A HYBRID FILTER–WAVETABLE OSCILLATOR TECHNIQUE FOR FORMANT-WAVE-FUNCTION SYNTHESIS

Michael Jørgen Olsen, Julius O. Smith III and Jonathan S. Abel
Center for Computer Research in Music and Acoustics (CCRMA), Stanford University
660 Lomita Drive, Stanford, CA 94305, USA
[mjolsen|jos|abel]@ccrma.stanford.edu

ABSTRACT

In this paper a hybrid filter–wavetable oscillator implementation of Formant-Wave-Function (FOF) synthesis is presented where each FOF is generated using a second-order filter and wavetable oscillator pair. Similar to the original time-domain FOF implementation, this method allows for separate control of the bandwidth and skirtwidth of the formant region generated in the frequency domain by the FOF synthesis. Software considerations are also taken into account which improve the performance and flexibility of the synthesis technique.

1. INTRODUCTION

Formant-Wave-Function (FOF) synthesis is a vocal synthesis technique inspired by the source–filter model of vocal synthesis that models the excitation of resonant frequencies in the vocal tract by the glottis [1, 2]. FOF synthesis has been used to create very realistic vocal sounds. This is due to the flexibility of the method in allowing the composer to shape the frequency spectrum of the sound and to morph between different vowel sounds or from vocal sounds to non-vocal sounds.

The majority of the previous implementations of FOF synthesis have focused on implementing a time-domain representation of the FOF bursts. This can be done quite cheaply, in computational terms, since table lookup can be used. However, an overlap-add scheme is needed to combine the FOF bursts into a single audio stream as the generation of a single FOF burst needs to be done independently of real-time changes of the input parameters. Also, since each burst decays exponentially, a suitable cut-off point must be chosen for when to end each particular FOF burst.

In this paper, a FOF synthesis algorithm is presented that uses a second-order filter in combination with a sinusoidal wavetable oscillator to generate the FOF bursts. The filters are triggered using an impulse train. By using a sample-and-hold mechanism in conjunction with a cycling bank of identical filters, artifact-free real-time control of input parameters can be facilitated.

The paper is organized as follows: in Section 2 prior work on FOF synthesis is reviewed; in Section 3 the second-order FOF envelope filter structure is derived; in Section 4 the details of the software implementation of the proposed algorithm are presented; in Section 5, the results are summarized and potential areas for future research are motivated.

2. PREVIOUS WORK

2.1 Speech and Vocal Synthesis

The desire to recreate human speech and singing has long fascinated humankind. After the advent of the telephone in the late 19th century, the need to send and receive the sound of human speech over transmission lines led to much research in the analysis/synthesis of human speech. Much of the original research along those lines was completed at Bell Labs during the early 20th century, yielding in particular Homer Dudley's *vocoder* ("voice coder")—a voice analysis and resynthesis device, and the Voder—a manually driven speech synthesizer demonstrated at the 1939 World's Fair [3]. The first computer-generated vocal synthesis was created with a software version of the vocoder in the 1960s, also at Bell Labs, and led to the generation of the first digital singing synthesis [4]. Later on, resonant bandpass filters were used to generate vocal synthesis [5, 6] but the technique was prone to audible artifacts being present due to the sharp discontinuity at the start of the exponential decay. A more complete history of singing-voice synthesis can be found in [7].

2.2 Formant-Wave-Function Synthesis

Developed in the 1980s by Xavier Rodet at IRCAM in Paris, the original FOF synthesis technique [1] involved determining the time-domain response $y(n) = (x * h)(n)$ of a filter with impulse response $h(n)$ to a particular excitation signal $x(n)$. Assuming that the excitation signal is a periodic repetition of impulses or other such excitation shape, the time-domain representation of the output of the filter can be determined and the synthesis can be performed in the time-domain by repeating the output signal at the period of a desired fundamental frequency and performing an overlap–add operation to obtain a single output stream.

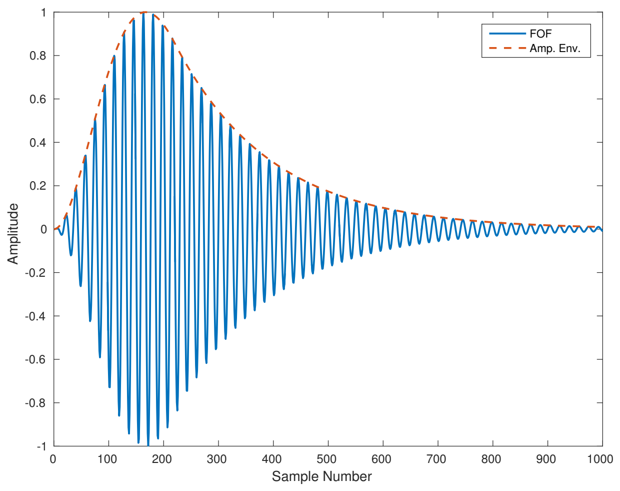

Figure 1. FOF with $\alpha = 80\pi$ rad/s, $\pi/\beta = 5$ ms, $\omega_c = 2500$ rad/s and SR $= 44.1$ kHz

