The Sampling Theorem and Its Discontents

Miller Puckette
Department of Music
University of California, San Diego
msp@ucsd.edu

After a keynote talk given at the 2015 ICMC, Denton, Texas.

August 28, 2016

The fundamental principle of Computer Music is usually taken to be the Nyquist Theorem, which, in its usually cited form, states that a band-limited function can be exactly represented by sampling it at regular intervals. This paper will not quarrel with the theorem itself, but rather will test the assumptions under which it is commonly applied, and endeavor to show that there are interesting approaches to computer music that lie outside the framework of the sampling theorem.

As we will see in Section 3, sampling violations are ubiquitous in everyday electronic music practice. The severity of these violations can usually be mitigated either through various engineering practices and/or careful critical listening. But their existence gives the lie to the popular understanding of digital audio practice as being "lossless".

This is not to deny the power of modern digital signal processing theory and its applications, but rather to claim that its underlying assumption—that the sampled signals on which we are operating are to be thought of as exactly representing band-limited continuous-time functions—sheds light on certain digital operations (notably time-invariant filtering) but not so aptly on others, such as classical synthesizer waveform generation.

Digital audio practitioners cannot escape the necessity of representing continuous-time signals with finite-sized data structures. But the blanket assumption that such signals can only be represented via the sampling theorem can be unnecessarily limiting. In Sections 4 and 6 I'll describe investigations by two recent UCSD graduates that each adopt a distinct approach to audio manipulation outside the framework of the sampling theorem.

A collection of accompanying patches that demonstrate some of these ideas can be downloaded from msp.ucsd.edu/ideas/icmc15-examples/.

1 The assumptions

Band-limited functions are a vector space: you can scale one of them, or add two of them, to get another. But that is where closure ends. The trouble begins as soon as we even go so far as to multiply one signal by another. Suppose two sampled signals, $X[n]$ and $Y[n]$, are used to represent two continuous functions of time $x(t), y(t)$, which we assume to be band-limited, containing only frequencies in the Nyquist frequency band, the interval $(-R/2, R/2)$ where R is the sample rate. The values can either be real or complex, and for simplicity we'll assume the computer can exactly represent the numerical values. (It isn't true but that is usually a comparatively minor issue).

There is, of course, a perfectly good continuous-time signal, call it $z(t)$, that is represented by the computable product, $Z[n] = X[n]Y[n]$. But it's not in general the case that $z(t) = x(t)y(t)$. We didn't in reality make the product of the two continuous-time signals we were representing when we multiplied their computer representations.

At this point we can look ruefully back at every occurrence of the character "*" in all the Csound, Pd, SuperCollider, Kyma, 4X, or MUSIC 10 instruments we've ever built and reflect on the fact that the result isn't really correct, if we regard our sampled signals as representing continuous-time ones. Often it's a very serviceable approximation. If, for instance, the signals $x(t)$ and $y(t)$ have frequency limits whose sum is less than $R/2$, the multiplication is exact; and when not exact, it is often a very good approximation. But the approximation's accuracy or lack thereof is rarely worked out explicitly.

We could always take action to band-limit two signals (by filtering them) before multiplying so that the multiplication itself doesn't yield frequencies outside the Nyquist frequency band. But this would cause delays and/or phase distortion, not to mention the computational cost this would incur.

One fundamental operation in electronic music practice (in my thinking, the most fundamental one) is table lookup, which is used in digital oscillators and samplers, and also in nonlinear techniques such as FM and waveshaping. Again sidestepping the comparatively minor issue of the accuracy limits of wavetable lookup, we instead again consider the possibility of frequency products landing outside the Nyquist band. Suppose the incoming signal is a sinusoid of frequency ω and that the wavetable lookup can be approximated as a power series,

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots$$

The highest possible frequency product of the kth term ($a_k x^k$) is $k\omega$. If the function is a polynomial (thus stopping at a finite k) then the situation is at least in principle possible to control by limiting $k\omega$ never to exceed $R/2$ (whether by fiat or by filtering). But for any other choice of $f(x)$ the result is in general not band limited at all, and some foldover is inevitable.

