A VERSATILE TOOLKIT FOR CONTROLLING DYNAMIC STOCHASTIC SYNTHESIS

Gordan Kreković
Faculty of Electrical Engineering and Computing,
University of Zagreb, Croatia
gordan.krekovic@fer.hr

Davor Petrinović
Faculty of Electrical Engineering and Computing,
University of Zagreb, Croatia
davor.petrinovic@fer.hr

ABSTRACT

Dynamic stochastic synthesis is one of the non-standard sound synthesis techniques used mostly in experimental computer music. It is capable of producing various rich and organic sonorities, but its drawback is the lack of a convenient approach to controlling the synthesis parameters. Several authors previously addressed this problem and suggested direct parameter control facilitated with additional features such as parameter automation. In this paper we present a comprehensive toolkit which, besides direct control, offers several new approaches. First, it enables controlling the synthesizer with an audio signal. Relevant audio features of an input signal are mapped to the synthesis parameters making the control immediate and intuitive. Second, the toolkit supports MIDI control so that musicians can use standard MIDI interfaces to play the synthesizer. Based on this approach we implemented a polyphonic MIDI-controlled synthesizer and included it in the toolkit along with other examples of controlling the dynamic stochastic synthesizer. The toolkit was developed in the widely used visual programming environment Pure Data.

1. INTRODUCTION

The usefulness of a sound synthesizer in practical tasks concerning musical composition depends not only on its capability to produce desired sonorities, but also on different aspects of its technical implementation [1]. Such aspects are, for example, suitability for a given hardware and software environment, intuitiveness of the user interface, flexibility in controlling the synthesis process, and many others. Nowadays composers have a wide range of possibilities when choosing sound synthesizers for their compositions.

Most well-known synthesis techniques have been implemented in various forms: as hardware synthesizers, software plugins, patches for music-specific programming languages, and applications for mobile devices. Interfaces for musical expression and parameter automation ensure convenient control over the synthesis parameters. Modern tools for sound synthesis generally open numerous opportunities in creating novel sonorities and successfully follow the growing ambitions of computer musicians.

However, there are still some insufficiently explored, yet interesting sound synthesis techniques which could widen the possibilities of musical expression, but have not yet been adapted for practical usage. One such example is dynamic stochastic synthesis devised by Iannis Xenakis in the early 1970s. This synthesis technique is characterized by distinctive and rich timbral qualities. Nevertheless, a convenient solution for controlling the synthesis parameters is still missing. We believe that the lack of an intuitive control is one of the reasons why this technique has not been employed in a larger number of compositions or further explored.

Dynamic stochastic synthesis (DSS) produces a waveform by interpolating a set of constantly varying breakpoints [2]. The waveform evolves over time in a nondeterministic manner which results in organic and complex sonorities. Composers can control the DSS process by restraining ranges, within which the waveform can change, and by specifying amounts and probability distributions of those changes. The problem is that manipulating the aforementioned ranges, amounts, and parameters of probability distributions is usually inconvenient for most practical tasks. Such synthesis parameters are not intuitive and do not allow the use of typical musical interfaces for playing. Moreover, the original implementation of the dynamic stochastic synthesizer did not even provide any kind of support for changing parameters during the synthesis process.

Several authors have already addressed the same problem and proposed various interface designs for direct parameter control [3-5]. They suggested graphical user interfaces, keyboard shortcuts, and MIDI controllers. One standout solution was a mobile application which obtained parameters from multi-touch gestures and accelerometers [6]. Even though these interfaces were straightforward and helpful, musicians still needed to cope with values of the synthesis parameters. To avoid numerical parameters and keep ideas in the musical domain, in our previous research we proposed an approach that uses an input audio signal for controlling the DSS process [7]. The algorithm was based on mapping selected audio features into the synthesis parameters, so that the control was as intuitive as possible.

