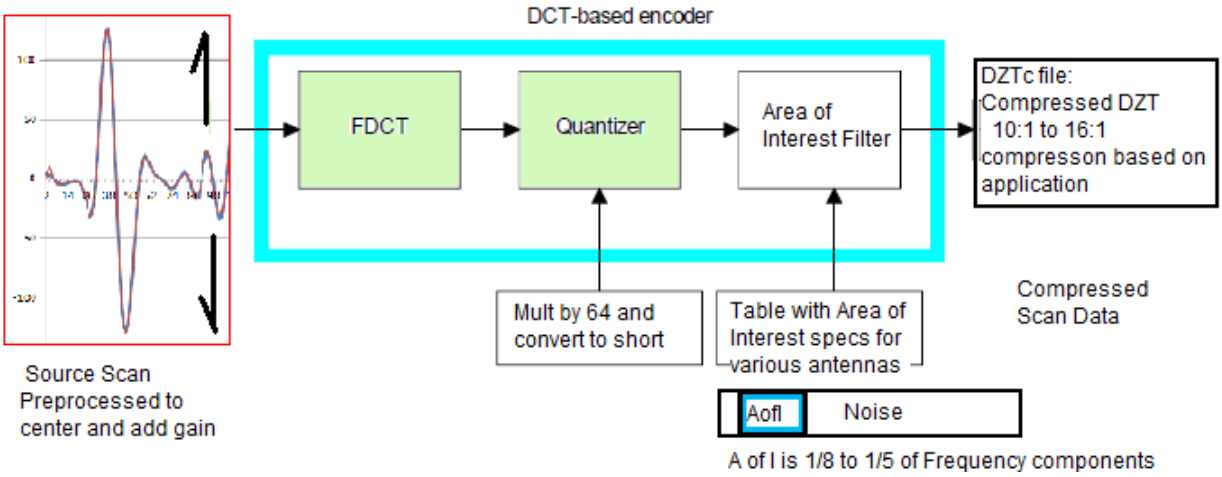

Figure 6: DCT based encoder for GPR data

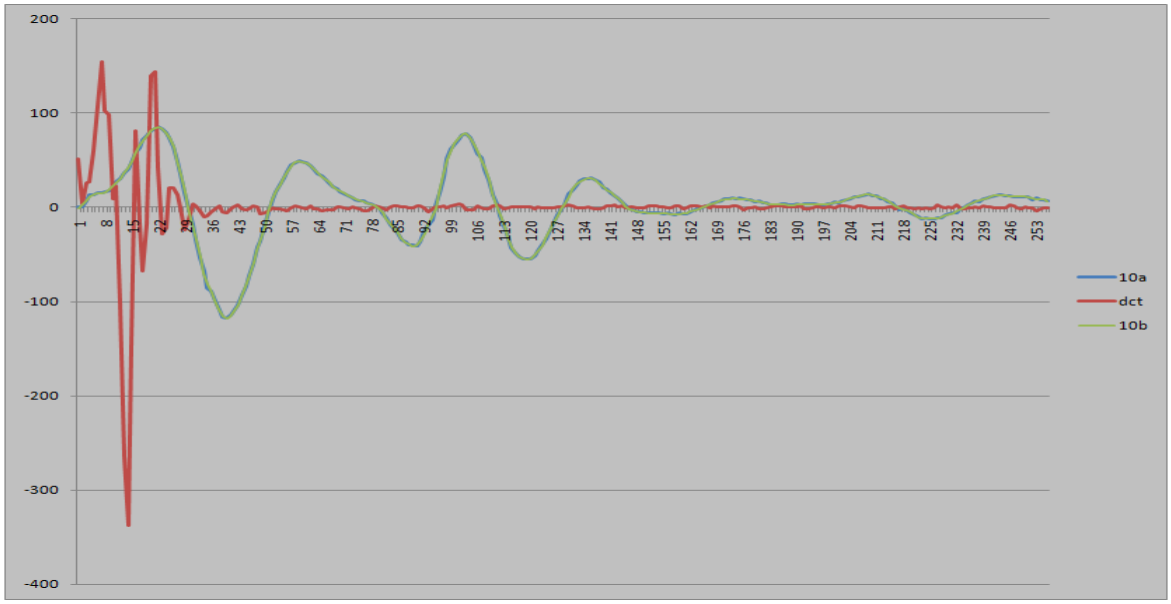

Figure 7: GPR Signal (Blue); Its resulting DCT (Red); and Reconstructed Signal (Green)

Note: The Blue and Green lines are on top of each other making it nearly impossible to distinguish between them. This signal was processed without adding gain which spreads out the components.

7. Processing a Sample

The following samples (Figs. 8-11) have been processed with the proposed encoding method. The software was written in Java and includes the compression stages mentioned. The series of images shows the effects of different levels of compression. When you cut into the area of interest and remove important frequency components the effect is blurred data with considerable aliasing artifacts.