There is a reasonably broad class of operations that can be carried out without departing from the safe zone of the Nyquist theorem. One can record and play back sounds. Delay networks and filters (both recursive and not) are safe as long as the coefficients do not change in time. One can spatialize sounds using level panning. But this still leaves a majority of electronic music practices that cannot be guaranteed band-limited in practice; in addition to the examples of FM and waveshaping cited earlier, even additive synthesis, which would seem to be safely band-limited at first thought, is in reality not, since at a very minimum we have to multiply the component sinusoids by time-varying envelopes. If, for example, these envelopes are constructed using line segments, then every envelope breakpoint gives rise to non-band-limited frequency products dropping off in amplitude as ω^{-2}. The resulting foldover is often inaudible but it is not hard to concoct situations in which it is not.

The most ready defense against the distortions arising from digital sampling is to train one's ears to hear it and, as necessary, adjust parameters or raise sample rates until it is no longer audible. But to learn to hear this, a young electronic musician would need examples of clean and dirty signals to compare. It is possible that practice will erode over the years as ears gradually get used to hearing foldover, until perhaps one day few people will have heard a cleanly synthesized sound, in much the same way that few North Americans or Europeans have ever tasted a tree-ripened banana.

2 Example of a non-band-limited signal representation

Any system for representing continuous functions digitally will only be able to exactly represent a small subset of all possible functions, and/or to approximate, more or less well, functions that can't be exactly represented. Any particular choice of representation will imply a certain subset of functions that can be represented, and perhaps a way of choosing which representable function to swap in for one that is not representable. For example, sampling at a constant rate allows us to claim the subset of functions that are suitably band-limited and to approximate any other one by leaving out whatever lies outside the band limit. This is clearly an excellent choice for digital audio in general, but for some applications other choices might be preferable.

Here for example is another possible choice: we could choose to represent arbitrary piecewise linear functions of time by specifying the endpoints of the line segments. For example, a function like the one shown in Figure 2 could be represented by a sequence of triples:

$$(t_1, x_1, y_1),\ (t_2, x_2, y_2), \cdots$$

This would allow, for example, a sawtooth wave to be represented exactly. Certain operations (adding two such functions together, for example) could be carried out in the representation, but others (for instance, multiplying them) could not (although we could allow that as well if we extended the format to allow arbitrary piecewise polynomials... but I won't belabor the point here).

The interesting thing about this format is that it can exactly represent classes of functions that can't be represented using the sampling theorem. Although

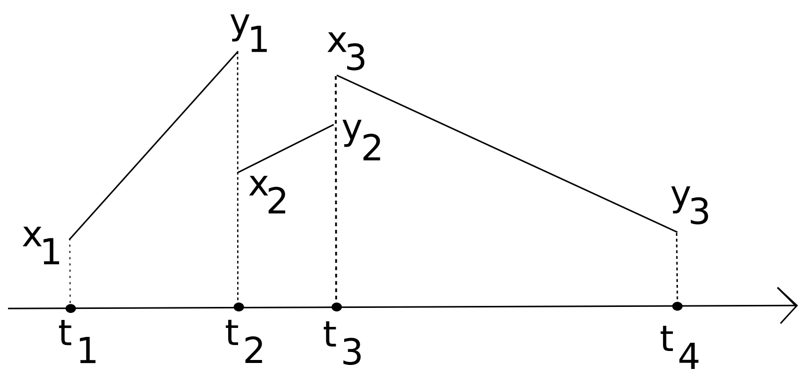

Figure 1: A digitizable representation of piecewise-linear functions of time

it is certainly less well adapted to the day-to-day operations of most electronic musicians than sampled functions would be, there is at least one piece of music that would have been quite naturally expressed in this way: Xenakis's S709, a few microseconds of which are shown in Figure 2, and which is described in *Formalized Music*[9] with an appendix showing a code listing of Marie-Hélène Serra's implementation. The piece is realized by generating repeated copies of a line-segment waveform in which the vertices vary at random, successively from cycle to cycle; the number of segments may vary as well. This is at least an existence proof that a line-segment-based signal representation may lead naturally to signal manipulations that at least some composers might find musically useful.

