Multiband Compression

Abstract: We have implemented multiband compression in Matlab. This compressor is a refined, more flexible version of both the standard compression method and dynamic range compression. What this type of compression does is it takes a signal, specifically an audio signal, then splits it up into bands, all made of different frequency bands. The bands are then altered where any band modification remains independent of the other band modifications. This is to balance out the different sounds in the signal so that they mesh well together without having to modify the audio signal in its entirety if not needed. Due to the control that one has over the final outcome on the signals, this method has increasingly been implemented into various applications. How we chose to approach this was by first creating an intuitive graphical user interface that contained the common parameters needed to carry out this type of compression. Our GUI then communicates with our main Matlab code which is responsible for the actual processing of the signals. Our code consisted of three adjustable filters that were responsible for splitting up the audio signal. After running sample audio files through our compressor, we found that we achieved sufficient results.

I. Introduction

The purpose of a dynamic range compressor is to map the natural dynamic range of a signal to a smaller range. This in turn allows the volume of loud sounds to be reduced as well as the amplification of quieter sounds and this method of signal compression has practical applications in things such as mixing, broadcasting, recording, as well as other things. Even though dynamic range compression can aid in smoothing out audio signals, the drawback of it is that it is restricted to the signal as a whole rather than specific instruments or sounds in the audio.

The multiband variation of dynamic range compression adds the ability to carry out this compression on independent frequency bands. The audio signal is split into multiple bands with each band getting passed through its own compressor where it is modified independent of the other bands. Each band provides the option to adjust for ratio, threshold, attack, and release. After each band is modified, the signals are then merged back together with some measures in place to make sure that the merged signals do not cause unwanted peak levels. A big advantage that multiband compression has over full-bandwidth compression is that unnecessary audible gain changes in the other frequency bands are avoided since signal levels in a specific band are separate from the signal levels in another band. This specific type of compression is used a lot in the mastering of audio and is a common part of an audio workstation due to its flexibility. The input-output relationship of a compressor is shown in figure 1.

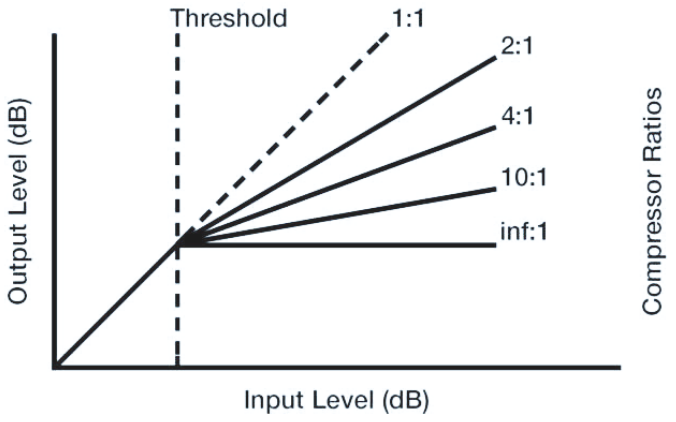

Figure 1: The input-output relationship of a compressor can be visualized in the above diagram, courtesy of Icon Collective College.

II. Background

Consistency is one of the main reasons for utilizing multiband compression. The ability to have the highs, mids, and lows of each instrument or voice balanced and evened out make for a powerful tool for editing audio signals, avoiding many of the problems that might be left unaccounted for with the basic standard compression. One of the main uses of multiband compression is in anything that has to do with modifying and refining audio signals. As mentioned before, this tool has been becoming more and more popular in the workstation used by sound engineers, producers, and others that deal with any kind of audio. In music production specifically, this is an extremely handy way to get the different instruments and vocals to pop in a song. Whether the vocals come out as being slightly too quiet after recording or if a specific instrument is just too loud and is drowning out some of the other sounds, modifying certain frequency ranges rather than altering the whole signal gives the engineer a lot more control of over how the final product will sound to the audience while also maintaining high sound quality. This is especially important since in the real world, most of the time vocal signals come out as being weaker and not as vibrant as they should be on certain words or vowels and this provides a great remedy for that and this type compression results in an overall natural and more transparent sound.

