Unsampled Digital Synthesis: Computing the Output of Implicit and Non-Linear Systems

David Medine
University of California, San Diego
Swartz Center for Computational Neuroscience
La Jolla, USA
dmedine@ucsd.edu

ABSTRACT

The author discusses what we call Unsampled Digital Synthesis (UDS), a technique for creating synthesis networks that uses continuous time models as its building blocks. These models are comprised of ordinary differential equations that describe some atom of dynamic behavior. Once the equations are known and networked together, a numerical integrator is employed to compute out the actual audio samples that correspond to the output of the network. It is shown that the use of UDS can simplify and improve some of the existing digitization schemes for solving non-linear and implicit equations that commonly present themselves in physical modeling, virtual analog, and computer music generally.

1. INTRODUCTION

In computer music synthesis, one may wish to eschew discrete time equations due to the ramifications of unit delay (a necessity in digital feedback). Unit delay is the basic building block of digital filters, digital waveguides (DWGs) and other staple computer music tools. However, when an implicit relation exists, unit delay makes for an inaccurate model. [1] In mechanical and analog electrical systems, such relations abound. It is principally in these cases that UDS can unlock the door to a more satisfactory computer emulation.

UDS may also simplify the digital synthesis of certain non-linear systems. Often times in physical modeling, when one wishes to represent physically accurate non-linearities (such as the elongation of a plucked string at the moment of plucking [1], or the non-linear characteristics of a clarinet reed [2]) the technique is to treat these parts of the system in some special way – i.e. with a lookup table – and then couple the non-linearity to the rest of the (linear and non-implicit) digital network. UDS can simplify this framework because non-linear equations may be solved numerically just as easily as linear ones. [2] While the details of carrying out numeric integration of dynamical systems are beyond the scope of the present discussion, the curious reader may refer to [3].

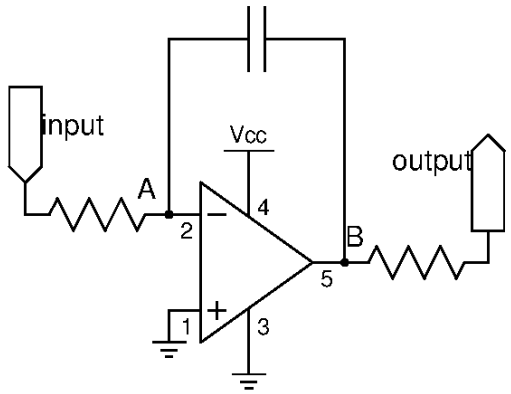

Figure 1. A basic analog integrator.

2. PREVIOUS WORK

By definition, models of analog circuits involve implicit relations. In the schematic shown in Figure 1 points *A* and *B* are implicitly related to each other. At any moment in time, the voltage and current at point *A* is a function of the voltage and current at point *B and* the voltage/current pair at point *B* is a function of the voltage/current pair at point *A*.

The circuit simulation program SPICE [4] is an exemplar of what we here refer to as UDS. Implicit networks of continuous time (and often non-linear) circuit element models are arbitrarily connected and a digital integrator is used to incrementally compute the state at every node. SPICE, however, is not meant for real time synthesis, nor is it intended for audio synthesis.

BlockCompiler [5] and the Sound Design Toolkit [6] are examples of computer audio tools that can enact some of the

[1] We define an implicit relation as one wherein the states of two or more variables depend on some combination of each other's *current* states, *not* some combination of their *past* states.

[2] Although, it must be acknowledged, numerical integrators do struggle with discontinuous equations.

techniques proposed here; although, at the time of writing, neither appear to be under active development. The work of Bilbao concerning finite difference method schemes [7] is also suggested reading for anyone interested in the present topic. In that work, the goal is to solve accross multiple dimensions (e.g. partial differential equations). There, the schemes are designed on a case by case basis. It is not an attempt to provide a framework for concocting arbitrary networks on the fly.

3. EXAMPLES OF UDS

3.1 A Quadrature Oscillator

Recall that the derivative of the cosine of x is the negative of the sine of x and that the derivative of the sine of x is the cosine of x. Therefor, in order to create a quadrature oscillator, all one needs to do is to evaluate the implicit pair of functions:

$$\dot{y} = -\omega x \tag{1}$$

$$\dot{x} = \omega y. \tag{2}$$

[3]

Here, ω is proportional by a factor of 2π to the desired oscillator frequency ($f = \omega/2\pi$ Hz). One caveat is that the above pair of equations can be evaluated correctly for any amplitude. We can control this amplitude with initial conditions (say $y = 1.0$ and $x = 0.0$).

