REPRODUCIBILITY AND RANDOM ACCESS IN SOUND SYNTHESIS

Hanns Holger Rutz *Eduardo Miranda*

University of Plymouth

Interdisciplinary Centre for
Computer Music Research

Gerhard Eckel

University of Music
and Performing Arts Graz

Institute of Electronic Music
and Acoustics

ABSTRACT

We elaborate on the feature of reproducibility and random access in the process of electroacoustic composition. Random access has an intrinsic relationship to concrete sound material by means of the cutting operation. We propose a concise model of the electroacoustic composition process and within it locate the importance of random access. Random access is then interpreted in the light of real-time sound synthesis and the potential and limitations of such interpretation are shown. Implementation approaches are demonstrated on the SuperCollider Server. A final discussion questions the object of effigy in the reproduced sound structure, instead arguing for the concept of a differential reproduction.

1. INTRODUCTION: TWO METAPHORS

The basis of the software technology used in contemporary electronic and electroacoustic music production has been laid with multitrack hard-disk recording applications (Protools, Cubase Audio, Logic Audio ...) and real-time sound synthesis applications (CSound, Pure Data, MSP, SuperCollider ...) becoming available on standard personal computers in the early and middle 1990s respectively. The ubiquity of their metaphors for representing sound and sound structures leads one to forget the historical impact they made on the way we organize and compose sounds, but also masks that these two types of applications have barely converged in the last fifteen years.

While the sound synthesis applications borrow either from the visual metaphor of a patch cord based modular synthesizer (PD, Max) or the conceptual metaphor of circuits and block diagrams (CSound, SuperCollider), the multitrack editing applications derive from the hardware tape recording machines. The former highlight the fact that sounds are the result of transformations which can be perpetually recombined, possibly with a live interaction during a performance. And while the latter also offer the possibility of engaging with live automation of sound parameters, they clearly focus on the "tape" as the manifestation of a stored sound that can be observed detached from its performance time.

In the course of this text, we will elaborate on the possibility of bringing the particular "tape" mode relationship to sound to the world of real-time sound synthesis systems. Although convergence means that *both* sides move towards each other, the perspective of the sound synthesis systems is taken as their programmability suggests that this side is the more general or encompassing one and that, as a consequence, an implementation is more easily carried out from this side.

2. FROM SURGICAL CUT TO THE TRIM BIN

In characterizing the quality of the image in the new medium of film, Walter Benjamin [1] had picked the profession of the surgeon as an analogy to the cameraman. While the traditional painter created a closed entity, which Benjamin compared to the work of a magician. The reasons for this comparison may be manifold. On the surface of his text, the focus is on distance and bodily relation between medic and patient—the intermediate technical apparatus. But also the surgical profession had undergone a major change with the introduction of anesthesia in the middle of the 19th century, coinciding with the early steps in the field of cinematography, which would in the later discourse of the "mass media" be attributed a narcotic potential.

In this context, focus is put on the procedure of precise cutting and re-combining, which of course formed the basis of the 1950s musique concrète. The composer would now assume successively the roles of the cameraman—the microphone being the acoustical camera—and the film editor, creating a new time series from fragments in the cutting room. The possibility to record, that is to freeze time, and to reproduce the frozen fragments at a later point, is a crucial precondition, although, as Klaus Schöning states, it is the montage ex post which transcends the purely mechanical reproduction by creative interference:

> «While the phonograph made acoustic recordings possible even before the turn of the twentieth century, the silent film was able to record optical images which—in contrast to the acoustic recordings—could be cut and rearranged. The art of film montage preceded the art of acoustic montage by decades.» [8]

The concept of random access has a peculiar relationship towards both disk and tape recordings. On the one

hand, the disk provides instant access to reproduction of all of the stored information, a kind of time canvas which merely requires a displacement of the needle to navigate to a particular spot. On the other hand, the tape provides access to randomly cutting and splicing parts of it, although cueing to a specific position is cumbersome. This second interpretation is what James A. Moorer considers important about random access:

> «In the days of manual editing, tape-based editing was essentially random-access, since new material could be inserted at any position in the tape. In film and sound-effects editing, this is still the case. The "trim-bin" becomes the random access storage and retrieval device: every different sound (or picture) has a different hook in the trim bin (or around the room) and the editor can swiftly retrieve any piece of sound and splice it into place.» [5]

Of course, in the contemporary hard-disk recording software both aspects of random access have been united, as the disk head moves quasi instantaneously to any position, allowing the physical cut-and-splice to be replaced by an in-memory edit list which, as it is displayed in the form of a timeline, produces the illusion of an edited tape, the tape-head being represented by a moveable cursor.