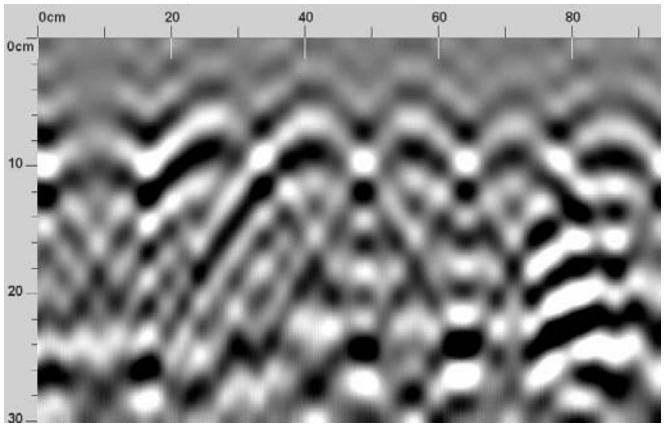

Figure 8: 19 Frequency Components
27:1 compression

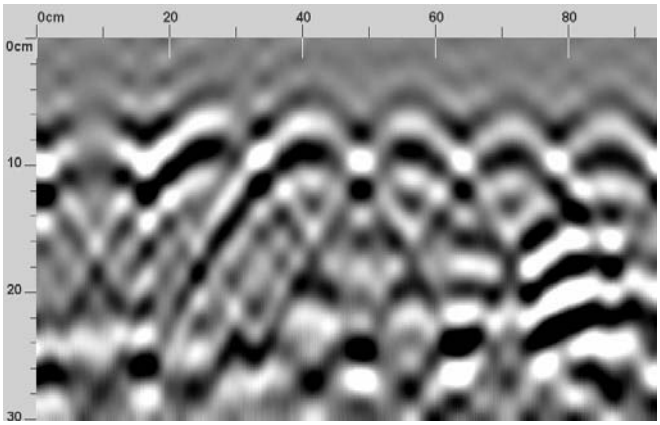

Figure 9: 21 Frequency Components
24:1 compression

Significant components are missing and changes are fairly obvious at the top.

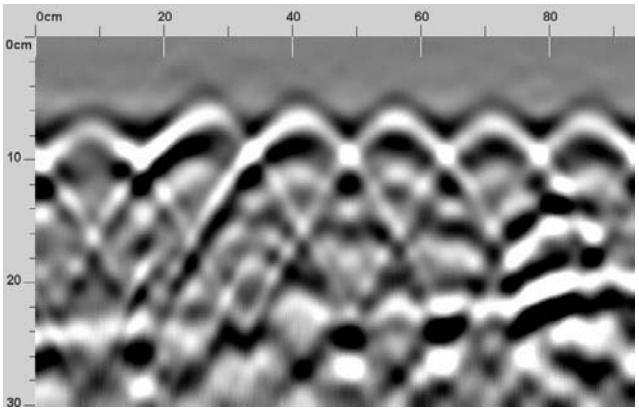

Figure 10: 33 Frequency Components
16:1 compression

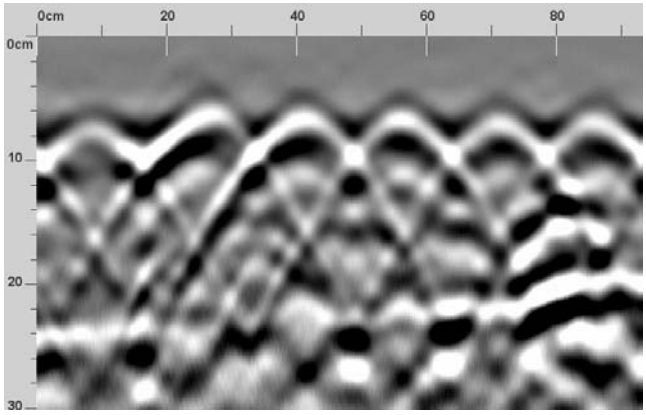

Figure 11: 51 Frequency Components
10:1 compression

Area above wire mesh is now quiet and there is no visually perceivable difference

In the data above you can see a process of refining the images. The visual differences between 16:1 compression and 10:1 compression are not recognizable. For most applications the 16:1 compression would be sufficient. We could for instance store 51 frequency components to ensure future processing capabilities but in order to speed the process of transmission for clients whose only purpose is viewing the data we could send only 33 components per scan for an additional 1/3 bandwidth savings.

The following image Figures 12 shows the same data set at a scale which will allow you to focus on the high frequency noise.

