14. Concrète Procedures

Splicing blocks and razor blades; no longer useful.

- ***Concrète***

To illustrate the impact recorded sound has had on music and communications, take a quick inventory of the music you listen do daily; live vs. recorded. Is it even close to 50/50? Would we have to extend the challenge to a week before we could include a single incidence of live performance? I've encountered students who have *never* attended a live concert of any kind, yet their collections contain thousands of works. Early recordings were used as parlor entertainment. Guests were shocked to hear, for example, horses in a room.

I love classic science fiction, in particular tech spotting; taking note of what they got right, or how far off they were in other ways. (They were much too pessimistic.) No flying cars from the Jetsons, but video phones? There is one sitting next to me on the couch. Two that still seem to elude us are teleportation and time travel. Yet one could argue that recordings do both. They are time and space machines.

The tape deck's intended contribution to the music world, reproduction, was quickly hi-jacked by the likes of Ussachevesky, Sheaffer, Leuning, McCartney, and the minimalist school. These pioneers were more interested in extending the capabilities of natural instruments—for example exploring the sound of a piano if the keyboard were several octaves lower, or higher—but were easily distracted by entirely novel treatment; sounds that had literally never been heard before, and were unique to the medium. The Beatles continued to explore Concrète treatment on nearly every album after Revolver, the most notable, striking, and ignored being *Revolution 9*. It would fit in nicely at most electro-acoustic concerts. Once dismissed as hair-brained, this technique is now common in Rock, a staple in Rap, Techno, Industrial, and similar styles. It is the foundation for sampling synthesis and sample libraries. Examples of Concrete treatment in popular media include *Strawberry Fields* and *For the Benefit of Mr. Kite* (Beatles), *Another One Bites the Dust* (Queen), pretty much everything Daft Punk does, but I like *Face to Face*, *Vox Humana* (Kenny Loggins), and the entire *Year Zero* album, NIN. I can't pick just one. Maybe *God Given*.

There used to be a pronounced schism between purely electronic and Concrète composers. The two have since merged. The Concrète camp were drawn to the built in complexity due to the rich source material. Concrète and additive synthesis, (covered later—which can also be very rich and surreal) are my favorite electronic techniques.

✓ Most think of Concrète as manipulating recordings of real sounds in artistic ways, any recorded sound is "set in concrete." That is in fact the inspiration for the term. Even a recording of a classical work with full orchestra is like a statue; set in stone.

This definition reminds us that live music has a dimension recorded pieces do not; the potential for growth and change. A written work that has to be realized is not the performance, but the potential for a performance, or a failed performance, or a brilliant performance, or in the case of aleatory and improvisation, something completely new, never heard before. I believe too many people equate a recording with a performance. In a sense it has tainted our expectations: we attend a concert expecting what we heard on the CD. The Beatles were one of the first to

realize they could subvert the standard practice of releasing recordings just to bring people to concerts. They saw that the recordings could stand on their own merit. They became the medium, and the principal source of income. As digital rights management fails there could be a return to the previous model; give away the recording to attract audiences to live performance, which holds the promise of fresh material.

Concrète introduced us to sounds that had never been heard before. Indeed all recordings, even those not intended for concrète treatment, have exposed us to rare and exotic sounds, previously only heard by natives of a particular environment. Exploring sounds unique to recorded reproduction is an important goal of this chapter. My first exposure to an EVI (electronic valve instrument) was a concerto that could have been, and probably should have been played on trumpet. I was disappointed that the performer never changed the timbre, never leapt six octaves, never bent the pitches, never fired off notes at the speed of sound. (Well, technically, yes.) Many electronic performances since have left me wondering; what is the point? I see the stage filled with mics, computers, wires, and speakers, but hear music that could have been performed before the electronic revolution.

The power of Tchaikovsky, Bach, Vivaldi, and Hendrix lies in the exploitation of their chosen medium; the power of the piano, carefully crafted vocal lines, flashy violin arpeggios, controlled harmonic feedback. We should judge electronic music the same way; does it exemplify its unique power? For Concrète, we have to ask; what can digital manipulation do that can't be done on any other instrument?