Multiband compression also plays a role in other applications such as gaming. With the rise of livestreaming video games on platforms such as YouTube and Twitch mostly replacing the old format of uploading gaming videos online, a user can no longer “fix” their audio after recording since they are being broadcasted live. A livestreamer may suddenly shout in response to something happening in their game, or they may become quiet while focusing. This sudden change in volume might annoy or even scare viewers. A compressor can fix this issue and allow for a more consistent volume level, and a multiband compressor can take this even further and allow for even finer adjustment. For instance, the livestreamer can adjust their low-frequency band to attenuate background noise (such as the humming of an air conditioner) while applying compression to their voice to reduce volume changes. In this scenario, the multiband compressor would act as a compressor on the voice and an equalizer to reduce background noise. This also has other applications such as filmmaking to highlight or even mask certain sounds such as the sound of rain when characters are speaking to each other.

III. Methodology

The working principle of the compressor is to see if a segment of audio passes a user-defined threshold and attenuate the signal by a user-defined ratio if the signal passes this threshold. Our multiband dynamic compressor implementation was inspired by Fruity Multiband Compressor, a plugin offered in FL Studio, a popular music production software. The user interface can be seen in figure 1.

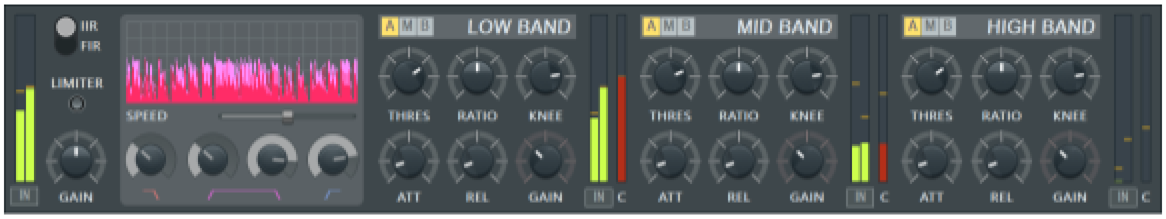

Figure 2: Fruity Multiband Compressor included in FL Studio 20

A multiband compressor uses three individual compressors, one for each band of the audio signal. The key parameters to be implemented for each compressor is the threshold, ratio, and gain since they both define how much the compressor will adjust the volume of the audio. We chose to neglect the other parameters since they are difficult for the average listener to notice and would add extra complexity to the compressor for very little benefit.

Our implementation began with the design of three adjustable filters to split the audio into bands. Each filter is configured to have a length of 1025, and we use MATLAB's built-in sinc function to avoid issues with $sinc(0) = \frac{sin(0)}{0}$ and having to perform L'Hôpital's rule. A low-pass filter can be generated using the general formula 1:

$$h_{LP}[n] = \frac{sin(2\pi f_c\ n/f_s)}{\pi n} \quad where \quad -M \le n \le M \qquad (1)$$

This can be modified for a bandpass and highpass filter. Additionally, each filter was windowed using a Blackman window. This type of window was chosen because it offers high stopband attenuation and very low stopband ripple, but it comes at the cost of transition width. In our case, a wide transition width is actually favorable since it offers a good blending effect between the bands. Using a rectangular window created unnatural sounding "jumps" in the compressor's output audio from where the filter was applied but the cutoff frequency was in the middle of a prominent frequency component in the audio.

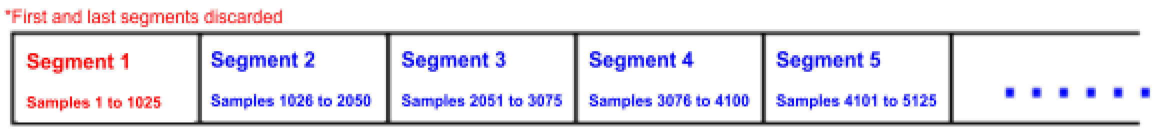

Figure 3: The compressor processes small segments of the signal at a time.