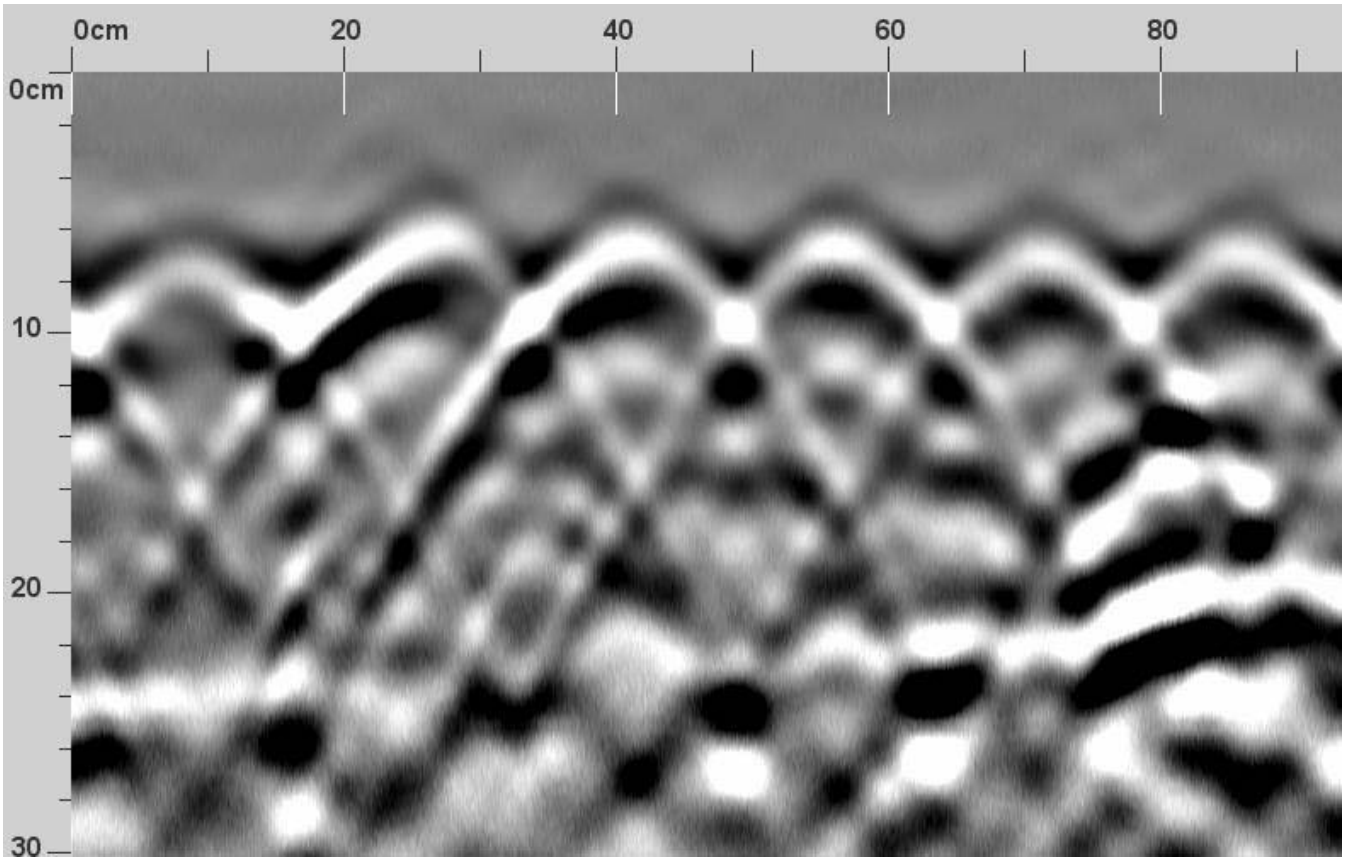

Figure 12: 128 Frequency Components 4:1 compression – *Notice the High frequency noise*

Note: As you look at this sample compare the Noise Level between this image and the previous images.