Reading "randomness" as chance operation rather than arbitrariness creates the missing link between the world of fixed tape composition and algorithmic approaches to music. Although he does not further investigate on this striking new meaning of "random access", Peter Manning notes that the trim bin was essential as a random selection mechanism for the realization of John Cage's Fontana Mix tapes in 1958 [4]. According to Schöning, Pierre Schaeffer later acknowledged that Fontana Mix was «more concrete than anything done in the early years of the Parisian studio» [8], and this underlines the importance of such a radical approach of "random access" for the conceptualization of electroacoustic music.

3. AGGLOMERATION AND DIVISION

The facility of the recorded sound to be subject to arbitrary precise cuts makes it, as musical material, distinct from the symbolic abstraction found in the notation of instrumental music. Although a symbol such as a crotchet can be chopped up, for example into two quavers, this procedure is very different from cutting an already complex realized sound into a beginning and ending (revealing the cut as a cut, or dividing an articulatory entity), or of taking a fragment from a very long amassed stream of sound such as an ambiance recording, or creating fragments of such short duration that the perceptual quality of sound changes completely (as in some types of granulation).

An electroacoustic composition is not so much formed from catenating symbols, but from taking complex entities and dividing them. Moreover, the so obtained fragments can be overlayed again or glued together to form new agglomerates. These, in turn, may be subject in another iteration to different cuts. Since the *form* of the material is completely neutral—a constant stream of sample frames—, it is possible to apply generalized transformations on these, the field which is called digital signal processing (DSP).

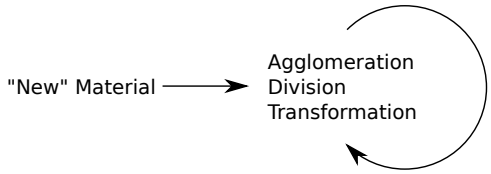


Figure 1: The composition process as a recursive system

This model of the electroacoustic composition process is shown in figure 1. The general recursive form of course is not restricted to electroacoustic composition but can be seen as model for all kinds of composition. The circular edge could be labeled as "decision making". The diagram is deliberately kept simple and does not claim to represent every aspect of the process. For example, making notes and sketches is invisibly attached to the recursion edge.

It is through these recursive steps of inscription into the material or the musical structure that the composition is actually "written". Where "written" is not so much meant in the narrow literal sense, but rather in the Derridarian sense which includes «the entire field covered by the cybernetic *program*» and which is indifferent to whether the inscription is carried out by a human author or a machine (that is an automated algorithm in this case) [2, p. 9]. Likewise, we put "new" in *new material* in quotation marks, anticipating the final discussion of this paper.

Confronted with the analysis of Schaeffer, one can find congruence and dissimilarity. The three elements can be linked to his usage of décomposition and analyse—through montage—, mixage, modulation and filtrage [7, ch. 23–2]. We do not follow the normative aspects of this electroacoustic chain, namely the distinction between montage and mixage as matter-preserving (transformation) and filtrage as form-preserving (transmutation) and the teleology shining through the sequential nature of the chain links (as opposed to the recursive nature of fig. 1)[1].

What is important here is that random access means the *guarantee* that the output of each iteration has the kind of neutral form that allows it to be subject in the next step to another general transformation or procedure of division (cutting up, thinning out, ...) and agglomeration (amassing, mixing together, overlaying, ...). It has been said that sound rendered as evenly sampled digital frames is the neutral form of the hard-disk recording application. One can apply transformations such as frequency filtering to chunks of the edit list, and "flatten" the result into another sampled sound—through an action called, again with reference to physical tape recorders, "bouncing".

Speaking with Agostino Di Scipio, we could say that in real-time sound synthesis systems «the array of DSP

[1] cf. the hierarchy of the operations of the solfège in [7, ch. 23–6]

algorithms, and the methods by which they communicate among themselves, should be seen as the material implementation of a compositional process or concept» [3]. The challenge is then to look at the form of such DSP algorithms and their interconnections in order to prepare a review of the possibilities to endow them with a random access mechanism.

4. SOUND SYNTHESIS BUILDING BLOCKS

The architecture we chose to conduct this investigation is the SuperCollider Server, mainly because the sonic structures are dynamically created as the result of code execution, a property which makes this system particularly suitable to experimental modification. However, this goes without loss of generality, since other sound synthesis systems such as Pure Data, Max/MSP and CSound are based on similar ideas such as a DSP graph made from simple building blocks—unit generations (UGens)—which are connected through signal and message slots.

The UGens are semi-opaque in that we generally have a knowledge of how they operate, although the precise internals of the their algorithms are not known (unless the C source codes are studied in great detail). A UGen has inputs for its customizable parameters, and one or more outputs that can be used as inputs of other UGens, forming multi-rooted acyclic graphs that do something meaningful on a higher abstraction level than a single UGen—for example output on a certain speaker a sine tone with a modulated frequency and an amplitude envelope. For this example, a sine oscillator UGen, named `SinOsc`, which has inputs for frequency and phase, would be connected to another oscillator that modulates the sine's frequency. The output of the sine oscillator is multiplied, using a `BinaryOpUGen`, with an envelope generator, say `EnvGen`. The result of the multiplication is fed to the speakers using an `Out` UGen.

