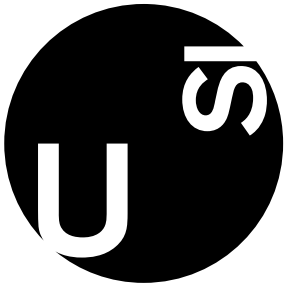


Università
della
Svizzera
italiana

Faculty
of
Informatics

Bachelor Thesis

June 26, 2023

Rotation of multi-dimensional signals with spectrally accurate schemes

Dylan Reid Ramelli

Abstract

The present work is concerned with rotations of three-dimensional digital signals, sampled evenly on Cartesian grids. Such settings are ubiquitous across the field of sensing, in particular imaging. Arbitrary rotation of 3D scalar fields represents a cornerstone of multi-modal imaging, where multiple signals are aligned under rigid body transformations. In this work we use them to compute light attenuation within large-scale volumetric dataset, for visualization purposes. We revise the foundations laid by Unser in 1995 [**?**] on how to accurately carry out 2D rotations on 2D images, and extend it to 3D rotations on volumetric images. Relying on classical digital filter design, we devise novel computational schemes that are algorithmically efficient while allowing the underlying computational workload to be effectively mapped onto contemporary processing microarchitectures. Achieving close-to-ideal performance on such applications remains nonetheless a challenge. Considerations about the careful software implementation of such schemes are therefore included in the present document.

Advisor
Prof Rolf Krause
Co-Advisors
Dr Diego Rossinelli, Dr Patrick Zulian

Advisor's approval (Prof Rolf Krause): Date:

Contents

Figure 1. Large-scale in-silico investigation of homeostasis in the subarachnoid space of the human optic nerve [Rossinelli et al. 2023, in-preparation]. Homeostasis is directly related with the interaction between the cerebrospinal fluid flow (not shown here) and the meningeal surface. Blue denotes poor levels of homeostasis, whereas red denotes sufficient material exchange between fluid and structure.

1 Introduction and Motivation

Sustained over the last decades, an exponential growth of acquisition rates across the wider field of sensing technology have been observed. While this unprecedented volume of high-quality digital signals holds the promise of critically assisting scientific inquiry at large, it poses grand technical challenges. To address them, we wish to develop and employ computational schemes able to extract insight buried in data by leveraging the large-scale data processing infrastructure available today. Data analysis takes place across the entire spectrum of abstraction: from low-level signal processing, all the way up to computational modeling relating experimental data against simulations, as exemplified in Figure 1 (courtesy of Rossinelli et al.[**?**]).

Whilst scientific visualization might be perceived as primitive, it is nonetheless an essential tool with which to reveal insight from 3D data sets, as demonstrated in Figure 2 which was provided by [**?**] and [**?**]. To better understand signals, we calculate how light is transported within the data volume. Massive amounts of light rays are cast from the sky, from every directions. To obtain accurate results in reasonable time on Terabytes, or hundreds of Gigabytes of data, the associated computational workload must be carried out at high fraction of nominal peak of the underlying computational infrastructure.

For the purpose of this work, radiative transfer governing the transport of light is limited so as to only consider light attenuation, by assigning a linear mapping from grayscale signal intensities to light absorption values.

Herein high-performance high-accuracy rotations of volumetric images are relied upon to align data with arbitrary directions of rays that are cast. Although 3D rotations are employed herein for a specific purpose, we emphasize that such operations are ubiquitous across most disciplines relating to imaging. For example, rotations are fundamental in multi-modal imaging, to align different signals of the same specimen under rigid body transformations. Typically solved in a supervised learning context, data augmentation for machine vision tasks is another example where fast but accurate 3D rotations are essential.

2 Prior Work and Methods

This work seeks to find compromises between algorithmic efficiency, which lower the number of required operations for a certain task, and high-throughput processing schemes, which maximizes the number of operations that can be performed over a certain period of time. More often than not, algorithmic efficiency and execution efficiency are conflicting goals. Careful considerations are hence required to successfully combine the two in order to aggressively

Figure 2. Two different types of data analysis, on the same data set. Gray-scale slice of the subarachnoid space of the optic nerve (left) and close-up view of the subarachnoid space of the optic nerve (right). The signal is the same, but the two data processing approaches lead to qualitative different insight.

decrease execution time.

2.1 Factoring 3D Rotations into 2D Rotations

We rely on Euler angles to represent arbitrary 3D rotations. To apply a rotation on a volumetric dataset we therefore perform a series of 2D rotations on the volumetric signal, in different directions. Each 2D rotation is carried out to each slice of the volume, independently. While such approach would already expose thread-level parallelism, abundantly available on CPUs and GPUs, it would lead to data access inefficiencies due to lack of spatial locality (e.g., while performing rotations on the XZ-plane). This issue is mitigated with global data transpositions, thereby ensuring that 2D rotations are always performed on the fastest varying indices of the dataset. While the cost of data transposition is not negligible, the underlying computational patterns expose enough regularity to be performed very efficiently on contemporary hardware including some of the latest ASICs [**?**].

