resample.doc

COLLABORATORS			
	TITLE : resample.doc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 14, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 resample.doc 1

1.1 resample.doc 1

1.2 What does it do? 1

1.3 Who needs it? 2

1.4 The filters that we use 2

1.5 Command line arguments 4

1.6 Examples 5

1.7 The 16bit version 6

1.8 Bugs and Features 7

1.9 History 7

1.10 Credits 7

Chapter 1

resample.doc

1.1 resample.doc

"Resample"/"Rs16":
Change the size and sampling rate of digital audio by an integer
ratio (e. g. 1:2), but avoid aliasing.

What does it do?

Who needs it?

The filters that we use

Command line arguments

Examples

The 16bit version

Bugs and Features

History

Credits

1.2 What does it do?

I hope you are familiar with the problem of aliasing in digitized sounds,
just let me summarize so much:

When playing a sample, the output sound must be low pass filtered, with a
cutoff frequency *half* the playback rate, else you will hear an
additional high frequency "mirror image" of your sample. (That's what the
despised 7 kHz-filter in the Amiga is for: so you can safely play samples
at 14 kHz.)

The same type of low pass filter must be applied when recording a sound sample, else freqencies higher than half the sampling rate will be mirrored into a lower frequency band. Such aliasing at low frequencies is even more annoying, and there is no way of removing it later.

And even when you just want to change the sampling rate of an existing sample by discarding or inserting data (decimation/interpolation), a low pass filter must be applied. This time it must be a digital filter, and here is where my program comes in.

Please note that a fast CPU is highly recommended for digital filtering. For each output sample our n-point FIR filter has to perform n integer multiplications (and additions), where typical values for n are 21, 31, or even higher.

1.3 Who needs it?

Why would one want to change the size and sampling rate of a sample (i. e. interpolate or decimate it) anyway?

- If your audio filter is constantly switched off, you should notice that samples with a low playback rate kind of chirp and just sound bad. Most of them don't have to, try and interpolate them.

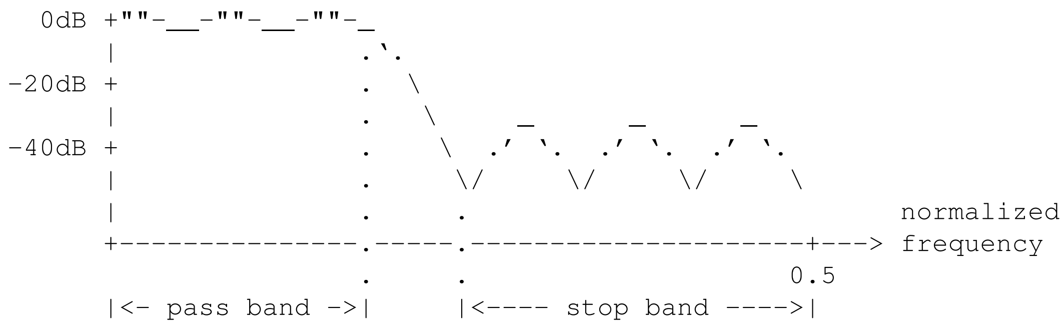
- If you want to record an ordinary 14 kHz-sample, but suspect that your audio input is not limited to 7 kHz bandwidth, record it at 28 kHz instead to avoid aliasing, then decimate it to the proper sampling rate of 14000/sec. (Note: TechnoSound Turbo II contains a "compress" option, that performs just this kind of decimation. But their built-in low pass filter is poor, so it doesn't really solve the problem.)

- CD audio tracks (which, for example, AsimCDFS lets you access directly as files) come in a resolution of 44 kHz/16 bit. Unfortunately the original Amiga audio hardware is limited to about 28 kHz playback rate (and 8 bit, but that's not the point here), so one usually decimates them to 22 kHz (or even 15 kHz, if you like). Doing this on the fly during playback, like e.g. Play16 can do, is not the best choice, because it only allows for simple filtering techniques like averaging, so aliasing cannot really be avoided.

1.4 The filters that we use

This is the frequency response you can expect from a digital low pass filter as designed by the Remez Exchange algorithm, with a moderate filter length of about 15.

```
 magnitude
      ^                                good quality FIR filter
      |
```



You see that it is imperfect, because

- there are ripples in the pass band, distorting the signal that passes through the filter

- there are ripples in the stop band, limiting the attenuation of the unwanted frequencies

- there is a significant gap between pass band and stop band (called transition band), where the behaviour of the filter is more or less undefined.

Please don't blame this on the design algorithm, it creates *optimal* filters (to be precise: Chebyshev optimal) for any given filter length, no more and no less. (You should see the frequency responses of filters designed by "window techniques".)

What can you do to influence the result?

- You have direct control over the pass bandwidth, but the narrower the transition band, the bigger the ripples (-p option).

