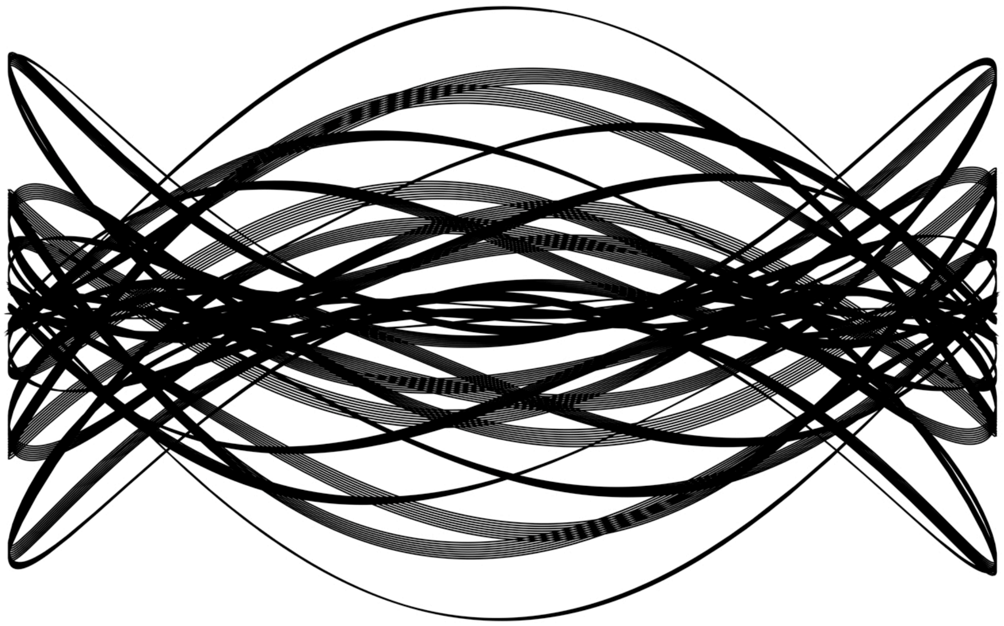
FIRST REFLECTIONS ON MY COMPOSITIONAL PROCESS

Casper Schipper 2009



FIRST REFLECTIONS ON MY COMPOSITIONAL PROCESS

Bachelor Thesis Sonology, 2009

Preface:

Considering that this is my fourth year at Sonology, time has been going very fast. Although a lot of progress has been made since my first pieces, I still feel I'm only at the start of something bigger. Now comes the time though to stand still for a moment and reflect on the past years. I must admit that I was not really trembling with joy having to write about my music, my feeling was (and still is) that the process that creates my music does not lend itself well to be written down in a linear way. I also noticed during the writing of this thesis that ideas that work very well in music, can sometimes appear naïve or simple on paper and vice versa. However, I think I still learned a lot of new things during the making of this Thesis.

Apart from my own pieces, I've mainly focussed on the work of Koenig, who obviously doesn't seem to have any trouble at all with writing about his music. His writings have been incredible helpful in reflecting on my own work. Because my latest works have been focussing more and more on the idea of algorithmic composition with the help of software I've developed myself, this thesis will focus especially on his two composition programs: Project 1 and Project 2.

This thesis will consist of three more or less separate parts: first of all I will give a description of a selection of my compositions until last year. Then (after a short reflection on what role algorithms play in my music) I will present an explanation and reaction on the two Project of Koenig and finally a explanation of my own software.

I would especially like to thank Kees Tazelaar and Paul Berg for their help and the fact that discussions with them have always been fruitful. I would also like to thank Patrick van Deursen for his inspiring lessons on classical music (Beethoven and Ligeti in particular). Further (and I'm probably forgetting a lot of people) : Gottfried Michael Koenig, Wim 'Wimster' Boogman, Cort Lippe, Billy Bultheel, Greg Grigoropoulos, Sophocles Arvanitis, Raviv Ganchrow, Joel Ryan, Rosalie Hirsh, Johan van Kreij, Babis Giannakopoulos, Peter Pabon, Richard Barrett, Paul Jeukendrup, Wouter Snoei, Emmanuel Flores Elias, Ji Youn Kang and all other people at Sonology that make it such a special and pleasant place to be.

Part 1 :
my works until 2008

Quite Analog

Influence of Koenig’s Terminus:

One of the goals for Koenig in the creation of Terminus was to ‘reduce handcraft’. He did not totally dispensed with it though, because he was convinced that it was not possible yet to get the amount of articulation and control over time structure. The basic material of the piece is a mixture of glissandi between 331 and 1062 Hz. The dynamics of the frequency is constrained between 1/4 and 4 octaves. The amplitude is modulated with 3 frequency’s 1/3, 1 and 4 Hz.
This results in a material having certain characteristics:
- A certain bandwidth
- An average frequency
- Average inner speed of the glissandi
- A ‘spectrum’ containing 5 (moving) sinewaves.

He then used 7 different transformations to make variations on this source material.
Each has a different influence on the three basic characteristics:

Name	**effects on:**			
	Bandwidth	Avg. Freq.	Inner speed	Spectrum
Transposition	X	-	X	-
Filtering	X	X	not really	X
Reverberation	-	-	-	-
Ringmodulation	X	X	-	X
Zerhackung (a)	-	-	-	adds clicks
Editing&permutation (b)	depends	depends	depends	depends
Synchronization (c)	depends	depends	depends	depends

a) Zerhackung is the multiplication of the sound with something what might best be described as a binary mask, thus creating a very rhythmic effect. It is achieved by recording onto a tape that has pieces of white-tape in it.
b) Editing and permutation means slicing the tape in pieces and reordering them.

c) Synchronization is the mixing together of separate layers.

Form:

“die Formkonstruktion von Terminus 1 beruht auf der Tatsache, daß Transformationen des Klangmaterials das seinerseits bereits aus Trnasformationen besteht, sich vom Ausgangsmaterial stets weiter entfernen” (Koenig, 1971 Äst. Prax. p 103)

english: “The form of Terminus 1 is based on the fact that transformations of the sound material which are themselves already transformations become less and less related to the original source material.”

Thus the process of making the material is also used as an inspiration to construct the large form.

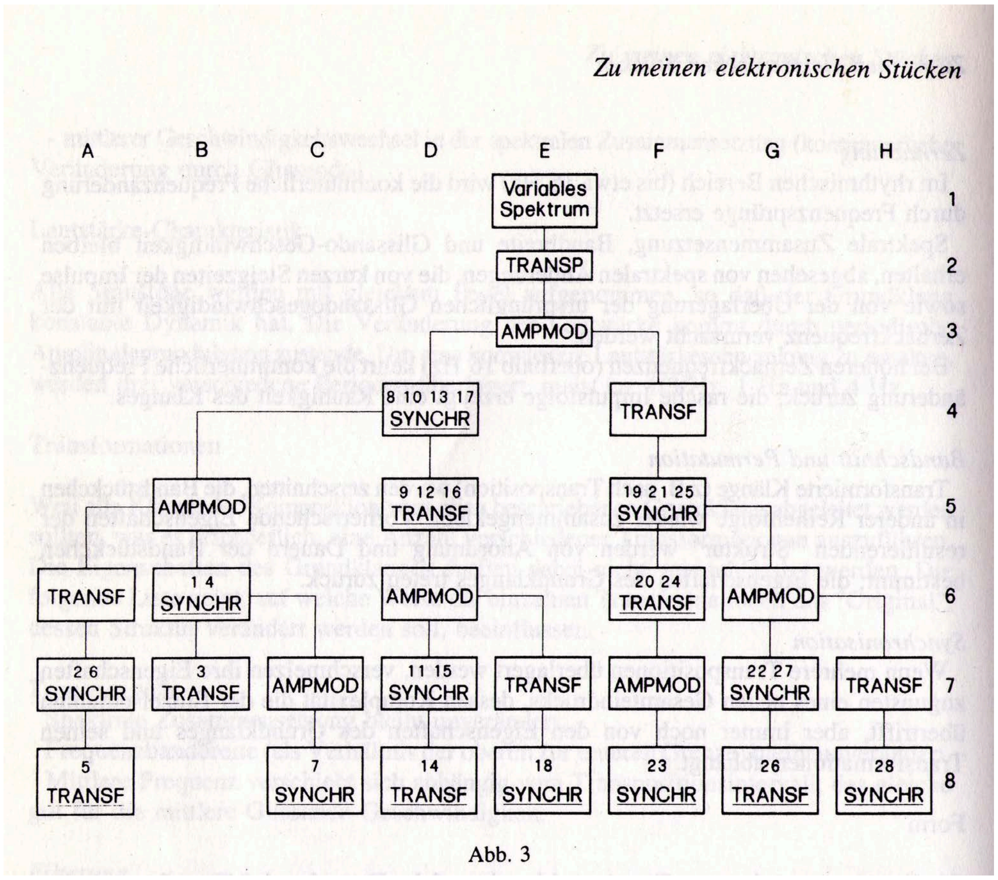
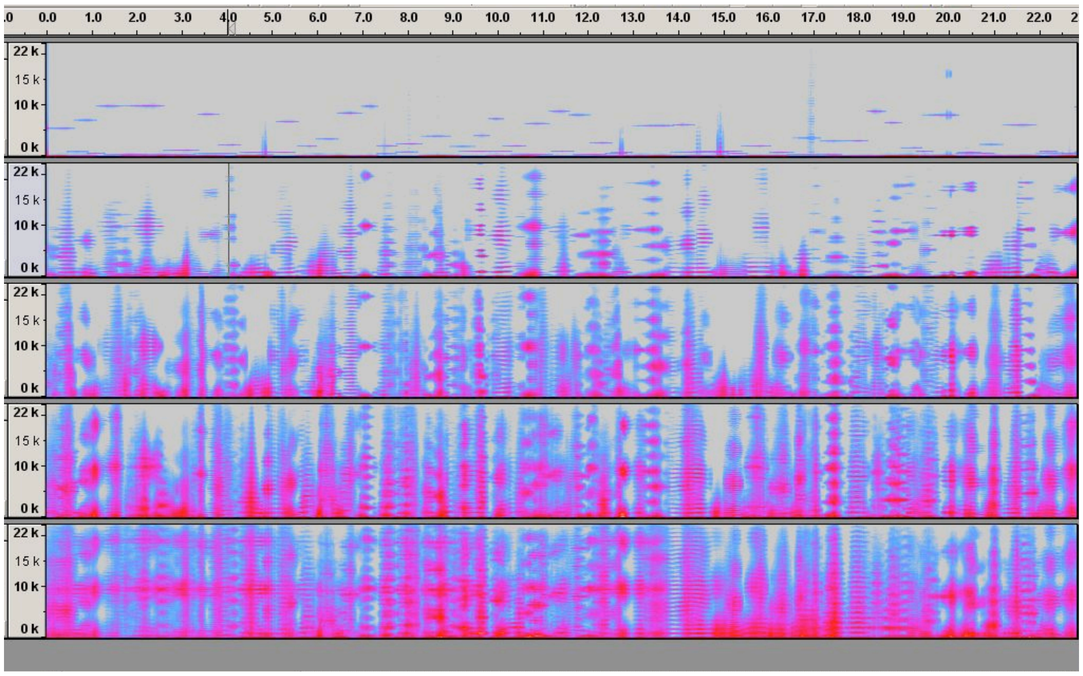


Image (8.1): The family tree of the material of Terminus, the numbers indicate the order in which the material is used in the final piece

The idea of transformations being transformed, formed the basis of my own piece. Instead of using all kind of different transformations, for the first generations I used only one: ringmodulation.


Spectra of the mother material

Working process of my own piece:

Apart from the editing, this piece was made completely in BEA5. The mother material was a simple dynamic sine-wave texture (no glissando's, but dynamic in amplitude) being ring modulated with itself recursively.
The patch contained 4 oscillators with random envelopes applied over the outputs. A new random frequency is triggered every time an envelope starts. This texture is then ring modulated with a previously recorded output.

As the image shows, after just a few recursive ring modulations the spectral aspect of the material doesn't change that much anymore. This is because the material becomes noisier every time it is ring modulated, and noise is (spectrally speaking) not very sensitive to ring modulation. This is also why I started with experimenting different ranges on the oscillators, when they where switched to a very low frequency range (1-10 Hz), I could increase the rhythmic aspect of the sound.
I also experimented with using other basic material (squarewaves,VOSIM,filtered noise), while still using the sine-waves for the ring-modulation. It turns out that these materials even sooner

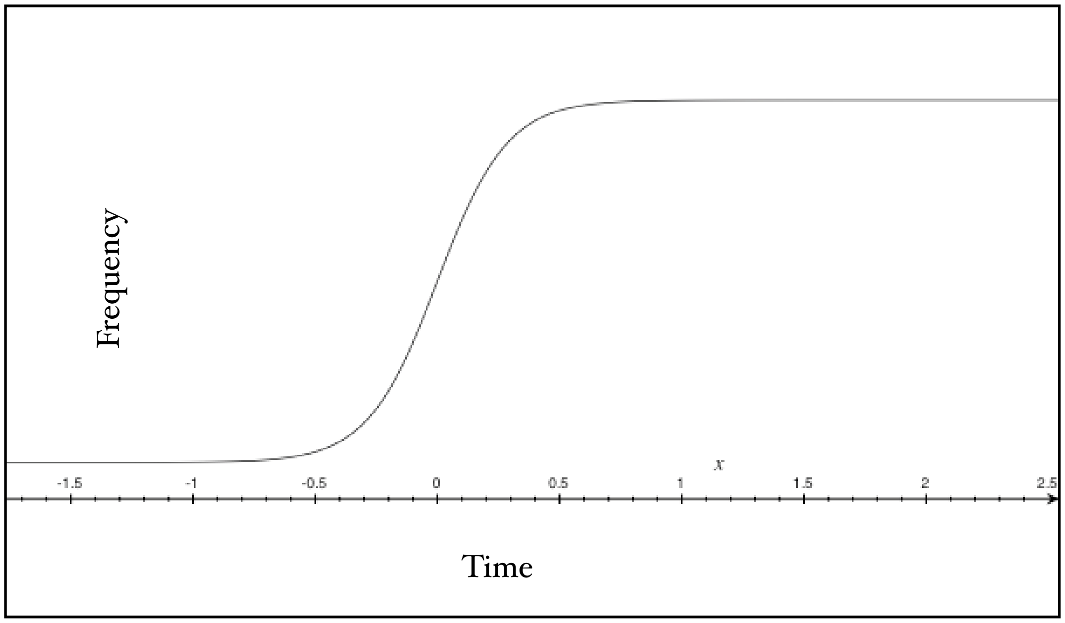
run towards a state were it doesn’t change that much any more every generation, still they provided possibilities to make contrasts in the final piece.

Some of the sounds were then put through a series of filtering, enveloping, and spatialisation patches. The final form of the piece was made partly intuitively, although sections of it are based on vertical travels through the ‘family tree’, thus starting with simple material developing towards complexity and through the other patches towards sparseness.

