freeHorn

Larry Polansky
Software by Phil Burk and Larry Polansky

User Notes
(in progress)

12/5/06

Introduction
freeHorn is a stand-alone application. It has no inputs, only outputs. The performer can modify its form, length, and behavior to make hisr own pieces.

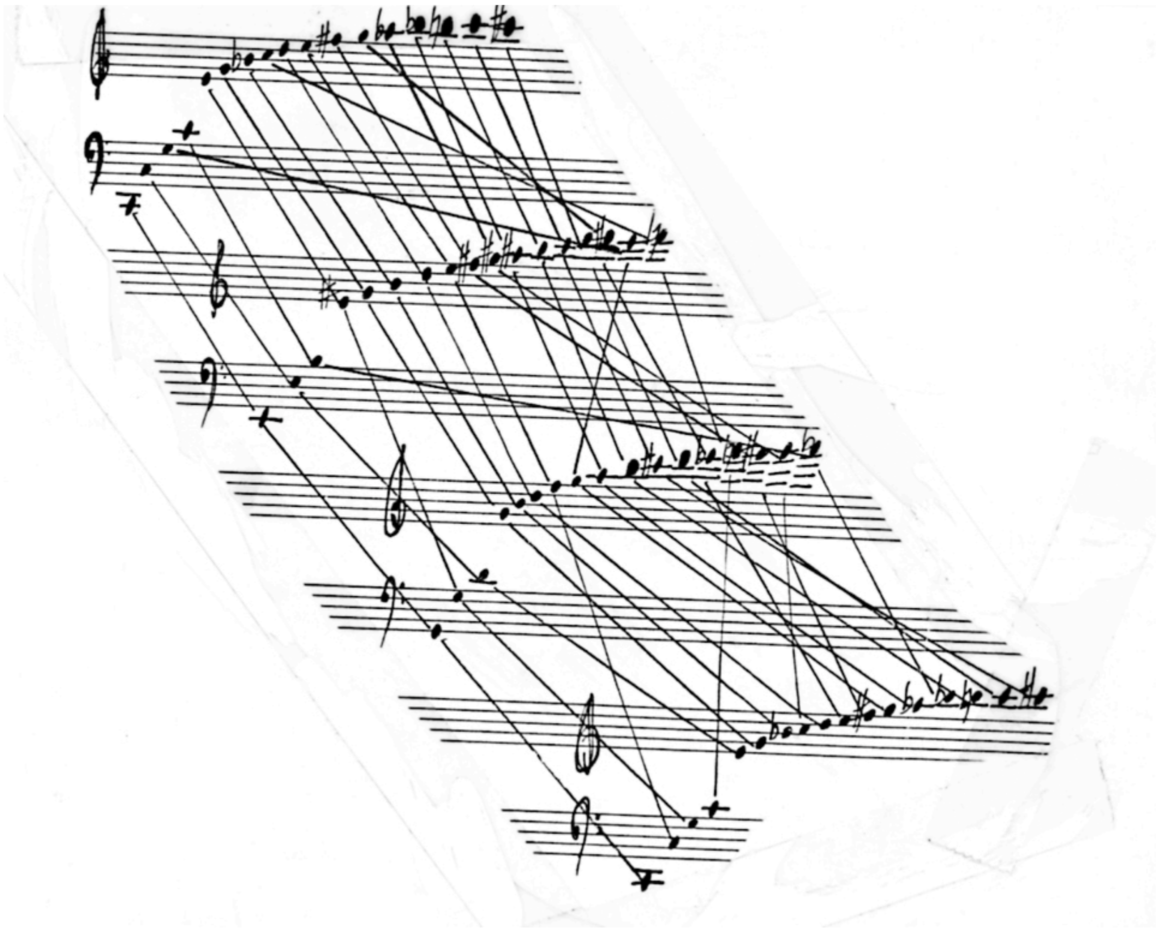
freeHorn is written in Java using Phil Burk's JSyn libraries. It is a Java .jar file, so should run on any platform.

Before running *freeHorn*, you will need to download the JSyn plug-in from Phil Burk's Softsynth site.

http://www.softsynth.com/jsyn/

***freeHorn*, and the *Psaltery* set of pieces**
freeHorn is an highly generalized software implementation of a set of pieces, beginning with *Psaltery (for Lou Harrison)*, in 1978. The form common to these works is a continuous modulation, by some morphing or replacement algorithm, between harmonic series on fundamentals related as simple integer ratios. In the previous pieces, pitches up to the 17th harmonic are used, and the fundamentals of the three series are related as 1:3:5:1 (often with octave equivalences).