Unlike the visual patcher systems, in which boxes permanently exist and are permanently wired, SuperCollider is a dynamic system in which these UGens do not exist timelessly. Instead, by evaluating a function, the required graphs are created on the fly. For reasons of efficiency and abstraction, sub-graphs are encapsulated into what is called SynthDef. The SynthDef is the description of the graph along with a name to identify it, it is thus similar to a class in an object-oriented programming language. Like a class, the SynthDef can be instantiated, creating an actual graph in which each UGen has its state—such as the current phase of the sine oscillator—and which is executed on the real-time DSP scheduler, until the graph is eventually destroyed again. The instantiated graph is called Synth and is associated with an integer identifier, so that several instances of the same SynthDef can be distinguished. Special UGens exists to parametrize the Synths, so that for example two instances of the same SynthDef describing the modulated sine oscillator can be created and customized to use different modulator frequencies.

The instantiation and parametrization of the Synths is performed by a client that connects to the SuperCollider server. This is illustrated in figure 2(a). It can be seen that the client is issuing different actions over time (passing on the horizontal axis). The organization of time is thus divided between client and server: The client discretely decides to run or stop a particular Synth, while the UGens on the server-side produce continuously sampled calculations such as oscillations, envelopes, etc. Figure 2(a) also shows that there are two types of resources, Buses and Buffers, which are administrated by the client and can be accessed by the Synths. The bus system can be used to connect different Synths.

```
SynthDef(\disk, { arg buf, amp = 1;
  var disk = DiskIn.ar(2, buf);
  Out.ar(0, disk * amp)
})

SynthDef(\verb, { arg gate = 1;
  var in, env, left, right, verb;
  in   = In.ar(0, 2);
  env  = EnvGen.kr(Env.asr(2), gate);
  #left, right = in * env;
  verb = FreeVerb2.ar(left, right, 1, 0.8, 0.8);
  FreeSelf.kr(TDelay.kr(Done.kr(env), 3));
  Out.ar(0, verb)
})
```

Figure 3: SuperCollider Language representation of the SynthDef diagrams of figure 2(b).

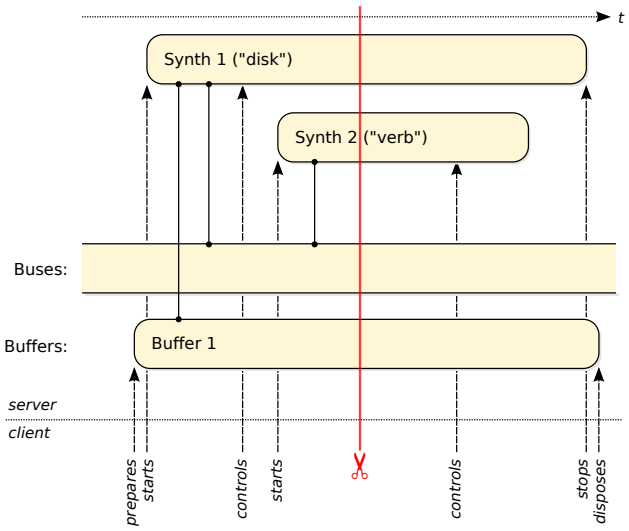
The code describing the SynthDefs of figure 2(b) in the default client—the SuperCollider Language—is shown in figure 3. The connections between a UGen and its inputs are made by passing in the inputs as arguments to the instantiation methods `ar` (for audio rate calculations) and `kr` (for calculations at control rate, a fraction of audio rate[2]). The `Control` UGens, which allow the client to parametrize the Synth, are implicitly created from the arguments to the SynthDef's function, i.e. `buf` and `amp` in the case of the `"disk"` SynthDef which reads a sound file as a stream from hard-disk, multiplies it by a gain factor and writes it onto bus 0, and `gate` in the case of the `"verb"` SynthDef which reads the signal from bus 0, applies an amplitude envelope, puts it through a reverberator, and mixes the result back onto bus 0 (this eventually becomes the output of the sound card).[3]

5. STEPS TOWARDS AN IMPLEMENTATION

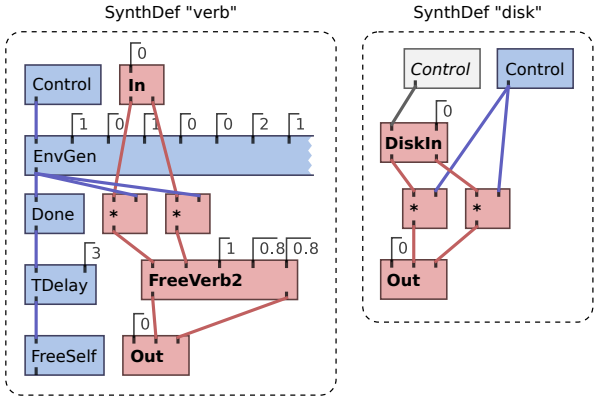
The attempt is now made to develop different approaches to random access for the sound structures created in the manner of SuperCollider.