The following time-domain function was preferred [1]:

$$\begin{aligned} y(n) &= \frac{1}{2}\left(1 - \cos[\beta nT]\right) e^{-\alpha nT} \sin\left(\omega_c nT + \phi\right) \\ &\qquad \text{for } 0 \le n \le k, \\ y(n) &= e^{-\alpha nT} \sin\left(\omega_c nT + \phi\right) \\ &\qquad \text{for } n > k, \end{aligned} \tag{1}$$

where $k = \pi/(\beta T)$ controls the amplitude envelope attack time as well as the skirtwidth of the formant region in the frequency domain, α controls the bandwidth of the formant region and ω_c controls the center frequency of the formant region. The term skirtwidth refers to the width of the formant region measured further away from the peak than the bandwidth is measured. In other words, this function generates an exponentially decaying sinusoid whose initial discontinuity is smoothed over k samples (Figure 1).

The form of Eq. (1), while expressed in the time domain, was chosen for its spectral properties. The bandwidth and skirtwidth can be controlled independently of each other and the spectral envelope of the formant region is symmetric about the center frequency.

By running multiple FOF generators in parallel and summing the outputs it is possible to generate a wide variety of vocal and instrumental sounds using this method. In particular, this method is used in the software program CHANT developed at IRCAM in the 1980s [2]. The original CHANT program was implemented at IRCAM using an FPS-100 array processor and was later ported to C in the early 1990s so that CHANT could run on any Unix or Macintosh system [8].

In the late 1980s, a version of FOF was ported to Music 11 as part of the VOCEL (VOiCe ELeven) project by Michael Clarke [9]. This version of FOF included additional features such as allowing different fundamental frequencies for individual FOF generators and an octaviation effect. Additionally, this implementation used a lookup table containing a user-specified attack shape which determined both the attack and decay envelope of a single FOF burst. This version of FOF was later ported to Csound [10].

A different approach was taken by Philippe Depalle et al. in 1992 [11]. Their approach was to separate the excitation signal from the resonant filter by developing a time-domain function for the excitation signal implied by Eq. (1). Then, the excitation signal could be run through a simple second-order resonant bandpass filter. This would allow utilization of the filtering schemes on DSP chips.

A nice property of this scheme is that it was no longer necessary to overlap-add time-domain functions nor worry about audible noise created by truncating the time-domain FOF before it has sufficiently decayed. Additionally, [11] provides a compact function for controlling the skirtwidth of the formant region created by the frequency response of the filter output.

An approach suitable for VLSI implementation on a DSP chip was developed in 1996 by J. Spanier et al. [12]. Their method involved developing a filter that would generate an amplitude envelope with favorable spectral properties. The filter could then be excited by an impulse train and the output of the filter used to envelope the output of a sinusoidal oscillator.

More recently, Michael Clarke and Xavier Rodet ported FOF synthesis to Max [13]. This version contained features of both the FOF version contained in Csound and early FOF objects already in Max.

SuperCollider has a uGen `formlet` [14] dating back to 2002 [1] which forms a FOF-type wave burst using the difference between two second-order resonant bandpass filters having different decay rates but the same center frequency. Its amplitude envelope is given by the uGen `decay2` [15] which uses the difference of two one-pole exponential decays to create an attack-smoothed exponential envelope.

3. HYBRID FILTER–OSCILLATOR FOF IMPLEMENTATION

In this paper a hybrid filter-wavetable oscillator model is employed in which an impulse train is fed into a second-order filter which generates a FOF amplitude envelope similar to the amplitude envelope in Figure 1. That envelope is then multiplied by a sinusoidal wavetable oscillator to generate the FOF waveform.

3.1 Filter Derivation

As seen in Section 2, previous hybrid FOF implementations have either found an explicit formula for a smooth excitation filter to feed into a resonant bandpass filter or designed a modified filter with a smoother attack that can be excited with a periodic impulse train. One such filter, which implements the amplitude envelope $t^2e^{-\alpha t}$, is mentioned in [12] but was not used as the authors found the frequency response to be unsuitable for FOF synthesis. Additionally, that filter does not allow for control of the skirtwidth of the generated formant region.

As shown in [16], in the context of artificial reverb generation, the convolution of two decaying exponential envelopes (Figure 2) yields an envelope with a smoothed at-

[1] James McCartney, personal and email communication, Apr. 22-23, 2016

tack that has a shape similar to other FOF envelopes (such as the one generated by Eq. (1) in Figure 1).

