Return to Index

# Creating and Exploring the Huge Space Called Sound: Interactive Evolution as a Composition Tool

Palle Dahlstedt

As a starting point for the operations, the user can begin either with a set of randomly generated genomes or any previously stored genome. Sometimes it is also possible to import genomes, if the sound engine is capable of transmitting its parameter values. This may be useful for example when mating factory sounds in a synthesizer.

## *2.1 The Genetic Operators*

Genetic operators take one or two parent genomes and generates variations or combinations of them, resulting in a new population of genomes to be evaluated. In MutaSynth, the genetic operators used are *mutation*, *mating*, *insemination* and *morphing*. These are described individually below. Mutation and mating (also known as *crossover*) are standard genetic operations modelled after the genetic replication processes in nature. Insemination is a variation on crossover where the amount of genes that are inherited from each parent can be controlled. What I call morphing is a linear interpolation on the gene level.

Every operator creates a set of new genomes that can be auditioned and further bred upon in the interactive process. Any sound can be stored at any stage in a gene bank, and the stored genomes can be brought back into the breeding process anytime, or saved to disk for later use. The parents used in an operation can be selected from several sources: a previously stored genome, either of the most recently used parents, any uploadable sound in the current sound engine or an individual from the current population (i.e., the outcome of the last breeding operation).

A genome is really just a constant length string of numbers. Another sound engine would interpret these numbers differently. This means that a genome is meaningless without the sound engine it was created for, and it will not work with any other engine.

It is sometimes useful to be able to prevent a number of genes from being affected by the genetic operations. For instance, when certain parameters of a sound (e.g., the filter settings) is good enough and the user does not want to run the risk of messing them up in further breeding operations, she can disable them, and they will stay as they are. If a gene is disabled, it will be copied straight from the first parent.

### Mutation

A new genome is generated from one parent sound's genome by randomly altering some genes. A *mutation probability* setting controls the probability of a gene to be altered and a *mutation range* sets the maximum for the random change of a gene. Together, these two allow control of the degree of change, from small mutations on every parameter to few but big mutations.

### Mating (crossover)

Segments of the two parent genomes are combined to form a new genome. The offspring's genes are copied, one gene at the time, from one of the parent genomes. A *crossover probability* setting controls the probability at each step to switch source parent. The starting parent for the copying process is selected randomly. Each parent will provide half of the offspring's genes, on average. The genes keep their position within the genome during this copying.

### Insemination (asymmetrical crossover)

For a new offspring genome (Q), the following process is applied, based on two parent genomes ($P_1$ and $P_2$): $P_1$ is duplicated to Q, then a number of genes are overwritten with the corresponding genes in $P_2$. An *insemination amount* controls how much of $P_2$ should be inseminated in $P_1$, and the *insemination spread* setting controls how much the genes to be inseminated should be spread in the genome - should they be scattered randomly or appear in one continuous sequence. If the *insemination amount* is small, the resulting sounds will be close in character to the sound of $P_1$, with some properties inherited from $P_2$.

### Morphing

A linear interpolation is performed on every gene of the two parent genomes, forming a new genome on a random position on the straight line in parameter space between the first parent ($P_1$) and the second parent ($P_2$).

### Manual mutation

Manual mutation is not a genetical operator, but still something that affects the current genome. When the user changes a parameter on the synthesizer, the program is informed about the change and applies the change to the

corresponding gene in the currently selected genome. The manual change then lives on through further breeding. This allows for the same level of direct control that the advanced synthesizer programmer is used to, and makes the method useful to both beginners and experienced users. Manual mutation may not be possible with all sound engines, depending on if they transmit parameter changes via MIDI.

