4_cmusic

July 28, 2025

INDEX

1 Historical framework

The first experiments in sound synthesis do not belong to the digital world but to the analogue one.

In the 1950s the study of the Cologne Radio (WDR) was in contrast to the GRM in Paris both in: * terms of sound generation and manipulation techniques. * terms of aesthetic positions.

The reference figure in this study was the composer Karlheinz Stockhausen.

At that time his compositional ideas were the opposite of those of Paris: he brought the strict rules of integral serialism into the generation of sound.

Every parameter of sound (pitch, spectrum, intensity, envelope, duration, etc.) was governed by rigid numerical rules.

The electroacoustic instruments available in the Cologne studio (oscillators, filters and ring modulators) and the possibility of recording the resulting sound on magnetic tape were the ideal tools for developing these ideas.

Karlheinz Stockhausen - Elektronische Studie II for tape (1954) - extract

Still in the electroacoustic field, a mix of the technical equipment of the two studios was found in the RAI Musical Phonology Studio in Milan where the composers of reference were Luciano Berio, Bruno Maderna and later Luigi Nono.

Their aesthetic positions were also less rigid than those of their French and German colleagues, mixing together different compositional techniques.

Bruno Maderna - Musica su due dimensioni for flute and tape (1958) - extract

Computer music was born later for reasons obviously linked to the birth and development of digital technologies but it takes its cue from these (and other) electroacoustic experiences.

The first computer used for playing music was the CSIR Mk1 developed in Sydney in the late 1940s and presented at a public concert in 1951 where it performed the piece “Colonel Bogey March”.

Alan Turing was also working on primeval computer music in the summer of 1951 at the Computing Machine Laboratory in Manchester.

He and his collaborators produced three short melodic sequences played by the computer.

In the late 1950s Max Mathews was the most important computer scientist working at Bell Telephone Laboratories in New Jersey.

He worked primarily with mainframe computers made by IBM on several series of their MUSIC X program, a musical programming language that was upgraded over the next twenty years.

An early piece created in these studios was Beat Canon (1960) by J.R.Pierce (director of studio).

In 1963 the human voice was synthesized for the first time in these laboratories (speech synthesis).

Later musicians and engineers alike began to collaborate more on the making of music with computers realizzando lavori più sofisticati.

Aa example an exrept from ‘Lyric Variations’ for Violin and Computer (1965-1968) by J.K.Randall.

In USA starting from the 60s artistic currents began to develop linked to the Studios and cultural environment where they arose.

- West coast → “Experimantal Music Studio MIT MEdia Laboratory” - Boston.

  Founded by Barry Vercoe in 1971, it develops software derived from the Music IV series on IBM 360 computers.

  In this studio Curtis Road created Computer Music Journal the world’s leading computer music magazine.

  Composers such as James Dashow, Miller Puckette, Charles Dodge worked there, whose approach to composition was characterised by a strong structural rigor derived from the ideas developed by Stockhousen in Cologne.

  Three exemples:

  - James Dashow - Conditional Assemblies for computer generated sounds (1980)
  - Charles Dodge - The Waves for voice and electronics (1984)
  - Curtis Road - ‘Granule’ for computer generated sounds (1999)

- California two main poles:

  - Stanford University CCRMA (Karma) → music production.
  - San Diego University (CARU) and Lucas film studios → software and hardware research.

  The CCRMA was founded and directed by John Chowning, the inventor of FM synthesis and a leading figure in computer music.

  A figure of synthesis between technology and music, he addressed in each of his compositions a specific problem linked both to the synthesis of sound and to musical language.

  Through his work, which also took place in important European production centers such as IRCAM in Paris, he created a bridge between the two continents.

  An audio extracs from his work Turenas for computer generated sounds (1972) and a paper about it.

Here we can find a comprehensive list of computer music research and production centers.

2 Randomness

Following or pursuing an random procedure means generating or transforming musical or acoustic material by integrating random choices into the decision-making process.

The concept of indeterminism affirms the denial of any determination of the will by external motives.

In our case a sequence of sonic events must be governed in whole or in part by chance or by a set of probabilities, regardless of the author's experience, taste, knowledge, and personality.

This makes it inevitable to rethink the composer's role in the creative act.

A reflection of this kind plays an important role in the poetics of American composer J. Cage who has often stated that his compositional principle is a departure from unity and a movement toward multiplicity:

"[. . .] to abdicate one's subjectivity, to abandon one's own judgment to move toward the multiplicity that is life itself, of which man is but a small component, in the sea of constant variation [. . .]"

In Western musical tradition three different indeterministic procedures have been theorized and used.

2.1 Chance

The musical or sonic materials of the piece are:

generated through random processes. fixed in symbolic form to allow a performance where the random elements fall within the limits of deterministic musical interpretation.

Two examples:

- The musical games in Europe between the second half of the 18th century and the beginning of the 19th century (C.Ph.E. Bach, F.J. Haydn, W.A. Mozart, etc.).
  - a set of musical bars that follow a harmonic scheme indexed and arranged by the composer in any order.
  - use of a pair of dice to determine the order of performance (recomposition).

    - the game allowed anyone even those with no musical knowledge to create harmonious and original pieces.

img/mozart_tables.png

img/mozart_tables_1.png

- Music of Changes by J.Cage for piano (1951) in which the formal structure, compositional method and choice of musical materials (pitches, dynamics and tempos) are determined by the roll of the dice of the I-Ching.
    - The collection consists of eight 'books' of music.
    - To compile the score Cage used 8x8 grids (two-dimensional matrices) to facilitate correspondence with the 64 hexagrams of the I Ching.
    - A first roll of the dice determines which sound event to write among those included in the sound grid, after which the next two rolls are dedicated to durations and dynamics.

img/Music_O_C.png

2.2 Indeterminacy

It seeks the variable multiplicity of possible performances not chance.

Indeterminacy is the opposite of chance which adopts a method.

Chance relies on the logic of the method.

Indeterminacy relies on the performer's sensitivity.

It excludes the precise determination of certain parameters (pitch, duration, intensity, timbre) to give the performer a wide margin of freedom.

When taken to its extreme limits it ideally coincides with the opposite deterministic extreme: total improvisation or 'libero arbitrio'.

The concept of indeterminacy in music can be developed in two different ways:

1. Free choice of performance mode or form
    - Missa Cuiusvis Toni (1539) by Johannes Ockeghem.
        - the composer did not specify the clefs at the beginning of the staff.

    - it can be performed in all four modes (Dorian, Phrygian, Lydian, Mixolydian).
    - the choice of which clef to use for reading (and therefore the mode) is left to the performers.

    img/Cuiusvis_toni.png

  - Serenata per un satellite (1969) by B. Maderna
    - It can be played by violin, flute (including piccolo), oboe (including oboe d'amore, even musette), clarinet (transposing the part, of course), marimba, harp, guitar, and mandolin (playing what they can), all together, individually, or in groups, improvising, in short, with the written notes.
    - The whole piece should last between 4 and 12 minutes.

    img/maderna_1.png

2. Graphic scores or verbal indications that suggest to the performer how the piece can be performed.
  - December '52 by Earle Brown.

    img/brown.png

  - Intersection 3 by Morton Feldman (1964).

    img/feld.png

  - Intuitive Musik (1968) by K.Stockhausen

    img/intu.png

  - Necessità espressive (Bruno Maderna)

2.3 Alea

These strategies became relatively common among composers of the European avant-garde after World War I.

