

Composing with Composed Waveforms having Multiple Cycle Lengths and Multiple Paths

Arun Chandra

Abstract

To describe a sound in terms of the structure of its waveform, and not in terms of its acoustic appearance, is the compositional premise of these programs. As a music composer, I am interested in *describing precisely* the path of transformation I desire, then discovering what its acoustic consequences are, instead of *describing precisely* the acoustic consequences I desire, then doing what is needed to achieve them.

wigout and *TrikTraks* are programs that allow for the structural specification of waveform states and how they change over time. A consequence of this procedure is that all aspects of the resulting sound (pitch, volume, color, and gestural behavior) are interdependent, continuously changing, and controlled by relatively few variables. This report describes some results of constructing states consisting of multiple *cycle lengths*, and some possibilities for using multiple *paths*.

1 Overview

Cycle lengths below a certain threshold (depending on the overall length of the state) generate frequencies in the auditory range. The choice of cycle lengths and paths generates sounds that vary between single-timbre to multi-voiced texture, between dense and still, to thin and quickly moving.

The grammar for waveform specification and transformation is conceptually simple. The *state* of a waveform is defined as a sequence of *segments*. Each *segment* has at least two variables: *number of samples* (length) and *amplitude* (height). (Other segment types, having more variables, can also be defined, see [4, 5].) The *state* is iterated until the desired duration is reached. Upon each iteration, each variable changes by its specified amount.

The amount by which a variable changes determines its *cycle length*: the number of iterations after which a variable returns to its starting value. Each variable for every segment can be given a distinct cycle length, and there can be up to 64 segments in a state. The cycle length can be relatively long (*e.g.*, 600 iterations), or short (*e.g.*, 3 iterations).

When a variable's cycle length is short, its periodicity generates audible frequencies. These audible frequencies are *in addition to* the base frequency, which is the sum of segment lengths for a particular state. The choice of *what* cycle length for *which* variable, and the *number* of segments within a state, determines the resulting sound and its behavior.

In contradistinction to *wigout*, *TrikTraks* is a program that allows the variables (described above) to be given a unique path that determines their changing magnitude over time. Each *path* can be specified as: 1) a sine wave; 2) a triangle wave; 3) a sawtooth wave; 4) a square wave; 5) a polynomial between degrees 3 and 10; 6) or as an amplitude modulated FM wave. Every variable of every segment can be given a path that is independent of all other paths. Upon each iteration of the sequence of segments, each variable of every element changes its magnitude as determined by its path of transformation.

2 Predecessors

The significant predecessors of both *wigout* and *TrikTraks* are SAWDUST, designed by Herbert Brün and written by Gary Grossman, Jody Kravitz, and Keith Johnson at the University of Illinois[3][8], and SSP (Sound Synthesis Program), designed by Gottfried Michael Koenig, and realized by Paul Berg, Robert Rowe, J.D. Banks, and David Theriault at the Institute for Sonology, Utrecht[2]. These programs can be classified as “non-standard sound synthesis programs,” following S. R. Holtzman's definition[7]. Paul

Berg's description of SSP is appropriate for *TrikTraks*: "... [it is] best suited for a user who wants to define structures and listen to the results rather than a user who at all costs must have a certain sound."

3 Postulate

By definition, a "tone" requires a repeating period. The "period" can be the consequence of a mathematical function (such as a sine function), or it can be constructed, sample by sample, and then repeated. The *repetition of a sequence of samples*, regardless of their amplitudes, gives that sequence the function "period," and will create a sound. The length of that sequence (the number of samples in it) determines the sound's base frequency, and the amplitudes determine its timbre. Changing the number of samples in the sequence changes the base frequency of the sound; changing the amplitude content of the sequence changes the timbre.

If it is the case that "repetition" is what generates both frequency and timbre, then constructing a waveform with multiple repetitions should result in one waveform with multiple sounds.

wigout allows the exploration of waveforms with multiple cycle lengths, and *TrikTraks* allows the exploration of waveforms with multiple transformation paths.