3 Violating the theorem's assumptions

On the subject on the sampling theorem, we should not forget that the whole practice of electronic music as sampled audio signals, and indeed the now-ubiquitous use of wavetables for sound synthesis, dates back to Max Mathews. Mathews himself was trained as an engineer and always took care to let people know about the limitations of the technology. Around 2007 he was showing visitors to his laboratory a wonderful demonstration which, since I haven't seen it published, I'll repeat here. The accompanying patch is "mathews-table-lookup-example.pd".

Mathews's idea is to put a square pulse in a wavetable (in my example, I put a one-sample-wide pulse in a 200-element table) and then to scan it,

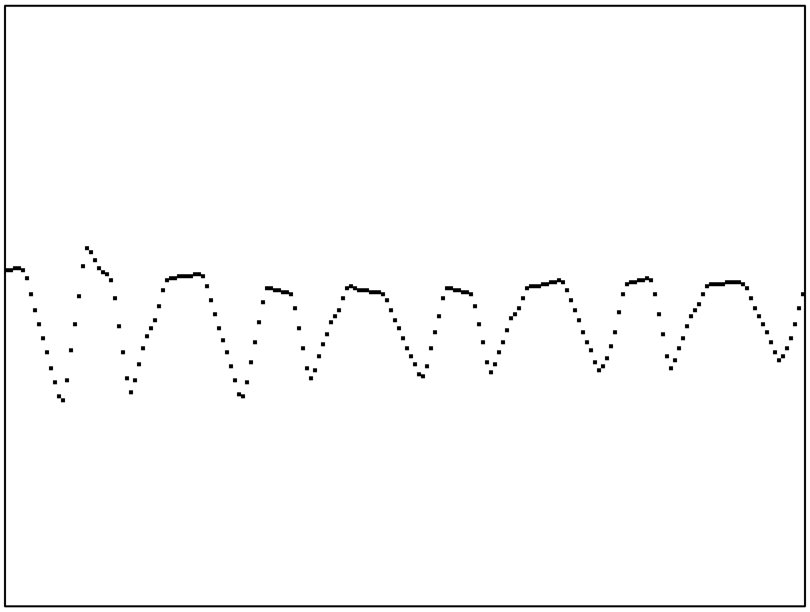

Figure 2: 200 samples of Xenakis's S709

without interpolating, with the phase advancing by various sampling increments. Choosing a sampling increment of $1/n$ where n is an integer (taking the table lookup domain to be from 0 to 1), you get a clean pitch of R/n where R is the sample rate. (This assumes that the phase accumulation itself is done to arbitrarily high precision before applying the non-interpolated table lookup). Choosing an arbitrary sample increment gives a characteristically dirty sound.

We now choose a sample increment almost equal to 1/100 but slightly detuned. If the sample increment were exactly 1/100, you would either hear a sound if the phase happened to pass between 0 and 1/200, but silence if the phase passes between 1/200 and 1/100 (thereby skipping over the pulse). Since the slight detuning makes the phase drift alternately between these two cases, we unexpectedly hear a tone that toggles on and off. You can think of this as a beating pattern between an infinite series of foldover products that just happen to line up to make a square-wave modulation of the tone.

4 Example: modeling the Moog ladder filter

We now consider one interesting way to approximate continuous-time processes in a computer, using numerical differential equation solvers instead of sampled processes. In this discussion I'll rely heavily on work by recent UCSD PhD graduates Andrew Allen[1] and David Medine[5].

