

# A Manual for WIGOUT

Arun Chandra

arunc@evergreen.edu

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The synthesis algorithm for WIGOUT</b>	<b>3</b>
<b>3</b>	<b>Segments, states, and events</b>	<b>5</b>
3.1	Segments . . . . .	5
3.2	States . . . . .	5
3.3	Events . . . . .	6
<b>4</b>	<b>A WIGOUT example to try</b>	<b>8</b>
<b>5</b>	<b>Wiggles, and twiggles, and ciggles</b>	<b>10</b>
5.1	Wiggles . . . . .	10
5.2	Twiggles . . . . .	10
5.3	Ciggles . . . . .	12
5.4	Slanted wiggles, twiggles, and ciggles . . . . .	12
5.5	Mixing and matching . . . . .	15
<b>6</b>	<b>Command line options</b>	<b>16</b>
6.1	Setting an alternative output soundfile . . . . .	16
6.2	Setting the sampling rate . . . . .	16
6.3	Setting the automatic ramping . . . . .	16
6.4	Setting the verbosity . . . . .	17
6.5	Setting the data confirmation . . . . .	17
<b>7</b>	<b>Acknowledgments</b>	<b>18</b>

# 1 Introduction

WIGOUT is a program for composing and synthesizing groups of waveforms. You create three datafiles that specify the waveforms you want, WIGOUT reads them and generates a 2-channel WAV soundfile you can hear.

WIGOUT can generate arbitrarily complex combinations of waveforms of arbitrary length, limited only by the amount of memory you have on your computer.

For a general explanation of how a computer makes sound, and the common fundamentals of both WIGOUT and TRIKTRAKS, please see *The Synthesis Algorithms Used by TRIKTRAKS and WIGOUT*.

Whereas TRIKTRAKS allows you to compose the waveform's transformational paths, WIGOUT allows you to compose the waveform's shape and the rate at which it changes.

The three datafiles WIGOUT needs are

1. *segment* definitions, (*data.sg*)
2. *state* definitions, (*data.st*)
3. *event* definitions. (*data.ev*)

- A *segment* is the smallest part of a waveform. It is an element with additional modifiers.
- A *state* is a sequence of segments.
- An *event* is a state with a start time, end time, and stereo location.

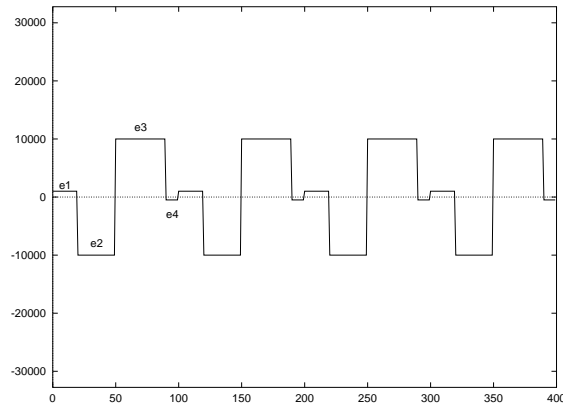
After these files have been created (and saved with the same root filename and distinguishing extensions), they are fed to WIGOUT like this:

```
wigout data
```

WIGOUT then reads in the files *data.sg*, *data.st*, and *data.ev*, synthesizes the sound, and places the samples in the output file *data.wav*.

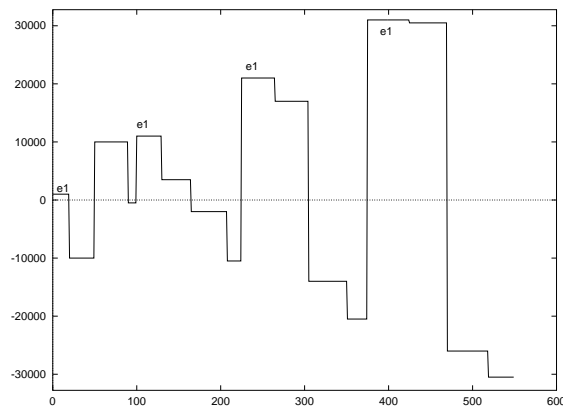
## 2 The synthesis algorithm for WIGOUT

Let's suppose you have the following waveform:



In this waveform, there are four elements: e1, e2, e3, and e4. These four elements are iterated (repeated) four times. At each iteration, each element remains the same: neither its amplitude nor its duration changes.