The compressor program outputs small segments of the audio signal at a time that are stitched together to create the output, as shown in figure 2. Our program is designed to work with segments of size N = 1025 (equal to the length of the filters, or 2M+1). These segments are then convolved with the three filters to split up the audio into its respective frequency bands. The key to performing this convolution is to convolve the filter with the previous segment prepended to the current segment that is going to be processed. If this hadn't been done, the segment will be zero-padded and the output segment will have transients from where the filter does not fully overlap the segment's data. This can be shown graphically in figure 3. The output of this convolution can be trimmed down to length N=1025 and placed into a temporary buffer. These two operations can be shown in the following code, there i is an integer counter for the segment number:

```
lowSegment = conv(lowPass,y((i-2)*segmentSize+1:i*segmentSize,1));
lowSegment = lowSegment(segmentSize:end-N)
```

After convolving each segment with the three filters and trimming the segments down, the maximum magnitude of each segment is found to get the peak audio level. This is simply done using MATLAB's max function on the absolute value of the segment. Once the maximum amplitude is found, we can calculate the attenuation to be applied to the signal. If the maximum amplitude of the segment is larger than the user-defined threshold for that band, the signal is compressed using formula 2. This involves converting the peak value into decibels and then converting the output back to magnitude from decibels using MATLAB's mag2db and db2mag functions. The segment is then multiplied by the ratio $target/maximum$ to scale it correctly to the desired level.

$$output\ (dB) \ = \ \frac{input\ (dB) - threshold\ (dB)}{ratio} + threshold\ (dB) \qquad (2)$$

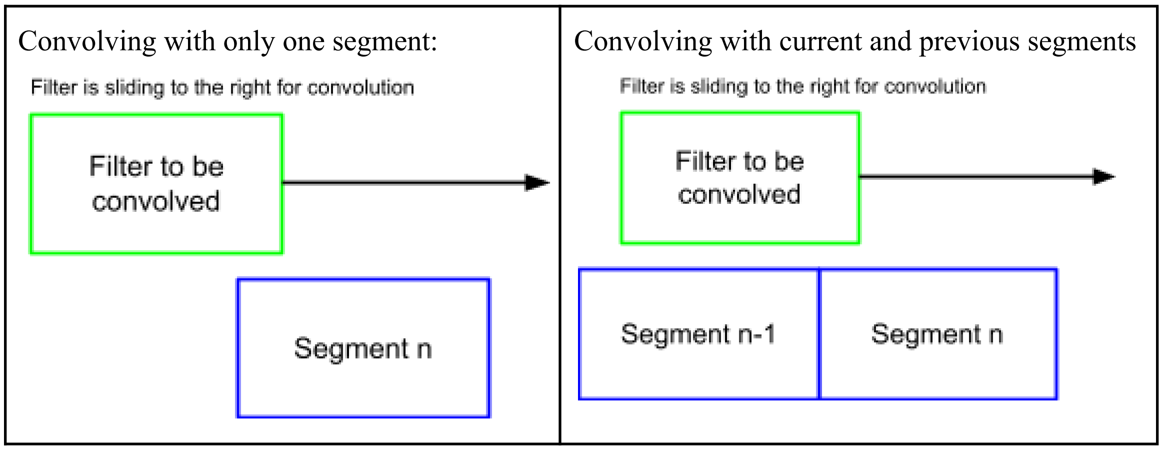
Convolving with only one segment:
Filter is sliding to the right for convolution
Filter to be convolved
Segment n
Convolving with current and previous segments
Filter is sliding to the right for convolution
Filter to be convolved
Segment n-1
Segment n

Figure 4: Comparing the convolution of one segment versus convolving two segments

The program then loops to work on the next segment until the audio clip ends. The processed segments are placed in an output buffer, and once the loop is done iterating through the audio clip the makeup gain (also sometimes called output gain) is applied to the entire clip. The processed audio is output to a file with the same name and filetype as the input.

A feature that we implemented is using a computer's graphics processing unit (GPU) to perform the matrix math. If the user has MATLAB's Parallel Computing Toolbox installed and a CUDA-compatible GPU in their computer, the program will automatically use GPU arrays to perform all matrix math. The program can take advantage of the GPU’s high speed memory and hundreds or thousands of processing cores to perform convolutions and matrix multiplications leaving the user’s CPU free to perform other tasks.

