AudioFAQ

Jarkko Vatjus-Anttila <quaid@kempele.fi>

COLLABORATORS			
	TITLE : AudioFAQ		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Jarkko Vatjus-Anttila <quaid@kempele.fi>	July 16, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

AudioFAQ

1.1 Audio FAQ -- A guide to sound effects

```
                Audio.FAQ -- A guide to sound effects (v1.0)
-------------------------------------------
  - (c) 1996-7 by Jarkko Vatjus-Anttila

1. Information
1.1
               Disclaimer
                Read this before anything!
1.2
               Introduction
                And this too.
1.3
               Author
                Author information.

2. Effects (in alphabetical order):
2.01
               Backwards
                2.09
               Minimize
                2.02
               Blur
                2.10
               Mix
                2.03
               Boost
                2.11
               Modulate
                2.04
               Downsample
                2.12
               Shadow
                2.05
               Echo
                2.13
               Shift up
                2.06
               Filter
```

1.2 Disclaimer

I cannot take any responsibility about misspellings or other ↩ flaws that may cause trouble if you rely on the information described in this FAQ file. I've spend hours and hours doing a sound effect generator so all of the ideas in this guide are tested in real-life and notifed to be working as they should be.

I cannot either say that this information about the sound effects is absolutely correct. This is because I've not found any info myself and I have had to improvise. So, you could say that in this FAQ file I've gathered my own ideas and sollutions for problems in the field of sound engineering.

There might be easier sollutions for the effect calculations described below, but again that's not fault. Don't read this FAQ like a bible, but still use it as your information resource.

If you have audio related problems, I'm very willingly to help. Just

contact
me and descibe the problem.

Note that all effects descibed in this FAQ file are used by my program

SinED
. It can be found from aminet from dir mus/edit/SinED.lha.
When reading this file, I'd suggest that you downloaded the editor and
tested the effects with it.

1.3 Introduction

Introduction

Well, this whole thing started a long time ago, when I was in an urgent need of a capable sample-editor. Unfortunately there was not a single editor available that would meet all of my needs. I was so desperate that I even pirated the AudioMaster but only to see that it was worse than most of the others. At that moment I decided to write my own editor that would do everything I want. At least this time I'd get what I want. (Besides my C skills needed a practice :D)

Before that I didn't know anything about processing the audio data, I began my research, but how odd: it seemed that nobody knew anything about the more complex effects. Hell, everyone knows how to flip the data backwards, but when it comes to for example modulation routines, I didn't get any info. That's why every single routine described in this guide is based on my thoughts and ideas that have come to my mind during the programming sessions. I cannot guarantee that these instructions are the best around, or that they were even correct, but they give a correct result and that's enough for me.

That's how this project started. If you own an Amiga with KS2.04+, then I highly recommend you to get my sample editor from aminet. At this moment (16.1.1997) the editor is not uploaded there, but it will be soon and the name will be:

mus/edit/SinED.lha
v1.0 The best sample editor/generatior so far.

With the program you can explore the routines and listen how they hear like. If you can't find the program from the aminet, you can always request it from
me
. I'd be more than happy to send
it to you.

You're possibly wondering why I've not made myself clear with example routines for example with C language. This is because the sources of SinED are already included in it's package. Download it and take a

look at them.

Have fun with this guide. I'd be more that happy to hear from your thoughts about this file, and if you knew new routines to include in this guide, then please
Contact me!
Signed: Jarkko Vatjus-Anttila

1.4 Author information

Author info:

If you have any problems with this file or audio problems that you need help with, then don't hesitate to contact me.

SMail: Jarkko Vatjus-Anttila EMail: quaid@kempele.fi
Linnukkatie 2
90450 Kempele
Finland

WWW: http://www.kempele.fi/~quaid/

Note: My EMail account will expire on the 1st of June 1997. Before that you can contact me, but after it, it might be more difficult. My new accound will open on the 1st of September 1997 but the exact address is still unknown. However, if you really need to contact me after June, then drop a note in some of the comp.sys.amiga.* newsgroups. I'll most probably notice it there.

