An Open Source, Fast Ultrasound B-Mode Software Implementation for Commodity Hardware

Release 1.0.0

Matthew McCormick[1]

March 19, 2010

[1]matt@mmmccormick.com

Abstract

This document describes an open source, high performance ultrasound B-Mode implementation based on the Insight Toolkit (ITK). ITK extensions are presented to calculate the radio-frequency (RF) signal envelope. A variety of 1D Fast Fourier Transform options are introduced including VNL, FFTW, and an OpenCL solution. Scan conversion is implemented for phased array or curvilinear transducers. The entire image processing pipeline is streamable to limit memory consumption during multi-frame or 3D acquisitions with the introduction of an `itk::StreamingResampleImageFilter`.

Latest version available at the Insight Journal [`http://hdl.handle.net/10380/3159`]

Contents

1 Introduction **2**

2 Signal Envelope Calculation **2**
2.1 B-Mode Image Formation Background . . . 2
2.2 Implementation . . . 4
FFT1D Image Filters . . . 4
VNL FFT1D Image Filters . . . 4
FFTW FFT1D Image Filters . . . 5
OpenCL FFT1D Image Filters . . . 5
itkAnalyticSignalImageFilter . . . 5
itkBModeImageFilter . . . 5
2.3 Results . . . 6

3 Scan Conversion **6**
3.1 Background . . . 6
3.2 Implementation . . . 7

StreamingResampleImageFilter 7
RThetaToCartesianTransform, CartesianToRThetaTransform 7
ResampleRThetaToCartesianImageFilter 7
3.3 Results 8

4 Discussion **8**

1 Introduction

Diagnostic medical ultrasound has a plethora of imaging modes that focus on different tissue phenomena, such as morphology, blood flow, or deformation. Brightness Mode (B-Mode) ultrasound has been around for the longest time, but it remains the most important and widely used mode. A B-Mode image reflects tissue morphology by displaying the intensity of scattered sound from beams of ultrasound that interrogate tissue in a raster-like pattern.

In this paper we present an open source B-Mode image implementation based on the InsightToolkit (ITK)[1]. In the past, clinical ultrasound machines required specialized signal processing hardware to create B-Mode images in real-time. Here we demonstrate how ITK's well-designed medical imaging framework written in C++ can deliver high frame rates on commodity hardware.

Extensions to ITK that may be useful outside of ultrasound imaging are also presented. A streaming version of the `itk::ResampleImageFilter` allows the image processing pipeline to maintain streaming capabilities when transformations are required. This is a common requirement, for example in resizing or registration operations. Also, a large set of Fast Fourier Transform (FFT) filters that operate along a single direction are presented.

First, we give a background on B-Mode image formation and describe the FFT-based implementation. Next, we describe scan conversion on sector type transducers and the use of the `itk::StreamingResampleImageFilter`. Finally, we conclude with a brief discussion of the results.

2 Signal Envelope Calculation

2.1 B-Mode Image Formation Background

The majority of medical ultrasound systems consist of an array of transducer elements that are excited at frequencies from 2 to 15 MHz. During pulse-echo acquisition, a short time pulse propagates through tissue, and a portion of the wave is scattered back to the transducer as it encounters acoustic impedance inhomogeneities. The volume of tissue investigated in a single pulse-echo is limited to a column of tissue, a beamline, by focusing with time delays applied to the elements and apodization of the array. Multiple beamlines varied by starting location along the array or electronic steering angle span the tissue to generate an image. Depth along the beamline is inferred by assuming a speed of sound, c , usually 1540 m/s, and applying the range equation,