[2]The audio rate corresponds to the sampling rate of the sound hardware. With a default fraction of 1/64 for the control rate, a typical setup has an audio rate of 44100 Hz and a control rate of approx. 689 Hz. Running slower changing processes at control rate saves CPU load.

[3]The two "cables" and the duplicate existence of the * UGen indicate that this is a stereo signal. While the code concisely represents multiple channels in single symbols, the diagram represents each channel of signal by a separate cable.



(a) A client instantiates (starts) Synth nodes on the server, customizes (controls) them, and eventually disposes (stops) them again. While the bus system is statically allocated, buffers must be dynamically created. The Synth nodes are ordered, so that the calculations for Synth 1 are performed just before those of Synth 2, allowing the output signal of Synth 1 to enter Synth 2 (via the indirection of a bus).



(b) The templates (SynthDefs) for the Synths in (a). Plain text / blue color indicates control rate calculations, bold-face / red color indicates audio rate, italics / grey color indicate scalar operations performed only at the initialization of the Synth. The number literals are also scalars (constants).

Figure 2: The SuperCollider Server architecture

5.1. Complete Reproduction of a Performance

The first requirement for accessing something ex post is that its conditions are somewhat stable or restorable. This is a rather weak requirement, as it reduces the necessary information regarding all deterministic elements to their initial conditions: No knowledge is needed about how the phase of an oscillator is advanced, since the play of sound structures is reproduced from time zero, and it suffices to reproduce the exact same input for frequency and phase. If frequency and phase are deterministic themselves, we are done.

In the example of section 4, the two SynthDefs *are* deterministic—when identically parametrized, the streaming of the sound file from disk as well as the reverberation will produce the exact same results. Assuming that the client has a measurable clock from which the time stamps of the commands sent to the server are derived, it would thus suffice to record these commands along with their time stamps, i.e. the time at which each synth was started, the position at which the buffer was cued, the time at which the volume was adjusted, and so forth.

Care needs to be taken of the initial state of the server, from trivial settings such as the sample rate and block size, to the seed of the random number generators (RNGs). It comes at no surprise that all seemingly indeterminate UGens such as noise generators (`WhiteNoise`, `PinkNoise`, `BrownNoise`, `Dust`, etc.) as well as the plain random number generating UGens (`Rand`, `TRand`, etc.) are based on a very fast pseudo-random number generating algorithm which is deterministic and merely depends on an initial *seeding* value. Since SuperCollider seeds the RNGs according to the computer clock when started, the seed needs to be explicitly overwritten using the special `RandSeed` UGen, and this new seed must be remembered.

So which are the truly indeterminate elements? These are all elements which depend on state other than the one defined by the setup of the server itself: Audio or control input from outside the system, most obviously live input from microphones or other audio sources and inputs from all kinds of physical sensors.[4] In a way these elements can be identified with the injection of new material in figure 1.

As we focus on the server, we will just assume that the client is able to record discrete events from the outside world, such as actions triggered by a GUI, MIDI controller, and so forth. In the standard distribution, the only discrete sensor input UGens monitor mouse coordinates, mouse button presses and keyboard typing, all of which are useful when playing around with the parameters of UGens, but are typically not found in the final sound structure (or at least could easily be replaced by equivalent client-side input).[5] One is therefore left with solving the problem of restoring input evenly sampled at control or audio rate from outside sources. Now the purpose of reproduction might be twofold:

(a) we might want to exactly replay a session in order

[4] These are only the most obvious sources, because all other input that is not synchronized with the server is also situated outside the system: Reading the computer's date or clock, retrieving a query from the internet, etc. Without loss of generality, we will focus on the live audio and sensor input as provided through the UGens found in the standard distribution of SuperCollider.

[5] Although these sensors are provided at control rate in their UGen variants, a client-side would most likely consider them as discrete events, so that for example no new data is produced for a mouse position, until the mouse is moved.

to pick up at a certain point in time in the fashion of "takes". In that case it may be necessary to record the live input signals to disk.[6]

(b) the live input is essential for the performance of the "piece", and hence the original `In` UGen which reads the live signal should not be replaced by a `DiskIn` replaying of what has previously been recorded.

The second case gets intricate when commands generated by the client are depending on the analysis of any of these live signals, that is when the live signal may cause structural changes. The more intricate when, as discussed in the next sections, considering situations of moving randomly into "the timeline", since—as it turns out—there is no more such thing as "the timeline". In this section's situation—a complete reproduction from time zero—the easiest way out is to not mix (a) and (b). In general, signals should not be recorded for a Synth β whose identity is unstable, meaning its duration or the number of times it is played depends on a live signal from another Synth α, because otherwise there would be no stable mapping between the instances of Synth β and the recorded signals.