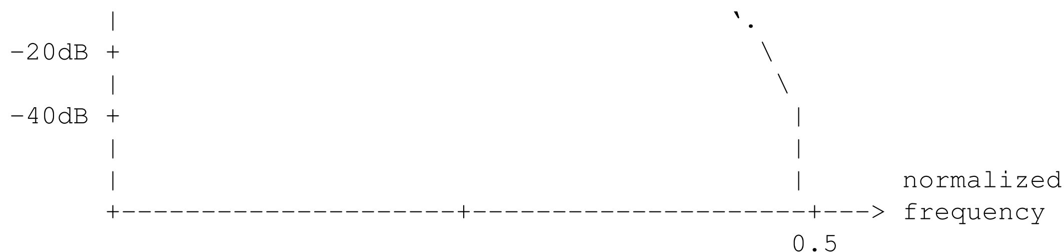
- You can exchange ripple size between pass band and stop band by putting more weight on stop band errors (-w option).

- You can increase the filter length to reduce the overall ripple size (-l option).

Normally all you have to do is specify an interpolation/decimation ratio (-i/-d) and let the program do the rest. It will print out the parameters it has chosen, plus some characteristics of the resulting frequency response. The most important one is the actual stop band attenuation, which will be by default about -40 to -45 dB.

Just as a side note: Why is simple averaging of samples a bad idea? Mathematicians could explain you that this is equivalent to using a two-point FIR filter, and its frequency response has the shape of a cosine function. That doesn't sound so bad - until you've seen it plotted on a logarithmic scale like in the diagram above:

```
 magnitude
     ^                                    poor FIR filter
     |
 0dB +------------------------------_________
```



1.5 Command line arguments

```
Argument:                   Default value:

<input file>                ---
<output file>               <input file>.out
-i<interpolation ratio>     1
-d<decimation ratio>        2  (or 1, if interpolation<>1)
-l<filter length>           21  (or more, depending on -i/-d)
-w<weight of stop band>     5.0
-p<pass band>               0.7 (70%, i.e. 30% transition band)
-g<gain>                    1.0
-v[erbose]
```

-i, -d: The stop frequency is automatically adjusted to eliminate aliasing in the desired interpolation/decimation process. To be precise,

$$f_{stop} = \frac{0.5}{\mathrm{Max}\ \{\ \mathrm{int.}\ |\ \mathrm{dec.}\ \}}$$

is required. So for example 3:1 and 2:3 both need a lower stop frequency than 1:2 or 2:1. I'm telling you all this, because there is an important side effect: with a lower stop frequency, the transition bandwith will decrease, too. And that means, the program has to choose a longer filter to do the job.
To put it in a nutshell: It's a bad idea to use complicated int./dec. ratios like 4:5 or 5:8 etc. They will cause long computation times.

-l: The longer the filter, the better its performance. But filter design and especially the filtering itself take longer, the more coefficients you demand. If the stop band attenuation is about -40 to -45 dB, the filter is good enough. (That's the attenuation needed to perfectly cancel any aliasing noise at 8 bit output resolution, and it may often be good enough for 16 bit, too.) The default value is not a constant but is always adapted to the specified interpolation / decimation ratio.
Please note that odd filter lengths should be preferred. Even-sized FIR filters are not generally bad, but the Remez Exchange algorithm has problems designing them.

-w: This allows exchanging ripple size between stop band and pass band. The default of -w5 will result in a pass band ripple of about 5 %, which I consider acceptable. If you specify less than 5, you will have to increase the filter length (-l) to maintain the required stop band

```
    attenuation.

-p: Relative size of the pass band, the bigger this number, the smaller the
    transition band will get. Don't put too much ambition in creating a
    narrow transition band: *there is no such thing* as a perfect low pass
    filter, just like there is no perpetuum mobile. The analogue audio
    filter in the Amiga, for example, has a transition band from 4 kHz to
    7 kHz. So the default of -p0.7 is already better than that, but if you
    must, go ahead, try -p0.8 or -p0.9. And don't forget to increase the
    filter length (-l), too.

-g: You can control the "volume" of the output sample with this option.
    This has nothing to do with the volume setting in the IFF VHDR chunk,
    it changes the sample values themselves. That means that you shouldn't
    play with this switch, but for 8bit output it can help to achieve
    better utilization of that limited sampling range. This is especially
    true when converting from 16bit to 8bit: I've found that CD tracks
    often sound better when converted with -g1.5 or even -g2.
    It is a good idea to keep an eye on the progress indicator when using
    this option. Make sure that the overshoot ratio doesn't exceed 1%,
    and that the peak level doesn't get above 150%.

-v: Well, actually the program output is always verbose ;-), but with this
    option enabled the filter coefficients are printed, too, in case they
    mean something to you.
```