Now let's suppose that on every iteration, each element got progressively higher (or lower), and got longer (or shorter). Then the four iterations might look like this:



Elements e1 and e2 keep getting higher, and elements e3 and e4 keep getting lower. All four elements get longer.

Acoustically, this means that the sound will get louder (since the maximum displacement of the waveform is increasing) and lower in pitch (since the overall duration of the waveform is increasing).

### The Algorithmic Heart of WIGOUT

This is the heart of WIGOUT :

At each iteration, every element changes its amplitude and duration by a fixed increment. Both amplitude and duration grow (by their respective increments) until they reach their separate maxima, then start shrinking (by the same increment) until they reach their minima, then start growing again.

By way of example, let's suppose there are two variables  $A$  and  $D$ . The limits of  $A$  are 1 and 5, and its increment is 1. The limits of  $D$  are 2 and 8, and its increment is 2. Here's what happens when they start iterating together:

A	D
1	2
2	4
3	6
4	8
5	6
4	4
3	2
2	4
1	6
2	8
3	6
4	4
5	2
4	4
3	6
2	8
1	6
2	4
3	2
4	4
5	6
4	8
3	6
2	4
(etc.)	

In this simple example, each variable has a small range, and a proportionally large increment. So they begin their sequences again after 24 steps.

In WIGOUT ,

- The ranges are much larger ( $\pm 32767$  for the amplitudes, and between 1000 and 0 for the durations).
- The increments can be proportionally much smaller.
- There can be up to 64 segments in each state. That means, there can be up to 128 variables (and more, under certain circumstances).

So, the number of steps for *all* the variables to begin their sequences again can be enormous. In practical terms, a sound can last for hours before repeating.

## 3 Segments, states, and events

A *state* is a sequence of *segments*, and an *event* specifies when a state starts and stops. Three separate data files are used for the segments, states, and events.

### 3.1 Segments

A *segment* is the smallest part of a waveform. Like an element, it has an amplitude (height) and a duration (measured in samples). In addition, a segment also has a range (maximum and minimum values) and an increment of change.

A segment can be of one of three types: a wiggle, a twiggle, or a ciggle. These will be explained in full later.

In the “segments” file, you define the segments you want to use. This file must have the extension “.sg”.

Here’s a definition for one segment:

```
a1 wiggle           # identifier, type
10 20 1 1          # initial, max, min, inc (duration)
0 32000 -32000 4000 # initial, max, min, inc (amplitude)
```

- The first line declares *a1* to be a segment of type “wiggle”.
- The second line determines the duration of *a1*. The first item on this line is its initial duration, followed by its maximum duration, minimum duration, and increment. All values are given in samples.

So, the initial duration for *a1* is 10 samples, the maximum duration it will reach is 20 samples, and the minimum duration it will reach is 1 sample. On each iteration of this segment, the duration will change by 1 sample.

- The third line describes the amplitude of *a1*: its initial value, followed by its maximum value, minimum value, and increment.

The initial amplitude will be 0, it will increase to 32000, and decrease to -32000, and will change by 4000 on each iteration.

Every time this segment is iterated, its duration and its amplitude will increase by their respective increments. Once they reach their maxima, they will start shrinking by their respective increments. When they reach their minima, they will start increasing again.

Since their ranges and increments are not the same, they will reach their minima and maxima at different times.

#### Segment range limits

- Duration: maximum = 1000 samples, minimum = 0 samples.
- Amplitude: maximum = 32767, minimum = -32768.

### 3.2 States

A *state* is a sequence of segments.

The “states” file is a list of state id’s, followed by the sequence of segments that makes up that particular state.