We rely on the work of Unser [**?**] to implement 2D image rotations by resampling the translation of 1D signal (as shown in Figure 3), independently for each line of each slice. This allows us to express rotations with schemes that expose both instruction-level and data-level parallelism, while maintaining close-to-ideal spatiotemporal locality.

2.2 Factoring 2D Rotations in 1D translations

Unser et al. [**?**] have shown that a rotation transformation,

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$

can be factored into three matrices each of which represents a shearing of the image in a cardinal direction:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} 1 & -\tan\theta/2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ \sin\theta & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -\tan\theta/2 \\ 0 & 1 \end{pmatrix}. \quad (1)$$

The intermediate results within the factorization are shown in Figure 3. A remarkable property of the shear matrices is that they involve no scaling: the diagonal entries of the matrices are the unity. This means that no frequencies are created or lost, as upsampling and downsampling are avoided, respectively.

The first matrix shears the image in the x direction by $\Delta_x = -y \cdot \tan(\theta/2)$, the second matrix shears the image in the y direction by $\Delta_y = x \cdot \sin(\theta)$ and the last matrix shears the image again in the x direction by Δ_x , as illustrated in (3).

We use data transpositions to ensure that 1D translations are performed only on the fastest varying indices of a given line thereby enforcing ideal spatial locality during this computational stage. Therefore we temporarily swap rows with columns, whenever needed.

This work is focused on the design and implementation of computational compact schemes that resample 1D signals translated by a (constant) fraction of the sampling distance.

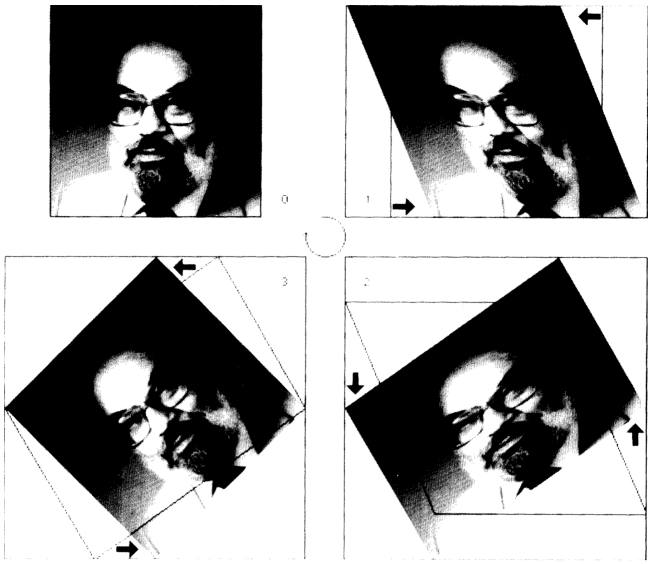


Figure 3. Rotation of an image is carried out by cascading translation operations, courtesy of Unser et al. [**?**].

2.3 Resampling Translation with Finite Impulse Response (FIR) Filters

In the work of Unser et al. [**?**], translation of each row (resp. each column) is carried out by evaluating an interpolation scheme:

$$f(x-h) \approx \sum_i c_i \phi(x-h-x_i), \tag{2}$$

where h is the translation parameter (also called "shift" in the present document), f is the input signal, ϕ is the reconstruction kernel, and c_i are the interpolating coefficients.

As reconstruction kernel, Unser et al. [**?**] consider B-splines of order 1, 3, 7. Except for the linear B-spline, which is interpolating by design, higher order B-splines are non-interpolating (in fact they are low-pass filters), and thus one has to solve a system of interpolating equations:

$$f_i = f(x_i) = \sum_j c_j \phi(x_i - x_j), \tag{3}$$

with $x_i = i$. The system size is the size of the input signal, and the latter can be in the order of ten thousand samples. Inverting a linear system of equations $Ax = b$ of such sizes may incur non-negligible computational costs.

In their work, Unser et al. [**?**] organize the computational workload of translating a 1D signal in two separate stages: computing the interpolating coefficients, and resample translated signals by evaluating the interpolation scheme at the sample locations.

When B-splines are used as reconstruction kernels, the system can be inverted very efficiently [**?**] by cascading a few infinite impulse response (IIR) filters. Despite such algorithmic advantages, the linear system remains global. This is somewhat disappointing: the resampled translation of the signal at a given point will "see" only a limited set of adjacent points.

