Proceedings of the 17th Linux Audio Conference (LAC-19), CCRMA, Stanford University, USA, March 23–26, 2019

CPU CONSUMPTION FOR AM/FM AUDIO EFFECTS

Antonio Jose Homsi Goulart
University of São Paulo
São Paulo, Brazil
ag@ime.usp.br

Marcelo Queiroz
University of São Paulo
São Paulo, Brazil
mqz@ime.usp.br

Joseph Timoney
Maynooth University
Maynooth, Ireland
joseph.timoney@mu.ie

Victor Lazzarini
Maynooth University
Maynooth, Ireland
victor.lazzarini@mu.ie

ABSTRACT

In this paper we present an assessment of the computational performance regarding the use of the AM/FM decomposition framework for the design and implementation of audio effects. The equations and intuitions are reviewed and audio examples are provided, alongside Csound code for real-time implementation. Two types of hardware and several computer music techniques were considered for the comparisons. We also introduce sqENVerb, a novel inexpensive reverb-enhancer effect.

1. INTRODUCTION

Following studies in areas like modulation vocoder [1] [2] [3] and modulation filtering [4] [5] [6], in our previous studies [7] [8] the non-coherent mono-component AM/FM paradigm was presented as a framework for the development of new audio effects. The theory was thoroughly revised and treated in [9], however, the computational effort required to run different types of effects was not addressed.

In this paper we present an assessment of the performance considering different computational systems and different audio processing techniques. Two kinds of computers were used, namely a RaspberryPi model 2B and a Lenovo ThinkPad x220. The former was chosen because it represents the category of low cost programming platforms, that can be used, among other applications, for audio processing; the later represents a more powerful and relatively popular computational system. Netbooks and old laptops might loosely fall in a category between these two examples. Notice also that many programming platforms similar to the Pi actually outperform it, in the same way that many computers assembled for gaming purposes outperform the ThinkPad. So the assessment presented here represents a somewhat conservative scenario; anything running satisfactorily on the Pi and ThinkPad should also run in these more powerful computers.

Beyond the CPU consumption, while our previous papers emphasised manipulations on the instantaneous frequency component of the AM/FM decomposition, now we also address an effect obtained by manipulating the envelope of the signal.

In Section 2 we will briefly review the AM/FM Hilbert-based framework and code for real-time implementation. In Section 3 a new reverb-like effect is introduced and evaluated with a brief objective assessment based on audio descriptors. Then we proceed in Section 4 to a presentation and discussion of the required computational power in order to run the AM/FM framework and effects. Finally, we conclude and point our current and future work. Audio examples will be referenced in the paper with the symbol [▶filename] and are available alongside Csound code for download[1].

2. THE AM/FM FRAMEWORK

The AM/FM decomposition unravels a signal $x(t)$ to a pair of components: an envelope $a(t)$ and an instantaneous frequency signal $f(t)$. Together these signals can modulate a sinusoid both in amplitude and frequency in order to obtain the original signal back, so

$$x(t) = a(t)\cos\left(\int_0^t f(\tau)d\tau\right). \tag{1}$$

We can also think of phasors and interpret the argument for the cosine as an instantaneous phase, which is given by regular increments (the sum represented by the integral) depending of the instantaneous frequency. For instance, a regular sinusoid is the projection on the x-axis of a phasor in which the increments are always the same (tied to its frequency).

In contrast to additive synthesis, where we think globally about the signal, the local aspect of the signals in the AM/FM framework tracks local dynamics in the envelope case, while the instantaneous frequency represents the frequency of a sinusoid that best fits the original signal at each instant.

One of the possibilities for implementing the decomposition is by means of an analytic signal

$$z(t) = x(t) + i\hat{x}(t), \tag{2}$$

where $i = \sqrt{-1}$ and $\hat{x}(t)$ is the Hilbert Transform of $x(t)$.

