A 3D Fast Hartley Transform Plugin for ImageJ

Robert P. Dougherty

OptiNav, Inc, 10914 NE 18 ST, Bellevue, WA, USA

ABSTRACT

A 3D Fast Hartley Transform (FHT) plugin has been developed for ImageJ. It is similar to the 2D FHT code that is built into ImageJ, and relies on the same underlying 1D FHT software. The availability of a 3D FHT code enables the construction of a range of 3D plugins for filtering and other operations in ImageJ. This paper gives an overview of some aspects of the FHT and relates it to the more-familiar discrete Fourier transform. The principle difficulty of creating a multi-dimensional FHT code based on multiple applications of a 1D code is reviewed and the solution used in the new plugin is presented. The paper is intended primarily for ImageJ developers who have some knowledge of Fourier analysis in image processing.

Keywords: ImageJ, Fast Hartley Transform, FHT, FFT, convolution, 3D deconvolution

1. INTRODUCTION

Many image processing operations benefit greatly from the application of Fast Fourier Transform (FFT) techniques. One example is convolution. Suppose two images with m dimensions and N pixels (or voxels) per dimension are to be convolved. A direct convolution would require $O(N^{2m})$ operations. The faster alternative is to apply forward FFT processing to convert the images to the spatial-frequency domain, perform frequency-by-frequency multiplication, and take an inverse FFT of the result. The number of operations for a 1D forward or inverse FFT is $O(N \log N)$.[1] By application of successive 1D FFTs, an n-dimensional forward or inverse FFT can be computed in $O(N^m \log N)$ operations. Since the frequency-by-frequency multiplication takes $O(N^m)$ operations, the total cost of the FFT-based convolution is $O(N^m \log N)$. The speedup can make the difference between a quick calculation and an infeasible one. For example with $N = 256$ in three dimensions, the number of operations for an FFT-based convolution is a small multiple of the number of voxels, 16,777,216. Without using an FFT algorithm, the calculation would take on the order of a million times as long. Practical deconvolution algorithms, such as the Richardson-Lucy algorithm,[2] depend on applying convolution within an iteration loop. In this sense, an FFT algorithm is an enabler for 3D deconvolution processing.

1.1. 1D HARTLEY TRANSFORM

The FFT algorithm in ImageJ[3] is based on a Pascal program developed by NIH Image by Arlo Reeves in connection with his Master's thesis.[4] It uses a variation of the FFT known as a Fast Hartley Transform (FHT). Whereas a Fourier transform is based on complex exponential functions, a Hartley transform uses only real numbers and expands the function or sequence of values in terms of cas functions: $\mathrm{cas}(t) \equiv \cos(t) + \sin(t)$. Let $V(0)$, $V(1)$, … $V(N\text{-}1)$ be a sequence of real numbers. The discrete Hartley transform, $H(k)$, of $V(n)$ is given by

$$H(k) = \sum_{n=0}^{N-1} V(n)\,\mathrm{cas}\left(2\pi \frac{kn}{N}\right) \tag{1}$$

with the inverse transform

$$V(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k)\,\mathrm{cas}\left(2\pi \frac{kn}{N}\right). \tag{2}$$

For reference, the discrete Fourier transform and its inverse are

$$F(k) = \sum_{n=0}^{N-1} V(n) e^{i2\pi \frac{kn}{N}}, \quad V(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) e^{-i2\pi \frac{kn}{N}}. \tag{3,4}$$

The discrete Hartley transform is equivalent to the discrete Fourier transform because they are related by

$$H(k) = \mathrm{Re}\big(F(k)\big) - \mathrm{Im}\big(F(k)\big) \tag{5}$$

$$2F(k) = \big[H(k) + H(-k)\big] - i\big[H(k) - H(-k)\big] \tag{6}$$

Reference 4 should be consulted for more information regarding continuous and discrete Hartley transforms and their relationship to Fourier transforms.

Notice that Equation (6) makes use of $H(-k)$, whereas Equations (4) and (5) suggest k in the range of 0 to N-1. It can be shown that $H(k)$ is periodic with period N, so $H(-k) = H(N-k)$. As k runs from 0 to N-1, the basis functions $\mathrm{cas}\left(2\pi\frac{kn}{N}\right)$ increase and then decrease in spatial frequency, as illustrated in Figure 1. Spectral plots can appear more intuitive when shown on a scale with the lowest spatial frequency in the center and the high frequencies at the ends of the range. This can be achieved by plotting k from $-N/2$ t $N/2$-1, as shown in Figure 2.