While comb filters with controllable feedback delay-time parameters are used in the setting of artificial reverberation, for the generation of a FOF amplitude envelope, one-pole resonators with unit delays are sufficient:

$$\hat{H}(z) = \frac{1}{1-\mu z^{-1}}, \tag{2}$$

where $\mu = e^{-\alpha T}$ is the coefficient of decay, T is the sampling period and $\alpha = 1/\tau$ is the inverse of the decay time-constant τ. This filter has impulse response

$$\hat{h}(n) = \mu^n u(n), \tag{3}$$

where $u(n)$ is the unit step, $u(n) = 1, n \geq 0$, and $u(n) = 0, n < 0$. Thus, the convolution of two such one-pole filters

$$h(n) = \hat{h}_1(n) * \hat{h}_2(n) = \mu_1^n * \mu_2^n = e^{-\alpha_1 nT} * e^{-\alpha_2 nT} \tag{4}$$

corresponds to the multiplication of their transfer functions, giving the following second-order filter

$$\begin{aligned} H(z) &= \frac{1}{1-e^{-\alpha_1 T}z^{-1}} \cdot \frac{1}{1-e^{-\alpha_2 T}z^{-1}} \\ &= \frac{1}{(1-\mu_1 z^{-1})(1-\mu_2 z^{-1})} \\ &= \frac{1}{1-(\mu_1+\mu_2)z^{-1}+\mu_1\mu_2 z^{-2}}. \end{aligned} \tag{5}$$

This filter will produce an amplitude envelope in the time-domain whose frequency response is a formant centered at dc (0 Hz). Assuming that $\alpha_1 > \alpha_2$ so that $\mu_1 < \mu_2$, $\alpha_2 = \pi \cdot \mathrm{BW}$ can be used to control the decay time of the amplitude envelope which will in turn control the -3 dB bandwidth of the formant (where BW is the bandwidth in Hz).

The other filter parameter α_1 can be used to tune the rise time of the amplitude envelope which will also control the skirtwidth of the formant region. For a constant value of μ_2, as $\alpha_1 \to \alpha_2$ it follows that $\mu_1 \to \mu_2$. When $\mu_1 = \mu_2$, the longest rise time is achieved which gives the narrowest possible skirtwidth in the frequency response for the bandwidth controlled by α_2. Conversely, as $\alpha_1 \to \infty$, $\mu_1 \to 0$ so the attack time becomes shorter with the filter becoming a one-pole exponential decay in the limit.

The impulse response of the amplitude envelope is given by [16] to be

$$h(n) = \frac{1}{\mu_2 - \mu_1}\left(\mu_2^{n+1} - \mu_1^{n+1}\right). \tag{6}$$

The rise time—the time of the impulse response maximum—may be found as the time in at which the impulse response time derivative

$$h'(n) = \frac{T}{\mu_2 - \mu_1}\left(\alpha_1 \mu_1^{n+1} - \alpha_2 \mu_2^{n+1}\right) \tag{7}$$

is zero. Expressing the time in seconds, we have

$$nT = \frac{\ln(\alpha_1/\alpha_2)}{\alpha_1 - \alpha_2} - T. \tag{8}$$

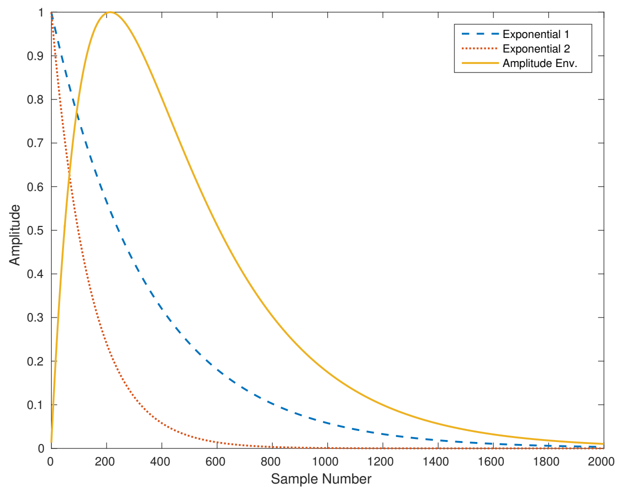

Figure 2. Amplitude envelope generated by the convolution of two decaying exponentials