4 Wigout

4.1 Waveform Specification

Input for *wigout* is given in the form of a datafile. The first row of the input datafile (see below) is the id for the segment, followed by the segment type. (Only the *wiggle* type is used in this paper. For others, see [4, 5].) The second row refers to the *number of samples* in the segment, and the third to the *amplitude* of the segment. The columns refer to: 1) the initial value for the variable; 2) its maximum value; 3) its minimum value; and 4) the amount by which it changes upon each iteration.

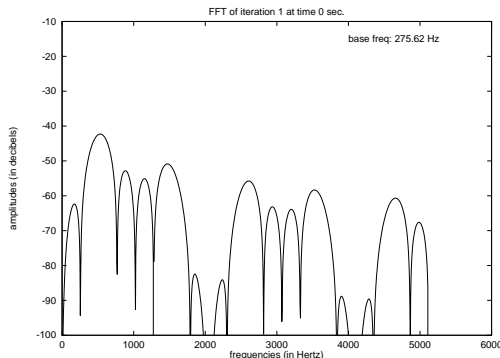
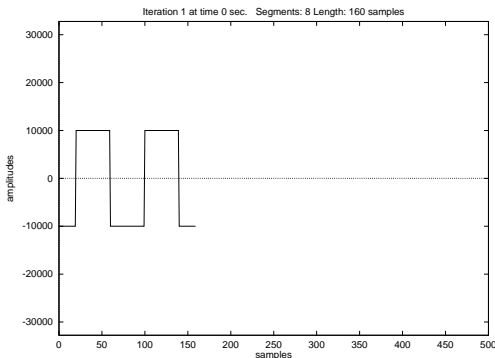
Below is an example of an input data file of two segments:

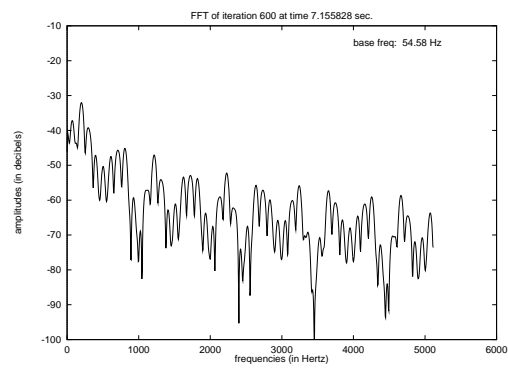
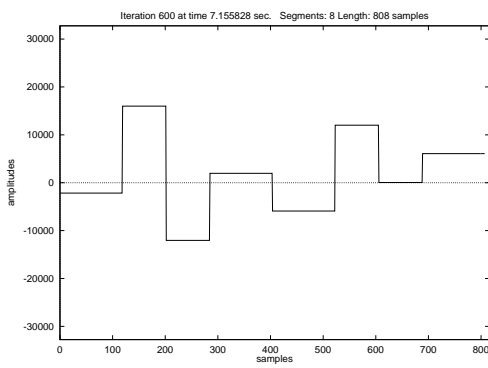
```
# 1st input datafile for wigout
w0 wiggle      # id, type
50 100 50 5    # samples (init., max, min, rate)
-20588 20588 -20588 2422.11 # amplitudes

w1 wiggle      # id, type
67 67 33 1.7   # samples (init., max, min, rate)
796 796 -796 106.133333 # amplitudes
```

So, for each of the variables (samples, and amplitude), one can set the range, rate of change, and starting value. Upon every iteration of the state, each variable changes by its rate, until it reaches its minimum or maximum value, whereupon its rate changes its sign. This process results in a *cycle length*, which can be calculated by $numCycles = 2 * (max - min) / rate$. Using this formula, segment w0 has a cycle length of 20 for its samples and 34 for its amplitudes; and segment w2 has a cycle length of 40 for its samples and 30 for its amplitudes.

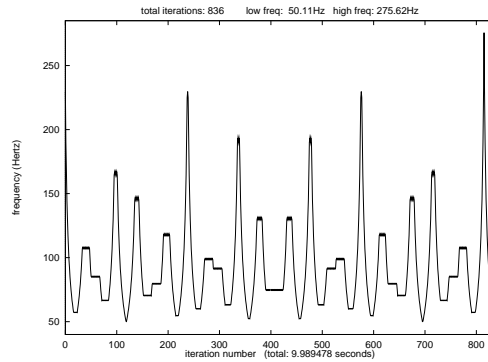
Below are the 1st and 600th iterations of an 8-segment waveform, followed by their FFTs. The iterations are about 7 seconds apart.