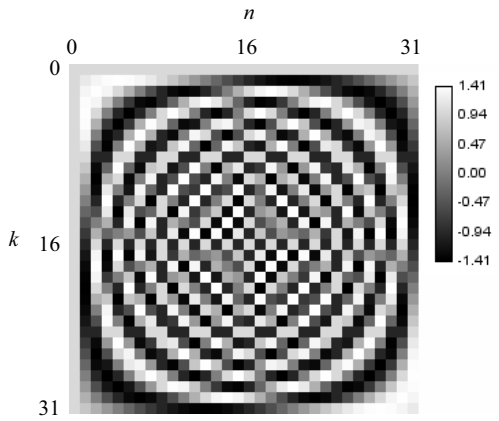

Figure 1. The 1-D discrete Hartley transform basis functions $cas\left(2\pi\frac{kn}{N}\right)$ for $N = 32$. The frequency index is k and the spatial index is n.

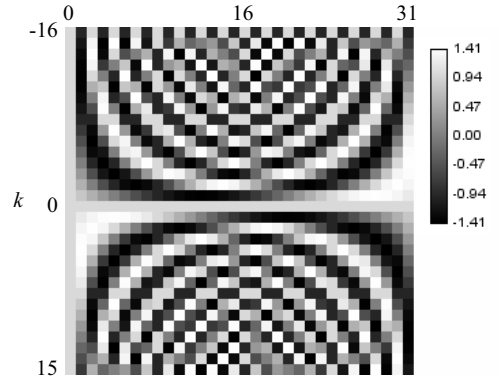

Figure 2. The 1-D discrete Hartley transform basis functions for $N = 32$ shown on a scale with the frequency index running from -16 to +15. On this scale, the slowly varying basis functions are near the center of the range.

The 1D FHT algorithm used in ImageJ is described in Reference 4. It is similar to the FFT, but actually faster in this application because it does not have the overhead of handling the imaginary part of the data. This algorithm is a convenient choice for Java, since Java does not have native support for complex numbers. The algorithm uses 32-bit floating point data and requires N to be a power of 2, but many of the methods that call it provide padding and mapping to support sequences of different lengths and types.

ImageJ plugins for 3D FFTs, convolution, and deconvolution using complex arithmetic are available.[5] This project is distinct from the FHT work discussed in this paper.

1.2. 2D Hartley Transform

The 2D FFT features of ImageJ are in the Process/FFT submenu. The FFT command (Process/FFT/FFT) performs a 2D FHT and displays the power spectrum of the result as an 8-bit image. The horizontal and vertical frequency indices are shifted to the range of $-N/2$ to $+N/2 - 1$ in order to place the lowest frequency a the center of the image. Padding (with the mean value) is applied to make N a power of 2, and the same N is used for the horizontal and vertical dimensions. The displayed power spectrum is not the actual FHT data; it is an image representing an FHT object. The FHT data is stored internally as part of the FHT object. As of ImageJ version 1.35e, there is an option command that permits displaying the 32-bit FHT power spectrum and the raw 32 bit FHT data. This last image is displayed in the native order with the lowest frequencies in the center. Equations (1) and (2) show that the forms of the forward are inverse discrete Hartley transforms are identical, except for the division by N in the inverse transform. This can be verified by first choosing Fast Hartley Transform in Process/FFT/FFT Options… and then performing an FFT on an image. Next, the FHT output window is selected and another FFT is performed. The FHT output of this second operation is identical to the original image except that is it scaled by N as is 32-bit, square and power-of-2 in size, even if the original did not have all of those characteristics. Using the official inverse transform (Process/FFT/Inverse FFT) on the 8-bit "FFT" window restores the original dimensions and image type from information that is hidden in the FHT object, and also divides by N to preserves the scaling.

The Process/FFT menu has a number of commands that perform useful functions with the 2D FHT, such as correlation, convolution, deconvolution, and filtering. These are beyond the scope of this paper. It is noted in passing that that filtering commands, based on Joachim Walter's FFT Filter plugin[6] use Tile-Mirror padding to reduce artifacts that would otherwise result from the periodic assumption of the discrete transform.

In extending the FHT from 1D to 2D, there is an interesting difficulty that does not arise with the complex-exponential FFT. The 2D FFT

$$F(k_1,k_2)=\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}V(n_1,n_2)e^{i2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)}=\sum_{n_1=0}^{N_1-1}\left[\sum_{n_2=0}^{N_2-1}V(n_1,n_2)e^{i2\pi\frac{k_2n_2}{N_2}}\right]e^{i2\pi\frac{k_1n_1}{N_1}} \tag{7}$$

can be performed by transforming the two dimensions successively. In the first step, the transform over n_2 is performed separately for each n_1 to give an intermediate result, say, $G(n_1,k_2)$. Then, for each k_2, the transform of $G(n_1,k_2)$ is done over n_1 to produce the final result $F(k_1,k_2)$.

