SLIDING IS SMOOTHER THAN JUMPING

Russell Bradford *Richard Dobson* *John ffitch*

Department of Computer Science, University of Bath, England

ABSTRACT

The existence of the sliding DFT has been known for some time, but it does not seem to be in wide use, possibly because of its perceived computational cost. In this paper we review the mathematical background, and implementation issues, and then consider the advantages and disadvantages of the Sliding Discrete Fourier Transform (SDFT) as compared with a more traditional FFT algorithm. We also propose a much more efficient Simple Sliding Inverse DFT that makes sliding a serious alternative to jumping between overlapping frames. Finally we assess the quality of transformations based on the SDFT in a Csound implementation.

1. INTRODUCTION

In his visionary lecture on audio in the new millennium, Moorer presented a number of ways in which he thought that audio would progress [6]. The one which inspired this work is the suggestion that with increase in machine power available eventually processes he calls the “Running” transforms will come into use. In particular he represents the discrete Fourier transform from his earlier paper [5] where he shows that this is computationally efficient. We reiterate the mathematics in the section below in order to identify the advantages and problems, and suggest an improvement.

We also make qualitative assessments of the audio results that can be obtained from a DFT with a maximal window overlap, using our implementation in Csound and comment on the vision of Moorer.

2. THE SLIDING DFT

The Discrete Fourier Transform is a translation of a function periodic over some period from the discrete time domain to the discrete frequency domain. In practice it is usual to apply this analysis to a windowed section of a signal and assume that it is periodic. Adopting the notation of Moorer, we have a function $f(n)$, with discrete samples $f_0, f_1, \ldots$. We assume a window size of N, which is to say that we assume that the signal repeats over that period

The DFT, starting at time t is

$$F_t(n) = \sum_{j=0}^{N-1} f_{j+t} e^{-2\pi i j n/N},$$

where $F_t(n)$ is the value in the nth bucket in the frequency domain. The standard algorithm to calculate the coefficients $F_t(n)$ is the Fast Fourier Transform (FFT) which uses a divide-and-conquer method and costs $N\log(N)$ operations to calculate for a window of size N.

In the traditional implementation the window is moved on by M samples, usually with $M < N$, and the DFT is recalculated.

The idea behind the Sliding Discrete Fourier Transform (SDFT) is to make use of the known values of $F_t(n)$ to calculate the value for the next window. In particular we assume than we are moving by 1 sample only:

$$\begin{aligned}
F_{t+1}(n) &= \sum_{k=0}^{N-1} f_{k+t+1} e^{-2\pi i k \frac{n}{N}} \\
&= \sum_{k=1}^{N} f_{k+t} e^{-2\pi i (k-1) \frac{n}{N}} \\
&= \left(\sum_{k=0}^{N-1} f_{k+t} e^{-2\pi i k \frac{n}{N}} - f_t + f_{t+N} \right) e^{2\pi i \frac{n}{N}} \\
&= \left(F_t(n) - f_t + f_{t+N}\right) e^{2\pi i \frac{n}{N}}
\end{aligned}$$

That means that in order to forget the oldest sample and accept a new sample, all that is needed for the n^{th} frequency amplitude is an addition, a subtraction and a complex rotation by $2\pi n/N$. As we know that the samples are real this is a little simpler than might appear.

In complexity terms this creates the transform one sample later in $O(N)$.

There remains the problem of starting, but if we assume that the signal was silent until the first sample, then the transform is also zero and so we can slide the samples in one at a time without creating an initial FFT.

Of course to move on by N samples is an N^2 process, but we have gained simple access to all the transforms, and also there is nothing in this algorithm that requires a power of 2 for the window size as required by the FFT.

3. INITIAL IMPLEMENTATION

In evaluation a process the proof is largely in the implementation in addition to the mathematics. We have made a number of test implementations, but we ended with two programs embodying this method; the first as a test for errors and stability, and the second as a potentially useful program.

The implementation of the algorithm above is simple. The stand-alone program makes no attempts to economise on space and is a simple translation. The only issue is the complex rotation (multiplication by $e^{2\pi in/N}$). This would appear to require the calculation of N sines and cosines, which could be expensive, although for a given N they can be pre-calculated. An alternative is to use the formulae $\cos((n+1)x) = \cos(x)\cos(nx) - \sin(x)\sin(nx)$ and $\sin((n+1)x) = \sin(nx)\cos(x) + \cos(nx)\sin(x)$ so only a single sine and single cosine of $2\pi/N$ are needed. We have experimented with both schemes, and with a hybrid which recalculates the sine and cosine after a number of steps. The incremental formula does not suffer from much accumulated error and is our preferred method.