8. Original versus Reconstructed Signals

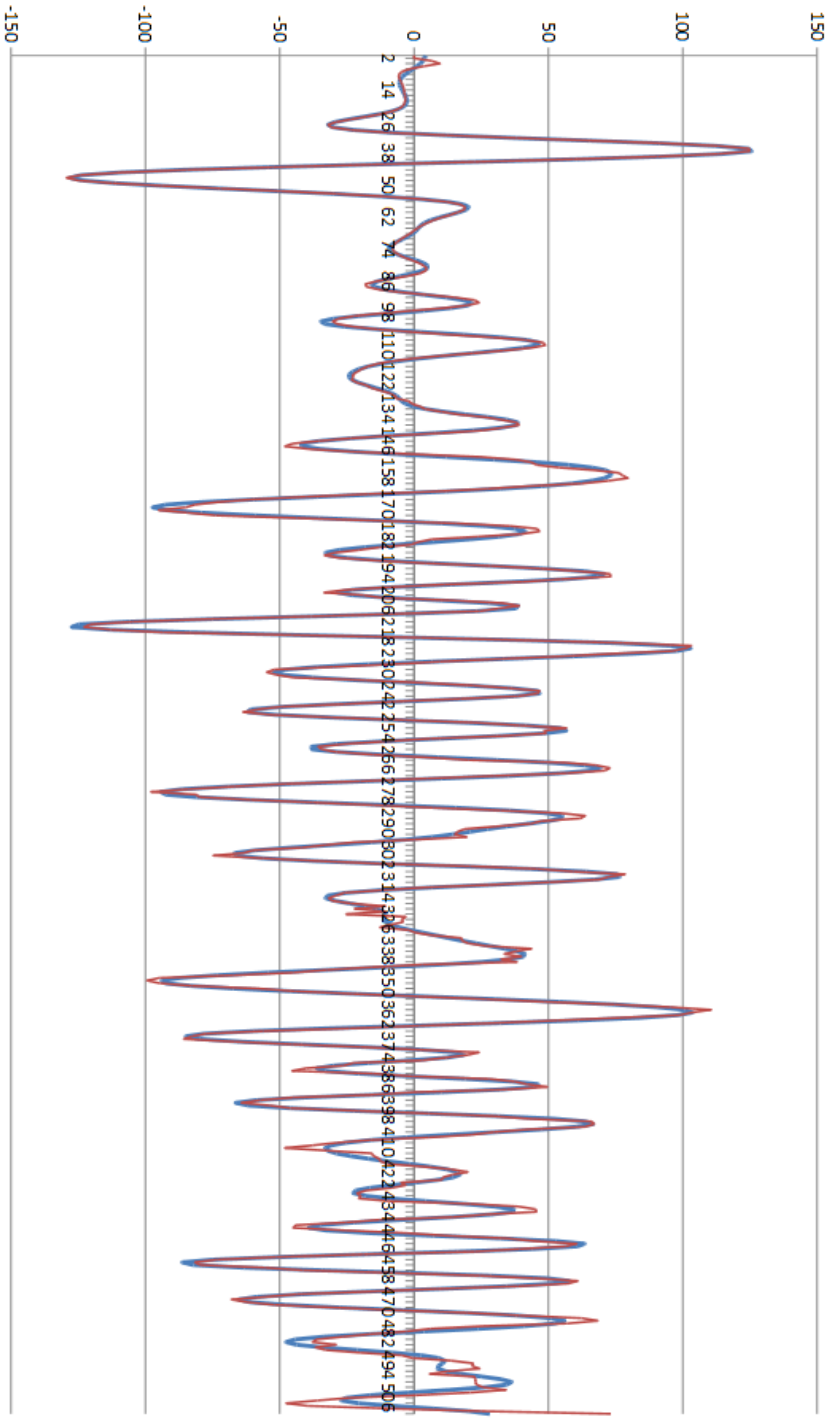

Figure 13: Original gained signal (Red) vs 10:1 compressed Signal (Blue)

Note: In the image above you can see the effect of noise in the data. The original red signal was gained to compensate for attenuation. The data at the bottom half has more obvious noise as the original noise was gained as well.

9. Comparison of Absolute Error between Original Oversampled Data and Reconstructed Signals

The following images (Fig. 14) were created from the absolute difference between the same scan in its original form and its reconstructed versions. (Original → DCT → IDCT → Reconstructed). This data set is the same as shown in Figure 13 which shows a 512 sample scan. In the top image we used signed shorts to store the data and in the lower image we used floats with higher precision. This tests whether or not lowering the bit depth from 24/32 to 15/16 bits has any real impact.