The research described in this paper takes a few steps further in making DSS more suitable for the practical needs of computer musicians. Several approaches to controlling synthesis parameters were developed and

packaged together in a comprehensive toolkit for the visual programming language Pure Data [8]. Besides direct parameter control and control using an audio signal, we introduced a new approach based on MIDI notes and controllers. This novel approach allows musicians to play a dynamic stochastic synthesizer using regular MIDI interfaces. The toolkit was designed so that it can be easily modified, extended, and integrated in compositions.

2. DYNAMIC STOCHASTIC SYNTHESIS

Before presenting the toolkit for controlling dynamic stochastic synthesis, here is a short overview of this synthesis technique. Dynamic stochastic synthesis was devised by Iannis Xenakis as a result of his ambition to achieve unified and simultaneous engagement on different time-scales within the composition, from the overall structure of the composition to its microstructure and tone quality. Before this breakthrough, he employed stochastic processes for choosing note attributes and forming musical structures. To expand the same principle on the microstructure level, Xenakis suggested applying stochastic processes to the sample level.

Dynamic stochastic synthesis generates samples by interpolating a set of breakpoints which change their amplitudes and positions in time stochastically. A breakpoint position is represented relatively to the preceding breakpoint in number of samples, so it is commonly called breakpoint duration. Initial amplitudes and durations are usually chosen randomly or taken from a trigonometric function. At every repetition of the waveform, these values are varied independently of each other using random walk. That means that both the amplitude and the duration of a certain breakpoint are changed by adding random steps to the values in the previous cycle as shown in Figure 1. A succession of random steps applied on all breakpoints causes the continuous variation of the waveform. The amount and character of the variation depend on a selected probability distribution and its parameters. Both amplitude and duration random walks are limited each with two reflecting barriers which bounce excessive values back into the predefined range. These barriers prevent breakpoints from going too far from their initial positions and therefore enable control over amplitude and frequency ranges of the overall waveform.

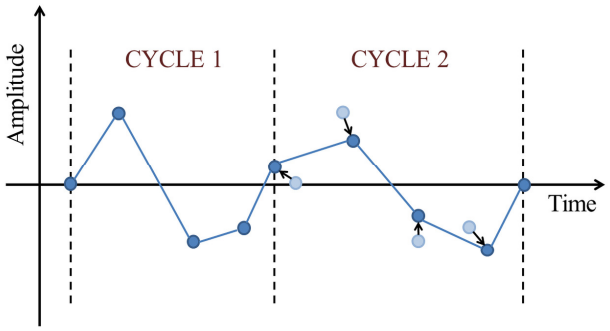

Figure 1. Breakpoints change their positions from one repetition to another. Light blue circles in the second represent positions from the first cycle, whilst darker circles represent new positions.

Parameterization of the algorithm is achieved through: (1) the number of breakpoints in a waveform, (2) barriers of the amplitude random walk, (3) probability distribution of the amplitude random walk and its parameters, (4) barriers of the duration random walk, and (5) probability distribution of the duration random walk and its parameters. The amplitude barriers provide control over the amplitude range of the generated waveform, whilst the duration barriers define minimal and maximal number of samples between two breakpoints. If changes in amplitude and duration in successive repetitions are small, the synthesized sound is relatively simple, but it can have interesting modulation effects. On the other hand, as the changes become larger, the sound becomes more complex and noisier. Detailed explanations of the original algorithm can be found in [9] and [10]. Several computer musicians later implemented this algorithm extending the basic concept with new ideas [3, 5, 11, 12].

3. TOOLKIT FOR PURE DATA

The motivation while developing this toolkit was to make DSS available to a wider community of computer musicians. Also, by providing several interfaces for controlling the DSS process, we wanted to bring this non-standard synthesis technique closer to the practical needs of composers and live performers. For implementation we chose Pure Data, a visual programming language which is freely available for different operating systems and which is popular among musicians and multimedia artists [8]. All parts of this library were developed as abstract patches, so that everyone familiar with Pure Data can easily modify and extend them.