5.2. Snapshot Markers

An extension of the previous approach is to allow the sound structures not only to be played back from the very beginning, but at any number of predefined points in time. That is to say, there is a mechanism for triggering the recording of snapshots of the currently playing synths, either manually or automated (e.g., taking snapshots at a regular interval). The red vertical line and scissor in figure 2(a) symbolize such a marker. To perform the snapshot, different categories of UGens must be considered:

Stateless These UGens have no internal state or memory, and thus they can be ignored in the analysis. Examples are the `UnaryOpUGen` and `BinaryOpUGen` operators (e.g. taking the square-root of a signal, multiplying or adding two signals, etc.), operators for constraining signal ranges (`Wrap`, `Fold`, `Clip`) or mapping ranges (`LinExp`, `LinLin`), selecting among different signals (`Select`) or panning a signal across several channels (`Pan2`, `PanAz` etc.), as well as UGens which instantly trigger a certain action (`SendTrig`, `Done`, `Pause`, `Free`)

Externally instructed These UGens have a state, but it is set externally by the client. An example is the `Control` UGen which allows to set parameters from the client while the Synth is playing. `Control` just passes these values on as input to other UGens, so its state always corresponds to the last message sent from the client. When creating a marker snapshot, the client must store the last control value it sent. Similarly, the `DiskIn` UGen which streams audio files from the hard-disk is instructed with a buffer cued to a certain position in a sound file. When it begins to play, it continues buffering from the audio file at the server's sample-rate. When a marker is set, the cue position for the audio file can thus be inferred and when replaying from the marker, the buffer can be exactly re-cued at the correct position.

Stateful, discrete These UGen use an internal state that can be represented with a few variables, so that when resetting these variables to their snapshot, the UGen will reproduce the original behavior. These include recursive processes such as chaotic oscillators, for example `Logistic`, implementing the logistic map formula $y_k = p \cdot y_{k-1} \cdot (1 - y_{k-1})$, where both p and the initial y can be specified. Likewise, all IIR filters work this way (although they use linear terms), such as `LPF` (lowpass), `HPF` (highpass), `BPF` (bandpass), or the general form `SOS` (second order section / biquad). Other UGens in this category are trigger based, such as `Latch` and `Gate` which perform a sample-and-hold, so they have a memory element that stores the last value that has passed the gate. A third subcategory are the oscillators—the sine oscillator has already been mentioned and it keeps account of an internally incremented phase. The problem with most of these UGens is that their internal state is *not directly accessible*, and we discuss workarounds further down.

Temporally constrained This applies to the spectral (phase vocoder) UGens which operate on FFT'ed signals. Since a full FFT frame is not available every control block (by default a control block contains 64 audio sample frames), this imposes a temporal grid on the possible marker positions. For example at an FFT size of 2048 samples and 50% overlap between windows, markers can only be generated every 1024 audio sample frames (or 16 control blocks). If several FFT based synths are running which have been started so that their respective FFTs are not completed in the same control block, a single marker cutting across all synths cannot be created.

Stateful, buffered These UGens require a sample buffer, typically of variable size, to work. Some UGens use internal buffers—which are thus inaccessible—such as the `FreeVerb2` reverberator, others use a client-provided buffer, such as the delay line realized by the pair of `DelTapWr` and `DelTapRd`. Some UGens can be exchanged for others, for example the comb filter UGen `CombN` which uses an internal buffer, can be readily replaced by `BufCombN` which provides the same functionality with a client-

[6] Although there is currently no UGen that does this at control rate, as a workaround the control rate data can be upsampled, written to a regular audio file, and downsampled again in the reproduction. Also, a third party UGen exists that interleaves several control signals into one audio signal, so bandwidth would be saved. Of course, preferably one would add a control rate version of the `DiskOut` UGen which records a stream to disk.

provided buffer. Nevertheless, in both cases these UGens pose a problem as buffers not only may take up a large storage space, in particular if the snapshots are taken frequently, but they can also not be accessed synchronously—querying the buffer data from the server is an asynchronous action and cannot be performed simultaneously with the rest of the snapshot taking.

This list defines specific properties of UGens of which combinations are also possible. The spectral UGen `PV_MagFreeze` for example comes both with the temporal constraint of the FFT and an internal buffer.

There are two solutions to the missing access to internal state: Either a UGen is substituted for one or several others that perform the identical operation but provide resettable state. Or the UGens themselves are modified to include extra inputs and outputs for managing their state from the client-side. We have conducted an experiment with the first solution, which has the advantage that no C code needs to be written (no server plug-ins need to be modified), but the disadvantage of taking extra CPU time as the number of UGens involved increases. The test case is a rather simple sound object, a sine oscillator which alternates between glissandi and stable frequencies, provided by a random ramp generator.

