Konstantin Tomashevsky
Music 207
5/14/2009

FractalSound: Algorithmic Self-Similar Resonance Space Generation

Musical harmony tends to rely on a mathematical synchronicity in which frequencies on an equally tempered scale happen to fall near values at rational ratios of each other, therefore sharing harmonics. The resonance model technique outlined here takes advantage of this to generate lists of harmonically coherent sinusoidal resonator specifications. Inspiration for this project came from the sympathetic strings on a sitar, which are tuned depending on the raga being played to flesh out a corresponding harmonic space through resonance. Likewise, the resonator sets generated by this model depend on user input, and are accordingly well-suited for resonating with audio conforming to different keys, modes, and harmonic paradigms.

Resonators in this model are represented by triples of frequency, gain, and decay rate. The first input argument consists of a base resonator triple, which should correspond to the tonic of the resonance-inducing audio. This is the first triple to be added to the output set. The next argument is a list of ratio triplets, to be multiplied by the respective parameters of triplets already in the output set to generate new triplets for the output set. Triplets are combined so that no two share a frequency value. This procedure is repeated for the number of iterations specified by the last input argument, and the output set it subsequently returned, normalized to retain the base triplet as provided.

The Algorithm

Capital F, G and D will denote frequency, gain and decay rate respectively. Lowercase f, g and d will denote ratios to be applied to these parameters.

Input Arguments: $(F_{base},G_{base},D_{base})$ [base triple], $R=\{(f_1,g_1,d_1),(f_2,g_2,d_2),...,(f_n,g_n,d_n)\}$ [ratio triples], m [number of iterations].

The procedure is presented in pseudocode:

$T:=\{(F_{base},G_{base},D_{base})\}$

REPEAT m TIMES:
    $S:=\{\}$
    FOR EACH (f,g,d) in R:
        FOR EACH (F,G,D) in T:
            INSERT (fF,gG,dD) into S
    FOR EACH (F,G,D) in S:
        IF T contains some triple (F,G',D'):
            REPLACE this triple in T with $(F,G+G',\frac{GD+G'D'}{G+G'})$
        ELSE:
            INSERT (F,G,D) into T

Normalize all G and D values in T so that T includes the triplet $(F_{base},G_{base},D_{base})$

RETURN T

Implementation

The above algorithm was implemented as the *resgen* MaxMSP object to function in conjunction with the CNMAT *resonators~* and associated objects. The object has three inlets and one outlet. Right to left:

Right inlet: Takes a triplet message of base frequency, gain and decay rate. A *bang* outputs triplets with updated threshold settings (see

Middle inlet: Takes a message containing a series of ratio triplets, i.e. "Fratio[1], Gratio[1], Dratio[1], Fratio[2], Gratio[2], Dratio[2], … , Gratio[n], Dratio[n]"

Left inlet: An integer sent to this inlet will prompt *resgen* to execute the above algorithm with the given arguments for the directed number of iterations. A list of *resonators~* triplets is returned. A *bang* outputs triplets with updated threshold settings (see below).

The left inlet accepts additional arguments:

lowfreq *float*: Directs *rengen* to discard output triplets with frequencies below *float*. The default value is 60.0, to avoid *resonators~* malfunctions.

highfreq *float:* As with **lowfreq** but now setting a lower frequency bound

thresh *float*: sets *float* as a threshold gain value, with triplets below this not returned. The default is 0.

n *integer*: Directs *rengen* to return only the first *integer* triplets with the highest gain values.

Example:

Figure 1 below shows the *resgen.maxhelp* patch, which illustrates the object's functionality. Through its right inlet, *resgen* is provided with a base resonator triplet of (440.0, 0.4, 0.2). Ratio triplets are provided through the middle inlet. The first one has a frequency ratio value of 1, to ensure that the base frequency remains most prominent. Its decay rate ratio is also 1 and its gain ratio is determined by the leftmost slider in the patch, effectively ranging from 0 to 1. The rest of the frequency ratios correspond to p or $1/p$, for the first six primes p. The remaining sliders control the gain ratio values for triplets corresponding to one prime each, in ascending order from left to right. The use of primes and their multiplicative inverses for ratios ensures the effective generation of partials at many different rational ratios from each other, with the "simplest" ones the most dominant.

Figure 2 shows resonance-display plots for the output of *resgen* with different numbers of iterations, generated with the ratio settings shown in figure 1. The message "n 100000" was fed into the left inlet to prevent low-gain triplets from being discarded. By five iterations, more triplets are outputted than would be accepted by *resonators~* (1024). As the number of iterations increases, the emergence of dense self-similar structure is visible.

I have included a stereo AIFF file, with a recorded guitar in the left channel and the response of a resonance model in the right channel, generated as described above after 7 iterations. The guitar plays an A major scale and a I IV V progression followed by an A minor scale and a i iv v minor progression. The file ends with the resonance model responding to pink noise being fed through it for a few seconds.

