Introduction

This paper is a definition of a multi-step semi-modular system for computer synthesis using prepared audio data inputs. It is intended for live performance and/or recorded synthesis, and uses quasi-random (and, rarely, random) stochastic algorithms to create an unpredictable aleatoric function that can produce an ongoing sonic output that never, or rarely, falls into a state of repetition.

The system described was originally constructed in the visual programming software stack Pure Data, and is composed of a series of segments which can be used as standalone functions but which, in series, can be used together to create a versatile and dynamic system for sound synthesis.

The initial segment features a central read function composed of two sawtooth waveforms, one which reads the data entries in a set and the other which corrects the phase of the reading waveform. This function was initially conceived as an attempt at emulating the functionality of the **2d.wave~** object in Max/MSP, which offers its own versatile set of parameters for synthesis. However, the mechanism between these two functions differs greatly and the two offer contrasting use cases and results.

The **2d.wave~** object seemingly works by duplicating an audio buffer into a set of evenly-sized buffers (called "rows") of identical length which each contain a defined fraction of data from the original buffer. These rows are then controlled by two signal inputs, usually sawtooth waveforms, though different ones are also useable. One of these inputs determines the position in the respective row that is being read by the function, while the other input specifies the position over the series of rows that is accessed. This is best visualized as a multiscale, two-dimensional function, the small-scale parameter affecting the sample-level playback of the audio and the large-scale parameter affecting the row-scale position, i.e. which row out of the set is being read at the given value of time. Each row is enveloped by a Hanning function with a hop length of some seemingly undisclosed proportion of the row's total size. The exact nuances of its mechanism are obscure due to its proprietary nature, though this description is based on official documentation provided inside Max/MSP, as well as extensive experimentation with the object.

There are a number of differences between this and the method defined in this paper:

1. There is only one instance of the source dataset in RAM for each running instance of the function.
2. There is no Hanning envelope over played intervals, nor is there any overlap.
3. The dataset is not divided into evenly sized rows; rather than calculating a defined fraction of the entire set, the position incriments of the function are based on a simple calculation of phase difference between two waveforms.
4. Because of the lack of windowing and overlapping, the function can be used to produce a different variety of modulations on the source data, which shall be described further in this paper.

The following segments of this algorithmic system are more modular, and can be rearranged and toggled as needed. Though they are less rigidly integrated than the initial segment of the system, they are presented in the order that they were originally used in the Pure Data implementation of the system. This implementation also included several additional feedback loops that will be omitted for the sake of clarity, as these processes are superfluous in comparison with the central steps of the overall algorithm.

Segment 1: Nonlinear data retrieval using modular arithmetic

The first segment of the system is a parallel series of functions that read values from finite dataset. These datasets represent audio files, with each individual value representing one sample of data in each respective channel of audio. Each dataset representing one channel of audio data, a stereo .wav file would require two parallel functions and a four channel file (or combination of files) would correspond to four functions, etc.

These functions scan through the samples in their assigned datasets in nonlinear order, allowing for reversal and "skipping" to different points within the set. It shall be shown how the apparently small number of input parameters can in combination produce a wide range of effects, from looping of small to large sections of audio, pitch shifting, stretching, freezing, and pitched tones.

1 Read function

We shall begin the explication of the system by defining the read function which reorders data within each audio channel.

The read function is based on a sawtooth waveform $x(t)$ which oscillates from 0 to 1 when the period variable T_x is given a positive value, and from 1 to 0 when it is negative. Another identical sawtooth waveform $y(t)$ exists with its own independent period T_y which modulates the phase of $x(t)$ only on specific calculations, defined by interval T_r. On such calculations, the phase difference ϕ between the two waveforms is taken and added to $x(t)$ to reset it to the same value as $y(t)$. This function is notated as follows:

$$x'(t) = \begin{cases} 1 - \frac{(t+\phi_{n-1}) \mod |T_x|}{|T_x|} & T_x < 0 \text{ and } t \mod T_r \neq 0 \\ \frac{(t+\phi_{n-1}) \mod |T_x|}{|T_x|} & T_x > 0 \text{ and } t \mod T_r \neq 0 \\ \frac{((t \mod |T_x|)+\phi) \mod |T_x|}{|T_x|} & t \mod T_r = 0 \end{cases}$$

T_x: Period of sawtooth waveform $x(t)$
T_y: Period of sawtooth waveform $y(t)$
T_r: Correction interval
ϕ: Phase offset

Where ϕ is calculated as:

$$\phi = (\text{Phase}y - \text{Phase}_x) \times |T_x| = (\frac{t_r \mod |T_y|}{|T_y|} - \frac{t_r \mod |T_x|}{|T_x|}) \times |T_x|$$

Assuming that t_r is the value of t at any point where $t \mod T_r = 0$.

It should be noted that ϕ_{n-1} is written here to demonstrate that $x(t)$ continues from the corrected phase value until the next reset occurs.

Given dataset D exists and has cardinality $n(D)$, $x'(t)$ is scaled to encompass the domain of dataset D:

$$x_{scaled}(t) = n(D) \times x'(t)$$

$x_{scaled}(t)$ is discretized in order to select from an integer list in dataset D:

$$i(t) = \lfloor x_{scaled}(t) \rfloor$$

For practical application, T_x is initialized to the original playback speed of the audio data. This is calculated by dividing the sample rate r by the cardinality s of the audio file's contents.

$$T_x = \frac{r}{s}$$

Thus, for a 16-second audio file with a sample rate of 44.1kHz:

$$T_x = \frac{44,100}{705,600} = \frac{1}{16} = 0.0625\text{Hz}$$

This value may of course be set to any value desired, such as the opposite of this formula if reversed playback is desired, however this is the most straighforward algorithm for determining the initial period of waveform $x(t)$, particularly when different series of data with varying cardinalities are successively loaded into D during synthesis.

2 Reset interval Markov chain variation

Given dataset D is read by x_{scaled}, subsequent calculations of $x'(t)$ shall have the variable T_r adjusted as follows:

$$T_r(t) = T_r(0) + |D[i(t)]| \times 2 \times T_r(0)$$

Where $T_r(0)$ is the initial value of T_r before the first iteration of the feedback loop, i.e. the input parameter for the period T_r of the phase correction.

This feedback loop adds an element of unpredictability to the timing of reset events in the function, preventing ridigity or gauche repetitiveness and allowing for a pseudo-organic musicality in the reordering function.

3 Density variable

The output of the function $i(t)$ can be additionally controlled by a density function that incorporates a Bernoulli random variable $Z(t)$ with a probability p. The variable $Z(t)$ updates its value only when t is a multiple of T_r.

$$f(t) = D[i(t)] \times Z(t)$$

$$Z(t) = \begin{cases} 1 & \text{with probability } p \text{ if } t \mod T_r = 0 \\ 0 & \text{with probability } 1-p \text{ if } t \mod T_r = 0 \\ Z(t - \Delta t) & \text{if } t \mod T_r \neq 0 \end{cases}$$

This function can be used to reduce the presence of a sound source, to add dynamic elements and silences when an ongoing sound is not desired, or to modify the texture of the function's output. The perceived effect of this function is highly dependent on the value of T_r.