The 2D discrete Hartley transform is

$$H(k_1,k_2)=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2)cas\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right] \tag{8}$$

The cas function of a sum of two terms does not have a convenient product form analogous to Equation (7). The solution, due to Bracewell,[7] is to temporarily pretend that the cas function is separable and apply the 1D FHT successively in the first and second directions. This results in the row-column sum expression

$$T(k_1,k_2)=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2)\text{cas}\left[2\pi\frac{k_1n_1}{N_1}\right]\text{cas}\left[2\pi\frac{k_2n_2}{N_2}\right] \tag{9}$$

Using trigonometry formulas to expand Equation (8) and comparing with Equation (9), it can be shown that

$$2H(k_1,k_2)=T(k_1,k_2)+T(k_1,N_2-k_2)+T(N_1-k_1,k_2)-T(N_1-k_1,N_2-k_2) \quad = \text{A} + \text{B} + \text{C} - \text{D}. \tag{10}$$

This expression is used to convert $T(k_1,k_2)$ into $H(k_1,k_2)$ in place. A double loop iterates over (k_1,k_2) pairs in the first quadrant: $0 \le k_1 \le N_1/2$, $0 \le k_2 \le N_2/2$. For each pair, expressions similar to Equation (10) to simultaneously update $H(k_1,k_2)$, $H(N_1-k_1,k_2)$, $H(k_1,N_2-k_2)$, and $H(N_1-k_1,N_2-k_2)$. This is described in Reference 4 and implemented in FHT class in ImageJ.

2. 3D FHT

The 3D forward and inverse discrete Hartley transforms are

$$H(k_1,k_2,k_3)=\sum_{n_3=0}^{N_3-1}\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2,n_3)\,cas\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}+\frac{k_3n_3}{N_3}\right)\right] \tag{11}$$

and

$$V(k_1,k_2,k_3)=\frac{1}{N_1N_2N_3}\sum_{k_3=0}^{N_3-1}\sum_{k_2=0}^{N_2-1}\sum_{k_1=0}^{N_1-1}H(k_1,k_2,k_3)\,cas\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}+\frac{k_3n_3}{N_3}\right)\right]. \tag{12}$$

The problem of applying the 1D FHT algorithm to this case can be treated by transforming in all three directions and re-arranging the results in a manner analogous to the 2D formulation.[8] The ImageJ plugin FHT_3D[9] uses a slightly different approach in which 2D FHT transforms are computed for each slice of an image stack, and then a 1D transform of the results is taken in direction between the slices. Finally, the results are used to produce the 3D transform.

2.1. Formulation of FHT_3D

FHT_3D uses the same 1D FHT code as ImageJ. The 2D FHT is similar to the low-level code in ImageJ, but does not require the width and height of the image to match.

The method used by FHT_3D to assemble the 3D FHT is described below. Let direction 3 be perpendicular to the slices in the image stack and suppose the 2D FHT has already been computed for each n_3, giving

$$U(k_1,k_2,n_3)=\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2,n_3)\,cas\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right] \tag{13}$$

For each (k_1,k_2), the FHT of *U* in the slice-direction is computed:

$$\begin{aligned}W(k_1,k_2,k_3)&=\sum_{n_3=0}^{N_3-1}U(k_1,k_2,n_3)\,cas\left(2\pi\frac{k_3n_3}{N_3}\right)\\&=\sum_{n_3=0}^{N_3-1}\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2,n_3)\,cas\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right]cas\left(2\pi\frac{k_3n_3}{N_3}\right)\end{aligned} \tag{14}$$

To express H in terms of W, write

$$2H(k_1,k_2,k_3)=\sum_{n_3=0}^{N_3-1}\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1}V(n_1,n_2,n_3)\,cas[\alpha+\beta], \tag{15}$$

where $\alpha=2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)$ and $\beta=2\pi\frac{k_3n_3}{N_3}$. Using

$$2cas(\alpha+\beta)=cas(\alpha)cas(\beta)+cas(\alpha)cas(-\beta)+cas(-\alpha)cas(\beta)-cas(-\alpha)cas(-\beta), \tag{16}$$

gives

$$\begin{aligned}2H(k_1,k_2,k_3)&=W(k_1,k_2,k_3)+W(k_1,k_2,N_3-k_3)+\\&\quad W(N_1-k_1,N_2-k_2,k_3)-W(N_1-k_1,N_2-k_2,N_3-k_3),\end{aligned} \tag{17}$$

where periodicity has been applied to remove the negative indices.