A good starting example is the famous Moog ladder filter design[6], a con-

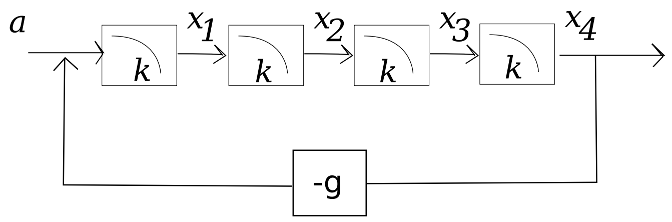

Figure 3: block diagram representation of the Moog ladder filter

ceptual block diagram view of which is shown in Figure 4. Each low-pass filter in the diagram is a one-pole design whose cutoff frequency (k, in radians per unit time) is voltage-controlled. (Here we're not showing the elegant circuit design that realizes this; Moog not only had to find a good signal processing model but also one he could realize with bipolar transistors).

This block diagram leads to the following system of ordinary differential equations:

$$\dot{x_1} = k \cdot [S(a - gx_4) - S(x_1)]$$
$$\dot{x_2} = k \cdot [S(x_1) - S(x_2)]$$
$$\dot{x_3} = k \cdot [S(x_2) - S(x_3)]$$
$$\dot{x_4} = k \cdot [S(x_3) - S(x_4)]$$

Here S denotes a nonlinear saturation function to reflect the fact that in any real circuit realization of the network, the filters' output would be limited by the available power supply. This is a good thing of course, because the filter can be made unstable by turning up the feedback gain g. We'd rather the outputs of the filters saturate than merely vaporize the planet as would otherwise happen when g first exceeded 4.

The usual and somewhat schematic explanation of how this filter works is that, at the frequency k, each low-pass filter retards the signal by 1/8 cycle, so that the four of them retard it by 1/2 cycle, so that multiplied by $-g$ the feedback path is in phase with the input (at $g/4$ times the amplitude), so that the circuit resonates. The difficulty of digitizing this circuit stems from the fact that in a digital realization there will be at least a one-sample delay in the feedback path, thus changing the frequency at which resonance occurs. This change can be quite significant; for instance, if k is set to one quarter of the sample rate we pick up a fifth quarter-cycle, so we would expect the resonant frequency to be off by a minor third (20%). The filter is often used as an oscillator, in which usage this will be heard as a tuning error—and it would be reasonable to ask that one control an oscillator's frequency to within a few cents, perhaps 1000 times better than the naïve digital implementation does. If we assume linearity this can be corrected satisfactorily using standard DSP techniques [7]; but if we take the nonlinearities fully into account it takes much hard work[3] to overcome the problems that result from digitizing the Moog ladder filter.

What I propose here will sound facile, and perhaps it is: why not go back to the differential equations and apply a traditional numerical ODE solver to them? Very little brainpower is required. One simply goes to the Wikipedia page for "Runge-Kutte" and types the familiar four-step version into a Pd extern. This is the basis of the `bob~` object released with Pure Data.

This approach has the disadvantage that it requires far more computation to generate output samples than the DSP approach does. If your end goal is a stand-alone product (software or hardware) that imitates the historical Moog ladder filter, it may well be worth the research and development time (months or years) required to implement one using the work cited above. But on the other hand, if your aim is to explore one or another possible refinement of, or deviation from, the modeled filter then you would have to re-do all this work for each possible modification. Furthermore, without any real filter to test your results against, you could never know how accurate your modeling really is.

For one thing, we cannot automatically assume that the many idealizations built into our model aren't causing us to lose something in translation[8]. To know that for sure we would have to make comparisons, one by one, of the simplified model against one in which each simplifying assumption was replaced with a more realistic one. This is feasible using numerical methods, but would be onerous to do using DSP techniques.

But things get even more interesting when we consider possible variations on the filter design itself (leaving aside the question of whether a "real" circuit might exist to exhibit them). After all, there is something self-defeating in the idea of using contemporary technology to try to recreate sonic experiences from the past, when instead we could be looking for new ones.

To make just one example, suppose we decided that the cutoff/resonant frequency k depended on the internal state of the filter, for instance taking one value when state variable x_1 is positive and a different one otherwise. If you drove such a filter to oscillation ($g \geq 4$) you would get a sort of self-FM, and if instead (or in addition) you drove it with an incoming sound you could get a variety of effects. This idea is realized in the accompanying example, `bentbob-test.pd`.

