Filter design for a sinusoidal synthesis engine using linear programming and sound synthesis examples

Tom Schouten, Bart De Moor

Abstract— **We concentrate on the design of a FIR filter for a fast transform based sinusoidal sound synthesis engine using linear programming and the construction of controller modules that emulate the behavior of contemporary and vintage commercial synthesizers or synthesis techniques. We will show that it is possible to create an entire highly controllable (playable) synthesis framework on top of the transform based engine. For a large class of signals with a sparse representation in the sinusoidal domain the cost for synthesizer modules is relatively low compared to time domain synthesis or processing techniques. Examples are modal synthesis of percussion instruments, combined synthesis and filtering for analog modeling synthesis, and abstract models for FM-like sounds.**

I. Introduction

In *physical modeling* of sound, the goal is to capture the characteristics of an instrument by a physically inspired model that uses a set of parameters reflecting the physical structure of the instrument or sound generating source. In *spectral modeling* of sound, one tries to decompose a sound signal into smaller parts of perceptual relevance to a human observer that can be grouped together to form sound objects. For instance, the data representing a single sinusoidal track in a sound can consist of a succession of windowed oscillations overlap–added together. In this paper, we will concentrate on spectral modeling, more precisely sinusoidal modeling. We will develop a design method for a sinusoidal synthesizer using linear programming and psychoacoustic motivations.

As with physical modeling it is usually the objective to reproduce the sound of a real instrument. This requires an analysis step in which parameter values (static model parameters for physical models and time series of modulation parameters for spectral modeling) are computed. There have been efforts in removing the time line from the spectral representation using simple linear techniques, like modal synthesis [1], on which we will comment later on, and nonlinear modeling techniques which will be the subject of further research.

As opposed to data based methods, abstract techniques have emerged in the past that have attracted many musicians because of the fact that they are usually easily controllable and produce a broad range of expressive sounds. Examples are FM, wave-shaping, modular analog (subtractive) synthesis, ... We will show that a sinusoidal synthesis engine can be used for doing just this by developing abstract sound generation modules with intuitive controls.

Tom Schouten is a Research Assistant with the K.U.Leuven. Email: tom.schouten@esat.kuleuven.ac.be, Tel: +32-(0)16-321811, Fax: +32-(0)16-321970, Postal address: Kardinaal Mercierlaan 94, B-3001 Heverlee (Leuven), Belgium. Bart De Moor is a Research Associate at the F.W.O. (Fund for Scientific Research-Flanders) and a Professor Extra-ordinary at the K.U.Leuven. Email: bart.demoor@esat.kuleuven.ac.be, Tel: +32-(0)16-321715, Fax: +32-(0)16-321970, Postal address: Kardinaal Mercierlaan 94, B-3001 Heverlee (Leuven), Belgium. This research work was carried out at the ESAT laboratory of the Katholieke Universiteit Leuven. This work is supported by several institutions: the Flemish Government (Research Council K.U.Leuven : Concerted Research Action GOA-MIPS and Mefisto-666), the FWO (projects G.0240.99, G.0256.97), and Research Communities(ICCoS and ANMMM), IWT projects (EUREKA 1562-SINOPSYS, EUREKA 2063-IMPACT, STWW), the Belgian State, Prime Minister's Office - OSTC - (IUAP P4-02 1997-2001 and IUAP P4-24 1997-2001, Sustainable Mobility Programme - Project MD/01/24 1997-2000) and the European Commission (TMR Networks: ALAPEDES and System Identification, Brite/Euram Thematic Network : NICONET). The scientific responsibility is assumed by its authors.

II. Sinusoidal oscillator using inverse FFT

A. Introduction

A lot of research has been performed on the efficient synthesis of sinusoidal partials. We refer to [2] for a collection of references. Our method improves the work described in the literature by explicitly designing a synthesis window $h[k]$ and a time interpolation window $w[k]$.