Additionally, the maximum allowable rise time for a particular bandwidth α_2 can be determined by taking the limit of Eq. (8) as $\alpha_1 \to \alpha_2$. In detail, making use of L'Hôpital's rule, it is found that

$$\begin{aligned} nT_{max} &= \lim_{\alpha_1 \to \alpha_2} \frac{\ln(\alpha_1/\alpha_2)}{\alpha_1 - \alpha_2} - T = \lim_{\alpha_1 \to \alpha_2} \frac{1}{\alpha_1} - T \\ &= \frac{1}{\alpha_2} - T. \end{aligned} \tag{9}$$

In order to calculate an α_1 that satisfies a particular choice of α_2 and rise time $nT \leq nT_{max}$, it is necessary to solve Eq. (8) iteratively or using the Lambert W function [17]. If Eq. (8) is put in the form $Y = Xe^X$, then $X = \mathcal{W}(Y)$ where $\mathcal{W}$ denotes the Lambert W function. Rearranging Eq. (8) so that it is in the prerequisite form and then applying the Lambert W function gives

$$\alpha_1 = \frac{-1}{(n+1)T}\mathcal{W}\left(-\alpha_2(n+1)Te^{-\alpha_2(n+1)T}\right). \tag{10}$$

3.2 Filter Amplitude Normalization

The filter developed in Section 3.1 has the largest amplitude response at dc. Therefore, normalization is achieved by normalizing the dc amplitude response of the filter. The frequency response of the filter is given by:

$$H(e^{j\omega T}) = \frac{1}{1-(\mu_1+\mu_2)e^{-j\omega T}+\mu_1\mu_2 e^{-2j\omega T}} \tag{11}$$

and so the magnitude response at dc is given by:

$$\begin{aligned} G(0) = |H(0)| &= \left|\frac{1}{1-(\mu_1+\mu_2)+\mu_1\mu_2}\right| \\ &= \frac{1}{(1-\mu_1)(1-\mu_2)}. \end{aligned} \tag{12}$$

Thus, the gain coefficient $\gamma = 1/G(0)$ will normalize the magnitude response of the filter to 0 dB. Then, any other magnitude level is easily obtained by applying the correct linear scale factor $g = 10^{\beta/20}$ where β dB is the desired offset in dB.

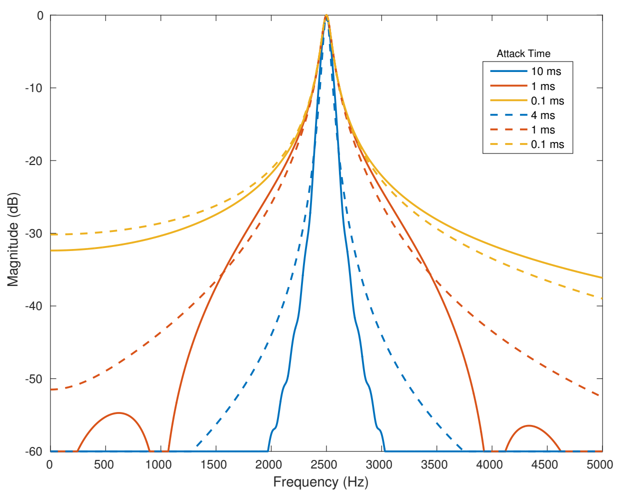

Figure 3. Comparison of frequency characteristics between original and proposed FOF with: $\alpha = 80\pi$ rad/s, $\omega_c = 2500$ rad/s and varying rise times (Orig.: solid lines, Proposed: dashed lines)

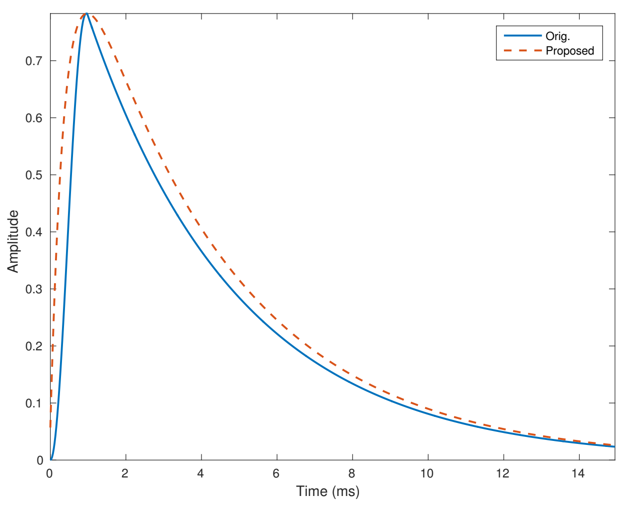

Figure 4. Comparison of amplitude envelope characteristics between original and proposed FOF with: $\alpha = 80\pi$ rad/s, $\omega_c = 2500$ rad/s and 1 ms rise times

3.3 Complete FOF Architecture

The complete FOF structure is comprised of an impulse train that triggers the amplitude envelope filter which is subsequently multiplied by a sinusoidal oscillator. The frequency f_0 of the impulse train determines the fundamental frequency of the synthesized sound and the sinusoidal oscillator determines the center frequency f_c of the formant region created by the FOF.

