MULTICHANNEL SIGNAL REPRESENTATION IN PWGLSYNTH

Mikael Laurson, Vesa Norilo

Centre of Music and Technology
Sibelius Academy, Helsinki, Finland
{laurson|vnorilo}@siba.fi

ABSTRACT

This paper gives an overview of one of the most important features in our synthesis language called PWGLSynth. We will concentrate on how to represent visually multichannel signals in a synthesis patch. PWGLSynth synthesis boxes support vectored inputs and outputs. This scheme is useful as it allows to construct compound entities which are used often in sound synthesis such as banks, parallel structures, serial structures, etc. PWGLSynth provides a rich set of tools that allow to manipulate vectors. For instance vectors can mixed, modulated, merged, or split into sub-vectors.

1. INTRODUCTION

In sound synthesis it is essential to be able to represent complex flow of sound signals in a compact and comprehensible manner. One important abstraction mechanism is to combine several parallel signals into one multichannel signal that can be manipulated by the user as one entity. Thus it should be possible to create, initialize, delete, connect and disconnect a multichannel entity with one simple operation. Furthermore these operations should have a simple and uniform syntax that would allow to build complex synthesis patches that are both easy to understand and to maintain.

Although sound synthesis has already a fairly long history, synthesis environments that have appropriate multichannel abstraction mechanisms have only recently been appearing. Early textual implementations in the Music-N tradition have mostly dealt with mono signals. Some exceptional modules that have multichannel properties, such as a panning module, were typically found only at the output of an instrument definition. The lack of appropriate abstraction mechanisms obviously restricts severely the user in more complex cases. These restrictions are perhaps even more pronounced in visual synthesis environments resulting in patches that tend to become crowded and confusing when dealing with multichannel cases.

During the 90s the textual synthesis language SuperCollider [1] introduced a powerful "Multi channel expansion" property, which allows to use multichannel operations in a systematic way. This scheme resulted in an expressive system where patch definitions can be extremely compact and flexible.

This paper gives an introduction to our synthesis environment PWGLSynth [2][3] with an emphasis on the multichannel representation of signals. PWGLSynth is an integral part of our visual Lisp-based programming environment called PWGL [4] which is specialized in computer assisted composition, analysis and sound synthesis. Although our system has been influenced by SuperCollider our approach is quite different as we aim to represent patch definitions visually. The aim is to develop a system where by introducing appropriate abstraction mechanisms, such as multichannel representation of signals, visual definitions can become as economic and compact as it is the case in some of the state-of-the-art textual synthesis languages.

The rest of this paper is organized as follows. First we introduce the main concepts in our visual system and give some examples on how to define patches that deal both with 'traditional' mono signals and vectored signals. Next we go over to a more detailed discussion and introduce some important vector manipulation modules. The final section shows how multichannel operations are used in conjunction with our copy-synth-patch scheme that has been used extensively in our work related with physics-based instrument models [3].

2. BASIC VISUAL ENTITIES

A PWGLSynth patch is a is a graph structure consisting of boxes and connections. Boxes, in turn, can be categorized in two main box types from a synthesis point of view. The first box type consists of ordinary PWGL boxes. These boxes can be found at the leaves of a synthesis patch and they are typically evaluated once before the synthesis patch is run. A special case of this category are sliders which can dynamically change the current value while the synthesis is running. The second box type consists of boxes, marked with an 'S' , that represent synthesis boxes that are used for the actual sample calculation. 'S' boxes support vectored inputs and outputs. Mono signals are only a special case where the vector length is equal to 1. A synthesis patch always contains a special 'S' box, called 'synth-box', at the root of the graph, which represents the output of the sample calculation. This output can either be sent to audio converters in real-time, or the output can be written to a file.

Figure 1 shows a simple sine wave oscillator with vibrato control. The patch contains four sliders for real-time control and four 'S' boxes. This basic example operates only with mono signals.

The next patch example is more complex and it demonstrates how multichannel signals are represented in our system. The patch is based on vectored 'S' boxes (Figure 2). The patch gives also a visual clue that helps to distinguish between mono signals (vector length is equal to 1) and vectored signals. The connections between vectored boxes are drawn using a thicker line width and a stipple pattern that contains holes.