The program was tested against a standard FFT implementation, and performed correctly as expected. Of course, when comparing with the FFT algorithm, N has to be set to a power of 2, which is not required for the SDFT, where N can be any size.

4. THE SLIDING INVERSE DFT

An nice feature of the FFT algorithm is that there is an inverse, taking the frequency components to the samples, which is very similar to the forward process, and can reuse some of the code. Unfortunately in Moorer's articles this does not seem to be the case for the SDFT. It is possible to use an oscillator bank to reconstruct the time-domain samples, but it is worth considering why the SIDFT is not just as simple as the forward process.

The inverse has an apparently similar format:

$$f_b(n) = \frac{1}{N}\sum_{k=0}^{N-1} F(k+b)e^{2\pi ikn/N},$$

where we start at bucket b. Sliding by one sample gives

$$f_{b+1}(n) = \left(f_b(n) - \frac{F(b)}{N} + \frac{F(b+N)}{N}\right)e^{-2\pi in/N}.$$

But that only gives the contribution to the sample from a single point in the collection of windows that contain the required sample point. Moorer suggests averaging over all relevant windows giving the formula

$$f_n = \frac{1}{N}\sum_{t=n-N+1}^{n} f_{n-t}^{(t)} = \frac{1}{N}\sum_{t=0}^{N-1} f_t^{(n-t)}$$

where $f_n^{(t)}$ is the contribution to f_n from the t^{th} window, that is

$$f_n^{(t)} = \frac{1}{N}\sum_{k=0}^{N-1} F_t(k)e^{2\pi ikn/N}.$$

So

$$\begin{aligned} f_n &= \frac{1}{N}\sum_{t=0}^{N-1} f_t^{(n-t)} \\ &= \frac{1}{N}\sum_{t=0}^{N-1}\frac{1}{N}\sum_{k=0}^{N-1} F_{n-t}(k)e^{2\pi ikt/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1}\frac{1}{N}\sum_{t=0}^{N-1} F_{n-t}(k)e^{2\pi ikt/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} y_n(k) \end{aligned}$$

where we define

$$y_n(k) = \frac{1}{N}\sum_{t=0}^{N-1} F_{n-t}(k)e^{2\pi ikt/N}.$$

We are now calculating the recovered sample by a sum over the frequency bins of y_n. Closer investigation shows that there is a recurrence relationship for the $y_n(k)$ quantities:

$$\begin{aligned} &y_{n+1}(k) \\ &= \frac{1}{N}\sum_{t=0}^{N-1} F_{n-t+1}(k)e^{2\pi ikt/N} \\ &= \frac{1}{N}\sum_{t=-1}^{N-2} F_{n-t}(k)e^{2\pi ik(t+1)/N} \\ &= \frac{1}{N}\left(\begin{matrix} F_{n+1}(k) \\ -F_{n-N+1}(k) \end{matrix} + e^{2\pi ik/N}\sum_{t=0}^{N-1} F_{n-t}(k)e^{2\pi ikt/N}\right) \\ &= \frac{1}{N}\left(F_{n+1}(k) - F_{n-N+1}(k) + e^{2\pi ik/N}Ny_n(k)\right) \\ &= \frac{F_{n+1}(k) - F_{n-N+1}(k)}{N} + y_n(k)e^{2\pi ik/N} \end{aligned}$$

That is, each of the y_n is obtained from the previous ones by an addition, a subtraction and a complex rotation. However this also indicates the problem starkly; we need to keep $2N^2$ complex values of F, or N^2 if we rely on the result being real.

There are applications where this large amount of data is justified, but for multiple stream analysis and resynthesis this seems to us to remain outside current capabilities, at least for the first decade of the millennium. Moorer does not identify this aspect with any clarity. We return to this problem of large data later.

The coding of the SIDFT is more complicated than for the SDFT, but the standalone program did implement it. This process with its memory requirements is very difficult to incorporate directly into the Csound f-variable structure, and our initial Csound experiments used oscillator bank reconstructions, later replaced by the process described below(section 6).