If ϕ has compact support, why then try to solve a global system ? This issue motivates the research on alternative computational schemes that are completely local. Herein we opt to resampling translated 1D signals by carrying out direct convolution with filters of compact support, and avoiding solving global system of equations in the first place. The type of filters sought herein are finite impulse response (FIR). Direct convolution of a FIR filter h to an input signal x reads:

$$y(n) = \sum_{k=A}^{B} h(k)x(n-k) = (h * g)(n). \tag{4}$$

While direct filtering (carried out by direct convolution) is generally seen as inefficient, it can be carried out at outstanding performance if the filter is reasonably small, when carefully implemented on contemporary microarchitectures. Firstly, direct convolution maps independent workloads to each output entry. When carefully implemented, it exposes instruction-level parallelism which can be discovered at runtime by superscalar microarchitectures featuring out-of-order execution. Secondly, the regular access pattern of direct convolution allows to expose data-level parallelism in the form of single-instruction multiple-data operations. Thirdly, the abundance of independent workload can be leveraged by thread-level parallelism. In addition, direct convolution with small filters may bring crucial

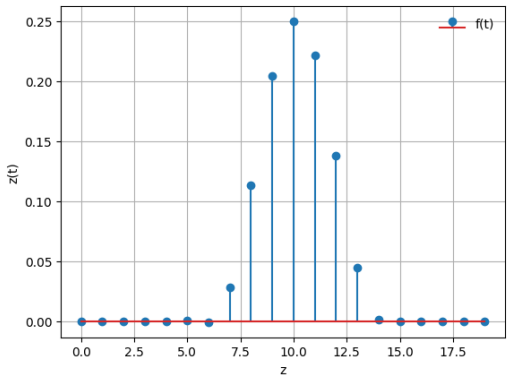


Figure 4. Example of a FIR filter $z(t)$ that we will create.

advantages on specific computing architectures such as digital signal controllers, or the A100 GPU, where one can use the tensor cores to carry out a block-sparse form of the direct convolution in single precision [**?**].

Our goal is therefore to design a FIR filter of relatively compact support (e.g., 20 - 40 coefficients) that will be used to shift a signal by any real value by direct convolution, while also allowing us to outperform both in terms of accuracy and efficiency the approach proposed by Unser et al. [?].

The technical strategy we embrace is as follows:

- We design our filter in Fourier space for any frequency in $[-\pi/2, +\pi/2[$ (DTFT space).
- We digitize it for the number of coefficients we desire (DFT space).
- We obtain the filter coefficients in the time domain by performing an inverse DFT (IDFT).

We would like to mention that the coefficients of a FIR filter is also its impulse response.

The target filter, exemplified in Figure 4, is obtained by cascading two filters:

- $H(\omega)$, the filter to carries out a spectral shift of the signal, thus leveraging the perfect interpolation property available in Fourier space.
- $F(\omega)$ filter mitigates the Gibbs phenomenon, arising from the truncation of the Fourier series to a finite size. This issue is more apparent in smaller filters than larger ones.

2.4 Design of the Translation Filter $H(\omega)$

For a digital signal of infinite length, an arbitrary shift δ applied on the signal can represented exactly with the transfer function:

$$H(\omega) = e^{-i\delta\omega}. \tag{5}$$

The (forward) DFT of a digital 1D signal f reads:

$$F_m = \sum_{k=0}^{N-1} f[k] e^{-i\frac{2\pi}{N}km},$$

where N the number of samples and m the wave number. We approximate "$f[k-\delta]$" where δ is the amount we want to shift:

$$\begin{aligned} Z_m &= \sum_{k=0}^{N-1} f[k-\delta] e^{-i\frac{2\pi}{N}km}, \\ &= \sum_{r=-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}(r+\delta)m}. \end{aligned}$$

We take the common term outside the summation:

$$\begin{aligned} Z_m &= e^{-i\frac{2\pi}{N}\delta m} \sum_{r=-\delta}^{N-1-\delta} f[r] e^{-i\frac{2\pi}{N}rm}, \\ &= e^{-i\frac{2\pi}{N}\delta m} \sum_{k=0}^{N-1} f[k] e^{-i\frac{2\pi}{N}km}. \end{aligned}$$