$$d_{echo} = \frac{c t_{echo}}{2}$$

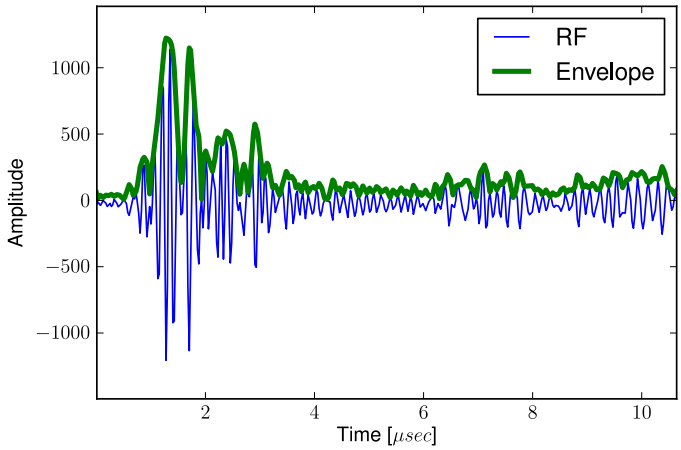


Figure 1: Sample RF signal and its corresponding envelope.

The received scattered energy, commonly designated radio-frequency (RF) data, is highly oscillatory. An envelope of the RF data reflects local scattering amplitude, and this is what comprises a B-Mode image.

Calculation of the RF envelope is often performed with the analytic signal. The analytic signal is used to decompose a signal into its local amplitude and local phase[2, 3][1]. The analytic signal, $f_A(x)$, of a real signal, $f(x)$, is defined to be

$$f_A(x) = f(x) - i\, f_H(x)$$

Where $f_H(x)$ is the Hilbert Transform of $f(x)$ given by

$$f_H(x) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(x')}{x' - x}\, dx'$$

The Hilbert Transform can be calculated without performing convolution by applying the following property in Fourier space,

$$F_H(\xi) = F(\xi) \cdot i\, \mathrm{sgn}(\xi)$$

As a result, the analytic signal's representation in Fourier space is

$$\begin{aligned} F_A(\xi) &= F(\xi) - i F_H(\xi) \\ &= F(\xi) - i F(\xi) \cdot i\, \mathrm{sgn}(\xi) \\ &= F(\xi) \cdot [1 + \mathrm{sgn}(\xi)] \end{aligned} \tag{1}$$

Even though it does not have ideal properties when applied to a discrete signal[4], an approach to calculate

[1] Local phase is a 1D concept, and the analytic signal is not defined for multi-dimensional signals. Therefore, `AnalyticSignalImageFilter` only operates along one direction of an `itk::Image`. To split a multi-dimensional image into structural and energetic information, see the monogenic signal[2].

the analytic signal is then to calculate the Fourier Transform, multiply the first half by two, multiply the second half by zero, and take the inverse Fourier Transform.

For narrow band signals, the local phase, $\phi(x)$, and the local energy, $A(x)$, of $f(x)$ can be interpreted as[2]

$$\begin{aligned} \phi(x) &= \tan^{-1}(f(x)/f_H(x)) \\ A(x) &= \sqrt{f^2(x) + f_H^2(x)} \end{aligned}$$

After the envelope, $A(x)$, is calculated, post-processing can be performed. The majority of an ultrasound image's content is speckle, random scattering caused by scatterers much smaller than the excitation wave. Speckle's amplitude has a Rayleigh distribution[5]. Since the Rayleigh distribution is skewed to lower values and a small proportion of the amplitudes have very large values, a logarithmic intensity transform is commonly applied to the envelope to improve image contrast.

2.2 Implementation

FFT1D Image Filters

A set of Fast Fourier Transforms were created, the `itk::FFT1DRealToComplexConjugateImageFilter`, `itk::FFT1DComplexToComplexImageFilter`, and `itk::FFT1DComplexConjugateToRealImageFilter`. Their design is based on the FFT filters present in ITK, such as `itk::FFTRealToComplexConjugateImageFilter`. Unlike their counterparts that perform the transform in all dimensions, these filters perform the transform along one direction, specified with the `SetDirection()` method. They are base classes that can be used without reference to the configuration and libraries available on a system. One of the three child classes that follow provide an FFT implementation using ITK's object factory design.

