MDFourier

Artemio Urbina

June 13, 2019

Abstract

MDFourier is an *open source* software tool set created to compare *audio signatures* and generate a series of graphs that show the differences between them.

It consists on a *custom binary* that runs on a *target platform* and an *analysis tool*. The *custom binary* produces a battery of sound test signals which are then recorded as an audio file. The *analysis tool* takes two such audio recordings and compares them, showing how different they are.

Te results can be used in a variety of ways: to identify how *audio signatures* vary between *systems*, to detect if the *audio signals* are modified by *audio equipment*, detect if *modifications* resulted in audible changes, to help tune *emulators*, *FPGA* implementations or *mods*, etc.

The present documents serves as an introduction, a manual and a description of the methods used and how they work.

Contents

Chapter 1

MDFourier Objective

To provide a free and open source[1] software based analysis framework for comparing audio signatures from a targeted video game system and its variants[2]. The resulting comparison can help tune such system[3] to better match the desired reference profile.

This is of course not limited to one specific system, and the software can be used to compare any other platform with new configuration files[4].

The intention is not to disparage any particular system; but to help understand and improve them whenever possible by identifying their differences.

A secondary but not less important goal is to create a community driven catalog of audio signatures from these systems, in order to help preservation efforts achieve their objectives.

1.1 What is it for?

MDFourier can be used to identify the differences between two audio signatures. For instance a *Sega Genesis Model 1 VA3* and a *Sega Genesis Model 2 VA 1.8*[5] can be compared in order to verify how different they are across the audible spectrum.

It can also be used to compare a vintage version of the console against any other variation, like an emulator or an *FPGA* implementation.

[1] Licensed under GNU GPL see appendix A

[2] Vintage retail variations, hardware modifications, clones and *Field-programmable gate array (FPGA)* implementations

[3] The term *system* will be used to cover vintage retail hardware, emulators, *FPGA* implementations and any other possible variant that executes binaries for the target hardware

[4] At the moment a profile for *Mega Drive/Genesis* is functional and implemented with more to follow.

[5] These two models are typically listed as having notorious audio differences [2]

Another possible application is to determine if there were changes in the audio spectrum after modifications to a console, like recapping or changing the audio circuitry.

These results can help determine if the signals are indeed different, and how they differ across the human hearing frequency spectrum. Such information can then be used for different means, such as recreating specific audio signatures, tuning to a different taste, etc; based on an objective, repeatable and measurable data set provided by the framework.

It can also be used to evaluate if equipment - such as switchers and upscalers with audio passthrough - have any effect on the signal, by comparing a recording with and without the device connected in the AV[6] chain.

Although I believe any present and future implementation of a gaming platform should offer a configuration based on vintage retail hardware, that doesn't mean there isn't room for improvement upon those configurations. Reducing noise while keeping the reference sound signature is one such case.

[6]Audio/Video

1.2 Disclaimer

MDFourier is a work in progress, just as the current document. It still has a few rough edges, and although I tried to adhere to the best practices known to me, my expertise in *digital signal processing* was almost non-existent before this project.

If you have suggestions, please contact me. Any corrections and improvements are encouraged and welcome. Contact information is available in appendix K.

This project was born from my curiosity to compare the audio signatures of different revisions of the *Mega Drive/Genesis*, and verify them with a tool assisted analysis. I was also curious about *FPGA* and software implementations, and how similar they were to vintage console audio signatures.

Chapter 2

How *MDFourier* works

The first thing to keep in mind is what *MDFourier* does. It takes two signals, the first one is the *Reference* file and the second one is the *Comparison* file.

The *Reference* file is used as a control. This means that its characteristics are considered the true values to be expected and against which the *Comparison* file will be evaluated. In consequence, all results are relative between the signals.

These files are audio recordings from the desired hardware, preferably captured with a flat frequency[1] audio capture card[2] and generated by a *custom binary*. This *custom binary* is targeted for the particular hardware capabilities and frequency range. Whenever possible, this binary will be part of the *240p Test Suite*[3].

The analysis software is itself *command line* based, in order to be multi-platform and offer it on every operating system that has an *C compiler*[4]. However a *GUI*[5] front end for *Microsoft Windows* is provided for simplicity and accessibility. Not all options from the command line tool are present via the *GUI*, but the most relevant ones are readily available. Full Source code can be downloaded from *github* [6].

[1] Flat frequency response refers to the capability of capturing the whole audible spectrum with little or no variation, regardless of frequency. A non flat frequency card can be used to gather relative information, but the captures won't match the ongoing catalogue in order to make further comparisons.

[2] See *Audio Cards* in appendix H

[3] A homebrew software suite for video game consoles developed to help in the evaluation of TVs, upscalers, upscan converters, line doublers and video processing in general.[4]

[4] *ANSI C99 compiler*

[5] *Graphical User Interface*

[6] See download link [3]

2.1 File alignment

MDFourier takes both files and auto detects the starting and ending point of the recording. These are identified by a series of *8820hz* pulses in the current *Mega Drive/Genesis* implementation. From these, a *frame rate*[7] is calculated in order to trim each file[8] into the segments that are defined in the *configuration file*[9] for further comparison.

Most importantly, it guarantees that the *Reference* and *Comparison* files are logically aligned, and that each note - or segment - is compared to its corresponding one, with no overlap and without any audio editing or trimming skills required from the user. Current pulse detection accuracy is around $^1/_4$ of a millisecond.

After alignment is accomplished, the software reports the starting and end points of both signals, in seconds and bytes.

2.2 The heart of the process

In order to compare the signals, audio levels are checked and a relative *normalization*[10] takes place, based on the maximum amplitude of the *Reference* file. A local maximum search is done in the same frame on the *Comparison* signal, and that amplitude is then used to normalize both files. This is done in the *frequency domain*, in order to reduce amplitude imprecision when comparing recordings with different *frame rates*. The software does have the option to do this in the time domain, but quantization and amplitude imprecision is to be expected in such case.

Then a *Discrete Fourier Transform*[11] is used in order to analyze the frequency content of the signal, as well as the amplitudes from each of the corresponding fundamental frequencies that compose the audio signal.

The software uses the *FFTW*[12] library in order to accomplish this, and then proceeds to sort out the frequencies of each block by amplitude. It can be configured to compare a range of these frequencies, but by default it compares 2000 of them for each element defined in the *mfn* configuration file[13].

I've found such comparison to be more than enough, and the *minimum sig-*

[7]The *frame rate* is the ratio at which the console sends video frames to a display. For more details see appendix G

[8]Since audio cards have their own *sample rate* clocks [8] [9] [10], and some implementations have different *frame rates*, frames are used as the base unit instead of time codes

[9]This file specifies the operating parameters for each hardware configuration. It is described in appendix B

[10]This process is detailed in appendix F

[11]Discrete Fourier Transform (DFT) is the variant of *Fourier Transform* applied to discrete values, such as the ones we have in the audio file [1]

[12]*The Fastest Fourier Transform in the West* [5]

[13]Described in appendix B)

nificant volume[14] even limits these 2000 to a lower number, based on significant amplitude[15]. In case more frequencies are needed, this can be changed via a command line parameter.

After comparing these frequencies between both files, matches are made and the differences in volume are plotted to a graph. Please read chapter 4 to help you understand the various plots created by this program.

Sometimes it is helpful to listen to the results of these limiting filters, in order to evaluate if - for instance - 2000 frequencies are either enough or too little for the current application. For this and other purposes I made an extra tool named *MDWave*[16], which creates segmented audio files as processed after the *Fourier Transform*, even including the effects of *window filters*[17].

If you are interested in learning what the *Fourier Transform* does and how it works it's magic, there are several resources online to help you out. Here are a few:

- But what is the Fourier Transform? A visual introduction.
  `https://www.youtube.com/watch?v=spUNpyF58BY`
- An Interactive Introduction to Fourier Transforms.
  `http://www.jezzamon.com/fourier/`
- The Uncertainty Principle and Waves.
  `https://www.youtube.com/watch?v=VwGyqJMPmvE`

2.3 Minimal significant volume

Although the whole frequency spectrum can be compared, there is little practical use in doing so - due to execution time and extra noise from low amplitude harmonics. As a result, rule of thumb defaults are set in order to minimize these issues.

One of these values is a *minimum volume* at which to stop comparing the fundamental frequencies that are found after decomposing the signal with the *Fourier Transform*. This is the *minimal significant volume* and it is the cutoff at which comparisons made by the software are stopped.

This volume is the *amplitude*, and it is measured in *dBFS*[18]. In the *dBFS* scale, the value of *0* is assigned to the maximum possible digital level and it

[14]See section 2.3

[15]The terms *volume* and *amplitude* are used interchangeably during the document, however they are technically not the same thing. *Volume* refers to the perception of *loudness*, and *amplitude* is the *quantifiable* quality of the signal related to its power. As a result, the software can only use amplitude for its calculations.

[16]Described in appendix C

[17]Window functions are described in section 3.1.1)

[18]*Decibels relative to full scale*

can go down to $-\infty$.[19]

Currently *MDFourier* can recognize three scenarios to define the *minimum significant volume* to compare the signals, and the three are derived from the first *silence block* in the file.

The first scenario is the grid power frequency noise, which currently searches for *60/50hz* noise[20].

The second one is *refresh rate noise*, again derived from the frame rate, in *NTSC* it is around *15697-15698hz*.

If neither is found, which would be surprising for a file generated by recording from a vintage console via analogue means, the frequency with the highest volume within the silence block is used.

Finally, in case none of the previous scenarios is met - or if those values are lower than *-60dbfs* - a default level of *-60dBFS*[21] is used.

2.4 Workflow

The first step is loading the custom binary to your console, this varies from a *flash cart*, burning a *CD-ROM* or using a *custom loader*.

The next step is setting up your *audio capture card* and computer, in order to record a *44 or 48khz 16 bit stereo* *WAV*[22] file.

Once the *capture card* is ready and the cables are hooked up from the console to the *capture card*, start recording on the computer and execute the *MDFourier* test from the console.

Wait for the console to show a message indicating you can stop recording, this typically takes less than 1 minute. After you have at least two files: a *reference* - which can be one of the provided files - and a *comparison* file, you are ready to go.

[19]The minimum volume when using *16 bit* samples is -96dBFS.
[20]*PAL* needs to be tested yet, since I don't have console for that video system
[21]This can be changed via *command line* options if needed.
[22]*Waveform Audio File Format, also referred as WAVE*

Chapter 3

How to use the Front End

The current version of the front end allows access to the main options of *MD-Fourier*. It is a *Windows* executable and all corresponding files must be placed in the same folder. *Uncompressing* the package to a folder should have all that is necessary to run the program.