The vector *length* is specified by the inputs at the leaves (i.e. inputs which are not connected to a 'S' box) of the patch. These inputs can be Lisp expressions (typically lists) or slider-banks which allow a separate real-time control of each individual vector element. If the lengths at the inputs differ, then the shortest vector determines the current vector length. In Figure 2 the vector length is equal to 4, because the inputs of the 'sine-vector', 'impulse-vector', and 'reson-vector' contain lists or sliders with 4 elements.

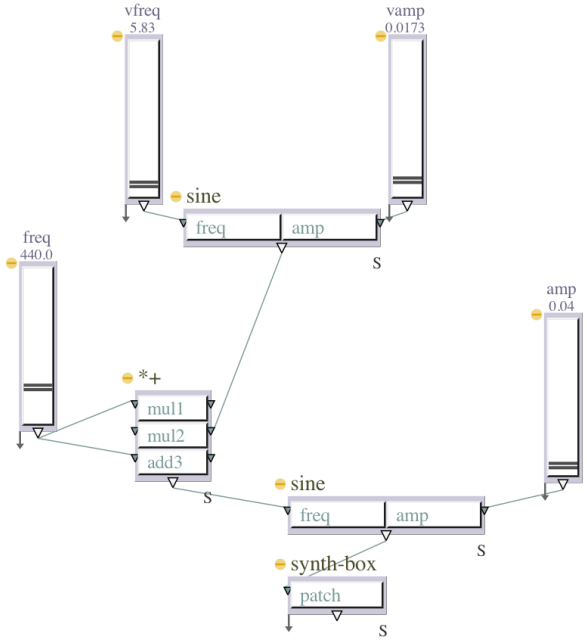

Figure 1: *A simple mono signal patch with some real-time sliders.*

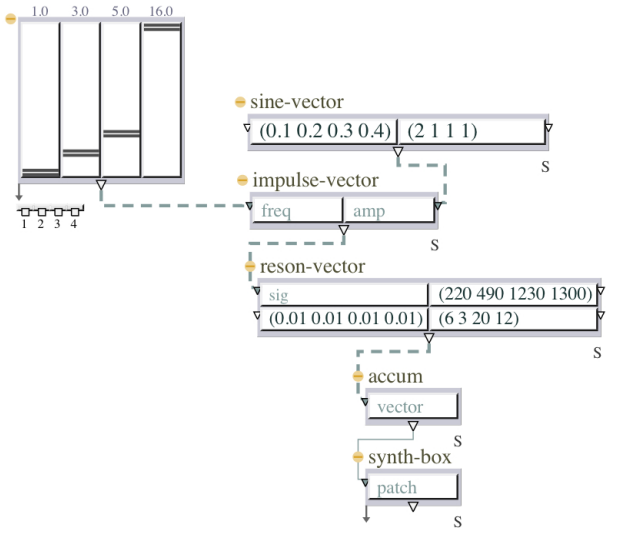

Figure 2: *A bank of 4 resonators that are excited by a bank of impulse generators.*

Thus the patch in Figure 2 can be interpreted as follows. A bank of 4 resonators ("reson-vector') is excited with a bank of 4 impulse generators ("impulse-vector'). The frequency input of the 'impulse-vector' box is controlled with a slider-bank, and the amplitudes of the impulses are controlled with a bank of sine wave oscillators ("sine-vector'). The slider-bank input of the 'freq' input of the 'impulse-vector' is of special interest as it allows to control in real-time the frequency of each impulse generator individually. The other inputs of the 'reson-vector' box (i.e. frequencies, amplitudes, and bandwidths) are given as static Lisp lists, each containing 4 elements. The output of the 'reson-vector' box is connected to a 'accum' box that accepts as input any vectored signal and mixes it to a mono signal (thus the final output is a mono signal).