There are a few other notable differences between these classes and the classes that operate on all dimensions. First, they do not have abstract `FullMatrix()` methods. This method provided a means for the FFTW[6]-based filters to specify that they take advantage of complex conjugate symmetry, which occurs when taking the Fourier Transform of a real image. As a result, their size after transformation is truncated. We used routines that generate the full output for all filters, including `itk::FFTW1DRealToComplexConjugateImageFilter`, since it was necessary for the `itk::AnalyticSignalImageFilter`. Secondly, these filters re-implement `GenerateInputRequestedRegion()` and `EnlargeOutputRequestedRegion()` to ensure the entire extent is processed along the direction the FFT will be applied.

VNL FFT1D Image Filters

Since ITK ships with VXL[7] included, the VNL-based transform classes, `itk::VnlFFT1DRealToComplexConjugateImageFilter`, `itk::VnlFFT1DComplexToComplexImageFilter`, and `itk::VnlFFT1DComplexConjugatetoRealImageFilter`, are default implementations. They may be slower than other alternatives, but they are available on all platforms. These filters are multithreaded.

FFTW FFT1D Image Filters

FFTW, the *Fastest Fourier Transform in the West*[6] is a well known open source, optimized set of FFTs that operate on `float` or `double` input. A *plan* is determined for best performance given the transform size. The `itk::FFTW1DRealToComplexConjugateImageFilter`, `itk::FFTW1DComplexToComplexImageFilter`, and `itk::FFTW1DComplexConjugateToRealImageFilter` can be used if FFTW libraries are available. The file `itkFFTWCommon.h` from ITK was modified and extended, and the file and classes therein were renamed to avoided naming conflicts. Suggestions in the recent paper by Lehmann[8] to thread plan generation and normalization were incorporated.

OpenCL FFT1D Image Filters

Recent evolution of computing hardware suggest that continued adherence to Moore's Law will not be achieved through higher clock speeds but through increased parallelization. Attention has focused on adapting the parallel processing capabilities of Graphics Processing Units, (GPUs), to general, data parallel problems like image processing.

OpenCL[9], Open Computing Language, is a new open standard released by the Khronos Group, the same developers of the popular OpenGL graphics standard, that provides a platform and hardware vendor independent API for parallel programming on heterogeneous systems. An API defines communication between a host system, a CPU, and a target system, a GPU or multi-core CPU. Additionally, an extended C language defines *kernels* that can be executed on the target system.

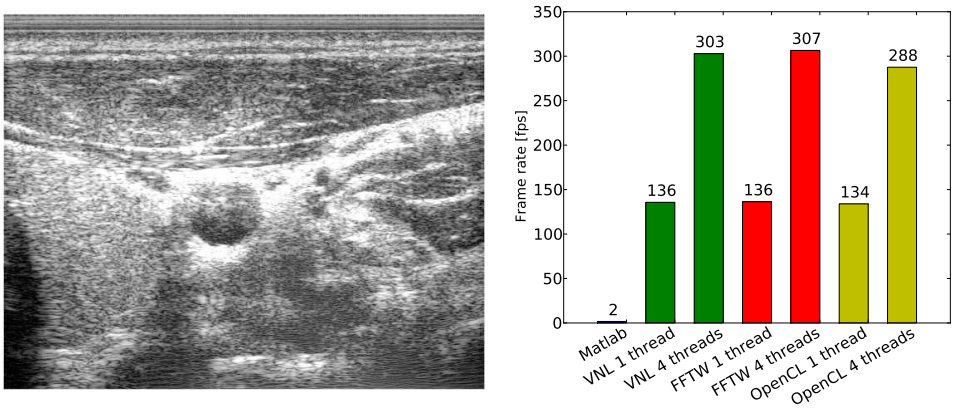
A number of hardware vendors have adopted the standard and provided initial drivers for the 1.0 standard. Apple™open sourced OpenCL FFT code[10] based on the papers by Volkov and Kazian[11] and Govindara et al.[12] The Cooley-Tukey framework is used to decompose a large Discrete Fourier Transform into smaller transforms whose size is optimized to the hardware architecture of the target GPU. An OpenCL system requires a nested compiler to generate kernel code for the target system at runtime in order to remain hardware independent. Apple's OpenCL FFT takes advantage of this capability by generating kernel source code at runtime optimized to the data's dimensions. We made slight modifications to Apple's library so it can be built on platforms other than the Macintosh™. The ITK classes that encapsulate the OpenCL FFT are `itk::OpenCL1DRealToComplexConjugateImageFilter`, `itk::OpenCL1DComplexToComplexImageFilter`, and `itk::OpenCLComplexConjugateToRealImageFilter`. The size of the input image in the FFT direction must be a power of two. These filters are not multi-threaded.