After executing *MDFourierGUI.exe* you should be presented with the following interface:

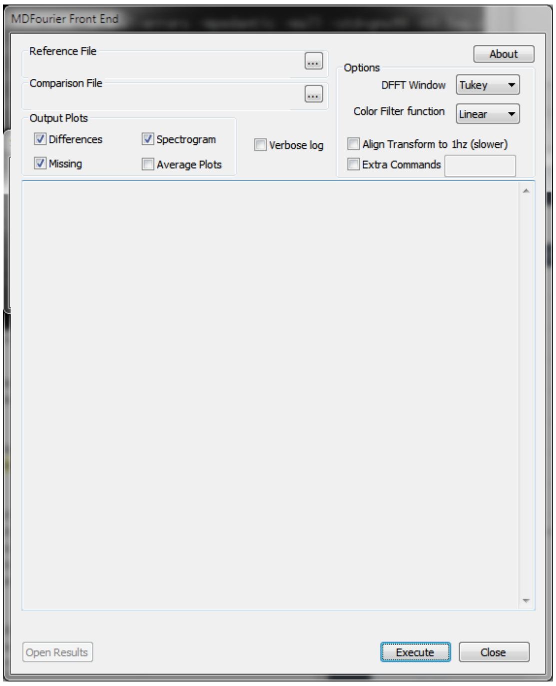

Figure 3.1: MDFourier Windows Front End

In order to generate the output plots, two files must be selected to compare them. One as a *Reference* and the other as the *Comparison* file, as detailed in section 2.

The following sequence of steps indicates the typical work flow within the GUI:

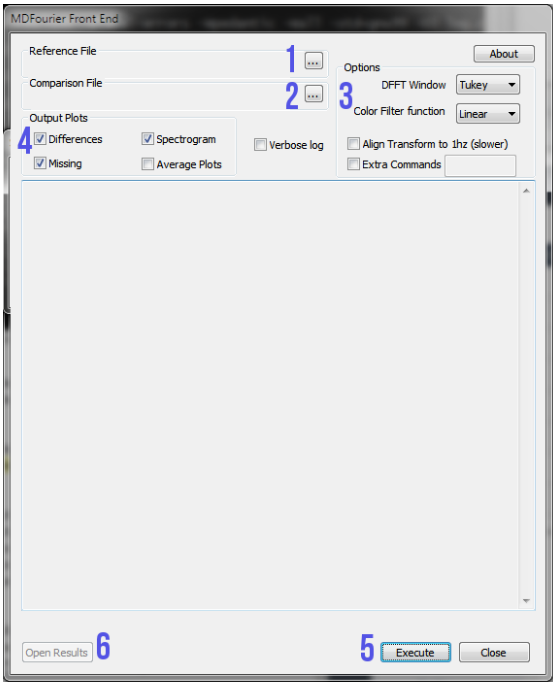

Figure 3.2: Typical sequence of steps

1. Select a *Reference* file
2. Select a *Comparison* file
3. Change the default *options* if needed
4. Select the desired outpit *plots*
5. Execute *MDFourier*
6. When execution ends, open the *results folder*

The *Front End* will display the output text from the command line tool, including any errors or progress as it becomes available.

Keep in mind that the *Open results* button will only be enabled after a successful comparison between files has been finished, and it won't open a second instance of the window if you have one already open.

The default options will generate plots that work on most situations. However, in some cases fine tuning the results could be desired in order to highlight specific aspects. These options will be described in the following sections.

3.1 Front End Options

The currently available options in the Front End are:

3.1.1 Window Functions

In order to reduce *spectral leakage* when applying the *DFT*[1] a *filtering window* is applied to each element to be compared between both signals. Since we are generating the signal ourselves from the *custom binary* for each *hardware platform*, the signal can be analyzed as periodic, and has a natural attack and decay rate if possible.

By default we use a custom *Tukey window*[2] with very steep slopes. *MDFourier* does offer alternate windows as options for further analysis.

3.1.2 Color Filter Functions

Each dot in the *Differences* graph uses the *X axis* for the frequency range and the *Y axis* for the amplitude difference between the *Comparison* and *Reference* signals.[3]

Color intensity of each dot is used to represent the amplitude for that frequency in the *Reference* signal. In other words, how relevant it was to create the original signal in that note. Please refer to chapter 4 in order to see examples of their use.

A color scale is presented in each graph, with the color graduation and the corresponding volume level.

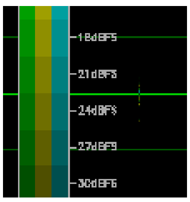

Figure 3.3: Detail of color scale in plot

[1] Discrete Fast Fourier Transform

[2] All details regarding the windows used, their formulas and graphs are in appendix D

[3] See section 3.1.4 for output file details

The options are useful to *highlight* or *attenuate* these differences by applying the range to one of the following functions.

They are sorted in descending order. The topmost option will highlight all differences; and the bottom one will attenuate most of them, and show just the ones with highest amplitudes in the *Reference* signal.

All filters, their graphs and effects are listed in appendix E.

3.1.3 Align Transform to 1hz

When designing the *audio signal* for use during analysis, one consideration is how to balance gathering more information versus the duration of each recording.

For practical reasons, it is desirable to have a short test that will be recorded for analysis. This reduces the time it takes to digitize audio from several systems, the storage used, analysis time by the software, and makes distributing such files easier.

In contrast, when applying the *Fourier transform* a compromise is made between frequency detail and time accuracy, very similar to the Heisenberg's uncertainty principle. If we compare a longer signal for each element, we end up with more frequency information. In our case, we don't care much about time accuracy but we do care about the length of the test. Nobody wants to record a 5 or 10 minute signal for each test to be made.

The compromise made is to use *sub second signals* for each element to be compared. Since the time and sample rate determine how the *DFFT*[4] *frequency bins* are spaced after analysis - and how much information we end up analyzing - the end result is a lower plot resolution.

Zero padding the input signal for Fourier analysis is a controversial subject[5], but for the present application no adverse effects have been found. This might be the case since we control how the source signal is generated, with a predetermined *attack* and *decay*. However, it is disabled by default so the results presented can be free from questioning regarding the effects, and available for cases where more precise frequency information is needed with no spectral leakage to adjacent *frequency bins*.

3.1.4 Output Plots

Several plots will be generated as a result of the analysis. Please read section 4 for examples and a guide to interpret their meaning.

[4] *Discrete Fast Fourier Transform*

[5] Zero-padding a signal does not reveal more information about the spectrum. See [12] [13]

Different Amplitudes

Enabling this option created the most relevant output plots from *MDFouriuer*. These plots contain the amplitude differences across the hearing spectrum using the *Reference* file as control.

If the files are identical, the plot will be a perfect line across the *0 dBFS* line.

Missing Frequencies

Plots the frequencies available in the *Reference* file but not found in the *Comparison* file within the significant volume range. This is in effect a *Spectrogram* limited to the frequencies that were expected but are not present.

Spectrograms

Plots all the frequencies available in each file. Two sets of spectrograms are generated, one for the *Reference* file and one for the *Comparison* file.

Average Plot

This option traces an *average line* on top of the *Difference* plots, making it easier to follow the trend when the output has severely scattered data. It is *off by default*.

The curve is created by averaging time segments from the frequency sorted data in *Difference* plots. A *Simple Moving Average* is then calculated to smooth out the results.

The curve is weighted according to the *Color Filter Functions* described in section 3.1.2, by a repeating each data point by the amount mapped in the *0-1* interval described by the function.

As a result, the average will follow the relative amplitudes from the *Reference* signal proportional to the selected *filter function*.

If there is need for a plot without weighting, please disable the *Color Filtering function*.

3.1.5 Verbose Log

A log is always created by default when using the *Font End*, however this option enables a verbose version with the whole frequency analysis and many other details dumped to the file.

Useful for reporting errors or unexpected behavior. (*Please send the audio files if possible as well!*[6])

3.1.6 Extra Command

This checkbox enables the text field to send any extra commands that are not available via the GUI to *MDFourier*.

Sending *-h* in this field will enlist all the currently supported options.

[6] Contact details are available in appendix K

Chapter 4

How to interpret the plots

The main output of the program is a set of different graphics that vary in quantity based on the definitions made in the *mfn* file detailed in appendix B.

In its current form for the *Mega Drive/Genesis*, there are three *active blocks*: *FM*, *PSG* and *Noise*. These will result in a plot of each type, and a general plot, being generated as output.

The files are saved under the folder *MDFourier* and a sub-folder named after the input *WAV* file names. They are stored in *PNG*[1] format, currently *1600x800* plots are used, although this can be dynamic.

For the current document *800x400* plots were used in order to fit within a *PDF* or *HTML* presentation.

4.1 Output Files

There are common features here that we'll describe. The output plots that are created by the software are listed and described in section 3.1.4.

- **Different Amplitude:** Plots the amplitude difference for the frequencies common to both files
- **Missing Frequencies:** Plots the frequencies available in the *Reference* file but not found in the *Comparison* file within the significant volume range.
- **Spectrogram:** Plots all the frequencies available in each file. Two sets of spectrograms are generated, one for the *Reference* file and one for the *Comparison* file.

[1] *PNG* support via *libpng* [14]

There will be several plots of each type in the output folder. One for each type, and one that summarizes all types in a single plot.

When enabling the *Average Plots* option an extra set of plot files with each average will be generated, as described in section 3.1.4.

In our current *Mega Drive/Genesis* scenario, we'll get four plots of each type: *FM*, *PSG*, *Noise* and a general one, named *ALL*.

We'll follow a series of results from different input files to *MDFourier*, starting with cases that have either none or a few differences, and build on top of each one so you can familiarize with what to expect as output.

4.2 Scenario 1: Comparing the same file against itself

The first scenario we'll cover is the basic one, the same file against itself. Let's keep in mind that *MDFourier* is designed to show the relative differences between two audio files.

So, what is the expected result of comparing a file to itself? No differences at all, an empty plot file as shown below.

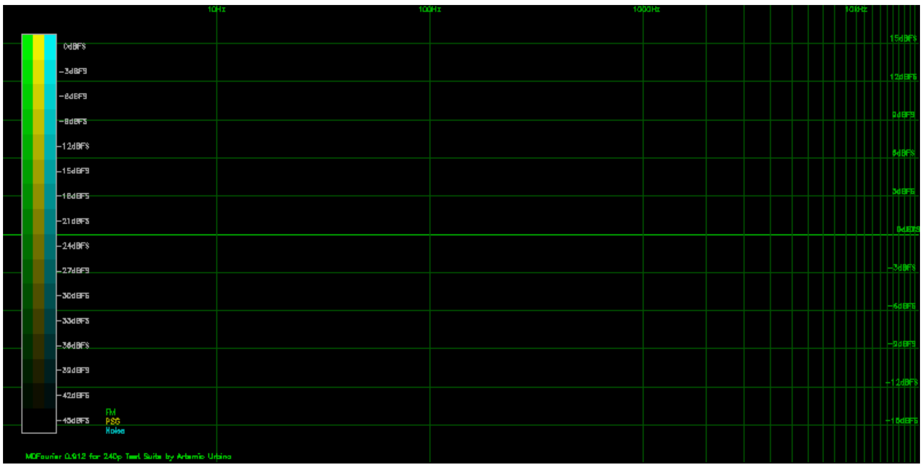

Figure 4.1: Different Amplitudes result file when comparing the same file against itself

Of course all of the *Differences* and *Missing* plots will only have the grid and reference bars, with no plotted information since both input files are identical.