You could make all sorts of other changes; for instance changing the number of stages from four to eight or twelve, possibly making several taps with independently controllable feedback coefficients, inserting input signals at more than one point in the circuit, making the saturation function asymmetrical, and so on without end.

This line of exploration should not be confused with the idea of simulating or modeling actual circuits. One could do that with the Spice circuit simulator, for example. But such an approach has several disadvantages. First, you have to design a real circuit, which is much harder to do than to arrange low-pass filters as described in the functional block diagram above. Also, "real" circuits are much harder to simulate, since they do not yield explicit expressions for the derivatives of the state variables; instead, a system of simultaneous equations must be solved to compute the derivatives. (In physical terms, this is because real electronic components don't have "inputs" and "outputs"; instead, causality flows bidirectionally along each physical wire.)

Instead, what we have here, as David Medine proposes, is a block-diagram-based system of components each of whose output's derivative is a function of its state and inputs, in such a way that we can construct a modular synthesis environment that is realizable in systems of differential equations in explicit form, readily solvable using straightforward techniques such as Runge-Kutte. Although the software doesn't exist yet, this could easily be made into a graphical patching language for quickly exploring all manner of dynamical systems.

5 Two more dynamical systems

The Moog filter simulation above is an example of a dynamical system, which is only to say, "it's a system that can be written as a set of simultaneous first-order differential equations, solved for the derivative terms". Such a system can be visualized as in Figure 5. Here the system of equations describes a simple forced oscillator:

$$\dot{x} = -ky + (1 - x^2 - y^2)x + f(t)$$
$$\dot{y} = kx + (1 - x^2 - y^2)y$$

This can be thought of as a vector field, where the points are possible states of the system and the vectors are the time derivatives which show how the current state flows through the state space. The flow may depend on time; in this example there's a forcing function $f(t)$ imposed from elsewhere. (The vector field is drawn in the figure under the condition $f(t) = 0$).

When not being forced, this oscillator converges to the unit circle where the term $1 - x^2 - y^2$ disappears; the result is simple harmonic motion. As with the Moog filter when pushed into oscillation, this example gives various results when forced with a sinusoid tuned a minor third or so from the natural oscillating frequency. This system is realized in the example named `forcedosc-test.pd`. (In truth it is much less interesting sonically than the Moog example, but its conceptual simplicity makes it suitable for a range of extensions that will not be explored here.)

For another example of a dynamic system, the famous Lorenz attractor is included as `lorenz-test.pd`. Here, for convenience, in addition to the usual parameters α, β, ρ there is a speed parameter, in MIDI units, that simply scales all the time derivatives so that the model runs globally faster or slower. The output can either be listened to directly (by connecting one or another state variable directly to a loudspeaker) or used to control the pitch of a sinusoidal oscillator—I find the latter choice the more interesting to hear.

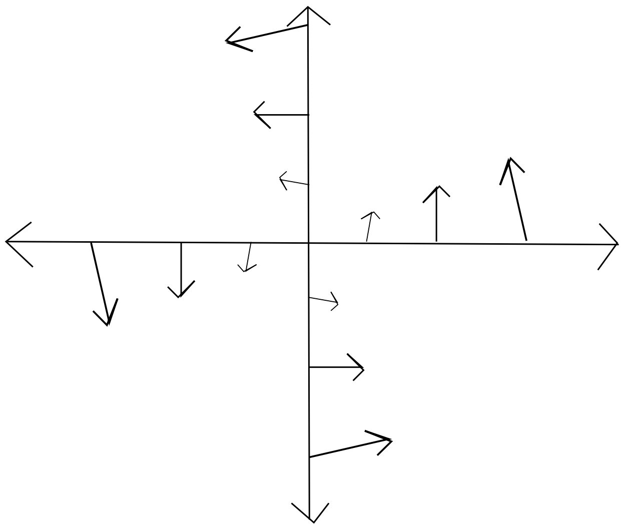

Figure 4: Example of a dynamical system: a forced oscillator.