1.6 Examples

```
The example file "wobble.iff" contains an artificial (i. e. calculated,
not sampled) sine wave of continuously rising frequency. Its playback rate
is 8000 samples/sec (rather low), and the maximum sine frequency is 4 kHz
(as high as the sampling rate allows).

The command "play" means a sound sample player like DSound or Play16, but
MultiView will do as well, if you have it. (You might even load the output
files into a sample editor to play them: Some of the filter effects are
easier to understand when you can see the wave, not only hear it.)

Now try the following commands (indicated by the prompt sign '>'):

> play wobble.iff

This will sound a bit strange: there is the rising sine wave, alright, but
some other whistling sounds, too. That's because the audio filter cannot
fully suppress playback aliasing at this low sampling rate.

> resample wobble.iff -i2
> play wobble.iff.out

This raises the playback rate to 16000 samples/sec. No more whistling
sounds.

> resample wobble.iff -d2
> play wobble.iff.out
```

```
This makes things worse, as it reduces the sampling rate to 4000/sec,
which is extremely low. But the interesting part: the second half of the
sine wave has gone. It was filtered out, because it contained too high
frequencies for the given sampling rate.
What would have happened without the filter? Try this:

> resample wobble.iff -d2 -l1
> play wobble.iff.out

Stop band attenuation is now 0 dB, because a 1-point FIR filter can do
absolutely nothing. The second half of the sine wave has been transformed
to an artificial sound, i. e. one, that was certainly not in the input
file. That's aliasing.

Now one last example, this time more complicated. Let's say you want a
playback rate of 12000/sec instead of the original 8000/sec. To achieve
this, you need an interpolation/decimation ratio of 3:2. As mentioned
above, the filter length will be automatically increased to maintain a
stop band attenuation of -40 dB, so just type:

> resample wobble.iff wob1 -i3 -d2

Or you could insist on a filter length of 21 but allow a wider transition
band (the -p value was found by trial and error):

> resample wobble.iff wob2 -i3 -d2 -p0.5 -l21

When you compare the results

> play wob1
> play wob2

you should notice that the end of wob2 (the part with the highest
frequencies) sounds a bit softer, in deviation from the original sound.
That's (at least in this case) the price of faster filtering.
```

1.7 The 16bit version

```
"Rs16" takes the same parameters as "Resample", plus the following:

    -sm     input samples are 16bit motorola (*)
    -si     input samples are 16bit intel
    -sb     input samples are 8bit
    -c1     one input channel (mono)
    -c2     two input channels (stereo) (*)

where (*) are the default values. The output format will be the same as
the input, or you can change it by -Sm, -Si, -Sb, -C1, -C2.

Note that the current version of "Rs16" only works on raw audio data. Take
it or leave it. :-(

In the following example we convert a CD track from its raw 16bit 44100 Hz
```

stereo format to 8bit 22050 Hz mono (which happens to be the default format
that Play16 assumes when it encounters raw data). Sorry for the long path
names, please think of the first two lines as a single shell command:

```
> Rs16 CD0:CDDA_MostSignificant/Indestructible DH3:Indestructible
    -Sb -C1 -g1.5
> Play16 DH3:Indestructible
```

Of course, if your CDDA track has intel byte ordering, you would also need
the -si option. And depending on the original recording, a different -g
parameter might be more appropriate. (And when in doubt: It shouldn't hurt
much not to use the -g option at all.)

1.8 Bugs and Features

- Rs16 badly needs to know some kind of file format other than "raw". But
  please let me know if anyone really needs such a feature, because _I_
  can do without it.
+ Samples aren't loaded into memory, so arbitrarily long files can be
  processed.
- Source and destination file must not be the same, and must be on devices
  that allow fseek() (which excludes PIPE: for example).
+ When you press Ctrl-C while an IFF sample is being filtered, the output
  sample will of course be incomplete, but nevertheless a valid IFF file,
  so you can still listen to the result. (Well, at least if it was a mono
  sample.)
- Cannot handle compressed samples (and I don't intend to change that).
+ Both looped and stereo IFF samples should be handled OK.
- I have honestly no idea what will happen to 8SVX instrument samples
  with more than one octave.

1.9 History

v1.1 (Dec 1999)
- added the 16bit version "rs16"
- better progress indicator

v1.0 (Jul 1996)
- first release (no version string included)

1.10 Credits

Compiler: GNU C 2.7.0, using libnix
Editor: CygnusEd Professional V4.2
AmigaGuide formatter: Text2Guide by Stephan Suerken

The Remez Exchange Algorithm (for design of Chebyshev optimal FIR linear

phase filters) was originally implemented by James H. McClellan,
Thomas W. Parks and Lawrence R. Rabiner, and published as a Fortran
program in:

Lawrence R. Rabiner/Bernard Gold: Theory and Application of Digital
Signal Processing. London: Prentice-Hall, 1975.

This program is freeware. Use it as you wish, but at your own risk.
If you like it, drop me a line:

Wilhelm Noeker <wnoeker@t-online.de>
Hertastr. 8
44388 Dortmund
Germany

Thanks to:

- Simon N Goodwin, for helpful discussion about the 16bit version