It should be apparent from the above plots that the iterative change of variables (which is the procedural paradigm of *wigout*) generates significant differences in frequency and spectrum over time.

Below is a plot of the changing base frequency from state 1 until state 836 of the above waveform:



As you can see in the above plot, the changing base frequency manifests some periodicities by its regularity of peaks and troughs.

In presenting the above FFT plots and the changing base frequency plot, I would like to show you how *wigout* allows for significant changes of base frequency that are simultaneous with significant changes in timbre. This, more than anything else, is what I, as a composer of music, am attracted to in *wigout*.

4.2 Multiple cycles

The ability of specifying sample changes independently from amplitude changes allows for these two variables to have distinct cycle lengths. Since a state can have up to 64 segments, up to 128 different cycle lengths can be explored, with the segment type *wiggle*.

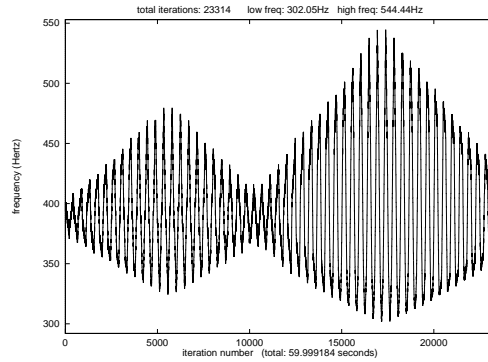
As is written above, the *cycle length* of a variable and its *rate of change* are inversely related. So if we know what the cycle length is to be, and we know the range of the variable, its rate of change can be determined by $rate = 2 * (max - min) / numCycles$.

Here is the input file for 4 segments, whose variables all have cycle lengths that are relatively prime. The ranges for each variable are given in parentheses following the hash mark. The ranges for the samples are written in standard pitch/octave notation, and assume a sampling rate of 44100 samples per second. The ranges for the amplitudes are in decibels, and assume 16-bit sound samples. The last number on each line is the variable's cycle length.

```
# 2nd input datafile for wigout
w2 wiggle
15 45 15 0.134529          # (b5 f#6) 223
-5107 5107 -5107 65.057325 # (80 dB) 157
w3 wiggle
42 42 16 0.114537         # (c5 f6) 227
2016 2016 -2016 24.736196 # (72 dB) 163
w4 wiggle
17 40 17 0.100437         # (c#5 e6) 229
-796 796 -796 9.532934    # (64 dB) 167
```

```
w5 wiggle
38 38 18 0.085837 # (d5 eb6) 233
20588 20588 -20588 238.011561 # (92 dB) 173
```

And this is a plot of the changing base frequency for the waveform, using only the segments defined above, over a duration of 60 seconds.



Note that gradually, over 60 seconds, the base frequency's range changes significantly. This occurs gradually because the sample's rate of change for each segment is very small, meaning that upon every iteration of the state, the *number of samples* for each segment will barely change.

The timbre is changing faster than the base frequency, as is shown by the shorter cycle lengths for the amplitudes (157, 163, 167 and 173). Unfortunately, the change in timbre is more noticeable when heard than when seen as a sequence of FFTs, so no plots of the changing timbres are included in this report.

5 TrikTraks

In *wigout*, the path of change followed by each variable is invariant. Only the speed at which it traverses its path can be adjusted. My work on *TrikTraks* began by addressing this consistency of *wigout*, and wondering what would happen if each variable could follow a path distinct from the others.

5.1 Standard Paths

"Standard path" refers to using a standard waveform to control the path of the variables *amplitude* and *number of samples*. Input to *TrikTraks* is from a datafile, where is specified 1) the path range (minimum and maximum values); 2) the control waveform type (sine, square, triangle, or sawtooth); 3) control waveform frequency; and 4) the control waveform phase.

Note: "control waveform frequency" refers to the *number of complete periods* of the control waveform that will occur over the total time requested. So, if the duration = 10, frequency = 3.5, and type = 3, the path will be three and 1/2 periods of a sine wave over 10 seconds.