Reflective Surfaces

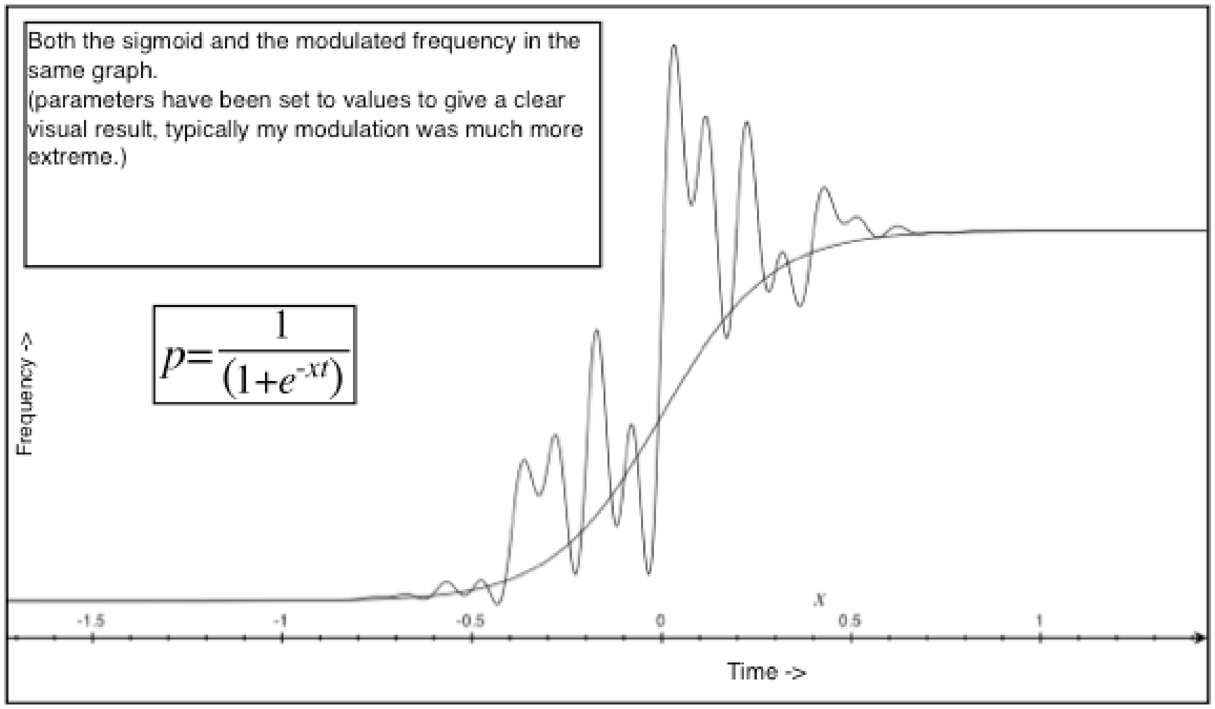
This piece was composed at the end of my second year. All sound material was created using Max/MSP. Although the complete process of the piece was not thoroughly documented there are still some important aspects I'd like to discuss here, because it influenced later pieces.

Fuzzy glissandi



Electronic glissandi have sometimes a tendency to sound unattractive, they (in my ears) sound a bit old fashioned or rather boring, especially when used on simple waveforms. It is something similar to the phenomena that LFO frequency modulation and theremin sounds can call up associations with 1950's science fiction movies. Still, I had the desire to try and connect sound events together (make a continues sound instead of separate events). And I started searching for ways to continuously change from one frequency to the next.
The first step in improving the sound of the glissandi was by smoothing the transition curve. Instead of a straight line between two points, I used a sigmoid, which has a nice curvy form. It changes slowly at the beginning and the end, and fastest in the middle.

This means that the onset and end of the glissando becomes much more subtle.
A lot of natural switching patterns in nature can be approximated with sigmoid curves and they are frequently used in neural nets.
A more important change was that I added FM-modulation at the point where the glissandi is located. The modulation depth was designed to have its maximum at the point where the speed of change is largest, hopefully masking the transition less typical.
In the end this results in a frequency that has a stable stage, becomes fuzzy, and suddenly becomes stable at a different frequency. It becomes a sort of theme in the final piece.

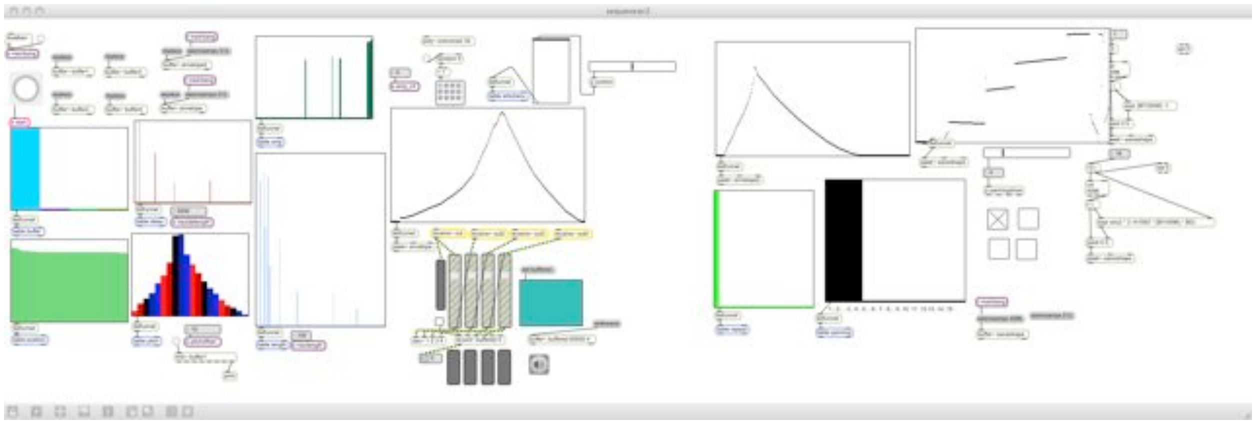


The Sequencer Patch

This patch is in a way very similar to the common granular synthesis patches that everybody uses. I almost completely used it on a time level much larger then 10 ms grains though.

It's parameters included:

- Duration
- Wait
- Amplitude
- Transposition
- Panning (4 channels binary)
- Read point in buffer
- Buffer number
- Envelope Shape



The parameters *duration,wait,amplitude,transposition,panning* and *read point*, where all controlled by histograms drawn by the user.

The panning was binary, with that I mean that there is no attempt to place sounds between speakers, it is either on a speaker or not. However it is possible for a sound to be played on two,three or four speakers simultaneously.

While I was using the software I discovered that it is much more interesting to make very spiky graphs for controlling the parameters then continuous ones. Which means that only certain values where selected, instead of ranges.

Another way I tried to make contrast with the randomness of time it created was by repeating certain ‘grains’. There was again a histogram to control the chance that an event might be repeated.

Reflection:

I was quite satisfied with this piece, although I did make a shorter edit later (it had a bit of a long exposition that didn’t really have a function). And although the histograms are quite primitive, they still allowed for quite different and expressive results.
There were however some problems with the phrasing of the music on the ‘phrase time level’ (let’s say 1 to 10 seconds): it had a taste of carelessness. This is especially apparent at the end, where I couldn’t really cut the material at a satisfying point, it just ends somewhere. The repetition parameter in the sequencer patch did help a little bit to solve this problem though, because it allowed to give a sequence a direction from uncorrelated events to something that had lots of repetitions in it. Repetition as an introducer of order and periodicity (to use the Koenig term) is something I explored in following pieces more deeply.
Another thing that maybe did not have such a clear place in the piece was the filtered sections, that gave a nice feeling of a space, without having a reverb taste to it.
Because the piece was sequenced using 4 groups of 4 channels it was easy to create a special 16 channel version for the WFS-speaker system. This opened the piece quite up, and made it in my opinion easier to follow separate events. It also showed, at least to me, that stationary sources on WFS can be just as interesting as sounds that move.

Jimmy

“I've been imitated so well I've heard people copy my mistakes.”

Jimmy Hendrix (unknown date)

This piece was the result of a research into (digital) audio feedback. The idea came to me after Paul Jeukendrup explained in one of his lessons that in live amplification situations, it is always best to have as few microphones (this also goes for speakers, by the way...) as possible, because if all your microphones are at different locations picking up the same sound sources, they will all have different time delays in their picked up signal. All these different time delays will result in different feedback frequencies, making it very hard to kill the feedback by simple equalization. Normally in a ‘one microphone-one speaker’ situation you can kill the main feedback frequency by filtering it (you will never totally get rid of it though because the resonances also take place at the harmonics). With many microphones the resonances become so dense that it is almost impossible to kill the feedback. I was fascinated by the idea though, because it sounds like the resulting feedback must be quite interesting and rich. I also wondered what would happen if you would move the microphones, would the system find different equilibria ? How would it switch from one state to the other ? Could the simple situation give rise to higher level sound structures ?

I decided to try and imitate the ‘many microphones situation’ in a Max/MSP patch. The original idea was to have a couple of virtual strings (Karplus-Strong like in nature) and have all off these connected to each other through delay lines (simulating the distance between them). The length of the delay lines was also modulated. Every ‘string’ had 7 outputs which connected it to the other strings. The amount of sound going to the other strings was controlled by random envelopes. So, it was not that all the ‘strings’ were fed to each other all the time. A separate patch allowed me to adjust the limits for the random envelopes. Between the output and the input was some switching mechanism that could act as a very extreme noise gate (0 ms attack, 0 ms release). For every string there was the option the ring modulate the output.

The final patch was creating very rich and diverse sounds ranging from soft violin like sounds to ear splitting distortion. True to it’s inspiration it was not very controllable, and it was difficult to predict what kind of sounds would result from a certain setting. If envelopes where long the system had interesting dynamic changes.

In the end a lot of separate material was created, and some of it cut up by the sequencer patch used for reflective surfaces. To create some developments in density material was recorded in 3 or 4 different versions. The density was increased every recording. By crossfading them from one to the other this gave a quite convincing continuous development from low density to high density. Of course I could have also automated this, but because of the instability of the patch this was a more practical solution.

Feedback for (a wet) bass-drum.

This piece was made during the 2-week installation workshop by Bennett. The original idea is closely related to the the Jimmy piece. It also deals with resonances. The goal was to use the resonances in the bass-drum itself to create an interesting sound structure.
The setup consisted of a (rather old) bass-drum, a microphone, a laptop running a Max/MSP patch and a small speaker located inside the bass-drum.

The first step in the creation process was measurement, I used a very slow sine-wave sweep and recorded it. I wrote down the resonances. The bass-drum had something like 11 clear resonances.

The second step was to design a Max patch that would keep the bass-drum in a constant state of feedback without going to much in distortion. My goal was to make the patch in such a way that the feedback did not just occur at the most obvious frequencies (lower resonances in the case of a bass-drum, but also at frequencies located at less obvious harmonics.

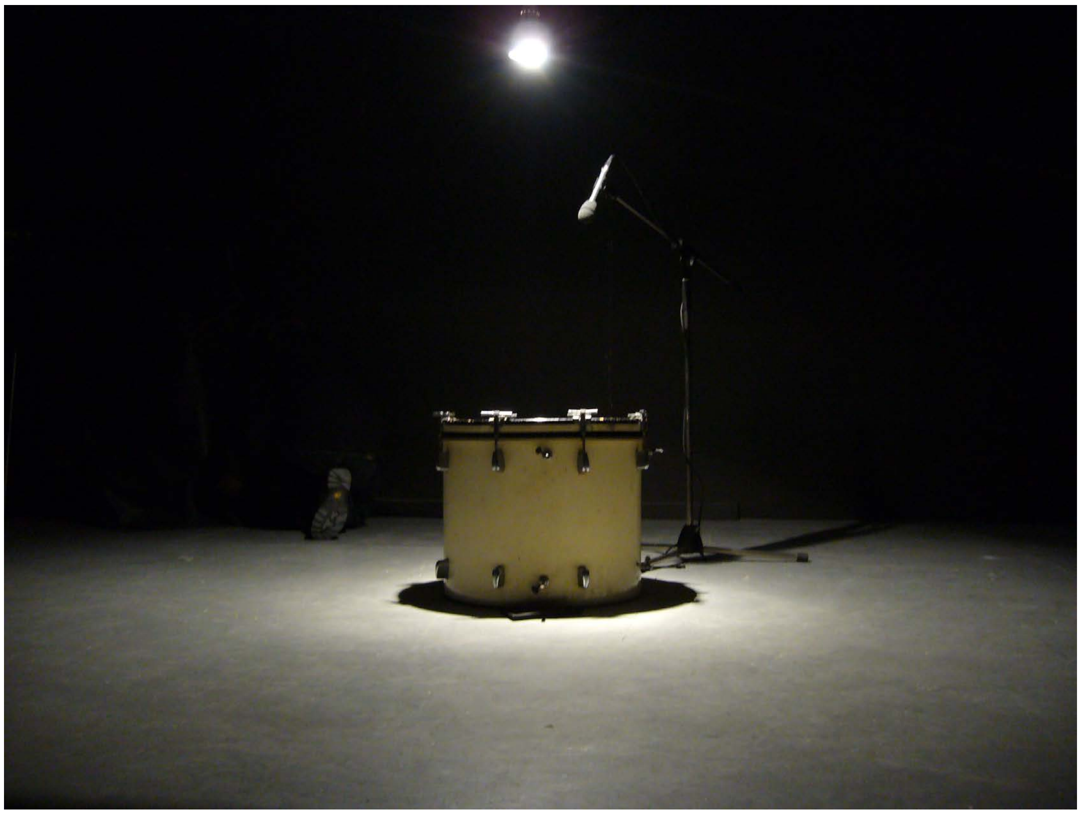
In the end the patch is very similar to a multi-band compressor. The sound coming from the microphone is filtered into bands (based on the measured resonances), and for each band the gain is determent. If the gain within a band gets above a certain threshold level, the gain for that band is reduced. Later I also added a sort of panic emergency valve, that turned the gain completely to 0.

Later I experimented with making the curve of the compressor hand drawn, this meant that I could make unconventional gain reactions, where the system would actually increase the gain of a band when it became softer, leading it to bounce back and become louder, only to be reduced etc... etc...

This gave the system a certain kind of instability that I found very appealing. Although it still appeared to stay in certain cycles gain wise (softer, louder, softer louder etc).
For the exhibition, I let the system run through different settings.

Reflection:
I think the whole setup was in general quite successful, especially because of the visual aspect with the chladni water patterns on top. I must say that I do regret the limited time that was available, I feel I was still discovering new things when I was working the last day. Also the 'going through a list of settings' approach, although effective, but certain settings could have done with much less time, while others where interesting enough to run for maybe even a few minutes.

A short video registration of this piece is available on youtube:
http://www.youtube.com/watch?v=uMWYm1fbh4k&feature=channel_page



Analog Repetition

BEA5 Max Sequencer

This piece was made as part of the MIDI to Voltage Control (from now on referred to as MIDI-to-VC) workshop of Kees Tazelaar. The MIDI-to-VC device allows you to control the analog equipment of BEA5 with software through MIDI. One important aspect of the assignment was that whatever the software did, it had to be something that would not be easy to do with the equipment in BEA5 itself.

My initial goal was to make a patch in Max that would be similar to the VT-VFG, a 12 step sequencer already present in the studio, but would extend the possibilities. The VT-VFG in BEA5 is a device that allows for 12 times 8 (+ 1) parameters to be hand set. The 9th parameter can also be used to set the time between steps.

Some other features of the VT-VFG:
- the 12 steps can be played in three types of order:
  normal, reverse, random.
- the amount of steps can be set between 1-12 (knob).
- there is the option to set the amount of repetition of this cycle between 0 - 255 or infinite (switches).
- the sequence can be stopped or started at any point by sending triggers (knob or externally).

One of the interesting features is that the VT-VFG can be triggered so fast that the stepwise function reaches the audible range, thus allowing it to be used as a kind of wavetable oscillator.
The audible range aspect of the VT-VFG was not implementable in Max because of constraints on the MIDI protocol. Instead I focused on the 'slow' functionality.

My Max Version:
Of course I wanted to add some unique functionality. One of the most important changes is instead of limiting myself to the 12 steps of the hardware device, this is changed into a list of 100 events. A separate part of the program then controls the way in which this list is read.
Another limitation of the VT-VFG is that the parameters of 'amount of steps' and 'number of repetitions' are set by knobs and can't be dynamically changed. Instead of setting these parameters by hand, I wanted this to be done by algorithms or stochastic processes. My ultimate goal was to extend the control of the composer to a slightly higher level. Instead of the control being limited to just the settings of parameters every step, I tried to deal with how these steps are then ordered in a dynamical way.

The event parameters:
Parameters for every event included:
- 4x Frequency
- Amplitude
- Entry-delay (this is the amount of time between the onset of events)
- Duration (expressed as a percentage of the entry-delay, it is always smaller then 100% in this version, because my BEA5 patch was monophonic.