However there will be two sets of *Spectrograms*, one for the *Reference* file and one for the *Comparison* file, with one plot for each type plus the general one.

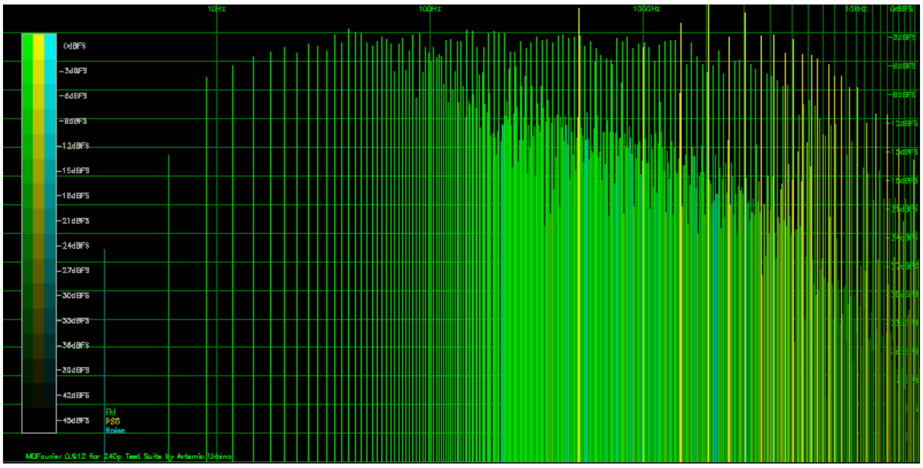

Figure 4.2: The Spectrogram for a Genesis 1 VA3 via hedphone out

The Amplitude, or volume, of each of the fundamental *sine waves* that compose the original signal is represented by vertical lines that reach from the bottom to the point that corresponds to the amplitude in *dBFS* [7]. The line is also colored to represent that amplitude with the scale on the left showing the equivalence.

Three colors as defined from the *mfn file*[2] are used to plot the graph, with each one of them plotting the frequencies from each corresponding block from the *WAV* file.

The top of the plot corresponds to the maximum possible amplitude, which is *0 dBFS*. the bottom of the plot corresponds to the *minimum significant volume*, as described in section 2.3.

Both sets of spectrograms in this case are identical, as expected.

4.3 Scenario 2: Comparing two different recordings from the same console

This is another control case, what should we expect to see if we record two consecutive audio files from the same exact game console using the same sound card?

[2]The configuration file is described in appendix B

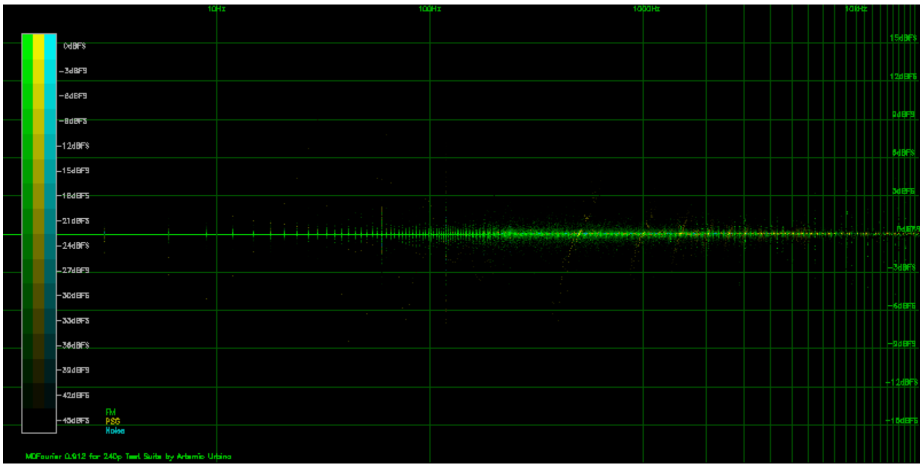

Figure 4.3: The same console, two different recordings

As you can see, we have basically a flat line around zero. This means that there were no meaningful differences found.

But wait, there *are* differences. Why is that? Due to many reasons: analogue recordings are not always the same for one. Then we have variations from the analogue part of console itself, and probably from the internal states and clocks from the digital side. It can also be noise generated by differences in frequency bins when performing the *DFT* after calculating the frame rates - we have that $1/4$ error after all.

We now know that there will be certain *fuzziness*, or variation, around each plot due to this subtle recording and performance nuances. It is a normal situation that is to be expected, and a baseline for future results.

4.4 Scenario 3: Comparing against a modified file

For demonstration purposes, the same *Reference* file was modified to add a *500hz 6 dBFS* parametric equalization across all the analyzed signal. This is an artificially modified and controlled scenario in order to demonstrate what the plots mean.

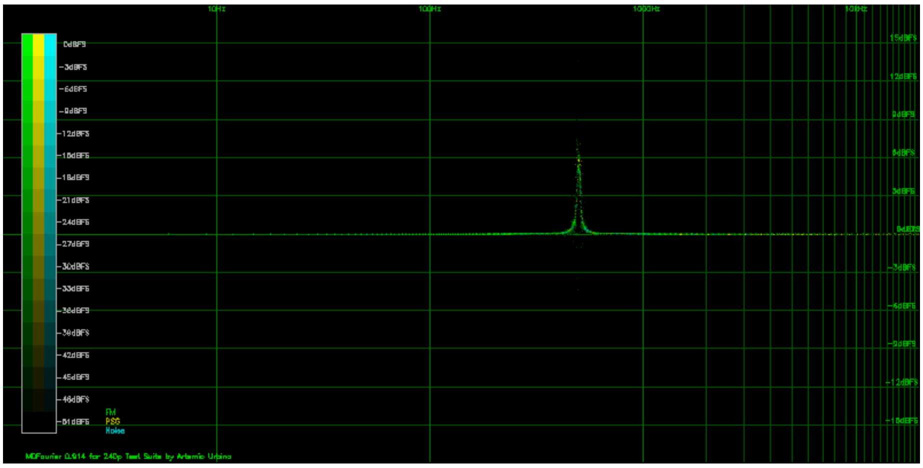

Figure 4.4: Compared against itself modified with a *500hz 6 db* equalization

As expected all three blocks (*FM*, *PSG* and *Noise*) were affected and show a spike, exactly *6 dBFS* tall and centered around *500hz*.

It is interesting to note both spectrograms, since the *500hz* spike is also shown there.

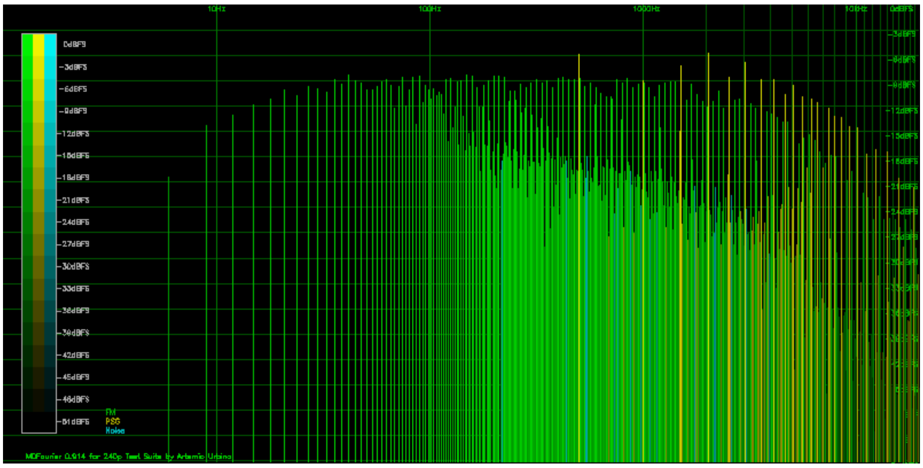

Figure 4.5: *Reference* File

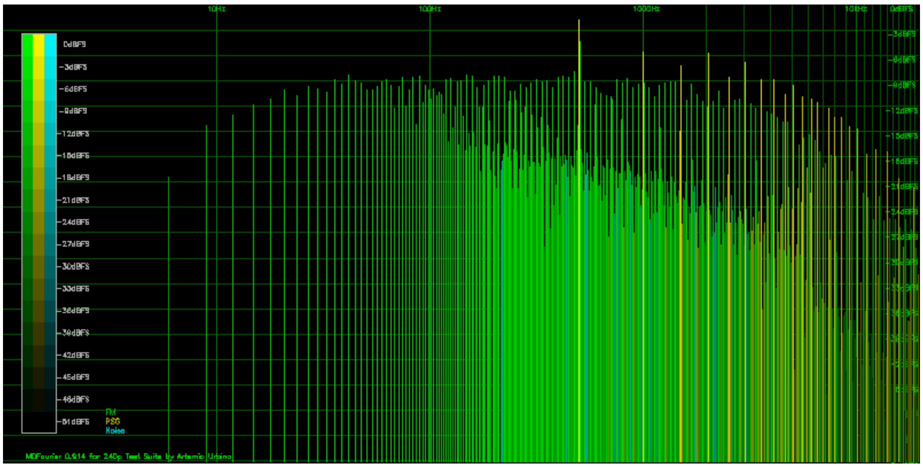

Figure 4.6: Reference file modified with *500hz* peak

And the *Missing Frequencies* plots are basically empty, since no relevant frequencies are missing from the *Comparison* file.