IV. Results

The compressor was first tested using a pure sine wave as the input with a frequency of 440 Hz. We can see the input and output shown side-by-side in figure 5. After the input passes a certain volume, the compressor turns on and attenuates the signal by the given ratio. In this example, a threshold of -10 dB (0.316 amplitude) was used and a ratio of 5 was used. We can clearly see the compressor turn on at -10 dB and decrease the amount that the signal rises by.

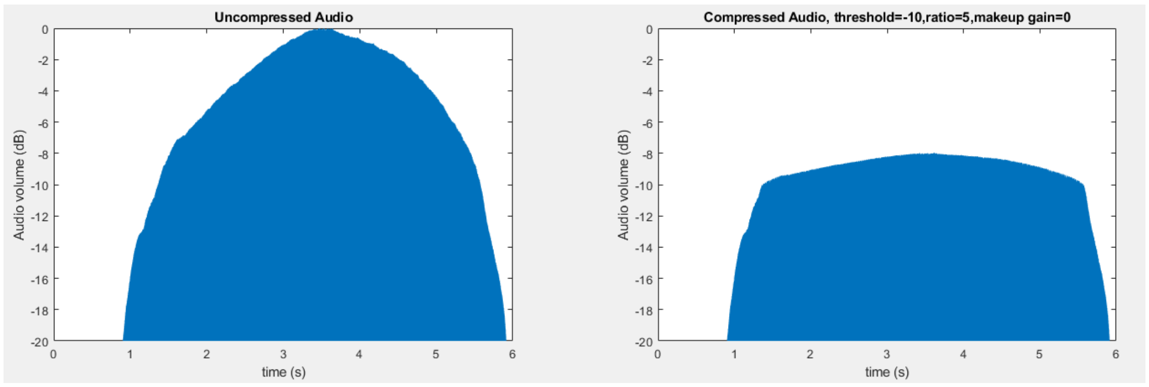

Figure 5: Compression being applied to a 440 Hz sine wave of increasing amplitude.

To test the multiband aspect of the compressor, we added the previous 440 Hz sine wave (reduced in volume to prevent clipping) to a much quieter 150 Hz sine wave. The goal of the multiband compressor is to decrease the volume of the 440 Hz component to allow the 150 Hz component to be more easily heard. The top two plots of figure 6 shows the two frequency components of the input signal. It should be noted that they are shown as separate signals on this report to help show the effects of the compressor easily, but they are actually combined in a single audio file. The multiband compressor compresses the frequency band that the 440 Hz signal is contained in while leaving the 150 Hz signal untouched since it was in a different frequency range that did not pass the threshold.