Table Creation:
Entry-delays are a weighted choice from a set of 6 values (set by the user). The weights for the values are controllable by me through a sort of histogram. These weights, however, are normalized according to their length. With this I mean that when given equal chance, the long durations are taking an equal amount of time when compared with the short durations. If the amount of occurrences would be equal, the long durations would completely overshadow the short ones. This however does give some difficulties when the difference in length between the shortest duration and the longest is very big, because in that case the chance for a long duration actually being chosen gets very small, and it is likely not to appear at all in the 100 events created. Values have to be chosen with care !
Limitation of possible values is very important for me because it allows better control over what possible rhythms. If for example the durations where just chosen from a range between 2 values,

the pattern resulting would contain so much information that it becomes noise like, the structure is no longer perceived.

The frequency values where chosen in a slightly different way. The frequency parameter is broken up in octave and pitch from a scale. Of course, I don't have fantasies about the accuracy of the voltage being concerted to exactly the pitch in my patch. However even without that accuracy, the splitting of frequency in Octave and Pitch on a scale is convenient because it allows to make some more subtle differences.

Parameters for ordering the events:

The events are then in another part of the patch ordered in what can be best described as a loop. The parameters of the loop are: start index, loop size, repetition of loop. All of these parameters are then controlled by weighted choices again provided by the user (i.e. the user draws histograms). The creation of these loop parameters is real-time, so it is not first stored as a list. I also choose not to normalize the weights for loop size. This means that if there is a equal chance on long loops and short loops the long ones do take much more time than short ones, but that is just a feature I accept. In the beginning all parameters were always connected to each other.

Modulations of parameters:

While testing my early version of the patch, I very soon got the feeling that straight repetition of a sequence of events was not satisfying me enough, it was too machine like for me. So I built in an option to do modulation on the parameters. This modulation is not just random deviation, it is based on what stage the loop is in. All the parameters described earlier can be modulated by a certain ratio. At the beginning of a loop a new ratio value is chosen between two boundaries provided by the user. Because every time the loop is repeated the ratio gets multiplied with itself, the result is that the values change exponentially. When applied to durations or frequency, it gives the sound structures a feeling of curvature.

BEA5 Patch

The analog patch itself was extremely basic: just 4 oscillators that go through two multipliers, one to control the amplitude and the other the envelope. The envelope is rectangular, but made slightly less harsh by using the rise decay filters.
Just to exaggerate the amplitude difference I sent the outputs of the oscillators through the harmonic distortion, this meant that the loud events contained more harmonics then the soft ones.
I also made some more experimental patches that instead of summing up the outputs of the oscillators, multiplied them with each other (AM-modulation).

Sequencing

All of the material used lots of transposition and different ways of combining. Sadly I can't go into much more detail then that, the original pro-tools session has been lost.

PART II : Algorithmic Composition

On Algorithmic Composition

Composing music with the help of algorithms is not something new, it has clearly been around in western-europe for centuries and probably much longer depending on your definitions. It is therefore useful to define what I think an algorithm actually is. Generally an algorithm is described as a finite sequence of instructions that solves a given problem step-by-step. It is expected to always yield an answer or at least a close approximation in the case of calculations. The goal of an algorithm is breaking a complex problem (like calculating the orbit of the moon or finding the square root of 2) down in smaller and simple steps. A good algorithm should make the steps so simple that something quite stupid like a computer can even do it.
However I would like to highlight some aspects of the concept further: first of all the concept of algorithm is not bound to computers or any other form of calculation machine. For example: if you want to divide 32264 by 537 by using long division, it does not matter for the algorithm if you use a calculator or do it with pen and paper or even with sticks and stones, when you are precise about the method, the result should still be the same (though one might be faster than the other). However it's simplicity, does not mean that an algorithm is not able to yield results that were not expected by the user that applies it.

To give the most obvious example of how powerful an algorithm can be I want to make a small sidestep to an algorithm that yielded complexity that is even more complex then music: evolution. In his book "Darwin's dangerous idea" , philosopher Daniel Dennet tries to explain why the Darwinist explanation of biodiversity is so shocking: it actually implies that all complexity we find in nature is not here because of some design but fundamentally the result of an utterly mindless process which we could consider an algorithm. And since our capability to think is probably also a result of evolution, one might even argue that music is too, through many intermediate steps, the result of an algorithm: evolution.
This is of course quite a reductionist claim, but what is wrong about reductionism ?
"The central image is of somebody claiming that one science "reduces" to another: that chemistry reduces to physics, that biology reduces to chemistry, that the social sciences reduce to biol-

ogy for instance. The problem is that there are both bland readings and preposterous reading of any such claim. According to bland readings, it is possible (and desirable) to unify chemistry and physics, biology and chemistry, and, yes, even the social sciences and biology. After all, societies are composed of human beings, who as mammals, must fall under the principles of biology that cover all mammals. Mammals in turn are composed of molecules, which must obey the laws of chemistry which in turn must answer to the regularities of the underlying physics. No sane scientist disputes this bland reading; the assembled Justices of the Supreme Court are as bound by the the law of gravity as is any avalanche, because they are, in the end, also a collection of physical objects. According to the preposterous readings, reductionists want to abandon the principles, theories, vocabulary, laws of the higher level sciences, in favor of the lower level terms. A reductionist dream, on such a preposterous reading might be to write a "A Comparison of Keats and Shelley from the Molecular Point of view" or "the Role of Oxygen Atoms in Supply-Side Economics," or " Explaining the decisions of the Rehnquist Court in Terms of Entropy Fluctuations.". Probably nobody is a reductionist in the preposterous sense..." (Dennet, 1995, p81)

When I read this passage one composer (that I respect highly, I must add) immediately came to my mind: Xenakis. He often seemed to have flirted with the idea of reducing music to it's axioms.

"Starting from certain premises we should be able to construct the most general musical edifice in which the utterances of Bach, Beethoven, or Schönberg, for example, would be unique realizations of a gigantic virtuality, rendered possible by this axiomatic removal and reconstructions. Xenakis 1992, page 207 "

Before we can proceed and ask the question wether musical composition can be done by algorithms, the first question to answer would be, what do we mean with composition ? In classical instrumental music one could say composition is the process of combining single notes into motives, build phrases, themes etc. all the way up to a large form. A good example of this idea is the book Fundamentals of Musical Composition (A. Schoenberg 1960). It describes an (almost) algorithmic process for building up a 19th century style composition from small elements (motive, phrase, theme etc.) towards a larger form. His opinion of form seem very rationalist:

"The chief requirements for the creation of a comprehensible form are logic and coherence. The presentation, development and inter-connexion of ideas must be based on relationship."

however:

"A composer does not, of course, add bit by bit, as a child does in building with wooden blocks. He conceives an entire composition as a spontaneous vision."

"... No beginner is capable of envisaging a composition in its entirety; hence he must proceed gradually, from the simple towards the more complex". Schoenberg 1967, (p. 1-2)

In his method he tries to break the process down to it's fundamentals as much as possible, so that almost no deep knowledge or experience is required.
" The standard textbook analogy notes that algorithms are recipes of sorts, designed to be followed by novices cooks. A recipe book written for great chefs might include the phrase "Poach the fish in a suitable wine until almost done," but an algorithm for the same process might begin, "Choose a white wine that says 'dry' on the label, take a corks-screw and open the bottle; pour an inch of wine in the bottom of a pan; turn the burner under the pan high...." (Dennet 1995,p51).

However since we are dealing with electronic music, the process of composition extends itself from just creating a form from elements, to the creation and design of the sounds themselves.

"In der frühen elektronischen musik legte man im Kölner Studio großen Wert auf die festellung daß nicht nur das Werk, sodern auch shon jeder Einzelklang "komponiert" werden müsse ; man verstand darunter eine arbeitersweise, mit der die Form des Werkes und die Form des klanges verküpft werden sollten... (p192 Äst. Praxis band 3).

This change from note towards a sound structure is very important. The normal hierarchy (of notes, phrases, themes, parts etc...) mentioned before is challenged, because the sound structure is not bound to the lowest level. It's duration can be much longer then a typical note, in a way you could even consider the whole piece of music as a single complexly modulated sound.

This shows that electronic music always has to deal with the lowest level, because it has to create the music from it's simplest part : the sound itself.

So the question now becomes, why do we still want to do it, why do we want to teach the machine to produce music ?
Koenig states (Äst. Praxis 3 p.197) that the computer can be used for three purposes:
1) For the composition of smaller parts or form elements
2) To experiment with a model of composition (a simplification) , that provides the composer with a blueprint that the composer has to work out in detail himself.
3) For the composition of a single work, where the program more presents itself as a score generator than a solver of compositional problems.

In the experience I had with my own software, aspects of all three of these strategies have turned up.

Maybe the most obvious reason I want to use software, is simply because the amount of calculation gets very large (especially when you're going to use stochastic processes). This is also a result of that the composer now has to deal with very low level sound, an area where it's interesting to let the computer take over some of the work.

One last thing I want to note is that even though algorithms may be less intelligent then composers, they can still sometimes come up with things, the composer himself would never have come up otherwise. As the mathematician Stanislaw Ulam once put it:
" When I was a boy I felt the role of rhyme in poetry was to compel one to find the unobvious because of the necessity of finding a word which rhymes. This forces novel associations and almost guarantees deviations from routine chains or trains of thought. It becomes paradoxically a sort of automatic mechanism of originality." (Ulam 1976, p 180)

So, to summarize, I don't expect my software to produce complete pieces, that is because the concepts and rules on a low level (the ones that my MacBook understands) do not translate very well to higher level problems I encounter. I use my software for what it's good at, creating textures that are in itself relatively simple. The large form of my pieces are not generated by the software, but decided for by myself, because for now that's much more efficient! However in making those decisions I always try to stay connected to the material. It is the most important inspiration for the form.

A little History:

I will now proceed with a short description of the programs of Hiller & Isaacson (Illiac Suite), Xenakis (ST-algorithm) and then a more deep study of Project 1 and 2.

Hiller and Isaacson:

Although experiments have been conducted before the work of Hiller and Isaacson, I would like to start there because their work stands out because what they have done has been influential on later work. Several experiments using quite different strategies were combined in a piece called 'Iliac Suite' for string quintet (finished in 1957).
Experiment one & two were an implementation of strict counterpoint. The rules where taken from the famous 18th century book by Johann Joseph Fux: Gradus ad Parnassum. The algorithm consisted of 2 simple steps:

1. Generate a new random pitch
2. Check if the generated pitch is allowed by the rules.

- Yes : add pitch to sequence
- No : go back to step 1 (if this step happens to often the sequence is deleted and a new sequence is tried.

The rules consisted of things like:
- Melodic restrictions:
    - range (within an octave)
    - forbidden intervals
    - after a large interval, following intervals should be smaller.
    - etc.
- Five rules controlled the vertical intervals
- "Combined rules" constraining parallel motion and voice movement.

When you listen to the first two sections, the first thing I noticed is that it 'fails' when it comes to giving direction, it doesn't seem to remember what it has played before.
After this, Hiller and Isaacson implemented more contemporary musical idioms like chromaticism and serialist techniques in experiment 3.

One must realize that the results from experiment 1,2 and 3 that are part of the Iliac suite, are edited outputs. Also dynamics and expression marks were added later by the composer. (Ariza, 2005 p49).
The final Experiment Four had as a goal to find a technique more based on the possibilities of the computer then of trying to enslave the computer 'external' musical ideas. They decided that instead of making a huge amount of complex rules, to try a more fundamental approach using Markov chains. Part 4 of the Iliac suite is the only part that is not a combination of outputs, although tempo, meter and dynamic indications were added later by hand. I must say that I find this part the most effective.

Xenakis' SMP software

The algorithms that form this program are not new to Xenakis, he had used the stochastic approach before in *Achorripsis* (1957). The choice of stochastic algorithms is very likely a result from his dissatisfaction with serial techniques, in his 1966 article he said about it:
"... linear polyphony is destroyed by its own present complexity. One hears in reality only aggregations of notes at various registers. The enormous complexity makes it impossible for the ear to follow tangles lines, and its macroscopic effect is that of an unreasoned and fortuitous dispersion of sounds throughout the entire frequency spectrum" (Xenakis, 1966, p. 11)

1	The work consists of a succession of sequences or movements each A seconds long.
2	Definition of the mean density of the sounds during A
3	Composition Q of the orchestra (from r classes of timbres) during sequence A
4	Definition of the moment of occurrence of the sound N within the sequence A
5	Attribution to the above sound of an instrument belonging to orchestra Q.
6	Attribution of a pitch as a function of the instrument.
7	Attribution of a glissando speed if class r is characterized as a glissando
8	Attribution of a duration x to the sounds emitted
9	Attribution of dynamic forms to the sounds emitted
10	The same operations are begun again for each sound of the cluster N a
11	Recalculations of the same sort are made for the other sequences.

J. Harley (2004) but originally from Formalized Music, 1992)

It might be interesting to compare this algorithm with Xenakis description of how he describes 8 fundamental phases of a musical work:

1	initial conceptions
2	definition of the sonic entities
3	definitions of the transformations
4	microcomposition (choice and detailed fixing of the functional or stochastic relations of the elements of 2)
5	Sequential programming of 3 and 4 (the schema and pattern of the work)
6	Implementation of calculations, verifications, feedbacks, and modifications of 5
7	Final symbolic result (traditional notation etc.)
8	Sonic realization (performance, playback, etc.)

J. Harley 2004, but originally from Formalized Music, 1992)

Apart from steps 1, 7 and 8 there seem to me a lot of similarities, that suggest that he wanted to generalize the compositional process to be reproduced as much as possible in the program. However it must be noted that in the final pieces made with this program, he often changed an added things in transcribing the date to the score.

As has been noted by James Harley (2004 p 30) *"... Atrées shows a concern for sonority and performance issues that go beyond the premise of the compositional algorithm. There are passages, such as the JW32 section of the third movement, where the sustained pitches are varied by the periodic intrusion of tremelo or flutter-tongue, or shifts between sul ponticello and sul tasto. The tradeoffs of timbral or dynamic shifts from one instrument to another creates a kind of hocketing dialogue as the spotlight of attention shifts back and forth. These passages were not programmed, but added by Xenakis in the process of transcribing and evaluating the computer data."*

P R O J E C T 1

Stockhausen:

So serial thinking is something that's come into our consciousness and will be there forever: it's relativity and nothing else. It just says: Use all the components of any given number of elements, don't leave out individual elements, use them all with equal importance and try to find an equidistant scale so that certain steps are no larger than others. It's a spiritual and democratic attitude toward the world." (Cott 1973, 101)

I put the quote from Stockhausen, because I think Project 1 has a very strong connection with serial thinking, however it also breaks or extends some of it's ideas. The quote also points out the idealistic aspect of serialism.