The “states” file must be given the extension “.st”.

Here’s an example of a “states” file:

```
# id    segments
s1     a1 a2 a3
s2     a2 a3 a2 a3 a4
s3     b5 a3 b3 b2
```

This file declares that

1. State *s1* consists of 3 segments: a1 followed by a2, followed by a3.
2. State *s2* consists of 5 segments: a2 followed by a3, followed by a2 again, followed by a3 again, followed by a4.
3. State *s3* consists of 4 segments: b5, a3, b3, b2.

As should be obvious from this example, a segment can be repeated any number of times within a state.

A state is iterated from its start time until its end time. (The start and end times are given in the “events” file.) On every iteration, each of its segments changes its duration and its amplitude.

Since the frequency we hear is the sum of durations of all the segments of the state, and since the duration of each segment changes from iteration to iteration, so the resulting frequency also changes from iteration to iteration.

### State range limits

A state can have a maximum of 64 segments in it. The same segment may occur more than once within a state.

## 3.3 Events

An *event* is a state with a start time, end time, and stereo location.

The “events” file is a list of states, their start and end times (in seconds), and their stereo location (a number between 0 and 1).

The “events” file must have the extension “.ev”.

Here’s an example:

```
s1 1 5 0.5 # stateId, startTime, endTime, stereo location
s2 10 15 1
s3 0 12 0
```

This says that

1. State *s1* will begin at time 1 second, iterate until time 5 seconds, and will be in the center of the stereo field (0.5).
2. State *s2* will begin at 10 seconds, iterate until 15 seconds, and will be all the way to the right in the stereo field (1).
3. State *s3* will begin at 0 seconds, iterate until 12 seconds, and will be all the way to the left in the stereo field (0).

### **Event range limits**

The start and end times must be positive floating point numbers. The maximum duration is limited by the amount of memory on your system and the sampling rate you choose.

The stereo field is a number between 0 and 1, inclusive, with 0 meaning “left channel only” and 1 meaning “right channel only”.

## 4 A WIGOUT example to try

Here's a complete WIGOUT example for you to try. Use your favorite word processor, type in the three following files, and remember to save them as plain text files (sometimes called "DOS files"). Save them in the same directory as your WIGOUT executable.

All three files must share the same name, but have distinguishing extensions:

```
.sg : for the "segments" data
.st : for the "states" data
.ev : for the "events" data
```

In a datafile, WIGOUT will ignore blank lines, multiple spaces (or tabs), and anything following a hash mark (#).

Here's the segments file. Type it in, and save it as a file called *try.sg*.

```
#
# An example segments file
#
w1 wiggle
20 10 60 0.001      # duration (init, max, min, inc)
-15000 20000 -20000 40 # amplitude (init, max, min, inc)

w2 wiggle
40 40 10 0.002     # duration
11000 20000 -20000 80 # amplitude

w3 wiggle
20 20 40 0.003     # duration
700 20000 -20000 50 # amplitude

w4 wiggle
30 30 20 0.004     # duration
-15000 20000 -20000 90 # amplitude
```

And here's the states file, defining three states. Type this in, and save it as a file called *try.st*.

```
#
# An example states file
#
s1 w2 w3 w1 w4      # state id, segment ids.
s2 w1 w4 w2 w4 w3 w4
s3 w4 w3 w1
```

And here's the events file, defining four events. Save this as a file called *try.ev*.

```
#
# An example events file
#
s1 0 0.5 1          # state id, start, end, stereo
s2 0 3 0
s1 1 2 1
s1 2.5 4 1
s3 4 6 0.5
```

Now, run WIGOUT on your datafiles, like this:

```
wigout try
```

WIGOUT should respond with this:

```
wigout try
Reading in datafiles try.sg try.st try.ev
4 segments read in
3 states read in
5 events read in
creating a 6 second sound

1 of 5 events ... 102 states
2 of 5 events ... 400 states
3 of 5 events ... 204 states
4 of 5 events ... 306 states
5 of 5 events ... 642 states amplifying by 6.02 dB done
writing sound to try.wav
```

If needed, correct any input errors that might be reported. If none are reported, you should get a soundfile called *try.wav*, which you can play on your system.