## *2.2 User Interface*

MutaSynth is made to be simple. It is also designed to give quick responses to user actions, to minimize all obstacles in the creative process. The current userface looks like this:

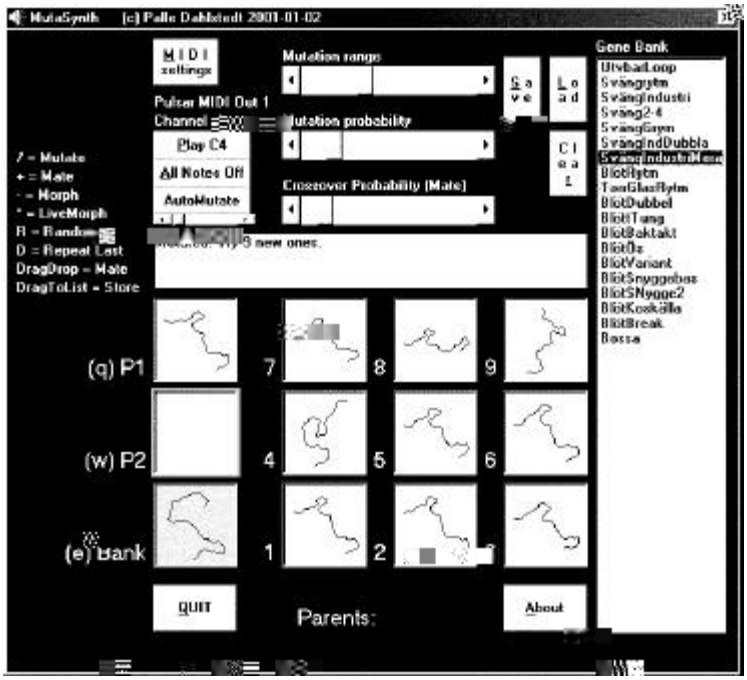


*Fig. 1: The current user interface of MutaSynth.*

The display shows a number of boxes, representing the population, the last used parents and the currently selected genome in the gene bank. The layout is chosen to correspond to the nine number keys on the computer keyboard. To listen to any individual from the current population, the user presses the corresponding number key, and the parameter interpretation of the genome is sent to the sound engine. The keys +, -, * and / invoke the different breeding operators. With these keyboard shortcuts the composer can keep one hand on her instrument and one on the number keyboard, for quick access to the operators and the individual sounds.

### 2.2.1 Visual Representation of Genomes

The genomes are represented graphically, to aid the aural memory and to visualize the proximity in parameter space between genomes. For this purpose, I have developed a simple mapping technique. The gene values are interpreted as distances and angles alternatively (scaled to a reasonable range) and drawn as turtle graphics. The resulting worm-like line is scaled to fit in a picture box.

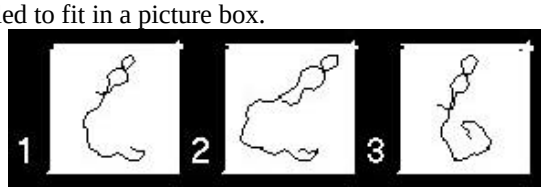


*Fig. 2: Visualization of three closely related genomes.*

When the different genomes are closely related, this is clearly visible (see fig. 2). Also, very often the graphics may have a resemblance to something figurative, which may aid your memory.

### 2.2.2 Communication and customization

All communication with MutaSynth and the sound engine, be it a hardware synth or another software, is by way of MIDI, using Midi Continuous Controllers (CCs) or System Exclusive Messages (Sysex). These messages can be customized to control virtually any hardware or software sound engine that supports MIDI remote control. This is done by creating a *profile* for that specific sound engine. A profile contains information on how to communicate with the synth, such as parameter-to-gene mapping, parameter ranges and communication strings.

It is sometimes useful to have several profiles for the same sound engine, biased towards different subspaces of the parameter space or, musically speaking, towards different sound types. This is easily done by enabling different parameters subsets and specifying different parameter ranges, mappings and default values.

## 3. Examples

In this section I will describe three different sound engines that I have used with MutaSynth, and discuss what kinds of sounds they can generate and how they can be explored. The examples are chosen to show the wide range of possible applications of this program.