An additional appeal to concrète material is the inherent contextual baggage. If a sound is drawn from the human experience (and how can it not?), then the audience will bring that association to the work. You can dodge association through obfuscation or foreign languages, or you can embrace it as part of the work; juxtaposing, for example, a speech with iconic religious inflection over cartoon sound effects. Auto-tuning the news is an excellent example; it is precisely the intended hubris and arrogant context that makes vocalized parodies so effective.

✓ Concrète is capable of four treatments (which we now take for granted) that cannot be performed on any other instrument; precise loops, pitch and time shift, reversal, and abrupt edits.

- ***Looping***

… is rampant in popular media, evidenced by the Apple Loops library. Early loops were executed on magnetic tape decks, splicing a section of tape to itself then threading it around the room using mic stands for support. *Tomorrow Never Knows*, Beatles, is an early example of loops. (These were created by Paul on a newly purchased open reel tape deck. As far as I have read Paul worked this out on his own, without exposure to any of the art music experiments happening at Columbia. From Geoff Emerick's book: "Every tape machine in every studio was commandeered and every available EMI employee was given the task of holding a pencil or drinking glass to give the loops the proper tensioning. In many instances, this meant they had to be standing out in the hallway, looking quite sheepish. … Add in the fact that all of the technical staff were required to wear white lab coats, and the whole thing became totally surreal.") Looping appeals to me in two ways; the precision of repetition, and the rhythmic patterns possible with mathematically related loops. Our brains are finely tuned to recognize pattern. We recognize in electronic loops minute details. No other musical instrument is capable of such accurate repetition.

- ***Pitch and Time Shift***

… were, until only recently, inextricably linked. Change the speed of playback and the pitched changed. Today's wizardry can change one and not the other. (This is analogous to increasing the length of a book without changing the content.) Taking this process too far results in aliasing (in itself an interesting effect). Speed change with spoken samples borders on cliché, but other sounds—percussion, ambient noise, instruments, sounds from nature, still keep my attention. The Beatles also experimented with tape speeds. The solos on *Misery* were recorded with the tape at half speed, partly out of necessity (George was having difficulty with them), partly for effect. This is not readily apparent, but blatantly obvious when the track is played at half speed. But the backing track to *Rain* was intentionally recorded at a faster tape speed, then slowed down for effect, not out of convenience.

- ***Reversal***

… also trickled from the experimental academic studios to popular recordings. *Rain*, Beatles (and nearly everything else on *Revolver* because of *Rain*) uses reversal. They hit on the idea because John mis-threaded a rough mix on his home machine. Other examples include *Another One Bites the Dust*, Queen, *2000 Light Years From Home*, Rolling Stones and the entire left channel guitar solo in *Castles Made of Sand*, Hendrix.

- ***Abrupt Edits***

… is another technique that is unique to recording technology. No natural sound or instrument can change it's timbre, subtly or radically, with millisecond accuracy. In *Yellow Submarine*, Beatles, Emerick snipped up the brass band section into small segments, threw them in the air, then spliced them back together again. His intention was to avoid copyright infringement, which worked, but also hit on a unique treatment. Daft Punk has built a career around sampling, abrupt edits and looping.

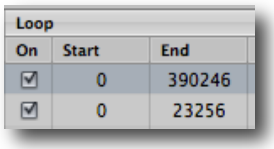
Of course there are, and I encourage you to explore, excellent examples of concrète from the academic world. I use popular examples to illustrate their viability. These techniques were dismissed and even derided among academics. They were obviously wrong to do so. The 1998 documentary *Modulations* chronicles the link between experimental academia and the electronic styles that evolved in clubs and raves.

There are two tools in Logic that give quick concrète results. They are designed as sampling synthesizers. Like recording technology in general, they are intended for exact reproduction of natural instruments, but can be repurposed for concrète. The first is the …

- ***EXS24***

You can import samples and play them back on the keyboard for precise control of a large collection of loops.

