Sliding DFT for Fun and Musical Profit

John ffitch[*] Richard Dobson[†] Russell Bradford[‡]

[*]Codemist Ltd
[†]Composer's Desktop Project
[‡]University of Bath

LAC Köln, Mar 2008

UNIVERSITY OF BATH

Where are we today?

- Running transforms — the Sliding DFT
- "There is some advantage to be gained by computing the transform at every sample"
- implication: by 2020 processing audio in the frequency domain will become routine
- compute power formidable: analysis rate = sample rate

UNIVERSITY OF BATH

The Sliding DFT

For details on the basics, see for example Bradford, Dobson and Fitch, ICMC2005 Barcelona.
Instead of using a window advancing by some hop size, we slide the window by one sample (hop size of one). The mathematics is not too complex!

UNIVERSITY OF BATH

The Sliding DFT

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

- $$F_{t+1}(n) = \sum_{j=1}^{N} f_{j+t} e^{-2\pi i (j-1) \frac{n}{N}}$$

- $$F_{t+1}(n) = \left(\sum_{j=0}^{N-1} f_{j+t} e^{-2\pi i j \frac{n}{N}} - f_t + f_{t+N} \right) e^{2\pi i \frac{n}{N}}$$

- $$F_{t+1}(n) = \left(F_t(n) - f_t + f_{t+N} \right) e^{2\pi i \frac{n}{N}}$$

UNIVERSITY OF BATH

The Sliding DFT (cont)

- Encoding: simple complex rotation - basic SDFT quite economical

$$F_{t+1}(n) = (F_t(n) - f_t + f_{t+N})\, e^{2\pi i \frac{n}{N}}$$

- SDFT highly parallelizable - given the hardware
- no requirement for the window size to be a power of two
- analysis rate = sample rate so transformations inflate CPU load
- analysis: frequency-domain windowing by convolution

The Inverse Sliding DFT

- resynthesis: sum over all bins = oscillator bank

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

but for a single point $t = 0$ this simplifies to

$$\frac{1}{N} \sum_{j=0}^{N-1} F_0(j)$$

UNIVERSITY OF BATH

SPV compared to PVOC

In ICMC2007 paper Dobson, Bradford and ffitch showed how to create a Phase Vocoder from this. There are however significant differences between the classic PVOC and the Sliding Phase Vocoder (SPV):

- conversion to amplitude/frequency the same
- Bin bandwidth: pvoc = $\pm$ SR/N; SPV = $\pm$ Nyquist
- so in SPV: any frequency in any bin!
- pitch scaling and frequency shifting fiddly in PVOC, trivial in SPV
- double precision floating point indicated as the phase updates every sample

UNIVERSITY OF BATH

HAMMING	$0.53836 - 0.4614\cos(2\pi n/(N-1))$
HANN	$0.5 - 0.5\cos(2\pi n/(N-1))$
KAISER	*unavailable*
BLACKMAN	$0.42 - 0.5\cos(2\pi n/(N-1))$ $+0.08\cos(4\pi n/(N-1))$
BLACKMAN_EXACT	0.42659071367153912296 $-0.49656061908856405846\cos(2\pi n/(N-1))$ $+0.07684866723989681 8572\cos(4\pi n/(N-1))$
NUTTALLC3	$0.375 - 0.5\cos(2\pi n/(N-1))$ $+0.125\cos(4\pi n/(N-1))$
BHARRIS_3	$0.44959 - 0.49364\cos(2\pi n/(N-1))$ $+0.05676\cos(4\pi n/(N-1))$
BHARRIS_MIN	$0.42323 - 0.4973406\cos(2\pi n/(N-1))$ $+0.0782792\cos(4\pi n/(N-1))$
RECT	1

Some Window Types available in SDFT

UNIVERSITY OF BATH

Spectral Processing in Csound

Csound had off-line (pre-analysed) spectral processing from the start.

Dobson introduced on-the-fly phase vocoding last century.

Already presented at Linux Audio Conference Lazzarini made use of this to provide a range of spectral processing.

We have changed the internals for an (almost) seamless integration of the sliding phase vocoder.

If the hop size is small then it slides....

Performing Analysis

Old form:

```
ain     in
ffin    pvsanal   ain,1024,256,2048,0
```

If this is replaced by

```
ain     in
ffin    pvsanal   ain,1024,1,2048,0
```

the sliding mechanism will be used.