Each section of the piece consists of one harmonic series "cross-fading" with the previous one. The replacement algorithms for these has varied slightly, as has the length, instrumentation, fundamental, and a few other things.



(Original replacement diagram for *Psaltery*)

Although each piece has used a different instrumentation, all have used the first 17 harmonics, the progression 1:5:3:1 for the harmonic series, and some *replacement algorithm* for the harmonics which specifies that a new harmonic (that is, an entering harmonic from a new series) will replace the (still remaining) harmonic from the old series which is closest in frequency. Each of these pieces has used the following order of entry for new harmonics:

17, 13, 11, 14, 7, 15, 10, 5, 9, 12, 6, 3, 16, 8, 4, 2, 1

— which may be described as the decreasing order of harmonic prime complexity. I call this, informally, "psaltery order" after its use in the first piece in the set. Higher primes enter first, starting with their highest powers. The first series enters, to begin the piece, in the opposite order (1, 2, 4, 8, 16, 3, 6, 9, 12, 5, 10, 15, 7, 14, 11, 13, 17), and the final "fade-out" of the first series, at the end of the piece, consists of pitches dropping out in the order of entry (from high prime to low).

The pieces in this set include several written in the late 70s (*Canon for Flute* (for flute and multitracked flutes), *Flutes* (for flute choir), *'Cello* (for cello and multitracked celli), *Choir* (for choir), and *Glass* (for 51 tuned water glasses)). Two more recent works (*Horn* (1990) and *Choir/Empi's Solo* (1997)), are computer generated, and scored for tape and solo live performer.

For recordings and scores of many of the pieces in this set, go to my website: http://music.dartmouth.edu/~larry

What *freeHorn* does
In *freeHorn*, the user can set the length of the piece, the *form*, and many of the sonic and musical parameters. The *form* is specified by a set of ratios, which determine how many sections the piece will have, as well as the fundamentals of those sections. The software uses a *replacement algorithm* to compute the progression between from one series and another.

For each new harmonic series (except the first, which enters in the reverse order to that stated below), pitches enter in the order described above (assuming the harmonics used are 1-17):

17, **13**, **11**, 14, **7**, 15, 10, **5**, 9, 12, 6, **3**, 16, 8, 4, **2**, 1

Highest primes enter first (shown now in boldface), and all multiples of these primes are "used up" before the next prime enters. Primes enter from their higher powers "on down." This algorithm is a kind of simplified version of functions like the Euler GS function (a measure of numerical complexity).

Thus, after the first series (on the first fundamental) has entered completely, the next series begins replacing it by first replacing some pitch in the previous (existent) series with pitches corresponding to the new fundamental's 17th harmonic, 13th, 11th, … 1st. In previous pieces, I've usually used a simple *nearest neighbor* scheme for the replacement algorithm, but others are possible (at the present time, *freeHorn* only implements one such algorithm, but there are "stubs" for a few others).

Though the user specifies some number of sections by entering in a list of ratios (real numbers, separated by colons), there is *one additional section* in the piece, added automatically by the software.

```
sectionLength = totalLength/(numberHarmonicSeries + 1)
```

After the final specified series replaces the penultimate, pitches drop out from most complex to most simple (17, 13, 11, … 16, 8, 4, 2, 1). This final (non-specified) section is the same duration as the others. In other words, if the user specifies three sections for the piece:

1 : 1.5 : 1.25 : 1

— the piece will really consist of

1 : 1.5 : 1.25 : 1 : (1, drop out)

In other words, if the performer specifies a length of 10 minutes and *four* sections, there will actually be *five* sections, each dividing the total duration more or less evenly.

In the above example, the first thing to occur (section 1) is the build up of harmonic series on 1 (times the fundamental). That series will then be replaced (section 2) by the harmonic series on 1.5 times the fundamental (3/2 ratio, or a perfect 5th), beginning with the new series' 17th harmonic. That will in turn be replaced (section 3) by a series on 1.25 times the fundamental (5/4 ratio, or a major third), in similar fashion. The series on 1 will then replace the series on 1.25 (section 4), and will then "drop out" (section 5, added by the software) from highest harmonic on down (thus, 4 ratios generate 5 sections).