To synthesize a finite discrete sinusoidal partial $p \in \mathbb{R}^N$ with

$$p[k] = A_0 cos(\omega_0 k + \phi_0) \tag{1}$$

and $\omega_0 \in [0, 2\pi)$ in a vector $x \in \mathbb{R}^N$ in the frequency domain

$$X[l] = \sum_{k=0}^{N-1} x[k] e^{-\frac{2\pi j k l}{N}} \in \mathbb{C}^N, \tag{2}$$

we have to add

$$P[l] = \sum_{k=0}^{N-1} p[k] e^{-\frac{2\pi j l k}{N}} \in \mathbb{C}^N, \tag{3}$$

to X. However if $p[k]$ is a sinusoid then $P[l]$ is sparse. The idea is to zero all values in P below a certain threshold or equivalently to add only a limited number of elements or taps T of P to X to obtain an approximation of the sinusoid. In general the DFT of $p[k]$ is

$$P(\omega) = \frac{A_0}{2}[e^{j\phi_0} H(\omega + \omega_o) + e^{-j\phi_0} H(\omega - \omega_0)] \tag{4}$$

with $H(\omega) = \sum_{k=0}^{N-1} e^{-j\omega k} h[k]$ the DFT of a time domain window function $h[k]$. If we sample the continuous spectrum in $\omega = 2\pi l/N$ we obtain the discrete spectrum. In

00-23

Schouten T., De Moor B., ``Filter design for a sinusoidal synthesis engine using linear programming and sound synthesis examples", in *Proc. of the 2nd IEEE Benelux Signal Processing Symposium (SPS-2000)*, Hilvarenbeek, The Netherlands, Mar. 2000, cdrom p., Lirias number: 178923.

other words $P[l] = P(2\pi l/N)$. The inverse FFT yields

$$p[k] = \frac{1}{N} \sum_{k=0}^{N-1} e^{\frac{2\pi jkl}{N}} P[l]. \quad (5)$$

This means that the partial $p[k]$ is a modulated version of the time domain window $h[k]$,

$$p[k] = \frac{A_0}{2} h[k](e^{j(\phi_0+\omega_0 k)} + e^{-j(\phi_0+\omega_0 k)}) \quad (6)$$

or $p[k] = h[k]A_o \cos(\phi_0 + \omega_0 k)$. The sinusoid we want to obtain is equal to $p[k]/h[k]$. If we can find an easy way to calculate $H(\omega)$, with $\omega \in [0, 2\pi)$, for example using *table lookup* and *interpolation*, we can reduce the number of function evaluations per partial needed to synthesize N samples of this partial from N to T if we allow an FFT as fixed cost. If we allow for $h[k]$ to be symmetric, $H(\omega)$ is real valued. This simplifies interpolation and enables us to use linear programming [3] for the window design.

By using a window function with low side-lobe level (and correspondingly high main lobe bandwidth) we can improve the performance even more. There is a trade-off here: the larger the main lobe, the more taps need to be preserved. On the other hand the side lobe level will be lower and the sparse approximation error decreases. In general the error level must be below the hearing threshold or the masking threshold to prevent the perception of these errors.

B. Window design

Instead of using a windowing function in analytically closed form, as proposed in the literature (see references in [2]), our contribution is to design a window that will have the properties we want. The sinusoidal synthesis method is completely numerical so we don't need the property that the window function is in closed form. As we will see this yields a measurable trade-off: noise energy vs. number of taps per sinusoid.

Now we translate the desired properties of the window into a design specification of an FIR filter. Suppose the window has N taps. In order for a sinusoid to be concentrated in T taps the one sided bandwidth of the window can be $T/2$ taps at maximum. The tap spacing in an N point FFT is $1/N$ in normalized frequency $f_n = f/f_s$ with f_s the sample frequency of the digital signal. In order for the T taps to be in the passband of the filter, the filter has to be low-pass with high attenuation for $|f_n| > T/2N$. The higher the number of synthesis taps T, the broader the passband of the filter needs to be. As a consequence, the stopband level, which is proportional to the approximation error, will be low for a large T.

The FIR filter design specification is as follows: design an N tap symmetric FIR filter with stopband edge $f_c = T/2N$. The transition band shape is unimportant. Then minimize the stopband magnitude bound $\delta(\omega)$.

C. Algorithm

We illustrate the algorithm used to synthesize the partials. Since we only found vague references to the actual algorithm used, we started from scratch. First we show how to compute $H(\omega)$. Suppose we can evaluate $H(l) = \sum_{k=0}^{N-1} e^{-2\pi jkl/N} h[k]$ not only for integer values of l but also for $l = k/S$ with $k, S \in \mathbb{N}$. S represents the interpolation factor of the FFT of the filter coefficients $h[k]$. We can build an interpolated lookup table for $H(l)$ and can discard the values for $l > T/2$ and $l < 0$. This table will have $ST/2$ entries. The indexing of $H(l)$ with fractional numbers is called *fractional bin indexing*. In the remaining of this subsection $H(f)$ means the table value with integer index $k = [fS + 1/2]$ or fractional bin index $f' = [fS + 1/2]/S$.