History:

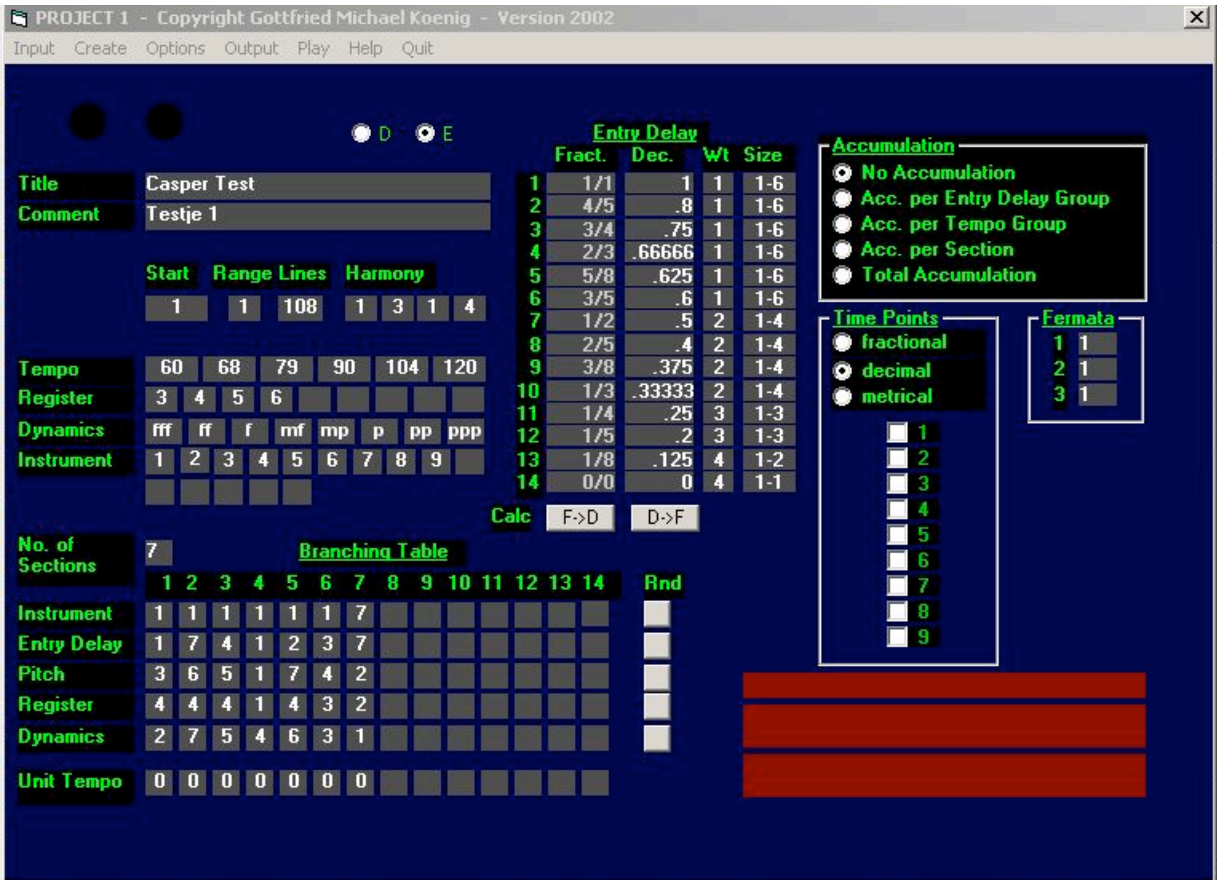
After an archeologist friend had suggested that the score of Essay looked to him like something generated by a computer, Koenig started following a programming course at Bonn University. (Genesis of form,Koenig, 1987, Interface vol. 16). In an interview with Curtis Roads, answering the question when he made his first musical software he replied:
"Von Anfang an. Im Programmierkursus wurden die praktischen Übungen gebieten entnommen, die nichts mit Musik zu tun hatten. Als ich erwähnte, ich würde gern einige Übungen auf musikalischen Gebiet machen, war der Dozent damit sofort einverstanden. Ich begann mit kleinen Programmen für die Erstellung von Zwölftonreihen, für Akkordenfolgen und ähnliches. Nach mehreren Experimenten machte ich mich an ein größeres Komponierprogramm, das später Projekt 1 heißen sollte."[1] (Koenig, 1978, ästetische praxis)

The first version (finished around 1966) was only intended for his own experiments, because in his opinion the amount of control over data and compositional rules were too limited for other

---

[1] English Translation: From the very beginning. In the programming course, the practical exercises were taken from areas that are not related to music. When I proposed that I would like to do some exercises dealing with musical issues , the lecturer immediately agreed. I started with small programs for the creation of twelve-tone rows and for chord sequences and the like. After several experiments, I wrote a larger program, that would later be called Project 1.

people to be interested. However, since then versions of Project 1 with more freedom have been written. The last version runs on Windows and has a graphical input window.



PROJECT 1 - Copyright Gottfried Michael Koenig - Version 2002

D [✓ E]

Title: Casper Test
Comment: Testje 1

Start	Range Lines		Harmony			
1	1	108	1	3	1	4

Tempo	60	68	79	90	104	120			
Register	3	4	5	6					
Dynamics	fff	ff	f	mf	mp	p	pp	ppp	
Instrument	1	2	3	4	5	6	7	8	9

Entry Delay

	Fract.	Dec.	Wt	Size
1	1/1	1	1	1-6
2	4/5	.8	1	1-6
3	3/4	.75	1	1-6
4	2/3	.66666	1	1-6
5	5/8	.625	1	1-6
6	3/5	.6	1	1-6
7	1/2	.5	2	1-4
8	2/5	.4	2	1-4
9	3/8	.375	2	1-4
10	1/3	.33333	2	1-4
11	1/4	.25	3	1-3
12	1/5	.2	3	1-3
13	1/8	.125	4	1-2
14	0/0	0	4	1-1

Accumulation
[✓ No Accumulation]
Acc. per Entry Delay Group
Acc. per Tempo Group
Acc. per Section
Total Accumulation

Time Points
fractional
[✓ decimal]
metrical
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9

Fermata	
1	1
2	1
3	1

No. of Sections: 7

Branching Table

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Rnd
Instrument	1	1	1	1	1	1	7								☐
Entry Delay	1	7	4	1	2	3	7								☐
Pitch	3	6	5	1	7	4	2								☐
Register	4	4	4	1	4	3	2								☐
Dynamics	2	7	5	4	6	3	1								☐
Unit Tempo	0	0	0	0	0	0	0								

Parameters:

The parameters for Project 1 are:
Instrument
Rhythm (through the concept of entry delay(*))
Pitch
Register
Dynamics
* = Entry Delay is the amount of time between the start time of two events.

For the parameters instrument, entry delay, dynamics and tempo the composer fills in lists to give possible values. The program will then order these values, and form chords

The harmony is not constructed explicitly but is controlled by the specification of two three-tone groups. Each group is constructed from two intervals provided by the composer. These three-tone-groups are then transposed 3 times in such a way that a complete 12 tone row is obtained. If you pick typical intervals, you can hear these pitch patterns quite clearly. However the choice of intervals is restricted to combinations that allow the construction of a 12 tone row. Some combinations prohibit the construction of a 12 tone row, here, the the software may invert the order of the sequence to achieve the 12 tone row.

In Project 1, the parameters of an event are normally treated completely separate. This comes of course from serialist thinking, because it assumes that parameters should be as independent from each other. In the manual Koenig says about it:

"With one exception, parameters are not interrelated; that is because serial theory's point of departure is that every element of a parameter can be linked with every element of another parameter, at any rate in an average range of values."

"...The serial axiom could thus be expanded into the statement: every well-formed sequence of one parameter will let itself be placed beside every well-formed sequence of another parameter in a constructive context: by complementing, supplementing, accentuating - at least 'not hiding' each other."
Koenig, Project 1 HelpCollection.pdf p. 8

The one exception mentioned in the upper statement is the link between entry-delay and chord-size. However we will see that in Project 2 Koenig, does seek 'interaction' in the selection of different parameters.

Compositional Rules (Order in parameters):

Something that keeps coming back in Koenig's work and also in his writing, is the idea of periodicity and aperiodicity. A sine wave is an example of a very regular or periodic source, white noise is on the other end of the spectrum. In Project 1 this contrast also plays a very important role.

The program has 7 steps in constructing the order in parameters, ranging from complete aperiodicity to periodicity. Because he did not find a satisfying way of going smoothly from one to the other, there is an intermediate step that is a combination of the two processes.
For creating aperiodicity Koenig uses serial techniques. Periodicity is constructed using limited repetition.
the Aperiodic steps:
In step 1: there are no repetitions allowed for the whole sequence.
example:

4376521

7653124

etc..

step 2: there are no repetitions allowed for part of the sequence.
example:

1345 4672 7651 2574

7342 1532 2564 2137

step 3: there are no repetitions allowed for an even smaller part of the sequence.

12 56 17 62 34

step 4: this is an intermediate step (see explanation below)

the Periodic steps :
step 5: repetition is very limited.

11 2 33 4 1 3 5 4 55 7 11 22 33

step 6: the repetition is limited to a certain value.

1111 22 3333 666 55 33333 22 1 77 777

step 7: the repetition is decided for between 1 and a maximum.

3333 44 22 1 7777777 2222222222 333 111111111 44444 333 222 111

Note: the examples are not representative for the all the output of the program, because for each parameters the specific amount of repetition (or the size for the repetition prohibition) is specified by an internal formula. Also see section ##

Step 4 is actually the most interesting one, it is made up out of two elements:

Set and Balance

Set and Balance are both combinations from series generated with step 3 and 5.

- First it is randomly decided how much S series will be generated.
- For every S series it is then randomly decided wether it is an Aperiodic (probably step 3) or a Periodic (step 5) style series.
- A size of the series is chosen randomly as well.
- Because it might be that either the Aperiodic series or the Periodic series dominate in number, a set Balance is created, which has exactly the same number of series as Set, but an inverse number Aperiodic and Periodic series. Then the series' order are shuffled randomly. In the end when Set and Balance are combined, there are an equal amount of Periodic and Aperiodic series. The result has thus a dialectic character, a mostly periodic series is followed by an aperiodic sequence etc. This aspect sets it quite apart from the other steps, because it creates structures that are larger in scope.

S=Set B=Balance

P=Periodic(step 5) A=Aperiodic (step 3)

S 63(A)	B 55555(P)
S 435(A) 644(A) 53(P)	B 66(P) 354(A) 555(P)

S 46465(A) 3456 35(A) 663(P) 464(A) 543(A) 655(P) 3(A) 4656 (P)

B 4444(P) 33(P) 6(A) 5(A) 333(P) 451(A) 4444(P) 3333 (P)

The large form can be controlled through something called the branching table. In the branching table the composer can select per section which rule has to be used for each parameter.

As mentioned before: Koenig designed for every parameter a slightly different formula's to calculate the size of the repetition check in step 1 to 3 (or the minimum and maximum amount of repetition in step 5 to 7). He refers to them as compositional rules.

An example :

the amount of repetitions in EntryDelay series in step 5 is calculated by: 3/4 * number of values provided by composer.

Koenig puts emphasis on the importance of interpreting the output of the score. He did not expect composers to accept the output literally. Durations for example are not specified at all by the program, it is a choice left to the composer himself. Also the dynamics, register parameters are explicitly kept symbolic, to allow for flexible transcription. In the manual Koenig suggests a method of working in which the composer has to interpret the data according to his original intentions. Concerning how to interpret chords he says in the manual:

"Dodecaphony stood model for Project 1 inasmuch as it is based on three-tone groups which, together with their transpositions, form complete twelve-tone series. Twelve-tone series are however not played in retrograde or backwards, but merely transposed. Nor do any motifs or themes derive from twelve-tone series. On the other hand, the constant recurrence of the three-tone groups prescribed by the composer may supply the composer during 'interpretation' with material for motivic combinations."

There are two things I find especially interesting about Project 1:
First of all I think for the limited amount of input data it is still able to create quite different results. I like that the most important parameter for the large form is based on periodicity, or how things are repeated or allowed to repeat.
I also think a lot of it's power comes from the freedom of interpretation of the output. It's power is that it's not too explicit in it's output.
Finally I find step 4 of the order generation principles interesting because it is a first step toward a higher level control of generating order within groups of events. It is interesting how it uses chance, but at the same time manages to retain symmetry.
The control through the branching table puts the composer in a very different position, instead of being in control of single events, the composers controls the periodicity or aperiodicity, a concept that is closer to what the listener might experience. It seems that on that point Xenakis and Koenig agree with each other (that high level characteristics are more important then low level ones)
Even though the compositional strategy of the program is quite hard-wired, still a wide range of possibilities are possible.

P R O J E C T 2 :

Immediately after Koenig had finished Project 1 and composed two pieces with it, he felt that there where some 'peculiarities' that he wanted to get rid of:

"*first of all there was the program's restriction to only a few parameters which - apart from the odd exception - were independent of one another. The values of each parameter therefore had to be distributed in time in such a way as to remain permeable for the other parameters; no parameter could bear the sole responsibility for the formal progression.*"

...

" *In the second place, there was the pre-established formal progression. Although this catered for the linking of variants, the variants had to satisfy a criterion of completeness : since the range between irregular and more predictable events was divided into 7 steps, the overall form of a "project 1" piece invariably consisted of 7 sections, so as to permit each parameter to display itself once in each of its 7 guises*"

[Koenig, Genesis of Form.p174].

Actually the last point maybe less relevant, because in the latest version of project 1, there is no longer a limitation on the amount of segments, and also the compositional rule (step 1-7) is open for the composer to influence.

Because the manual of project 2 exists of 160 pages, I will not be able to give a complete overview of all the functionality of this program. I'll try to focus on the basic structure and some specific details that will become relevant in explaining my own program later.

generally:

- The program is fed musical data such as possible instruments, durations, dynamics play methods etc..
- The composer provides the program with information about what methods to use to order and combine the data. (similar in function to the compositional rules in project 1, however implemented in quite a different way)

The list-table-ensemble principle:

Musical data is entered as a *list*. The list is constructed by the composer, there are no generation functions of any sort. The amount of values and the range are completely free (except hardware constrictions of course).

Let's take a list containing durations as an example:

Position	1	2	3	4	5	6
Value	0.1	0.2	0.5	1	4	8

Then the user specifies a *table* that enables the user to form certain *groups* within the list, for example:

index	Index Numbers from List	index	Represented Durations:
1	1 2 3 4 5 6	1	0.1 0.2 0.5 1.0 4.0 8.0
2	1 4 5	2	0.1 1.0 4.0
3	1 1 2 6	3	0.1 0.1 0.2 8.0
4	1 2 3	4	0.1 0.2 0.5
5	4 5 6	5	1.0 4.0 8.0

(37.1)
group 1: complete list, group 2: certain values selected, group 3: a selection with repetitions etc...

The control over how the data will be used in the composition is of course much greater then it was in Project 1, where always all the values were available in every section. Also note that the first two steps (the generation of the list and forming of groups in the table) are made completely by the composer, and not with any aleatoric processes. It seems to me that Koenig wanted to limit the randomness.
The next step however is the formation of an ensemble, and here selection becomes more automated. The formation of an ensemble happens through the combination of 1 or more groups. The composer can use either ALEA, SERIES or specify himself with SEQUENCE. This may seem a

little double because if you look at the table(37.1), the combination between group 4 and 5 is exactly equal to group 1. Actually, any combinations of groups, could have been constructed already in the table. The reason for the concept of the ensemble lies therefore elsewhere. First of all it allows for an automated selection from the groups (with the use of ALEA or SERIES). Secondly ensemble*s* play a role in how the program constructs what Koenig calls ‘vertical density’ or Layers. I will explain this further in the topic combination/union.

The final score is constructed only from the data that is in the ensemble (so not directly from a table or list). For each of the parameters the composer must specify what selection/order principles he uses:

Selection principles:

ALEA:

a random choice between a and b without repetition check.
1 4 2 3 3 7 6 6 2 1 3 1 5 4 etc...

SERIES:

a random choice with repetition check. No element can be chosen again until all have been used, so early repetitions like in Project 1 are not implemented in Project 2. I think Koenig expected the composer to construct his own way of generating periodicity aperiodicity through the use of the table.

1 3 4 5 2 7 6   6 5 7 4 2 1 3   3 1 4 5 7 6 2 etc.

RATIO:

a serial 'weighted choice' that works by selecting from a supply. For each element the user can specify the amount of times an element can be selected before it is 'used up'. For example entries A B C with the weights 1 3 4 would stand for this:

```
supply:              A B B B C C C C
possible outcomes:    B C A C C B B C
                      C C B B A C C B
                      B C B C B C C A
```

Note: if all weights are set to 1, its behavior is exactly the same as SERIES.

GROUP:

Produces repetitions. The repetition is controlled through two parameters: what element, and the amount repetitions.

The user decides for both wether these decisions are made by ALEA or SERIES.

An example where element is controlled by SERIES and size by ALEA:

1 1 1 4 4 3 3 3 3 5 7 7 7 7 7 6 2 2 2   4 4 4 2 2 1 3 3 5 6 6 6 6 7 7 7 etc.

