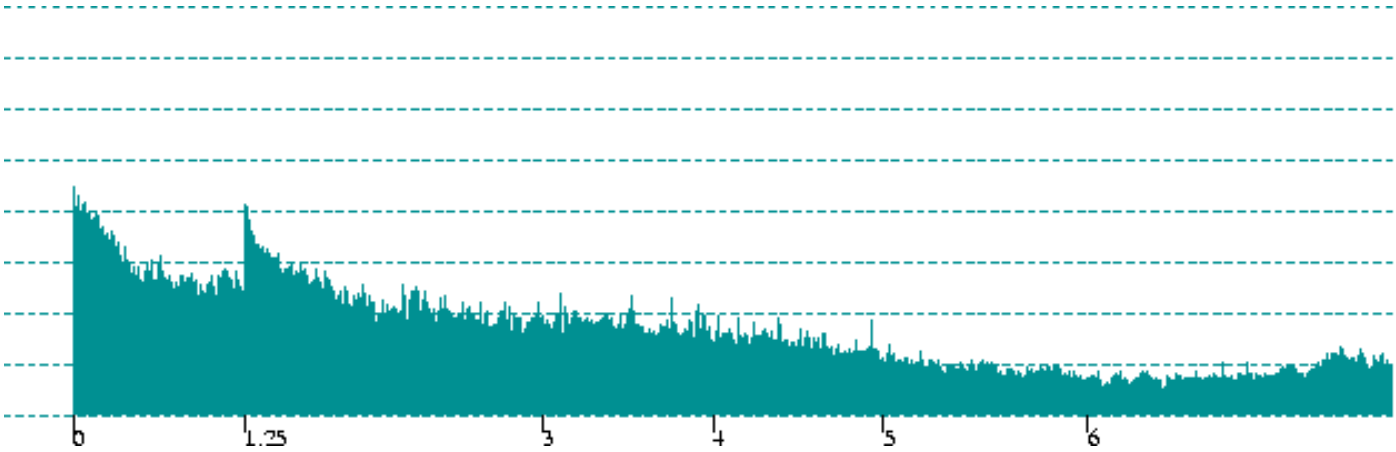
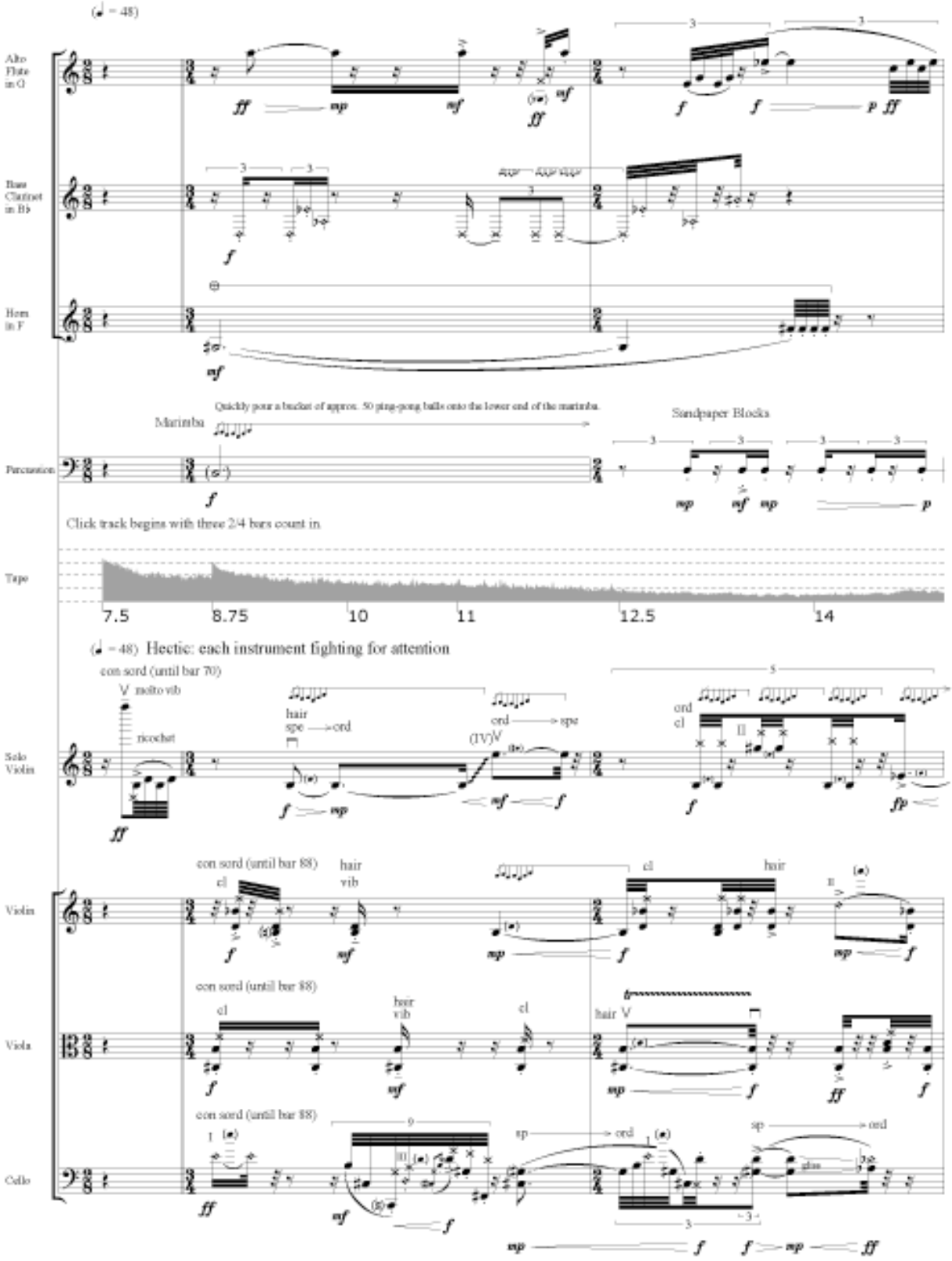
`rmsps`