Now we will show how to synthesize a partial with amplitude A_0, phase ϕ_0 and fractional bin frequency f_0 in a spectrum vector $X \in \mathbb{C}^{\mathbb{N}}$ using an odd value for T. The frequency $f_0 \in \mathbb{R}$ will be split into 2 parts: the center index $i = [f + 1/2]$ and the frequency displacement $\Delta f = f - i$.

We will use a vector $Q \in \mathbb{R}^{\mathbb{N}}$ as an approximation for the symmetric part of the spectrum of the sinusoid $h[k]e^{j2\pi f_0 k}$. An additional phase shift and scaling can be accomplished by a complex multiplication of this symmetric part. We use circular indexing meaning $Q[k] = Q[k \bmod N]$. The construction of Q goes like this:

1. Set $Q = 0$
2. Set center tap $Q[i] = H(|\Delta f|)$
3. Set side taps: for $k = \pm 1, \pm 2, \ldots, \pm\frac{T-1}{2}$ do $Q[i+k] = H(k - \text{sign}(k)\Delta f)$.

Define $Q' \in \mathbb{R}^{\mathbb{N}}$ as $Q'[k] = Q[-k]$. The spectrum of the sinusoid $X \in \mathbb{C}^{\mathbb{N}}$ can be synthesized as follows:

$$X = \frac{A_0}{2}(Qe^{j\phi_0} + Q'e^{-j\phi_0}). \quad (7)$$

Using this method more partials can be added to X in the same way. For our implementation we used a half complex to real transform. This allows a reduction of a factor two with respect to a full transform. This also means we only have to add the Q part of (7) to the spectrum and we have to correct for the wrap around at the DC end of Q. We don't have to do this at $f_s/2$ by imposing a maximum fractional bin frequency $f_{\max} = \frac{N-T}{2}$. For the high part this is acceptable. We can't impose this constraint on the DC side since it would make the synthesis of very low frequencies ($< \frac{T}{2}$) impossible. We implement this wrap around by extending Q by $(T-1)/2$ elements on the DC side (a small portion of the negative frequency axis). As a final step, after all the partials are summed, we once correct the entire spectrum by taking the complex conjugate of the negative part and adding it to the positive part of the half complex spectrum.

D. Reducing the error signal

Stopband shape

The resulting error signal $e[k]$ is a sum of sinusoids that will have an impulse like character with much of its energy

concentrated at the window edge rather than a noise like character because of strong phase coherence in the stopband of the synthesis filter. This impulsive character will allow the auditory system to pick up the error energy outside the center of the peak frequency resulting in noticeable distortion. That is why a flat stopband filter, often called an optimal filter, is not optimal for our approach, especially for low frequency sinusoids. If we allow the side-lobe level of the synthesis filter $h[k]$ to decrease rapidly, for example as $1/\omega^s$, with s the slope of the stopband error in a log-log plot, we can move the distortion energy to a region where it is more likely to be masked: near the sinusoidal peak. This results in a windowing function $h[k]$ with a rapid decay in the time domain also. As a consequence, division by $h[k]$ becomes ill conditioned. We therefore have to take a better look at the time domain interpolation mechanism.

Time domain interpolation

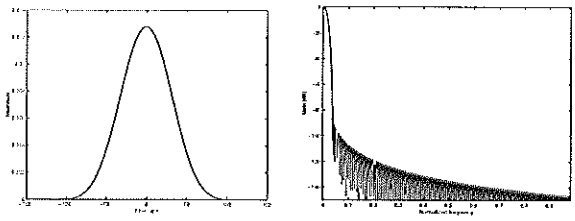
The time domain interpolation between consecutive frames will have a serious effect on the amplification of the residual error signal. This interpolation is rather unspecified, it only has to add up to 1. Let us investigate the effect of the interpolation. After division by $h[k]$ and multiplication by the time domain interpolation (overlap-add) window $w[k]$ the weighting of $e[k]$ will be largest at the window edges[1]. This means that there is some optimal $w[k]$ for which these distortions are minimal. We can say some things about the amplification of errors by using the combined window $c[k] = w[k]/h[k]$. This is the weighting vector that represents the overall weighting for demodulation and overlap-add of the time vector $x[k] = \frac{1}{N}\sum_{k=0}^{N-1} e^{j\omega k} X[l]$ coming out of the inverse FFT. This is in fact the net amplification of the error signal $e[k]$. Because of the rapid decay of $h[k]$ its inverse will grow very rapidly at the edges. In a sense we can say that the tail of $w[k]$ should resemble the tail of $h[k]$ to avoid this, because large amplitude differences of the error signal are unwanted. If we take $c[k]$ constant at the edge of the window, we can smooth out the peak error amplification at the edges. When we impose this constraint, we can't say anything about $c[k]$ in the center of the window because the overlap constraint must hold. Another way of reducing the error amplification is to increase the overlap factor of the time domain interpolation. We used a $1/2$ overlap for N even and a $(N-1)/2N$ overlap for N odd.