Acousticist Werner Meyer-Eppler during the Darmstadt summer courses defined it this way:

“A process can be defined as”aleatory" [. . .] if its development is established in general terms but depends on chance in the details".

We explore the possible use of random procedures in SuperCollider using a SynthDef used in the previous chapter and then applying them to the sound synthesis techniques presented in this chapter and in the algorithmic composition procedures presented in the next chapter.

[339]:
```
s.boot;
s.meter;
{s.plotTree}.defer(5);
```

```
-> a Function
```

Let’s define a Buffer and a SynthDef.

[14]:
```
Buffer.freeAll;
b = Buffer.read(s, "/Users/andreavigani/Desktop/GHub/EMC/3_fixed/suoni/bach.
 ↪wav");

SynthDef(\smp, {arg buf=0,pos=0,dur=0.2,amp=0,trsp=0,dir=1,pan=0,t_gate=0,done=2;
                var sig,env;
                    sig = PlayBuf.ar(1, buf,                                //␣
 ↪UGen (oscillator)
                                    BufRateScale.kr(buf) * trsp.midiratio  //␣
 ↪Trasposition
                                                          * dir,            //␣
 ↪Direction
                                    t_gate,                                 //␣
 ↪Trigger
                                            BufSampleRate.kr(buf)* pos);        ␣
 ↪  // Absolute start position
                                     // BufFrames.kr(buf) * pos);              //
 ↪ Relative start position (0 to 1)
                    env = Env.linen(0.01,dur-0.02,0.01);                    //␣
 ↪Trapezoidal envelope
                    env = EnvGen.kr(env, t_gate, doneAction:done);
                    sig = Pan2.ar(sig * env * amp, pan);
                Out.ar(0,sig)
        }).add;
```

```
-> a SynthDef
```

We can obtain random numbers in several ways, here the main objects.

- boolean → 0.5.coin (0.0 = false, 1.0 = true, 0.x = probability). For example we can use it in a control structure to define the percentage of rests in a sequence.

[53]:
```
~rest = 0.8;       // probability of rests in sequence
r = Routine({
            20.do({
                  if(0.8.coin==true)
                        {Synth(\smp,[\buf,b,
                                    \pos, 0.23,
                                    \amp, 1,
                                    \dur, 0.08,
                                    \pan, 0,
                                    \t_gate, 1])};
                  0.1.wait;
                  })
                }).reset.play;
```

```
-> a Routine
```

- within a range (linear distribution) → rrand(min, max). For example we can use it to define random positions in a Buffer.

[ ]:
```
~rest = 0.8;        // probability of rests in sequence
b.duration.postln;  // range max
~posi = [0.5,3.4];  // start-end position points

r = Routine({
            20.do({
                  if(0.8.coin==true)
                     {Synth(\smp,[\buf,b,
                                 \pos, rrand(~posi[0], ~posi[1]), // random␣
 ↪position
                                 \amp, 1,
                                 \pan, 0,
                                 \dur, 0.08,
                                 \t_gate, 1])};
                     0.1.wait;
                  })
                }).reset.play;
```

```
12.455646258503
-> a Routine
```

- between 0 and max (linear distribution) → rand(max). For example we can use it to define random amplitude.

[ ]:
```
~rest = 0.8;       // probability of rests in sequence
~posi = [0.5,3.4]; // start-end position points

r = Routine({
            20.do({
                  if(0.8.coin==true)
```

```
                        {Synth(\smp,[\buf,b,
                                      \pos, rrand(~posi[0], ~posi[1]),
                                      \amp, rand(1.0),
                                      \pan, 0,
                                      \dur, 0.08,
                                      \t_gate, 1])};
                     0.1.wait;
                     })
                   }).reset.play;
```

```
-> a Routine
```

- bipolar (linear distribution) → rand2(+/- max)). For example we can use it to define random pan.

[ ]:
```
~rest = 0.8;        // probability of rests in sequence
~posi = [0.5,3.4]; // start and end position points

r = Routine({
             20.do({
                    if(0.8.coin==true)
                        {Synth(\smp,[\buf,b,
                                      \pos, rrand(~posi[0], ~posi[1]),
                                      \amp, rand(1.0),
                                      \pan, rand2(1.0),
                                      \dur, 0.08,
                                      \t_gate, 1])};
                     0.1.wait;
                     })
                   }).reset.play;
```

```
-> a Routine
```

- choose in a finite set with a linear distribution → [3,5,67,89].choose. For example we can use it to define random choice among a set of durations.

[ ]:
```
~rest = 0.8;                  // probability of rests in sequence
~posi = [0.5,3.4];            // start and end position points
~durs = [0.08,0.04,0.2,0.5]; // duration set

r = Routine({
             20.do({
                    if(0.8.coin==true)
                        {Synth(\smp,[\buf,b,
                                      \pos, rrand(~posi[0], ~posi[1]),
                                      \amp, rand(1.0),
                                      \dur, ~durs.choose,
                                      \pan, rand2(1.0),
                                      \t_gate, 1])};
```

```
                0.1.wait;
                })
            }).reset.play;
```

```
-> a Routine
```

There are other objects dedicated to nonlinear distributions but the principles on which they are based are the same as those just illustrated.

3 Sound synthesis

In this paragraph we will not delve into all the aspects of the individual synthesis techniques.

Only their main sonic and expressive possibilities related to composing music are focused.

Before starting let's define some utility: * one SynthDef → set of source signals. * one SynthDef → set of control signals (mouse, continuos). * two audio bus to connect the source Synths with filters and FFT Synth. * two control buses to control dynamically parameters (x, y and random signal values). * one spectroscope.

[253]:
```
SynthDef(\srcs,{arg type=0, freq=400, amp=0, buf=b, busOut=2, gate=0;
                var sig,env;
                    sig =  Select.ar(type,
                                [SoundIn.ar(0),             // 0
                                 WhiteNoise.ar,             // 1
                                 Impulse.ar(freq),          // 2
                                 Dust.ar(freq),             // 3
                                 Saw.ar(freq),              // 4
                                 PlayBuf.ar(1, buf, loop:1) // 5
                                 ]);
                        env = Linen.kr(gate,doneAction:2);
                    sig = sig * env * amp;
                Out.ar(busOut, sig)
        }).add;

SynthDef(\ksig,{arg type=0, range=#[0,1], freq=1, busOut=0;
                    var sig;
                        sig = Select.kr(type,
                                [MouseX.kr(range[0], range[1]),
                                 MouseY.kr(range[0], range[1]),
                                 LFNoise1.kr(freq).range(range[0], range[1])
                                ]);
                    Out.kr(busOut,sig)
        }).add;

x     = Bus.control(s, 1);
y     = Bus.control(s, 1);

s.freqscope;       // a spectroscope
```

```
-> Synth('sdel' : 2297)
```

3.1 Additive

With additive synthesis techniques we can realize: * harmonic spectra (pitched sounds). * inharmonic spectra (non-pitched sounds).