The Hilbert Transform shifts all the components in a signal by 90° [10], so it might be implemented by using a set of all-pass filters, as is done in the `hilbert` Csound opcode. The important characteristic of the analytic signal is the absence of the negative frequencies; its spectrum resembles the original spectrum of $x(t)$ on the positive frequencies, while the negative components are void, so

$$z(t) = \frac{1}{2\pi}\int_0^{+\infty} X(\omega)e^{i\omega t}d\omega, \tag{3}$$

[1] `https://www.ime.usp.br/~ag/dl/lac19.zip`

where $X(\omega)$ is the Fourier Transform of $x(t)$ [11]. In such a way we can interpret the analytic signal as a superposition of infinite phasors with different frequencies and radii, as shown in Figure 1.

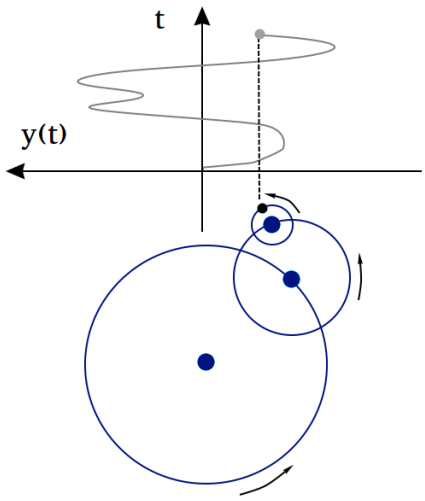

Figure 1: *Analytic signal as a superposition of phasors. The original signal is the projection of the analytic signal onto the real axis. Source: reproduced from [9]*

In Csound the AM/FM decomposition might be implemented with the following code:

```
opcode Udiff,a,a
 setksmps 1
 asig xin
 /* differentiation */
 asig diff asig
 ksig = downsamp(asig)
 /* phase unwrapping */
 if ksig >= $M_PI then
   asig -= 2*$M_PI
 elseif ksig < -$M_PI then
   asig += 2*$M_PI
 endif

 xout asig
endop

opcode AmFmAna,aa,a
  asig xin
  aim,are hilbert  asig /* xhat and x */
  a_am = sqrt(are^2 + aim^2) /* envelope */
  aph = taninv2(aim, are) /* inst. phase  */
                          /* inst. freq. */
  a_fm = Udiff(aph)*sr/(2*$M_PI)

  xout  a_am, a_fm
endop
```

Notice that the `hilbert` opcode is used in order to obtain the analytic signal, and also that the phase needs to be unwrapped. This opcode works in the time domain using 6^{th}-order recursive filters to keep signals in quadrature. Alternatively, we could also employ the `hilbert2` opcode, which implements the same process using a frequency-domain approach implementing a finite impulse response filter (FIR) using a Fast Fourier Transform (FFT) algorithm. However, for this paper we have concentrated on using the former method due to the fact that the FIR approach introduces a latency between input and output that is proportional to the analysis window, and therefore it might not be as well suited to hard real-time applications. In the tests section, we will compare the costs of the time-domain AM/FM process against the application of FFT analysis-synthesis to a signal.

In order to design AM/FM effects we proceed to manipulations in $a(t)$ and/or $f(t)$ followed by a resynthesis step considering the modified signals, as represented in the following code:

```
opcode AmFmRes,a,aa
  a_am_p,a_fm_p xin
  xout a_am_p*cos(integ(a_fm_p)*2*$M_PI/sr)
endop
```

Notice that `a_am_p` and `a_fm_p` represent the potentially processed versions of the estimated `a_am` and `a_fm` (remember that Csound's audio variables names must start with "a").

3. SQENVERB: A NEW AM/FM EFFECT

In our previous papers different families of manipulations were described and thoroughly explained. For instance, the octIFer [8], a beautiful sounding octaver-like effect might be obtained by multiplying the instantaneous frequency signal by 0.5 [▶octifer-half] or even by 0.25 [▶octifer-quarter]. We emphasize, though, that these manipulations are not directly altering frequencies in the spectrum of the original signal, but are actually changing the increments that drive the phasor in the resynthesis process.