3.1 gendyn~

The central patch in the toolkit is a straightforward implementation of the basic DSS algorithm. It was named *gendyn~* after the original program by Xenakis. The purpose of this patch is to synthesize audio signal accordingly to input parameters. Through the inlets it receives the number of breakpoints in a waveform n, frequency limits $f_{\min}$ and $f_{\max}$, amplitude range a, and statistical parameters for the both random walks p_1 and p_2.

Frequency limits $f_{\min}$ and $f_{\max}$ are used to calculate barriers of the duration random walk. Frequency limits expressed in Hertz are more meaningful then duration limits expressed in number of samples. They are also more convenient for direct integration with patches that provide DSS process control using audio or MIDI signals. For that reason, *gendyn~* receives frequency limits through the inlets and converts them to the duration limits using these simple formulae:

$$d_{\min} = \lceil f_S / (f_{\max} \cdot n) \rceil, \quad (1)$$

$$d_{\max} = \lceil f_S / (f_{\min} \cdot n) \rceil, \quad (2)$$

where d_{min} and d_{max} are the maximal and the minimal duration expressed in number of samples, f_S is the sampling frequency, f_{max} and f_{min} represent the frequency

limits, whilst n stands for the number of breakpoints in a waveform.

The amplitude range of the waveform is controlled with the parameter a so that the amplitude random walk has reflecting barriers at $-a$ and $+a$. Therefore, this parameter defines the maximal absolute amplitude of the breakpoints.

The only probability distribution available in our current implementation is the normal distribution. Its mean value for both random walks is zero, because symmetrical probability densities generally prevent breakpoints from gravitating towards one of the barriers. The standard deviation of the distribution for the amplitude random walk is calculated by scaling the parameter p_1 proportionally to the amplitude range a. Similarly, for the duration random walk, its input parameter p_2 is scaled accordingly to the range contained between minimal and maximal duration. Extending the patch with more probability distributions is simple, but requires adding a new parameter for selecting among available distributions.

3.2 audio2gendyn~

An approach to controlling the dynamic stochastic synthesis with an audio signal was proposed in our previous work [7]. The purpose of that research was to reduce the need for manipulating numerical parameters and to allow musicians to control a synthesizer by playing a musical instrument, singing, or experimenting with different sound sources. The algorithm was designed to extract relevant audio features from the input signal and map them to the synthesis parameters so that the relation between the input signal and the synthesized signal is as natural as possible.

As our original implementation version was done in C++, for the Pure Data toolkit we developed a new patch from scratch and also introduced several improvements and simplifications. This new patch is called *audio2gendyn~* and uses features of the input audio to calculate synthesis parameters. The synthesis engine *gendyn~*, which is included in this patch, receives these parameters and produces the resulting sound accordingly. The amplitude, frequency, and timbral qualities of the synthesized sound are expected to follow the corresponding characteristics of the input audio signal. The aim was not to imitate the input signal (as it is not possible with DSS anyway), but to achieve intuitive control over the synthesis process.

The most appropriate audio features of the input signal for calculating the frequency limits f_{min} and f_{max} are fundamental frequency f_0 and spectral centroid f_C. Whilst for periodic signals the fundamental frequency works well, for noisy signals much better results are obtained by using the spectral centroid. Spectral centroid indicates the center of the gravity of a frequency spectrum and it is perceptually related to the impression of timbral brightness.

In case of the periodic input signal, the fundamental frequency is extracted using the object *sigmund~* which is one of the standard Pure Data extras. The frequency limits $f_{\min}$ and $f_{\max}$ are then defined as a perfect fifth below and a perfect fifth above the fundamental frequency, i.e.

$$f_{\min} = 2f_0/3\,, \quad f_{\max} = 3f_0/2\,. \tag{3}$$

In contrast to our initial algorithm [7], here the frequency limits are strictly related to the fundamental frequency by the given musical intervals (i.e. frequency ratios). Timbral qualities of the input sound are not considered for determining the frequency limits. The advantage of this simplification is that the frequency of the overall synthesized waveform depends only on the fundamental frequency of the input signal and never drifts too far from it. However, timbral qualities of the input signal are not neglected here; they affect the standard deviation of the probability distribution for the duration random walk as will be described later.