The filter defined in Eq. (5) can be expanded using partial fraction expansion into a difference of one pole filters similar to `decay2` [15] in SuperCollider. The sinusoidal oscillator can be combined with Eq. (5) transforming it into a fourth-order resonant bandpass filter which can similarly be expanded into a difference of second-order resonant bandpass filters like `formlet` [14]. The performance differences between the implementations will be touched upon in Section 5.

A comparison of the spectral qualities of the method in this paper with Rodet's original FOF technique (Eq. (1)) is illustrated in Figure 3. A single decay rate with a variety of rise times are plotted. Overall, the bandwidths are quite comparable between both techniques with the main difference being that the original FOF formula can achieve considerably narrower bandwidths for a fixed decay rate. For the 80 Hz bandwidth used in the example, our technique was not able to match the 10 ms rise time and reached an upper bound of approximately 4 ms. The biggest difference, as seen in Figure 3, is the narrowness of the skirtwidth achieved by the original FOF technique. The authors did not investigate whether or not this difference is perceptible. While the shapes of the skirts are slightly different, it is clear from the figure that independent skirtwidth control is provided by the proposed technique.

Figure 4 shows the difference in the amplitude envelopes between Rodet's technique and the proposed technique. The attack times match exactly but the rise and decay shapes are slightly different which makes sense given the spectral differences observed in Figure 3.

3.4 Comparison to SuperCollider

A particularly notable feature of this implementation comes from the design of the FOF envelope filter. If the bandwidth and rise parameters `bw` and `a` are set equal to each other, which causes the poles μ_1 and μ_2 to be equal, the filter in Eq. (5) becomes a second order filter with repeated poles. Additionally, for `bw` < `a`, the roles of the parameters are just reversed so that `a` controls the decay time and bandwidth whereas `bw` controls the rise time and skirtwidth.

This is in contrast to the behavior of the `decay2` and `formlet` uGens in SuperCollider. With `bw` = `a`, both objects produce a constant signal of zero and when `a` < `bw`, `decay2` produces an inverted envelope and `formlet` similarly produces an inverted FOF wave burst.

4. FAUST IMPLEMENTATION

The FOF filter structure developed in Section 3.1 was implemented using the FAUST (*Functional Audio Stream*) programming language [18]. FAUST allows for the quick prototyping and development of DSP algorithms with short, succinct lines of code. The FAUST code compiles down to C++ code that can then ported to standalone applications, externals for a variety of other computer music languages or embedded within a larger C++ project.

4.1 A Basic FOF Generator

A single FOF generator consists of an amplitude envelope, which is generated using a second-order filter, multiplied by a sinusoidal oscillator. In order for good quality synthesis using this configuration, the sinusoidal oscillator must be hard-synced to the same starting phase at the beginning of each new amplitude envelope. At the time of this writing, the FAUST libraries did not contain a hard-syncing oscillator, so the wavetable oscillator `osc` found in the library `music.lib` was modified to include that functionality (Listing 1).

Listing 1. Hard-Syncing Wavetable Oscillator

```
// import FAUST music library
ml = library("music.lib");

// resettable phasor, clock val > 0 resets phase to 0
ph(f0,c) = inc : (+ : d)~ (-(_<:(_,*(_,clk)))) : *(ts)
with {
        clk = c>0;
        d = ml.decimal;
        inc = f0/float(ml.samplingfreq);
        ts = float(ml.tablesize);
      };

// sin lookup table with resettable phase
oscpr(f0,c) = rdtable(ml.tablesize, ml.sinwaveform,
                      int(ph(f0,c)));
```

In particular, the phasor that controls table-lookup was modified to reset the phase to zero every time a non-zero clock signal is received.

Next is the code for a FOF generator. The generator takes four parameters:

- `fc`: the center frequency in Hz of the formant region
- `bw`: the bandwidth in Hz which controls the decay rate of the amplitude envelope
- `a`: the rise-time bandwidth in Hz which controls the rise time of the amplitude envelope
- `g`: a linear gain factor where $\texttt{g} = 1$ corresponds to a 0 dB peak frequency response

Additionally, the FOF signal block must be connected to a clock signal which should impulse the FOF generator at the desired fundamental frequency. The corresponding FAUST code for the FOF generator is given in Listing 2 and the block diagram in Figure 5.