Now we describe an effect not yet considered in our previous studies. The manipulation is based on extracting the square root of the estimated envelope signal. The analytic signal envelope lies within the [0,1] range, and considering this interval as our domain for the square root function, we can affirm that the `sqrt` will always return values greater than the argument. Notice that

$$\frac{\sqrt{x}}{x} = \frac{1}{\sqrt{x}}, \tag{4}$$

so the closer the argument is to 0, the greater will be the relative gain. As a consequence, moments of low-intensity sound will be emphasized, leading to pronounced tails. Albeit reverberation is characterized by both early and late reflections [12], the reverberation is arguably more noticeable in the tail of the sound. In such a way the effect can be seen as a sort of compressor/expander [13] which in this case acts extending an already present reverberant tail in the sound.

Differently than a regular gain operation that multiplies the whole signal by the same amount, the square root application results in a selective gain along the signal duration, directly influencing its decay and thus the perception of length. In Figure 2 we can actually check the influence of the Root Mean Square in both the original signal [▶original] and the one with sqENVerb [▶sqenverb]. The RMS is an audio descriptor related to the perception of level in a sound.

As we would expect, the spectral information is not considerably altered by extracting the envelope's square root. In Figure 3 we can check the spectral centroid for the original audio and the sqENVerb edition. The spectral centroid [14] is an audio descriptor related to the perception of brightness in a sound. Both the RMS and spectral centroid evaluation were realized with the Essentia [15] library.

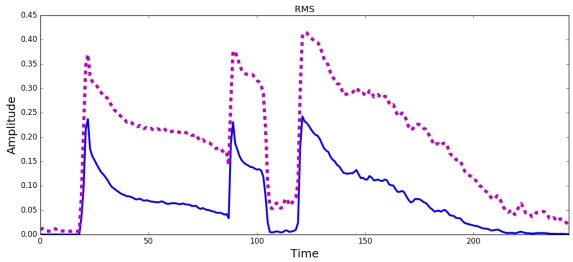

Figure 2: *RMS of dry (solid blue) and processed (dashed magenta) signals. Low intensity moments are greatly influenced by the square root operation.*

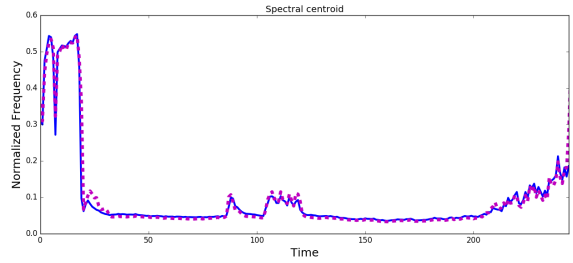

Figure 3: *Spectral centroid of dry (solid blue) and processed (dashed magenta) signals. The operation on the envelope does not considerably influence the spectral centroid.*

4. AM/FM CPU CONSUMPTION

The CPU consumption assessment in important both for artistic considerations (e.g. maximum tolerated latencies to avoid difficulties in musical performance) and also technical reasons (e.g. hardware sizing). In order to evaluate the computational effort to run the different effects, the software `time`[2] was used. It is executed from the shell with the command

```
time csound amfmdafx.csd
```

Here `amfmdafx.csd` refers to a Csound code with an AM/FM effect implemented. The default `time` execution returns three measures:

- real: total duration of the process under analysis;
- user: time taken to work directly on the process;
- sys: time taken to work on system tasks related to the process.

The CPU consumption is then given as $\frac{user+sys}{real}$.

The results[3] are shown in Table 1, which is divided in several parts:

- in the first part of the table some simple and inexpensive computer music tasks are evaluate just to set the scale for the comparisons;
- the second part shows the consumption for realising a FFT and an inverse FFT, considering different windows and hop sizes (shown as number of samples);
- the performance for classic octaver and reverb implementations are then shown;
- then the raw AM/FM framework performance is presented (decomposition followed by resynthesis, with no effects implemented);
- the second to last part shows the performance for some AM/FM effects explored in [9];
- the last part shows the performance considering the octIFer and the sqENVerb cases.