As an example, here is an input datafile, specifying the variables for a sequence of 3 segments. The first row of each pair refers to the *number of samples* and the second to the *amplitude*. The first column sets the variable's maximum value, the second its minimum, the third the type of controlling waveform, the fourth the controlling waveform's frequency, and the last is the controlling waveform's phase.

```
# TrikTraks: input file for standard waveforms
duration: 10 # duration of sound in seconds

100 10 sin 3.8 0.75 # segment 1: samples
10000 -10000 tri 1.5 0.75 # amplitude

50 20 squ 5 0.5 # segment 2: samples
10000 -10000 tri 3 0.25 # amplitude

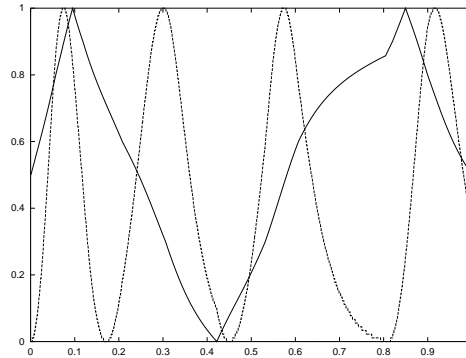
200 100 saw -3.5 0.75 # segment 3: samples
30000 -30000 sin 11 0.83 # amplitude
```

The above is one *sequence of segments* that are iterated and written to disk until the total duration is reached. As in *wigout*, a sequence can have up to 64 segments.

The first segment (in the above datafile) has samples that will vary in number between 100 and 10 (their “range”), the path will be that of a sine wave, 3.8 periods of the sine wave will be generated over the total duration (10 seconds), and the starting phase of the sine will be 0.75.

The first segment’s *amplitudes*, however, will range in value between ± 10000 , the path will be 1.5 periods of a triangle wave over 10 seconds, whose phase is 0.75.

Here is a normalized plot of both the sample and the amplitude paths for the first segment:



The distortions in the plot are due to the procedure with which the lookup algorithm (used to generate the triangle and other waves) was called. They were allowed to exist, since they were the result of the proper operation of the lookup algorithm, and generated a few more resistances to periodicity and symmetry.

By choosing distinct values for the type of path, its frequency and phase, each variable of each segment can have a path independent from all other segments’ paths.

The sounding frequency of this waveform is the sum of the segments’ samples at every iteration. Since this number is constantly changing, the frequency is constantly changing. The square and sawtooth paths have a periodic jump from the minimum to the maximum value (or vice versa). This results in an immediate and drastic frequency change, when applied to the variable *number of samples*. The amount of the change depends on the user specified maximum and minimum values for that variable, and the content and behavior of the neighboring segments. The sine and triangle path, in obvious contrast, have smooth rises and falls.

The compositions *smear pulse no sneer* and *The thin red line of subject matter* were composed using only the above waveform paths.

5.2 Polynomial Paths

A consequence of using the standard paths was periodic oscillations of the variables between their maximum and minimum values. (The overall periodicity of the resulting sound was far more complex than the periodicity of any one path, but, nonetheless, the resulting regularity was noticeable.) Since periodicity is (in information theoretical terms[9]) a greater degree of redundancy than needed to convey a message, my interests as a composer led me to look for ways in which it was minimized.

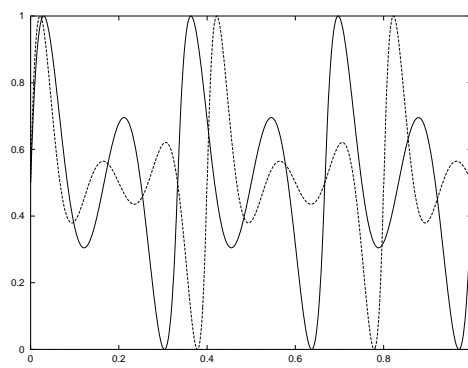
Polynomials are used in Herbert Brün’s SAWDUST system in the implementation of the VARY algorithm. I decided to try to use them in *TrikTraks* by specifying equally spaced zero-crossings, then scaling them to their specified limits. The values are loaded into a table, and interpolating wave table lookup algorithms are used to extract values with the desired frequency and phase. This procedure allows for a polynomial to occur with a variable number of periods over the requested duration.