✓ When you first launch the EXS24 there is no instrument assigned. (If there is an instrument, choose no instrument.) Pressing the edit button when no instrument is loaded will create a new instrument and open the editor. You can drag and drop samples onto this field. Use the bars just above the keyboard to specify which keys trigger a given sample. Set the **Key** field to where you want normal speed played back. In the local **View** menu choose **Zone**: **Loop** to turn looping on and to specify the start and end point (in frames) of each loop. Now you can play these samples on the keyboard and record them as MIDI events. They will play at different speeds for each key. The most interesting application, for me, is precisely coordinated loops. Try intervals such as a fifth, fourth, or octave. As discussed in Chapter 8, the samples will go in and out of sync at those ratios.



- ***UltraBeat***

… has similar functionality with drag and drop simplicity. The advantage over the EXS24 is the built in looping. It can manage 24 separate samples, each with 32 step sequences, and virtually unlimited sets of patterns. It is a Concrète composer's dream.

In a Logic session add an instrument track and insert UltraBeat in the **I/O** input menu. UltraBeat has three synthesizers to generate its drum loop sounds. We will work with the sampler at the bottom (see Figure 14.1, first image). In the **Media**/**Library** choose the **Hip Hop 90s Kit** preset or the **Drag and Drop Samples** preset. (I choose Hip Hop kit because most of the sounds are real samples that can be replaced, and it is pre-loaded with sequences and loops. Feel free to explore, others, but find something that uses primarily samples. Use Drag and Drop to start from scratch.) Press the power button, then the play button. Turn on **pattern mode**, set to **toggle** or **one-shot trigger**, then to the bottom left click on the pull down menu next to **pattern** to select one of the 24 sequences assigned to the lower octaves of a MIDI keyboard (C -1 through B 0).

The far left of the UltraBeat window is a keyboard (see Figure 14.1), each key labeled with the name of the loaded sample, a mute, solo, and pan control. While playing back, load a new concrète sample by dragging it to the window displaying a wave form. Click the red arrow to reverse, click the piano key to test. Continue to experiment with different sequences and replacing existing drum sounds with your concrète material. This alone, if improvised, would be an interesting concert performance.

The sequencer (Figure 14.1, second image) has 32 steps. Each can be shortened by dragging the bar just below the step numbers. Clicking on a number adds a gate to that step. A gate is a trigger with duration. The sound continues to play until the gate is released. A 4 second sample needs at least the same length of gate (though it can certainly be shorter). Once you have loaded and tested a Concrète sample, click on the numbers to add several gates. Click and drag across several trigger positions to create longer or shorter gates. Repeat for each new key and each new sample, building a collection of looped Concrète sequences.

Fig. 14-1 UltraBeat

To the right of the sample window is a complex dial (labeled **OSC2**) that allows you to modify the pitch, amplitude, and velocity. Experiment with the reverse switch as well as the pitch of the playback. Feel free to load the same sample into several key locations for different treatments (reverse, pitch, envelopes).

In the lower left hand corner of the UltraBeat window click **full view**. This changes the display from synthesizer to a graph showing all sequences. Control click any track to reveal a handy set of editing features, including copy, paste, clear, randomized creation and replacement. No one takes me seriously, but seriously, you could stop coming to class and do an entire project with this plug-in alone.

There is a mysterious grid icon just to the left of the MIDI key assignment pull down menu. Drag the small box to the arrange window to duplicate the pattern as a region.

For a generative dimension, name all your samples 01.aif, 02.aif, etc., have a colleague collect new samples (perhaps before the concert from the audience) and also name them 01, 02, etc. Drag these new samples to your sample folder, relaunch the session and it will read and play the replacements.

- ***Concrète using SC***

… is a little more involved, but the results, especially the granular treatment in the next chapter, are worth it. Since this is the first example in this chapter (and perhaps coming to this example afresh), remember to launch the server before running any examples.