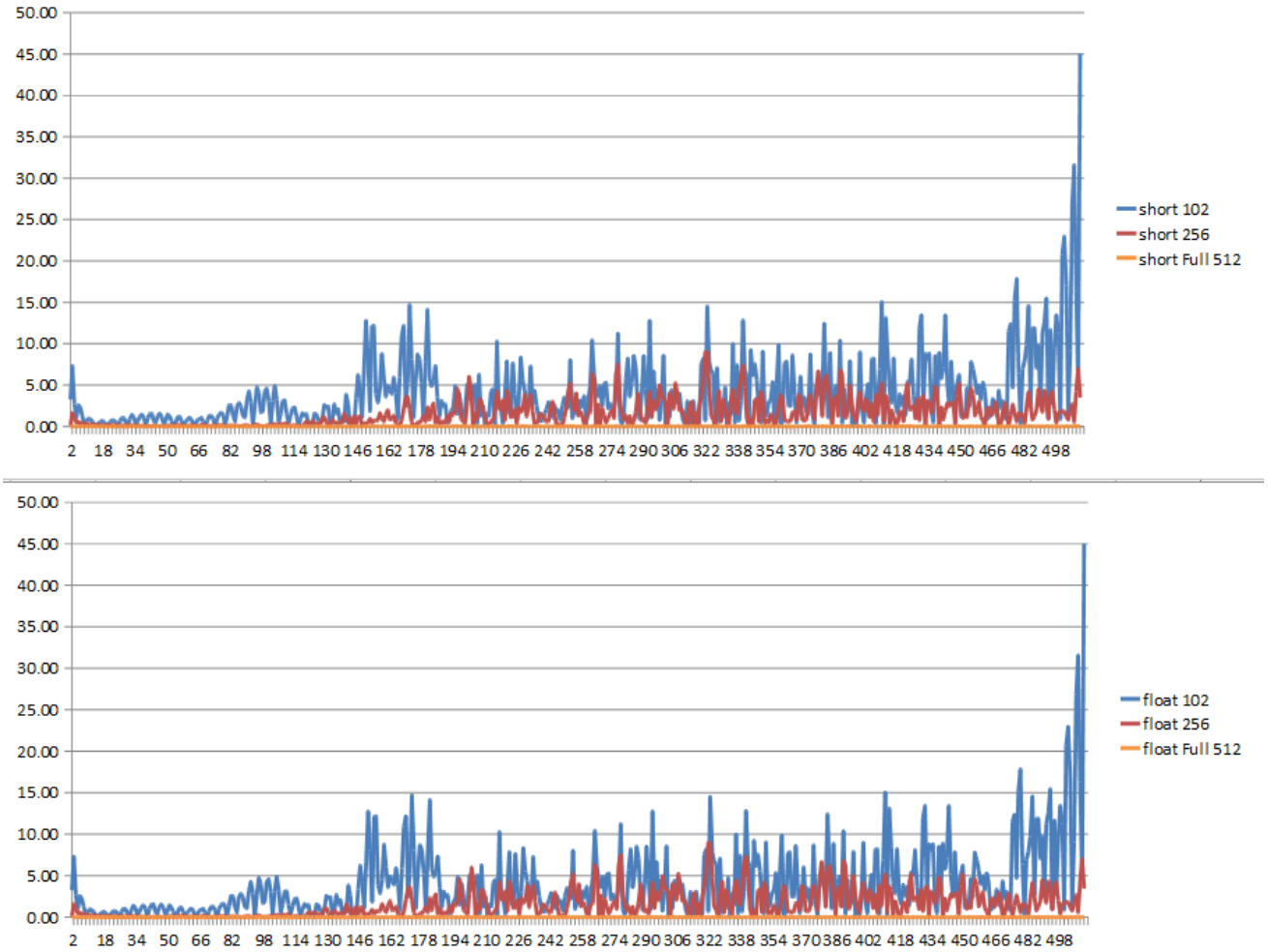

Figure 14: Absolute difference between original and reconstructed scan
Top using short (16 bits) and Bottom using float (32 bits)
Blue: 10:1/5:1 compression with Average Difference 4.0/128
Red: 4:1/2:1 compression with Average Difference 1.5/128
Orange: 2:1/1:1 compression with Average Difference 0.00/128

Note: Storing the data in signed 16 bit format has almost no effect on the difference between original and reconstructed signal. Dropping the noise components has a considerably greater effect. This makes sense as removing noise will affect the signal especially at the peaks. With 1/5 of the original terms in use the difference is approximately 3% on average. Using ½ the components you keep much of the noise and only have a difference of 1% on average. Using all the frequency components and storing the data as a short versus a float has a difference of < 0.01% on average. Figure 15 shows the difference.

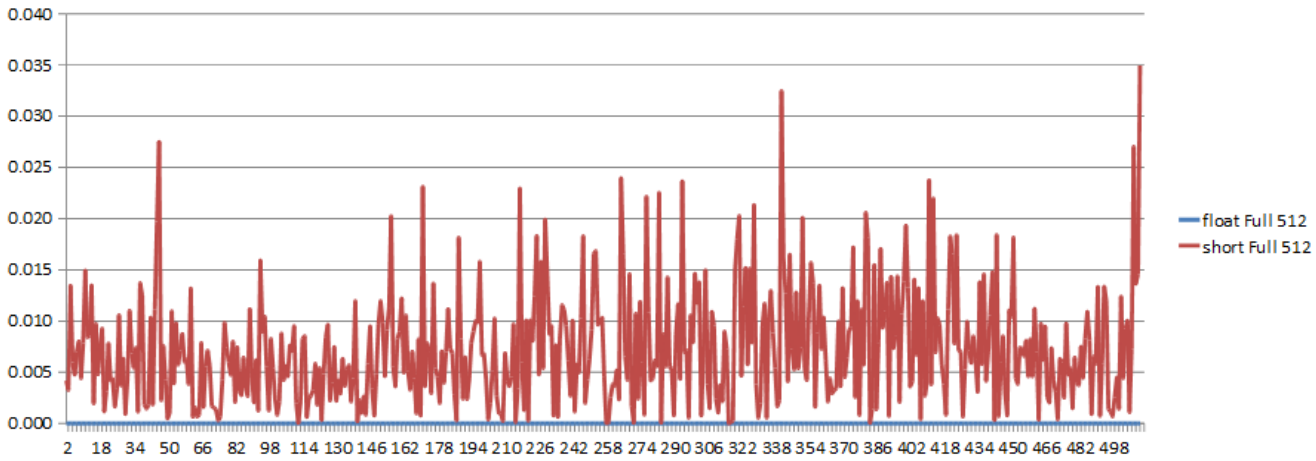

Figure 15: Absolute difference between original and reconstructed for float (32bit) and short (16bit)

10. Alternate Method of Selecting Frequency Components

Early on in this project several variations attempting to select frequencies by threshold were attempted. This method was reasonable and resulted in slight additional gains in compression. It was decided against as you would have to store a series of bits covering the compression range and pack the frequency components are no longer simply sequential. For instance, Figure 16 shows how a threshold could be used to drop out lesser components.