TENDENCY:

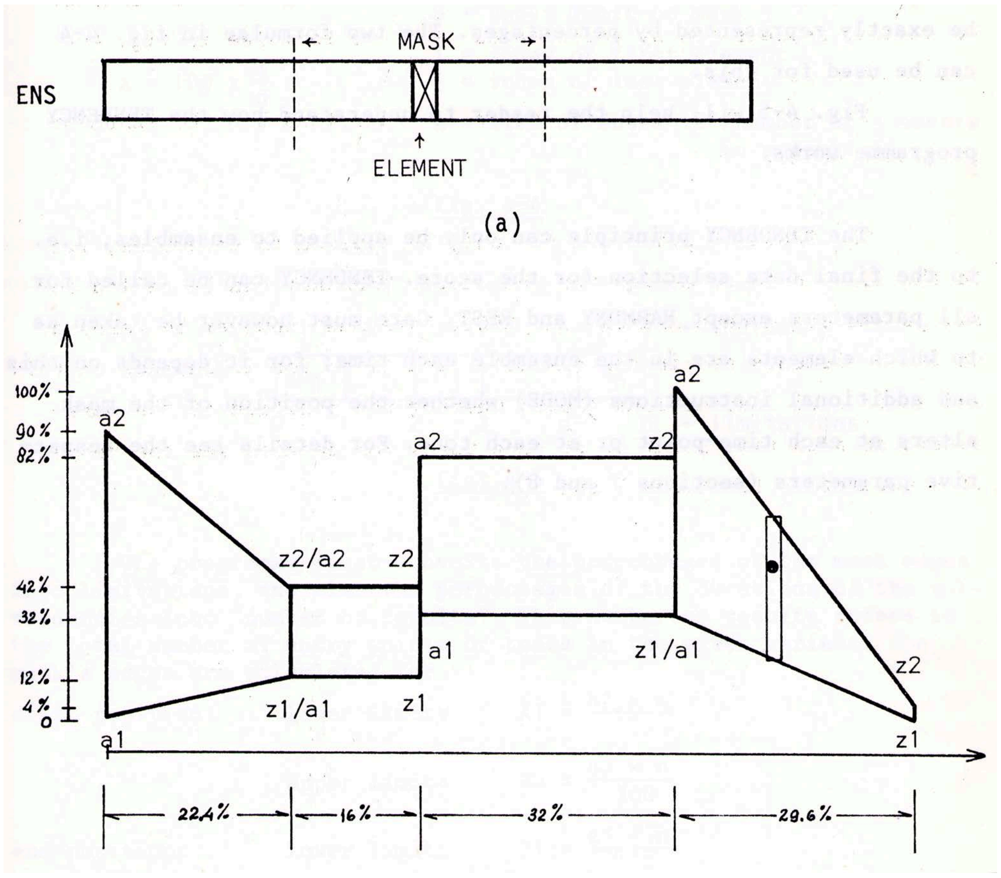


Image from project 2 manual

Tendency allows for ALEA choices within changing boundaries. Masks are defined by specifying the limits of sub-tendency's. Because the amount of total events and the size of the ensemble can differ per variant, all the values are expressed in percentages. It is important to note that these tendency's are thus not used to select completely random between two boundaries, but actually guide a choice in an ensemble that is itself already a certain selection from the original list.

SEQUENCE(abr. SEQ):

This enables the composer to make the choice by hand. Can only be used when the size of the input data is known precisely. This might not be the case if size is decided by ALEA or SERIES

Harmony:

Although the other parameters are all controlled through the list-table-ensemble method, harmony is an exception. It is notable that just as in Project 1, it gets a separate treatment. Harmony can be constructed in three ways:

- CHORD:

the composer provides a table with chords (interval values should fall within an octave): which will be sorted using a selection principle. If a chord contains a interval 0, it will not be transposed.

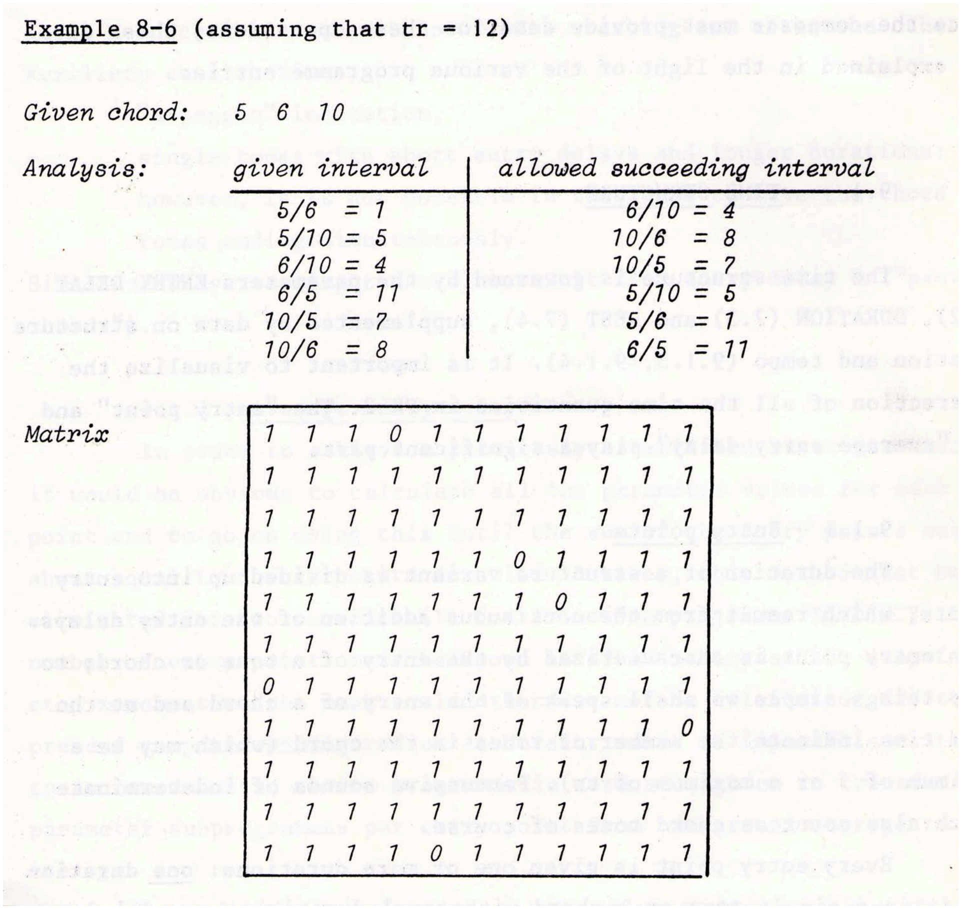
- The order of the chords is decided through using either ALEA, SERIES or SEQUENCE.
- When a chord has been used, it is transposed (also through ALEA,SERIES or a row provided by the composer) and put back in the table. Of course when the order of the chords is decided for by SERIES, it will not be used again until all chords have had their turn.
- ROW:

A row is provided by the composer himself. unlike CHORD, ROW Doesn't control the chord size.

A row must be specified by the composer, which will then be transposed according to ALEA or SERIES, chromatic or serial (using the notes within the row itself as the basis for transposition). There is also an option to not transpose the row at all.

- INTERVAL:

The composer puts in a chord that will then be analyzed and turned into a transition matrix.
The transition matrix will represent what intervals are allowed or not. When the program is run, a endless row results from this method, for every entry point and chord note, it will select one of the allowed intervals from the table.



Example 8-6 (assuming that tr = 12)

Given chord: 5 6 10

Analysis:

given interval	*allowed succeeding interval*
5/6 = 1	6/10 = 4
5/10 = 5	10/6 = 8
6/10 = 4	10/5 = 7
6/5 = 11	5/10 = 5
10/5 = 7	5/6 = 1
10/6 = 8	6/5 = 11

Matrix

$$\begin{array}{|ccccccccccc|}
\hline
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}$$

Analysis and the resulting transition matrix. 1 indicates a forbidden interval.

Final construction of the score:

So the final score is now constructed from the ensembles with the help of the selection principles specified by the composer, however the choice of values is not free for every parameter.

Hierarchy:

One of the most important aspects that sets Project 2 apart from Project 1 is the ability of the program to make parameters interdependent of each other. One of the reasons that it is in Project 2 that the output is to be played by instruments, so playability is a relevant issue. However the composer is of course free to make restrictions that do not come from instrumental limitations,

but just from a compositional strategy. This was certainly something Koenig was aiming for (see quote in the introduction).
The composer can specify limitations for each instrument he's going to use. This includes: pitches, range, durations etc... The composer then specifies which parameter will be the 'main parameter'. The selection of values for the 'main' parameter are then completely free, while the others are restricted.
Let's look at an example how this would work when a composer decides to make instrument the main parameter.
The selection of the instrument parameter for each entry point are then free (that is according to the chosen selection principle), but the other parameters are restricted. If for instance an ALEA function provides an instrument with a duration it is not allowed to play, the program will try again until it finds a duration that does fit. If it can't find one then it will insert a wrong one and print a comment in the score.

Combination/Union:
This is actually one of the more tricky concepts in Project 2. Originally only the Instrument parameter was meant to have more then one group in it's ensemble. Without combination means that for all other parameters (so except instrument), only one group is selected for the formation of the ensemble.
It is however possible to have more then one group for the ensemble's of other parameters. Combination/Union settings then control how the program uses these different groups.
Combination requires 2 things:
- The amount of groups in the table have to be exactly equal to that of the Instrument ensemble.
- The groups must refer to list items that tolerate each other. (see hierarchy)

Then there's the concept of union/no union:
- With union, ensembles consisting of more then one group will just be considered one group.
- Without union, for every instrument group there is created a new layer, the group indexes being the same for the instrument and the other parameters.

The goal of combination is similar to that of hierarchy: it allows the composer to connect certain values with each other, for instance certain group of durations will always be played with a certain group of loudness.

Total Duration:

The composer provides a total duration for the variant. One would expect the program to just fill the score with events until it reached this desired duration, however a different method is used. This is because the tendency mask is not read according to the time in the score, but on the number of the event. So the tendency mask needs to know the total number of events on beforehand. There was also a technical reason because of the limitation of hardware in that time.
For these two reasons Project 2 has to calculate an 'average entry delay' and make an educated guess about the total duration. The amount of needed entry points is then found for by dividing the total duration with the average entry delay. The given amount of entry points are then generated regardlessly wether the desired duration has been attained or not.

Summarize:

The list-table-ensemble concept allows the composer to construct different parameter groups from the same source list. The distinction between list, table and ensemble, creates a certain distance from the material, in that the material is treated on different levels of abstraction. I think this division in steps makes it easier to construct complex structures.
"*There is no pre-established overall form in "project 2", the program is arranged so as to enable the composer to work out structure-variants systematically. I envisaged a method by which the composer designs variant groups, scrutinizes the individual variants once they are composed, decides to have more variants , or programs a new group.*"[Koenig,Genesis of form .p174]
This actually very much reminds me of working in BEA5 on Quite Analog, where I made transformations of my initial sounds, the next settings of my patch would be largely influenced by the previous result.
One starts with the lowest level (the allowed values) then selects certain combinations of them in the table. It also eases the generation of variants, one can change the outcome just by changing the settings for one step.
With the help of tendency masks, the composer can make transitions and combinations from the states contained within the table. I find it inspiring that tendency masks are not just applied at the lowest level (providing boundaries for generating pitches or durations, which is the more casual place you would find the use of tendency masks).

The hierarchy then allows for parameters to conflict limitations on each other, making one parameter more important then the others, thus giving it enough weight to carry the formal progression of a musical segment.

PART III:

My Python Composition Tools

What will follow now is a description of my Python software in it's current state.

My python program is a compositional tool to create CSound scores. Input to CSound always consists of an Orchestra and a Score file. The orchestra contains the oscillators, table read functions, filters, multipliers etc. It is the synthesis part of Csound.
All the durations, start times, frequencies, amplitudes, tablenumbers are contained within the score file, it represents the input data.

The rendering of these scores into sound files is not the final step in the composition though, my material is always edited and sometimes even put trough further processing. It provides a starting point for further exploration.

Before I start explaining, I want to make some 'disclaimers' to avoid misinterpretation.

First of all, I do not consider my software in a finished state. It's interface consists of the python IDLE environment (which is actually not much more than a text editor that highlights the python code in colors). Visualization of output is extremely limited and happens through the LCD object in Max. Because of all this, it is now only meant to use for my own research, I don't expect other people to want/be able to use it. Therefore this chapter will not be a manual to the program, but more a general description of it's concepts. Many of it's functions are only the first steps toward the goal they have been designed for. At the end of this chapter I will try to explain what my ambitions are with some of these functions.

Secondly, this program's history does not lie directly in trying to imitate or develop further the Project 1 & 2 concepts of Koenig, it grew out of wanting to explore the issues that formed the piece Analog Repetition further. Actually one of the strongest inspirations came from listening to some old fortran/VOSIM pieces by Cort Lippe (Tapewalk 1 & 2, Samba, Vosive 1&2). I encountered them during my work digitalising the tapes from the Sonology Archive. I was quite impressed with the timbral structure and the way repetition was used. I later found out that these were actually live improvisations. Sometimes misunderstanding something can lead to new things, I guess.

Through the writing of this thesis I've become, of course, much more aware of the context in which this program might be placed. There are of course similarities between Project 1&2 and my software. I also found solutions in Project 2 that might be able to solve issues that are still present in my own software. However there are also decisions from Koenig that were based on the limitations of hardware and the choice of programming language. Then there is of course the strong serial character, which might be not as relevant for me as it was then (although I'm not allergic to it, like some composers).

Lastly, Project 1 & 2 are designed and used mostly for the creation of instrumental music. Although VOSIM has been connected to the output of Project 1 for example, this was more meant to give a preview of the final instrumental results. However I think it is important to note that my software has also been limited by the instrument : CSound. Although it is much more free in parameters then let's say a piano, there is still a very strong instrumental concept behind it: it's structure devision in Orchestra and Score is a clear indication for that. In the description of my CSound orchestra's I will also explain some of the methods I used to avoid the instrumental limitations.

The discussion of my software will go in three steps:
- The lowest level (the Note, Construction of spectra)
- Intermediary level (the Group, listPointer)
- CSound orchestra's
    - Extending the Note: the FM and AM instrument.
- Upper level, how the material is used in the large form.

And lastly some final conclusions and future developments.