III. Formal filter design problem

A. Remark about symmetry

Using a symmetric window allows us to keep $H(\omega)$ real valued. Although it is perfectly feasible to apply this technique to a base 2 FFT or a window $h[k]$ with even number of taps, this introduces a $1/2$ term in the equations because the symmetry center is located in between two taps of $h[k]$.

[1]The effect of the interpolation window $w[k]$ on the intended part of the signal is zero if the partial is constant (if its parameters excluding the needed phase correction stay the same from one frame to the other.) For our practical purpose the partials can be considered constant and we can design $w[k]$ for minimal error amplification.



(a) Time Plot (b) Frequency Plot

Fig. 1. Filter with $T = 9$, $N = 3^5$, $s = 2$.

If we allow for an odd window size, this term disappears and notation will be much simpler.

B. Symmetric window with an odd number of taps

A symmetric low pass filter with a curved stopband ($1/f^s$ form) needs to be designed. For this we use linear programming. The constraints for the filter are specified for the stopband in the form $|H(\omega)| \leq \delta/\omega^s$ for $\omega \in [\omega_s, \pi)$. With $H(\omega)$ the Fourier transform of the filter coefficients $h[k]$ which if N is odd can be written as

$$H(\omega) = e^{-j\omega\frac{N-1}{2}}\left(h[0] + \sum_{t=1}^{\frac{N-1}{2}} 2h[t]\cos(\omega t)\right) \qquad (8)$$

and is linear in the $h[k]$. The optimization problem we need to solve to find $h[k]$ can be formulated as follows:

$$\min_{\delta, h[0], \ldots h[N-1]} \delta$$

subject to

$$\begin{cases} -\delta/\omega^s \leq H(\omega) \leq -\delta/\omega^s, \ \omega \in [\omega_s, \pi) \\ H(0) = 1 \end{cases} \qquad (9)$$

C. Results

The constraints are continuous in ω. To find the solution we have to find a discrete linear program with the same solution or a solution that is close enough for practical purposes. Therefore we sample the constraints (9) with an equal spacing in the interval $[\omega_s, \pi)$. For the filter displayed in figure 1 we used 2000 points. This creates a small overshoot for some of the points in between the discrete sampled points. This overshoot is overall of the order of 1dB so we can ignore it. We used the Matlab program linprog to solve the problem. We had to add the constraint $0 \leq \delta \leq .5$ to make the dual program feasible. We chose $N = 243$, $L = 9$ and $s = 2$. This particular problem takes 10 minutes to run on a Sun Ultra 5 270MHz workstation.

IV. Applications

A. Modal synthesis

Modal synthesis is a linear sound modeling technique in which an instrument is represented by its dominant modes of oscillation [1]. This technique can be used to model impulsive sounds (bell, wood-block, marimba, etc...). The model behaves as a physical model for the instrument since it has a notion of the *state* of the instrument. The true advantage of a modal model is the principle of *strike* and *re-strike*. When the model is first excited it starts oscillating. When it is excited when it is still ringing, the excitation can re-enforce some modes of the model and cancel out others. This adds a significant 'realness' to the sound.

It is usually implemented by using a parallel bank of biquad (2-pole) filters, This is a very efficient modeling technique because a lot of impulsive sounds can be modeled by a small number of modes (e.g. 4). However, when we look at the musical setup in which a percussive instrument is applied, we have to conclude it is most of the time used together with other percussive instruments.