3.2 FM

Since UDS can approximate a delay-free network, we may instantiate two oscillators and use them to frequency modulate each other. This cannot be done with 'traditional' computer music techniques like lookup oscillators because there will necessarily be at least one sample phase error between at least two of the oscillators in the network. Consider, for example, the Pure Data patch showing a reciprocal FM network in Figure 2. It is evident that at any point in time, the frequency of oscillator B depends on the current state of oscillator A, but the frequency of oscillator A depends on the state of oscillator B at least one sample in the past. Furthermore, in order to reduce the delay from the usual 64 samples-wide delay to 1 sample-wide delay, we must decrease the computation block size to 1 sample. Thus, this patch is inaccurate as a model of an analog synthesizer patch of the same topography.

UDS provides a methodology for digitizing such a circuit. We create a continuous time model given by:

$$\dot{y}_n = -(\omega + \lambda_n)x_n \tag{3}$$

and

$$\dot{x}_n = (\omega + \lambda_n)y_n. \tag{4}$$

Here, there are n oscillators that modulate one another (given by x_n and y_n), and λ_n is some linear combination of eachother's outputs:

$$\lambda_n = \alpha_{n1}x_1 + \alpha_{n2}x_2 \ldots \alpha_{nm}x_n, \tag{5}$$

[3] In this paper we stick to the convention of writing differentiation with respect to time using 'dot' notation: $\dot{x} \triangleq \mathrm{d}x/\mathrm{d}t$.

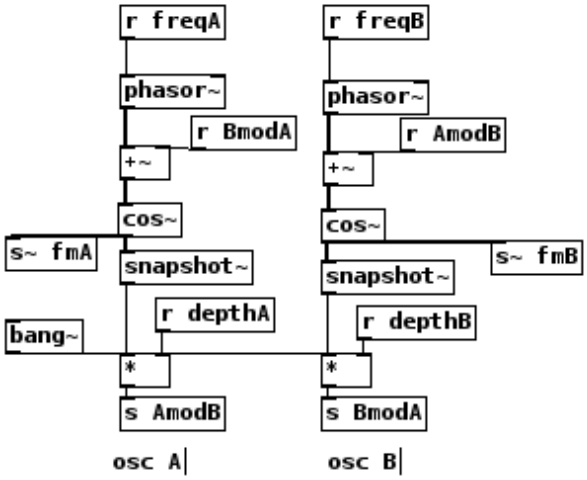

Figure 2. A Pure Data patch depicting two reciprocating FM oscillators.

where the coefficients α_{nm} define the depth of modulation on oscillator n by the current ('continuously' varying) state of oscillator m. Note that these oscillators modulate themselves when α_{nm} is non zero for $n = m$.

3.3 Oscillator Sync

Another example of virtual analog that cannot be precisely represented without a delay-free loop is the classic analog technique of oscillator sync. In analog synthesizers, one oscillator may be 'slaved' to another so that when its 'master' crosses a threshold, the phase of the slave is reset. This is a common tool for creating complex timbres in analog synthesis and it can be emulated in the traditional digital ways. However, if we wish to slave the master to its slave there is an implicit relationship between the instantaneous phases of each oscillator. With UDS we can achieve accurate emulation of such a network.

There is a small problem with resetting the phase in our UDS model: numeric integrators don't handle discontinuities very well. A way to avoid a discontinuity is to have a rule that says if the oscillator is to reset its phase, then proceed directly and quickly, but continuously, towards $\phi = 0$. For example, we say that when a master crosses the threshold, and so long as the slave has not yet reached its initial phase, the slave's path is given by:

$$\dot{y} = (1 - y)G \tag{6}$$

$$\dot{x} = (0 - x)G, \tag{7}$$

where G is some gain factor. This rule forces y towards 1 and x towards 0 (corresponding to the desired initial phases of the cosine and negative sine functions). When the sync regime is not in effect, we simply use Equations 1 and 2.

Figure 3 shows two oscillators that are each slave and master to the other (the x values are plotted). If such a scheme were implemented with digital delay, one of the oscillators would be one block of samples behind the other at the moment of

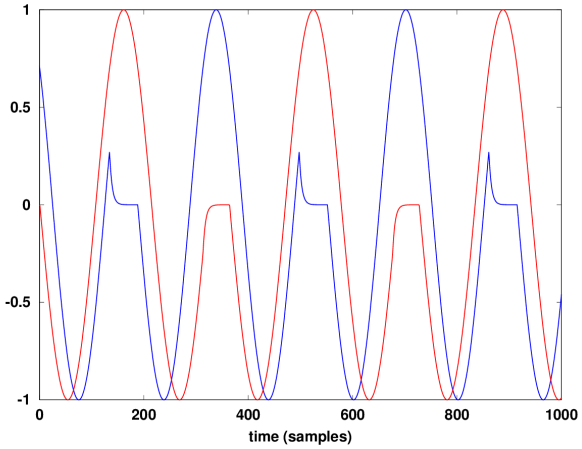

Figure 3. Both oscillators are masters and slaves of one another.