If the input signal does not show significant periodicity, the spectral centroid is used similarly as the fundamental frequency in the earlier case. First, the spectral centroid f_C is calculated using the object *specCentroid~* from *timbreID* toolkit [13]. Then the frequency limits are defined as:

$$f_{\min} = f_C/8\,, \quad f_{\max} = f_C/4\,. \tag{4}$$

The scaling factors were obtained experimentally so that switching between periodic and non-periodic input signals does not cause unpleasant glitches in the synthesized signal. These factors were chosen after numerous tests with different types of sounds including those with both periodic and non-periodic parts such as speech signals and sounds of plucked instruments.

Defining the barriers for the amplitude random walk was a much simpler task. The amplitude of the synthesized signal is expected to follow the amplitude of the input signal, so the algorithm uses the root mean square amplitude of an input frame to control the parameter a.

Finally, the only remaining parameters are p_1 and p_2. Standard deviations of the probability distributions in random walks significantly affect timbral qualities of the synthesized sounds. Wider probability density functions result with a less stable waveform and consequently less predictable frequency content of the synthesized signal. For that reason, the parameters p_1 and p_2 should be defined accordingly to the level of how tone-like the input sound is, as opposed to being noise-like. A suitable measure for this purpose is spectral flatness [14]. This feature is one of audio descriptors in the MPEG-7 standard and it is commonly used for robust retrieval of song archives. Spectral flatness quantifies amount of peaks or resonant structure, as opposed to the flat spectrum of white noise. A low flatness suggests that the spectral power is concentrated in a small number of spectral bands, whilst higher values indicate that the power is more equally distributed among all bands. The spectral flatness is defined as a quotient of the geometric and the arithmetic mean of the power spectrum, i.e.

$$S_F = \frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\frac{1}{N}\sum_{n=0}^{N-1} x(n)}\,, \tag{5}$$

where $x(n)$ stands for the magnitude of the n-th frequency bin.

To calculate the spectral flatness in *audio2gendyn~* we employed the object *specFlatness~* from *timbreID* toolkit [13]. The scaled spectral flatness is then used for the both parameters p_1 and p_2:

$$p_1 = p_2 = s \cdot S_F \,, \tag{6}$$

where s stands for a scaling factor and S_F denotes the spectral flatness. Many subjective tests proved the suitability of such mapping. The value of the scaling factor was obtained experimentally so that the character of the synthesized signal is notably affected by the spectral flatness of the input signal.

3.3 midi2gendyn~

The second solution for controlling DSS included in this toolkit is based on the standard MIDI interface. The usage of MIDI controls was suggested earlier [5], but only for direct parameter control. The musician could manipulate parameters with a MIDI controller and send values to the dynamic stochastic synthesizer in the same way as if using a graphical user interface. Evidently, this was not a different approach to control, but only facilitation.

Most sound synthesizers can be played with MIDI keyboards and other MIDI interfaces which generate notes and not just control values. To apply this traditional playing approach to DSS, we implemented *midi2gendyn~*. It is the first polyphonic MIDI synthesizer based on DSS. The patch receives MIDI notes, velocities, and other controls, maps them into synthesis parameters, and employs sound units based on *gendyn~* to generate the sound.

To determine frequency limits f_{min} and f_{max} from the input note, the algorithm converts the MIDI note number into the frequency and puts the limits symmetrically around it. This way, the frequency of the input note is in the middle between f_{min} and f_{max}. The width of that frequency range is specified with a separate MIDI control value. This approach is convenient in practical cases as the musician can play the synthesizer using a keyboard and simultaneously change the frequency width using a slider, knob, or pedal.

The amplitude range a is calculated by scaling the note velocity, whilst the parameters p_1 and p_2 are separately obtained from corresponding MIDI controls. The synthesizer also receives the pitch bend control which affects the tone frequency and therefore the frequency limits f_{min} and f_{max} accordingly.