DC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
-26.1	-44.9	58.4	-69.3	38.4	-36.6	172.3	-114.9	65.1	-202.6	233.6	233.9	-77.5	-350.1	-517.4	680.8	161.0	-75.6	-243.6	-41.4	57.2	197.1	61.6	-117.1	-121.5	63.8	0.0	35.5	-25.5	-48.8	-21.5	45.0	0.0	0.0	0.0	21.6	0.0	0.0	0.0	0.0
-23.0	-36.8	31.0	-42.8	0.0	0.0	77.3	0.0	46.8	-128.8	0.0	170.8	117.3	-152.4	-432.9	0.0	328.4	357.6	-173.2	-277.4	-150.6	145.1	334.3	32.1	-258.6	-47.3	0.0	104.3	37.0	-55.2	-80.0	33.0	31.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-34.6	-21.1	0.0	0.0	0.0	49.8	-31.2	106.4	67.6	-97.8	-120.1	-24.9	345.9	0.0	-301.4	-401.6	200.3	504.4	85.6	-282.6	-235.4	-31.9	215.8	156.8	-39.8	-34.3	-175.3	28.6	59.9	60.4	-24.2	-65.1	0.0	34.0	0.0	0.0	0.0	0.0	0.0	0.0
-31.8	0.0	0.0	0.0	-34.9	85.9	-132.2	198.4	118.9	-87.9	-151.5	-231.4	345.3	175.3	-157.8	-296.3	-129.7	245.3	331.2	0.0	-148.3	-215.2	-35.3	94.8	190.3	0.0	-144.4	0.0	0.0	24.3	75.7	-40.8	-32.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-45.6	0.0	0.0	0.0	-50.5	97.6	-193.1	228.2	195.2	-106.1	-114.3	-307.7	118.6	218.4	205.4	-239.9	-283.3	-84.7	305.8	268.3	-29.9	-219.8	-58.8	-90.3	112.9	90.2	0.0	0.0	-73.8	-33.5	73.4	0.0	-20.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-37.4	0.0	0.0	0.0	-44.1	99.8	-203.5	190.9	263.3	-102.9	-83.6	-302.5	-63.6	81.4	595.4	-252.9	-321.1	-46.3	0.0	268.4	186.0	-148.1	-107.9	-80.4	0.0	73.9	33.1	0.0	-43.6	-25.6	38.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-27.2	-28.1	0.0	0.0	-35.1	72.5	-151.0	107.3	318.8	-21.3	-153.7	-289.9	-60.1	-54.0	597.8	-22.9	-339.1	-92.6	-67.5	169.4	224.2	-41.2	-78.6	-63.6	-21.3	33.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-24.4	-33.3	24.4	-21.4	0.0	34.9	-65.6	0.0	350.0	66.3	-231.0	-325.9	82.7	-38.5	228.6	259.4	-188.9	-246.1	-105.3	198.8	124.4	-20.8	37.4	-30.8	-66.5	0.0	0.0	0.0	0.0	0.0	20.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-27.0	-32.9	28.9	-36.6	0.0	-21.0	45.0	-110.9	339.3	131.3	-256.1	-326.3	48.4	228.8	-84.1	202.6	-51.8	-125.1	-202.7	56.6	229.1	0.0	0.0	0.0	0.0	-53.6	-84.3	43.7	37.6	0.0	0.0	-35.6	0.0	23.7	0.0	0.0	0.0	0.0	0.0	0.0
-32.1	-31.6	41.8	-53.3	27.9	-46.4	118.9	-149.4	247.6	178.8	-233.5	-189.9	-211.5	449.2	0.0	-132.1	0.0	42.8	-113.4	-80.4	158.8	142.6	0.0	-69.8	-43.2	-24.4	-39.7	46.8	20.1	0.0	-24.8	0.0	0.0	24.6	0.0	0.0	0.0	0.0	0.0	0.0
0.0	-51.3	51.8	-49.8	31.3	-69.1	181.8	-121.3	84.6	151.1	-137.5	28.9	-416.2	320.6	213.8	-231.3	-119.6	47.9	104.2	25.2	-36.8	50.9	33.8	-40.5	-63.1	0.0	-36.8	59.4	45.3	-84.2	-42.6	50.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-27.3	-37.4	51.4	-60.3	32.7	-84.4	228.8	-74.7	-48.8	51.5	23.9	172.6	-361.1	0.0	62.8	98.1	-101.8	-92.1	175.6	0.0	68.5	0.0	-146.5	0.0	62.5	72.3	-94.3	-51.8	24.6	24.9	-27.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	-52.1	66.8	-74.8	49.3	-96.3	251.0	-38.2	-128.4	-43.8	112.5	279.6	-180.1	-225.9	-363.8	387.6	220.9	-82.0	-59.1	-187.9	206.6	144.2	-104.3	-124.9	0.0	128.3	-65.8	-54.8	0.0	0.0	28.2	0.0	-26.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	-55.5	60.1	-53.1	29.8	-82.6	226.2	0.0	-162.6	-45.3	50.5	340.3	-57.9	-236.1	-577.1	259.1	526.4	39.6	-164.4	-366.1	127.8	296.6	38.3	-102.4	-143.1	0.0	0.0	56.4	37.9	-68.6	-37.1	43.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-22.5	-38.9	46.4	-44.8	0.0	-49.2	164.3	44.3	-145.8	44.9	-118.0	268.4	0.0	-120.6	-310.8	-135.2	398.9	137.8	33.3	-240.1	-87.6	154.3	26.2	42.3	0.0	-57.6	-102.8	0.0	63.4	0.0	-20.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-25.5	-32.8	36.8	-37.1	0.0	0.0	90.1	43.8	-30.2	89.1	-212.4	49.2	45.1	101.6	-28.2	-293.8	-82.3	215.6	305.8	-113.3	-76.3	-80.4	-124.1	152.6	171.5	-45.9	-140.0	-68.0	27.5	50.9	40.1	0.0	-32.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-37.3	0.0	0.0	0.0	-23.1	39.9	0.0	42.6	125.5	50.8	-209.1	-183.7	24.1	359.1	54.8	-284.4	-249.6	47.9	366.3	0.0	0.0	-70.5	-190.8	62.7	168.8	-47.0	-22.3	-39.4	-41.5	0.0	49.6	0.0	0.0	0.0	20.7	0.0	0.0	0.0	0.0	0.0
-41.1	0.0	0.0	0.0	-37.6	72.9	-79.2	44.5	293.4	-79.8	-116.7	-292.4	-37.3	434.4	83.4	-224.1	-103.1	-181.0	90.9	240.3	138.7	-131.9	-72.5	-44.6	20.5	0.0	57.7	-22.4	-48.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-40.4	0.0	0.0	0.0	-41.9	89.0	-131.8	43.1	426.3	-260.8	0.0	-281.2	-89.1	265.8	268.4	-119.1	-135.1	-255.6	-37.6	333.9	151.7	-104.7	-37.6	-34.1	-67.5	0.0	24.9	36.4	-33.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-35.8	0.0	0.0	0.0	-44.3	96.3	-145.6	35.3	483.5	-358.7	81.3	-254.7	-93.9	88.0	399.8	-28.1	-257.1	-250.1	62.9	274.9	0.0	86.3	-40.2	-86.2	-78.4	44.0	-43.2	52.3	32.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-33.8	-21.3	0.0	0.0	-32.6	74.1	-107.1	0.0	439.7	-326.4	94.0	-268.3	-46.1	146.9	218.9	0.0	-196.5	-222.9	55.9	263.3	0.0	55.1	-34.4	-78.5	-38.1	0.0	0.0	74.9	0.0	-35.1	0.0	-26.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-32.7	0.0	0.0	0.0	-37.5	63.5	-50.5	0.0	309.8	-205.3	67.1	-259.3	-24.6	340.4	-107.4	-52.4	-45.3	-183.2	72.9	187.8	72.8	-42.8	-47.5	-40.6	0.0	-22.4	83.6	24.3	-44.8	-49.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-35.5	0.0	0.0	0.0	-31.9	31.9	25.2	0.0	116.3	-38.6	0.0	-152.6	-51.7	354.3	-227.3	-115.7	0.0	-52.9	159.5	52.2	23.3	-68.6	-54.4	23.4	30.9	0.0	70.6	-39.8	-34.0	-67.7	20.3	64.8	-23.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-26.7	-23.8	0.0	0.0	0.0	0.0	108.0	-27.3	-49.8	95.4	-34.1	98.1	-145.6	117.7	-207.4	-52.9	108.1	-20.8	154.9	-40.8	32.4	-126.5	-43.6	136.0	22.2	-28.9	0.0	-57.8	0.0	0.0	43.9	0.0	-37.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-30.7	0.0	0.0	-23.8	0.0	-45.8	199.7	-67.1	-162.9	159.6	-61.1	376.8	-298.9	-100.3	-332.0	221.8	370.9	-221.5	38.9	-113.3	133.9	-80.1	0.0	92.3	-30.1	-39.6	-23.1	-35.6	67.3	0.0	0.0	0.0	-31.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-27.9	-26.8	22.4	-36.8	29.4	-72.3	233.6	-89.4	-198.4	198.9	-85.9	499.9	-368.9	-155.9	-438.6	381.3	518.2	-377.9	-50.6	-152.7	231.3	0.0	0.0	26.9	-78.8	-64.1	58.9	0.0	25.3	0.0	-29.9	38.1	-20.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
-30.7	-26.7	31.1	-42.0	28.3	-57.1	219.5	-109.3	-160.6	219.3	-114.8	381.9	-289.8	-77.3	-381.9	295.0	332.8	-245.3	0.0	-113.9	207.9	-69.8	0.0	55.9	0.0	-23.8	0.0	-54.1	37.7	0.0	0.0	0.0	-22.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 16: Frequency components selected by a threshold.