Both categories can have:

- fixed spectrum.
- variable spectrum.
- spectral envelope.

There are different ways to do that.

3.1.1 Wavetables

Harmonic spectra or periodic noises.

Fixed spectrum.

1. Draw a single waveform period.
2. Store it in a Buffer.
3. Play it back at a defined frequency by an oscillator.

The SynthDef.

[22]:
```
Buffer.freeAll;
b = Buffer.alloc(s, 2048);
b.sine2([1,3,4,6,8,9],[20,30,10,6,7,3].normalizeSum);  // Draw Partial_number,␣
↪magnitudes
b.plot;

SynthDef(\wts, {arg buf=b, freq=500, amp=0, dur=1, t_gate=0, pan=0, done=2;
                    var sig,bpf,env;
                        sig = Osc.ar(buf,freq);
                        env = Env.triangle(dur);          // Try to change␣
↪envelope
                        env = EnvGen.ar(env,t_gate,doneAction:done);
                        sig = Pan2.ar(sig*amp*env,pan);
                    Out.ar(0,sig)}
         ).add;
```

```
-> a SynthDef
```

We can paly it as in previous exemples.

- Deterministic.

[ ]:
```
~pitch = [60,64,65,68,72,73,68,67,60].midicps;
~amps  = [0.3,0.5,0.7,0.3,0.9,0.1]*0.2;
~pans  = [-1,0,1];
```

```
~durs  = [0.1,0.5,0.2,0.8,0.4];
~delta = [0.1,0.3,0.1,0.2];

r = Routine({
            20.do({arg i;
                    Synth(\wts,[\buf,b,
                                            \freq, ~pitch.wrapAt(i),
                                            \amp,  ~amps.foldAt(i),
                                            \dur,  ~durs.wrapAt(i),
                                            \pan, ~pans.foldAt(i),
                                            \done,2,
                                            \t_gate,1]);
                    ~delta.foldAt(i).wait;
                    })
            }).reset.play;
```

```
-> a Routine
```

- Random.

[ ]:
```
v = Routine({
                inf.do({
                    Synth(\wts,[\buf,b,
                                         // \freq,rrand(50,100).midicps,      ␣
 ↪// Atonal
                                            \freq, [68,72,75,80].midicps.choose,␣
 ↪// Triade
                                                        \amp,rand(0.2),
                                            \dur,rrand(4,8),
                                            \fade,rrand(0.2,8),
                                                        \pan, rand2(1.0),
                                            \done,2,
                                            \t_gate,1]);
                            rrand(0.1,2).wait;
                        })
            }).reset.play
```

```
-> a Routine
```

We can try to play them together.

Stop and kill.

[30]:
```
v.stop; v.free; r.free;
```

```
-> a Routine
```

3.1.2 Vectorial

This technique is a variant of wavetable lookup synthesis and allows the generation of variable spectra.

Harmonic spectra or periodic noises.

Variable spectrum.

1. Define an array of different waveforms.
2. During playback interpolate the reading dynamically with a control parameter.

[46]:
```
Buffer.freeAll;
b = Buffer.allocConsecutive(4,          // Number of Buffer
                            s, 1024);

b[0].sine2([1,3,5,7], [1,0.3,0.5,0.7]);   // Fill the buffers
b[1].sine2([2,4,6,8], [0.2,0.4,1.3,0.6]);
b[2].sine2([2,3,10,12,13,14], [1,0.2,0.5,0.6,1.3,0.5]);
b[3].sine2([1,3,4,7,8,9,13,18], [1.3,0.3,0.5,0.7,0.8,0.9,0.2,0.3]);
```

```
-> Buffer(3, 1024, 1, 44100.0, nil)
```

The SynthDef.

[47]:
```
SynthDef(\vec, {arg bufn=4,freq=400,amp=0,pos=0,pan=0,gate=0,done=2;
                var sig,env;
                    sig  = VOsc.ar(pos.linlin(-1,1,0,bufn), freq);
                    env  = Linen.kr(gate,doneAction:done);
                    sig  = Pan2.ar(sig*amp*env,pan);
                Out.ar(0, sig)
                }).add;
```

```
-> a SynthDef
```

As usual we can change timbre parameter in four different ways.

1. by sending single values (o to number of buses - 1).

[48]:
```
a = Synth(\vec, [\bufn,b.size-1, \freq,400, \amp,0.5, \pan,0, \gate,1]);

r = Routine({
            5.do({
                rrand(2,20).do({
                                a.set(\pos,rand2(1.0)); // Change position
                                0.1.wait;
                                });
                rrand(0.2,1).wait;
                });
                a.free;
            }).reset.play;
```

```
-> a Routine
```

2. by mapping a control signal on a control bus as argument.

[49]:
```
k = Synth(\ksig, [\type,0, \range,[-1, 1], \busOut, x]); // Mouse x
a = Synth(\vec, [\bufn,b.size-1, \freq,400, \pos, x.asMap, \amp,0.5, \pan,0,␣
↪\gate,1]);
```

```
-> Synth('vec' : 1364)
```

Change the control signal type and subaudio frequency.

[51]:
```
k.set(\type,2, \freq,1);
```

```
-> Synth('ksig' : 1363)
```

Fade out and kill.

[52]:
```
a.set(\gate, 0); k.free; p.free;
```

```
-> nil
```

Another classic technique in digital synth design is to map the amplitude envelope with spectral modification.

A simulation of real acoustic natural sounds.

[56]:
```
SynthDef(\vmap, {arg buf=0,bufn=4,freq=400,dur=1,amp=0,pan=0,t_gate=0,done=2;
                  var sig,bpf,env;
                      bpf   = Env.sine(dur);
                      env   = EnvGen.ar(bpf,t_gate,doneAction:done);
                      sig   = VOsc.ar(env.linlin(0,1,buf,bufn-1), freq);
                      sig   = Pan2.ar(sig*amp*env,pan);
                  Out.ar(0, sig)
        }).add;
```

```
-> a SynthDef
```

[57]:
```
r = Routine({
            10.do({
                Synth(\vmap, [\buf,b, \bufn,3,
                              \freq,rrand(40, 76).midicps,
                              \dur,rrand(4,12),
                              \amp,rand(1.0),
                              \pan,rand2(1.0),
                              \t_gate,1]);
                rrand(3,5).wait;
                    })
            }).reset.play;
```

```
-> a Routine
```

Stop and kill.

[58]:
```
r.stop;
```

```
-> a Routine
```

3.1.3 PWM

Pulse Width Modulation.

This technique is a variant of wavetable lookup synthesis and allows the generation of variable spectra.

Harmonic spectra.

Variable spectrum.

It is based on the concept of duty cycle.

img/pwm.png

By changing this parameter we modify the timbre.

The SynthDef.

[59]:
```
SynthDef(\pwm, {arg freq=300,duty=0.5,amp=0,gate=0,pan=0,done=2;
                    var sig, env;
                        sig  = Pulse.ar(freq, duty);
                        env  = Linen.kr(gate,doneAction:done);
                        sig  = Pan2.ar(sig * amp * env,pan);
                        Out.ar(0,sig)}
        ).add;
```

```
-> a SynthDef
```

The Synth.