The patch example in Figure 3 demonstrates how the PWGL environment can be used to calculate input values for vectored 'S' boxes. The two first inputs, 'low' and 'high', of the 'randi-vector' box contain special PWGL shorthand expressions, '(14*(0.995))' and '(14*(1.005))', for generating lists (here we get 2 lists of 14 elements consisting of the values 0.995 and 1.005). The third input of the 'randi-vector' box, called 'freq', is connected to an 'interpolation' box, that returns a list of 14 values (the result of interpolating values from 5.0 to 20.0). Thus the vectored output of the 'randi-vector' has 14 elements which are fed to the first input of a 'mul-vector' box. The second input of the latter box is connected to a standard PWGL 'value-box' that returns a list of 14 frequency values (only the first values are visible in the figure). The output of the 'mul-vector' box consists of 14 frequency values where each value is individually modulated by an interpolating random number generator. This output is connected to the 'freq' input of a 'reson-bank' module. Like 'reson-vector' given in the previous example, 'reson-bank' is a bank of resonators. The first input is different, however, as it accepts only a mono signal (here a simple impulse) instead of a vectored input. The other inputs of the 'reson-bank' module, amplitudes and bandwidths, are lists of 14 values (again only the beginning values are visible). Finally, the output of the 'reson-bank' box is mixed to a mono signal with a 'accum' box, exactly as was done in the previous example.

3. VECTOR MANIPULATION

In this section we discuss some of the synthesis modules that allow to manipulate vectors. In the simplest case *arithmetic* operations can be applied either to two vectors, or to a mono signal and to a vector. Figure 3 gave already an example of this kind of a box, where the 'mul-vector' box multiplied its input vectors resulting in a vector of modulated signals. Similar boxes, called 'add-vector' (see Figure 5), 'sub-vector', and 'div-vector', allow to add, subtract and divide input vectors.

Vectors can be *combined* to a single vector using a box called 'combiner', that can have an arbitrary number of inputs. A typical application of this box is when the user wants to combine several mono boxes so that the resulting vector can be fed to a vectored box. Thus in Figure 4 a 'combiner' box combines two mono random number generators, and the resulting vector is fed to an 'impulse-vector' box. This vector is in turn mixed to a mono signal resulting in a stream of 7 impulses per second where every seventh impulse is strongly accented (see the first 'freq' input of the 'impulse-vector' box containing the list '(1 7)').

Our next example shows how vectors can be *split* into subvectors by using the 'indexor' box (Figure 5). This box has 3 inputs, 'vector', 'index', and 'len'. The starting point is a bank of 29 resonators. The vectored output is split into two sub-vectors so that vector elements 0-15 (the indexing starts from 0) form the first sub-vector (see the 'indexor' box to the left), and the remaining elements form the other sub-vector (the 'indexor' box to the right). After this both sub-vectors are mixed to 2 mono signals, which are in turn fed to 2 spatialization boxes, called 'vbap2d'. A 'vbap2d' box implements a 2-dimensional version of the Vec-

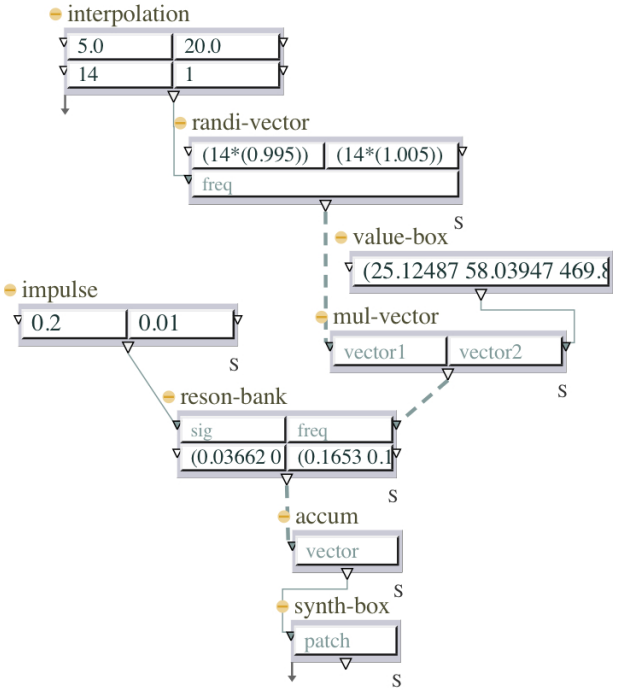

Figure 3: *A bank of 14 resonators where the frequencies are modulated by a bank of interpolated random number generators.*