Any ratios are possible, any number of sections (with the implicit assumption that specifying, say, 13 sections actually yields a piece in 14 sections).

freeHorn uses mainly sine waves, though a little *spectral complexity* can be added (see below). The performer can specify a number of parameters (duration range and distribution, attack, decay, etc.).

Make your own piece
freeHorn is, after all, just a small piece of portable software which plays a lot of sinusoids, and morphs between harmonic series. Please use it in any way you like. Make your own piece(s), use it as part of other pieces, or use it as a tool for music and experimentation as your imagination dictates.

Interface and Features
The following is a feature-by-feature description of the *freeHorn* software, documenting the user interface. There are three windows:

- *main* window (where you can make your own version of the piece);
- *clock* window (which tells you where you are in the piece);
- *JSyn waveform scope window* (you can close that, it doesn't really do much except provide a simple oscilloscope for the sound).

In the main window, to *change values*, select the text box (will usually cause the text area to be blue), type in new values, and hit *enter*. Most values need to be entered as real numbers, with something before and after decimal point (the only exception is the Section Ratios box).

Except for Total Length, Section Ratios, and Num Harmonics, all values may be changed in real time during the piece.

Clicking on Start begins the piece with the current settings.

Feature-by-feature Description of the Interface

Total Length

The length of the piece, in minutes. 10.5 minutes is 10'30". The number of sections specified in the Section Ratio box (plus 1, see above) divides the length of the piece more or less evenly.

This value may not be changed once the piece has begun.

Section Ratios

This value may not be changed once the piece has begun.

These ratios might be simple harmonic ones (as in the default), but the software allows the user to select any ratios whatsoever. The ratios must be specified in decimal form. Thus, for the rational values:

1 : 7/4 : 3/2 : 11/8 : 5/4 : 1

— the user must type in

1 : 1.75 : 1.5 : 1.375 : 1.25 : 1

The first series does not have to be 1. For example, the structure 3:5:7, is fine, but the fundamental is still considered to be 1. That is, the first series would be built on harmonics of a pitch 3 times the *fundamental*.

Fundamental Freq

The frequency, in Hertz (cycles per second), to which all other frequencies in the piece relate. This is, in some way, the "key" of the piece. For example, if this is set to 100 Hz, and the first section ratio is 1, the first harmonic series created will be 100 Hz, 200, 300, … 1700. If the second section ratio is 1.5, the frequencies of that section will be 150 Hz, 300, 450, 600, … 2550.

This value may be changed in real-time.

Num Harmonics

This specifies the number of harmonics that each series will use. Typically, I use 17. Note that since each section more or less divides the total duration of the piece, the number of harmonics will divide the section's duration more or less equally. The musical result of this is that the more harmonics, the more active the modulation will be (that is, the piece will change more rapidly).

This value may not be changed once the piece has begun.

Duration Mean

The mean duration for sound events. Values will be chosen by a kind of Gaussian (or bell curve) function around this mean. See Duration Range, Min, and Max for related parameters.

Values are specified in seconds, as real numbers. 0.5 is ½ second.

Long values for duration mean will create more drone-y, static, chorale-like textures. Short values will create a more events.

Duration Range

This is a value centered on the Duration Mean, with limits specified by the Duration Min and Duration Max.

The specification and computation of duration in *freeHorn* is a little unusual. The range is the argument to the standard Gaussian algorithm, specifying the width of the bell curve. The min and max are independent of that. If either is within the range, the values will simply be clipped. This allows the user to create some assymetric Gaussian distributions about a mean. The code is as follows:

```
public static double nextGaussian( double mean, double
range, double min, double max )
      {
            // gaussian distribution scaled by range,
            added to mean
          double gauss = (random.nextGaussian() *
            range) + mean;
         // clip to min and max
          if( gauss > max ) gauss = max;
          else if( gauss < min ) gauss = min;
          return gauss;
      }
```

nextGaussian is the standard Java method that returns a "normal" distribution with mean = 0, SD = 1. The SD is multiplied by the range, and added to the mean. A small range will result in a tighter curve around the mean; a large range will spread durations more around the mean (up to min or max). The mean need not be centered between min max; they are hard limits.

The best way to gain a "feel" for how this all works is to try different sets of values, beginning with some conservative, simple ones (say mean = 1, min = .25, max = 1.5, range = 3. Different combinations will yield different duration distributions and thus, different musical behaviors for the piece.