1.5 Backwards

Backwards

I think this one needs no other introduction that the name. The sample data is simply flipped backwards. The last value is switched with the first one, the second last is switched with the second, and so on until every sample value is switched with its corresponding value.

1.6 Blur

Blur

The blur effect mixes every sample value about 1%-15% from the sample's amplitude. If you are processing 16bit sample data, the

length of the amblitude would be 65536 and the maximum blur value 0.15 * 65535 = 9800. Now, create a random number between 0 and 9800 and then randomly add or substract it from the current sample value. Loop this procedure until you have affected all the sample values, and by result you have a blurred sample. This gives a foggy feeling to the waveform.

1.7 Boost

Boost

Boosting means that the sample peaks are increased and sharpened. One of the methods to create an effect like this, is to use a linear interpolation. That method applies to boost effect more that better because we actually are calculating first degree polynomes. One thing that remains, no matter how you boost is that the first and the last value in the sample are not affected in anyway.

Create a
interpolation
line from the second sample value to the
fourth one. That's right, skip the third one. Then calculate the value for the third samplevalue with the interoplation line. This would now be the value that the third sample value would have if the three values were in a row. However, it's highly unlikely that the tree values were in a row, so now you have to calculate the distance of the third value from the interpolation line. This can be easily done by subtracting the interpolated value from the real value.

Now we have the distance stored somewhere for later calculation. Because we had to increase the sample peaks in the boost operation, we need to increase the distance for the third value from the interpolation line. In this way the peak rises and sharpens. By this point you can add a boost-factor in the game. Use the factor to add a multiply of the distance to the third sample value. For example if the factor was 2 (which is very high) the new value for the third sample value would be:

3rd samplevalue = 3rd samplevalue + factor*distance

Usually the factor is a fraction.

You have to take special care when boosting the sample, that the sample values may never increase or decrease so much that they would change their sign. For example if the 16bit signed sample had a value of 15000 before the boost operation, then the maximum positive boost would be 32767-15000 = 17767 and the maximum negative boost value would be -32768-15000 = -47768. Graphically thinking this means that the sample peaks in the picture of
sample wave
may
never raise over 32767 or lower below 32768.

After calculating the value for this sample value, go to calculate the next one. Jump one value forwards and repeat the calculations. Note that you jump only one, not three values forward. This is because when calculating this effect, all of the values in the sample have to be affected during the boost, but from the bunch of the three values only the middle one is affected. That's why you jump only one value forwards, so that the middle value is now the fifth one. Loop to the end of the sample.

Note too that you need two buffers for this operation. This is because if you calculated values from one buffer to the same, over writing the values, only the first calculation would be accurate, the rest would be something else. This is because the previous calculation would always affect the next one. This may no happen, so two buffers are required.

Figure 1: White noise
Figure 3: Boosted white noise

1.8 Echo

Echo

Echo sounds cool, but is nothing more than
mixing
waveforms
togeter. With echo you need 3 values: delay, decay and repeat. Delay is the time period how long is the difference between echos. You may decide yourself if this value was seconds of sample values or something else. Decay is a value that affects the echo volume. Usually this is percentages. For example if the value was 80% it would mean that every time the echo is repeated, the volume would be only 80% from the previous echo volume. Repeat tells only how many times the echo is repeated.