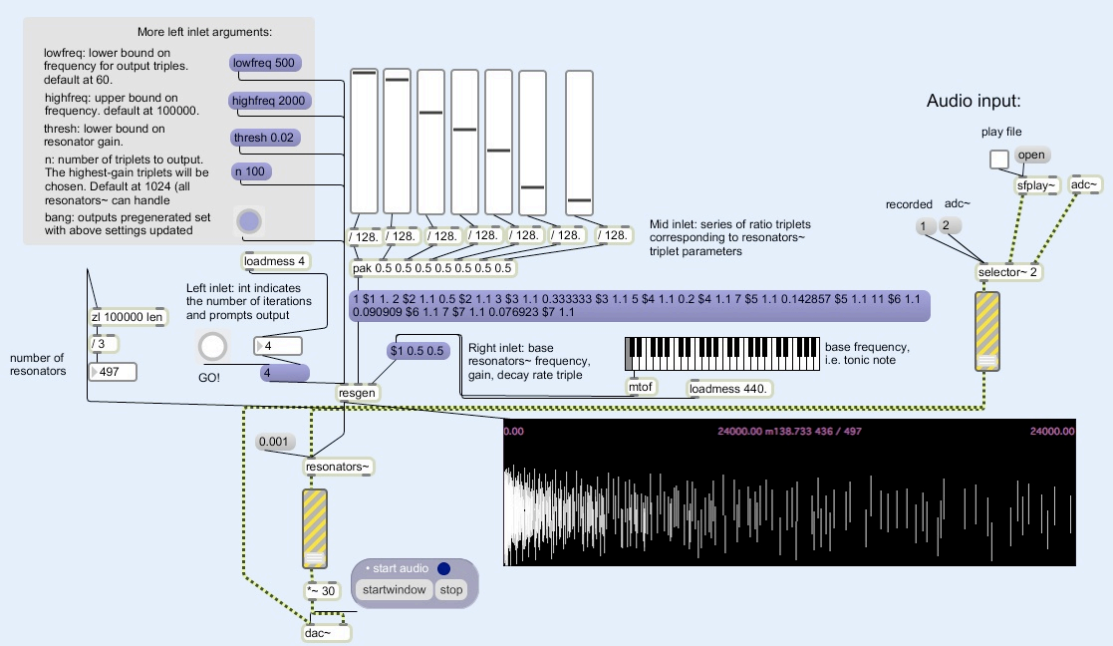


Figure 1: The resgen.maxhelp patch

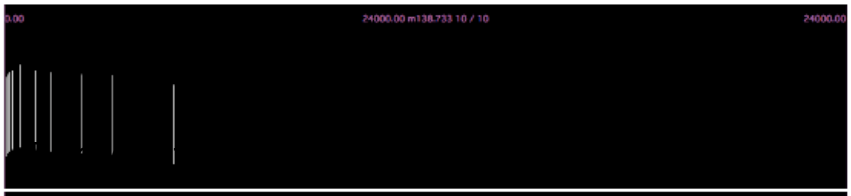
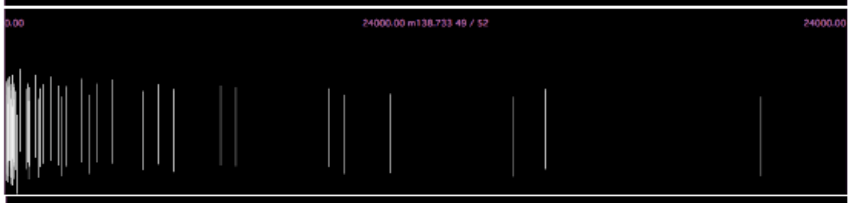
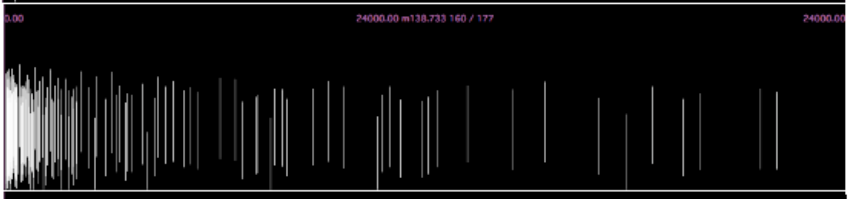
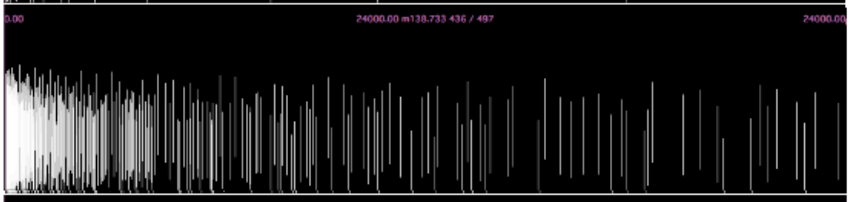
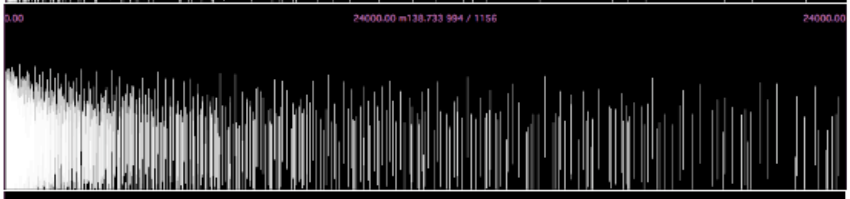
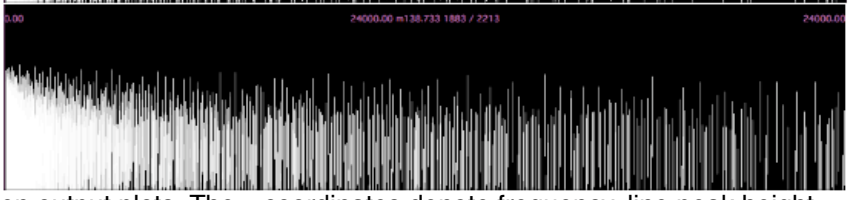
Iterations	Partials	Graph
1	10	
2	52	
3	177	
4	497	
5	1156	
6	2213	

Figure 2: A table of resgen output plots. The x coordinates denote frequency, line peak height denotes gain value, length denotes sustain.