## 5 Wiggles, and twiggles, and ciggles

A segment can be one of three types: a *wiggle*, a *twiggle*, or a *ciggle*. I'll explain their differences and how they are defined.

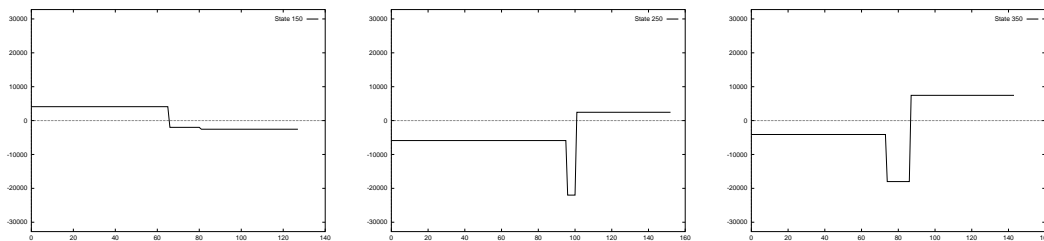
### 5.1 Wiggles

A *wiggle* is essentially a square wave whose amplitude and duration you can control.

A wiggle has a duration and an amplitude, both of which can be independently changing. Here's an example of a wiggle segment definition:

```
w0 wiggle          # identifier, type
20 40 10 0.5       # duration: initial, max, min, increment
1000 20000 -10000 500 # amplitude: initial, max, min, increment
```

Here are some plots of a state with three segments, and each segment is a wiggle. The three plots are a few hundred iterations apart in time (i.e., they are *not* sequential iterations).



#### Wiggle range limits

- Duration (in samples): maximum = 1000, minimum = 0.
- Amplitude: maximum = 32767, minimum = -32768.

### 5.2 Twiggles

A *twiggle* is a triangular wiggle.

Like a wiggle, a twiggle has a duration and an amplitude which can vary independently of each other. These are called the “base amplitude” and the “base duration”, corresponding to the “base” of a triangle.

In addition to these variables, the twiggle also has a “peak amplitude” and a “peak location”, corresponding to the “peak” of a triangle. These can vary independently of each other, and independently of the base.

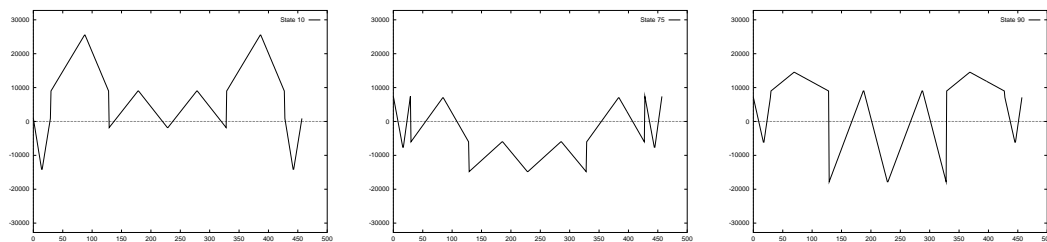
The peak amplitude is the height of the peak of the triangle.

The peak location is the location of the triangle’s peak relative to its base. (This can only be a value between 0.1 and 0.9, representing a proportion of the length of the base.)

Here’s an example of a twiggle segment definition:

```
t1 twiggle          # id, type
30 60 30 .01        # base duration (initial, max, min, inc)
0 8000 -8000 100    # base amplitude (initial, max, min, inc)
-15000 0 -30000 100 # peak amplitude (initial, max, min, inc)
.5 1 .25 .001       # peak location (initial, max, min, inc)
```

Here are plots of a state with 6 segments, and each segment is a twiggle. The three plots are a few hundred iterations apart in time.



### Twiggle range limits

All ranges are inclusive of their endpoints.