Table 1: *CPU consumption for different types of effects. *The RaspberryPi could not handle a 5000-sample long convolution reverb.*

	CPU consumption (%)	
	RaspberryPi 2B	ThinkPad x220
looped audio	9.49	4.69
clip distortion	10.77	5.06
FFT pair (1024/512)	26.35	8.52
FFT pair (1024/256)	37.38	10.37
FFT pair (1024/128)	45.76	12.31
FFT pair (512/256)	26.05	8.68
FFT pair (512/128)	33.68	10.47
FFT octaver (1024/128)	48.00	12.41
convolution reverb 2500	88.98	14.73
convolution reverb 5000	–*	22.97
simulation reverb	26.13	7.67
AM/FM framework	25.23	7.53
AM/FM IF filtering	29.72	7.73
AM/FM IF compression	33.21	7.78
AM/FM IF modulation	29.23	7.92
AM/FM octIFer	29.87	7.81
AM/FM sqENVerb	28.51	7.77

The FFT algorithm [16] is used widely for the design of audio effects, therefore we adopt it here as a benchmark against which we can measure the computing costs of the AM/FM framework. From the table we can check that both the FFT and AM/FM schemes are computationally accessible, and also that the AM/FM framework is lighter than the FFT/iFFT in all its cases. Another observation is that, in both frameworks, the implementation of a manipulation in the alternative domain does not cause a large increase in the CPU consumption, in comparison to the case where the raw frameworks are applied without any actual effects.

The octIFer effect delivers a high quality sonority [▶octifer-half] for a cost considerably lower than the classic contender [▶octaver], bearing good resemblance in the sonority.

The sqENVerb effect shows a similar consumption in relation to the simulated reverb case [▶simu-reverb], and a huge economy in relation to the convolution reverb. We emphasize that the 5000-sample impulse response convolution [▶conv-reverb5000] required almost twice CPU as the heaviest FFT case; it was not even possible

[2] `http://manpages.ubuntu.com/manpages/xenial/man1/time.1.html`

[3] In order to give more meaning to the numbers, the hardware specifications are: RaspberryPi 2B / quad-core ARM Cortex-A7 @ 900 MHz 32 bits / 1 GB SD-RAM @ 400 MHz / Raspbian / Csound 6.08; ThinkPad x220 / dual-core i5-2520M @ 2.5 GHz 64 bits / 8 GB RAM DDR3 @ 1333 MHz / Debian / Csound 6.09.1. The sample rate considered was always 44100 samples per second.

to run in the RaspberryPi, so another IR with 2500 samples was considered [►conv-reverb2500]. The sonority obtained in this case was not bad, but such a limitation might be questionable, and even with the short IR the Raspberry CPU was almost entirely taken. All the tested AM/FM examples leave considerable CPU headroom so other effects might be applied concurrently.

5. CONCLUSIONS

In this paper we presented, for the first time, a computational performance assessment of the AM/FM audio effects framework. The new AM/FM effect sqENVverb was also developed and compared to the established reverb techniques.

All the examples we explored are based on the non-coherent mono-component Hilbert Transform case of AM/FM decomposition. Different techniques for the decomposition are available, and richer scenarios might also be considered, for instance a filter bank framework, where the dry signal is separated in bands and the subsequent decomposition and processing are applied individually to each band, increasing the computational cost.

The AM/FM decomposition takes the signal to an alternative representation, where even subtle modifications in the envelope or instantaneous frequency signals might result in deep effects after the resynthesis.