Example: Delay = 300 samplevalues, decay = 80, repeat = 5

Delay value is this time samplevalues, and it means that first echo is repeated when the sample has been played for 300 values, the second echo would happen after 600 sample values and so on. Take the original waveform to another buffer and decrease it's volume to 80% from the original. Then mix it to the original waveform starting from the samplevalue 300. For the second echo dercease the volume from the previous echo again to 80% and
mix
in to the original
sample from the sample value 600. Repeat these mixing procedures as many times the repeat count says and you most probably have a fine echo. ;)

1.9 Filter

```
                Filter
------

Filter is just the opposise to
               boost
              . Calculate the values with

               interpolation
               line, but instead raising and sharpening the sample
peak, lower and smooth it.  This makes  the sample to go smoother.

I think I don't need to explain this further. Take a look at the

               boost
               operation.

 Figure 1:  White noise
 Figure 2:  Filtered white noise
```

1.10 Flip

```
Flip
----

Flipping is  flipping  the samplewave  upsize down. :) If  you are
processing 16bit sampledata, topmost value in the sample window is
32767. Every sample value (unsigned 0 -> 65535)  must be thrown to
other side of  this value. This can  easily done for  example with
this algorithm:

newsamplevalue = 65535 - samplevalue

or for 8bit sample data:

newsamplevalue = 255 - samplevalue
```

1.11 Maximize

```
Maximize
--------

In maximize routine the main purpose  is to raise the  volume of the
waveform without  trashing the sample.  This is done by  calculating
the smallest value that  every samplevalue  can be multiplied  with,
without  the sample  value changes  it's sign.  If the origial value
was positive, then find the value that  multiplied with the original
value gives a result 32767.  If the sample value was  negative, then
Find a value that multiplied with  the original value gives a result
-32768.  Loop the  whole  sample and take  the *smallest*  number to
```

memory. If this value is 1 then the sample is already at its maximum but if the value is something else (usually a floating point value between 1 and 2) then multiply every value in the sample with this value. Resulting you have a maximized sample.

1.12 Minimize

Minimize

Minimize is the oppoiste to maximize . This time the main idea is to lower the volume without losing any of the sample peaks. This can be done by doing following calculations: If the original samplevalue is positive or zero, then calculate a number that would have to be substracted to get a zero. If the sample value was negative, then calculate the value that would have to added to get a zero. Take the smallest number in memory and then loop the whole sample with adding the value to negative samplevalues and substracting the value from the positive sample values. Resulting you have a waveform with minimiun possible volume.

1.13 Mix

Mix

Mixing is copying two samples together. Let's assume that you have two signed waveforms (they have to be signed!) You might have heard about the interference between two waves. This works exactly the same. Add the samplevalues togeter and divide the result with 2 to get the sample approximately to the same level than the two sample before mixing. This is not enough though. The volume goes too low, so you have to add a factor to both samples to the mixing procedure. If the volume is wanted to get to the same level than the previous waveforms, the factor should be about sqrt(3). With this information the mixing routine get this form:

newsamplevalue = (1st_samplevalue*1.7 + 2nd_samplevalue*1.7) / 2

Basically thinking the mixing is to calculate the average for two or more samples.

1.14 Modulate

Modulate

Modulation is one of the more exotic routines and you can get really weird sounds with this one. The idea is first to create a base waveform. Usually this is a sine or similar. The length of the base

wave is almost obsolete, and I use a wave with length about 2048 bytes. After creating the base wave, get the skip value from somewhere. This value tells you how many samplevalues you have to skip in the base sampledata when you proceed to the next sample value in the original sampledata. Of course, when you reach the end of the base sampledata, loop back to start. Everyone from the original sample values have to be processed.

The modulation is done by multiplying the current samplevalue with the current base sample value and then dividing with the half of the sample amplitude. That depends on the sample accuracy you are using. If you use 8bit data, then this value would be 128 but for 16bit data this value is 32767.

The algorithm used to modulate is:

newsamplevalue = base_samplevalue * samplevalue / 32767

The datas have to be signed in this case too, and the accuracy of the samples has to be the same.

1.15 Shadow

Shadow

I'm not quite sure if this one is obsolete or not. Well, the idea is to reset every other sample value to zero. The sound changes to hear like insects in some cases. Perhaps this one is my weird imagination. :D

1.16 Shift up

Shift up

Shift up and Shift down exists because of some dumb samplers that sample the data in a way that the result is no longer steadily around the base level. With these effects you can adjust the sample to the right position. This does not affect the sound at all, if sample peaks are not lost.