- Base duration (in samples): 0 and 1000.
- Base amplitude: -32768 and 32767.
- Peak amplitude: -32768 and 32767.
- Peak location: 0.1 and 0.9.

### 5.3 Ciggles

A *ciggle* is a *twiggle* with curved sides.

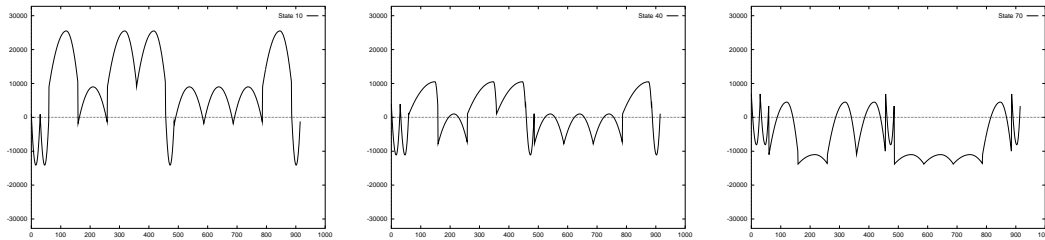
The general function used to calculate the curves was written by Jerry Keiper, formerly of Wolfram Research, Inc.:

$$y = y1 + (x - x1) \left( \frac{-y1 + y2}{-x1 + x2} + \frac{(x - x2) \left( \frac{y1 - y2}{-x1 + x2} + \frac{-y2 + y3}{-x2 + x3} \right)}{-x1 + x3} \right)$$

The data specifications for a *ciggle* are identical with those for a *twiggle*, except the modifier “curved” is added after the word “*twiggle*”:

```
c3 twiggle curved      # id, type, modifier
100 150 100 .01        # base duration (initial, max, min, inc)
0 0 -20000 200         # base amplitude (initial, max, min, inc)
0 20000 -20000 1000   # peak amplitude (initial, max, min, inc)
.5 0.75 0 .001        # peak location (initial, max, min, inc)
```

Here are three iterations of a state with 12 segments, and all are *ciggles*. Again, the states do not immediately follow one another in time.



#### Ciggle range limits

These ranges are the same as those for *twiggles*. All ranges are inclusive of their endpoints.

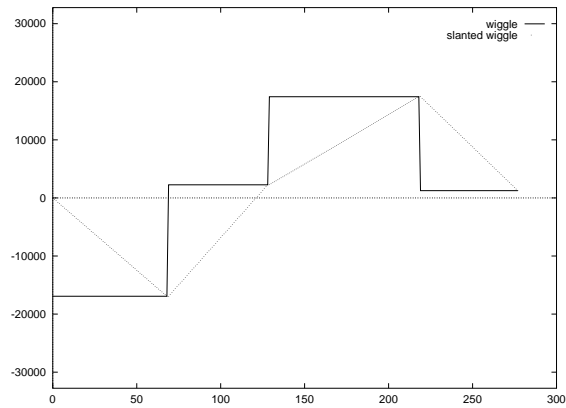
- Base duration (in samples): 0 and 1000.
- Base amplitude: -32768 and 32767.
- Peak amplitude: -32768 and 32767.
- Peak location: 0.1 and 0.9.

### 5.4 Slanted wiggles, twiggles, and ciggles

Any three of these segment types can be given the modifier “slanted”.

When a segment is “slanted”, that means its initial amplitude is the current amplitude of its previous segment, and its final amplitude is its specified amplitude.

Here is a plot of 4 slanted wiggles superimposed on the same 4 not-slanted wiggles.



