$\mathbb{R}^2$m sound programming techniques

for version 0.1.0

Fritz Menzer

fritz.menzer@epfl.ch

20. 4. 2001

Contents

1 The basics

The natural parameters of a sound are the spectrum and the amplitude, so in this section will be examined how these parameters correspond to the parameters of $\mathbb{R}^2$m ($f(x,y)$, A_x, A_y, C_x, C_y, f_x, f_y). In a classic subtractive synthesizer (structure: oscillators $\rightarrow$ filter $\rightarrow$ amplificator) it is easy to say what determines what: the oscillators and the filter (with the corresponding envelope) determine the spectrum and how it changes in time while the amplificator (also with envelope) determines the amplitude of the sound.

With $\mathbb{R}^2$m this is more complicated as basically everything depends on all parameters[1]. This may make it more difficult to program sounds, but it is far more natural than classic subtractive synthesis. In a non-electronic music instrument the spectrum and the amplitude always vary together (for example if you hit a key on piano harder, the sound will not only get louder, but also more agressive, with more overtones).

1.1 Amplitude

The amplitude depends on A_x and A_y: if A_x and A_y are very small, the amplitude of the output signal will also be very small (depending on $f(x,y)$). The sound `spt_amplitude_Ax_Ay.xml` shows how one can use this dependency to make a fade-out in the release phase of the sound. Of course this can also be used in the beginning of the sound.

The amplitude depends on C_x and C_y: if the path $(x(t), y(t))$ moves to a region where $f(x,y)$ is almost flat, the amplitude of the output signal will become almost zero. This is illustrated in `spt_amplitude_position.xml` where C_x and C_y increase rapidly in the release phase of the sound in order to move the path to a region where $f(x,y)$ is almost zero.

Of course if the mapping function was different, the effect of a certain path could be completely different.

1.2 Spectrum

The spectrum depends on A_x and A_y: if A_x and A_y are very small, the spectrum of the output signal will be sine-like and if A_x and A_y become very big, the spectrum becomes very rich (or agressive) (depending on the mapping function $f(x,y)$). The sound `spt_spectrum_Ax_Ay.xml` shows how one can use this dependency to make a sound whose spectrum becomes more and more agressive while you keep a key pressed.

The spectrum depends on C_x and C_y: if the path $(x(t), y(t))$ moves in a region where $f(x,y)$ is changing very rapidly, the spectrum of the output signal will be very rich. This is illustrated in `spt_spectrum_position.xml` where C_x increases slowly in order to move the path from a region where $f(x,y)$ changes rapidly to one where it is more flat. This produces a sound with a spectrum that starts out quite agressively and becomes more and more sine-like in the end.

The spectrum also depends on the oscillators' frequencies f_x and f_y. These frequencies determine mainly the shape that the path $(x(t), y(t))$ has in one

[1]there is just one exception: the amplitude does not depend on the oscillators' frequencies f_x and f_y

period. For $f_x = 1$ and $f_y = 1$ (as in `spt_spectrum_fx_fy_0.xml`) this shape is a circle. For other values such that $f_x \neq f_y$ the shape will be a Lissajous shape. In general one can say that the higher the values for f_x and f_y are and the stranger the ratio between them, the more strange becomes the sound. Have a look for example on `spt_spectrum_fx_fy_2.xml` where $f_x = 5$ and $f_y = 6$. If f_x and f_y are not integers the shape will change from period to period. This can be seen in `spt_spectrum_fx_fy_1.xml` where $f_x = 1$ and $f_y = 2.001$.

The most important factor in $\mathbb{R}^2$m sounds is of course the mapping function. Some ideas on how to create interesting mapping functions are illustrated in the following section, but just to get an idea how the mapping function changes the sound, you could look at the examples `spt_spectrum_fxy_1.xml` and `spt_spectrum_fxy_2.xml` which have exactly the same path, but different mapping functions.

2 Advanced techniques

2.1 Imitating filters

Filters have the property of changing the spectrum (or the shape) of a signal. In synthesizers most oftenly lowpass filters are used to limit the highest overtones of the output signal.

For this manual I came up with two ideas on how to simulate lowpass filters with $\mathbb{R}^2$m. The first is to create a mapping function that is changing very rapidly in one region and very slowly in another. If you have a path that moves from the first to the second region, the sound will change from rich to sine-like, almost like a lowpass-filtered sound. A way to achieve such a mapping function is to make it depend on $\frac{1}{x}$. An example of this technique can be found in `spt_freq1.xml` where

$$f(x, y) = \sin\left(\frac{20}{x}\right)$$

In this case the mapping function is very "wild" around $x = 0$ and quite flat for $|x| \gg 0$. The nice thing about this method is that the sound automatically sounds a bit as if it was produced with a resonant lowpass filter.

The other idea is to create a function that permits to imitate a lowpass-filtered sawtooth or square wave with no resonance. A possible function is

$$f(x, y) = \frac{y}{\frac{x}{2} + |y|}$$

as in `spt_nores1.xml`. This function "jumps" from -1 to $+1$ when you move on the y-axis upwards over the origin $(0, 0)$. If you have a path that verifies $x > 0$ and in the beginning passes very close to the origin in vertical direction, the sound will be close to a sawtooth wave. If this path now moves away from the origin to the right ($\frac{dC_x}{dt} > 0$) then the waveform will be "smoothed" as by a lowpass filter.

2.2 Variable transformations

One way to create functions that depend on $\frac{1}{x}$ and are suitable for the first method described in the previous section is to take a periodic function (such as the functions named `fm2d*.xml`) and perform the variable transformation $x = \frac{c}{x'}$ where c is a constant. In the examples `spt_vartrans0.xml` and `spt_vartrans1.xml` I took the function

$$f(x, y) = \sin(x + y \sin(2x))$$

and performed the variable transformation $x = \frac{10}{x'}$ resulting in the function

$$f(x', y) = \sin\left(\frac{10}{x'} + y \sin\left(\frac{20}{x'}\right)\right)$$

3 A remark on `math.h`

When you introduce the mapping function you actually introduce a C expression as the mapping function is compiled using the file `./functions/addon.c`. In this expression you may use all the methods defined in `math.h`. In `math.h` almost all methods are defined in a `double` and a `float` version. As the float versions are supposed to be faster and most of the calculations in $\mathbb{R}^2$m are made in single precision (i.e. `float`) it would make sense to use the `float` versions, which are marked with an `f` at the end. Use for example `sinf()` instead of `sin()`.

To take a variable to a certain power, it is better to multiply it several times by itself. For example x^2 becomes `x*x`.