Shift up and
Shift down
can be linked togeter to a routine
that would automagically calculate the best location for the sample waveform. If this kind of effects exists in the program it's usually called "Normalize" or "Normal DC"

1.17 Shift down

Shift down

Exactly the same as the Shift up but this one shifts the data downwards.

1.18 Volume slide

Volume slide

Volume slide is for changing the sample volume smoothly from some point to another. This can be done for example by asking the user for percentages that the sample volume should be in the beginning from the original sample and in the end. Then create a factor that is changed all the time sliding forward on the sample. For example to slide a volume from 200% to 50% the factor would get values from 2 to 0.5. Just calculate the correct factor value for each and every sample value and multiply every value in the sample with it.

Always remember that if the factor is greater than one, the sample can be clipped and distorted.

Figure 4: An example of a small
talk sample that has been volume slided from 300% to 0%. You can see the distorsion in the beginning and the fading in the end.

1.19 Downsample

DownSample

Downsample means that the sample is changed in a way that it sounds lower although it is played with th e same frequency. As you might imagine this is done enlarging the sample buffer. If you have a sample that you would like to hear twice as low without lowering the playing frequency, you would double the sample buffer and copy the sample values from the source sample into every other spot in the target buffer. Then fill the empty spots in the target buffer for example with interpolation lines. Just calculate one for every single space the target buffer contains and fill them. In this way the sample is longer and that's why it sounds lower although you play it with the normal frequency.

If you don't want to halve the sound frequency, you can always allocate for example a buffer that is one third bigger that the source one, and then
interpolate
every third value. This would
enlarge the buffer size one third larger and then the frequency would

drop one third. Simple as that.

1.20 Upsample

Upsample

Upsample works just like downsample, except that this time the buffer size is reduced instead of enlarging. This is more risky than downsampling because when the buffer size is reduced, some of the sample information must be thrown away. Too much upsampling and the sample is distorted.

Halve the buffer size, and the sample hears like twice as high. Reduce one thirds and the frequency raises one thirds, etc.

1.21 Box wave

Box

This one is simple. Just fill the half of the buffer with the value that is maximum positive value for your current sample accuracy. For 8bit samples this would be 127 and for 16bit samples 32767. Then fill the latter half from the buffer with the minimum value. For 8bit samples this would be -128 and for 16bit samples -32768.

Figure 6: Example of a box-wave.

1.22 Saw wave

Saw

This one is as easy as the box wave if you consider the sample buffer to be unsigned. The scale would then be for 8bit samples from 0 to 255 and for 16bit sample from 0 to 65535. Then just create an interpolation line of which starting point is (0,0) and ending point is (sampleend,65535). Sampleend is the sample length in sample values. After creating the interpolation line just calculate with it the values for every space in the sample buffer.

Figure 7: Example of a saw-wave.

1.23 Sine wave

```
Sine
----

The period of the sine is 2*PI. Divide this value with the length of
your buffer in sample values. This gives you the delta angle that you
must jump forwards when calculating the wave. Begin from 0 and end to
2*PI.

Sample code:

deltavalue = 2*PI/samplelength;
for (loop=0; loop<2*PI; loop++
  buffer[loop] = sin(loop*deltavalue);
```

Figure 8: Example of a sine-wave.

1.24 Triangular wave

```
                Triangular
----------

Triangular wave is similar to saw,  but this time the interpolation
is divided into 3 sections. You have to calculate three lines, with
following coordinates:

   Start        End
   -----        ---
Line 1:  (0,0)        (samplelength/4,32767)
Line 2:  (samplelength/4,32767)   (3*samplelength/4,-32768)
Line 3:  (3*samplelength/4,-32768)  (0,0)

Then fill  the first part with the first
               interpolation
               line, the
second part  with the second line and the  third part with the third
line. As you can see,  the first line raises from  the base position
to the  maximum, the second  lowers from the  maximum to the minimum
and the third one raises from the minimum to the base position. When
these three lines are connected, there appears a triangular wave.

Triangular wave is similar to
               sine
               wave.
```

Figure 9: Example of a triangular-wave.