It is possible to use an FFT sound synthesis engine for modal synthesis if we can justify some approximations and the fixed cost of the FFT not present in the biquad implementation. The damping factor of the modes is usually very small. This means we can generate a fair approximation of the exponentially decaying sinusoid if we synthesize it using interpolation of overlapped windows of undamped sinusoids using the FFT engine. We only need to update the change of state once per synthesis frame for every mode which is a severe reduction in computational complexity.

We now construct a system model that behaves like a biquad without inputs. We denote a mode by two complex numbers a representing amplitude and phase and p representing frequency and damping. The state equation for a free model (without excitation) is $a[k+1] = a[k]p$. The output equation is $x[k] = \Re(a[k])$, with $\Re(y)$ the real part of y.

Now we make an abstraction of the noise excitation in the biquad implementation. The effect of the noise burst is that there will be energy added to or subtracted from the different modes. We can denote this as: $a[k+1] = a[k]p + e[k]$. With $e[k] = A_k e^{j\phi_k}$ for $k \in \{k_1, k_2, \ldots\}$ and $e[k] = 0$ elsewhere. The k_i denote instances of excitation and A_k and ϕ_k represent the amplitude and phase (usually random) of the excitation. This we can easily translate to an update at the frame rate of the FFT engine. If we compute $p' = p^S$ we can write the state equation at the frame rate as: $a[q+1] = a[q]p' + e[q]$ with q being the frame number and S the frame spacing in samples. The $a[q]$ are the complex amplitudes that need to be sent to the FFT engine, together with the frequency of the pole p.

B. Virtual analog synthesis and abstract models

In the pioneering days of electronic sound generation analog synthesis was very popular. Nowadays digital synthesis is a lot cheaper than analog so there have been different approaches to bringing still not forgotten analog oscillators and filters to the digital domain, which is by no means a trivial operation [4] [5]. We show a method for performing virtual analog synthesis of a combined oscillator and resonant filter.

If we want to generate undamped sinusoids using the principles explained above, we can't use the complex multiplication scheme because of stability issues. Instead a lookup table can be used to compute the function $\cos(\phi_0 + \omega Sq)$.

What about the initial state $a[0]$ of a sinusoid? We align M of these states in a vector in which each element $a_{m,0}$ is associated with a pole $p_m = e^{j\omega_{m,0}}$. If we assume $\omega_{m,0} = m\omega_0$, we can see this vector as an initial state vector for some harmonic oscillator which represents its timbre.

We now compute a filtered version of a sawtooth wave. The complex amplitude of the mth harmonic of a sawtooth wave $a_{m,0} = j/m$. The moog 4–pole resonant filter [4] is characterized by the transfer function $H_f(\omega) = \frac{1}{k+(j\omega/\omega_c)^4}$ with k the resonance control and ω_c the cut-off frequency. The initial phase of the *filtered* partials $a'_{m,0} = \frac{j}{m}\frac{1}{k+(jm\omega_0/\omega_c)^4}$. This calculation only has to be performed when a parameter changes.

By adding different controls besides the 3 specified above, efficient, though easy controllable abstract synthesis boxes can be built on top of the FFT engine. For example a constant and quadratic term in the equation for the frequency of the mth harmonic, instead of just a linear relation adds a lot of flexibility to the model. This has been implemented in c++ using a tcl user interface.

V. Conclusion

In this paper we commented on current FFT based synthesis techniques. We showed how to improve these techniques by designing a synthesis window that is in a sense optimal for a sinusoidal synthesis engine with a given number of frequency domain taps per sinusoid. We used psycho-acoustical motivations for specification of the filter and linear programming for the calculation of the filter coefficients. We explained some applications of the engine that we have implemented to demonstrate its flexibility.

References

[1] Cook P. R., "Toward phisically-informed parametric synthesis of sound effects", in *Proc. 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, New York*, October 1999.

[2] Serra X., "Musical sound modeling with sinusoids plus noise", in *Musical Signal Processing*. 1997, Swets & Zeitlinger Publishers.

[3] R.J. Vanderbei, *Linear Programming: Foundations and extensions*, Kluwer Academic Publishers, Boston, 1996.

[4] Stilson T., "Analyzing the moog vcf with considerations for digital implementation", in *International Computer Music Conference, Hong Kong*, 1996.

[5] Stilson T., "Alias-free digital synthesis of classic analog waveforms", in *International Computer Music Conference, Hong Kong*, 1996.