[60]:
```
a = Synth(\pwm, [\amp,0.5,\duty,1/2,\gate,1]);
```

```
-> Synth('pwm' : 1385)
```

We can change th duty cycle from 0.0 to 0.5.

Let’s observe how the spectrum changes by set 1/3, 1/4, 1/5 etc.

[62]:
```
a.set(\duty, 1/4);
```

```
-> Synth('pwm' : 1385)
```

The duty cycle and other parameter control methods are the same as those illustrated previously.

Stop and kill.

[63]:
```
a.set(\gate, 0);
```

```
-> Synth('pwm' : 1385)
```

Some variations of this technique.

Blip → classic pulse train derived from MusicN languages (buzz) implemented in commercial synthesizers of the 90s.

We can define the number of harmonics as an argument.

The SynthDef.

```
[64]: SynthDef(\blip,{arg freq=300,nharm=20,amp=0,gate=0,pan=0,done=2;
                      var sig,env;
                      sig   = Blip.ar(freq, nharm);
                      env   = Linen.kr(gate,doneAction:done);
                      sig   = Pan2.ar(sig*amp*env,pan);
                      Out.ar(0,sig)
              }).add;
```

```
-> a SynthDef
```

The sequence.

```
[65]: a = Synth(\blip, [\amp,0.5,\gate,1]);
      r = Routine({
                  10.do({
                      rrand(2,10).do({
                      a.set(\freq, [50,57,62].midicps.choose,
                            \nharm,rrand(1,30));
                      0.1.wait;
                                      });
                  0.2.wait;
                        });
                  a.set(\gate,0);
                  }).reset.play;
```

```
-> a Routine
```

Control signal.

```
[68]: k = Synth(\ksig, [\type,2, \range, [1, 20], \freq,1, \busOut, x]);
      a = Synth(\blip, [\amp,0.5, \nharm, x.asMap, \gate,1])
```

```
-> Synth('blip' : 1390)
```

Stop and kill:

```
[ ]: a.set(\gate,0); k.free;
```

```
-> nil
```

We can also realize PWM with different waveforms stored in a Buffer (timbre stretching).

```
[77]: Buffer.freeAll;
      b = Buffer.alloc(s, 2048);
```

```
b.sine2([1,3,4,6,8,9],                 // Draw Partial_number
        [20,30,10,6,7,3].normalizeSum, // magnitudes
        asWavetable: false);

SynthDef(\wsth,{arg buf=b, freq=500, sfact=1, amp=0,gate=0,pan=0,done=0;
                   var p,f,c,sig,env;
                       p   = BufFrames.kr(buf);                        //␣
 ↪Wavetable size (p)
                       f   = sfact;                                    //␣
 ↪Stretching factor
                       c   = LFSaw.ar(freq) *  0.5 * p * f + (p * 0.5); //␣
 ↪Bipolar phasor (a)
                       sig = BufRd.ar(1,buf,c,1,4);
                       env = Linen.kr(gate,doneAction:done);
                       sig = Pan2.ar(sig*amp*env,pan);
                    Out.ar(0,sig)}
        ).add;
```

```
-> a SynthDef
```

The Synths.

[80]:
```
k = Synth(\ksig, [\type,0, \range, [1,10], \freq,1, \busOut, x]);
a = Synth(\wsth, [\buf,b,\freq,100,\amp,0.12,\sfact,x.asMap,\done,2,\gate,1]);
```

```
-> Synth('wsth' : 1399)
```

Fade out and kill.

[ ]:
```
a.set(\gate,0); k.free;
```

```
-> Synth('ksig' : 1398)
```

3.1.4 Classic

Harmonic and inharmonic spectra.

Spectral envelope.

The techniques just described belong to the world of digital audio.

In the early electroacoustic music studios (Cologne and Milan), additive synthesis was achieved by recording each individual partial and then mixing it with the others.

Let's simulate this technique defining a SynthDef that play only one partial.

[82]:
```
SynthDef(\par, {arg freq=500,amp=0,pha=0,dur=1,pan=0,t_gate=0;
                   var sig, env;
                       sig = SinOsc.ar(freq, pha, amp);
                       env = Env.perc(0.1*dur,0.9*dur);
                       env = EnvGen.ar(env, t_gate, doneAction:2);
                       sig = sig * env;
```

```
                    sig = Pan2.ar(sig, pan);
                    Out.ar(0, sig)
        }).add;
```

```
-> a SynthDef
```

Then we can define spectra as Arrays and play it through a loop structure.

[87]:
```
//~freqs = [1,    3,    4,    5,     8]* 52.midicps;   // Harmonic
~freqs = [1169,1733,854, 1596, 1480];                // Inharmonic
~amps  = [10,   100,   80,    30,   171].normalizeSum; // Relative amplitudes (sum=1)
~dur   = 3;                                          // Overall duration
~durs  = [0.1, 0.8,  0.5, 0.3,   1.0] * ~dur;        // Partials durations         ␣
↪

~freqs.size.do({arg i;      // loop without .wait
                    Synth(\par, [\freq, ~freqs[i],
                                 \amp,   ~amps.at(i),
                                 \dur,   ~durs[i],
                                 \t_gate,1])
                })
```

```
-> 5
```

Deterministic sequence.

[117]:
```
~seq = [60,  64, 67, 68, 71, 72].scramble.midicps;
~amp = [0.9,0.7,0.5,0.3,0.2,0.1].scramble;
~dur = [0.1,0.4,0.2,0.8,0.5,  1].scramble;
g    = ~seq.size;

u = Routine({var parz, pamp, pdur;
                 parz = [ 1,    3,   4,   5,   8];  // Spectrum
                 pamp = [10, 100, 80, 30,171].normalizeSum;
                 pdur = [0.1,0.8,0.5,0.3,1.0];
                 h    = parz.size;
             g.do({ arg i;
                    var freq, amps, durs;
                        freq = parz * ~seq[i];
                        amps = pamp * ~amp[i];
                        durs = pdur * ~dur[i];
                        h.do({arg i;
                                Synth(\par,[\freq, freq[i],
                                        \amp,  amps[i],
                                                \dur,  durs[i],
                                                \t_gate, 1])
                          });
                        0.1.wait;
                    })
```

```
            }).reset.play;
```

```
-> a Routine
```

Random sequence - difference between chords and spectra.

[118]:
```
~bpm = 72;
t = TempoClock.new(~bpm/60);
r = Routine({
             inf.do({var nharm;
                     if(0.2.coin==false) // Random rest percentage
                       {nharm = rrand(2,9);
                        nharm.do({arg i;
                                Synth(\par,[\freq, [70,74,75,78,82,86,90].
 ↪midicps.choose, // rrand(500,1200),
                                            \amp,  rand(1/nharm),
                                            \dur,  [0.05,0.2,0.4].choose,
                                            \pan, [-1,1].choose,
                                            \t_gate,1])
                                });
                        };
                        [0.125,0.25].choose.wait;
                     })
            }).reset.play(t);
```

```
-> a Routine
```

Stop and kill.

[ ]:
```
r.stop;
```

```
-> a Routine
```

3.2 Subtractive

Sound source ranging from white noise to rich spectra filtered by one or more filter(s).