14.1:: **Loading an Audio Buffer**

```
// Boot the server first

b = Buffer.loadDialog; // Switch to sclang to open file

[b.bufnum, b.numChannels, b.path, b.numFrames] // confirm that it opened correctly

b.play; // test playing it back
```

::

Buffer.loadDialog brings up an OS X dialog (in the interpreter) which you will use to find an audio example. I suggest you use a mono, 16 bit, 44.1k aif file longer than 10 seconds. It

will play back only in the left channel. Stereo comes later. And while I'm suggesting, how about a two second clip from any cut out of Tom Waits' *Bone Machine* or a segment from Cage's brief interview on the activity of sound. The sound is stored at the variable *b* using the equals sign. The next line of code can be used to make sure the buffer loaded correctly. It displays the buffer number, the number of channels in the file, the path, and the number of frames. A frame is the collection of samples for all channels. So a 5 channel audio file will have five samples per frame, but that single frame represents a single sample of time. A one second stereo file will have 44100 frames, but 88200 samples.

For now we will loop using *PlayBuf*. The help file for *PlayBuf* shows the third argument is the playback speed. Negative numbers will play backward.

14.2:: **Pitch and Direction**

```
// change the third argument to values between -2.0 and 2.0
{PlayBuf.ar(1, b, 1)}.play;

// Be patient. You may have to listen a while.
{PlayBuf.ar(1, b, LFNoise0.kr(1, 3))}.play;
```

::

The fourth and fifth argument of *PlayBuf* can be used to generate loops. The fourth is a trigger that resets playback to the start position. The fifth argument is the position, in frames, where the loop returns to. Example 14.3 is a loop browser. A *MouseX* returns values based on mouse position and is used to supply the playback position of each loop. It uses a minimum value of 0 and a max value of the total number of frames in the sample. Move the mouse to the left for the beginning of the sample, right for the end.

The trigger argument is the frequency, in seconds, of the trigger to reset the loop. This essentially becomes the length (in time) of the loop. The example below uses *Impulse*. The *LFNoise0* from the previous example has been removed so you can experiment with the playback start and trigger frequency without speed being changed. Replace it if you'd like. Try values between 0.1 and 4 for the trigger frequency.

14.3:: **Loop, Playback Position, Triggers**

```
// The numFrames message returns the number of frames.
// It's used below for the MouseX min, max value.
b.numFrames;

// Impulse sends a single pulse as a trigger.
// You will hear a tick
{Impulse.ar(1)}.play

//
{PlayBuf.ar(1, b, 1, Impulse.kr(1), MouseX.kr(0, b.numFrames))}.play;
```

::

- ***Envelopes***

You might notice that some start positions produce a pop, or click at each loop. This is the same effect caused by poor edits; an abrupt change from positive to negative values. When fishing around for a start position, it's impossible to know where 0 crossings are, so the solution is to apply an envelope over each loop. Envelopes are essential components for instrument design. They describe the change of the sound over the duration of a single instance of the sound (usually amplitude, but also timbre and sometimes pitch). A piano, for example, will have a fast attack and a long decay. A violin can create complex envelopes, including a long attack and a very fast decay. Envelopes are set in motion with a trigger or gate. Pressing a piano key is a real world example of a trigger. It is a single event that sets the sound in motion. You have no control once the key is pressed. A gate is similar to an organ key. It will continue to produce sound until the key is released, at which point it will decay.

Example 14.4 illustrates envelopes. (The plot windows will appear in the interpreter. You will have to switch back and forth.) The fist shows a plot of noise without an envelope. The second shows a millisecond of plotted noise to better illustrate the probability that the beginning and ends of the audio are not at 0 crossings. If this example were looped, you would hear a tick. The next two multiply the white noise source by an envelope generator. The first plots, so a trigger is unnecessary. The last uses a trigger (*Impulse*) at one time per second.