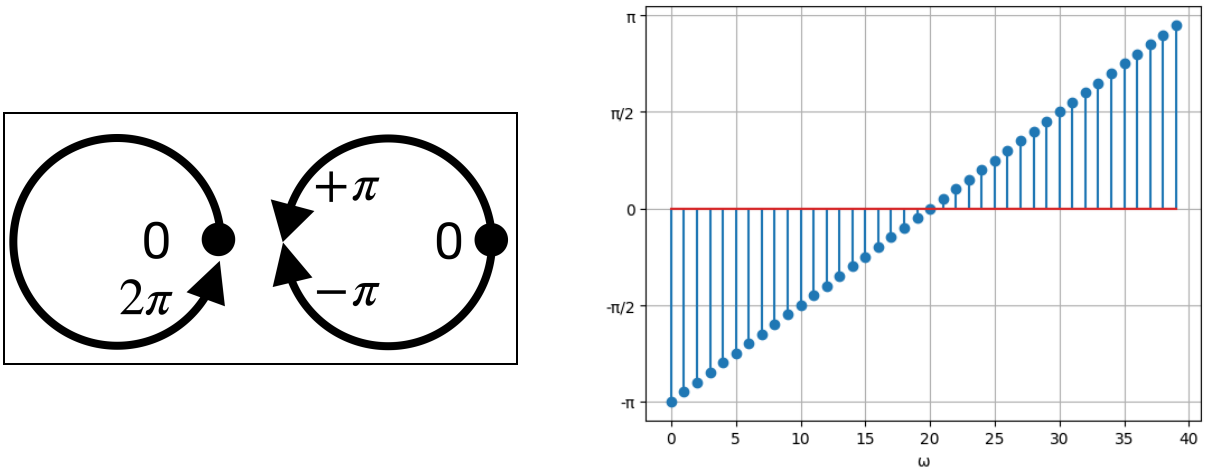


Figure 5. Different enumeration of the discrete frequencies (left), and correct enumeration for translating signals (right), function on $n = 40$ samples.

Therefore, the transfer function of the shift filter in the DFT space reads:

$$H(\omega_m) = \frac{Z(\omega_m)}{F(\omega_m)} = e^{-i\delta\omega_m}, \tag{6}$$

with $\omega_m = \frac{2\pi}{N}m$.

The last equation works well for any integer shifts that we want to perform on our signal but for any fractional amount we encounter some problems with the use of the correct frequency indexes. The small diagram in Figure 5 depicts this problem on the left and our solution on the right. We need to take into consideration the negative frequency indexes for fractional shifts in order to maintain the Hermitian symmetry of the spectrum of real-valued signals. As such we define a function called wave_n that maps indices to the correct wave number:

$$\text{wave\_}n(m) = (m + \lfloor N/2 \rfloor) \bmod N - \lfloor N/2 \rfloor \tag{7}$$

This equation will assign each frequency it's correct index (Figure 5).

2.5 Integer Shifts

Any shift s on the input signal can be decomposed into integral and fractional parts $k = \lfloor s \rfloor$, $\delta = s - \lfloor s \rfloor$, respectively. Shifting a signal by k is a trivial operation, as it boils down to moving the array by k entries. Therefore, if the shift does not have a fractional component then we skip the convolution with the filter entirely.

2.6 Design of the Smoothing Filter $F(\omega)$

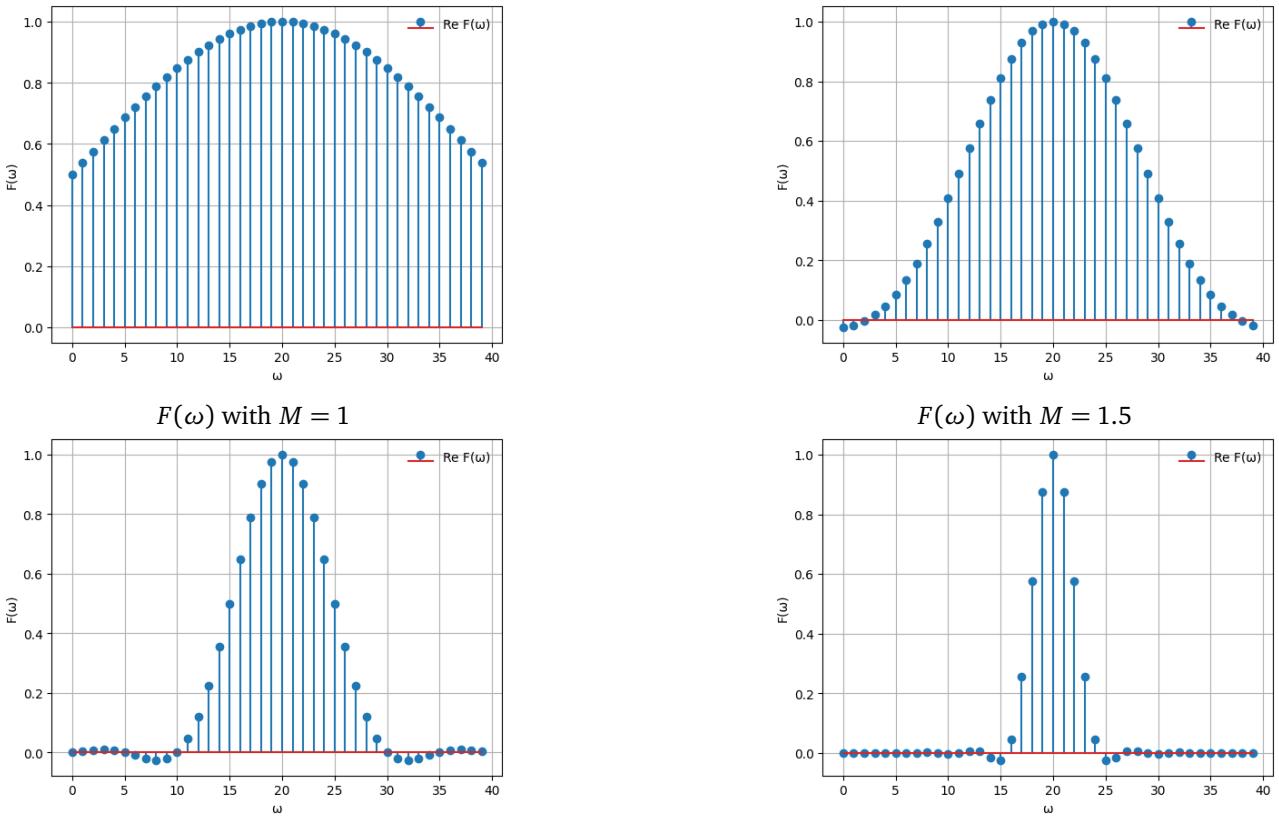
For a digital signal of infinite length, an arbitrary shift δ applied on the signal can represented exactly with the transfer function $H(\omega) = e^{-i\delta\omega}$. When dealing with finite signals of N samples, however, the frequency spectrum is sampled into N frequency bins. In such settings, any δ that is not a multiple of $1/N$ would lead to spurious oscillation due to the Gibbs phenomenon. The strongest spurious oscillations are obtained for $\delta = 0.5$.