5. SDFT WITH WINDOWING

All the explanations above use a rectangular window, and it is common currency that some kind of bell-shaped window is preferable in order to minimise spectral smearing.

Recalling that convolution in the time domain is multiplication in the frequency domain, we can transform our spectral frame if we know or can calculate the frequency response of the window. The sliding method can easily be adapted to the calculation method, but as Moorer points out, if the window shape has a representation in cosines (or sines), it is easy to solve the general case.

The well-known Hamming window has the time representation for a window of size N of

$$w_k = 0.54 - 0.46\cos\left(\frac{2\pi k}{N-1}\right) \quad \text{for} \quad k = 0, \ldots, N-1$$

or in phasor format

$$w_k = 0.54 - 0.23e^{2\pi ik/(N-1)} - 0.23e^{-2\pi ik/(N-1)}$$

from which it follows that the frequency form of the windowed signal is

$$F_t^{(w)}(n) = 0.54F_t(n) - 0.23[F_{t-1}(n) + F_{t+1}(n)]$$

Similar formulae exist for the von Hann window:

$$F_t^{(w)}(n) = 0.5F_t(n) - 0.25[F_{t-1}(n) + F_{t+1}(n)]$$

and Blackman window:

$$\begin{aligned} F_t^{(w)}(n) &= 0.42F_t(n) - 0.25[F_{t-1}(n) + F_{t+1}(n)] + \\ & \quad 0.04[F_{t-2}(n) + F_{t+2}(n)] \end{aligned}$$

.

In providing a user-level package we can allow these simple windows, and could no doubt add others which take this simple form. There is room for experiment here.

6. SIGNAL RECONSTRUCTION

We return to the difficult question of reconstructing a signal from the set of sliding frames. The definition of the inverse is

$$f_t = \frac{1}{N}\sum_{k=0}^{N-1} F_t(k)e^{-2\pi ikt/N}$$

but if we are only attempting to reconstruct the first sample of the window, that is when $t = 0$ this simplifies to

$$\begin{aligned} f_0 &= \frac{1}{N}\sum_{k=0}^{N-1} F_0(k)e^{-2\pi ik(0/N)} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} F_0(k) \end{aligned}$$

because $e^{0\pi/N}$ is unity. So the reconstructed sample is just the average of the frequency components. The formula becomes more complex for $t = 1$ but that does not matter as we can use the next frame for that case. This observation means that the need to maintain $O(N^2)$ data is removed, and the algorithm is $O(N)$ additions only. This simplified inverse (SSIDFT) makes a significant change to the potential applications of the technique.

A simple variant is to reconstruct the sample at the midpoint of the window, where it can easily be seen that the formula is an average of alternating signs. When combined with windowing this corresponds to the full amplitude of the sample. The last sample in the window can be constructed by the more complex formula

$$f_{N-1} = \frac{1}{N}\sum_{k=0}^{N-1} F_{N-1}(k)e^{-2\pi ik(N-1)/N}$$

but as the roots of unity have already been calculated in the SDFT, this is not too large a computational task, and offers low latency.

It took time to convince ourselves of this simplification, but experiments confirmed this simple result. The effect of the single sample overlap obviates the need for the averaging process that is needed in the standard FFT/IFFT method. The implementation issues are now trivial; the algorithm requires little space and is fast.

7. CSOUND IMPLEMENTATION

The current version of Csound[7, 2] includes a system for streamed phase-vocoding [1], where at a predetermined overlap factor, new standard FFT frames are calculated as needed. The increased power of commonly available computers made such a process feasible. The analysis stage of these opcodes creates a new variable class of f-sig, and these f-sigs are updated for each overlapping window. We have usurped this structure to extend it to maintain a set of frames, one for each `ksmps` in MusicN parlance, as well as housekeeping. An opcode translates an audio signal to spectral representation using the algorithm above, and in our case there is no need for the frame size to have any special relationship with the k-rate. Windowing is applied in the spectral domain as indicated above; we provide Hamming, von Hann, Blackman and rectangular windows. All the streaming spectral opcodes have an analogue in this version, and we have most of them implemented.

The current Csound implementation is sufficient for experimental purposes, but for deployment to the wider community we are redesigning the code so there is a smooth transition from FFT-based analysis to SDFT when the overlap warrants this. This means than in Csound5[3] we will be ready for when processor speeds make this method competitive.