The main types are: * Low Pass Filter (LPF) * High Pass Filter (HPF) * Band Pass Filter (BPF) * Band Reject Filter (BRF) * Resonant Filter (Resonz)

In SuperCollider are all 2nd order Butterworth filters (-12 dB/oct).

We can control cut/central frequency (Hz).

In BPF, BRF and Resonz we can also control the bandwidth (reciprocal of Q). * low values → narrow band . * high values → wide band).

Let's define: * one SynthDef for signal source. - we can switch among three types (0 = whitenoise, 1 = sawtoothwave, 2 = soundfile) - we can select the bus out. * one SynthDef for each filter type. - we can select the bus in and the bus out.

In SuperCollider there are many audio buses.

In a stereo system: * public buses → 0 and 1 (L and R) * private buses → 2 and up.

A simple signal flow.

img/filt_1.png

The SynthDefs.

[126]:
```
Buffer.freeAll;
b = Buffer.read(s, "/Users/andreavigani/Desktop/GHub/EMC/3_fixed/suoni/bach.
 ↪wav");

SynthDef(\lpf, {arg freq=500, amp=0, busIn=2, busOut=0, pan= -1;
                    var sig;
                        sig = In.ar(busIn);
                    sig = LPF.ar(sig, freq) * amp;
                    sig = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig)
         }).add;
SynthDef(\hpf, {arg freq=500, amp=0, busIn=2, busOut=0, pan= -1;
                    var sig;
                        sig = In.ar(busIn);
                    sig = HPF.ar(sig, freq) * amp;
                    sig = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig)
         }).add;
SynthDef(\bpf, {arg freq=500, amp=0, bw=1, busIn=2, busOut=0, pan= -1;
                    var sig;
                        sig = In.ar(busIn);
                    sig = BPF.ar(sig, freq, bw) * amp;
                    sig = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig)
         }).add;
SynthDef(\brf, {arg freq=500, amp=0,  bw=1, busIn=2, busOut=0, pan= -1;
                    var sig;
                        sig = In.ar(busIn);
                    sig = BRF.ar(sig, freq,bw) * amp;
                    sig = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig)
         }).add;
SynthDef(\res, {arg freq=500, amp=0,  bw=1, busIn=2, busOut=0, pan= -1;
                    var sig;
                        sig = In.ar(busIn);
                    sig = BRF.ar(sig, freq,bw) * amp;
                    sig = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig)
```

```
        }).add;
```

```
-> Synth('lpf' : 2222)
```

Now we can test it.

The Synths (in right order).

[146]:
```
a = Synth(\srcs, [\type,1, \buf,b, \amp,1, \busOut,12, \gate,1]);
f = Synth(\lpf,  [\freq,1000, \amp,1, \busIn,12, \busOut,0, \pan,␣
↪-1],s,\addToTail);
```

```
-> Synth('lpf' : 2233)
```

Change the filter cutting frequency parameter in the usual ways.

[160]:
```
f.set(\freq, rrand(200,2000).postln);
```

```
1280
-> Synth('lpf' : 2235)
```

Stop and kill.

[161]:
```
a.set(\gate, 0); f.free;
```

```
1849
-> Synth('lpf' : 2235)
```

We can sculpt the spectrum using filter banks in two combinations:

1. series → output of one filter is the input of the next until the public output.

[164]:
```
a = Synth(\srcs, [\type,5,   \buf,b, \amp, 1,             \busOut,12,          ␣
↪ \gate, 1]);
b = Synth(\lpf,  [\freq,100,         \amp,0.6, \busIn,12, \busOut,13, \pan,␣
↪-1],s,\addToTail);
c = Synth(\brf,  [\freq,800, \bw, 2, \amp,0.6, \busIn,13, \busOut,14, \pan,␣
↪-1],s,\addToTail);
d = Synth(\brf,  [\freq,800, \bw, 1, \amp,2.5, \busIn,14, \busOut,0,  \pan,␣
↪0],s,\addToTail);

r = Routine({
            inf.do({
                    b.set(\freq, rrand(8000,12000));
                    c.set(\freq,rrand(200,1000));
                    d.set(\freq,rrand(800,3000));
                    0.25.wait;
                    })
            }).reset.play
```

```
-> Synth('brf' : 2239)
```

Stop it and kill.

[ ]:
```
r.stop; a.set(\gate, 0);  b.free; c.free; d.free;
```

```
-> a Routine
```

2. parallel → output of source is the input of all filters, ouputs of filters mix together.

[174]:
```
~bw     = 0.1;     // Try to change
~amp    = 1;
~source = 5;
~rib    = 0.1;

a = Synth(\srcs, [\type,~source, \buf,b,\amp,   1,            \busOut,12, \gate,␣
↪1]);
b = Synth(\bpf,  [\freq,100, \bw, ~bw,  \amp,~amp, \busIn,12,  \busOut,0, \pan,␣
↪-1],s,\addToTail);
c = Synth(\bpf,  [\freq,800, \bw, ~bw,  \amp,~amp, \busIn,12,  \busOut,0, \pan,␣
↪-1],s,\addToTail);
d = Synth(\bpf,  [\freq,800, \bw, ~bw,  \amp,~amp, \busIn,12,  \busOut,0, \pan,␣
↪0],s,\addToTail);

r = Routine({
            inf.do({
                    b.set(\freq, rrand(100,700));
                    c.set(\freq,rrand(1700,2100));
                    d.set(\freq,rrand(3000,3100));
                    ~rib.wait;
                    })
            }).reset.play
```

```
-> Synth('bpf' : 2259)
```

[175]:
```
r.stop; a.set(\gate, 0); b.free; c.free; d.free;
```

```
-> a Routine
```

In SuperCollider we have also pre builded filterbanks which can be used to simulate the resonant modes of an object.

The SynthDef.

[181]:
```
SynthDef(\reson, {arg freqs=#[800, 1071, 1153, 1723],
                      amps=#[0.1,0.1,0.1,0.1],
                      ringtimes=#[1, 1, 1, 1],
                      busIn=2,
                      busOut=0,
                      pan=0;
                  var in, sig;
                      in  =  In.ar(busIn);
                      sig = DynKlank.ar(`[freqs, amps, ringtimes ], in);
                      sig = Pan2.ar(sig, pan);
                  Out.ar(busOut, sig);
}).add;
```

```
-> a SynthDef
```

The Synths.

[203]:
```
e = Synth(\srcs,  [\type, 2,\freq,3,\amp,0.5,\busOut,12,\gate,1]);
a = Synth(\reson, [\busIn,12, \busOut,0,\pan,0], s, \addToTail);
```

```
-> Synth('reson' : 2269)
```

Change resonator parameter.

[205]:
```
a.setn(\freqs, Array.rand(4, 500, 9000));
a.setn(\amps, Array.rand(4, 0.1, 1));
a.setn(\ringtimes, Array.rand(4, 0.2, 2.4) );
```

```
-> Synth('reson' : 2271)
```

Change source. 1. → White Noise. 3. → Impulse.Dust. 2. → Dust.

[210]:
```
e.set(\type,3,\amp,0.1);
```

```
-> Synth('srcs' : 2270)
```

Fade out and kill.