The goal of the smoothing filter is therefore to attenuate the oscillations that occur. We rely on an approach inspired from the Lanczos smoothing kernel [**?**]. While the Lanczos analog filter in the time domain is a tiny box covering 4 samples, our analog filter g is the Dirac function, mollified over $2M$ samples:

$$g(t) = \frac{1}{2} + \frac{1}{2}\cos(\frac{\pi t}{M}), \tag{8}$$

with M chosen between 1.5 and 3. Along the same approach of Lanczos smoothing, we perform a Continuous Fourier Transform (CFT) of our filter and resample it to obtain its Discrete Fourier Transform (DFT) representation, and combine it with the DFT version of $H(\omega)$. The Continuous Fourier transform (CFT) on the cosine part of g reads:

$$F(\omega) = \int_{-M}^{M} \cos(\frac{\pi}{M}t)\mathrm{e}^{-\mathrm{i}\omega t}\mathrm{d}t. \tag{9}$$



$F(\omega)$ with $M = 1$ $F(\omega)$ with $M = 1.5$

$F(\omega)$ with $M = 2$ $F(\omega)$ with $M = 3$

Figure 6. Compact schemes of different M for local computation.

This equation allows us to define our CFT in an interval M, that we choose based on the fractional amount we want to shift. The parameter M is useful for us to fine tune the filter and choose how big of a smoothing effect we want:

$$F(\omega) = \frac{2M^2\omega \sin(M\omega)}{M^2\omega^2 - \pi^2}. \tag{10}$$

Using this function we can build filters of different M sizes(Figure 6).

For better accuracy we generally choose our interval M to be small when we have a small fractional shift and big when we have a shift that gets closer to 0.5. We can write a simple expression that defines M in respect to the fractional shift x to be:

$$M = \begin{cases} 1.5 & \text{if } 0.1 < x < 0.25 \\ 2 & \text{if } 0.3 < x < 0.5 \end{cases}$$

With some experimenting we found that the filter with $M = 3/2$ is a good compromise between eliminating the oscillations and translating the signal as accurate as possible.

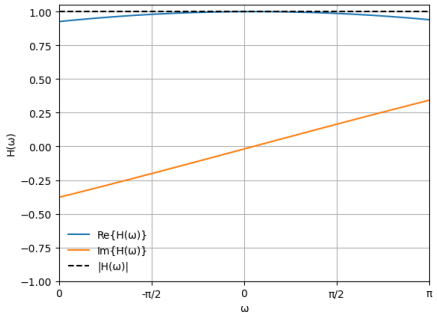
We will create one of size 40 with $M = 2$. We then follow equation (10), while also paying attention to evaluating the limit when $\omega = \frac{\pi}{M}$ or $\omega = -\frac{\pi}{M}$. This results in $F(\omega)$ which we plot in Figure 7.

Now let us say that the fractional amount to shift is 0.1234. We then multiply our smoothing filter $F(\omega)$ by our shift filter $H(\omega)$ (Figure 7), which gives us $Z(\omega)$.