itkAnalyticSignalImageFilter

An `itk::AnalyticSignalImageFilter` processes a real input image to produces a complex output image. A forward Fourier Transform is applied with `itk::FFT1DRealToComplexConjugateImageFilter`, modulation is applied as suggested by Equation 1, and an inverse Fourier Transform is applied with `itk::FFT1DComplexToComplexImageFilter`. The direction in the image that the analytic signal is calculated can be set with `SetDirection()`.

itkBModeImageFilter

A composite filter intended to calculate the B-Mode envelope image from RF input is the `itk::BModeImageFilter`. Since the OpenCL FFT requires data whose size is a power of two, and



(a) Input image: carotid artery. (b) Performance test.

Figure 2: Performance in frames per second of Matlab and ITK implementations with different FFT libraries.

the VNL and FFTW require or perform optimally when they can be factored into small primes[8], a `itk::ConstantPadImageFilter` and `itk::RegionFromReferenceImageFilter`[8] precede and follow the `itk::AnalyticSignalImageFilter` if they need to be zero padded to a power of two. A `itk::ComplexToModulusImageFilter` calculates the envelope from the analytic signal. Logarithmic intensity transform is applied with a `itk::AddConstantToImageFilter` to avoid taking the logarithm of zero, followed by a `itk::Log10ImageFilter`.

2.3 Results

Performance of the various FFT backends was tested on a carotid artery image of size 1556x508x14. The platform was an Intel™Core 2 Quad 6600 CPU running Gentoo Linux connected to an NVidia™8800GT GPU via a PCI Express 2.0 bus. A similar implementation written in Mathworks Matlab™was also tested. The VNL, FFTW, and OpenCL FFT backends were tested with one and four threads by setting the `ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS` environmental variable. Results are shown in Figure 2.

3 Scan Conversion

3.1 Background

Like other medical imaging modalities, ultrasound has anisotropic spacing. The resolution along a beamline's axis is high; it is determined by the received signal's frequency content. In contrast, the resolution in directions orthogonal to beam's propagation, the lateral and elevational directions, is low; they are determined by transducer element pitch and focusing. In the case of a linear array, ITK's built-in sample spacing mechanism addresses the anisotropy.

However, there are other common ultrasound transducer configurations that do not sample on a rectilinear grid. A curvilinear array's elements are positioned on a curved instead of a flat surface. A phased array keeps the center of the beam at the same initial location, but it electronically steers the direction of propagation with appropriate time delays during excitation. Also, single element transducers rotated about a fixed pivot point have a similar sampling pattern. In order to address these configurations, we introduce an `itk::Transform`

to scan convert images given r, a sample's distance along the beamline, and θ, the angle between a line perpendicular to the transducer at $r = 0$ and the direction of beamline's propagation.

3.2 Implementation

StreamingResampleImageFilter

`itk::StreamingResampleImageFilter` inherits from `itk::ResampleImageFilter`. Its behavior is similar to its parent class, but it does not set the input's `RequestedRegion` to the `LargestPossibleRegion` during the `PropagateRequestedRegion()` stage of a pipeline update. Instead, `GenerateInputRequested()` is overwritten to request only the region implied by the bounding box that results from transforming the output's `RequestedRegion` bounding box. As a consequence, streaming can continue upstream from the filter.