The means by which both the octIFer and the sqENVerb effects emulate the octaver and reverb effects might not be orthodox, but the sonorities obtained in both cases resemble the classic techniques, at a considerably lower computational cost. The octIFer sound is quite similar to the classic octaver, and the sqENVerb works fine as a reverb, albeit lacking any control besides a dry/wet mix parameter (which is actually extremely efficient for tuning a reverb).

While it is true that powerful computational systems are increasingly available at decreasing cost, low-consumption algorithms will always be on demand: draining the battery of devices like tablets or smartphones with audio effects might not bring a good user experience; contemporary small single-board computers are still very limited in processing power; old laptops and netbooks, nowadays usually discarded, can instead be harnessed as terrific multi effect pedals.

Plugins for the octIFer and sqENVerb are currently being developed, to be released as open-source software.

6. ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

7. REFERENCES

[1] Sascha Disch and Bernd Edler, "An amplitude and frequency-modulation vocoder for audio signal processing," in *Proceedings of the International Conference on Digital Audio Effects (DAFX-08)*, Espoo, Finland, September 2008.

[2] Sascha Disch and Bernd Edler, "An iterative segmentation algorithm for audio signal spectra depending on estimated local centers of gravity," in *Proceedings of the International Conference on Digital Audio Effects (DAFX-09)*, Como, Italy, September 2009.

[3] Sascha Disch and Bernd Edler, "An enhanced modulation vocoder for selective transposition of pitch," in *Proceedings of the International Conference on Digital Audio Effects (DAFX-10)*, Graz, Austria, September 2010.

[4] S. Schimmel and L. Atlas, "Coherent envelope detection for modulation filtering of speech," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, March 2005, vol. 1, pp. 221–224.

[5] Qin Li and Les Atlas, "Over-modulated am-fm decomposition," in *Proceedings of the SPIE - Advanced Signal Processing Algorithms, Architectures, and Implementations*, Bellingham, WA, 2004, pp. 172–183.

[6] P. Clark and L. Atlas, "Time-frequency coherent modulation filtering of nonstationary signals," *IEEE Transactions on Signal Processing*, vol. 57, no. 11, Nov 2009.

[7] Antonio José Homsi Goulart, Joseph Timoney, Victor Lazzarini, and Marcelo Queiroz, "Psychoacoustic impact assessment of smoothed AM/FM resonance signals," in *Proceedings of the Sound and Musical Computing Conference*, Maynooth, Ireland, July 2015.

[8] Antonio José Homsi Goulart, Joseph Timoney, and Victor Lazzarini, "AM/FM DAFx," in *Proceedings of International Conference on Digital Audio Effects (DAFx)*, Trondheim, Norway, December 2015.

[9] Antonio José Homsi Goulart, Marcelo Queiroz, Joseph Timoney, and Victor Lazzarini, "Interpretation and control in AM/FM-based audio effects," in *Proceedings of International Conference on Digital Audio Effects (DAFx)*, Aveiro, Portugal, September 2018.

[10] Stefan Hahn, *Hilbert Transforms in Signal Processing*, Artech House, Norwood, MA, 1996.

[11] A.V. Oppenheim and R.W. Schafer, *Digital signal processing*, Prentice Hall, New Jersey, USA, 1975.

[12] F. Richard Moore, *Elements of computer music*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1990.

[13] Udo Zölzer, Ed., *DAFx: Digital Audio Effects*, Wiley & Sons, 2nd edition, 2011.

[14] James Beauchamp, "Synthesis by spectral amplitude and 'brightness' matching of analyzed musical instrument tones," *J. Audio Eng. Soc*, vol. 30, no. 6, 1982.

[15] Dmitry Bogdanov, Nicolas Wack, E. Gómez, Sankalp Gulati, Perfecto Herrera, O. Mayor, Gerard Roma, Justin Salamon, J. R. Zapata, and Xavier Serra, "Essentia: an audio analysis library for music information retrieval," in *International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, Brazil, 04/11/2013 2013, pp. 493–498.

[16] Julius Smith, *Spectral Audio Signal Processing*, W3K Publishing, 2011.