4.5 Scenario 4: Comparing against digital low pass and high pass filters

We'll use the same *Reference* file, and compare it to a file with a several filters:

- A *low pass filter* to the *FM* section of the file
- A steeper *low pass filter* at a different cutoff frequency to the *PSG* section
- A *high pass filter* to the Noise section at a different frequency

This is the general plot with the three sections:

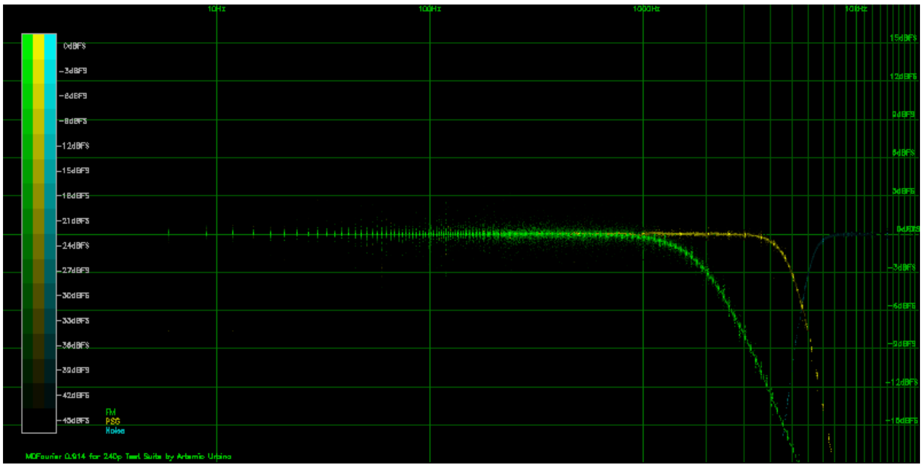

Figure 4.7: *FM*, *PSG* and *Noise* with low pass, low pass and high pass filters

We can now see that the higher frequencies above *1khz* in the FM plot steeply go to $-\infty$ *dBFS*, so the first low pass filter is there.

The second low pass filter for *PSG* is at *3khz*, and is steeper.

But we can barely see what is going on with the *Noise* part of the plot. We can see that there is some black dots on top of the *0dBFS* line.

In order to better see what is going on, we'll change the *color filter function* to $\sqrt{dbFS}$ so we can have higher contrast.[3]

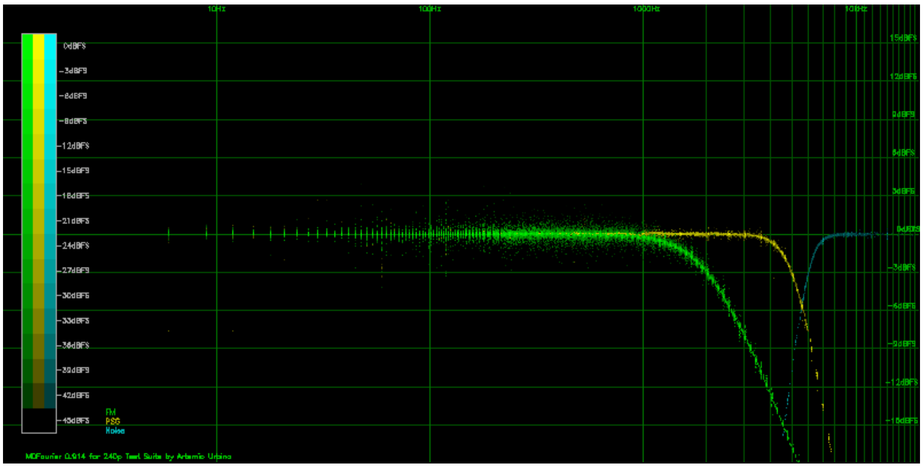

Figure 4.8: Using the $\sqrt{dbFS}$ color filter function

With the higher contrast, we can now make out the curve that raises from $-\infty$ *dBFS* to *0 dBFS*, and it aligns with *8khz*.

[3]Described in section 3.1.2)

We can still do a little bit better, by using the *Average Plot* option.[4]

Here is the resulting plot for only the *Noise* section of the signal with average enabled:

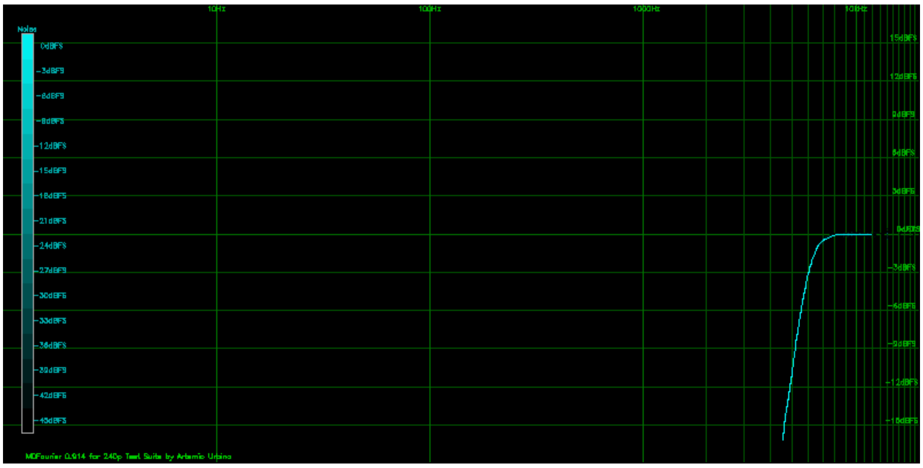

Figure 4.9: *Noise* plot with Average

There are some other interesting plots that result from this experiment. For example, the *Missing* plots now show all the frequencies the *low/high pass* filters cut off.

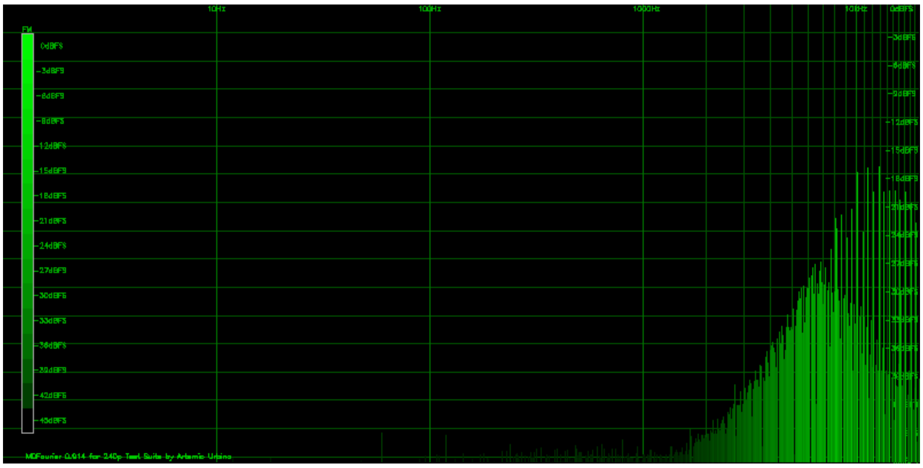

Figure 4.10: Missing frequencies in *FM* cutoff by low pass filter

As show in figure 4.10, there is a curve in the spectrogram and only frequencies above *1khz* show up, slowly rising in amplitude.

[4]An average and then a moving average are applied to the plot, see section 3 for details

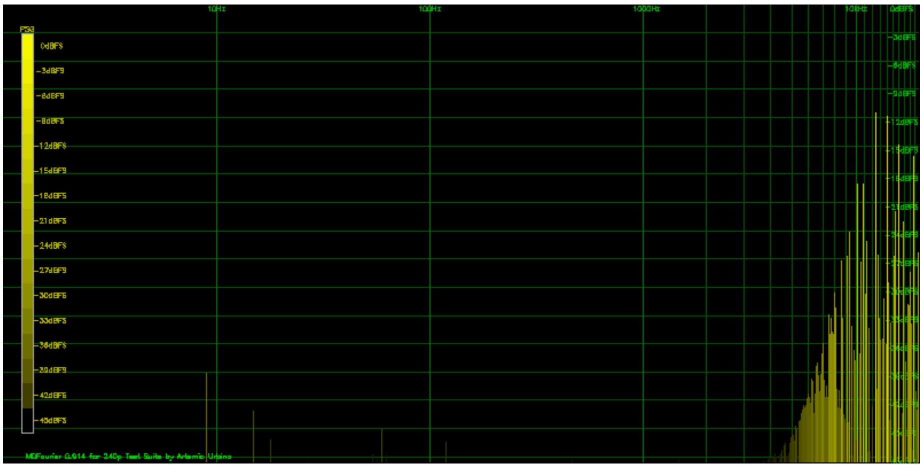

Figure 4.11: Missing frequencies in *PSG* cutoff by low pass filter

The same behavior can be observed in the *PSG spectrogram*, but with a different curve that starts at *4khz* in figure 4.11.

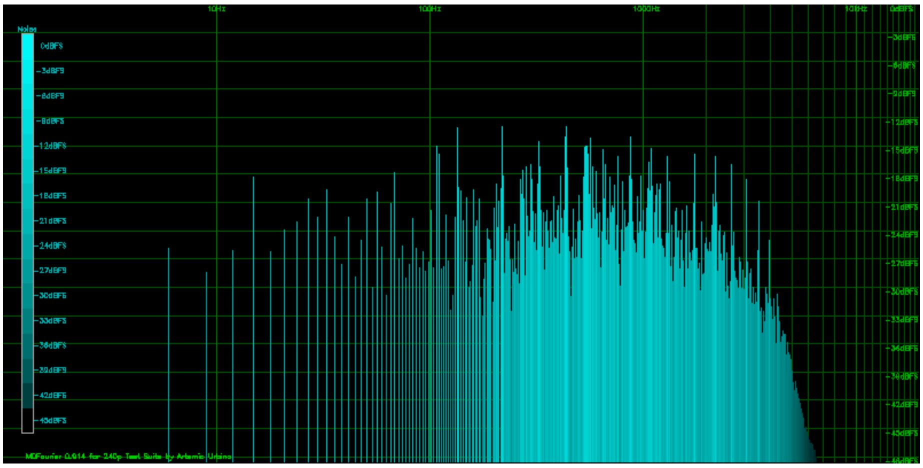

Figure 4.12: Missing frequencies in *Noise* cutoff by high pass filter

And finally, figure 4.12 shows the opposite kind of curve, the *high pass filter* cuts off everything higher than *8khz* in the *Noise* section.

It is a good moment to emphasize that these are relative plots. They show how different the *Comparison* signal is to the *Reference* signal. And so far we've compared the same signal to itself although modified with very precise digital manipulations. An analog filter would look the same, but a bit fuzzier.

However, some interesting ideas arise. What would happen if we take this *low/high pass filter* signal and use it as *Reference* and the original one as *Comparison*?

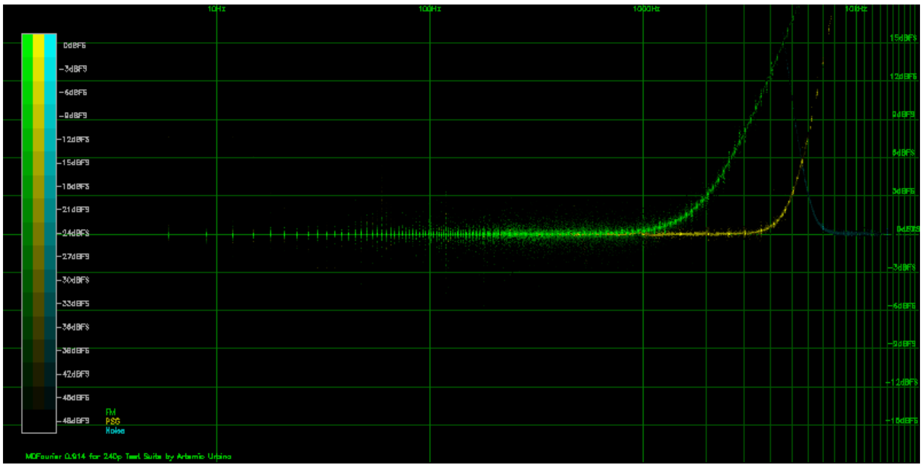

Figure 4.13: Results when using modified signal as *Reference*

Based on this, one could jump to the conclusion that everything will simply be inverted. After all, the original signal now rises to $+\infty$ $dBFS$ at the same spots - and that makes complete sense - since those frequencies now have a higher amplitude.