14.4:: **Envelopes**

```
// Steady state noise.

{WhiteNoise.ar}.plot(1)

{WhiteNoise.ar}.plot(0.001)

// Same signal multiplied by an envelope with 0.1 second attack
// 0.6 second sustain, and 0.1 second decay.

{WhiteNoise.ar*EnvGen.kr(Env.linen(0.1, 0.6, 0.1))}.plot(1)

{WhiteNoise.ar*EnvGen.kr(Env.linen(0.1, 0.6, 0.1), Impulse.kr(1) )}.play
```

::

- ***Variables, Frequency and Duration***

The single letters *a* through *z* are defined by SC as global variables during startup. Some of these are reserved by the program. The variable *s* is always used for the server. Example 13.3 used the variables *a*, *b*, and *c*. Variables store items that will be used several times in a patch or several patches. In these examples, the variable *b* retains the buffer for each new execution of a patch. Otherwise, it would require the load-buffer dialog to choose a file each time.

Local variables are confined to the function where they are declared. They are used to link components of a patch together, and to organize code by breaking it down into smaller statements. In the example below, each section is stored in the variable *mix*. Think of it as a bowl into which you mix and combine ingredients; *mix = eggs and flour; mix = mix + butter;*. They are declared in the top line as *var myVar1, myVar2, myVar3*;

Periodic events have both a frequency and duration. They don't have to correspond. For example, a trigger with a frequency of 2 (times per second) will have a 0.5 second duration. Each event between those triggers can be independent of the trigger duration. They could have a *duration* of 0.5 seconds, but also 4 or 0.1 seconds. If 0.1, then the sound will die away between each event. If 4 seconds the events would overlap.

Either way, there is a connection, and it is useful to set durations in relation to frequency. As discussed in an earlier chapter, frequency and duration are reciprocal. Invert each to convert one to the other; d = 1/f, and f = 1/d. A 10 second duration has a frequency of 1/10 (think of it as one in ten seconds). A 10 Hz frequency will result in a duration (between each cycle) of $1/10^{th}$ of a second. To link the two, set the duration to some percentage of the frequency. For example, d = 1/f * 0.5. If the frequency changes the duration will change too, and will always be 50% of the time between each event.

Example 14.5 introduces a handy convention. The first patch is everything between the first, the code that actually produces the sound, is everything from *{ // patch…* and *}.play;*. But this entire patch is enclosed in parentheses. These parentheses have no real impact within the patch, but they define a region. You can double click on the top parentheses to select the entire patch, or use ⌘-return to select and execute.

Try inserting the *CombN* used in the rising sines. Insert a line before the final *mix;* with something like *CombN(mix, etc.)*.

14.5:: **Variables**

```
(
{ // patch without variables
    PlayBuf.ar(1, b, 1, Impulse.kr(1), MouseX.kr(0, b.numFrames))*
        EnvGen.kr(Env.linen(0.1, 0.8, 0.1), Impulse.kr(1));
}.play;
)

(
{ // same patch with shorter duration (higher frequency) Impulse
    PlayBuf.ar(1, b, 1, Impulse.kr(0.5), MouseX.kr(0, b.numFrames))*EnvGen.kr(Env.linen(0.05, 0.4,
0.05), Impulse.kr(0.5));
}.play;
)


(
{ // Variables for clarification
    var mix, loop, duration;
    duration = 0.5; // duration in seconds of the loop
    loop = Impulse.kr(1/duration); // duration converted to frequency
    mix = PlayBuf.ar(1, b, 1, loop, MouseX.kr(0, b.numFrames));
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, 1, 1, duration);
    mix;
}.play;
)
```

::

The last line of code is simply *mix*. This is called the *return*, or the final results of the function that are passed to the message play.

Aside from spreading out this patch for clarity, the variables solve a critical problem. The envelope (in this case at least) needs to match the duration of each loop. In the second patch, not only did I have to change both *Impulse* frequencies from 1 to 0.5, but I had to calculate the attack, sustain, and decay of the envelope given a duration of 0.5. All that is done automatically in the last example. *Mix*, which is a single loop, is multiplied by the envelope, which in turn is scaled to the duration with the fifth argument of *EnvGen*; *timeScale*. The result is an envelope that changes for any given duration, and changes in the other channels (covered soon). Imagine doing 8 channels, each with a different duration, each with different attack, sustain, and release times. With a variable, when the value for duration is changed, the attack, sustain, and release is set to 10%, 80%, and 10% of the duration, whether $1/10^{th}$ of a second or 5 seconds. Variables are also (albeit slightly) more efficient since this patch uses a single Impulse in two places rather than two Impulse UGens.