4. EXPERIMENTS AND EXAMPLES

The patches from this toolkit can be used in different ways. For that reason we prepared several typical usage examples and included them in the package. Those examples can be reused, modified, and extended to meet specific practical needs.

4.1 Direct control and automation

The first two examples show how a dynamic stochastic synthesizer can be controlled by direct parameter manipulation. In the first example, sliders on the graphical user interface are connected to the inlets of *gendyn~* (Figure 2). These sliders also receive MIDI controls, so that they can be managed from a MIDI interface with physical sliders or knobs. Audio effects can be applied on the pure audio signal synthesized by a dynamic stochastic synthesizer. In these examples we added a simple reverb, which was very efficient in making the sound richer and characteristically colored.

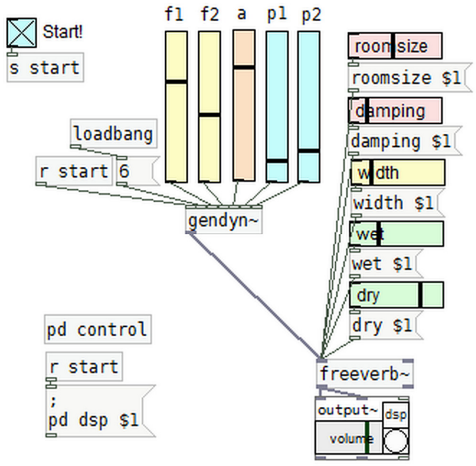

Figure 2. A patch which demonstrates direct parameter control using sliders. Beside sliders on the graphical user interface, it is possible to use MIDI controls defined in the subpatch called *control*.

The second example of direct parameter control demonstrates parameter automation (Figure 3). The patch reads parameter values from tables. As Pure Data supports drawing values on graphical representations of tables, such automation could be convenient both for composing and live performing.

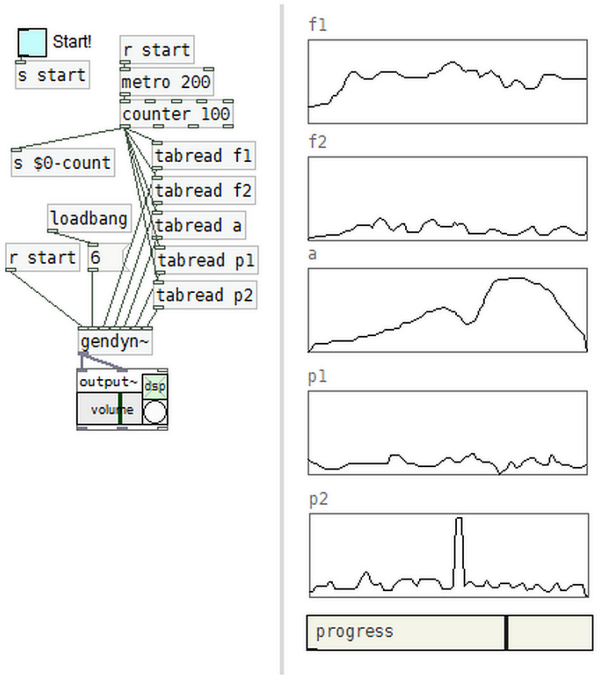

Figure 3. An example of parameter automation. Parameter values are stored in the tables on the right side.

4.2 Examples of audio control

Experimenting with different types of sounds for controlling the DSS showed that this synthesis technique can produce more than just buzzing sounds with characteristic drifts in frequency and amplitude. Mapping relevant audio features into the corresponding synthesis parameter enables the synthesizer to mimic some characteristics of the input sound. It is easier to make simultaneous and quick changes in parameter values than by direct parameter manipulation. The most interesting sounds for controlling the DSS are those with high variability of their audio features such as percussive sounds and human voice.

Within the toolkit we provided two examples of receiving an audio signal for controlling the DSS. The first one uses inputs from the audio interface, whilst the second one reads a wave file as shown in Figure 4.