This is how you would request slanted segments in a datafile.

```

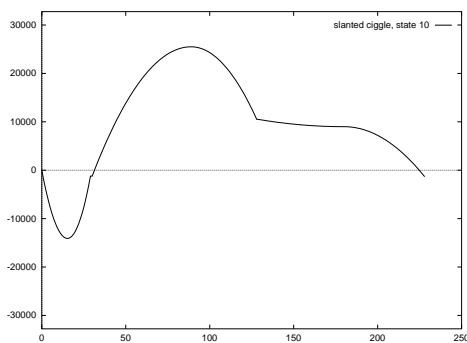
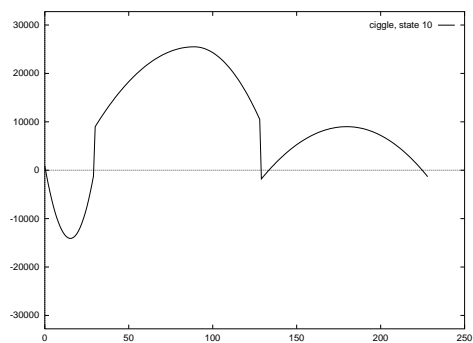
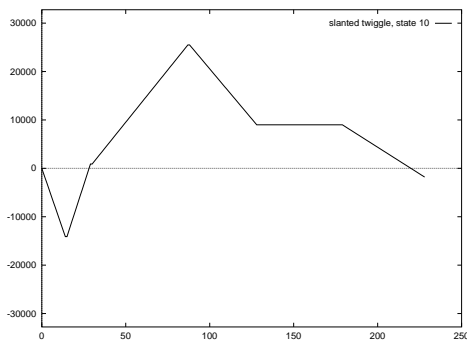
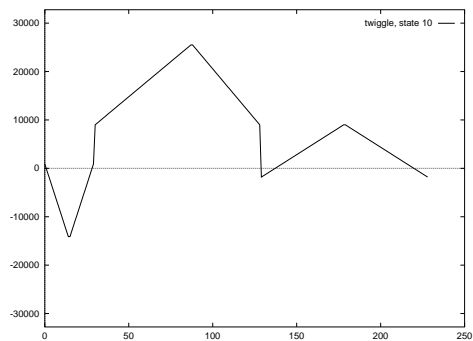
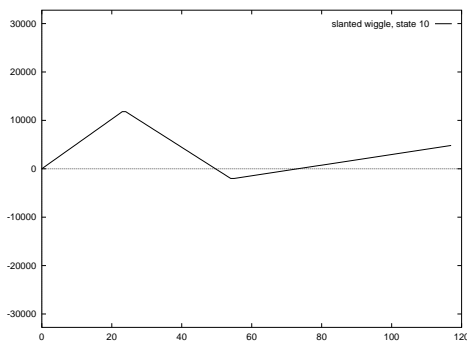
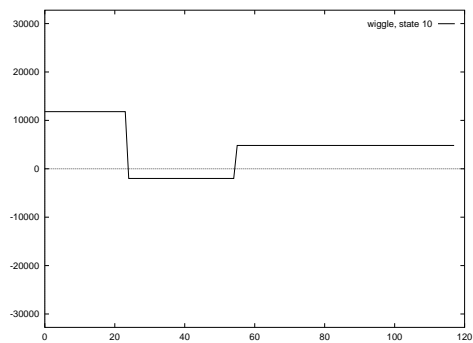
wsl wiggle slanted      # A slanted wiggle
20 25 10 .5            # duration
10000 20000 -20000 200 # amplitudes

tsl twiggle slanted    # A slanted twiggle
30 60 30 .01          # base duration
0 8000 -8000 100      # base amplitude
-15000 0 -30000 100  # peak amplitude
.5 1 .25 .001        # peak location

csl twiggle curved slanted # A slanted ciggle
30 60 30 .01          # base duration
0 8000 -8000 100      # base amplitude
-15000 0 -30000 100  # peak amplitude
.5 1 .25 .001        # peak location

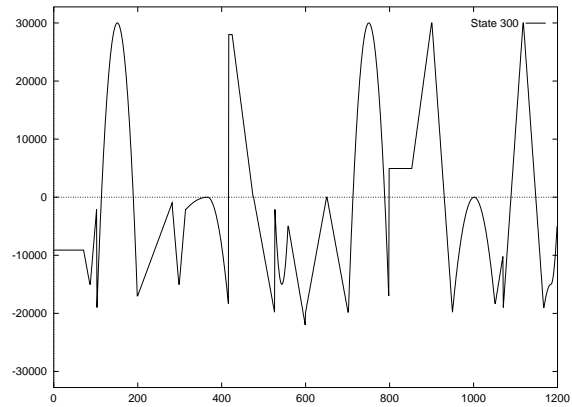
```

Here are pairs of plots of identical waveform. The ones on the left are not slanted and the ones on the right are. The first is a pair of wiggles, the second of twiggles, and the last of ciggles.