6 Ruratae: unphysical modeling

Andrew Allen takes continuous-time modeling in a quite different direction, realized in his Windows game (to use the word loosely), named Ruratae. Here the model is that of a physically vibrating network of interconnected objects, much as in physical modeling systems such as Cordis Anima[2] or Modalys[4]. Unlike those systems, the emphasis here is not on modeling exactly a real physical system. Such modeling has limitations similar to those of circuit modelers as either would be applied to music synthesis: expertise is required to “build” reasonable sounding instruments, and once the instruments are built they cannot be quickly modified.

Ruratae takes a higher-level approach, in which fanciful collections of point masses are connected by generalized “springs” that may exhibit nonlinear responses, damping, and/or may snap when elongated past a maximum value. The system makes no distinction between the act of building an instrument and of playing it. The user hears the instrument vibrating in reaction as masses and connections are added or deleted (or snap). This encourages a highly intuitive and exploratory style of instrument design.

Compared to dynamical systems in general, Ruratae’s focus on idealized physical systems constrains them in a way that allows a tailored approach to solving the equations. Allen found that, although Runge-Kutte has excellent stability properties, in practice it tended to attenuate high frequencies. After trying several solvers, he settled on Beeman’s method as a good trade-off, for this problem space, between fidelity at high frequencies, computation time, and stability.

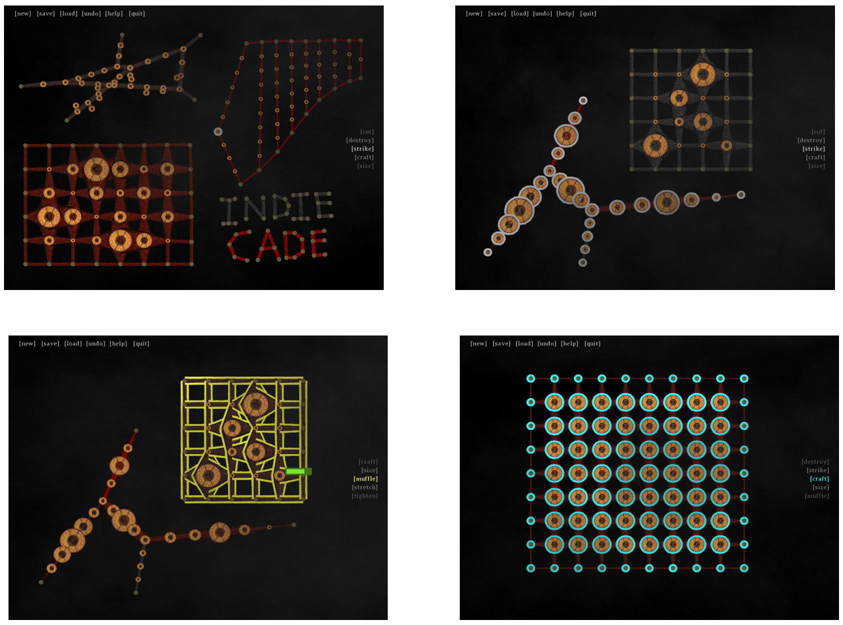

Figure 5: Screen shots from Andrew Allen's Ruratae software (reprinted from his PhD dissertation).

A carefully optimized graphical rendering system (Figure 6 visualizes the vibrations of the system in real time at computer-game-worthy frame rates (the graphical optimization was tricky and system-dependent, which is why the game runs only on Windows).

To draw a conclusion from the work of both Allen and Medine, the exploration of ODE systems is still uneven terrain where no single approach is without its own particular set of limitations. At the same time, both approaches are powerful and offer much potential to build compelling and fun computer music instruments. This should continue to be an active area of research.