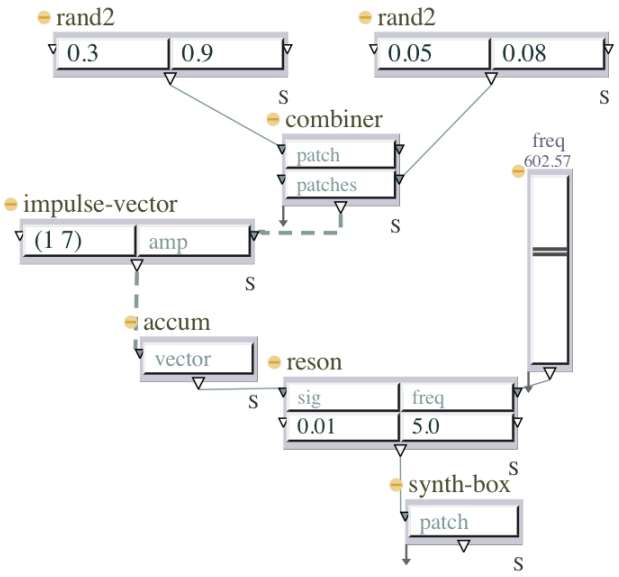

Figure 4: *Combining 2 impulse streams to excite a single resonator.*

tor Base Amplitude Panning (VBAP) algorithm [5]. Here the first input is panned in a 2-dimensional space according to the second input, called 'azim'. The third input defines the speaker configuration to be used by the system. This input has 4 speaker positions and thus the patch results is a 4-channel output.

4. VECTORED SIGNALS COMBINED WITH THE COPY-SYNTH-PATHC SCHEME

In our final section we discuss some more complex cases that combine vectored signals with a copying scheme for patches [3]. One problem with the system that was presented in the previous sections is that typically one needs box versions for both mono and vectored cases (e.g. 'sine' and 'sine-vector'). Next we introduce a scheme that allows to combine any collection of mono or vectored boxes. This collection can be copied and the output of the result will be one or several vectored signals. We use for this purpose a special box called 'copy-synth-patch' with 2 required inputs, 'count' and 'patch'. A third, optional, input can be given for a name string. A 'copy-synth-patch' box duplicates a patch connected to the 'patch' input count times. The output of the box is a vector having the length which is determined by the count input (internally the system uses the 'combiner' box discussed above to achieve this result).

Figure 6 gives a simplified overview of a guitar model that copies a 'string' abstraction 6 times. Note that the upper part of the figure shows only a part of the content of the 'string' abstraction. The output of the 'copy-synth-patch' box is a 6-element vector that is mixed to a mono signal by the 'accum' box.

To demonstrate some of the possibilities when combining the vectored-based approach with the copy-synth-patch scheme we add to the output of the string abstraction a 'vbap2d' box with a 4-channel speaker configuration (see the upper part of Figure 7). Note that as the output of the abstraction has changed from a mono signal to a vectored one with 4 channels, the output of the 'copy-synth-patch' box is now a vector of 24 elements (6 strings * 4 channels). In order to get a 4-channel output we mix the 24 element vector to a 4-element vector using a special vector manipulation box called 'accum-vector'. This box is a vectored version of the 'accum' mixer box that has been used in the previous examples. The difference is that 'accum-vector' translates an input vector internally to a matrix, where the number of columns is determined by the second input, called 'len' (in our case equal to 4). The number of rows is gained by dividing the number of elements of the input vector by the 'len' input (in our case 24/4 = 6). After this the box mixes all columns resulting in the final output vector.

5. CONCLUSIONS

This paper gave an overview of some multichannel capabilities of our visual synthesis environment. The system can be used to effectively to realize complex DSP oriented tasks such effects, reverbs and filter banks. The scheme has also proven to be useful when using it in conjunction with our copy-patch-scheme to realize instrument models.

A PWGL beta version that includes our synthesis environment can be loaded from the following homepage:
http://www.siba.fi/PWGLhttp://www.siba.fi/PWGL

6. ACKNOWLEDGEMENTS

The work of Mikael Laurson and Vesa Norilo has been supported by the Academy of Finland (SA 105557).