1.25 White noise

```
White noise
-----------

White noise is just random numbers with the complete scale of your
current sample accuracy. Just write a decent randomnumber generator
and you'll create a white noise just fine.

 Figure 1:  White noise
```

1.26 How to draw a sample

```
                 How to draw a sample
-------------------


               Pictures
               Some waveform pictures here!

This can be done easily if the sampledata is in signed form.
Imagine a box of this kind:

32767 +-------------------------------------------------------------+
|               |
|               |
|               |
|               |
|               |
    0 X*************************************************************|
|               |
|               |
|               |
|               |
|               |
-32768 +------------------------------------------------------------+

Where the *** line is the midline. Each  horizonal line in that box
represent  its own  sample  value  and the  *** line  is the  zero.
Consider the  box as coordinate axles where the *** line is x-axle,
the leftmost vertical line is the y-axle  and the X is the  origin.
The length of  the x-axle is  the same as the length  of the sample
and the length of the Y  axle is  the same as  the aplitude  of the
sample. For  8bit samples  this would be  256 (-127 -> 128) and for
16bit samples 65536  (-32767 -> 32678).  Of course  these kinds  of
sample boxes cannot  be drawn, so the you have to  scale one to fit
on your screen or window.

Ok, the drawing then is simple. Just take the value from the sample
and  move  your pen to  that spot on the  sample box. If  the first
value is (8bit  sample) 69 then  the coordinates are (0,69). If you
imagined  the sample  box to  be the conjunction of the  coordinate
axles the only think you need to do, is to scale the coordinatepair
(0,69) to your screen coordinates.  Chech all of  the sample values
```

```
  and connect the plots  on the screen with lines, and there you have
  a sample wave drawn on your screen.

   Figure 5:  Simple samplewave example.

  Drawing speedups:
  -----------------
  - Calculate first how may sample  values you  have to skip  before
    drawing. I  mean that  if you have a screen width 640 pixels and
    the sample length is 102400 bytes. In this case only every 160th
    value would have to checked and displayed in the screen. In this
    way you only need to check 640  values in total  from the sample
    instead of the 102400. The speedup is remarkable.
  - You can use  just pixels for  drawing, too.  Lines are  not slow
    when displaying a sample, but pixels are always faster.

  Special notes:
  --------------
  - Let's assume  that the  screen width is  640 pixels. If  you are
    drawing a sample  which length is smaller than 640, you have  to
    skip pixels on the screen instead the  vaules in the sample,  to
    fit the  sample in the 640 pixels.  This time you cannot connect
    the values with lines  because it wouldn't be  drawn right then.
    If the two values were just connected with a line, it would tell
    the  user  that the  sample  actually  wasn't so  short  and  it
    contained values  between those connected samplevalues,  even it
    didn't contain them.  The idea is that when sample value occurs,
    it lasts until the next value comes up.  That's why  you have to
    draw  a  horizonal  line  until  the  next  samplevalue and then
    connect the end of the horizonal line to the  next  sample value
    with a  vertical  line,  this  makes the  samplewave  to like  a
    stairs in most cases, but at least it's the truth.
```

1.27 SinED v1.0

```
  SinED v1.0
  ----------

  This is the .readme file of the SinED v1.0

------------------------------------------------------------------------------


Description:  V1.0 The most powerful sample editor/generator ever made!
Author:   Jarkko Vatjus-Anttila <quaid@kempele.fi>
Uploader: Jarkko Vatjus-Anttila <quaid@kempele.fi>
Version:  1.0
Dir:    mus/edit

 Welcome to SinED v1.0, the most powerful sample editor/generator so far. :)
 ---------------------------------------------------------------------------

 (c) 1996-7 Jarkko Vatjus-Anttila <quaid@kempele.fi>
```

This program is without a doubt the most powerful sample editor generator so far. The reason this is written, is because I needed a decent sample edtior, but the ones available were more or less pathetic. It doesn't have to be that the only good sample editors are available only for those who have a DSP installed in thier machines. SinED will fix that problem.