Internally it uses a very different structure

Pitch Shift and Pitch Scaling:

```
instr 3
      al   line      400, p3, 500
      asig in
      fsig pvsanal   asig,128,1,128,1
      fs   pvshift   fsig, al, 10
      acln pvsynth   fs
           out       acln
endin
instr 4
      asig oscili    16000, 440, 1
      fsig pvsanal   asig,512,1,512,1
      fs   pvscale   fsig, 1.1
      acln pvsynth   fs
           out       acln
endin
```

Spectral Transformations

Pitch Shift and Pitch Scaling:
Looks the same but as there is an analysis frame for each sample, the shift amount can change at audio rate.
This is part of the musical advantage

UNIVERSITY OF BATH

Spectral Transformations

The following “just work”:
pvsfreeze, **pvstencil**, **pvsmix**, **pvscent**, **pvsmaska**, **pvsmooth**

These work the same, with use of audio rate where appropriate.

Spectral Transformations

The following mostly work:
pvsfilter: Runs but sound very different

pvsbin: Limited to control-rate output

pvscross: Not really tested

pvsinfo: OK but does not emphasise the sliding nature

UNIVERSITY OF BATH

Spectral Transformations

Not looked at: **pvsdemix**, **pvsmorph**, **pvspitch**, **pvsifd**, **pvsosc**, **pvsblur** and **pvsarp**.

Problem opcodes: **pvsdiskin**, **pvsftr**, **pvsftw**, **pvsfread**, **pvsin**, **pvsout**, **pvsdisp**, **pvsfwrite**, **pvsvoc**, **pvsbuffer**, and **pvsbufread**.

Not needed: **pvsadsyn**

UNIVERSITY OF BATH

Advantage — what advantage?

- reduced latency - as much as $2/3^{\text{rds}}$ reduction compared to PVOC
- predictable - avoid artifacts from changing frame overlap
- frequency modifications easy, no bin-remapping required
- drums and transients - subtle differences, no free lunch

UNIVERSITY OF BATH

Implementation

The Good News:

Our current implementation is imbedded in Csound with little modification to user programs.

All existing programs are uneffected.

UNIVERSITY OF BATH

The Bad News:

It is slow: way too slow for real-time.

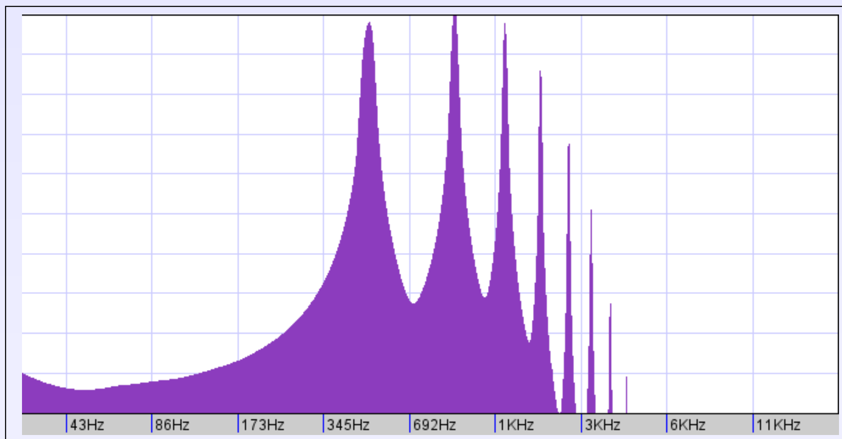
We are investigating the use of vector processing to improve things, but too early to be sure how well it will perform.

Music

With the SPV, the possibility arises to perform genuine per-sample modulation at the full range of audio rates. Indeed, we can mimic classic FM this way.

```
instr 1
    amod oscili 500,500,1     ; modulator
    acar oscili 16000,1000,1 ; carrier
    fcar pvsanal acar,1000,1,1000,1
 ; frequency-domain FM
    fs   pvshift fcar,amod,0
    asig pvsynth fs
    aout dcblock asig
         out  aout
endin
```

Transformational FM



Spectrum of simple Transformational FM

Transformational FM

```
instr 2
         ; carrier is input audio
    acar in
         ; 1KHz modulator
    amod oscili  0.2,1000,1
    fcar pvsanal acar,1000,1,1000,1
    fs   pvscale fcar,1 + amod,0
    asig pvsynth fs
    aout dcblock asig
         out     aout
endin
```

UNIVERSITY OF BATH

Other software

Csound is callable from Pure Data

Csound reads and writes OSC so can be used by other synthsis systems

UNIVERSITY OF BATH

Acknowlegment

This project was supported in part by grant funding from the Arts and Humanities Research Council under their "Speculative Research" theme.

UNIVERSITY OF BATH