All of these values may be changed in real-time.

Attack Min

Minimum time, in seconds, of attacks. A Gaussian distribution between Attack Min and Attack Max chooses attack times.

Attack Max

Maximum time, in seconds, of attacks. A Gaussian distribution centered between Attack Min and Attack Max chooses attack times.

Decay Min

Minimum time, in seconds, of decays. A Gaussian distribution centered between Decay Min and Decay Max chooses decay times.

Decay Max

Maximum time, in seconds, of decays. A Gaussian distribution centered between Decay Min and Decay Max chooses decay times.

Duration Min

Minimum duration that will be picked by the software for a note during the piece (see Duration Mean, Duration Range).

Duration Max

Maximum duration that will be picked by the software for a note during the piece (see Duration Mean, Duration Range).

Inter Event Rest

Within a given voice, there are silences between events. The *interEventRest* specifies the values that the software uses to determine how long to wait between the next event in a given polyphonic segment of a voice. This is specified as a % of the duration, after the duration is computed in the usual way. That is there are currently 4 notes sounding in a given voice (playing one harmonic), and one ends, the *interEventRest* time is how long the software will wait before "filling up" the polyphony with the desired fourth event.

The performer specifies a number, as a fraction of the *durationRange*, for the *interEventRest*. The software multiples that number by the *durationMean*, and then computes inter-event rests on an event-by-event basis stochastically using a Gaussian distribution with that value as the mean, and the *durationRange* * *interEventRest* as the range around that mean. The simplest way to think of it is that these *interEventRests* will be some fraction of the mean duration.

Note that rests, as specified in the above *restProbability*, count as events. That gives the user several different, not necessarily independent ways, to control the overall density "feel" of the piece.

Rest Probability

The probability that instead of playing a note, there will be a silence for the duration that would have otherwise been selected for that note. This value and Inter Event Rest are two, independent ways of varying density.

Spectral Complexity

A value between 0 and 1 that adds harmonics to the sinusoids, which comprise the piece. These harmonics are added by a simple waveshaper, the value of Spectral Complexity determines, in a simple way, the degree of deformation of the sinusoids. A value of 0 will result in pure sine waves; a value of 1 will result in a richer, noisier, brighter spectrum.

Loudness

A simple loudness scalar for the entire piece. 1.0 is full, 0.0 is silent. This is just a way of turning the output of the software up or down from the screen. It is independent of the computer's own volume control.

Pitch Scaling Factor

Not documented yet

Pitch Scale By

Not documented yet

Replace

Not documented yet

The Clock Window

This window shows:

- the actual running time of the work (as a clock)
- the CPU load (usually this can be ignored, but it can be a diagnostic against using too many voices)
- the *section* the work is currently in (updating as sections change, 1-n)
- a current list of the "activity" of the piece at the moment (e.g "3.0 * 17 replacing 1.0 * 17", meaning that the 17th harmonic of the 3 series is currently replacing the 17th harmonic of the 1 series)

Some Performance Notes and Suggestions

- You can run as many versions of *freeHorn*, at the same time, as your computer can handle (it's pretty efficient, shouldn't use much processor time). Each version you run is independent of the others (they each have their own clocks, formal structure, fundamentals, etc.). This is a good way to make canons, thicken the texture, allow for very complicated simultaneous harmonic fabrics, and so on. You can sync them by simply hitting START reasonably close together.

- There's not much done with stereo spacing in *freeHorn*. Note events are "panned" out to the stereo field in simple, kind of random way. If you want different versions of the piece to go to different sets of speakers or mixer channels, use several laptops.

- Interesting performances can be created with unusual section ratios, which may involve complex real-number relationships. I sometimes use very small ratios, such as 1: 1.001: 1.003: 1, which means that the harmonic compass of the piece will be quite small, but that the replacement algorithms will generate a lot of beating and dissonance.

Notes, bugs, things I need to improve (but may not get around to):

1) Right now there is no way to save the settings, so that you can easily repeat a performance. Sorry. I usually just jot them down.
2) It's good to pick low fundamentals for the Fundamental Frequency. The harmonics picked by the software are in their proper octave, so they can get quite high.
3) I should add sliders for the values, and ways to move multiple values at once.
4) There should be ranges for amplitude, spectral complexity, maybe other values (even the fundamental), like the duration range.