Just take a look at the list about the features that SinED provides to you. You will not believe your eyes.

Hype? Perhaps, but still take a look. :)

Advantages:
---

* It's free.

* All calculations are made in 16 bit accuracy. This ensures that all calculated effects and operated samples have the best possible quality. 8bit samples are just an option.

* Currently SinED is capable to create Sines, box, triangular and saw -waves, not forgetting the white noise (and clear :).

* This version includes following effects: Backwards, Flip, Shift up, Shift down, Shadow, Blur, Boost, Filter, Maximize, Minimize, Mix, Modulate, Tunnel echo, Repeat echo, Convert and Translate.

* Some effects are combined in a way that you can create cool sound sweeps. For example you can combine the filter and the boost options to create for example an opening filter. I remind that all effects are smooth due the 16bit accuracy.

* Samples can have unlimited length. Only your amount of memory will slow down you work. It works with virtual memory too, although only VMM is tested.

* Samples can be displayed in four different drawing styles. Lines, Pixels, solid and inverse.

* Most common sample format are now supported, like IFF-8SVX, IFF-16SV, WAV and AIFF. SinED includes routines for saving and loading mono and stereo modes. Not surround.

* Includes two stereo channels. Effects can be calculated for both channels or just for other. Channel lengths are independent, so both channels can contain for example their own sample.

* Multilevel undo option. Only your memory limits the actions.

* Of course the samples can be played, with user definable settings.

* Samples can be edited by hands.

* It's far more better than for example AudioMaster, the most pathetic sample-editor ever made. Take a look at this one!!!!

Download this now! It's worth it.

1.28 Audio pictures:

```
                Audio pictures:
---------------

Here are all of the pictures gathered from this guide onto this one
page. All pictures are created with
               SinED v1.0
               and grabbed
with QuickGrab.

 Figure 1:  White noise
 Figure 2:
               Filtered
               white noise
 Figure 3:
               Boosted
               white noise
 Figure 4:  An example of a small talk sample that has been

               volume slided
              from 300% to 0%. You can see the
    distorsion in the beginning.
 Figure 5:  Simple samplewave example.
 Figure 6:  Example of a box-wave.
 Figure 7:  Example of a saw-wave.
 Figure 8:  Example of a sine-wave.
 Figure 9:  Example of a triangular-wave.
```

1.29 Linear interpolation

```
Linear interpolation
--------------------

In this guide  I refer a lot  to a simple mathematical solving method
called linear interpolation. Let's see how it works.

The main idea is to calculate  a formula for a line that would  match
most of  the values  we are currently  processing. In  most cases the
more complex formula changes  to a first degree approximation formula
that can easily be used to assist in work. And more importantly, it's
far more  faster than  calculating  with the  original values. Take a
special care noting that this  really is just  an approximation.  You
will not get accurate results.

As you might imagine,  to create a first  degree formula, you need to
know the start  and the end  coordinate pairs  in order to  calculate
```

the slope and to generate the formula.

Let's assume that the start point is (x0,y0) and the end point is (x1,y1). Then the slope would be:

$$k = \frac{y1 - y0}{x1 - x0}$$

The first degree formula can be written like this, where the plain Y and X are variables:

$$Y - y0 = k*(X - x0)$$

Adding the slope to the function and moving the y0 to right side we get:

$$Y = \frac{y1 - y0}{x1 - x0}*X - \frac{y1 - y0}{x1 - x0}*x0 + y0$$

Everyone can optimize that to suit oneself's needs.

Now when passing values for X, which usually means the sample buffer spaces, you can calculate the value for Y which now represents the sample value. This is the function that can be user for many simple operations.