7 Uniform flows on locally flat surfaces

We turn now to a very different possible approach to modeling continuous-time processes. Returning to the idea of using dynamical systems as audio generators, we propose a methodology for designing ones for which we can find exact solutions despite the availability of interesting non-periodic behavior. Specifically, we can impose a constant vector field as the flow, so that locally we get motion in straight lines. Interesting results can come from connecting flat sheets together in geometries that have cantankerous global properties.

A physical system that suggested this approach is pictured in Figure 7. Two ideal mass-spring systems, with equal masses but tuned to different frequencies,

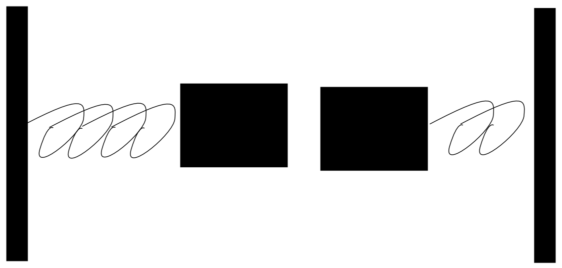

Figure 6: Dynamical system: two colliding, detuned, ideal mass-spring systems.

are held at a distance apart so that they collide, either occasionally or constantly. Collisions are elastic: each mass recoils at its speed of incidence as if it had bounced off a hard surface. (This isn't really correct; the masses should in fact exchange velocities; but it is much easier to model this way since each oscillator's energy then stays fixed.)

This is realized in the provided example patch named `coupled-sampled.pd`. Both of the two systems are assumed to oscillate with amplitude 1, and can thus be represented by their phases ϕ_1, ϕ_2, which we take to range from $-\pi$ to π, and equal to 0 when a spring is at its most stretched. At moments where the phases are such that the two masses come into contact, say at $phi_1 = -\phi_a$ and $phi_2 = -\phi_b$, we simply advance the phase so that they are in the same location but moving away from each other instead, that is, wrapping around forward to phases $\phi_1 = +\phi_a$ and $\phi_2 = +\phi_b$. To be exactly correct, we should measure by what amount the two phases have exceeded the values at which the collision occurs and the rebound phases should be forwarded by the same amount, but the provided patch does not take care of this detail.

Here then is an analysis of the behavior of the system, slightly further simplified but presented in a way that can readily be generalized. The phase space is a square whose coordinates are the two phases, with a centered, diagonally oriented square corresponding to points at which the two masses would occupy the same space (see Figure 7. This is a simplification; in the original physical model the forbidden areas are not truly triangles. Many other boundary shapes could be used instead.)

Without the middle square cut away, the phase space would be a torus and the flow would be a constant vector field, so that trajectories would be the familiar geodesics known to players of 1960s-vintage SPAWAR. The missing square acts as a wormhole in the space. Whereas the dotted path in the figure represents a possible trajectory in the absence of the wormhole (so that the two oscillators advance independently), in the presence of the wormhole the trajectory is altered as shown by the solid path.

We can then listen to any suitably smooth function of the phase space. For instance, to hear a mixture of the two oscillators we would choose the function $\cos(\phi_1) + \cos(\phi_2)$, but other choices abound. We would require only that the function take the same value on any two diametrically opposed points so that

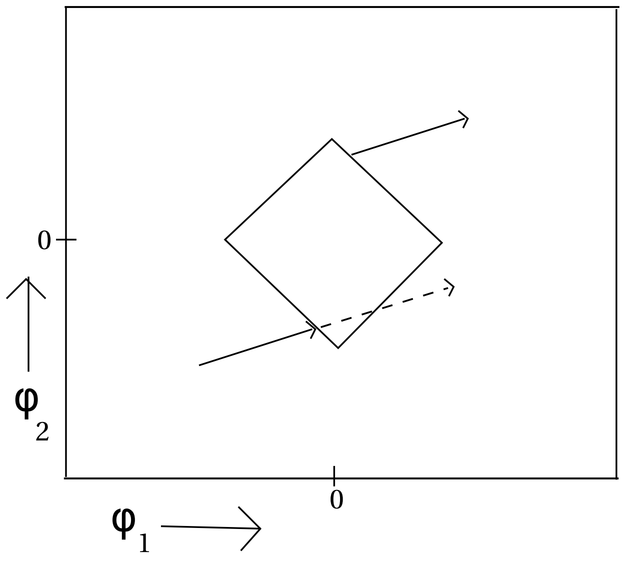

Figure 7: Trajectories through toroidal phase space: dotted path, normal; solid path: with wormhole.

the result of crossing the wormhole is continuous. (If we wish, we could work somewhat harder and arrange for matching slopes as well.)