8. APPLICATIONS

In order to investigate the quantity and quality that the SDFT provides, we have as a preliminary experiment implemented a pitch shifter from the maximally overlapping frames. The implementation is a basic one based on the code of Bernsee[1], not optimised beyond the first level; each bucket is moved to a destination bucket with phase modification.

[1] Introduced to Csound in v4.13 by Richard Dobson, August 2001

Size	Relative	Size	Relative
64	0.66	256	2.58
100	1.00	512	5.14
128	1.30		

Table 1. Proportion of real-time for a pitch shift

Size	Comments
512	Clean, slight smearing
256	Clean, very slight smear, acceptable pitch
128	Very tight attack, frequency artefacts
64	Very tight attack, more artefacts

Table 2. Quality of loop shifted

The main application test has been pitch shifting of a drum loop and other critical sounds. This represents a specially onerous process as all buckets are modified in each frame, which will update at the sample rate. Furthermore the smearing artifacts of the phase vocoder pitch-shifter are well known.

The time performance is certainly slower that the phase vocoder, as might be expected from an N^2 algorithm as against a $N \log N$. On an Athlon64 3400+ Linux system real-time performance for analysis, windowing, shift by 1.2 and resynthesis of a mono 44.1khZ audio stream is possible with a frame size of 100. Further timings for this processor are given in table 1, showing linear behaviour with the frame size.

More interesting are the quality issues. In table 2 we present the qualitative measures of the same process; a pitch-shift of a 5 second drum loop by a small amount. The cost of applying the various windows seems to be very small, and is lost in the general uncertainty. It is interesting however that the choice of window does have quite a noticeable impact on quality, even with the large frame sizes. The rectangular window smears massively, as one might expect, while Hamming and Von Hann are subtly but perceptibly different, and Blackman at 1024 seems to reduce smearing even further; though it is less useful at smaller sizes; there is scope for more investigation here. These differences are also detectable, if very subtle, on the plain no-effect anal/resynth [2] .

The sizes at and below 256 exhibit virtually no smearing, but varying degrees of obvious pitch distortion, with 64 hardly sounding as if any shift has actually been achieved. 512 seems a very good compromise for quality , and definitely an improvement on normal FFT-based phase vocoding with 8-fold overlap.

9. CONCLUSIONS

We have presented the sliding discrete Fourier transform as not just interesting mathematics but a real practical tool for audio analysis and processing. The computation times for both the transform and its inverse are comparable to the FFT algorithm in practice, and the simplified inverse obviates the need for excessive data remembering. Our simple applications have shown that these operations can be embedded in general processing tools, as well as providing simple standalone programs for further experimentation.

Moorer suggests that this and similar processes are the future of DSP. Our experiments add empirical evidence to that belief, but processor speeds are still short of practical real time use. Qualitatively we are seeing advantages to the maximal overlap, but we need to develop some more ideas to achieve our original aim of a clean shift of a drum sound. This requires attention to the accumulation phase error in the simple pitch-shift algorithm[4] [3] .

While it is still early days for this study we are seeing clear indications that the SDFT offers the possibility of clean pitch shifting using smaller windows than are required for the regular phase vocoder with low latency. We also speculate that the simplicity of the complex rotation in equation in section 2 is highly suited to a parallel hardware implementation in which a new window is generated in one clock cycle. Such a device may achieve Moorer's vision of a CPU which can process continuously 700 channels of audio in real time.

10. REFERENCES

[1] Stephan M. Bernsee. Pitch Shifting Using the Fourier Transform. `http://www.dspdimension.com`, 1999.

[2] Richard Boulanger, editor. *The Csound Book: Tutorials in Software Synthesis and Sound Design*. MIT Press, 2000.

[3] John ffitch. The Design of Csound5. In *Linux Audio Conference*, April 2005.

[4] Jean Laroche and Mark Dolson. New phase vocoder techniques for pitch-shifting, harmonizing and other exotic effects. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, New Paltz, NY., 1999.

[5] James A. Moorer. Algorithm Design for Real-Time Audio Signal Processing. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, March 1984.

[6] James A. Moorer. Audio in the New Millennium. *J. Audio Eng. Soc.*, 48(5):490–498, May 2000.

[7] Barry Vercoe. *Csound — A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Media Lab, M.I.T., 1993.

[2] Audio examples can be found at `http://dream.cs.bath.ac.uk/SDFT`

[3] Their algorithm is patented and it remains to see if such enhancements can be made to the LGPL Csound implementation