- ***User Defined Arguments***

The problem with variables is they cannot be changed once sent to the server. Like existing Ugens, you can create your own arguments, which are variables that have a link to the outside world and can be changed once a synth is running. They are declared before variables, directly after the opening brace of a function. One additional component, the synth must be loaded into a variable so that we can refer to it when changing the parameter. (But see SynthDefs below.) The syntax is *a.set(\argName, value)*.

14.6:: **Arguments**

```
// b = Buffer.loadDialog;

(
a = { // Arguments for control
    arg speed = 1, position = 0.5;
    var mix, loop, duration = 0.5;
    loop = Impulse.kr(1/duration); // duration converted to frequency
    mix = PlayBuf.ar(1, b, speed, loop, position * b.numFrames);
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, 1, 1, duration);
    mix;
}.play;
)


a.set(\speed, -0.5);
a.set(\speed, 1.25);
a.set(\position, 0.2); // must be between 0 and 1
a.set(\position, 0.75);

a.set(\speed, rrand(-1.2, 1.2)); // execute these several times for different values
a.set(\position, rrand(0, 1.0));
```

::

- ***Multi-Channel Expansion***

✓ I love this trick: if any UGen argument, anywhere in a patch, is an array (e.g., [1, 3]), the entire patch is duplicated using the first value for the left channel, the second for the right. (There are some exceptions, discussed in later chapters.) If arrays are used for other arguments, they are likewise split between the two channels. This is an extremely efficient way to expand a patch into 2, 4, 8, heck, 16 channels.

Steve Reich's *Come Out* illustrates both looping and phase shift, which became a staple of the minimalist style. The patch above is ready for similar treatment. We only have to change one value—the *Impulse* trigger—replacing the single 0.5 to an array of [1, 61/60]. This makes the duration of the left channel 1 second, the right 61/60 seconds, or slightly longer. Stop for a second and imagine what the resulting loops will do.

Both channels will start in phase, slowly shift out of phase, then back over sixty seconds. In order to focus better on the phase shift I've replaced the *MouseX.kr* with a *rrand*, which picks a value between 0 and total number of frames. Change it back if you'd like. Also replace it with *Line.kr(0, b.numFrames, 60)* and it will not only move slowly out of phase, but also gradually advance across the entire buffer.

Try running it several times.

14.7:: **60 Second Phase**

```
(
{
    var mix, loop, duration;
    duration = [1, 61/60]; // duration in seconds of the loop
    loop = Impulse.kr(1/duration); // duration converted to frequency
    mix = PlayBuf.ar(1, b, 1, loop, rrand(0, b.numFrames));
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, timeScale: duration);
    mix;
}.play;
)
```

::

You will know when the loop as come full cycle because it will sound like a mono signal; both left and right channels will be the same.

Steve Reich slowed down one of the channels by pressing his thumb against the reel of tape. In SC, it's just as easy, but with much more flexibility and precision. Change the 60/61 to other ratios. May I suggest common interval ratios: 2/1, 3/2, 4/3, 8/5, 15/16, 45/32, etc. After working with this patch for a few days you may bolt up in bed realizing the flaw; a 1 second loop in the left channel and a 3/2 second loop in the right will have different durations, so the smoothing envelopes also should be adjusted. If, for example, the left channel is a *frequency* of 1 and the right has a frequency of 61/60, it's loop will be slightly shorter, or is it longer? and by how much? Well lay back down and rest easy. Since we used a variable it happens automatically. If the duration is an array, say [1, 1.234] then the left envelope will be 1 * the envelope, the right will be 1.234 * the envelope, because we used the *timeScale* argument.