Although the *Differences* and *Spectrogram* plots will indeed be inverted under these controlled conditions, the *Missing* plots are different. Most of them are now empty:

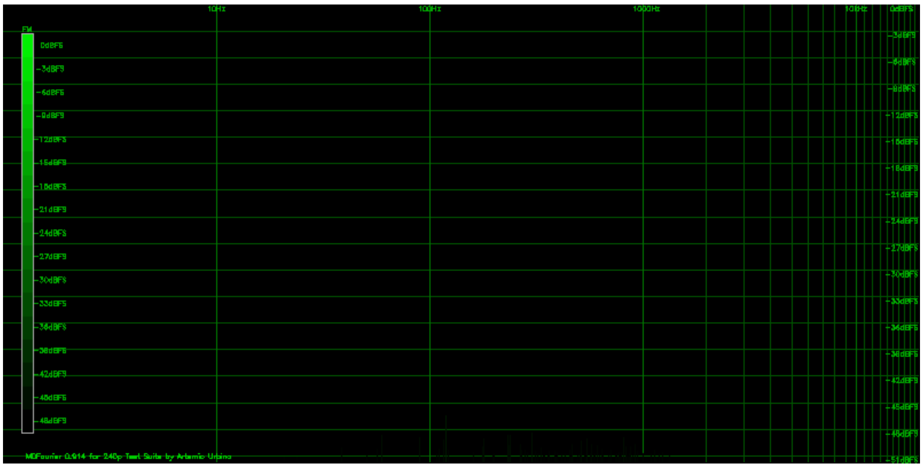

Figure 4.14: *Missing Frequencies* plot for *FM* is empty

This happens because we cut a lot of frequencies with such steep *low* and *high pass filters*, and all the frequency content from this modified signal is present in the original, but not the other way around as we saw before.

4.6 Scenario 5: Comparing two recordings from the same console made with different Audio Cards

We'll now compare the same console using two different recordings, one made with an internal *PCI M-Audio 192* and the other with a *USB Lexicon Alpha*. Here are the results:

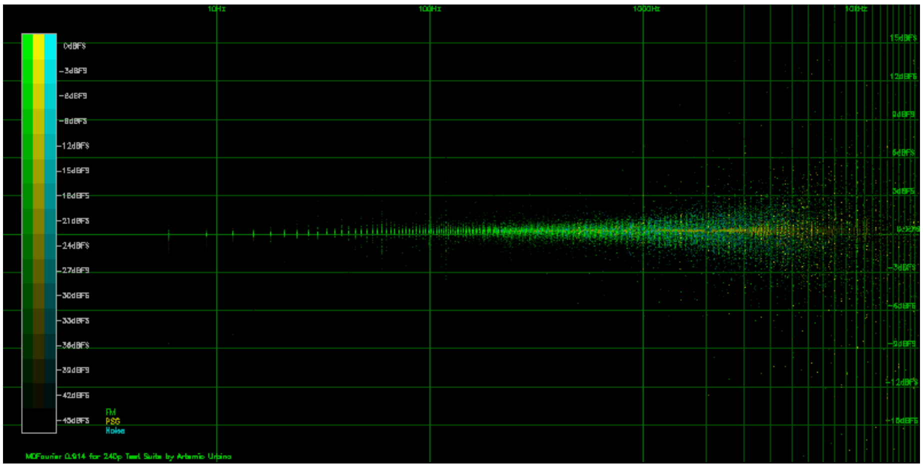

Figure 4.15: Differences using same hardware and cables, different capture cards

Well, I am guessing that was unexpected. We can tell a few things though. First, the frequency response is slightly different, since we now have that huge scatter at the higher end of the spectrum, the treble.

But we can still clearly tell that the scatter is centered around the *0 dBFS* line, which means that even using different sound cards we can tell the differences between systems[5].

Also, there was a slight difference in the detected frame rates. This happens since the sampling clock in not exactly the same in both audio cards. Here is the output text from the analysis that shows the detected frame rates:

```
* Loading 'Reference' audio file A-MD1UTVA3_M-192.wav
- WAV file is PCM 44100hz 16bits and 63.51 seconds long
- Starting sync pulse train:  1.25499s [221380 bytes]
- Trailing sync pulse train:  59.0045s [10408396 bytes]
- Detected 59.914 hz video signal (16.6906ms per frame) from WAV file

* Loading 'Comparison' audio file A-MD1UTVA3-lexicon.wav
- WAV file is PCM 44100hz 16bits and 60.39 seconds long
```

[5] Under the assumption both cards have a relatively flat frequency response, like the ones used here. See the Audio Cards appendix for specifications H.1

```
- Starting sync pulse train:  1.30923s [230948 bytes]
- Trailing sync pulse train:  59.0523s [10416824 bytes]
- Detected 59.9208 hz video signal (16.6887ms per frame) from WAV file
```

The *USB Lexicon alpha* has a more accurate sampling clock, although the difference is minimal[6] and *MDFourier* compensates for such issues.[7]

Here is the graph with the *Average plot* option turned on.

4.7 Scenario 6: Comparing wav vs mp3

4.8 Scenario 7: Comparing two vintage consoles

4.9 Corner cases

[6] We are talking 0.0019ms in this case, that is 0.0000019 seconds!

[7] The expected frame rate from the vintage console analyzed here is 16.688ms per frame, or 59.92hz as measured with a scope. See appendix G

Chapter 5

Results from vintage retail hardware

The following table lists all the hardware used to make the recordings for the plots that will be shown. All had stock parts at the time of the recording, and used original power supplies. They were all connected to a 4" *CRT* via *RGB*, although the *CRT* was turned off while recording.

Although the recordings were made using several audio cards as described in section H.1.

Type	Model	Revision	FCCID	Serial	Region	Made in	MDFourier ID	Recorded from
Model 1	HAA-2510	VA1		89N61751	Japan	Japan	A-MD1JJVA1	Headphone Out
Model 1	1601	VA3	FJ846EUSASEGA	30W59853	USA	Taiwan	A-MD1UTVA3	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	B10120356	USA	Japan	A-MD1UJVA6	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	59006160	USA	Taiwan	A-MD1UTVA6-1	Headphone Out
Model 1	1601	VA6	FJ8USASEGA	31X73999	USA	Taiwan	A-MD1UTVA6-2	Headphone Out
Model 1	HAA-2510	VA6		A10416197	Japan	Japan	A-MD1JJVA6	Headphone Out
Model 2	MK-1631	VA1.8	FJ8MD2SEGA	151014280	USA	China	A-MD2UCVA18	AV Out
Nomad	MK-6100		50059282		USA	Taiwan	A-NMUT	Headphone Out
CDX	MK-4121		Y40 014198		USA	Japan	A-CDXUJ-LO	Line Out
CDX	MK-4121		Y40 014198		USA	Japan	A-CDXUJ-HP	Headphone Out

5.1 A-MD1UTVA3 vs A-MD1JJVA1

5.2 A-MD1UTVA3 vs A-MD1UJVA6

5.3 A-MD1UJVA6 vs A-MD1UTVA6-1

5.4 A-MD1UTVA6-1 vs A-MD1UTVA6-2

5.5 A-MD1UTVA3 vs A-MD2UCVA18

Appendices

Appendix A

Licensing

MDFourier Copyright (C)2019 Artemio Urbina This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see `https://www.gnu.org/licenses/`.

Appendix B

Configuration file

All these parameters are defined in the file *mdfblocks.mfn*[1]. Here is the current file we are using to compare *Mega Drive/Genesis* audio characteristics:

```
MDFourierAudioBlockFile 1.0
MegaDriveAudio
16.688
8820 -25 25 14 18 10
7
Sync s 1 20 red
Silence n 1 20 red
FM 1 96 20 green
PSG 2 60 20 yellow
Noise 3 14 20 aqua
Silence n 1 20 red
Sync s 1 20 red
```

This file defines what *MDFourier* must do and how to interpret the WAV files. For now it can read *44khz* and *48khz* files, in *Stereo PCM* format[2].

The first line is just a header, so that the program knows it is a valid file and in the current format.

The second line is the name of the current configuration, since I plan to add support for any console or arcade hardware in the future. This would imply creating a new *mfn* file for each configuration, and a specific binary to be run on the hardware.

The third line is the expected frame rate. This is only used as a reference

[1]This is the default file name, a different file can be used and selected via the command line. In future *Front End* revisions it might be selected via a combo box. One for each supported hardware profile

[2]This is more than enough for the human hearing spectrum [11]

to estimate the placement for the blocks within the file before calculating the frame rate as captured by the audio capture card. After that is calculated, each file uses its own definition in order to be fully aligned. Variations in the detected frame rate are natural, since we have an error of 1/4 of a millisecond[3], dictated by the sample rates and audio card limitations.

The fourth line defines the characteristics of the pulse tone used to identify the starting and end points of the signal within the wave file. Its frequency, relative *amplitude difference* to the *background noise* (silence), and length *intervals* that will be better explained in future revisions of this document.

The fifth line defines how many different blocks are to be identified within the files. There are seven blocks in this case.

Each block is composed of five characteristics: A *Name*, a *type*, the *total number of elements* that compose it, each element *duration* specified in frames and the *color*[4] to be used for identifying it when plotting the results. Each block must correspond to a line with these parameters.

For example, FM audio has been named *"FM"*, type *1*, *96* elements of *20* frames each and will be colored in *green*. Definition is in frames since emulators and *FPGA* implementations tend to run at different frame rates than the vintage retail platform, which result in different durations. The only way to align them, is by respecting the driving force that tied this up in the old days: the video signal.

There are currently two special types, identified by the letters *'s'* and *'n'*. The first one defines a *sync pulse*, which is used to automatically recognize the starting and ending points of the signal within the wave file.

The second one is for null audio, or silence. This *silence* is used to measure the *background noise*[5] as recorded by the audio card.