LOWEST LEVEL

The Note:

The Note is the basic event unit in my program. It’s most used attributes are closely related to the fact that I use CSound, which also has a note-like event as it’s smallest possible structure. It might better be called a sound event, but Note is easier to type.
Attributes that are always used:

Attribute	Usage
instrument	CSound instrument number
time	Start time of event
entry-delay	Time between this event start time and the next event’s start time
duration	Duration of envelope
amplitude	Amplitude of sound

more instrument specific attributes, with this I mean that they are optional:

Attribute	Usage
frequencies	List of frequencies, how they are used is decided in the orchestra.
amplitudes	to enable user to make spectra with different amplitudes for the frequency components (strangely enough I very rarely used this)
pan	amplitude per channel
reso	extra frequency for optional band filtering
readpoint	read-point for buffer read processes
speed	read speed for buffer reading (transposition
source	this is an attribute not be used by CSound but that can give the program some information about the way the note was created (see ##)

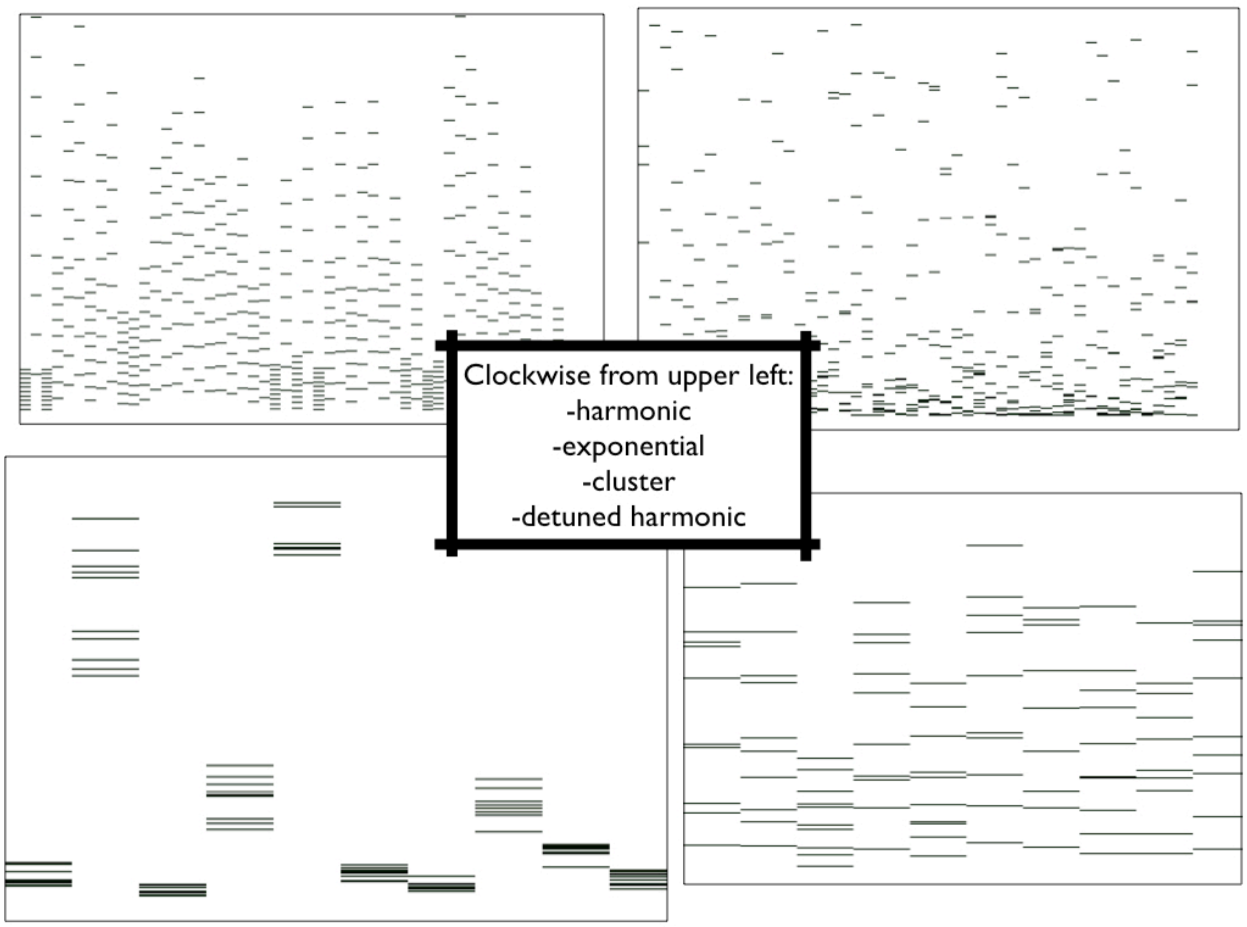
The Group:

The group is the next level in my hierarchy, it consists of a bundle of Note's. The group also contains methods to create durations, frequencies, envelopes...etc. for all the Notes it contains simultaneously.

Frequency creation:

Most of my orchestras use 10 frequencies as a basis, here I will describe some of the methods used.

Type	Description
- Uniform random	*Uniformly distributed frequencies*
- Beta Random	Can be biased to choose close to the limits (or not)
- Cluster	Creates a cluster of frequencies above a randomly chosen base frequency.
- Harmonically	A ground frequency is chosen and upper partials are generated by multiplying the fundamental
- Cluster	A ground frequency is chosen and upper partials are chosen randomly


Clockwise from upper left:
-harmonic
-exponential
-cluster
-detuned harmonic

Methods for creating entry delays:

Type	Description
- Ratio	*A source list is generated with time values related by a ratio*
- From a list	A source list is chosen by the composer and ordered in a way he specifies

- Ratio entry delays:

A basic duration has to be given and normally a single ratio. Then amount durations will be generated by repetitively multiplying the original duration by ratio.
Example:
(base duration: 1 second ratio: 0.5 amount: 5)
1 0.5 0.25 0.125 0.0625

Optionally you can also provide a list instead of one single ratio. For every multi plication one of the ratio's will be chosen randomly.
Example:
(base duration: 1 second ratios: [0.5,0.2,1.5] amount 10)
1 0.5 0.1 0.09 0.045 etc...

- Entry Delays from list:

The user provides a list with durations and specifies how he wants them ordered.

For each duration a weight can also be specified. The weight version is my most used method of creating durations.

- FillDurBOEntryDelays:

This fills the durations attribute based on the entry delays of the notes.

The composer can provide a list with ratio's and the function then makes a random selection from this list. I'm not to extremely happy with it though, because when a entry delay is very short and a short ratio is chosen the resulting sound might become extremely short, or even simply inaudible. This is solved by a special function called clip dur, that fixes a minimum duration for all events in the group.

- fillAmps:

Uses same ratio concept as durations, so a initial amplitude is multiplied with a ratio recursively:

fillAmps (4,0.5)

1, 0.5 , 0.25, 0.125

fillPans:

provides a random pan, the user can provide how many speakers may sound at a time:

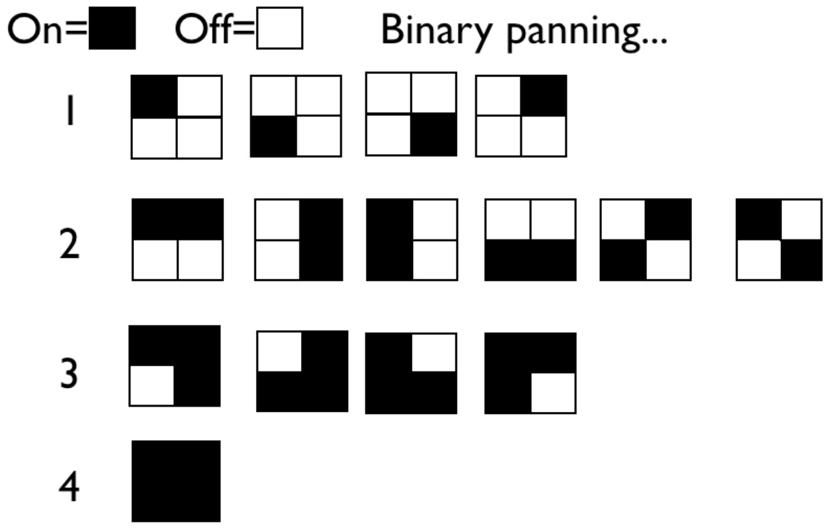


Image (53.1)

Special Group Methods

Combining and weaving Groups.

CopyFromGroup:
this allows you to copy a certain parameter from one group to another replacing what was already there. This makes it possible to use a lPointer for one parameter exclusively (thus breaking the link between parameters).
For example if you want a certain lPointer to only effect the order of the entry delays:

```
b = a.runlPointer(apointer) # run lPointer
a.CopyFromGroup(b,entrydelay) # copy EntryDelays from b.
```

This method is actually very important in the use of this software, it allows me to reorder a Group, but only use one parameter from it.

Weaving Groups:

The idea behind this is that you first make 4 groups that are different from each other in character. Then with the 'supergroup' methods you can weave notes from these groups with each other. With weaving I mean that a new Group is formed from combinations of chunks from the 4 source Groups.
The pattern in which this happens is definable by the user. The pattern is defined by a string containing the letters 'a' 'b' 'c' 'd'. Every letter is replaced by a note from the corresponding Group.
a a a a a b b c c d a b b b c d d.

Note: More interesting generation methods should be possible. I would like to increase the number of Groups and to be able to use tendency masks to make the choice out of which groups to use values.

Shape Shotgun.

This is a tool that I created to solve the problem of creating structures with changing densities. The idea of it's function is to put in a Group containing a balanced (high) density of events. Then for every event the shape specifies a chance that the event will be deleted from the group. The control for density is thus not exact, but for large numbers of events this difference will not be noticeable. The shape can be specified through providing lists of points, or by drawing it in a max patch.

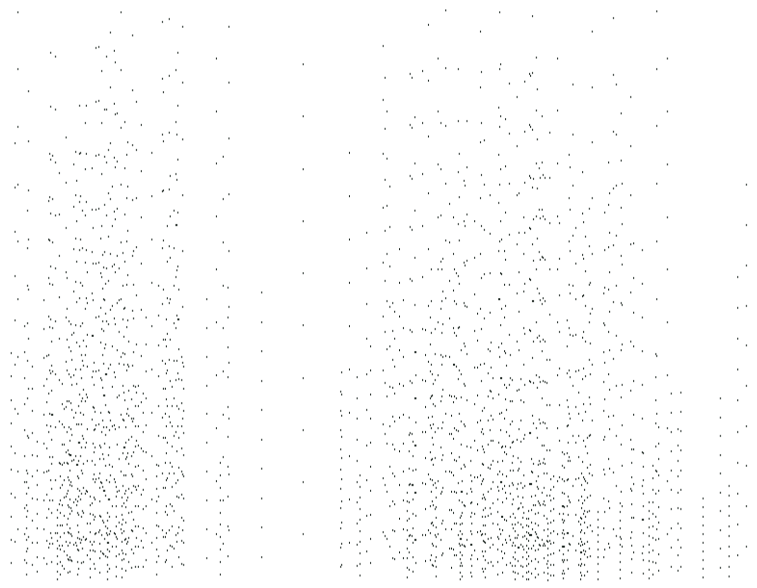


Image 55.1

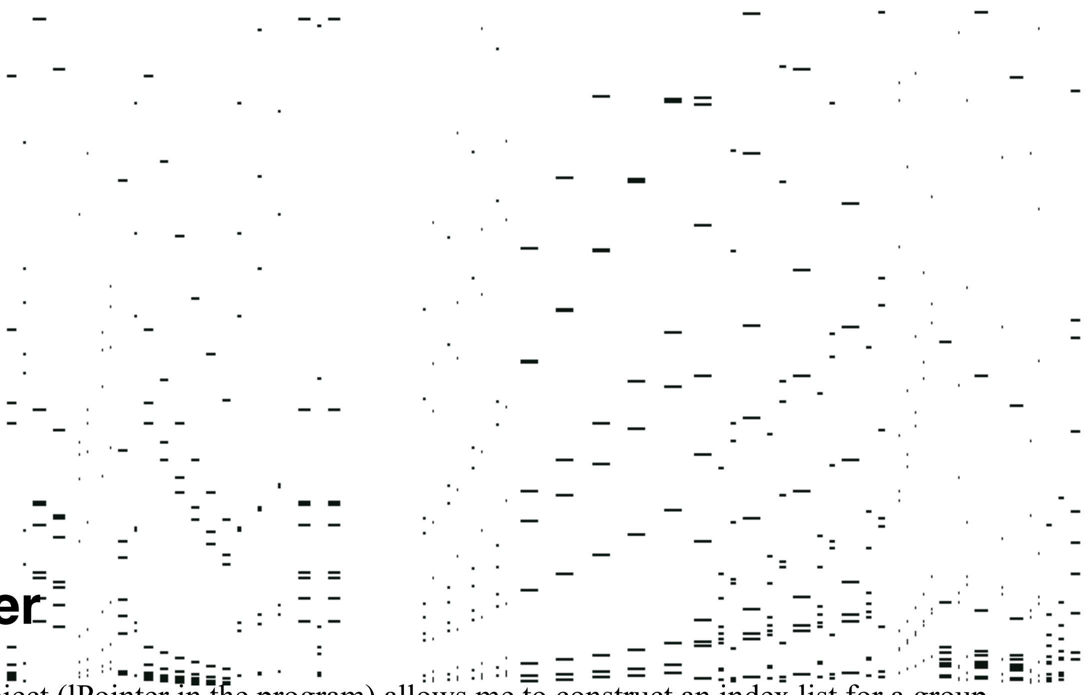
Modulations.

Parameters that can be modulated:
start times
durations
entry delay
frequencies
read point in buffer

Shape Modulations:
The shape itself can be provided in numbers, or drawn in a max patch.
The amount of allowed (random) deviation is then based on the Shape. These methods I mostly used to create subtle differences between 4 otherwise identical channels.

lPointer based modulations.
These modulations are very similar to the ones I used in my piece Analog repetitions. At every first Note of a loop (the loop concept will be explained in the lPointer part) a new modulation

factor is chosen, and for every note in the sequence this factor is multiplied with itself, thus resulting in an exponentially increasing modulation.


Lpointer
based modulation
of frequency:
Image 56.1

ListPointer

The ListPointer object (lPointer in the program) allows me to construct an index list for a group to be reordered. It contains different methods to create these lists of indexes. The Group object itself has a method runlPointer that can be used to reorder the group with the indexes provided by the ListPointer. For me the initial interest was not so much what elements are selected, but much more how they are ordered and repeated.
Lists of indexes can themselves also be shuffled or mutated to create variants. The ListPointer can be compared to the selection programs in Project 2.

VintageLoops

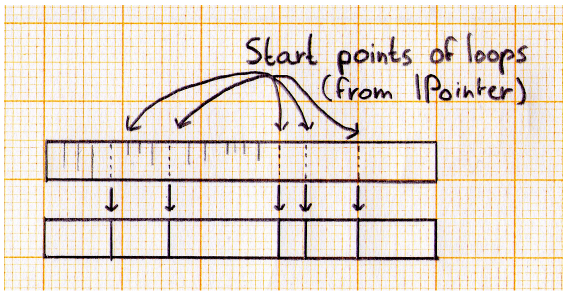
These are called this way because they are inherited from my earlier BEA5 piece (analog repetition). The loop has 3 parameters that are defined as such:
inner loop size :the amount of indexes included in the repetition:
outer loop size : the amount this set of indexes is repeated
start index : the index where the loop should be started

so an example with start index = 4, inner loop size = 4, outer loop size = 3 :
4,5,6,7 4,5,6,7 4,5,6,7

An important aspect of the ListPointer object is that it also stores the number of the loop (so the repetition) and the local index of the loop. For above example this looks something like this:

Indexes	4 5 6 7	4 5 6 7	4 5 6 7	8 9	8 9	8 9	8 9	2 3 4 5 6 etc...
Loop nr	1 1 1 1	2 2 2 2	3 3 3 3	1 1	2 2	3 3	4 4	1 1 1 1 1
Local	1 2 3 4	1 2 3 4	1 2 3 4	1 2	1 2	1 2	1 2	1 2 3 4 5

This extra information can be used to do special modulation on the first event of a loop, or to even make a new group that has a time structure with events placed exactly on the beginning of the Source Groups-loops.
I used this to make different layers of material that are structurally related in this way. The way one group is ordered becomes the time structure of the other.