is a command-line program that analyses sound files and creates graphics of the RMS (loudness) curve (loosely termed the wave form) in Encapsulated Postscript (EPS) form. It was written to generate an arbitrary number of graphics files, in one pass, for inclusion in traditional music scores of pieces for instruments and tape or digital sound, but is not limited to such uses. Below is an example of the output of `rmsps`. RMS is represented vertically and time in seconds horizontally:



Traditional scores space music according to a non fixed-width spacing algorithm that results in different relative widths being assigned to, for example, a sixteenth note and a half note (i.e. a half note does not necessarily take up eight times more horizontal space than a sixteenth note). This makes the synchronization of wave form representations of sound (which are by nature fixed-width) with notes in a score quite problematic.

`rmsps` offers a solution to this problem by allowing segmentation of the analysis: in one output EPS file there can be several segments, and in each segment the width occupied by one second of sound can differ. In this way, arbitrary durations of sound can be displayed over arbitrary image widths thus securing visual alignments of events. For example, there is an attack in the tape on the first beat of the first two bars in my piece *slippery when wet* as is clearly visible in the score:



The synchronization of the tape attacks with the first notes of these bars was achieved by specifying to `rmsps` the alignment of the coordinates of these notes (in whichever scale is convenient: cm, inches, pixels, percent, etc.) with the corresponding real-time values in seconds (see below for more details).

Supported platforms

`rmsps` is intended for UNIX platforms and has been tested on Redhat Linux 7.1 (but should work on all Linux systems), Silicon Graphics with IRIX 6.5, and Mac OSX computers. It also compiles and runs as a command line program under Windows (Win32, see below).

Note that if you want to build the program on IRIX, you will need the Silicon Graphics C++ compiler "CC", or, if you like, you can try your luck with gcc.

Failing that, `rmsps` is an ANSI C++ program that should be easily portable to every system.

Building the program

On Windows, you can either compile the sources yourself or just download the Win32 executable. Compiling should be easy as no special options are needed for the Microsoft Visual C++ compiler. Just make a new project, insert the C++ source files and compile.

If you're running any other platform, download the zip archive, open a shell window, cd to the directory where you downloaded `rmsps` and type:

- `unzip rmsps1.03.zip`
  [or whatever you have to do to unarchive/decompress this file]
- `cd rmsps`
- `make`
  [if you're running linux or]
- `make -f Makefile.sgi`
  [if you're running irix or]
- `make -f Makefile.osx`
  [if you're running Mac OSX]

The program is created in the current working directory and is called `rmsps`.

For me, the program compiles and runs without any errors or warnings on all three systems. I've had complaints, but can't get it to fail myself. The SGI seems particularly tricky. First of all, the "cc" compiler will not work as this is only a C compiler, not C++. Secondly, I suspect that on the SGI there might be some problems with the compiler and newline/linefeed combinations: if you get lots of errors, try converting the source code (*.cpp, *.h, Makefile*) to UNIX format (with the to_unix command) before doing a make. On Mac OSX I've also heard of segmentation faults when running rmsps. Again, this doesn't happen to me, but I saw it once when I used the optimization flag to the compiler. Hence

optimization is not called in the Makefile.osx. If you changed this, expect problems.

Program usage

`rmsps` is started from the command line (there is no graphical user interface) and takes only one argument: the path of the input file. This is a text file in which all the analysis parameters are defined. Simple comments may be inserted into the input file: they begin with the # character. Lines beginning with # will be ignored but mid-line comments (e.g. "`this is to be read # this is not`") are illegal. Program parameters, e.g. the path of the sound file, the width of the image etc. (see below) come first but may be changed at any point in the input file in order to have different parameters for different output EPS files. Most parameters have reasonable defaults but obviously the sound file, image width and height, and xscale (again, see below) have to be given in order for the program to run. Program parameters are of the form `parameter=value` and several may be given on a single line (separated by space, see below for an example).