[211]:
```
e.set(\gate,0); a.free;
```

```
-> Synth('srcs' : 2270)
```

3.3 FM

In additive synthesis and in some cases in subtractive synthesis the signals are added together to obtain complex spectra.

The computational cost is high since for each partial or filter we have to generate a UGen.

Through the different basic modulation techniques (RM, AM, FM and PM) we can obtain equally rich spectra with only two signals: * carrier * modular

In this paragraph we explore characteristics of frequency modulation (FM) which in its most basic form consists of using a rescaled sinusoidal control signal (modular) to dynamically vary the frequency of a sinusoid (carrier).

Three parameters; * carrier frequency (Hz). * modular frequency → MouseX from 2 to 500 Hz. * deviation from carrier frequency (ampitude of modular signal) → MouseY from 2 to 500 Hz.

If the frequency of modular signal is sub audio and deviation is under ~5 Hz → ‘vibrato effect’.

[218]:
```
SynthDef(\fm_1, {arg carf=440;
                     var modf, dev, mod, car;
                    modf = MouseX.kr(1,500);
                    dev  = MouseY.kr(1,500);
```

```
                         mod = carf + (SinOsc.ar(modf) * dev);
                         car = SinOsc.ar(mod);
                      Out.ar(0,car*0.3)
                  }).add;
```

```
-> Synth('fm_1' : 2273)
```

The Synth.

[219]:
```
a = Synth(\fm_1);
```

```
-> a SynthDef
```

We can already recognize the characteristic sound of this technique

Kill the Synth.

[220]:
```
a.free;
```

```
-> Synth('fm_1' : 2274)
```

In classic FM algorithm we can control: * modulation index → quantity of modulation. * harmonicity index → quality of modulation.

We can realize a lot of different sounds changing only these two parameters.

[233]:
```
SynthDef(\fm, {arg freq=440, amp=0, hid=0, mid=0, grain=0.001, pan=0, gate=0;
                    var mod, fmod, por, env, sig;
                        fmod = freq * hid.round(grain);
                        mod  = freq + (SinOsc.ar(fmod) * fmod * mid);
                        por  = SinOsc.ar(mod);
                        env  = Linen.kr(gate, doneAction:2);
                    sig  = por * amp * env;
                    sig  = Pan2.ar(sig, pan);
                    Out.ar(0, sig)
                 }).add;
```

```
-> a SynthDef
```

Let's explore it with mouse position.

[241]:
```
k = Synth(\ksig, [\type,0, \range, [0, 15], \busOut, x]); // Harmonicity index
j = Synth(\ksig, [\type,1, \range, [0, 25], \busOut, y]); // Modulation index
a = Synth(\fm,   [\freq,200, \amp,0.4, \hid,x.asMap, \mid,y.asMap, \gate,1]);
```

```
-> Synth('ksig' : 2281)
```

Fade out and kill.

[242]:
```
a.set(\gate,0); k.free; j.free;
```

```
-> Synth('fm' : 2285)
```

I suggest exploring the sonic potential of this technique empirically through mouse controls.

The x and y positions of the mouse could be dynamically reported in the Post window with '.poll' method invoked on MouseX and MouseY UGens in SynthDef.

When we hear a sound that interests us we can write it down.

We can then recall them and eventually interpolate them with other sounds through envelopes or other.

One Example.

[243]:
```
SynthDef(\env, {arg levels=#[0,1,0.5,0.7,0], times=#[1,1,1,1], dur=1, busOut=0,␣
↪t_gate=0;
                var env;
                    env = Env.new(levels,times.normalizeSum * dur);
                    env = EnvGen.kr(env, t_gate);
                Out.kr(busOut,env)
}).add;
```

```
-> Synth('ksig' : 2284)
```

The Synths.

[244]:
```
h = Synth(\env,[\levels, [3.5,3.4,3.6,3.7,3.35],  // Harmonicity index
                \times,  [3,4,2,5],
                \dur, 40,
                \busOut, x]); // write on control bus
i = Synth(\env,[\levels, [0,1,0.5,1.2,0.7],       // Modulation index
                \times,   [1,4,9,5],
                \dur, 42,
                \busOut, y]); // write on control bus

a = Synth(\fm, [\freq,100, \amp,0.4, \hid,x.asMap, \mid,y.asMap]);

r = Routine.new({
                a.set(\gate,1);
                h.set(\t_gate,1);
                i.set(\t_gate,1);
                50.wait;
                a.release(8);h.free;i.free;k.free;j.free;
                }).reset.play
```

```
-> a SynthDef
```

3.4 FFT

Some classical techniques related to the analysis and resynthesis of sound via the Fast Fourier Transform.

We analyze a signal by transforming it from time domain to frequency domain obtaining values of: * magnitude * phase

of a set of equidistant frequencies (bins) within the spectrum (think of them as the teeth of a comb).

We can manipulate these values in various ways and then bring them back into the time domain for playback.

There are two main procedures: * analyze, process, and resynthesize a single signal. * analyze two signals combine the data in various ways and resynthesize the result.

In SuperCollider there ia a set of Classes dedicated to Phase Vocoder which is a historical FFT analysis and resynthesis technique.

To use classes in these examples download and install sc3-plugins.

Be careful because FFT processes are CPU expensive.

Spectral Delay It differs from non-spectral delays in the fact that we can dynamically change delay time without pitch artefacts.

[714]:
```
Buffer.freeAll;
b = Buffer.read(s, "/Users/andreavigani/Desktop/GHub/EMC/3_fixed/suoni/bach.
↪wav");

SynthDef(\sdel,{arg busIn=0,busOut=0,amp=0.5,fadeT=0.
↪5,gate=0,pan=0,maxdel=1,del= 0;
                    var in,size,chain,env,sig;
                        in    = In.ar(busIn,1);
                        size  = 2048;
                        chain = FFT(LocalBuf(size), in);                   //␣
↪analysis
                        chain = chain.pvcollect(size, {arg mag,fasi,bin;  //␣
↪process
               [mag + DelayN.kr(mag, maxdel, del.linlin(0,1,0,maxdel)),
                fasi] },0,256,-1);
                        env  = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                        sig  = IFFT(chain) * env * amp;                   //␣
↪resynth
                        sig  = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig);
        }).add;
```

```
-> Synth('ksig' : 2322)
```

The Synths.

[ ]:
```
k = Synth(\ksig, [\type, 0, \range, [0, 1], \busOut,x]); // Normalized on maxdel
e = Synth(\srcs, [\type, 5, \amp,1, \buf,b, \busOut,12, \gate,1 ],s,'addToTail');
a = Synth(\sdel, [\busIn,12, \amp,1, \maxdel,5, \del,x.asMap,␣
↪\gate,1],s,'addToTail');
```

```
-> a SynthDef
```

Fade out and kill.

[260]:
```
k.free; e.set(\gate,0); a.set(\gate,0);
```

```
-> Synth('sdel' : 2306)
```

BrickWall filter Only bins above or below a threshold between +/-1 pass.

We can define ideal low and high pass filters.