You would need the following bit set to unpack row 2:

DC to 7 = 11110010
8 to 15 = 11011110
16 to 23 = 11111111
24 to 31 = 11011111
32 to 39 = 10000000

You would need 3 32 bit values to store the bit set and then you would need a minimum of 27 shorts to store the frequency components in a packed fashion. With alignment you might calculate the necessity for 3 x 32 + 28/2 x 2 (27 rounded up) = 17 32-bit values. The primary advantage is that you would only need 17x4 68 bytes to store this information vs. 80 bytes to store the first 40 frequency components regardless of their effect when modeling the signal. The major drawback in this method is that each scan can be a different length and to get the best compaction you would need to use it. This would be a burden especially since you wouldn't be able to index into a file a set number of scans.

11. Gain vs. No Gain on the Frequency Distribution

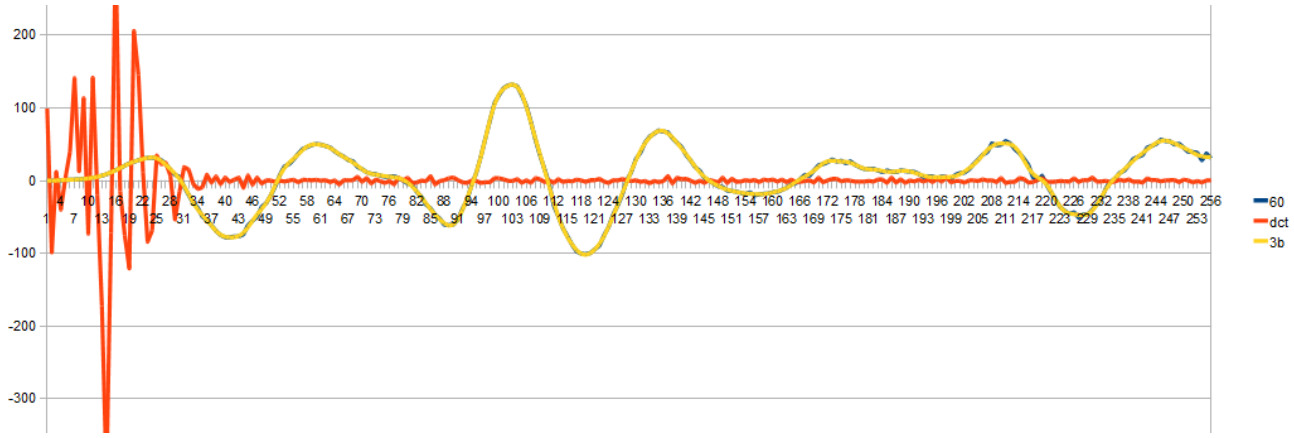

Figure 17: Frequency components of gained signal (with correction for attenuation)