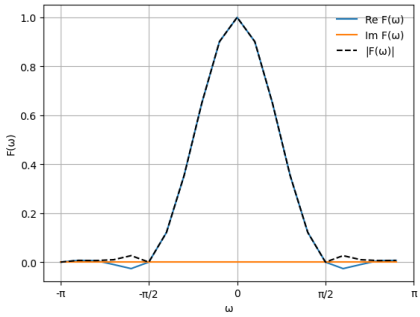
Finally, we carry out the Inverse Discrete Fourier Transform (IDFT) of the digitized $H(\omega)F(\omega)$ to obtain the coefficients of the target FIR filter in the time domain $z(t)$, as already depicted in Figure 4.

3 Software Implementation Details

The computational schemes discussed herein have the advantage of being data-independent. That is, they lead to opcode streams that contain no branching instructions that depends on the input data (except for the input and filter lengths). This provides two key advantages. Firstly, it allows us to benefit from static performance analysis. Secondly, the synthesis of C kernels can be straightforwardly automated in a programming language with a higher abstraction.



$H(\omega)$ Filter with $M = 2$



$F(\omega)$ Shift filter with $\delta = 0.1234$

Figure 7. The two filters $H(\omega)$ and $F(\omega)$.

```c
#include <stddef.h>

void kconv(
    const float w0,
    const float w1,
    const float w2,
    const float w3,
    const float w4,
    const float w5,
    const float w6,
    const float w7,
    const int n,
    const float * in0,
    const float * in1,
    const float * in2,
    const float * in3,
    const float * in4,
    const float * in5,
    const float * in6,
    const float * in7,
    float * out)
{
    for(int i = 0; i < n; ++i)
    {
        out[i] =
        w0 * in0[i] +
        w1 * in1[i] +
        w2 * in2[i] +
        w3 * in3[i] +
        w4 * in4[i] +
        w5 * in5[i] +
        w6 * in6[i] +
        w7 * in7[i] ;
    }
}
```

images/asm-listing.png

Figure 8. Static assessment with *Godbolt Compiler Explorer* of a direct convolution with an 8-coefficients filter on an arbitrary long digital signal. C kernel (left), successfully compiled for AVX2 enabled microarchitectures (right, top), and microarchitectures featuring AVX-512 SIMD instructions(right,bottom).

In our case C kernels are generated by a python script executed at compile time by Makefile. The only parameter we define is the desired filter size. The script will generate a header and source file in C implementing the direct convolution.

To assess our one-dimensional direct convolution kernel, we use the *Godbolt Compiler Explorer* [**?**]. This tool allows us to assess how effectively instruction-level and data-level parallelism are exploited by the compiler at hand. As stated in the introduction, what we are looking for is a balance between algorithmic efficiency and execution efficiency, whereby increasing the number of instructions per cycle while lowering the number of operations. This is depicted in Figure 8, targeting two different microarchitectures. From the signature of the C kernel we note that the single instruction stream is aliased into multiple input pointers (shifted by consecutive entries). This is to direct the compiler to the full SIMDization of the compute kernel. On the right hand side of Figure 8 we note how well the compiler can transform the computational scheme into AVX and AVX-512 instructions. The AVX-512 instruction stream is exceptionally compact, featuring 1.5 FLOP/instruction and thus able to attain 75% of the nominal peak performance, when not memory-bound.

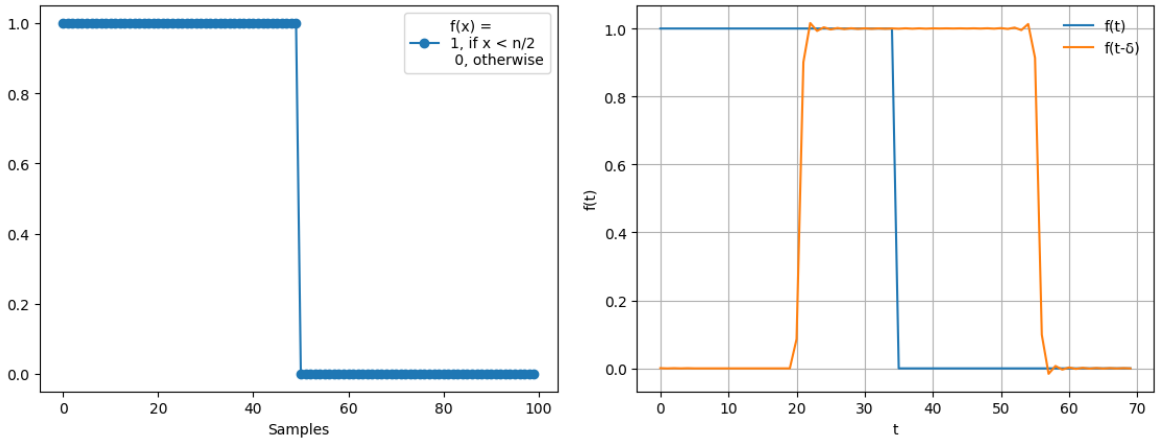


Figure 9. Original signal of size $n = 100$ samples (Left) shifted by $\delta = 20.01$ with $M = 1$(Right).