Listing 2. FOF Generation System

```
// import FAUST filter and music libraries
fl = library("filter.lib");
ml = library("music.lib");

// function to generate a single Formant-Wave-Function
fof(fc,bw,a,g) = _ <: (_',_) :(f * s) with {
  T  = 1/ml.SR;              // sampling period
  pi = ml.PI;
  u1 = exp(-a*pi*T);
  u2 = exp(-bw*pi*T);
  a1 = -1*(u1+u2);
  a2 = u1*u2;
  G0 = 1/(1+a1+a2);          // dc magnitude response
  b0 = g/G0;                 // normalized filter gain
  s  = oscpr(fc);            // wavetable oscillator
  f  = fl.tf2(b0,0,0,a1,a2); // biquad filter
};
```

Taking the code from Listing 1 and Listing 2, it is possible to generate high quality vocal synthesis. However, it is necessary that the formant regions, as controlled by the bandwidth and rise-time parameters, are held constant or varied slowly over time so that audible artifacts are not introduced by time-varying the filter coefficients.

4.2 Filter Cycling and Coefficient Management

Since the FOF bursts typically overlap but the filter coefficients need to remain fixed during the audible duration of a single FOF burst, it is desirable to develop a more robust control structure to allow for the realtime manipulation of the FOF envelope parameters. This robustness can be achieved by introducing filter cycling and a sample-and-hold mechanism on the decay and rise filter coefficients.

Listing 3. Cyclic Impulse Train Streams

```
// import the oscillator library
ol = library("oscillator.lib");

// impulse train at frequency f0
clk(f0) = (1-1')+ol.lf_imptrain(f0)';
// impulse train at frequency f0 split into n cycles
clkCycle(n,f0) = clk(f0) <: par(i,n,resetCtr(n,(i+1)));
// function that lets through the mth impulse out of
// each consecutive group of n impulses
resetCtr(n,m) = _ <: (_,ctr(n)) : (_,(_==m)) : *;
// function to count nonzero inputs and reset after
// receiving x of them
ctr(x) = (+(_)~(negSub(x)));
// function that subtracts value x from
// input stream value if input stream value >= x
negSub(x)= _<: (_>=x,_,_):((-1*_),_,_):((_*_),_):(_+_);
```

With filter cycling, n identical filters are implemented in parallel and the impulse train is distributed so that the jth impulse is routed to the $[(j \bmod n) + 1]$th filter. Thus, if the filter coefficients are held constant using a sample-and-hold mechanism until the next impulse is received and enough filters are used so that each filter has sufficiently decayed in amplitude prior to its next impulse arriving, the filter coefficients can be swept as quickly as desired with no audible artifacts.

To handle the distribution of the impulses to the n different filters in FAUST a resetting series of counting mechanisms were developed (Listing 3, Figure 6). The main function in the listing is `clkCycle` which cyclically distributes unit gain impulses among n different output streams. The function `clk` outputs a single unit gain impulse stream at the frequency provided in the input argument. It is a slight modification of the function `lf_imptrain` included in FAUST's `oscillator.lib` library that compensates for a one period delay in the arrival of the first impulse. The