When designing sound engines for MutaSynth, there are two ways to go. Either you take an existing sound engine of some sort, such as a stand-alone synthesizer, a softsynth or a granular software engine. Then you try to make a gene-to-parameter mapping that is relevant for the type of sound you want to produce, by carefully selecting parameters to be mapped and reasonable parameter ranges. Or, you build a sound engine from scratch, with the potential to generate the sounds you want, and probably many other sounds. This second approach is more flexible and open-ended, so I will take my examples from that category.

These examples are all implemented on the Nord Modular synthesizer (Clavia 1997), which is a stand-alone virtual modular synthesizer – basically a number of DSPs controlled by a computer editor. One can freely create and connect modules from a library of about 110 different types. This environment was chosen because it is quick and easy to use and does not put any load on the host processor, while allowing for quite complex synthesis and triggering techniques. Any parameter on any module can be assigned a MIDI Continuous

*Fig. 3: The sound engine 4Sine, as a screenshot from the Nord Modular editor. The knobs that are included in the genome are marked with white dots.*

This is basically four sine oscillators connected in a network, so that each oscillator is frequency modulated (FM) by a weighted sum of the others' output, and amplitude modulated by the same sum but with different

weights (the two upper rows of mixers). Each oscillator has its own amplitude envelope, which is controlled by a MIDI keyboard, as is the fundamental frequency. So this patch generates playable keyboard instruments.

To make the FM part a little bit more controllable, the frequency ratios are not arbitrary, but always integer multiples or fractions of the fundamental frequency. This is controlled by the “Partial Quantizer” (the fourth module from the top). The output is a mix of the four parts.

The levels of the upper mixers, the partial numbers, the FM amount, and the attack, decay and sustain of each envelope are mapped to genes, a total of fourty-four genes, each spanning the parameter’s whole range.

Sound examples 1a-1d show different instruments evolved in a few generations from random starting genomes. Only the final results are played.

Sound examples 2a-2h form a mutation sequence from a strange voice-like sound to a buzzing organ. Note how the attack of the sound is gradually changing from example 2e. The main timbre change happens between 2a and 2b.