From *TrikTraks*, a polynomial path is specified in the following way, here given for only one segment:

```
# TrikTraks: input file for polynomials
200 100 poly 5 3 0 # samples
30000 -30000 poly 7 2.5 0 # amplitude
```

The range is given first (maximum and minimum values), then the type of path (“poly”), followed by the degree of the polynomial to be generated, the number of periods over the sound’s duration, and the initial phase of the polynomial. Polynomials can be requested of degree 3 to degree 10.

Here is a plot of the paths for both the samples and the amplitudes, for the data given above:



Please note that the above are the paths for *one* segment of a waveform, and that a waveform can have up to 64 segments, and thus there can be up to 128 distinct paths.

5.3 Amplitude modulated FM functions

Although the polynomial paths do generate a variety of peaks and troughs (as compared to the standard paths described above), because of the equidistant zero-crossings used to generate the polynomials, the *rate* of the peaks and troughs remains constant. One can have multiple periodicities, but they remain periodicities. As a result, I wondered what would happen if one used an amplitude modulated FM function for the generation of the paths: $v = \sin(2\pi f_1 t) * \sin(2\pi f_2 t + mi * \sin(2\pi f_3 t))$

The postulate was that the change of *rate* of peaks and troughs would be determined by the FM function, and the change of *magnitude* of peaks and troughs would be determined by the AM function.

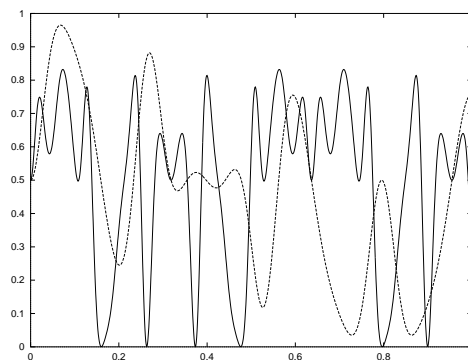
Please note that the FM and AM functions are *not* being used to directly generate sound samples. Rather, they are being used to generate the path that will be followed by the variables in every segment.

Specification of the amfm function is done in the following way:

```
# TrikTraks: input file for amfm (1)
200 100 amfm 0.75 1 1.25 1.5 3 # samples
30000 -30000 amfm 0.5 0.4 0.6 0.8 # amplitude
```

The first two numbers specify, again, the range of the path (maximum and minimum), followed by the type of path chosen (“amfm”). The following numbers represent: 1) f_1 , the frequency of the amplitude modulation; 2) f_2 , carrier frequency for FM; 3) f_3 modulation frequency for FM; and 4) mi , modulation index for FM.

This is a normalized plot of both the sample and amplitude paths for the above data:



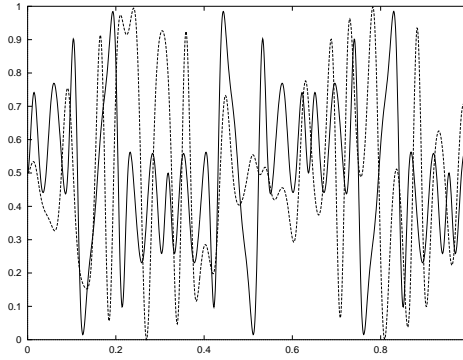
As you can see, the resulting complexity of the path is greater than that of either the polynomial paths or the standard waveform paths.

The amfm path can be used in conjunction with the polynomial paths or the standard paths described above. Thus, an waveform can be constructed using standard waveforms, *and* polynomials, *and* the amfm function.

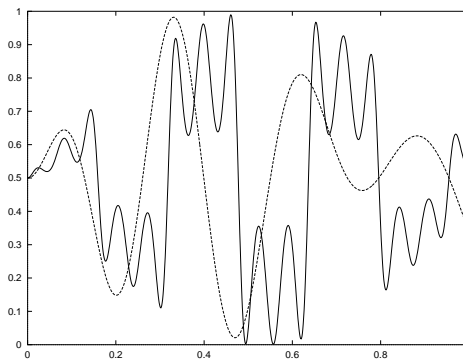
The implementation of the amfm paths is similar to that described for the polynomial paths above: the function is evaluated, the output is scaled and loaded into a table, and output values are interpolated from the table.