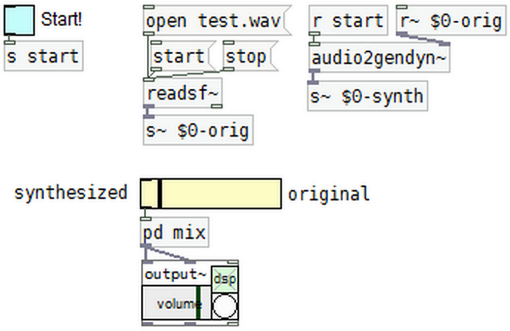

Figure 4. Controlling the dynamic stochastic synthesis with an audio signal from a wave file. It is possible to mix the synthesized and the original signals using the yellow slider.

4.3 Polyphonic MIDI-controlled synthesizer

To test the *midi2gendyn~* patch we used a MIDI keyboard. Phenomena which most strongly affected the playing experience were frequency drifts. They always occur when the frequency range f_{max} - f_{min} and duration standard deviation obtained from the parameter p_2 are higher than zero. Changes in the waveform frequency are characteristic to DSS and result with buzzing, unstable and drifting sounds. One of the possible applications of such sounds in compositions is to layer them with the sounds generated by other synthesis engines.

Demonstration of the toolkit and highlights from all of the mentioned experiments are shown in the video which is available at the following link:
http://www.youtube.com/watch?v=1Uk6KeglvnI

5. CONCLUSIONS

By implementing several different approaches to controlling DSS in a single toolkit, we made the synthesis technique more convenient for particular use cases. This should motivate musicians to experiment further in their compositions and live performances. Since it is a non-standard synthesis technique, we cannot expect DSS to suddenly become popular in a wider range of music genres even when researches like this one are available. However, it is now more accessible to musicians than it was before and it is ready to be used in numerous ways.

6. REFERENCES

[1] D. A. Jaffe, "Ten Criteria for Evaluating Synthesis Techniques". *Computer Music Journal*, vol. 19, no. 1, pp. 76–87, 1995.

[2] I. Xenakis, *Formalized Music: Thought and Mathematics in Music*, Stuyvesant NY: Pendragon Press, 1992.

[3] P. Hoffman, "The new GENDYN program", *Computer Music Journal*, vol. 24, no. 2, pp. 31-38, 2000.

[4] S. Bokesoy and G. Pape. "Stochos: software for real-time synthesis of stochastic music", *Computer Music Journal*, vol. 27, no. 3, pp. 33-43, 2003.

[5] A. R. Brown, "Extending dynamic stochastic synthesis", *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005, pp. 111-114

[6] N. Collins, "Implementing stochastic synthesis for SupperCollider and iPhone". *Proceedings of the Xenakis International Symposium*, London, 2011.

[7] G. Kreković, I. Brkić, "Controlling Dynamic Stochastic Synthesis with an Audio Signal", *Proceedings of the International Computer Music Conference*, 2012, pp. 100-104.

[8] M. Puckette, "Pure Data", *Proceedings of the International Computer Music Conference*, San Francisco, USA, 1996.

[9] M. H. Serra, "Stochastic composition and stochastic timbre: GENDY3 by Iannis Xenakis", *Perspectives of New Music*, vol. 31, no. 1, pp. 236-257, 1992.

[10] S. Loque, "The stochastic synthesis of Iannis Xenakis", *Leonardo Music Journal*, vol. 19, no. 1, pp. 77-84, 2009.

[11] S. Russell. (2012). *Gendyflext* [Online]. Available: https://github.com/ssfrr/gendyflext

[12] J. Young, "Rethinking synthesis: extending and exploring Gendyn", BA thesis, University of Sussex: Department of Informatics, 2010.

[13] W. Brent, "A timbre analysis and classification toolkit for Pure Data", *Proceedings of the International Computer Music Conference*, New York, USA, 2010

[14] J. Johnston, "Transform coding of audio signals using perceptual noise criteria", *IEEE Journal on Selected Areas in Communications,* vol. 6, no. 2, pp. 314-323, 1988.