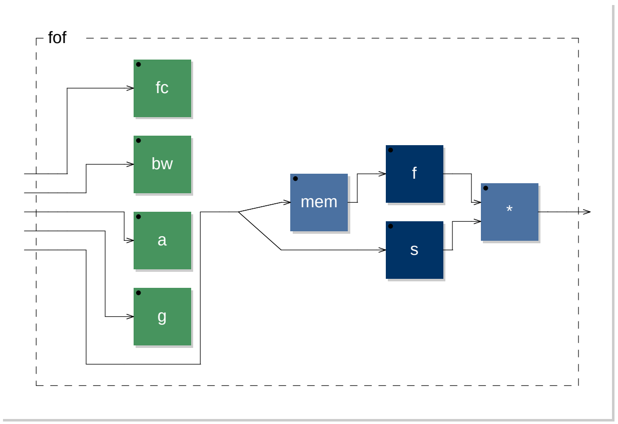

Figure 5. Block diagram of the code from Listing 2

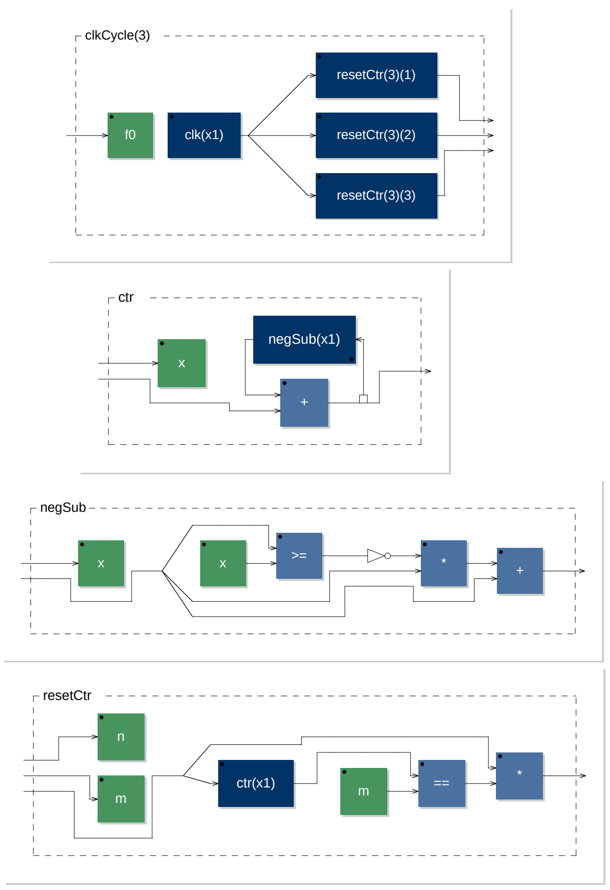

Figure 6. Block diagrams corresponding to Listing 3

functions `ctr`, `resetCtr` and `negSub` work in tandem to accomplish the internal management that `clkCycle` needs to function correctly.

With the distribution of the impulse train accomplished, it is then necessary to implement the sample and hold mechanism. The `filter.lib` library provides a sample-and-hold filter called `latch` which takes two signals: an input signal and a clock signal. It samples the input signal every time the clock signal goes nonnegative and always outputs the currently held value. Thus, to implement sample and hold, it is just necessary to connect the relevant parameter signals to `latch` then feed `clkCycle` to `latch` and feed a one-sample delay of `clkCycle` to `fof` so that the sampling occurs just before the filter is excited. The code and block diagram for the sample-and-hold mechanism and the corresponding FOF implementation are provided in Listing 4 and Figure 7.

Listing 4. FOF with Sample-and-Hold

```
// import the filter library
fl = library("filter.lib");

// sample and hold filter coefficients
curbw = (_,bw) : fl.latch;
cura = (_,a) : fl.latch;

// FOF sample and hold mechanism
fofSH = _ <: (curbw,cura,_) : (fc,_,_,g,_') : fof;
```

With the sample-and-hold mechanism in place, all parameters can be mapped from GUI elements in a standalone or DAW plugin or can be manipulated via control signals in another computer music language.

Listing 5. Example Program

```
/************** parameters/GUI controls **************/
// fundamental freq (in Hz)
f0 = vslider(``F0'',220,0,2000,0.01);
// formant center freq (in Hz)
fc = vslider(``Fc'',800,100,6000,0.01);
// FOF filter gain (in dB)
g  = ml.db2linear(vslider(``Gain'',0,-40,40,0.01));
// FOF bandwidth (in Hz)
bw = vslider(``BW'',80,1,10000,1);
// FOF attack value (in Hz)
a  = vslider(``A'',90,1,10000,1);

// number of S&H cycling filters
n = 5;

// main process
process = clkCycle(n,f0) <: par(i,n,fofSH) :> _;
```

The code in Listings 1–4 is all that is necessary to perform FOF synthesis using FAUST. A simple complete example FAUST program is provided in Listing 5. The program generates a single FOF wave stream with the bandwidth, rise time, center frequency, gain and fundamental frequency values controlled externally or by GUI elements. The program features five identical FOF streams running cyclically in parallel. Keep in mind that the code from Listings 1–4 are not being included in example program listing for brevity's sake but would need to be included in the actual program in order for it to compile and run.

5. CONCLUSIONS

In this paper a system for FOF synthesis was presented that uses a hybrid filter–wavetable oscillator architecture. The convolution of two exponential decay one-pole filters is used to generate an exponentially decaying amplitude envelope with smoothed attack. The amplitude envelope is then multiplied by a hard-synced wavetable sinusoid generator to generate FOF wave bursts which can then be used for FOF synthesis. The proposed technique was implemented in the FAUST audio programming language. Finally, filter cycling and sample-and-hold mechanisms were

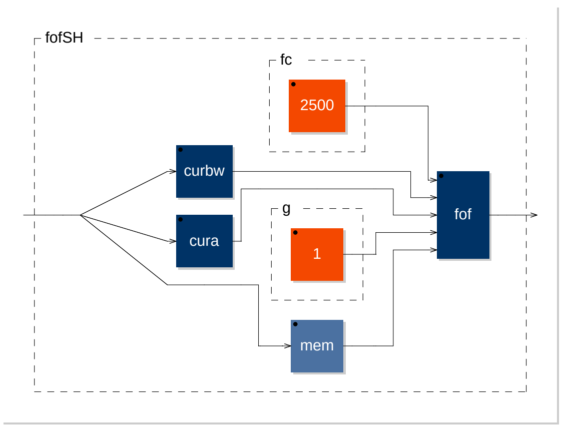

Figure 7. Block diagram corresponding to Listing 4

added to improve the robustness and flexibility of the synthesis technique.