Once the program parameters have been defined, the page parameters come next. Program parameters and page parameters are not allowed in the same line of text. Page parameters consist of one line of text (they may not be separated over two or more lines) for each page to be written. Each line specifies the output EPS file followed by the analysis coordinates. These consist of simple "x-start sound-file-start" pairs, where x-start is the point at which we should start drawing in the output EPS file (relative to the xscale program parameter) and sound-file-start is the analysis start point in the sound file (given either as seconds or minutes:seconds). The analysis continues to the point specified in the next pair, hence it is important always to specify the last point. As an extreme example, with almost five minutes of sound squashed into 50% of the image width, followed by half a second of sound in 40%, take the following page parameters:

```
p01t.eps 10 0    20 1.25    120 5:00    200 5:00.5
```

These specify that `rmsps` should write in the file p01t.eps from x point 10 to 20 the rms analysis of seconds 0 to 1.25, then from x point 20 to 120 the analysis of 1.25 seconds to 5 minutes (5:00), then from x point 120 to 200 an analysis of 5:00 to 5:00.5. My arbitrary xscale here is 200. The result of `rmsps` using the tape part of my *slippery when wet* and these parameters is:

Program Parameters

parameter	description	default value
sndfile	The (relative or full) path of the sound file to be analysed. This can be an AIFF, Next/Sun SND, or WAVE file, and it will be automatically recognised irregardless of file extension. Files of any number of channels (and any endianness) are supported but are limited at the moment to 16-bit linear.	NO DEFAULT: REQUIRED!
width	The width of the output image in your (optionally specified) units.	NO DEFAULT: REQUIRED!
height	The height of the output image in your (optionally specified) units.	NO DEFAULT: REQUIRED!
xscale	The maximum x value. This allows you to think on any scale you want and the correct scaling will be applied to convert your given x values to your given width. You can think in percent, between 0.0 and 1.0, 0 and your defined width, whatever you want.	NO DEFAULT: REQUIRED!
units	The units for 'height' and 'width.' Can be 'cm' (centimeters), 'in' (inches), or 'pt' (postscript points, where there's 72 points per inch).	cm
read-offset	An offset in seconds (or mins:secs) to be added to all the sound file start points given in the page parameters	0.0
write-offset	An offset in seconds (or mins:secs) to be added when writing the time scale in the	0.0

	postscript file. N.B. This is independent of read-offset, i.e. if we have a page parameter start time of 12.5, a read-offset of -10.0 and a write-offset of 1.0, then the time written will be 13.5, not 3.5.	
rms-size	The size of the RMS window in milliseconds. You can generally leave this at the default but if you're drawing, say, one minute of sound or more in one file, you may want to increase this (e.g. to 100) to reduce output image file size. Obviously, the smaller the RMS size, the more accurate the representation of the sound, but there's always a trade off.	10
rms-scaler	Your defined height represents RMS values of 0.0 to 1.0. Sometimes your RMS is consistently low so you might want to (vertically) scale the output of the RMS routine to make your sound more visible.	1.0
rms-exponent	The RMS can also be raised to an exponent in order to compress the vertical range displayed. This is useful when the RMS ranges over large values: if an exponent less than 1 is used, low RMS values will be made more visible.	1.0
gray	The grayscale fill shade for the wave form. 0.0 is black, 1.0 is white, 0.5 is 50% gray etc.	0.0 (black)
line-thickness	The thickness in postscript points of the lines used in the time and amplitude scales.	0.2
outdir	The directory in which the EPS files should be written.	The current working directory

hlines	How many dotted horizontal lines should be used to draw the vertical scale across the full width of the wave form. This does not include the base line, which is always drawn.	8
colour	You can optionally replace grayscale with this command to get a colour wave form fill. The argument is an RGB (red, green, blue) value of the form "r,g,b" i.e. three values between 0.0 and 1.0. The values are separated by commas but NO SPACE!	None, default fill is grayscale 0.0
font	The postscript font used to write the time scale.	Times-Roman
font-size	The font size for 'font.'	8pt
time-line	Whether you want the times in seconds to be printed at the bottom of the image or not. If you don't want this, be sure to write 'no', otherwise it will be printed.	yes

Example Input File

```
sndfile=/snd/michael/slippery-tape-181100.wav
outdir=/usr/people/musik/michael/eps units = cm
width=15.85  height = 1.3  xscale = 200  rms-size =
10 rms-scaler=1    colour=0,.5,.5 hlines = 4
font=Verdana font-size=10  # we're going to write
three EPS files but note that with this # one input
file I could generate all graphics necessary for # a
score, then change the program parameters above and
# regenerate the graphics in different sizes, colours
etc.  p01t.eps  10 0.000    34.66 1.25    126.67 5
200 7.5 p02t.eps  8.07 7.500    68.5 10.000    144.77
12.500    200 15 # change the width for the next file
width = 22 # NB we're 11 minutes into the file now
p33t.eps 8.46 11:15  67.25 11:17.5  122.15 11:20  200
11:22.5
```