Acknowledgements (notes)
freeHorn is written in Java and Phil Burk's Jsyn API. Jsyn is written by Phil Burk. The software was designed by the composer and Phil Burk, and written by Phil Burk with assistance from the composer. *freeHorn* was commissioned by Kirsten Barrow

Thanks to Evan Geiger, Phil Burk, David Dunn, Ed Carroll, Benedict Schlepper-Connolly and others who have performed this piece, made suggestions, and helped in various ways.

Lp
12/15/06
rev. 1/10/07

Some Example Settings

Example 1

freeHorn with Neil Sweeney (sax), 25 november 2006, Benedict Schlepper-Connolly, computer.
Sligo, Ireland, The Printing House Festival of New Music
(These are the settings for different times in the piece, which change
10 mins (total duration)

fundamental 58.43	*num harm 17*	*section 1:1:3:5:8*	*attack min 0.01*
decay min 0.2	*duration min 0.05*	*interevent rest 0.5*	*spect. complexity 0.1*
decay max 1.0	*rest probability 0.5*	*loudness 0.5*	

	dur. Max	**dur. Min**	**dur. Mean**
0:00	0.5	0.05	0.2
1:00	1.0	0.05	0.4
2:00	2	0.05	0.8
3:00	4	0.05	1.6
4:00	8	0.1	3.2
5:00	16	5	6.4
6:00	8	0.1	3.2
7:00	4	0.05	1.6
8:00	2	0.05	0.8
9:00	1.0	0.05	0.4
10:00	0.5	0.05	0.2

Example 2

freeHorn Canon
6/28/06

International Trumpet Seminar performance, 6/30/06
Enfield, New Hampshire
10 trumpets, computer, fretless elecric guitar

Two versions of the same program (duration and other settings not included here), run "in canon" by starting the second version of the program 5′35″ into the piece. Score example shows written score for trumpets in Bb, voice 1.

Harmonic progression (data for program)

1 : 1.083 : 1.166 : 1.125 : 1.0416 : 1

Ratio equivalents

1 : 13/12 : 14/12 : 27/25 :25/24 : 1

(Bb: 58.2705 Hz.)

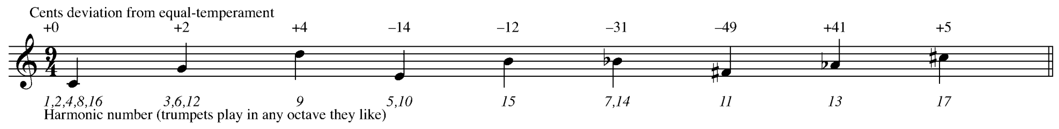
Voice 1, duration 720 seconds (12 minutes)
Voice 2, duration 445 seconds, (7:25 minutes)
Voice 2 starts a 4:35

freeHorn (canon)
trumpets (I)

Transposed (trumpet in Bb) score

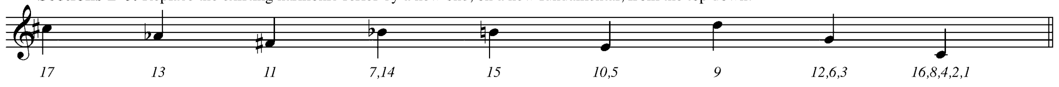
polansky

Section 1: Build up harmonic series on C



Cents deviation from equal-temperament
+0
+2
+4
–14
–12
–31
–49
+41
+5
1,2,4,8,16
3,6,12
9
5,10
15
7,14
11
13
17
Harmonic number (trumpets play in any octave they like)

Sections 2-6: Replace the existing harmonic series by a new one, on a new fundamental, from the top down.



Play notes heard, "around" notes heard, with simple sounds, different mutes. Be creative but don't try to stick out too much.

Form (7 sections, each about 1'03" long, new notes enter about every 6 seconds)

Section 1: Build up harmonic series on C
Then, gradually replace existing harmonic series, from the top down, by harmonic series on fundamentals:
- (**section 2**) a very wide minor second higher (13:12, or 141 cents)
- (**section 3**) a small minor third higher (7/6, or 269 cents)
- (**section 4**) a slightly wide major second higher (27/24, or 208 cents)
- (**section 5**) a very narrow minor second higher (70 cents)
- (**section 6**) the original C series

Finally, (**section 7**) drop out the original C series from the highest harmonic down to the lowest

Version for the The Chosen Vale International Trumpet Seminar, Ed Carroll, director. 1/5/07