## *3.2 GestureB*

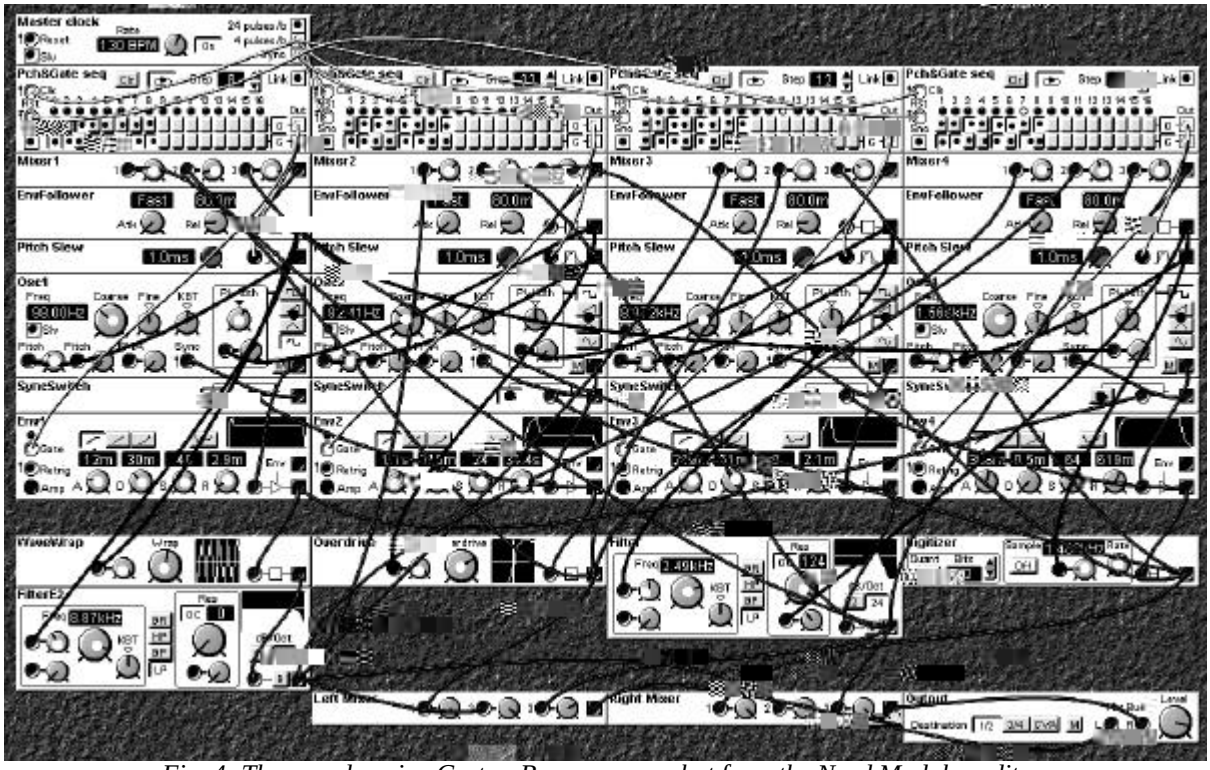


*Fig. 4: The sound engine GestureB, as a screenshot from the Nord Modular editor.*
*The knobs and buttons that are included in the genome are marked with black or white dots.*

This is a techno loop generator, basically made of four similar parts, each consisting of : an oscillator, an amplitude envelope, a two-track event sequencer where one track trigs the envelope and the other modulates the pitch of the oscillator, making it glide between two notes with a glide speed controlled by the Pitch Slew module. Each oscillator is optionally synced by the output of the part to the right.

Sound examples 3a-f show the wide range of techno patterns possible with this engine.

Sound examples 4a-j shows two different patterns and the results of a mating between them with a crossover probability of 0.1. Note how different elements of the structures make it into the offspring.

Sound examples 5a-j is another example of a mating operation.