This method warrants caution because it is not valid for all transforms. For example, transforms that "bulge" outside of the transformed bounding box will generate incorrect output. An abstract `IsLinear()` method is defined in `itk::Transform`, but this is too restrictive a condition to throw an exception on `SetTransform()`. Therefore, the user must apply `itk::StreamingResampleImageFilter` only when appropriate.

RThetaToCartesianTransform, CartesianToRThetaTransform

The `itk::RThetaToCartesianTransform` and `itk::CartesianToRThetaTransform` define the transforms between $r-\theta$ space and Cartesian, $x-y$ space. `itk::RThetaToCartesianTransform` is needed to scan convert data collected from a curvilinear array.

Internally, five parameters define the transform. However, these parameters and not usually set with `SetParameters()`. Instead, `SetRmin()` specifies the distance from the center of rotation to the first sample in the input's `RDirection`. Similarly, `SetRmax()` specifies the distance in physical units to the last point in the `RDirection`. Since the input's `itk::Image` spacing in the `ThetaDirection` is not meaningful, it must be specified with `SetSpacingTheta()`. Also, `SetThetaArray()` takes an `itk::Array<double>` with the angles that every sample in the `ThetaDirection` were sampled. The angles should be in radians, and it is assumed they are equally spaced.

ResampleRThetaToCartesianImageFilter

The `itk::ResampleRThetaToCartesianImageFilter` is a convenience wrapper around an `itk::StreamingResampleImageFilter` and a `itk::RThetaToCartesianTransform`. The filter takes the input's information and `itk::MetaDataDictionary` contents, and relays this information to its transform. The input should have `itk::MetaDataDictionary` entries for "Radius" or "RadiusString" and "Theta" or "ThetaString". "Radius" is of type `double` and contains the distance in the `RDirection` to the first sample. "RadiusString" is a string representation of the value. "Theta" is an `itk::Array<double>` containing the angles in radians for every value in the ThetaDirection. "ThetaString" is a string representation of the values containing the floating point value followed by a space for each value.

While a `itk::StreamingResampleImageFilter` is used, streaming is not allowed in the $r-\theta$ plane.

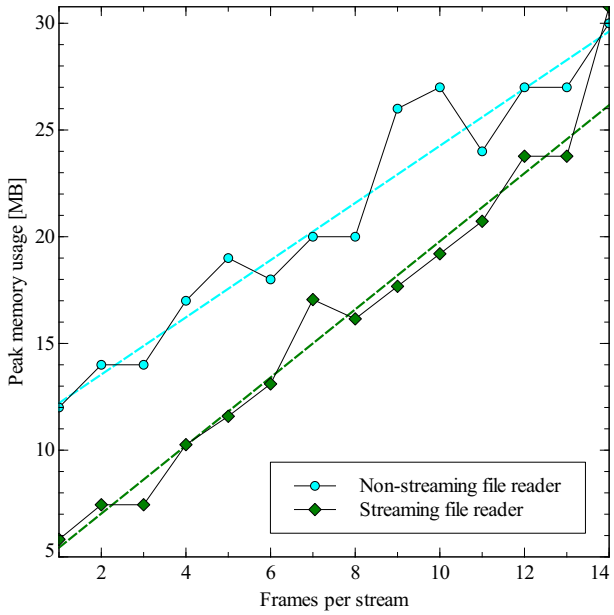


Figure 3: Peak heap memory usage versus image frames per stream during scan conversion. Two curves are given: one for an `itk::ImageFileReader` with an `ImageIO` capable of streaming, and one not capable of streaming. The slope and intercept for a fit to the streaming and non-streaming data are 1.3 $MB/frame$ and 10.9 MB, and 1.6 $MB/frame$ and 3.8 MB, respectively.

3.3 Results

Streaming behavior was verified by measuring peak memory usage with Valgrind's Massif tool[13] (Figure 3).

Figure 4 demonstrates successful scan conversion when imaging a phantom with anechoic spheres. This image is very large, 5414x573x192 voxels, and could not be processed in memory without streaming.