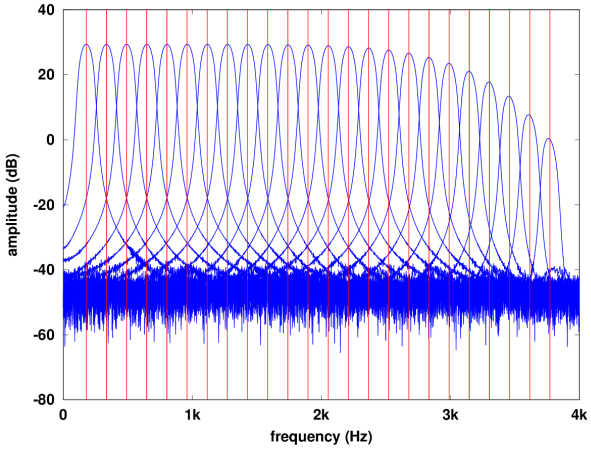

Figure 4. Bode plots for the UDS Moog filter.

sync. That is to say its phase would reset late by at least one sample because the threshold of the master would have been crossed in the previous sample block.

3.4 The Moog Ladder Filter

Digital implementations of the Moog ladder filter (a non-linear, four-stage analog network) abound [8][9]. Schemes used to digitize this analog filter require unit delay to induce resonance. This unit delay causes the resonance parameter r to couple with the gain parameter g which controls the cut-off frequency. This coupling occurs because the unit delay in the feedback loop causes additional phase shifting so that the effect of the resonance parameter varies with frequency and vice versa [10].

The non-linear, implicit, ODE describing one stage of the filter is given by:

$$\dot{V}_i = g(\tanh(V_{i-1}) - \tanh(V_i)) \tag{8}$$

where g is the gain that governs the cutoff frequency of the filter, $\dot{V}_i$ is the change in voltage over time, V_i is the voltage at stage i, and V_{i-1} is the voltage at the previous stage for $i = 1,2,3$. Ideally, the actual cutoff frequency has a ratio of $f_c = g/2\pi$ for all frequencies.

To induce resonance, at the top of the filter ($i = 0$) we sum the input to the filter with $-rV_3$.[4] Thus the equation for the first stage of the filter is:

$$\dot{V}_0 = g(\tanh(V_{input} - rV_3) - \tanh(V_0)) \tag{9}$$

When the filter is given a resonance of $r = 4$ (1 for each stage), it will self-oscillate. Shown in Figure 4 are Bode plots of the UDS implementation of this filter after it is stimulated with an impulse. These measurements were made by setting $r = 4$, feeding the filter a unit impulse signal and incrementally increasing the cutoff frequency f_c. The vertical red lines

[4] The sign is negative because each stage of the filter shifts the phase of the input by $\pi/4$ radians. Thus the feedback has opposite phase and will emphasize frequencies (resonate) when it is inverted [8].

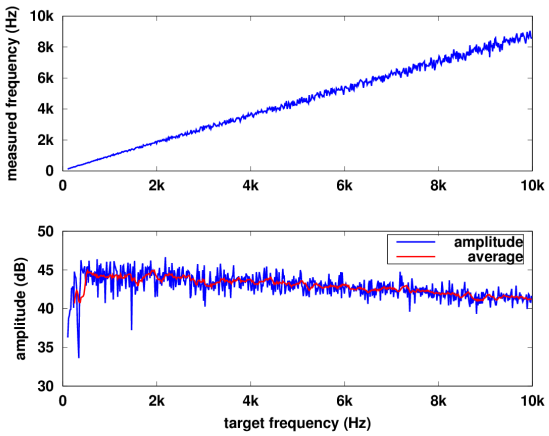

Figure 5. Measured frequency and amplitude plotted against target frequencies of the UDS Moog filter stimulated with noise ($r = 4.0$).

indicate the value of f_c during the analysis period. It is clear that the target frequencies are very near the peaks of the magnitude spectra, indicating a linear proportion between r and f_c. It is also evident (but unfortunate) that the filter looses energy once it crosses $f_c \approx 2.5$ kHz. This is due to error in the integrator (Runge-Kutta) used to evaluate the filter. This can be improved by using a more accurate solver.

We can look at the frequency and amplitude response of the filter when stimulated with constant white noise (Figure 5). Here the constant noise at the input keeps the filter ringing, but we see that there is a 'soft knee' in the response around $f_c \approx 2.5$ kHz. At this point the filter output becomes less focused (the noise begins to appear more readily in the spectrum), slower (the pitch literally flattens), and slightly lower in amplitude as well. This 'sag' occurs at the same point that we start to see energy leak in the impulse response measurement, which comes as no surprise.