4 Results

4.1 Results with 1D signals.

One-dimensional signals are the starting point of our tests and as such we chose the reversed unit step, a pretty aggressive signal to test (Figure 9). For a signal of $n = 100$ samples the reversed unit step reads:

$$g(x) = \begin{cases} 1 & \text{if } x \leq \frac{n}{2} \\ 0 & \text{otherwise} \end{cases}$$

By taking this signal and shifting it by 20.25 we get a different result depending on the M chosen when creating the filter (Figure 10).

At first glance we can see that with $M = 1$ we have many oscillations present in the resulting signal and with $M = 3$ the sharp features of the signal are completely smeared out. We can see though in the right of Figure 9 that if we set $M = 1$ and the fractional value is small, for example 0.01 we have smaller oscillations and the output signal preserves well the features of the input signal.

This result is attributed to the fact that the filter contains fewer oscillations as well. If we take a look at the filter defined for $M = 1$ we can see that we have the strongest oscillations at 0.5 (Figure 11). This is why for a fractional shift that gets closer to 0.5 it could be beneficial to consider a larger value of M, such as $\frac{3}{2}$ or 2, in order to damp out the oscillations.

5 Conclusion and Outlook

Information Technology (IT) has reached unprecedented rates of data acquisition and processing power. Data visualization is a crucial component in IT-assisted scientific inquiry, as it is employed both for gaining the initial insight from the acquired data and for the qualitative assessment of the results obtained by after computational modeling. High quality visualization relies on approximating how light interacts with volumetric structures, and such calculations can be simplified by leveraging 3D rotations.

The work presented herein is motivated by the need of processing of large volumes of data by leveraging contemporary distributed computing infrastructure. As such, it seeks to find computational patterns that leads to very high accuracy while executing exceptionally well on the latest computing microarchitectures. Following this direction, this report has elaborated the design and application of relatively small filters that translate large volumetric datasets to accurately rotate them in Euclidean space. Our approach implements 3D rotations by shifting the “1D lines” of the dataset by fractional amounts, and executes very well (i.e., at high fractions of the nominal peak) on contemporary CPU microarchitectures. Future work includes the extension of such filtering techniques on GPUs. Moreover, the choice of M for the filter could be chosen automatically based on the fractional amount we want to rotate.