As an example of the variety of path combinations available with the amfm function, the input files and normalized plots for two segments are given below.

```
# TrikTraks: input file for amfm (2)
50 10 amfm 1 1.25 1.5 1.75      # samples
10000 -10000 amfm 0.15 0.5 0.9 4 # amplitude
```



```
# TrikTraks: input file for amfm (3)
100 80 amfm 0.075 0.5 1 2      # samples
20000 -20000 amfm 0.1 0.02 0.03 20 # amplitude
```



6 Conclusions

Since this has been an exploratory project, there are no conclusions *per se*, but rather indications for further work. Some possibilities are:

1. Allow for a variable in a waveform to have zero change over the course of a sound.
2. Allow for the amplitude changes to happen logarithmically rather than linearly.
3. Allow for the shifting from one waveform type to another *in medias res*.
4. Develop secondary control waveforms for the variables. This would begin to approach an elementary calculus of waveforms, in which the rates of change are themselves changing.

This approach of structural modification of waveforms in time generates a high degree of unpredictability with regard to the acoustic consequences. As a result, a composer of music has to develop new ways of specifying what she wants with regard to the output. Relatively simple input can generate wildly complex output behavior. This potential richness is, more than anything else, the most seductive aspect of this project for this composer.

7 Acknowledgments

I'd like to thank Jim Beauchamp and Sever Tipei, the co-directors of Computer Music Studios in the School of Music at the University of Illinois, for allowing me access to its facilities. Herbert Brün has been a constant and invaluable conversation partner in this, and many other, investigations. Conversations with Michael Brün helped develop the the AMFM function in *TrikTraks*.

The ideas that generated *wigout* could not have been had without the work of Heinz von Foerster[6] and W. Ross Ashby[1].

The plots for this article were created by *gnuplot* version 3.5, available from the Free Software Foundation. *gnuplot* ran as a child process under *wigout*, and gave the Postscript output that was later included into this \LaTeX document.

I wrote *wigout* and *TrikTraks* in C, and they run under UNIX or DOS. Up till now, six compositions have been written with them: *An untitled poem in 16 stanzas by Keith Moore* for trombone and tape (13 minutes, 1992), *A Bit, A Curve, Alas, A Wave!* for tape (8 minutes, 1993), *smear pulse no sneer* for tape (10 minutes, 1994), *700,000 Dead* for voice and tape (7 minutes, 1995), *The Last Statement* for voice and tape (6 minutes, 1995), and *the thin, red line of subject matter* for tape (6 minutes, 1995).

8 References

1. Ashby, W. Ross. *An Introduction to Cybernetics*. Methuen and Company, Ltd. London: 1956.
2. J.D. Banks, P. Berg, R. Rowe, D. Theriault. *SSP: A Bi-Parametric approach to Sound Synthesis*. Institute of Sonology, Utrecht: 1979.
3. Brün, Herbert. *My Words and Where I Want Them*. Princelet Editions. Urbana, Illinois: 1991.
4. Chandra, Arun. "The linear change of waveform segments causing non-linear changes in timbral presence" In *Contemporary Music Review*, vol. 10, part 2. Harwood Academic Publishers: Basel, Switzerland.
5. Chandra, Arun. "CounterWave: a program for controlling degrees of independence between simultaneously transforming waveforms" in *Proceedings of the International Association of Knowledge Technology and the Arts*, Osaka, Japan: September, 1993.
6. von Foerster, Heinz, editor. *Cybernetics of Cybernetics*. Biological Computer Laboratory, Report Number 73.38, University of Illinois. Urbana, Illinois: 1974.
7. Holtzman, S.R. "A Description of an Automatic Digital Sound Synthesis Instrument," in *D.A.I. Research Report No. 59*. Department of Artificial Intelligence, Edinburgh: 1979.
8. Roads, Curtis, editor. *Composers and the Computer*. William Kaufmann, Inc. Los Altos, California: 1985.
9. Shannon, C.E., and Weaver, W. *The Mathematical Theory of Communication*. University of Illinois Press. Urbana, Illinois: 1949.