- ***Experiment***

Now we can have some fun. First up are *TRand*, *TIRand* and *TChoose*, which return random values with each trigger. *TIRand* returns integers only, which will be useful for our ratios. *TChoose* selects items from an array.

14.8:: **Generative Choices**

```
(
{
    var mix, loop, duration, rate, gen, start;
    gen = Impulse.kr(1/15); // every 15 seconds
    // choose another speed and direction at each gen trigger
    rate = TRand.kr(-2.0, 2.0, gen); // Choose values between -2 and 2
    start = TIRand.kr(0, b.numFrames, gen); // start location
    duration = [1, 15/16]; // match the gen trigger; could I use a variable?
    loop = Impulse.kr(1/duration); // duration converted to frequency
    mix = PlayBuf.ar(1, b, rate, loop, start);
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, 1, 1, duration);
    mix;
}.play;
)

(
{ // Be patient; change happens gradually.
    var mix, loop, duration, rate, gen, start, baseloop, buffer;
    gen = Impulse.kr(1/20); // every 20 seconds
    baseloop = TRand.kr(0.5, 3.0, gen); // select a base loop time
    // choose another speed and direction at each gen trigger
    rate = TRand.kr(-2.0, 2.0, gen);
    start = TIRand.kr(0, b.numFrames, gen);
    duration = [baseloop, baseloop * TChoose.kr(gen, [3/2, 4/3, 5/4, 6/5])];
    loop = Impulse.kr(1/duration); // duration converted to frequency
    mix = PlayBuf.ar(1, b, rate, loop, start);
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, 1, 1, duration);
    mix;
}.play;
)

// Load a new buffer on the fly
b = Buffer.loadDialog(bufnum: b.bufnum);

// This will switch you over to the sclang interpreter.
// To continue editing, or stop the sound, switch back to SuperCollider
```

::

The second differs slightly. The generative choice is lengthened to once every 20 seconds to allow each pattern to play through, but each pattern is shorter, between 1 and 4 seconds. The duration first takes a base loop time, chosen with a TRand, then the left channel uses that base

loop while the right uses the base loop multiplied by one of a collection of ratios, which is chosen by *TChoose* at each gen trigger.

Finally, we wrap up the patch by loading a new buffer on the fly.

- ***Exercises***

14:1 Modify any of the patches above to include arguments rather than variables so that you can change parameters on the fly.

14:2 In the studio, record samples of instruments or items to use in the experiments in the next few chapters. Save the files as 01.aif, 02.aif, etc., so that you can easily swap with other students.

14:3 With your prepared Concrète files handy, locate an existing preset in UltraBeat with built in loops. Start the loop and drag and drop your samples, gradually turning the drum beat into a sample beat.

14:4 Record yourself saying "come out to show 'dem." Import the sample into an empty EXS24 instrument. First, try to reproduce Reich's seminal work with a gradual phase shift. Continue with a study of phased patterns using intervals (octave, fifth, fourth, third, second) and no more than two notes at a time.

14:5 Finish the comments in the following patch, documenting what each line does and how it modifies the patch.

```
(
{
    // This line ...
    var mix, loop, duration, rate, gen, start, baseloop, buffer;
    // Every 20 seconds gen...
    gen = Impulse.kr(1/20);
    // baseloop is...
    baseloop = TRand.kr(0.5, 3.0, gen);
    // rate changes... TRand chooses values...
    rate = TRand.kr(-2.0, 2.0, gen);
    // start resets the loop... TIRand returns...
    start = TIRand.kr(0, b.numFrames, gen);
    // duration is the duration of each...
    duration = [baseloop, baseloop * TChoose.kr(gen, [3/2, 4/3, 5/4, 6/5])];
    // loop defines...
    loop = Impulse.kr(1/duration);
    // mix is used to combine...
    mix = PlayBuf.ar(1, b, rate, loop, start);
    // The envelope generator smooths...
    // The arguments are loop and duration, which...
    mix = mix * EnvGen.kr(Env.linen(0.1, 0.8, 0.1), loop, 1, 1, duration);
    mix;
}.play;
)
```