4 Discussion

In this article we described extensions to the InsightToolkit that allow creation of ultrasound B-Mode images. The implementation was fast relative to a version written in Matlab™, and was capable of streaming even for images that required scan conversion. The ability to perform streaming is increasingly important as 3D ultrasound machines become prevalent. Profiling was not performed, and improved throughput may be possible by optimizing the log filter or replacing some of the simpler filters with `itk::ImageAdaptor`s. The included code may be useful for research or the basis for a clinical scanner's software. Initial achieved frame rates, 300 fps, are higher than what is perceived as real-time by a human observer, 15-30 fps, and the rate a typical clinical scanner collects an image, up to approximately 80 fps. However, these rates were achieved while monopolizing system resources on a desktop workstation. Resources are needed for other system tasks, and real-time imaging requires consistency not examined here. Furthermore, the trend in the medical ultrasound industry is towards compact, low power systems, which increases the need for computational efficiency.

Results in Figure 2 suggest Fourier Transform calculation was not rate limiting as expected. This is evi-

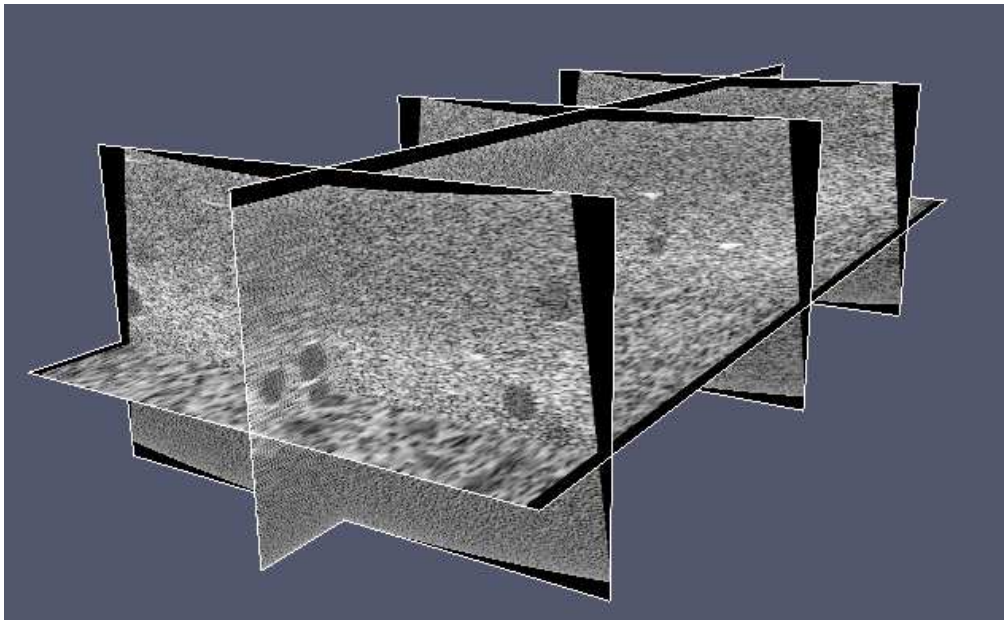


Figure 4: Large, scan converted B-Mode image of anechoic spheres in a tissue mimicking ultrasound phantom. The image was processed without swapping to disk.

dent from the consistent relative performance across multi-threaded (FFTW and VNL), and single-threaded (OpenCL) FFTs while different maximum thread counts were utilized.