Equation (17) is used to update $H(k_1, k_2, k_3)$ eight points at a time. The code is presented in Algorithm 1, where the $N_3 \times N_1 N_2$ array "data" contains W and the start of the algorithm and H at the completion. The indexing scheme is that $W(k_1, k_2, k_3)$ and $H(k_1, k_2, k_3)$ are stored in $\text{data}[k_3][k_1 + k_2 N_1]$. The arrays in data are actually pixel arrays for the ImageJ FloatProcessors in the stack.

Algorithm 1. Convert the array data[N3][N1*N2] from a 1D FHT of 2D FHTs into a full 3D FHT.

```
float A,B,C,D,E,F,G,H;
int k1C,k2C,k3C;
for(int k3 = 0; k3 <= N3/2; k3++){
        k3C = (N3 - k3) % N3;
        for(int k2 = 0; k2 <= N2/2; k2++){
                k2C = (h - k2) % h;
                for (int k1 = 0; k1 <= N1/2; k1++){
                        k1C = (N1 - k1) % N1;
                        A = data[k3][k1 + N1*k2C];
                        B = data[k3][k1C + N1*k2];
                        C = data[k3C][k1 + N1*k2];
                        D = data[k3C][k1C + N1*k2C];
                        E = data[k3C][k1 + N1*k2C];
                        F = data[k3C][k1C + N1*k2];
                        G = data[k3][k1 + N1*k2];
                        H = data[k3][k1C + N1*k2C];
                        data[k3][k1 + N1*k2] = (A+B+C-D)/2;
                        data[k3C][k1 + N1*k2] = (E+F+G-H)/2;
                        data[k3][k1 + N1*k2C] = (G+H+E-F)/2;
                        data[k3C][k1 + N1*k2C] = (C+D+A-B)/2;
                        data[k3][k1C + N1*k2] = (H+G+F-E)/2;
                        data[k3C][k1C + N1*k2] = (D+C+B-A)/2;
                        data[k3][k1C + N1*k2C] = (B+A+D-C)/2;
                        data[k3C][k1C + N1*k2C] = (F+E+H-G)/2;
                }
        }
}
```

2.1. Attributes FHT_3D

The plugin FHT_3D is simpler than the FHT class in ImageJ in that it presents the data values directly. Calling the plugin once on an image stack converts it from the spatial to the frequency domain. The native ordering in used; the low frequencies are in the center of the images and the stack. Running the plugin again converts back to the spatial domain. A normalization of $1/\sqrt{N_1 N_2 N_3}$ is applied, so forward and inverse transforms are exactly the same; the plugin does not have an input to specify the forward or inverse direction. The values of N_1, N_2, and N_3 need not match, but N_1 and N_2 must be powers of 2. If N_3 is not a power of 2, then the calculation proceeds with a slow Hartly transform in the slice direction.

3. CONCLUSION

A 3D FHT plugin for ImageJ is available to users and developers. At a low level, it is consistent with the 2D FHT that is built into ImageJ, but it lacks the power spectral display and user-shielding features of the ImageJ's FHT. It provides a basis for developing 3D filtering and other image manipulation techniques. One example is Convolve 3D.[10]

ACKNOWLEDGMENTS

This work is based on ImageJ by Wayne Rasband. Thanks to Lucina Emerson for pointing out Bracewell's formulas for multidimensional FHTs.

REFERENCES

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.
2. M. Bertero and P. Boccacci, *Introduction to Inverse Problems in Imaging*, Institute of Physics Publishing, Bristol, UK, 1998.
3. W.S. Rasband, *ImageJ*, U. S. National Institutes of Health, Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij/ , 1997-2006.
4. A.A. Reeves, (1990) Optimized Fast Hartley Transform for the MC68000 with Applications in Image Processing, MSc Thesis, Thayer School of Engineering, Dartmouth College, http://rsb.info.nih.gov/ij/docs/ImageFFT/ 1990.
5. Nick Linnenbrügger, "FFTJ and DeconvolutionJ," http://rsb.info.nih.gov/ij/plugins/fftj.html, 2001-2002.
6. Joachim Walter, "FFT Filter," http://rsb.info.nih.gov/ij/plugins/fft-filter.html, 2001-2003.
7. R.N. Bracewell, O. Buneman, H. Hao, and J. Villasenor, "Fast two-dimensional Hartley Transforms," *Proc. IEEE* **74**(9) pp. 1282-1283, 1986.
8. H. Hao and R.N. Bracewell, "A three-dimensional DFT algorithm using the fast Hartley transform," *Proc. IEEE* **75**(2) pp. 264-266, 1987.
9. R. Dougherty, "3D Fast Hartley Transform," http://www.optinav.com/imagej.html, 2005.
10. R. Dougherty, "Convolve 3D," http://www.optinav.com/imagej.html, 2005.