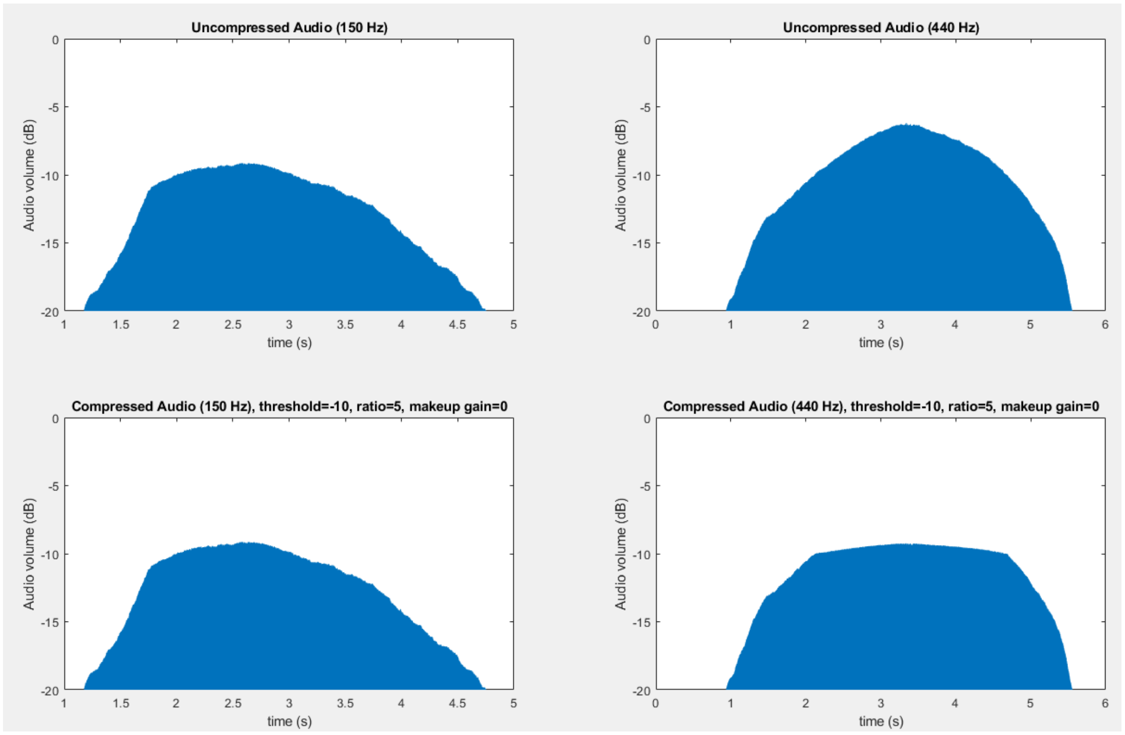

Figure 6: The top two plots are the 150 Hz component and 440 Hz components of the input signal (displayed on separate plots for clarity), and the bottom two plots are those same frequency components after going through the dynamic compressor (settings: transition 1 = 250 Hz, transition 2 = 1000 Hz, all thresholds = -10 dB, all ratios = 5, makeup gain = 0). The louder 440 Hz component was reduced in volume, making the 150 Hz signal easier to hear.

V. Discussion

When carrying out this project our group did run into some difficulties. The first obvious problem that we encountered was learning exactly what multiband compression, or even just standard or dynamic compression, was and how it could be useful to anyone. This was especially difficult since in order to effectively implement multiband compression, we had to be able to develop a thorough understanding of it in a short period of time.

Another problem that we ran into was being able to modify the bands of our audio signals in real time. The professional software services on the market that implement multiband compression almost always allow for the compression of real-time audio such as speaking into a microphone or modifying an audio file as it plays. What we opted to do was just read the data of the .wav file, split it into the different frequency bands to edit them individually, then merge the bands back together in the form of a new .wav file.

If given more time we could work on better understanding the intricacies of this compression method rather than just getting a high level understanding of it so that we would be able to refine our code for it, making our results a lot more precise. Also including the real-time analysis/processing of the audio signals would be a very practical addition that would allow someone like an audio engineer to efficiently process their records. Some more minor features that are common in the professional multiband compressors would also be a good addition as well. These include things like the knee setting which refers to how the compressor transitions between the non-compressed and compressed states of an audio signal running through it, lookahead process which processes signals that are some time ahead of the current signal, and more.

VI. Conclusion

Our project implements a multiband dynamic compressor. This type of compressor is seeing an increasing use in the music, gaming, filmmaking, and many other industries because of its flexibility and its ability to balance audio better than a standard compressor. Our compressor analyzes an input file's audio in three separate bands and applies a compression algorithm to achieve dynamic compression. It achieves this by passing the input through a low pass, band pass, and high pass filter, analyzing a small segment of the audio signal, and applying a calculated amount of attenuation to each segment. The end product is then amplified by the user-specified makeup gain. Our GUI makes it easy to interface with our program without having to write any code. Applying our program to a few test signals gave us great results, with the compressor properly compressing frequency components of the audio signal that pass the user-defined thresholds. While this compressor works well in our testing, it can further be improved by adding additional features found in professional multiband compressor plugins or dedicated multiband compressor devices.

VII. Bibliography

"Audio Compressor Ratio Explained." *Icon Collective College of Music*, 27 May 2020, iconcollective.edu/audio-compressor-ratio-explained/.

"Fruity Multiband Compressor." *Fruity Multiband Compressor - Effect Plugin*, Image-Line, www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/plugins/Fruity%20Multiband%20Compressor.htm.