The best performing FFT was FFTW, followed by VNL and OpenCL. Surprisingly, OpenCL throughput operating on the GPU was poorer than pure-CPU FFTW and VNL implementations. While GPUs have good arithmetic computational capacity, high performance is not obtained without careful attention to the hierarchy of memory transfers that take place on the hardware and the bandwidth at each level. The first and last transfers that must take place is from target to host and host to target, which are expensive operations. Two sets of the transfers take place in `itk::BModeImageFilter`: one set for the forward transform and one for the inverse transform. These transfers may have nulled any parallel computing benefit even for the arithmetically intensive task of computing the Fourier Transform. OpenCL holds promise to bring cross-platform, hardware vendor independent highly parallel computing to ITK, but simply replacing select filters with an OpenCL version will not bring performance improvements because of limited CPU-GPU bandwidth. The image buffer must remain on the target device throughout pipeline execution. One possible solution would involve creating an `itk::OpenCLImage` that allocates its buffer with OpenCL. The infrastructure and work that has gone into existing `itk::ProcessObject`s could be maintained through inheritance. An inherited `ProcessObject` could also inherit from an OpenCL class providing an OpenCL Context, and it would need to provide a `GenerateData()` method that launches an OpenCL kernel on the target.

The `itk::ResampleImageFilter` plays a pivotal role in ITK; it is central to resampling, transforms, and registration. The `itk::StreamingResampleImageFilter` presented makes streaming possible in many more situations. However, there are significant limitations to the transforms it can operate with it. If it is questionable whether a transform can be used with `itk::StreamingResampleImageFilter`, probing a transform's `IsLinear()` method generates insight– if it returns `false`, the transform may not work with `itk::StreamingResampleImageFilter`.

References

[1] The Insight Segmentation and Registration Toolkit. URL http://www.itk.org/. 1

[2] Felsberg, M. and Sommer, G. The monogenic signal. *IEEE Transactions on Signal Processing*, 49(12):3136–3144, 2001. ISSN 1053587X. doi:10.1109/78.969520. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=969520. 2.1, 1, 2.1

[3] Woo, Jonghye, Hong, Byung-Woo, Hu, Chang-Hong, Shung, K. Kirk, Kuo, C.-C. Jay, and Slomka, Piotr J. Non-Rigid Ultrasound Image Registration Based on Intensity and Local Phase Information. *Journal of Signal Processing Systems*, 54(1-3):33–43, 2008. ISSN 1939-8018. doi:10.1007/s11265-008-0218-2. URL http://www.springerlink.com/index/10.1007/s11265-008-0218-2. 2.1

[4] Bracewell, Ronald N. *The Fourier Transform and Its Applications*. McGraw-Hill, 3rd edition, 2000. ISBN 0-07-303938-1. 2.1

[5] Wagner, Robert F, Smith, Stephen W, Sandrik, John M, and Lopez, Hector. Statistics of Speckle in Ultrasound B-Scans. *IEEE Transactions on Sonics and Ultrasonics*, 30(3):156–163, 1983. 2.1

[6] Frigo, Matteo and Johnson, Steven˜G. The Design and Implementation of {FFTW3}. *Proceedings of the IEEE*, 93(2):216–231, 2005. 2.2, 2.2

[7] VXL: C++ Libraries for Computer Vision Research and Implementation. URL http://vxl.sourceforge.net/. 2.2

[8] Lehmann, Gaetan. FFT based convolution. *Insight Journal*, (January-June), 2010. URL http://hdl.handle.net/10380/3154. 2.2, 2.2

[9] Khronos Group. OpenCL - The open standard for parallel programming of heterogeneous systems. URL http://www.khronos.org/opencl/. 2.2

[10] OpenCL_FFT. URL http://developer.apple.com/mac/library/samplecode/OpenCL_FFT/index.html. 2.2

[11] Volkov, Vasily and Kazian, Brian. Fitting FFT onto the G80 Architecture, 2008. URL http://www.cs.berkeley.edu/~{}kubitron/courses/cs258-S08/projects/reports/project6_report 2.2

[12] Govindaraju, Naga K., Lloyd, Brandon, Dotsenko, Yuri, Smith, Burton, and Manferdelli, John. High performance discrete Fourier transforms on graphics processors. *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, (November):1–12, November 2008. doi:10.1109/SC.2008.5213922. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5213922. 2.2

[13] Massif: a heap profiler. URL http://valgrind.org/docs/manual/ms-manual.html. 3.3