[3]For reference 1/4 of a millisecond corresponds to 0.00025 seconds. Modern systems have different frame rates adapted for modern displays, small differences are more likely caused by the audio capture hardware [8]

[4]Available colors are listed in Appendix I

[5]How analysis is affected by this is described in section 2.3

Appendix C

MDWave

MDWave is a companion command line tool to *MDFourier*. During development and while learning about *DSP*, I needed to check what I was doing in a more tangible way. So in order to visualize the files in an audio editor and listen to the results *MDWave* was born.

It takes a single wave file as argument, and loads all the parameters defined in the configuration file in order to verify the same environment. (see section B).

The output is stored under the folder *MDWave*, and a subfolder with the name of the input *WAV* file. The default output is a *Wave* file named *Used* which has the reconstructed signal from the original file after removing all frequencies that were discarded by the parameters used.

This means that it does a *Fourier Transform*, applies the selected *window* (section 3.1.1) and estimates the noise floor. The highest amplitude frequencies are identifies and limited by range for each element defined in the configuration file, and rest are discarded. An *Inverse Fourier Transform* is applied in order to reconstruct the wave file and the results are saved.

The opposite can be done as well by specifying the *-x* option, and the result is a *Discarded* wave file, that has all the audio information that was deemed irrelevant and discarded by the specified options. With this you can listen to these and determine if a more severe comparison is needed.

In addition, the *-c* option creates a wave file with the chunk that corresponds to each element from the *Reference* file being used, trimmed using the detected frame rate. Two chunks are created for each element, the *Source* wav chunk has the element trimmed without modification and the *Processed* wav chunk has the same element but with the windows and frequency trimming applied.

It has a few more command line options, which I'll detail in later versions of the document. You can type *mdwave -h* in your *mdfourier* folder for details.

Appendix D

Window Function equations and plots

This appendix lists the equation and curve of each *Window Function* used to limit spectral leakage as described in section 3.1.1.

D.1 Tukey

The default is a *Tukey* window selected for this purpose. It uses $\alpha = 0.6$, zeroing just a few samples, with minimal spectral leakage and good amplitude response.

The following equation is used to create the slopes:

$$tukey(x) = \frac{1}{2}(1 + \cos(\frac{\pi(|x - \frac{N-1}{2}| - \alpha\frac{N-1}{2})}{(1-\alpha)\frac{N-1}{2}})) \tag{D.1}$$

And this is the resulting plot of the *Tukey* window, ranges are 0.0 to 1.0 horizontally and -0.1 to 1.1 vertically.

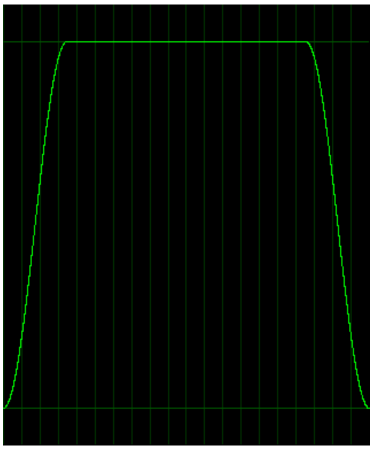

Figure D.1: Tukey window used by *MDFourier*, vertical lines are frames on a 20 frame signal

Detailed information can be found in the reference webpage [18].

D.2 Hann

When selected, a typical *Hann* window is used. This should be used to get the last *spectral leakage*, with a very small trade off in amplitude accuracy.

$$hann[x] = \frac{1}{2}(1 - \cos(\frac{2\pi(x+1)}{n+1})) \tag{D.2}$$

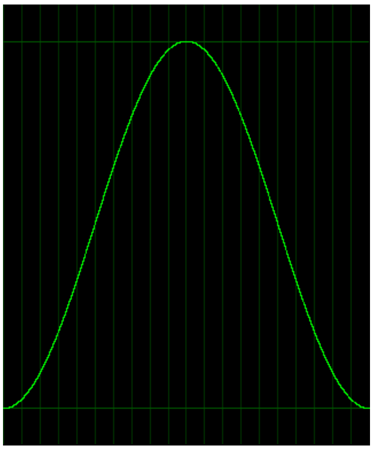

Figure D.2: Hann Window

D.3 Flattop

A typical *Flat top* window is used, selecting this will target amplitude accuracy against frequency bin precision.

$$
\begin{aligned}
flattop(x) = 0.21557895 - 0.41663158\cos(2\pi\frac{x}{n-1}) + 0.277263158\cos(4\pi\frac{x}{n-1}) \\
-0.083578947\cos(6\pi\frac{x}{n-1}) + 0.006947368\cos(8\pi\frac{x}{n-1})
\end{aligned}
$$

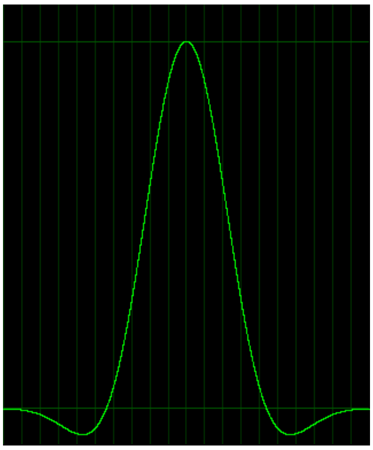

Figure D.3: Flat Top window

D.4 Hamming

A typical *Hamming* window is used, presented for completeness and reference since the samples are never zeroed out.

$$hamming[x] = 0.54 - 0.46\cos(\frac{2\pi x}{n-1}) \qquad (D.3)$$

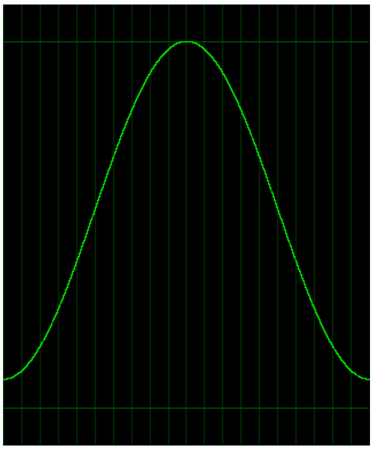

Figure D.4: Hamming window

D.5 No Window

No window is applied to the signal before applying the *DFT*, equivalent to a *rectangular window*. This leaves the signal unprocessed and any uncontrolled decay and audio card noise will be factored in as part of the periodic signal.

There is more information on windows and their usage in the reference webpage [6].

Appendix E

Color Filter Function details

This appendix contains a description, plot and example of each *Color Filter Function* from section 3.1.2.

E.1 None

No filtering is applied, as a result all differences are plotted with the brightest color.

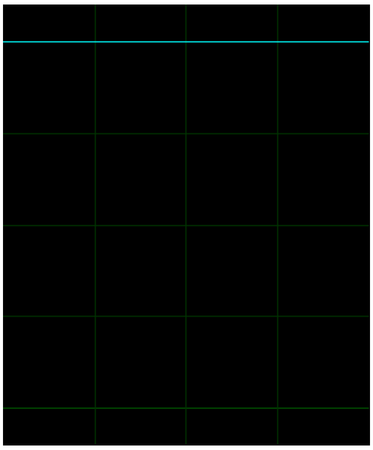

Figure E.1: No Filter

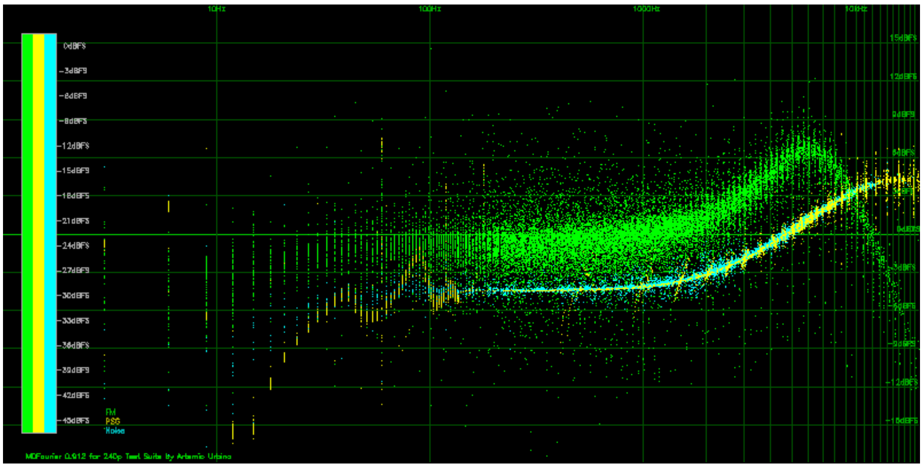

Figure E.2: No Filter Applied

E.2 $\sqrt{dbFS}$

A square root function will only attenuate the lowest amplitude differences.

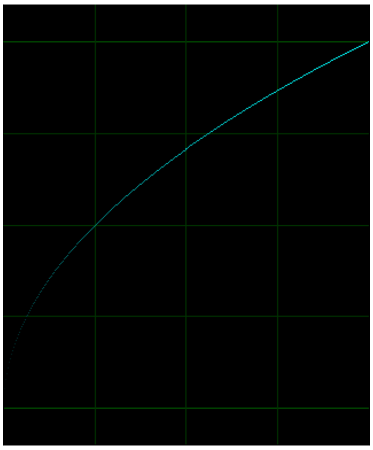

Figure E.3: Square Root filter

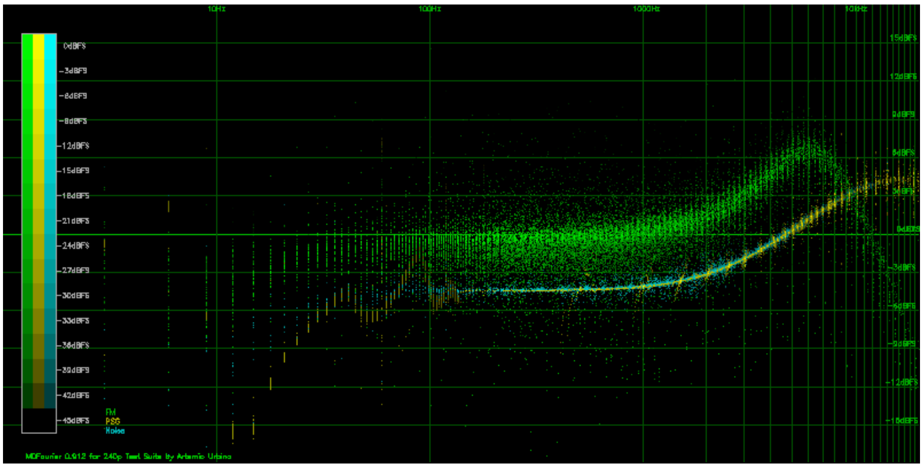

Figure E.4: Square Root filter Applied

E.3 $\beta(3, 3)$

A Beta Function filter with parameters (3, 3) will attenuate a bit more from the lower range, still showing most of the differences.