## *3.3 Struct*

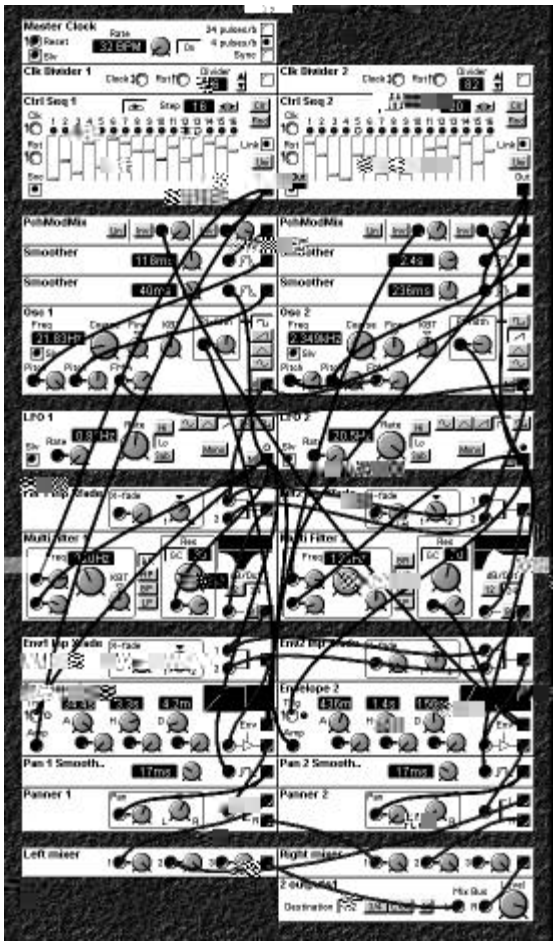

*Fig. 5: The Struct engine with all cables.*

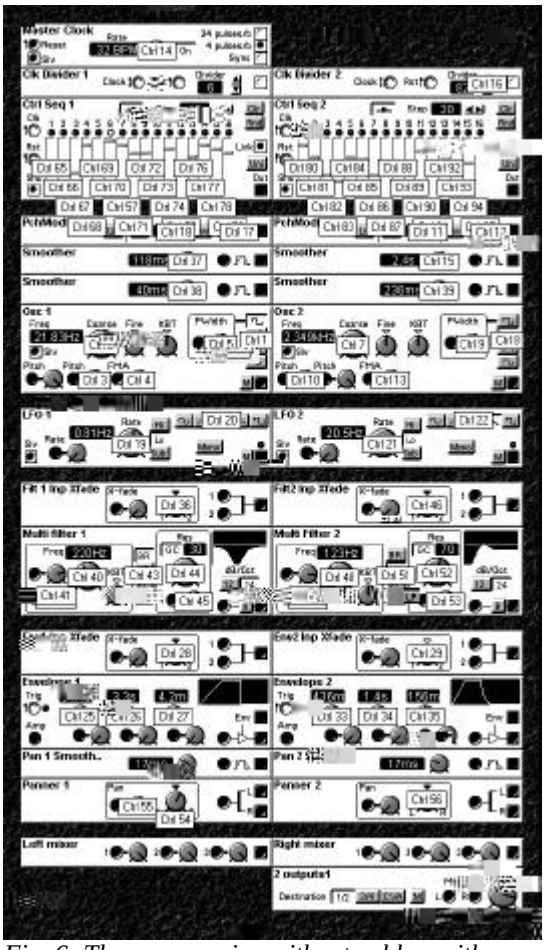

*Fig. 6: The same engine without cables, with gene mapping shown.*

The idea behind the design of Struct was to allow for widely different sonic gestures to be generated. Two low frequency oscillators (LFOs) and two control sequencers are used to modulate two oscillators and two filters, with rather intricate relationships. The pitch of each oscillator is modulated by one LFO, one envelope and one sequencer, in variable amounts. To make the timbres more interesting, the two oscillators have the possibility of modulating each other's frequency.

The cut-off frequencies of the two filters are similarly modulated by one sequencer and one LFO, in variable amounts. The filters take as input weighted sums of the two oscillators (controlled by the "Filt Inp Xfade" module just above the filter), and the envelopes/amplifiers take weighted sums of the two filters' outputs as inputs.

The oscillators can generate sine, triangle, up- and down-saw and square waveforms. In case it is a pulse wave, the pulse width is modulated by the envelope in the first oscillator and the LFO in the second. The filters can be of any of the basic types (low-pass, band-pass, hi-pass and band-reject). The output is a mix of the two modulated panners located at the bottom of the patch.

Many of the sounds generated from this engine are of the banal, very synthetic "computer game effects" type, but sometimes one stumbles across surprisingly organic and complex gestures and textures, and when these

interesting spots in parameter space are found, they often show the way to many good, related sounds, sometimes enough for a whole piece.

### 4.1 Material Generation

MutaSynth can be used on the sound level, for developing sound objects, loops, timbres and structures to be used in manual compositional work. In this case, it offers a way of investigating the possibilities of a certain sound engine. A musical advantage is that the material created during a breeding session often is audibly interrelated to a quite high degree, which opens up for compositions based on new kinds of structural relationships.

### 4.2 Interactive evolution of synthesizer sounds

MutaSynth can be used for programming almost any MIDI synth, without any knowledge about the underlying synthesis techniques. This depends on the existence of a suitable profile for the synth model in question, mapping the genome to a relevant parameter set in the synth. These profiles are quite easy to create, given the parameter set and communications protocol of the synthesizer in question.