## 5.5 Mixing and matching

When building states, any of the segment types described above can be used with any other. Here's a plot of a state which has 18 segments and uses all the described types:



## 6 Command line options

Here are the command line options that WIGOUT will accept:

- v: verbose (print the contents of the waveform being generated)
- V: super verbose (print everything).
- C: confirm segment data (bounds, name duplication).
- r *N*: ramp start and end of each event. *N* = length in samples.
- o *file*: sound to *file*.
- R *N*: set sampling rate to *N*. Default: 22050.

### 6.1 Setting an alternative output soundfile

By default, WIGOUT will create a soundfile that will have the same name as your input datafile. The extension “.wav” will be added to the name.

If there is a soundfile with the same name, WIGOUT will prompt you for confirmation before overwriting it.

If you'd like to specify the name of the soundfile on the command line, here's how you do it:

```
wigout -o new.wav datafile
```

Now the samples generated from “datafile” will be written to the file “new.wav”.

### 6.2 Setting the sampling rate

The default sampling rate for WIGOUT is 22050 samples per second. The length of the sounds you can produce is limited by the amount of available RAM you have on your computer.

If you're trying to make a relatively long soundfile, and WIGOUT is crashing due to lack of memory, you can reduce the sampling rate on the command line.

```
wigout -R 11025 datafile
```

This will set the sampling rate to 11025 samples per second.

WIGOUT will accept sampling rates of: 11025, 22050, 44100, 48000.

### 6.3 Setting the automatic ramping

All the waveforms that you enter into your “events” file will have an instantaneous on. This can be annoying. As a way of dealing with the annoyance, you can specify a “ramp” which will ramp up every waveform the same amount.

```
wigout -r 100 datafile
```

This will cause every sound to ramp up with a duration of 100 samples, which is very short.

The most useful lengths for ramps are between 100 and 500 samples.

## 6.4 Setting the verbosity

The options `-v` and `-V` will set the “verbosity” of the the displayed output. This may be helpful when trying to track down an error in the input. `-V` gives more information than `-v`.

Here’s how to use it:

```
wigout -V datafile  
wigout -v datafile
```

## 6.5 Setting the data confirmation

Using the option `-C` on the command line will cause `WIGOUT` to confirm whether the input data is acceptable.

This option is only marginally useful.

## 7 Acknowledgments

So that's WIGOUT ! I hope you have some fun with it, and please drop me a line if the program crashes on you, or just stares blankly back at you from the computer screen.

I'd like to thank Professors Jim Beauchamp and Sever Tipei, co-directors of the Computer Music Studios, University of Illinois, who were encouraging to my work, and generous with access to the Studios.

Herbert Brün and Keith Johnson taught me about SAWDUST, the principles of its algorithms, and its radical address to compositional premises.

Robert Naiman and Jerry Keiper, both formerly of Wolfram Research, Inc., generously contributed their skills in mathematics and numerical analysis, and patiently answered the many questions I had on implementation.

I prototyped the synthesis algorithms for WIGOUT using *Mathematica V2.1*, then rewrote them in C, under NeXTStep 3.0. Since then, WIGOUT has been ported to an IBM RS6000 running AIX 3.1, an SGI Indy running IRIX 5.3, an SGI O2 running IRIX 6.3, and an Intel PC running Windows95. On all these machines, the GNU C compiler was used to compile the code.

The graphics in this article were produced with *gnuplot*, version 3.5, written by Colin Kelley and Thomas Williams.