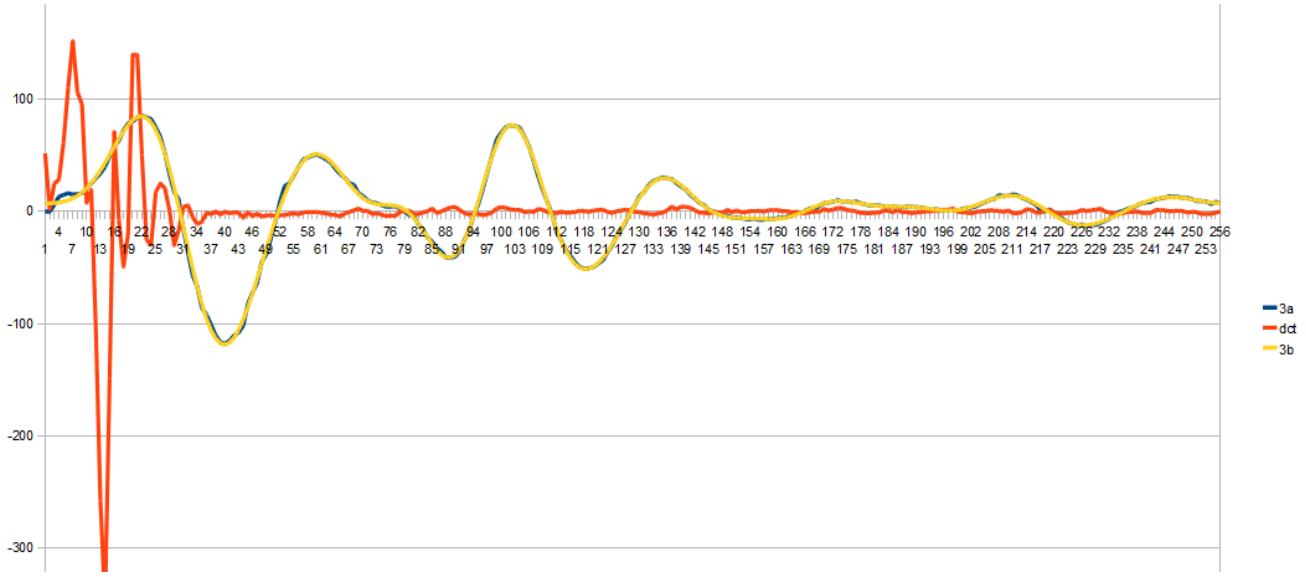

Figure 18: Frequency components of ungained signal (without correction for attenuation)

In Figure 17 the frequency components shown in orange are better balanced in their clustering then the components of Figure 18 which have no correction for attenuation.

12. Conclusion

Using time tested techniques such as the DCT to compress GPR data make sense. We have determined that it is feasible to compress the data using the DCT that it is best to pre-process the data and that we could send only a portion of the frequency information in order to improve bandwidth constraints. The outstanding question that remains is the speed at which the transform can take place. Without any attempt to tune it for performance, it appears to process 8Gb/hour while generating a physical image at the same time. Although this is pretty good for a first pass an effort should be made to tune the speed and determine if the throughput is sufficient.

13. References

[1] Falstad, Paul, *Fourier Series Applet v1.6b*. Retrieved October 15, 2010, from http://www.falstad.com/fourier/

[2] Nyquist–Shannon Sampling Theorem. Retrieved October 15, 2010, from http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[3] Ma, Jianwei and Plonka, Gerlind, The Curvelet Transform, *IEEE Signal Processing Magazine*, March 2010.

[4] Discrete Cosine Transform. Retrieved October 15, 2010, from http://en.wikipedia.org/wiki/Discrete_cosine_transform

[5] List of Fourier-related Transforms. Retrieved October 15, 2010, from http://en.wikipedia.org/wiki/List_of_Fourier-related_transforms

[6] Wendykier, Piotr, *JTransforms version 2.3*. Retrieved October 15, 2010, from http://sites.google.com/site/piotrwendykier/software/jtransforms

[7] Frigo, Matteo and Johnson, Steven. *FFTW version 3.3.2*. Retrieved October 15, 2010, from http://www.fftw.org

[8] ITU-T T.81 Information technology – Digital Compression and Coding of Continuous-tone Still Images – Requirements and Guidelines. Retrieved October 15, 2010, from http://www.w3.org/Graphics/JPEG/itu-t81.pdf

[9] JPEG. Retrieved October 15, 2010, from http://en.wikipedia.org/wiki/JPEG

[10] MPEG Encoding Basics. Retrieved October 15, 2010, from http://www.media-matters.net/docs/resources/Digital%20Files/MPEG/MPEG%20Encoding%20Basics.pdf

* **MICHAEL JEFFORDS** is a Masters Student in Computer Science at Rivier College. He lives in Manchester, New Hampshire with his wife and 5 kids. His research interests include geophysics, data visualization, and image processing.