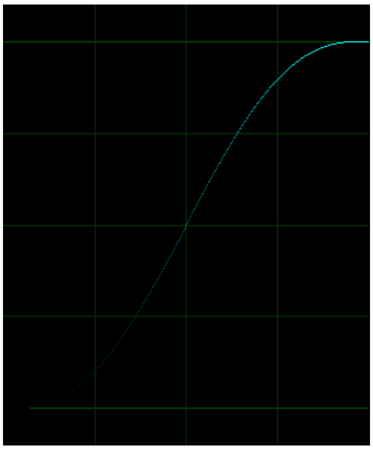

Figure E.5: Beta Function(3,3)

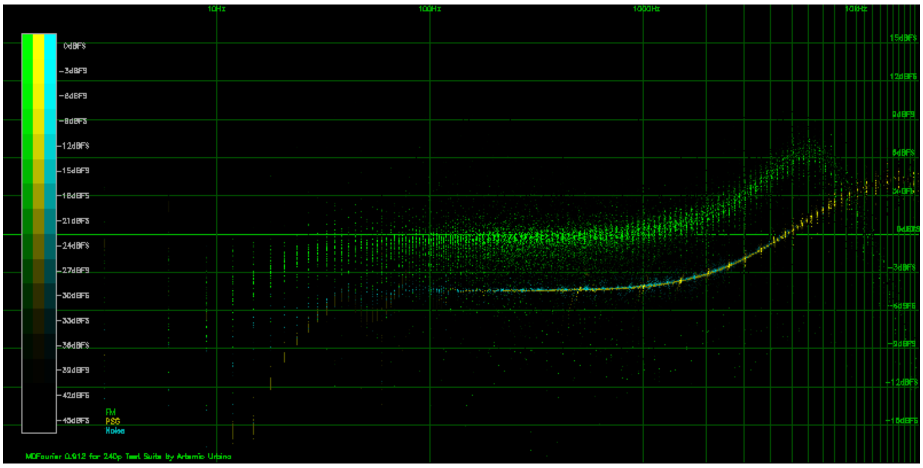

Figure E.6: Beta Function(3,3) Applied

E.4 *Linear*

The linear function is the default, and has no bias. Half the dinamic range corresponds to half the color rage.

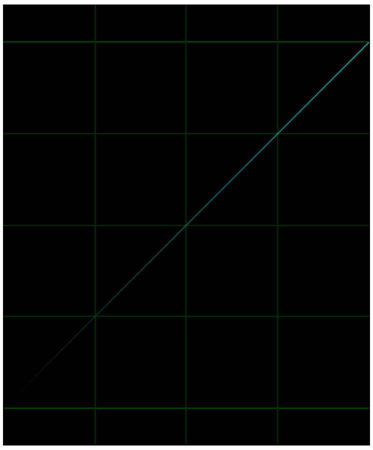

Figure E.7: Linear Function

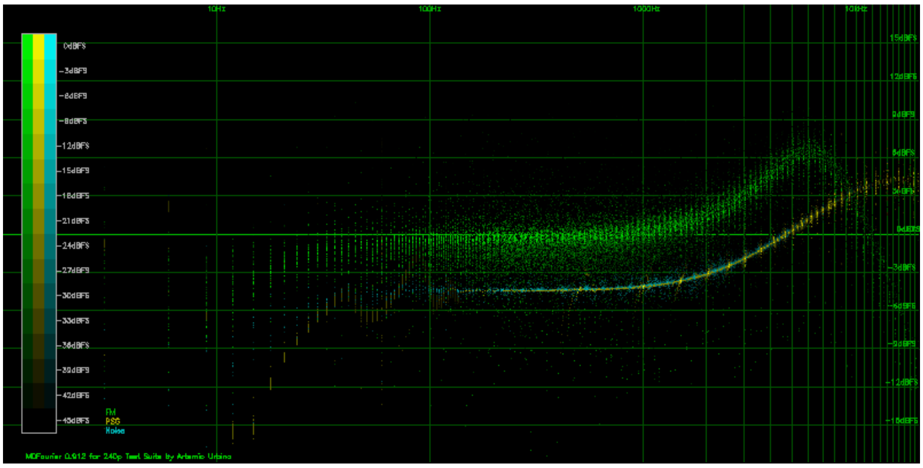

Figure E.8: Linear Function Applied

E.5 $dBFS^2$

A squared function will attenuate a lot more differences, as a result frequencies with the highest amplitude in the reference signal will be brighter.

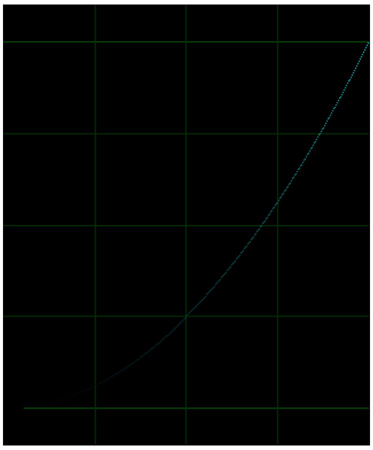

Figure E.9: Linear Function

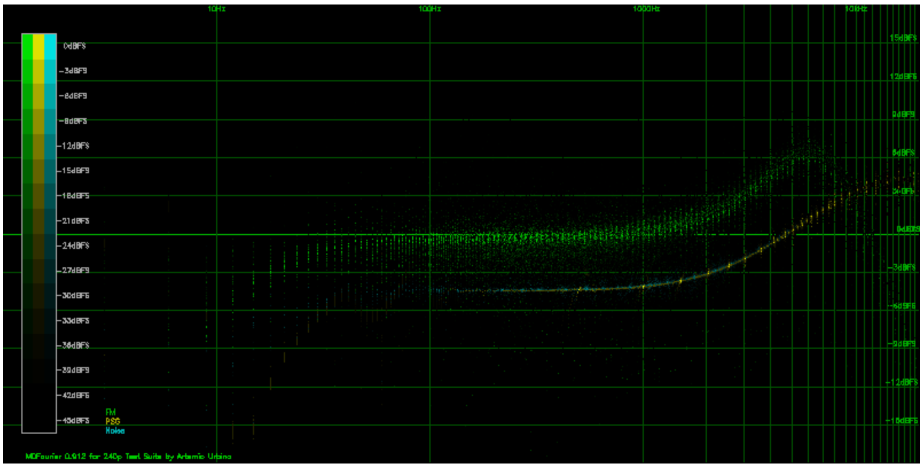

Figure E.10: Linear Function Applied

E.6 $\beta(16, 2)$

A Beta Function filter with parameters (16,2) will attenuate almost all the differences, and only the frequencies with the highest amplitude in the reference signal will be brighter.

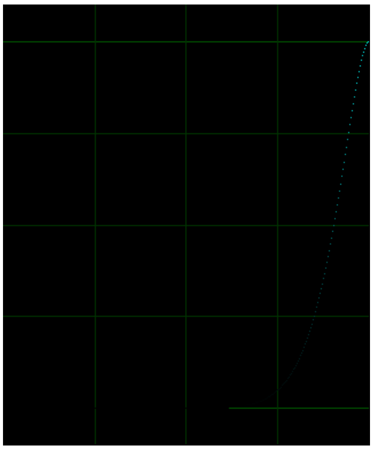

Figure E.11: Beta Function(16,2)

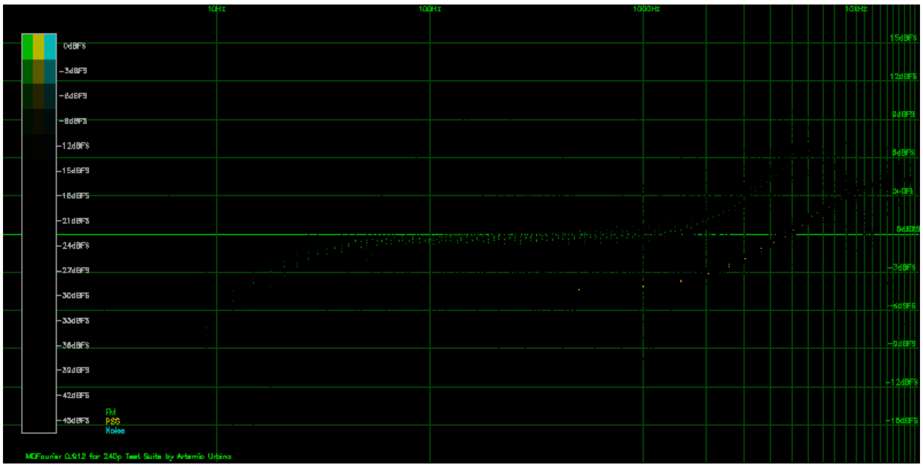

Figure E.12: Beta Function(16,2) Applied

Appendix F

Normalization and amplitude matching

Since each signal is probably at its own distinct volume, we need to perform a a *normalization* process in order to have a common compare point between them.

Each type of *normalization* has its own strengths and weaknesses, but the *frequency domain* normalization designed for this process is always accurate with respect o the *Reference* signal, but might be confusing in some corner cases[1] if the underlying causes are not well understood when interpreting the results.

Since silence can't be used as reference[2], the only other option is fixing points from within the signal. Every normalization process used follows a different logic for fixing one such point for comparison.

The default is to do this in the *frequency domain*[3], but there are two other options available via command line[4]. All three methods are described in the following sections.

F.1 Frequency domain normalization

This is the default option used my the software. It involves finding the *highest magnitude*[5] from the *Fourier Transform* of the *Reference* signal before ampli-

[1]One such example is available in section 4.9

[2]Noise floor will vary by console and by audio card

[3]This means it is performed after the Discrete Fourier Transform and using the data generated by it.

[4]These can be enabled via the *extra command* option from the GUI as explained in section 3.1.6

[5]At this point in the analysis there are no amplitudes defined, since we have no reference points. Hence we don't use amplitude, and raw magnitude values from each transform are

tudes are calculated. Then a corresponding match in the *Comparison* signal's frequency spectrum for the same block is searched for.

This means that the exact same fundamental frequency with the highest magnitude value is searched for, occurring at the same position in time - which corresponds to the block. Having both points, a meaningful *reference point* is set for the comparison, and the relative amplitudes between the signals can be calculated.

In order to calculate the amplitudes, *0dBFS* is matched against the absolute highest magnitude from the *Comparison* file[6], after both signals have been relatively normalized in amplitude against the reference point.

This method has shown to be always accurate within the tests. However, results can be unexpected in certain corner cases as the one shown in section 4.9.

F.2 Time domain normalization

The process is very similar to the *frequency domain* variant but it is done using the raw audio samples directly from the *WAV* file[7].

The highest amplitude is searched for within the samples of the meaningful audio signal, as dictated by the configuration file[8]. The corresponding segment in time is then located in the *Comparison* file, and a pre-defined duration[9] is searched for the sample with the *maximum local* value.