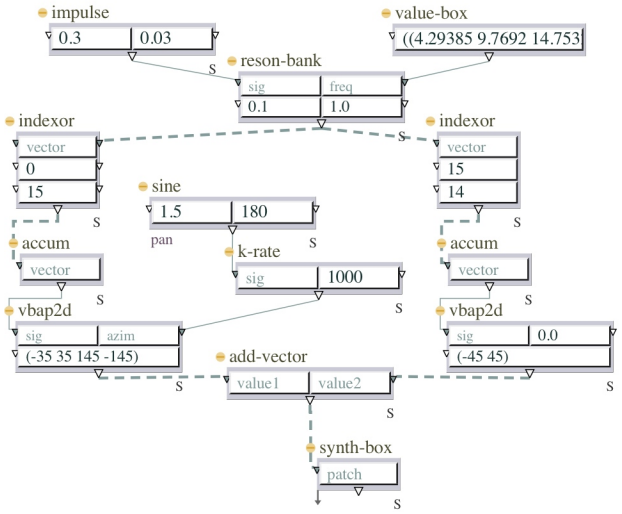

Figure 5: *A bank of 29 resonators where the output vector is split into two sub-vectors that are panned individually.*

7. REFERENCES

[1] J. McCartney, "Continued evolution of the Super-Collider real time environment," in *Proc. Int. Comp. Music Conf. (ICMC'98),* Ann Arbor, USA, 1998, pp. 133–136.

[2] M. Laurson and V. Norilo, "Recent developments in PWSynth," in *Proc. Int. Conf. on Digital Audio Effects (DAFx-03),* London, UK, 2003, pp. 69–72.

[3] M. Laurson, V. Norilo, and M. Kuuskankare, "PWGLSynth: A visual synthesis language for virtual instrument design and control," *Computer Music J.*, vol. 29, no. 3, pp. 29–41, Fall 2005.

[4] M. Laurson and M. Kuuskankare, "PWGL: A novel visual language based on common Lisp, CLOS and OpenGL," in *Proc. Int. Comp. Music Conf. (ICMC'02),* Gothenburg, Sweden, 2002, pp. 142–145.

[5] V. Pulkki, "Virtual source positioning using vector base amplitude panning," *J. Audio Eng. Soc.*, vol. 45, no. 6, pp. 456–466, June 1997.

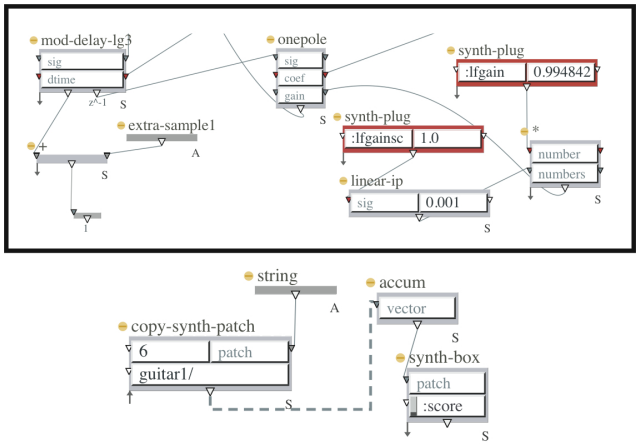

Figure 6: *A guitar model consisting of 6 strings. Top: part of the 'string' abstraction. Bottom: a 'copy-synth-patch' box duplicates the 'string' abstraction 6 times.*

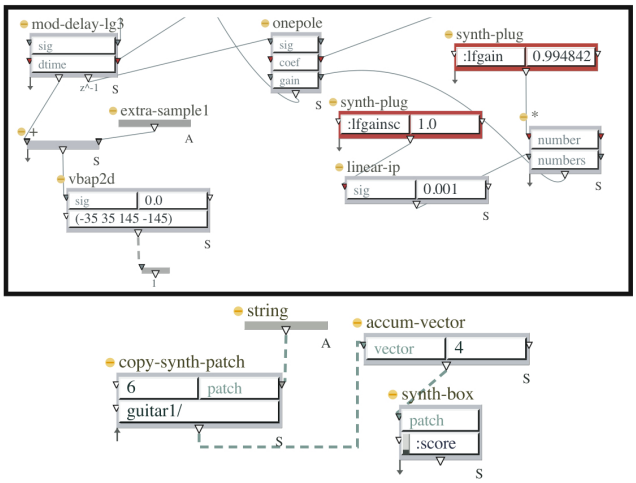

Figure 7: *A modified 4-channel guitar model that allows to pan each individual string separately.*