The original SynthDef, shown in figure 4, contains 6 UGens, while the modified version, shown in figure 5, creates 33 UGens. The graph transformations, of course, would not be carried out manually, but by an automatic rewriting stage in the SynthDef finalization. The missing phase state of most oscillators can be emulated by using the `LFSaw` UGen, while the `LFNoise1` needs a more complex massage to capture the phase dependent triggering of new random numbers. Since the random seed at the time of the snapshot taking must be known, the workaround here is to choose a new seed in the client and the snapshot taking is carried out by setting the trigger control named `t_memo` to the new seed value. This trigger adjusts the seed using the `RandSeed` UGen, and reports back the current state, formed by variables `o_state0` to `o_state5`, using the `SendReply` UGen. The Synth can then be restarted at the snapshot point by feeding this state information into the scalar `Control` UGen (formed by SynthDef arguments `i_state0` to `i_state5`), and the sound will be identical to the original down to sample level.

5.3. Convergence

It remains to answer the question of buffered UGens. Snapshots clearly cannot be taken, but with the introduction of a *convergence* property for a set of UGens this may not be actually a problem in many cases. We say a UGen α is convergent if we can take at any time a copy α' which has exactly the same inputs—however not necessarily the same internal state—and the magnitude of the difference between the output of α and α' approaches ε, a very small number denoting the roundoff noise in floating-point math.

```
SynthDef(\gated, {
  Out.ar(0, SinOsc.ar(Gate.ar(
    LFNoise1.ar(20), LFPulse.ar(1.333, 0.5))
    .linexp(-1, 1, 200, 20000)))
})
```

Figure 4: The original SynthDef for a simple frequency-modulated sine oscillator. While `Out` and `LinExp` are stateless, `SinOsc`, `Gate`, `LFNoise1`, and `LFPulse` carry state, none of which is resettable in this setup.

```
SynthDef(\snap, {
  arg i_state0 = 0, i_state1, i_state2,
    i_state3 = 0.5, i_state4 = 0, i_state5 = 1,
    i_seed = 1, t_memo;
  var o_state0, o_state1, o_state2, o_state3,
    o_state4, o_state5, noise, pulse, gate, freq,
    sine, noiseMem, pulseMem, oscMem, gateMem;

  noiseMem = { arg freq, i_phase, i_rnd1, i_rnd2;
    var phas, tri, im0, tr1, tr2, rnd1, rnd2, rng;
    phas = LFSaw.ar(freq * 0.5, i_phase);
    tri  = phas.abs * 2 - 1;
    im0  = Impulse.ar(0);
    tr1  = (phas.neg > 0).max(im0);
    tr2  = (phas     > 0).max(im0);
    rng  = Dwhite(-1, 1, inf);
    rnd1 = Demand.ar(tr1, 0, Dseq([i_rnd1, rng]));
    rnd2 = Demand.ar(tr2, 0, Dseq([i_rnd2, rng]));
    [LinXFade2.ar(rnd1, rnd2, tri),
      phas, rnd1, rnd2]
  };

  pulseMem = { arg freq, i_phase, width = 0.5;
    var saw = LFSaw.ar(freq.neg, i_phase);
    [saw > width.linlin(0, 1, 1, -1), saw]
  };

  oscMem = { arg freq, i_phase;
    var phase1 = LFSaw.ar(freq, i_phase);
    var phase2 = (phase1 + 1) * pi;
    [SinOsc.ar(0, phase2), phase1]
  };

  gateMem = { arg in, gate, i_init;
    var out = Gate.ar(in, gate) +
      (i_init * (1 - SetResetFF.ar(gate)));
    [out, out]
  };

  RandSeed.ir(1, i_seed);
  RandSeed.kr(t_memo, t_memo);

  #noise, o_state0, o_state1, o_state2 =
    noiseMem.(20, i_state0, i_state1, i_state2);
  #pulse, o_state3 = pulseMem.(1.333, i_state3);
  #gate, o_state4 =
    gateMem.(noise, pulse, i_state4);
  freq = gate.linexp(-1, 1, 200, 20000);
  #sine, o_state5 = oscMem.(freq, i_state5);

  SendReply.kr(t_memo, "memo",
    [o_state0, o_state1, o_state2, o_state3,
     o_state4, o_state5]);
  Out.ar(0, sine)
})
```

Figure 5: The non-resettable stateful UGens have been replaced by functions `noiseMem`, `pulseMem`, `oscMem`, and `gateMem`. The functions take additional arguments for initial state, and return the equivalent signal along with current state for snapshot storage.