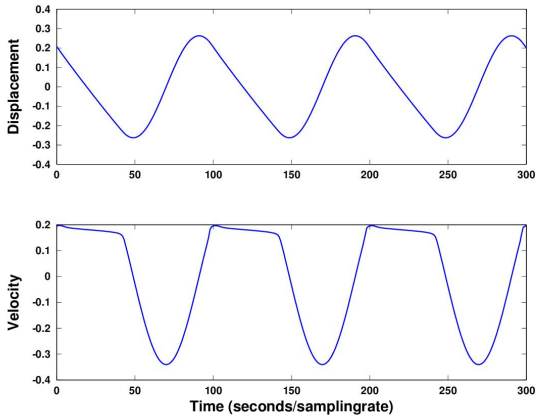

Figure 6. The displacement and velocity of a bowed oscillator.

3.5 A Bowed Oscillator

The physics of bow-string interaction is well understood. The curious reader may be referred to [11] and [12] for details. The model is an implicit one as the influence of bow friction depending both on the bow parameters as well as the instantaneous velocity at the point of bowing. To simplify this example, we shall couple a bow model to a simple harmonic oscillator, rather than a whole string. Drawing from the physics of bow-string interaction we have the following equation to describe the system:

$$\ddot{u} = -\omega^2 u - F\phi(v - v_b), \tag{10}$$

where u is displacement the oscillator, $\omega/2\pi$ is the fundamental frequency, F is the force of the bow, v is the velocity of the oscillator, v_b is the velocity of the bow, and ϕ is our non-linear friction model. [5] This can be written as a pair of first order equations, one for velocity, the other for displacement:

$$\dot{v} = \omega u - F\phi(v - v_b) \tag{11}$$

$$\dot{u} = -\omega v. \tag{12}$$

Equations 11 and 12 are evidently those that give the quadrature oscillator, Equations 1 and 2, with an extra term to model the bow's driving friction. Shown in Figure 6 is displacement and velocity over time. Plots of this nature are typical in the literature.

4. FUTURE WORK

Conspicuously missing from the above is a discussion of methods for implementing UDS. It is with UDS in mind that 'timelab' was created [13]. This C language API is currently evolving and aspires to become an interpretative language so that interactive audio programming (so desirable and available in Pure Data, SuperCollider, etc.) is possible. At the moment, it is a fully functional API and framework for programming UDS routines. It is also available as a Pd extern so that these UDS synthesis can be run within Pd.

[5] In the results shown in the current example, $\phi(x)$ is given as the integrator friendly $\phi(x) = \sqrt{2a}xe^{-ax^2+1/2}$.

5. REFERENCES

[1] T. Smyth, J. Abel, and J. O. Smith, "A generalized parametric reed model for virtual musical instruments," *Proceedings of ICMC 2005*, pp. 347–350, 2005.

[2] K. Legge and N. Fletcher, "Nonlinear generation of missing modes on a vibrating string," *The Journal of the Acoustical Society of America*, vol. 76, no. 1, pp. 5–12, 1984.

[3] D. Morgan and S. Qiao, "Accuracy and stability in mass-spring systems for sound synthesis," in *Proceedings of the 2008 C 3 S 2 E conference*. ACM, 2008, pp. 69–80.

[4] L. W. Nagel and D. O. Pederson, *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

[5] M. Karjalainen and C. Erkut, "Digital waveguide vs. finite difference schemes: Equivalence and mixed modeling," *EURASIP J. Applied Signal Processing (June 2004)*, pp. 978–989.

[6] "Sound Design Tooklkit," http://soundobject.org/SDT/, accessed: 2015-05-25.

[7] S. Bilbao, *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. Wiley Online Library, 2009.

[8] T. Stilson and J. Smith, "Analyzing the Moog VCF with considerations for digital implementation," in *Proceedings of the 1996 International Computer Music Conference, Hong Kong, Computer Music Association*, 1996.

[9] V. Välimäki and A. Huovilainen, "Oscillator and filter algorithms for virtual analog synthesis," *Computer Music Journal*, vol. 30, no. 2, pp. 19–31, 2006.

[10] A. Huovilainen, "Nonlinear digital implementation of the Moog ladder filter," in *Proc. Int. Conf. on Digital Audio Effects (Naples, Italy, October 2004)*, 2004, pp. 61–4.

[11] L. Cremer and J. S. Allen, *The physics of the violin*. MIT press Cambridge, MA, USA:, 1984.

[12] S. Serafin, "The sound of friction: real-time models, playability and musical applications," Ph.D. dissertation, Stanford University, 2004.

[13] D. Medine, "Timelab: Yet, Yet Another Audio Programming Environment," in *Proceedings of the International Computer Music Conference, Perth*, 2013.