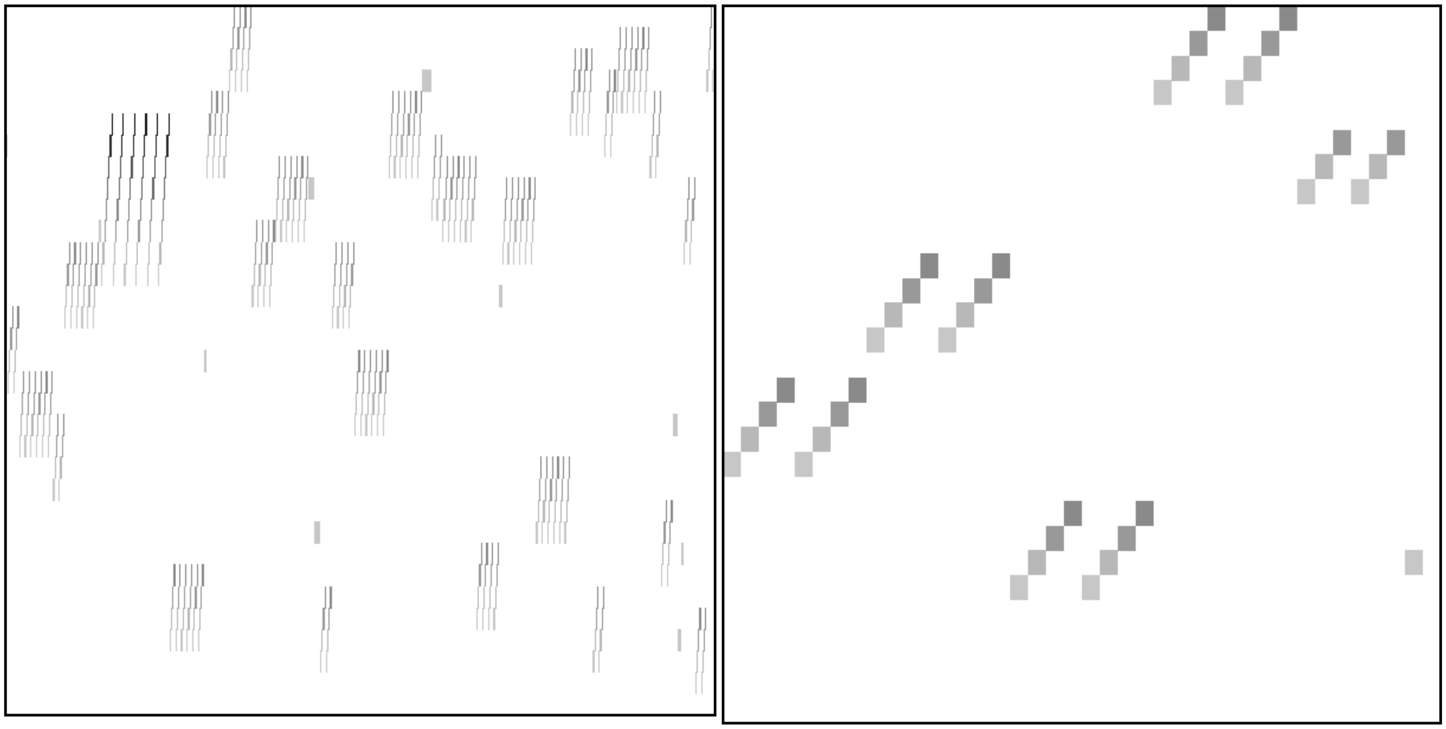


image (57.1) A visual representation of a vintageLoop, and on the right a close-up

Weighted-loops

Very similar in form to the Vintage Loops, only the specification of the length of the loops and the amount of repetition is no longer specified as a range but by means of weights.

for example:

weight list:	**[1 ,0 ,0 ,2 ,1 ,5]**	**weight list:**	**[1 ,1 ,0 ,3]**
loop length:	***1 2 3 4 5 6***	***nr of repetitions***	***1 2 3 4***

this means that:
- only loop-sizes of 1, 4, 5 or 6 values will be created in the ratio 1:2:1:5
- the amounts of repetition will be 1,2 or 4 in the ratio's 1:1:3

Possible improvement:

Direct input of values and corresponding weights for long repetitions:
[[1,2],[3,5],[14,1]] etc...

Walking Loops:

Initially there is a list with a certain number of randomly chosen indexes:
30,12,22,23
The next values will be a repetition of the old list but some of the indexes have changed by a single step up or down, thus introducing a different event in the loop.
29,12,22,24
this process is repeated:
29,13,22,24
29,13,21,25
28,13,21,25
28,13,21,25
28,12,21,25
27,12,22,26

Horizontally the indexes thus behave like a random walk. The amount of mutation can be set through a parameter.

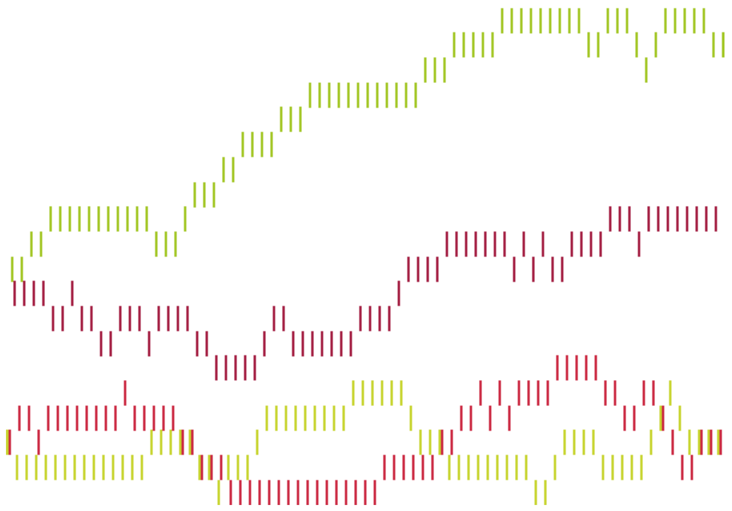


Image (59.1) A walking list

Mutating List

This is very similar to the walking list method, only in that the size of the loop changes over time because new indexes are added without overwriting the old one. A minimal and a maximal amount of values can be specified to keep the size under control.

28,13,21,25
27,13,21,24,17
26,13,20,24,18
13,20,24,18
etc...

Serial Loops:
This is based on the concept on wether indexes are not allowed to repeat until all indexes in the input have been used. When all items have been used, the indexes are serially transposed.

transposition	Resulting Series
0	1347652
3	4673215
4	5714324

UNDER DEVELOPMENT:

Now that I've become familiar with the structure of Project 1 & 2, I think that there are certain things that could be improved in my own software.

Vertical Density or 'chords':

Reading about project 2, I realized that there is not really a concept of vertical harmony (or dis-harmony) in my software. Of course I'm able to create different layers that have the same time structure (through the use of the method copyFromGroup). This doesn't solve however completely the problem of how elements that start at the same time are related to each other.

lPointer control through tendency masks:

I would really like to control the behavior of the lPointer over time, so I can make transitions of for example from small repetitions to large repetitions, thus gaining control over periodicity just like Koenig had in Project 1, the difference being that he had no tendency mask available in Project 1, these changes in periodicity were controlled in a stepwise way through the branching table. One fantasy I have is a sound-structure that starts chaotic and then gradually gains structure or periodicity.

Horizontal Density:
Density was problematic until very recently, I've now solved it (for high density's) with the shotgun concept, which I explained earlier. However at low densities, control becomes quite difficult. I think that I might devise a method of generate and test, similar to the concept of hierarchy in Project 2. What I mean is that I would make density a main parameter.
Entry delays should then be chosen in such a way that the resulting density of a section is equal to the required amount.
Probably this will only work well for low densities, or at least a small number of events, because if to many events are involved, the chance of finding the right solution by random selection will become vanishingly small. Maybe a sort of switching process has to be designed that controls low density's through hierarchy, and high density's with the shotgun method. Research will have to be done. My goal with all this is creating a variable density without losing all control of event to event characteristics.

lPointer shuffle:

A normal shuffle would be:
1 2 3 4 5 6 7 8 9
4 6 5 2 1 8 7 9 3

By using smaller chunks, the order can be shuffled locally, but global structure remains. example:

1 2 3 4 5 6 7 8 9

[1 2] [3 4 5 6] [7 8 9]

[2 1] [4 6 5 3] [7 9 8]

2 1 4 6 5 3 7 9 8

By redoing this shuffle multiple times the order will deviate more and more from the original. Thus I could make a collection of lPointers and then have control over how similar they are.

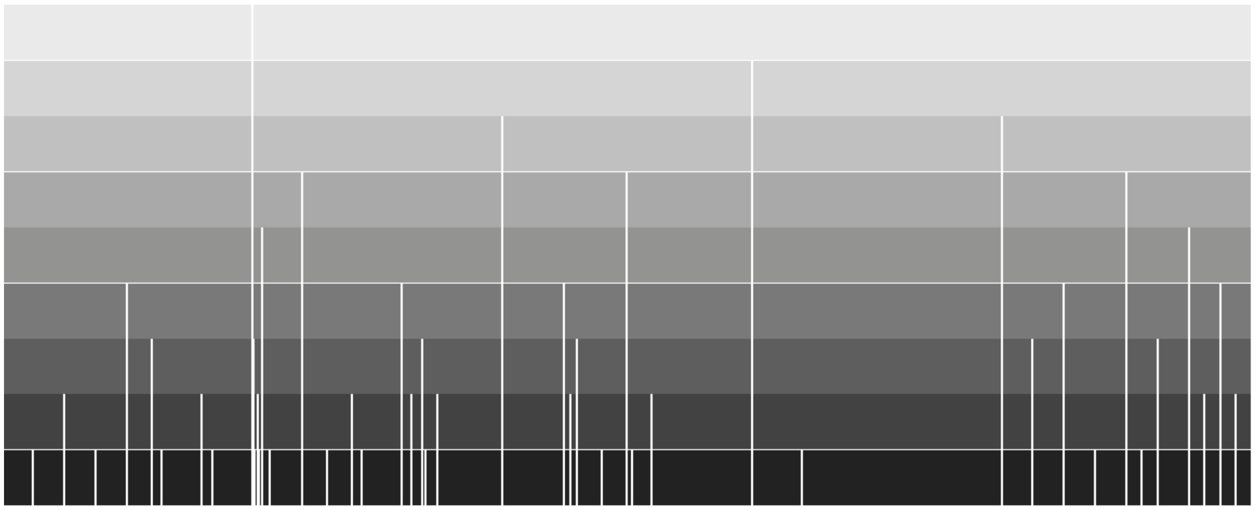
Time Trees: A different approach to connecting layers

The idea behind time trees is very simple. This is the algorithm:

```
For every Note in the Group:
3. Randomly decide wether the note is to be split or not:
- Yes -> goto step 2
- No -> go back to step 1
2. Choose a ratio from the list.
3. Split the Note into two copies, splitting the entry delays according to ratio.
4. Store list and go back to step 1
```

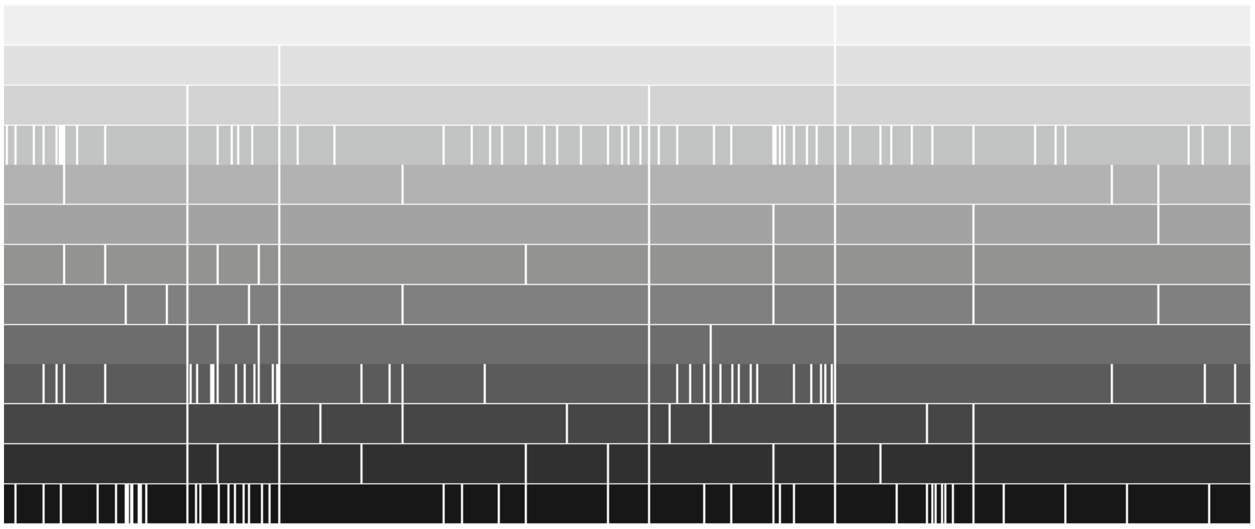
This results is a set of entry delays like this:

1
0.5 0.5
0.5 0.25 0.25
0.5 0.125 0.125 0.25
0.5 0.025 0.100 0.125 0.20 0.05 etc...



visual representation of a time structure tree:

The reason why this could be a useful method of creating duration is that it allows me to create different time structures that are more or less related by each other. In the following example a duration is divided 4 times (rule 1 to 4) and then 10 variations are presented.



Another interesting aspect of this is that on all scales the same ratio’s apply.

However I think all this will only really be relevant if the other parameters are also effected by how the duration are divided.

Csound Orchestra’s

First of all it might be a good idea to explain why I choose Csound for rendering my sounds and not Supercollider or Max/MSP. CSound has some specific drawbacks:

- Realtime function is quite limited.
- It’s more aimed for the creation of events instead of streams.
- Sometimes rendering in high quality requires patience, although my instruments tend to be not to extreme in this respect.

The advantages though:

- Polyphony is never an issue, events that overlap do not cause any problem whatsoever.
- Start phases of oscillators can be controlled very exactly for every event, this gives better sounding attacks.
- Doesn’t crash. Ever.
- Timing is exact to the sample, timing in max/MSP appears extremely messy to me in normal mode, and still sloppy when the schedular is put in audio interrupt mode (for < 1ms durations) This can be very annoying when I’m trying to make multiple layers with synchronized events.

The choice of CSound as my rendering software for my structures has been quite influential in the way I constructed my software.

Sine:

My most basic orchestra just contains 10 sine-wave generators that have an amplitude envelope applied to them.

Filtered Noise:

Same as above only instead of oscillators, this instrument uses band-filtered noise.

Filtered Pulse:

This instrument uses the lowest frequency in the list as a frequency for an anti-aliased pulse generator. The pulses are then band pass filtered as in the filtered-noise instrument.

Filtered File:

Same as filtered noise and filtered pulse, only the input for the filters can now be a 4 channel file. This gives rise to 2 extra parameters, read-point and speed (transposition)

Ring-modulated File:

Same as the sine instrument with the only difference that it's output is multiplied with a file.

File:

Just playing a file, frequencies are ignored, only amplitude, speed, readpoint, duration remain.

Voc Osc:

This instrument is a variation on FOF, but highly simplified, the only parameters being the fundamental frequency and 9 formant frequencies. Just like filtered pulse it uses the lowest frequency as the 'exciter'.

Extending the Note

All before mentioned instruments work best for events not much longer then a few seconds.
The results generally are very quantized sound structures, which was something I actually aimed for when I was designing the python software and orchestra's. However after making my first pieces with it, I worried that my software was maybe getting too 'instrumental' i.e. limited to this quantized textures.
One way I tried to avoid this is by using less 'instrumental' amplitude envelopes (square envelopes), making durations larger then the entry delays. Another way of achieving blur, is processing the output of CSound with simple Max/MSP patches doing granular synthesis Amplitude modulation, convolution with extremely long impulse responses, or filtering with high resonance etc...

However the most successful solution came by finding a way to modulate parameters within the note event. The result is a sound event which has a changing spectrum over time. It is interesting to note that Koenig also differentiates the electronic sound event (Klangereigniss is the german term) from the instrumental note , by stating that it has parameters that can change over time. (äst. prax. 3 p197)

- FM Instrument

In designing this instrument I choose not to use line-segment generators, but actually fill function tables (buffers) with values, and read them with cubic interpolation.

To fill the values of the table I actually combine the amplitude parameter of multiple notes into one table. This is still a bit arbitrary method, and I wish to try out more subtle ones in the future. This would have to lead to more precise control of how the sounds timbre changes over time.

The structure of the FM synthesis is based largely on an old soft synth I was using a lot in my early sonology days: the FM7 (which is in it's turn was inspired by the famous YAMAHA DX7)

- Seven oscillators, six of them modulating the fundamental.
- The mix between the 7 oscillators to the output can be controlled through the orchestra.

Some sound examples can be found on the cd.