The FreeVerb UGen[7] is an example: If we cut into a Synth which uses this reverberation algorithm, employing a snapshot marker as described in the previous section, the newly started FreeVerb begins with its internal buffers being empty, producing thus a signal that is different from what had been output in the uncut original synth. However, as time passes, due to the decay of the recursive reverberation filters and delays, the difference in output will become smaller and smaller, until it approaches the floating-point noise floor. Figure 6 shows the results for the FreeVerb UGen where the input signal is white noise. A snapshot marker cuts the Synths at 5 seconds, and the discrepancy towards the uncut signal is plotted. The reverb output has a constant RMS of −14.6 dBFS, and the error signal drops 60 dB below this level at approximately 2.4 seconds. The slope becomes shallow at around 6.1 seconds, reaching −146 dBFS, and interestingly, after around 9.3 seconds, the error completely disappears.

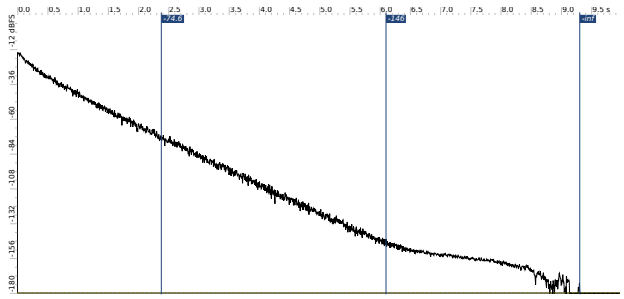


Figure 6: Discrepancy (RMS) over time between cut (reproduced) and uncut (original) reverberation signal. Levels are given in decibels full scale. The original signal has an average level of -14.6 dBFS.

Note 1: Convergence composes—feeding a convergent UGen into a stateless UGen makes the stateless UGen attain convergent behavior; feeding it into another stateful, convergent UGen will merely slow down the overall convergence speed. However, it cannot be fed into a non-convergent UGen (e.g. any of the chaotic oscillators), as the latter's output may become completely different.

Note 2: The undamped oscillators—SinOsc, Saw, Pulse, etc.—are not convergent, and therefore storing their state is still indispensable.

5.4. Integration and Interpolation

This second note is the reason why we conducted another experiment. The overall aim is to spare the snapshots and instead allow random access at any point in time. Again the UGens are enhanced (by rewriting the graph or by modifying their source code) to be able to reset state, but this time no extra state outputs are added and recorded in a snapshot. Instead the required state in a reset action is obtained by ways of integration and interpolation. The example used for demonstration is shown in figure 7. The sound is a slowly frequency-modulated pulse oscillator fed through a resonant lowpass filter RLPF whose center frequency is modulated by another sine oscillator. Finally a comb filter CombL reverberates the result.

```
SynthDef(\gliss, {
  var in = LFPulse.ar(SinOsc.kr(0.05)
    .madd(80, 160), 0, 0.4) * 0.05;
  var freq = SinOsc.kr(0.6).madd(3600, 4000);
  var filt = RLPF.ar(in, freq, 0.2);
  Out.ar(0, CombL.ar(filt, 0.3, 0.2, 2))
})
```

Figure 7: SynthDef used for the interpolation experiment

The comb filter and the lowpass filter are convergent and will be excluded from manipulation. It remains to predict the state of the filter's in and freq arguments. The latter is simpler, a sine oscillator at constant frequency, being scaled into the value range of 400 to 7600. Given i_{off}, the cutting time in seconds relative to the original Synth's start, this oscillator should theoretically have an initial phase of $2\pi f \cdot i_{\text{off}}$, hence the line could be rewritten:

```
var freq = SinOsc.kr(0.6, 1.2pi * i_off).madd(...)
```

This initial phase is theoretical as the implementation of the UGen performs phase increments in discrete steps. As a result, the error signal increases with the cutting point i_{off}: The error's RMS for $i_{\text{off}} = 10\,\text{s}$ is $-89.1\,\text{dB}$. Performing the cut another ten seconds later, the noise floor shrinks to $-83.1\,\text{dB}$.

The situation for the frequency input of the pulse oscillator is more difficult. Looking into the implementation, this input is integrated over time and taken modulus 1 to obtain the instantaneous phase. A scaled and offset sine oscillation must thus be integrated which yields

$$\int (a \cdot sin(\omega t) + b)\mathrm{d}t = bx - \frac{a \cdot cos(\omega t)}{\omega}$$

Unfortunately, even when rounding the frequency of the sine to match internal workings of SinOsc, it is not possible to reconstruct the initial phase of the pulse oscillator, most likely due to the fact that the discrete (sample) steps of the actual integration are not taken into account. Another problem arises when UGens are further nested, as it may become impossible to find the integral of a given input signal.

We still think that this integration and interpolation approach may be useful if restricted to the low frequency components in the SynthDef, such as LFOs, ramps, envelope generators, and so forth, because perceptually it may be just relevant to restore the slowly moving envelopes of the sound processes, while ignoring the phases of the oscillators in the audible range. Also the difficulty of performing the interpolation automatically may just be turned into the feature of this approach: To ask the composer explicitly to provide an interpolated SynthDef so that the decisions regarding the perceptual or conceptual relevance for each UGen's interpolation are left to human decision.