A potential next stage in this research would be to develop a gain correction computer based on the fundamental frequency of the synthesized tone. The gain computer would be used to automatically normalize the increase in energy and, hence, volume that occurs when the fundamental frequency is increased which causes an increase in the overlapping between each successive FOF wave burst.

It would also be of potential interest to develop an alternative FOF envelope filter that does not suffer from the limitation of the maximum rise time being coupled to the value of the bandwidth parameter. That would lead to a hybrid technique capable of producing the extremely narrow skirtwidths possible with the original time-domain FOF synthesis technique.

Finally, some of the features of the Csound FOF implementation [10] such as octaviation, controlling different FOF streams with different fundamental frequency clock signals and using the FOF envelope with signals other than pure sinusoids could be added to the proposed implementation.

Acknowledgments

Michael Jørgen Olsen would like to acknowledge fruitful and insightful conversations with Romain Michon, Elliot Kermit-Canfield, Paul Batchelor and Maximilian Rest.

6. REFERENCES

[1] X. Rodet, “Time-domain formant-wave-function synthesis,” Computer Music Journal, vol. 8, no. 3, pp. 9–14, 1984.

[2] X. Rodet, Y. Potard, and J. B. Barrière, “The CHANT project: From the synthesis of the singing voice to synthesis in general,” Computer Music Journal, vol. 8, no. 3, pp. 15–31, 1984.

[3] M. R. Schroeder, “Vocoders: Analysis and synthesis of speech (a review of 30 years of applied speech research),” Proc. IEEE, vol. 54, pp. 720–734, May 1966.

[4] J. L. Kelly and C. C. Lochbaum, “Speech synthesis,” Proc. Fourth Int. Congress on Acoustics, Copenhagen, pp. 1–4, September 1962.

[5] L. R. Rabiner, “Digital-formant synthesizer for speech-synthesis studies,” The Journal of the Acoustical Society of America, vol. 43, no. 4, pp. 822–828, 1968.

[6] D. H. Klatt, “Software for a cascade/parallel formant synthesizer,” The Journal of the Acoustical Society of America, vol. 67, no. 3, pp. 971–995, 1980.

[7] P. R. Cook, “Singing voice synthesis: History, current work, and future directions,” Computer Music Journal, vol. 20, no. 3, pp. 38–46, 1996.

[8] J. B. Barrière, F. Iovino, and M. Laurson, “A new CHANT synthesizer in C and its control environment in PATCHWORK,” in Proc. Intl. Computer Music Conf., Montréal, Canada, Oct. 16–20 1991, pp. 11–14.

[9] J. M. Clarke, P. Manning, R. Berry, and A. Purvis, “VOCEL new implementations of the FOF synthesis method,” in Proc. Intl. Computer Music Conf., Kologne, Germany, Sept. 20–25 1988, pp. 357–371.

[10] J. M. Clarke, “FOF and FOG synthesis in Csound,” in The Csound book: Perspectives in software synthesis, sound design, signal processing and programming, R. Boulanger, Ed. MIT Press, 2000, pp. 293–306.

[11] P. Depalle, D. Matignon, and M. Stroppa, “Source-filter formulation and analytic control of the skirtwidth of CHANT formant-wave-functions,” in Proc. Intl. Computer Music Conf., San Jose, USA, Oct. 14–18 1992, pp. 372–373.

[12] J. R. Spanier, S. Johnson, and A. Purvis, “Optimisations of the FOF algorithm for VLSI implementation,” in Proc. Intl. Computer Music Conf., Hong Kong, Aug. 19–24 1996, pp. 493–495.

[13] J. M. Clarke and X. Rodet, “Real-time FOF and FOG synthesis in MSP and its integration with PSOLA,” in Proc. Intl. Computer Music Conf., Singapore, Sept. 29–Oct. 4 2003.

[14] J. McCartney, “Formlet,” 1996. [Online]. Available: http://doc.sccode.org/Classes/Formlet.html

[15] ——, “Decay2,” 1996. [Online]. Available: http://doc.sccode.org/Classes/Decay2.html

[16] K. Lee and J. Abel, “A reverberator with two-stage decay and onset time controls,” in Proc. Audio Eng. Soc. (AES) Conv., vol. 129, San Francisco, CA, Nov. 4–7 2010.

[17] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth, “On the Lambert W function,” Adv. Comput. Math., vol. 5, no. 4, pp. 329–359, 1996.

[18] Y. Orlarey, “Faust,” 2002. [Online]. Available: http://faust.grame.fr/