The whole scheme could easily be extended to higher-dimensional spaces (representing more than two oscillators) with as yet unexplored results. Even with only two dimensions, a variety of rich interactions between the two oscillators can be quickly found.

The interesting thing about this model is that it allows for exact solutions. To know our position in phase space at any point in time, we merely propagate forward in a straight line until we hit a boundary (at a time point that in general won't be an integer number of samples at any fixed sample rate). Whenever we reach a boundary, we jump to the diametrically opposed boundary point and continue as before. This gives us a list of segments in a format similar to that of Figure 2. To listen to the output, we convert it to a sampled signal.

8 Observations and conclusions

Early Bell-Labs-resident composers such as James Tenney, Jean-Claude Risset, and Charles Dodge set out a theory and praxis of computer music that many composers have since followed, privileging precise execution of carefully specified and planned-for musical desiderata. The hankering of late twentieth-century Western composers for order and structure fit in perfectly with the computer's ability to accurately manipulate data, and their musical practice did not suffer much from the computer's main early failing: the impossibility of real-time audio computations. It is in a spirit of appreciation for their contributions that I am here exploring the spaces beyond the pale they constructed—if for no other reason, for the light it sheds on what we're doing as we follow in their footsteps.

Meanwhile, traditional musical instruments (especially that most traditional one, the human voice) refuse to give up their secrets, and remain capable of musical gestures that no computer can yet imitate. Part of the secret undoubtedly lies in the real-time interaction between player and instrument, and perhaps another aspect is the complexity and inherent unpredictability of the physical processes that take place inside the instruments.

It is no accident that all the examples I have invoked here are in one way or another unpredictable. Because of this they practically require real-time exploration to unlock their musical possibilities. In this respect they are all also beholden to another tradition perhaps best exemplified by Michel Waisvisz's famous Crackle Box. They lie on the fringe of what is considered correct electronic music practice. Fringes are interesting loci, and any reasonably complex domain will have many of them; so even if each individual one is limited in range their aggregate might offer a large range of possibilities. Besides, what seems like a fringe one day might be understood as the mainstream sometime in the future (for example: electronic music itself).

References

[1] A. S. Allen, "Ruratae: a physics-based audio engine," Ph.D. dissertation, University of California, San Diego, 2014.

[2] C. Cadoz, A. Luciani, and J. L. Florens, "Cordis-anima: a modeling and simulation system for sound and image synthesis: the general formalism," *Computer music journal*, vol. 17, no. 1, pp. 19–29, 1993.

[3] A. Huovilainen, "Non-linear digital implementation of the Moog ladder filter," in *Proceedings of the International Conference on Digital Audio Effects (DAFx-04)*, 2004, pp. 61–64.

[4] F. Iovino, R. Caussé, and R. Dudas, "Recent work around modalys and modal synthesis," in *Proceedings of the International Computer Music Conference.* International Computer Music Association, 1997, pp. 356–359.

[5] D. Medine, "Dynamical systems for audio synthesis: Embracing nonlinearities and delay-free loops," *Applied Sciences*, vol. 6, no. 5, p. 134, 2016.

[6] R. A. Moog, "A voltage-controlled low-pass high-pass filter for audio signal processing," in *Audio Engineering Society Convention 17.* Audio Engineering Society, 1965.

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

[8] T. E. Stinchcombe, "Analysis of the Moog transistor ladder and derivative filters," Citeseer, Tech. Rep., 2008.

[9] I. Xenakis, *Formalized music: thought and mathematics in composition.* Pendragon Press, 1992, no. 6.