[7] This is the monophonic version of FreeVerb2. The results apply equally to FreeVerb2 and any FIR or damped IIR filter.

6. FROM MECHANICAL TO DIFFERENTIAL REPRODUCTION

Some effort has been spent on different technical ways to realize random access in a sound synthesis system. It has been shown that there may be different degrees of freedom regarding the choice as to whether a signal coming from the outside of the system is to be recorded or not, whether a total-recall is necessary or it is sufficient to achieve convergent behavior or even just approximations to low frequency phases.

We have almost forgotten about the reasoning of the recursive model of composition—random access denoting the *connectivity* of each iteration's output to the further process. The initial point in the technical investigation was the possibility of reproduction of the system's state from a previous iteration cycle, to facilitate the analysis and re-synthesis of the structure, what has been summarized in the triplet agglomeration / division / transformation. But what kind of reproduction? Benjamin's mechanical reproduction has long become *technology*, a «stable subroutine» as Hans-Jörg Rheinberger would call it [6, p. 80]. Applying Rheinberger's concept of the experimental system to art production can, in our opinion, be very fertile, and it is thus tried here. Such a system is said to be productive when it is driven by *differential* reproduction.

Mechanical reproduction talks of the original and its effigy—and we have used this terminology in section 5 when speaking about the original performance and the (faithfully) reproduced cut—, an obsolete notion which Derrida contrasts with his idea of signification as a chain of traces:

> «The trace ... means that the origin ... was never constituted except reciprocally by a non-origin, the trace, which thus becomes the origin of the origin. From then on, to wrench the concept of the trace from the classical scheme, which would derive it from a presence or from an originary nontrace and which would make of it an empirical mark, one must indeed speak of an originary trace or arche-trace. Yet we know that that concept destroys its name and that, if all begins with the trace, there is above all no originary trace.» [2, p. 61]

It is in this manner that reproduction for Rheinberger denotes the «material process of generating, transmitting, accumulating, and changing information», a process of repetition which aims at variation, rather than a process of replication which aims at identity. Because the experimental system (whether in science or art) aims at generating questions which have hitherto been unknown[8], it must look for differences, and this process of differential production must be sufficiently open to allow unforeseen things to happen, but at the same time provide enough cohesion to allow its comparison: «To arrive at new results, the system must be destabilized–but without a previously stabilized system there will be no "results."» [6, p. 80]

We must thus, if random access is to be made productive for new forms of electroacoustic composition, turn it into a facilitator of differential production. Is is an open question whether as an implementor of computer music software one is able to change from the role of the *engineer* into the role of the *bricoleur*, in order to perform the inverse step of destabilizing a *technology* so that it turns into a *research device*.[9]

In Schaeffer's defense it must be said that his insistence on a thoroughly planned process of composition, sequentially carried out, is also explained by the restrictions of the analog studio, in which every experiment "wastes" material and time. In the contemporary computer system in which information can be multiplied almost without limits these constraints do not hold anymore, and it is precisely because of this liberty that the infinite recursion can be realistically considered. The techniques outlined in section 5 may be incorporated into a cohesive system in which the composer cautiously oscillates between replicative and meandering cuts, in which he or she can exploit both aspects of writing – mnemotechnique and the power of forgetting.

7. REFERENCES

[1] W. Benjamin, *Das Kunstwerk im Zeitalter seiner technischen Reproduzierbarkeit*. Frankfurt/Main: Suhrkamp, 1963 (1936).

[2] J. Derrida, *Of grammatology*. Baltimore: Johns Hopkins University Press, 1997 (1967).

[3] A. Di Scipio, "'Sound is the interface': from interactive to ecosystemic signal processing," *Organised Sound*, vol. 8, no. 3, pp. 269–277, 2003.

[4] P. Manning, "The Influence of Recording Technologies on the Early Development of Electroacoustic Music," *Leonardo Music Journal*, pp. 5–10, 2003.

[5] J. A. Moorer, "Hard-disk recording and editing of digital audio," in *Audio Engineering Society Convention 89*, 9 1990.

[6] H.-J. Rheinberger, *Toward a history of epistemic things: Synthesizing proteins in the test tube*. Palo Alto: Stanford University Press, 1997.

[7] P. Schaeffer, *Traité des objets musicaux: essai interdisciplines*. Paris: Editions du Seuil, 1977 (1966).

[8] K. Schöning, "The Contours of Acoustic Art," *Theatre Journal*, vol. 43, no. 3, pp. 307–324, 1991.

[8] A situation very similar to what Manning describes: «Matching expectation to reality in the search for genuinely new musical horizons presupposes the ability to hypothesize the characteristics of the unknown, and thus many early attempts to explore recording technology as a creative tool were motivated by simple curiosity rather than clarity of purpose.» [4]—if the experiment had no vagueness, was completely defined, there would be no sense in carrying it out, the outcome would be redundant.

[9] cf. [6, p. 81]