[262]:
```
SynthDef(\bwall, {arg busIn=0,busOut=0,amp=0.5,fadeT=0.
↪5,gate=0,pan=0,lpf=20000,hpf=20;
                      var in,size,chain,env,sig;
                          in     = In.ar(busIn,1);
                              size  = 2048;
                          chain = FFT(LocalBuf(size), in);
                          chain = PV_BrickWall(chain, lpf.linlin(20,20000,-1,0));
↪ // Low pass
                          chain = PV_BrickWall(chain, hpf.linlin(20,20000,0,1));␣
↪ // High pass
                          env   = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                          sig   = IFFT(chain) * env * amp;
                          sig   = Pan2.ar(sig, pan);
                      Out.ar(busOut, sig);
         }).add;
```

```
-> a SynthDef
```

The Synths.

[266]:
```
e = Synth(\srcs,  [\type, 5,  \amp,1, \buf,b, \busOut,12, \gate,1 ]);
a = Synth(\bwall, [\busIn,12, \amp,1, \lpf,6000, \hpf, 1000,␣
↪\gate,1],s,'addToTail');
```

```
-> Synth('bwall' : 2308)
```

Change filter freqs.

[267]:
```
a.set(\lpf, 500, \hpf,200);
```

```
-> Synth('bwall' : 2310)
```

Dynamically (mouse x and y).

[268]:
```
k = Synth(\ksig, [\type,0, \range, [500,10000], \busOut,x]);
j = Synth(\ksig, [\type,0, \range, [200, 1000], \busOut,y]);

a.set(\lpf, x.asMap, \hpf, y.asMap);
```

```
-> Synth('bwall' : 2310)
```

Fade out and kill.

[269]:
```
e.set(\gate,0);a.set(\gate,0);k.free;j.free;
```

```
-> Synth('bwall' : 2310)
```

Rect Comb filter Make gaps in spectrum. * teeth → number of teeth in the comb. * fase → starting phase of comb pulse. * width → pulse width of the comb.

[715]:
```
SynthDef(\rcomb, {arg busIn=0,busOut=0,amp=0.5,fadeT=0.
↪5,gate=0,pan=0,teeth=10,fase=0,width=0.5;
                      var in,size,chain,env,sig;
                          in    = In.ar(busIn,1);
                              size  = 2048;
                          chain = FFT(LocalBuf(size), in);
                          chain = PV_RectComb(chain, teeth, fase, width);
                          env   = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                          sig   = IFFT(chain) * env * amp;
                          sig   = Pan2.ar(sig, pan);
                      Out.ar(busOut, sig);
         }).add;
```

```
Buffer UGen: no buffer data
-> a SynthDef
```

The Synths.

[716]:
```
e = Synth(\srcs,  [\type, 5, \amp,1, \buf,b, \busOut,12, \gate,1 ]);
a = Synth(\rcomb, [\busIn,12, \amp,1, \gate,1],s,'addToTail');
```

```
-> a SynthDef
```

Change phases.

[727]:
```
a.set(\fase,rand(1.0).postln);
```

```
0.97739613056183
-> Synth('rcomb' : 2035)
```

Change width and teeth dynamically.

[728]:
```
k = Synth(\ksig, [\type,0, \range, [0.01, 0.5], \busOut,x]);
j = Synth(\ksig, [\type,0, \range, [1, 30], \busOut,y]);

a.set(\width, x.asMap, \teeth, y.asMap);
```

```
0.40265238285065
-> Synth('rcomb' : 2035)
```

Fade out and kill.

[713]:
```
e.set(\gate,0);a.set(\gate,0);k.free;j.free;
```

```
-> a SynthDef
```

Mag Smear Average magnitudes across bins.

We can set the number of bins to average on each side of bin.

[290]:
```
SynthDef(\smear, {arg busIn=0,busOut=0,amp=0.5,fadeT=0.5,gate=0,pan=0,bins=1;
                      var in,size,chain,env,sig;
                          in    = In.ar(busIn,1);
                              size  = 2048;
                          chain = FFT(LocalBuf(size), in);
                          chain = PV_MagSmear(chain, bins);
                          env   = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                          sig   = IFFT(chain) * env * amp;
                          sig   = Pan2.ar(sig, pan);
                      Out.ar(busOut, sig);
         }).add;
```

```
-> Synth('ksig' : 2322)
```

The Synths.

[295]:
```
k = Synth(\ksig,  [\type,0, \range, [0, 10], \busOut,x]);
e = Synth(\srcs,  [\type, 5, \amp,1, \buf,b, \busOut,12, \gate,1 ]);
a = Synth(\smear, [\busIn,12, \amp,1, \bins, x.asMap, \gate,1],s,'addToTail');
```

```
-> Synth('ksig' : 2326)
```

Fade out and kill.

[296]:
```
e.set(\gate,0);a.set(\gate,0);k.free;
```

```
-> Synth('smear' : 2331)
```

Freeze Freeze magnitudes when freeze argument > 0.

[298]:
```
SynthDef(\frz, {arg busIn=0,busOut=0,amp=0.5,fadeT=0.5,gate=0,pan=0,frz=0;
                    var in,size,chain,env,sig;
                        in    = In.ar(busIn,1);
                            size  = 2048;
                        chain = FFT(LocalBuf(size), in);
                        chain = PV_MagFreeze(chain, frz);
                        env   = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                        sig   = IFFT(chain) * env * amp;
                        sig   = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig);
         }).add;
```

```
-> a SynthDef
```

The Synths and the sequence.

[299]:
```
e = Synth(\srcs, [\type, 5, \amp,1, \buf,b, \busOut,12, \gate,1 ]);
a = Synth(\frz,  [\busIn,12, \amp,1, \gate,1],s,'addToTail');

r = Routine({
         10.do({a.set(\frz,0);
                rrand(0.2,2).wait;
                a.set(\frz,1);
                rrand(0.6,2).wait;
                });
         e.set(\gate,0); a.set(\gate,0);
            }).reset.play;
```

```
-> a SynthDef
```

Cross synthesys. There are several cross-synthesis techniques.

In this example, the bins of signal A are replaced by those of signal B.

[300]:
```
SynthDef(\cross, {arg busIn=#[0,1],busOut=0,amp=0.5,fadeT=0.
↪5,gate=0,pan=0,morph=0;
                    var inA,inB,size,chainA,chainB,chain,env,sig;
                        inA    = In.ar(busIn[0],1);
                            inB    = In.ar(busIn[1],1);
                            size   = 2048;
                            chainA = FFT(LocalBuf(size), inA);
                                chainB = FFT(LocalBuf(size), inB);
                        chain  = PV_Morph(chainA, chainB, morph);
                        env    = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                        sig    = IFFT(chain) * env * amp;
                        sig    = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig);
         }).add;
```

```
-> a Routine
```

The Synths (two different sources).

[301]:
```
k = Synth(\ksig,  [\type,0, \range, [0, 1.0],    \busOut,x]);
e = Synth(\srcs,  [\type, 5, \amp,1, \buf,b,     \busOut,12, \gate,1 ]);
g = Synth(\srcs,  [\type, 2, \amp,0.8, \freq, 8, \busOut,13, \gate,1 ]);
a = Synth(\cross, [\busIn,[12, 13], \amp,1, \morph, x.asMap,␣
↪\gate,1],s,'addToTail');
```

```
-> a SynthDef
```

[302]:
```
e.set(\gate,0); g.set(\gate,0); a.set(\gate,0);k.free;
```

```
-> Synth('cross' : 2337)
```

4 Algorithmic structures

In this chapter we expanded our sonic vocabulary.