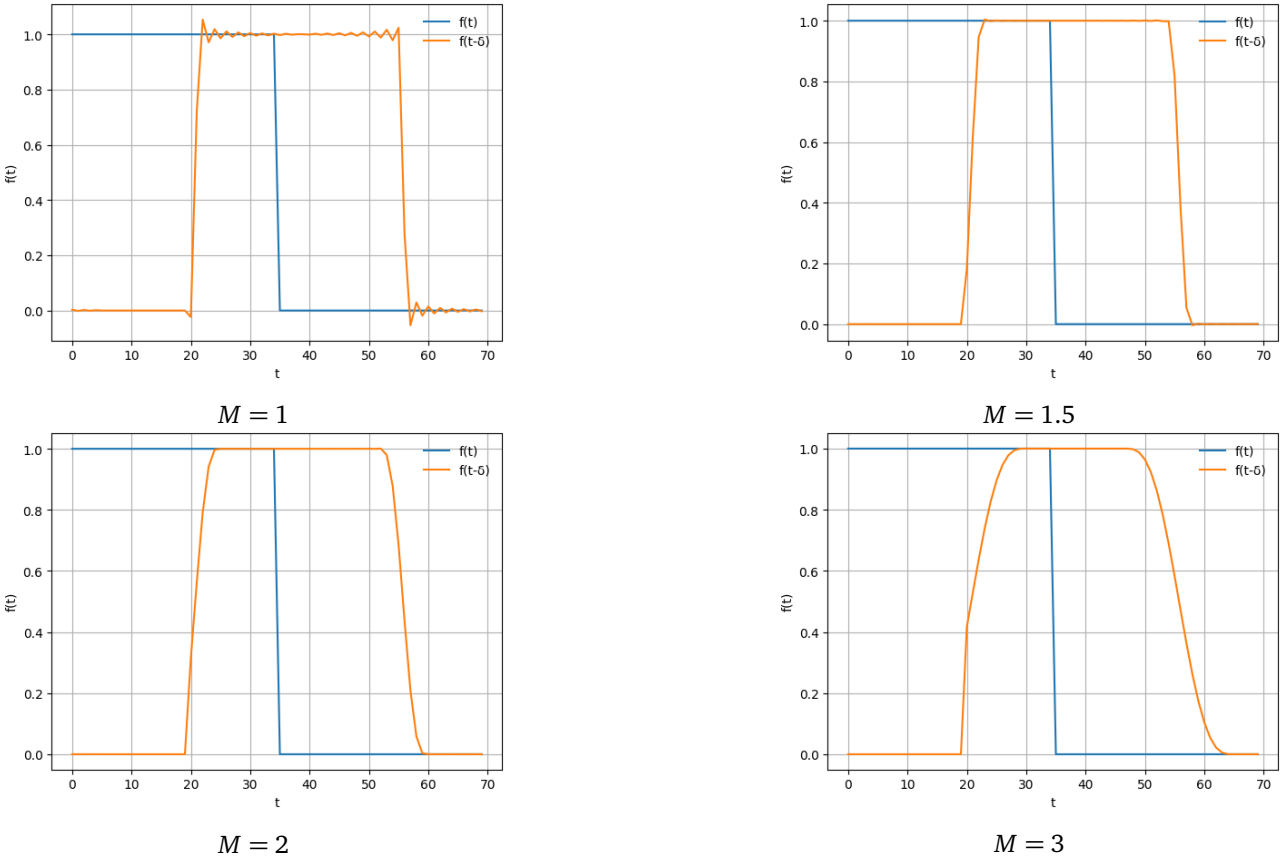


Figure 10. Results of shifting the original signal $g(x)$ by $\delta = 20.25$ with different sizes of M.

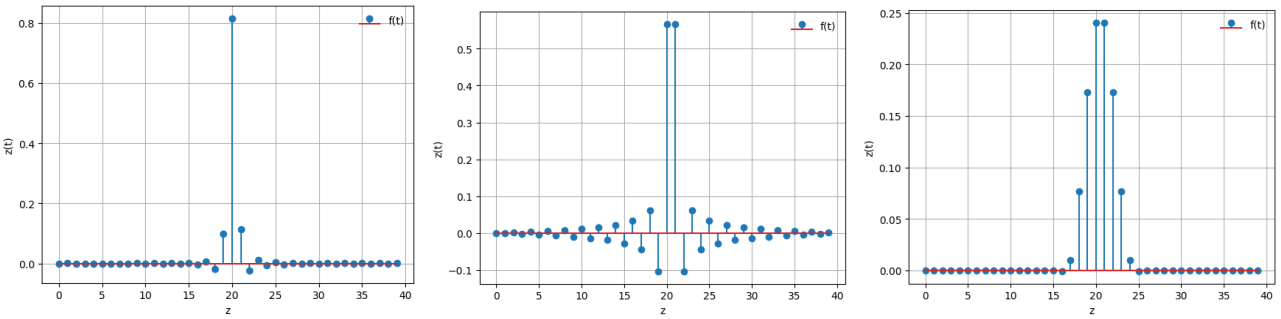


Figure 11. Difference in oscillations with varying fractional shifts. $F(\omega)$ shifted by 0.01 with $M = 1$ (Left), $F(\omega)$ shifted by 0.5 with $M = 1$ (Middle) and $F(\omega)$ shifted by 0.5 with $M = 2$(Right).

Supplementary Information

Numpy and Matplotlib were a huge help in visualizing the results of the Fourier Transform and of the shifts. These libraries were also useful for when we started writing code in C. Where we would output the binary result to a file and then read it and plot the result in python. There are a few plotting libraries for C like the gnuplot library but we felt that this was just the faster/easier solution. The whole project started out as a CMake and this was chosen mainly because of familiarity with this tool and teh ability to quickly setup an environment that would build the executable. However, towards the end used a plain Makefile to streamline the work. The added advantage of the makefile was to enable running a python script at compile time to create other files needed by the main target.

Jupyter Notebook was used extensively to test our method and quickly plot the results, while also documenting everything so that it would be easier to complete this report. Since we embraced a bottom-up approach we were able to re-use parts of the code to test some other aspect of the shift. We started with simple shifts of a 1D signal by an integer value which we then attempted to shift the same signal by a fractional amount. At this point some problems arose with the usage of the correct frequency index but in code was easily implemented it with a basic method. The source code of this project is available at: `https://github.com/DylanReidRamelli/Bachelor-Project-2023`. The tools that were used: MakeFile, CMake, Jupyter Notebook and some built-in libraries of C and Python like numpy and matplotlib.

Acknowledgments

I would like to express my deepest appreciation to Dr Diego Rossinelli without whom this endeavour would have been impossible to complete. Next I would like to express my deepest gratitude to Dr Patrick Zulian for his belief in my ability to complete such a project and finally, my sincere gratitude to Prof Rolf Krause for his invaluable feedback and support.