Some example code (more available on the cd)

from may21 import * # Import Software

```
# An example of making variants



CSound = CScore() # Contains the methods to generate the final .sco files for csound, out of the user defined
groups

modclass = modData() # Class for modulation

functions = function()



a = Group()

a.fillEmpty(300) # create 100 events

a.fillFreqs(20,20000,3) # create frequencies for every note

a.fillEntryDelaysList([[1,5],[2,4],[3,3],[5,2],[8,1],[13,1]],'customweight') # use a weighted choice to fill
entry delays

a.fillDurBOEntryDelays([0.5,1.0,2.0,3.0]) # ratio's for the durations

a.fillAmps(5,0.7) # fill amplitudes

a.fillPans([1,0,0,0]) # randomly choose pans (only 1 channel at a time will sound)



# The next section will re-order the frequencies with a listPointer.



apointer = lPointer()

apointer.weightedLoops([4,2,1,1,1,1],[4,3,2,1,1,1,1,1],100,300)



b = a.runlPointer(apointer) # create a temporary new group containing the listpointed frequencies



a.copyFromGroup(b,'freqs') # the listpointed freqs are recombined with the original group



a.stockorder() # adds up the entry delays to create the timestructure

a.forceLength(60) # normalize a score to wanted duration



CSound.fstatements = """f 1 0 16384 10 1

f 2 0 16384 25 0 0.01 100 1 14000 1 16384 0.00001

f 3 0 16384 20 2 1

f 4 0 524288 1 \"/Users/casper/Documents/Pythonstuff/CsoundOrcGen/bloup.aiff\" 0 0 0

f 5 0 16384 25 0 0.00001 100 1 16380 0.001 16384 0.000001 \n""" #fill functiontables (envelopes, soundfiles
etc)

CSound.addFunctions(functions) # Fill the amptables

CSound.fillWithGroup(a) #Write csound .sco file
```

APPENDIX...

Considerations in choosing Python:

Advantages:

- I had used it before once, and it was a pleasant experience.
- Rather good documentation, good (external) tutorials.

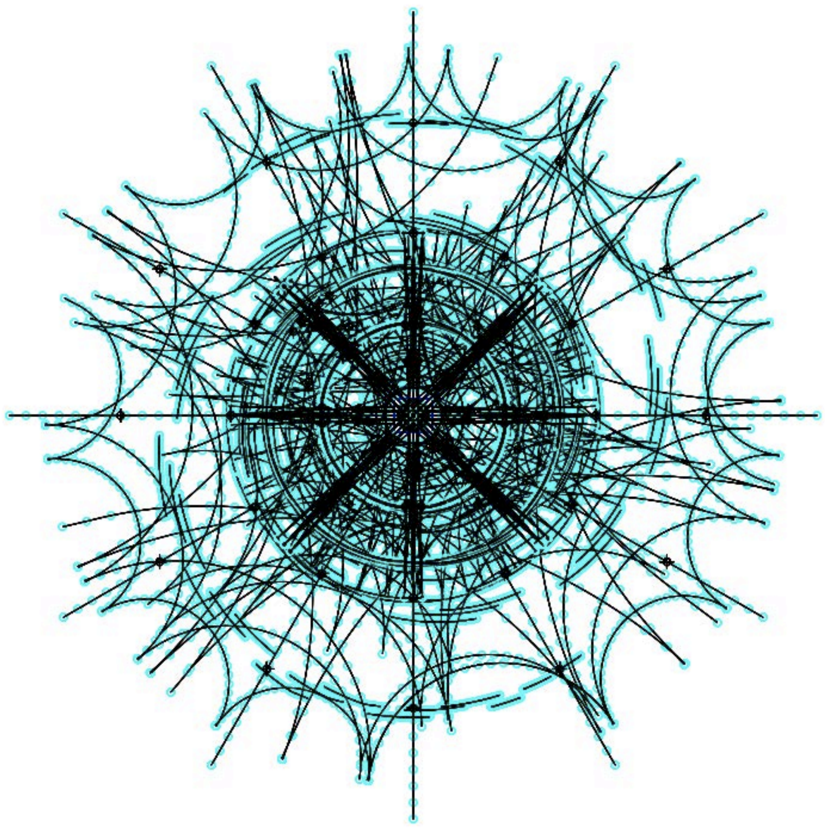
(I must note that the official tutorial gets a bit vague with the classes part, starts to talk about all kinds of issues of design that I was not interested in.)

- Python is an interpreted language: this I find an extremely important factor, because it means that I can test things while I'm programming them, even single lines of code to check if I understood some new concept properly. Because I'm not a hard-core programmer this probably saved me lots of time.
- Portable, not platform depended (not important at the moment)
- It's treatment of lists is very developed, tasks like sorting a list in a custom way are very easy to implement.
- The 'object oriented'-aspect is clear and understandable for me. It's nice that you can write __add__ methods for your own objects so you can use '+' operators on your own objects. This is called operator overloading.
- Lots of library's and examples are available, and because of python's relative strictness, other's people code is actually readable.
- I can program things with reasonable speed.

Drawbacks:

- Slow, because it is always interpreted language, and can't be compiled like LISP. For me this has never been a real issue, because I don't do heavy computation with it.
- Using the print function seems to slow down output, so it is better to write a log file.
- the IDLE development environment is still somewhat underdeveloped, but this might also be an advantage, because if compared to X-Code, it's at least
- It's error messaging could be better, for example it often gives the rule where an error occurs but it doesn't

Work on Sternenrest



A Part of the spatial movements

This is a project I got involved in together with (fellow student) Billy Bultheel and Johan van Krey, somewhere around 2006. It is a piece composed for Wave Field Synthesis (WFS) and ensemble by Willem Boogman. The scale of the piece was just colosal , we have been working on it for 3 years now (not continuously, of course). The last of the electronic music was finished this year. It has been preformed 3 times now together with Olaf Tarenskeen, Arnold Marinissen and the Spectra Ensemble.

The inspiration for Boogman to write the piece came through a lecture of astrophysicist Connie Aerts called ‘kosmische symfonieën’ (cosmic symphonies), in which she told about her research on a star called HD-129929. What made this star special is it’s rare inner structure that produces particular loud vibrations within the star. These vibrations are then detectable as fluctuations in the light reaching earth. With this information something can be told about the inner structure of the star, something which is impossible to do in any other way. It is a field called astro-seismology, and closely related to earth seismology. The only way we can tell something about the inner structure of the earth (size of the core, size of mantel etc.) is by the vibrations that travel through the earth. The same goes for stars.

The piece consists of 3 movements:
- Seeds of Structure: a guitar solo lasting 18 minutes with just 3 minutes of very minimal electronics that where only finished after the 2nd preformance of the piece)
- GLAS: HD-129929 the most important movement of the piece, which is about the lifecycle of the star itself. The electronic music is accompanied by a part played on glass objects by percussionist Arnold Marinissen.
- Liminale: This is about two pulsars slowly rotating towards each other finally merging forming a black hole. The main part of the music is the ensemble playing pitches and chords based on the star frequencies.

I created the electronics for the first part, seeds of structure. The electronic music was actually finished last for this section. It uses the glassy sounds from the second part to represent a starry sky. Billy made the WFS score with point sources.

Most of the time spend on making the piece went into the 11 minute 2nd part of the piece, which is the star itself. The material in this piece consists of a few main groups:
- Areas of noise (created by Johan, later I made some changes to reduce CPU cost)
- Traveling waves (paths created by Johan, sounds created by me)
- R-modes (this was a mode of vibration that went right through the centre, glass filtering provided by me)
- Coresounds (sound provided with a patch of Johan controlled by Billy)
- Core traveling modes (Johan)
- Core pulse (Billy)

Of course Willem was present with most of the creation of these sounds, or the tuning of them.

Willem always had the ambition to be accurate about the data, but that doesn't mean that the piece is a direct model of the star. It is a combination of the reconstruction of the star but also a , and also a artistic view on what a .
The frequencies, rhythms and spatial movement of the electronic music (and the score for the instruments) are all based on the frequencies of the star. Of course the frequencies had to be transposed to be even usable at all (some of the period sizes are in the range of a few days).

The timbre of the electronic sounds are either based on noisy crackle like sounds (produced with a patch designed by Johan). Other glassy sounds based on the objects of the percussionist. These glass objects consisted of tubes, vases, a lampshade, little square plates, round plates, and a big glass table. They were very diverse timbrally.

One of my tasks in this project was to make the synthetic glass sound. For this I first did frequency analysis on one of the recordings and tried to reproduce the sound electronically.
The first thing I noticed is that glass has very pure resonances, but almost no harmonic relation between them. Some objects are very rich in resonances (hundreds of them) while others only have just a few clear ones. Generally the lowest is the strongest, although there are certainly exceptions on that rule. My first experiments was to try and reconstruct the glass sounds by just adding up pure sine waves, which did make the original sound recognizable, but I was not satisfied yet with the quality. In the end I used very high resonance bandpass-filters, which also solved another requirement of Willem, the ability to insert the glassiness in the crackle sounds further on in the composition. I also added a final band pass filter over the whole spectrum to give the spectrum a more natural envelope.

So then I analyzed all the remaining sounds with the help of Audacity, because the latest version has very descent peak detection in it's spectral analysis function, I also wrote down what was the most prevalent frequency in the spectrum. I tried to automate this process but decided in the end that it was faster and more safe to just do it by hand.

The glassy sounds formed certain chords (6 in total) al containing a maximum of 5 different tones. I made a patch that created the envelopes and also added a certain modulation in the amplitude of the sounds again related to the star's frequencies.

The paths for the traveling waves (see image) where created by a patch made by Johan, I constructed a patch that was able to link the movements to the right soundfile and directly generated a WFS score file (we thus bypassed the use of Wouter Snoei's interface).
In testing these movements we soon discovered that the doppler effect was very strong on these kind of movements, it completely transformed the sound. We then tried to make the star smaller and this made the doppler effect of course much less prevalent, but also the movement less dramatic. So in the end we decided to let the star 'grow' during the piece. In the beginning the sounds move from a maximum of 40 meters away and later almost 100 meters.

A rather interesting effect (but also complicating) was that because of the size of the composition (not in time or complexity, but literally the size) things that where synchronous in the score turned out to be asynchronous in the final result, because of distance differences. I realized that in WFS time becomes relative. I think you can imagine in what way this complicated providing a click track for Arnold and the ensemble.

Billy produced the sound for the core.

When we started to put all the separate parts together, some new problem was raised, the system couldn't handle all the stuff at the same time. More annoyingly when the system overloaded it started to produce ear-splitting beeps at unexpected moments. I don't just mean a little click, but really full gain beep, something like a fire truck at a distance of 1 meter. Especially at the moment where the star explodes, and a lot of sources are in the center of listening area, it became clear that we needed to think about optimization, or at least reducing the sound sources. Luckily for us, one of the PowerMacs drowned in it's own cooling fluid, and both servers were replaced with Intel Macs, eventually giving a lot more CPU power, easing the problems a bit.

First performance of the piece was in an old warehouse in Den Bosch. The acoustics were very nice and the piece sounded much better then it did in Leiden.

Recently Wouter has changed some of the ways of rendering in the software, resulting in an even more accurate rendering of the sound-sources. Especially the traveling waves profit from this improvement, finally the listener should be able to clearly to follow their curved trajectories. Sadly the CPU use increased and again we got problems with huge beeps that could appear at any moment during performance of the piece. So we had to again remove some material (some of R-modes are completely removed after 6 minutes into the piece. Still I think it was worth the sacrifice, the sound has improved quite a lot.

The third part: Liminale

This part is about the remnants of the star moving outwards (represented by glass samples), something Billy designed the sound for together with Willem. At the same time two pulsars are rotating around each other and near the end of the piece form a black hole, which doesn't make sound itself, but it does deform the glass sounds that remain.

The sound of one of the Pulsars is based on a sample that is an actual recording of astronomers. It is quite an interesting sound, because it has all kind of strange rhythmic qualities, although pulsars are very stable objects (mega high mass in very very small area). Because the signal is so weak it's quality probably has a lot to do with the way it was filtered out of the background noise. It was 'tamed' though, and it's rhythmic quirks where controlled by a list provided by Willem

I do feel the piece suffers a bit from it's huge density, you really need to hear it a few times to get everything out of it.

-

Experiences making the Bilthoven course.

In the early days of electronic music, the equipment used to make electronic music was still very primitive, and the majority of it not even designed for composing electronic music. Because of the limitations, making complex sounds often required lots of (manual) steps. Even for as little as a minute of electronic music consisting of sounds we would now consider trivial and basic, people spend hours cutting and splicing little pieces of tape in the studio.

I recently have acquired some feel with the immense dedication required making electronic music this way, by making the Bilthoven course by Koenig. It was given in 1964-1965 and meant as an introduction course for composers into the then still very young genre of electronic music. The equipment in the Bilthoven studio was very limited (even for it's time). For Koenig himself it must have been like traveling back in time compared to the equipment he was used to have at his disposal in the Cologne studios.

Even though I actually 'cheated' at many points during the process (using automated programs and sequencing software to do some of the *zerhackungen* and making the sine-wave mixtures), it was still quite a labour intensive process. Surprisingly, there were many moments during the process that I found very enjoyable and inspiring. Even though the exercises are quite strict in terms of freedom on behalf of the realizer, you never feel out of touch with the material. It is always clear what effect the transformations have on the material. This is not as trivial as it may seem. For me it was a refreshing experience having spent a lot of time programming my python software, where it would sometimes take me a day or two before I could really hear the results of my ideas.

Because you are having your hands (and ears) on the material I also felt more responsible about certain small decisions that where left free by Koenig. It forces you to be focussed on the details of the compositional process. Examples are the beginning and end frequencies of glissandos and the center-frequencies of bandpass filtering. Although these choices are completely left to you, doesn't imply that you can't make a strict plan or use some kind of simple algorithm/strategy for choosing these parameters, but it does mean that you are very directly confronted by the consequences of that particular choice.

The way it is sequenced or synchronized I also found quite fascinating, often Koenig provided *zahlenmaterial* that consisted of numbers representing tape centimeters. These then have to be used by the composer to make an interesting time structure. There is a lot of emphasis from Koenig on planning the time structure by drawing a plan for this.

Finally even though the process takes a lot of time, there is also a certain efficiency to it. When working on one of my own pieces I start by creating massive amounts of different material and large groups and sections of it are thrown away because they are not usable or no longer fit the aesthetic of the piece. If a problem arises, the solution was mostly in the creating even more material. The Bilthoven Piece is much more efficient in that at least all of the final material is used in the piece, and it is even optional to combine it with some of the material needed in the production of the sounds. But at least all the sounds where necessary for the end result.

Structure of making the Bilthoven piece.

- Syntheses:

The basic source material consists of two types:

- mixes of sine waves -> spectrum
- mixing different layers of pulses -> puls-chords

These are then put through a series of transformations:

- cut and spliced with pauses to form little rhythm structures.
- filtering
- enveloped
- ring modulated with filtered noise, sine tone, or a sine tone with a glissando.
- '*zerhackung*': sliced with a paspartout band.
- transpositions

In the end almost all of the material is synchronized to make small form sections.
These little sections are then put in a certain order that forms the whole piece.

Content Of Cd:

- Pieces (in stereo format)

Quite Analog	(BEA5)
Reflective Surfaces	(Max/MSP)
Analog repetition	(BEA5)
A useful waste of time	(python)
Decatola	(python)

- Full Python Code & some examples (with audio)
- Csound Orchestra’s

Note: My first idea was to include the python code in this papaer version but since that would take about the same size as the rest of the writings, I changed my mind.

BIBLIOGRAPHY

Ariza C. "An Open design For Computer-Aided Algorithmic Music Composition" Boca Raton, Florida Dissertation.com, 2005

Dennet, Daniel C., "Darwin's Dangerous Idea"

1995, Simon & Schuster Paperbacks, Inc.

Harley, James, "Xenakis, His Life In Music"

New York 2004, Routledge

Koenig, G.M. Ästhetische Praxis Texte zut Muzik" band 2

PFAU-Verlag,

Koenig, G.M. Ästhetische Praxis Texte zut Muzik" band 3

PFAU-Verlag,1993

Koenig, G.M. Ästhetische Praxis Texte zut Muzik" band 5

PFAU-Verlag,2002

Koenig, G.M. Genesis of Form

Interface Vol 16 (1987) pp 165-175

Koenig, G.M. Projekt 1 HelpCollection (version 2002.38)

(distributed with Project 1 programme, available from the composers website http://www.koenigproject.nl'

Koenig, G.M, "Electronic Music Reports No3"

"project 2: A programme for musical composition"

december 1970, Institute of Sonology, Utrecht

Schoenberg, Arnold "fundamentals of Musical Composition"

1970 Faber and Faber Limited pages 1 - 2

Xenakis, Iannis "Formalized Music"

1992 pendragon press

Ulam, Stanislaw "Adventures of a mathematician"

1979 Scribners