We can now design formal structures by adopting the same strategies illustrated in the previous chapter or through non-deterministic algorithms.

Let's give an example of this second case.

Let's suppose we want to create a piece with dreamlike characteristics based on three musical objects: 1. a continuos changing crystal sound (filter bank). 2. a percussive brilliant sound (vector). 3. a strange piano (FFT filter).

Let's design the first.

Define one SynthDef and a function that design the musical object. - random spectra, - random duration - random release

[840]:
```
SynthDef(\res, {arg freqs=#[800, 1071, 1153, 1723],
                    amps=#[0.1,0.1,0.1,0.1],
                    ringtimes=#[1, 1, 1, 1],
                    busOut=0, gate=0;
                var sig, amp, env, pan;
                    sig = WhiteNoise.ar;  // source
                    amp = LFNoise1.kr(0.2).range(0,0.07); // ksig random amp
                    sig = sig * amp;      // amp source
                    sig = DynKlank.ar(`[freqs, amps, ringtimes ], sig);
                    env = Linen.kr(gate, 3, doneAction:2);
                    pan = LFNoise1.kr(0.3);           // ksig random pan
                    sig = Pan2.ar(sig * amp * env, pan);
                Out.ar(busOut, sig);
}).add;

a = {Routine({var fqs, aps, rng, res;
              fqs = Array.rand(4, 500, 9000);
              aps = Array.rand(4, 0.1, 1);
              rng = Array.rand(4, 0.2, 0.4);
              res = Synth(\res, [\freqs,fqs,\amps,aps,\ringtimes,rng,\gate,1]);
              rrand(0.5,4).wait;
              res.release(rrand(6,8)) // note off
              }).play;
    }
```

```
-> a Function
```

Turns the musical object on and off randomly.

Test it.

[841]:
```
~glass = Routine({
                  inf.do({
                        if(0.8.coin==true){a.value};
                        [0.2,3,8].choose.wait;
                        })
                }).reset.play;
```

```
-> a Function
```

Kill it.

[842]:
```
~glass.stop;
```

```
-> a Routine
```

Let's design the second in the same manner.

- random pitch from a set.
- random rythm.
- random amplitudes.
- random sequence activation.
- random pan.

[831]:
```
Buffer.freeAll;
b = Buffer.allocConsecutive(4,         // Number of Buffer
                            s, 1024);

b[0].sine2([1,2,5,7], [1,0.3,0.5,0.7]);   // Fill the buffers
b[1].sine2([2,3,8,9], [0.2,0.4,1.3,0.6]);
b[2].sine2([3,4,8,9,13,14], [1,0.2,0.5,0.6,1.3,0.5]);
b[3].sine2([1,3,4,7,8,9,12,15], [1.3,0.3,0.5,0.7,0.8,0.9,0.2,0.3]);

SynthDef(\pik, {arg bufn=4,freq=400,dur= 1,amp=0,pan=0,t_gate=0;
                var sig,env;
                    env = Env.perc(0.01,dur-0.01);
                    env = EnvGen.kr(env,t_gate,doneAction:2);
                    sig = VOsc.ar(env.linlin(0,1,0,bufn), freq);
                    sig = Pan2.ar(sig * amp * env,pan);
                Out.ar(0, sig)
                }).add;

b = {Routine({
            rrand(2,10).do({
                            Synth(\pik,[\freq,[90,94,95,97,100,104,105].choose.
↪midicps,
                                        \dur, exprand(0.3,4.3),
                                        \amp, rrand(0.01,0.1),
                                        \pan, rand2(1),
                                        \t_gate,1]);
                            [0.15,0.1,0.2,0.25].choose.wait;
                            })
            }).play;
```

```
    }
```

```
-> a Routine
```

Turns the musical object on and off randomly.

[832]:
```
~plink = Routine({
                inf.do({
                        b.value;
                        [3,6,8].choose.wait;
                        })
                }).reset.play;
```

```
-> a Function
```

Kill it.

[833]:
```
~plink.stop;
```

```
-> a Routine
```

Let's design the third musical object.

[834]:
```
v = Buffer.read(s, "/Users/andreavigani/Desktop/GHub/EMC/3_fixed/suoni/bach.
↪wav");

SynthDef(\spy,{arg buf=v,busOut=0,amp=0.
↪5,fadeT=4,gate=0,pan=0,teeth=10,fase=0,width=0.5;
                    var sig,size,chain,env;
                        sig   = PlayBuf.ar(1, buf, loop:1);
                            size  = 2048;
                        chain = FFT(LocalBuf(size), sig);
                        chain = PV_RectComb(chain, teeth, fase, width);
                        env   = Linen.kr(gate,fadeT,1,fadeT,doneAction:2);
                        sig   = IFFT(chain) * env * amp;
                        sig   = Pan2.ar(sig, pan);
                    Out.ar(busOut, sig);
         }).add;
c = {Routine({var piano;
              piano = Synth(\spy, [\buf, v,
                                   \amp,rrand(0.5,1),
                                   \fadeT,rrand(1,5),
                                   \pan,rand2(1),
                                   \teeht,rrand(4,20),
                                   \width, rrand(0.1,0.4),
                                   \fase,rrand(0.4,0.7),
                                   \gate,1]);
              rrand(0.5,4).wait;
              piano.release(rrand(5,10)) // note off
              }).play;
```

```
    }
```

```
-> a Routine
```

Test it.

[835]:
```
~piano = Routine({
                inf.do({
                        c.value;
                        [3,6,8].choose.wait;
                        })
                }).reset.play;
```

```
-> a Function
```

Kill it.

[836]:
```
~piano.stop;
```

```
-> a Routine
```

We can now put it all together in a score that turns different combinations on and off randomly.

[845]:
```
~sec = 10; // Number of sections
~score = Routine({
                ~sec.do({
                    if(0.6.coin==true){~plink.reset.play}{~plink.stop};
                    rrand(0.1,3).wait;
                    if(0.8.coin==true){~piano.reset.play}{~piano.stop};
                    rrand(0.1,3).wait;
                    if(0.4.coin==true){~glass.reset.play}{~glass.stop};
                    rrand(0.1,3).wait;
                });
                ~plink.stop;
                3.wait;
                ~piano.stop;
                5.wait;
                ~glass.stop
                }).reset.play
```

```
-> a Routine
```

At each performance the piece will be a little different in its form but will always maintain the characteristics that we had set ourselves.

This is a simple example for explanatory reasons but it is clear how through this process we can arrive at designing extremely complex structures and forms.

5 Composition sketches proposal

Create a musical work using the instruments and the procedure just described.

Suggestions: * research other synthesis techniques or delve deeper into some of them. * modify the contents of the proposed SynthDefs based on your musical parameter control needs. * explore algorithms that dynamically modify the form through random control structures such as if, case, and switch (search for help files). * verify how and how much random parameters can influence the realization of the initial musical idea.