The *Comparison* signal is then absolutely normalized to *0dBFS*[10], and the *Reference* signal is then relatively normalized using the adjusted *local maximum* value. This follows the same rationale described for the *frequency domain* equivalent.

Although the results can sometimes be deceivingly familiar, they are not correctly referenced in corner cases[11], and they do not represent the real relation between the signals. However, they can be useful for analysis while in the process of understanding how to interpret the plots.

used.

[6] Since the highest point of the *Reference* signal was matched to one other point in the *Comparison* file, there are only two options: either both peaks are in the exact same spot, or there is a higher peak in the *Comparison* file

[7] Values are changed only in RAM during execution, input files are never modified

[8] Described in section B

[9] One frame in the current implementation

[10] Relative to *0xFFFF* - the top value in *16 bit* samples in the *WAV* file

[11] See section 4.9 for an example

F.3 Highest fundamental average normalization

This normalization option also takes place in the *frequency domain*. The idea is to average the highest magnitudes - the fundamentals - from all segments and use the resulting ratio calculated between both signals to normalize them.

The results are always centered around the *0dBFS* line in the *Differences plot*, allowing a globalized view. However, the amplitude differences are not to be relied upon for calibration, since they are not relative to a fixed point from the *Reference* signal. This option is just available for cases when both signals present a high difference rate between them, which is unexpected but possible.

Appendix G

Frame rate

The *frame rate* is the amount of frames per second a system sends to the display. Since the *custom binary* runs from within the target system, we are subject to the internal timing. Every time a frame starts the process for being sent to the display, an event called *vertical sync* occurs. This is the driving clock for the whole console[1].

It is vital for the process to have a basic unit of time, in order to segment the file in the chunks needed for analysis and to send the correct values to the *Discrete Fourier Transform*.

Signals generated from different sources can have dissimilar *frame rates*, even when running the same programs and representing the same platform. This can be due to several reasons: having a different display technology as target, hardware inconsistencies between revisions, etc.

Another source for variation is the audio capture device, since these can vary slightly in their sample rate clocks. But usually these are lower variations that are reported and compensated for internally.

The frame duration is the basic unit of measurement, since we are dealing with signals that are well below one second. Because of this, the configuration file[2] defines the expected frame duration in milliseconds as measured from a vintage console. One such example is the frame duration as measured from an *NTSC*[3] *Sega Genesis* using an *oscilloscope*[4] as shown in figure G.1.

[1] As a matter of fact, almost all the code for running games - or the custom binary in out case - is executed during a segment called *vertical blank*.

[2] See appendix B

[3] *National Television System Committee*

[4] Electronic tool for measuring electric signals.

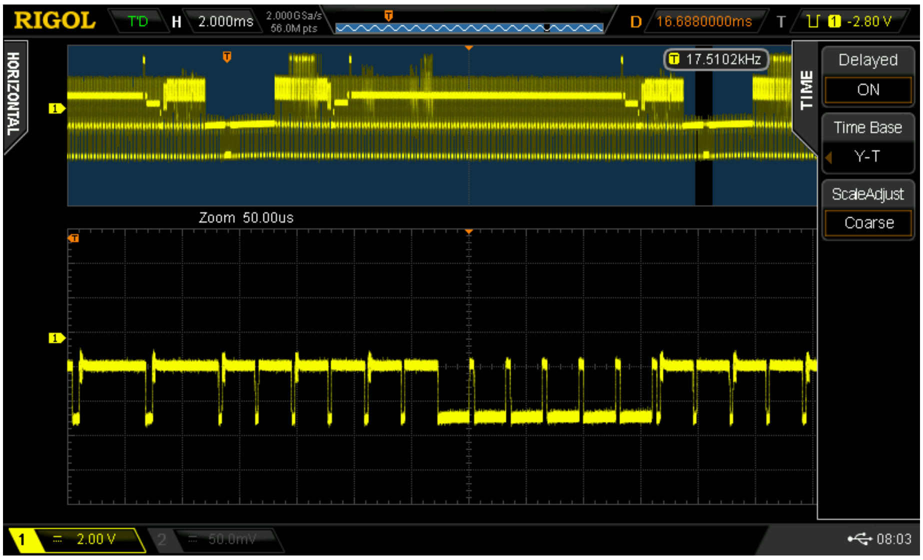

Figure G.1: Frame duration as measured with a scope from a *Model 1 VA3 system*

Appendix H

Requirements

H.1 Audio capture device

For capturing the audio files, an audio capture device is needed. It is recommended to use a musical grade audio card in order to get a flat frequency response across the human hearing spectrum.

So far two audio cards have been used in my personal setup. The reference recordings available for download were made with both cards, a set with each one of them.

I have no association or business relationship with these products, they are just presented as the references used. As more people use the software and we as a community compare files, this list can be expanded with recommendations.

- **M-Audio Audiophile 192:** An internal audio card that is no longer available in the market [20]
- **M-Audio M-Track:** A USB audio card. [22]
- **Lexicon Alpha:** An affordable USB audio card. [21]

We have tested some cards that don't have a flat frequency response, you should try to use a sound card that is aimed to musicians or instrument recording.

Sound cards have their own sampling internal clock, which tends to deviate enough that frame rate differences can be detected by *MDFourier*. This is compensated for in the time domain while trimming, and it shouldn't be a problem in the frequency domain due to the small variation. [8]

H.2 Computer

Any computer can be used if you are compiling the source code from scratch [3], but a statically linked *Microsoft Windows executable* is provided for convinience, alongside a front end. See chapter 3 for instructions on using the GUI.

H.3 Game Consoles or emulators

You'll need either the provided example audio files or create your own by recording from the desired source, which makes more sense since you probably want to compare how these behave.

H.4 Flash cart, or means to run the binary

The console needs to run a custom built binary, a ROM. I will build this functionality into each version of the *240 test suite* [4] as possible.

In order to run these, you'll either need a flash cart or a custom loading solution compatible with the target platform.

H.5 Cables and adapters

You'll need cables and maybe some adapters to connect the audio output fomr the console to the input of your audio capture card.

H.6 Audio capture software

Your capture card will probably be bundled with some audio editing software, or you can use Audacity [23] or Goldwave [24] options depending on your operating system.

Appendix I

Colors available for plots

This is the list of colors that can be used in the *MFN configuration file* described in section B:

red, green, blue, yellow, magenta, aquamarine, orange, and *purple*.

Appendix J

Compiling from source code

J.1 Dependencies

MDFourier needs a few libraries to be compiled. In *Linux*, *UN*X* based systems and Cygwin [17]; you can link it against the latest versions of the libraries.

- Fastest Fourier Transform in the West (fftw) [5]
- The GNU plotutils package [15]
- PNG Reference Library: libpng [14]
- Incomplete Beta Function [19] (included with source code)

The pre-compiled binary for *Windows* is created with *MinGW*[16] and statically linked for distribution against:

- fftw-3.3.8 [5]
- plotutils-2.6 [15]
- libpng-1.5.30 [1]
- incbeta [19] (included with source code)

The *makefiles* to compile either version are provided with the source code [3].

[1] This older version was used to simplify the build process in this statically linked executable. Sources at `https://sourceforge.net/projects/libpng/files/libpng15/1.5.30/`

Appendix K

Contact the author

You can contact me via twitter `http://twitter.com/Artemio` or e-mail me at *aurbina@junkerhq.net*.

Appendix L

Acknowledgements

I'd like to thank the following people for helping me so these tools could be completed. First and foremost to my family, who helped me find the time to learn and make progress.

Bibliography

[1] Bracewell, Ronald N. *The Fourier Transform and Its Applications (2 ed.).* McGraw-Hill. ISBN 978-0-07303938-1.

[2] Ace, *GUIDE: A complete overview of the many Sega Genesis/MegaDrive models* `http://www.sega-16.com/forum/showthread.php?7796-GUIDE-Telling-apart-good-Genesis-1s-and-Genesis-2s-from-bad-ones`

[3] Github, *MDFourier source code C99*, `https://github.com/ArtemioUrbina/MDFourier/`.

[4] 240p test Suite, *Wiki web page*, `http://junkerhq.net/240p/`.

[5] Fastest Fourier Transform in the West., *web page*, `http://fftw.org/`.

[6] Window Types: Hanning, Flattop, Uniform, Tukey, and Exponential , *web page*, `https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Window-Types-Hanning-Flattop-Uniform-Tukey-and-Exponential/ta-p/445063/`.

[7] dB Full Scale, `https://www.sweetwater.com/insync/dbfs/`

[8] Sound Card Sampling clock variation `http://www.stu2.net/wiki/index.php/Calibrate_Sound_Card/`

[9] So how accurate is a typical PC sound card? How stable is the output? How would one measure this? Experiments with a PC sound card by leapsecond `http://www.leapsecond.com/pages/sound-1pps/`

[10] Each sound card has it's own sampling clock, Goldwave support forums `http://www.goldwave.ca/forums/viewtopic.php?p=17470`

[11] D/A and A/D | Digital Show and Tell (Monty Montgomery @ xiph.org) `https://www.youtube.com/watch?v=cIQ9IXSUzuM`

[12] Spectral Leakage and Zero-Padding of the Discrete Fourier Transform `https://dspillustrations.com/pages/posts/misc/spectral-leakage-zero-padding-and-frequency-resolution.html`

[13] Why is it a bad idea to filter by zeroing out FFT bins? `https://dsp.stackexchange.com/questions/6220/why-is-it-a-bad-idea-to-filter-by-zeroing-out-fft-bins`

[14] libPNG *web page*, https://libpng.sourceforge.io/

[15] GNU Plot Utils *web page*, https://www.gnu.org/software/plotutils/

[16] MinGW, *Minimalist GNU for Windows*, http://mingw.org/.

[17] Cygwin, *a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.* https://www.cygwin.com/

[18] Tukey window, Recording Blogs https://www.recordingblogs.com/wiki/tukey-window

[19] incbeta, *Incomplete Beta Function in C*, https://codeplea.com/incomplete-beta-function-c/.

[20] M-Audio Audiophile 192, *Specifications*, https://www.soundonsound.com/reviews/m-audio-audiophile-192/.

[21] Lexicon Alpha USB card, *Product web page*, https://lexiconpro.com/en/products/alpha/.

[22] M-Audio M-Track USB card, *Product web page*, https://www.m-audio.com/products/view/m-track.

[23] Audacity *web page*, https://www.audacityteam.org/.

[24] Goldwave *product web page*, https://www.goldwave.com/.

Glossary

dBFS Decibels relative to full scale. 9

DFFT Discrete Fast Fourier Transform. 14

DFT Discrete Fourier Transform. 8

FPGA Field-programmable gate array. 4

GUI Graphical User Interface. 7

NTSC National Television System Committee. 51

WAV Waveform Audio File Format, also referred as WAVE. 9