

Automatic Long-Term Loudness and Dynamics Matching

Earl Vickers

Creative Advanced Technology Center
Scotts Valley, CA, USA
earlv@atc.creative.com

ABSTRACT

Traditional audio level control devices, such as automatic gain controls (AGCs) and compressors, generally have little or no advance knowledge of the dynamic characteristics of the remainder of the current audio program. If such advance knowledge is available (i.e., if audio files can be pre-analyzed), it becomes possible to match desired values of overall loudness and dynamics. We introduce two new measures, “long-term loudness matching level” and “dynamic spread,” and present new methods for long-term loudness and dynamics matching.

0 INTRODUCTION

Loudness is a subjective measure relating to the physical sound pressure level (SPL) as perceived by the human ear. A number of devices have been created for controlling audio levels to modify either a signal’s loudness or its dynamic change in loudness.

Automatic Gain Controls (AGCs) are typically used to minimize loudness differences between audio programs (for example, between one song and the next). Compressors are similar to AGCs but operate on a faster time scale; they are primarily intended to minimize the loudness changes within a single song or audio program [1, 2]. Compressors have a number of uses, including increasing the loudness of the softer parts of an audio program so they can be heard above the noise floor (e.g., for automotive listening), decreasing the loudness of the loudest segments (for example, to avoid disturbing neighbors during late-night listening), and keeping signal levels within technical limits required for radio broadcast.

Compressors and AGCs typically operate in real-time with little or no advance knowledge of the contents of the remainder of the current audio program. It seems likely that if we had additional information about the dynamic characteristics of the audio program as a whole, we could do a better job of matching a desired loudness or dynamic behavior. Since music data is often stored in sound files on computer hard drives, we are in a position to generate and use loudness metadata in order to improve performance and reduce artifacts.

In this paper, we present a method for matching the loudness of an entire song or sound file to a desired level using a novel measure, “long-term loudness matching level.”

In addition, we present a compressor that analyzes the dynamic characteristics of a sound file and matches the output to a desired statistical behavior, using a new measure called “dynamic spread.” This prevents over-compressing audio that already has limited dynamics.

One side effect of dynamic compression is that it can alter the overall loudness in a way that may vary from one recording to the next, making it difficult to perform post-compressor loudness matching if the compression is done in real-time. Therefore we present a method for estimating the effect of any given compressor settings on a particular sound file, so we can automatically compensate by scaling the gain accordingly.

1 LONG-TERM LOUDNESS MATCHING

Normalization is a way of matching the levels of multiple sound files by scaling each one to the maximum extent possible without clipping. Unlike traditional compressors and AGCs, which operate in real-time with minimal look-ahead capability, normalization operates on a sound file as a whole, applying a single gain to the overall signal. By examining the entire sound file in advance, the normalizer is able to scale the audio without making any (possibly unwanted) gain adjustments during playback.

Unfortunately, there is no guarantee that two normalized sound files will sound equally loud. The peak amplitude of a song is not a very robust measure of its loudness. What we actually want is to normalize the perceived loudness, not the peak amplitude.

While a number of attempts have been made to define and quantify the loudness of a single, short-duration tone [3-5], there is little agreement as to how to combine a series of short-term loudness values to define the loudness of an extended, dynamically changing signal such as an entire song.

I. Allen, in an analysis of the loudness of movie soundtracks [6], determined that the equivalent loudness,

$$Leq_m = 10 \log_{10} \left(\frac{1}{T} \int_0^T \left(\frac{P(t)}{P_0} \right)^2 dt \right)$$

(where P is the sound pressure level and the subscript m refers to the type of frequency weighting, for example, the “A” equal-loudness curve), yields a good match to the relative subjective loudness of various soundtracks. Allen concluded that the Leq is better than a (C-weighted, fast) peak level measurement for determining subjective loudness.

The software program “Sound Forge” [7] includes a similar loudness definition as an optional part of their normalization process, using the average RMS effective amplitude instead of the average sound pressure level.

1.1 Emphasizing Louder Frames

Zwicker and Fastl proposed performing a critical-band analysis and calculating the percentage of time for which a given loudness is reached or exceeded [8]. They presented evidence suggesting that the loudness of a dynamically changing sound can be well-characterized by its N_5 loudness; i.e., the level which only 5% of audio frames will reach.

Neoran and Shashoua proposed a filterbank approach to loudness dependent normalization [9]. After creating a histogram from a sequence of loudness level estimates, they determine a single loudness number by taking the integral of the P% highest histogram levels, where P may be set to around 20%. The implicit assumption seems to be that the louder segments will more heavily influence human judgment of long-term loudness. (Note that we speak of judging the loudness, not the softness, of a song. A bias toward the level of the louder segments may be reflected in our language.)

Personal experience suggests that if we were to try to match the loudness of two songs having very different dynamic ranges, sole

reliance upon either the loudest or the average frame levels might result in a mismatch of perceived loudness. For example, in Figure 1, column 1 represents a signal with a wide dynamic range of 80 dB (from -80 to 0 dB), while the other columns have only a 20 dB range. Signals 1 and 2 may have the same average frame loudness, but the song represented by column 1 will probably be perceived as louder, since louder segments appear to have a stronger influence on our perception of overall loudness. On the other hand, signals 1 and 3 have the same maximum loudness, but the audio of column 3 may be perceived as louder because the signal is generally stronger than that represented by column 1.

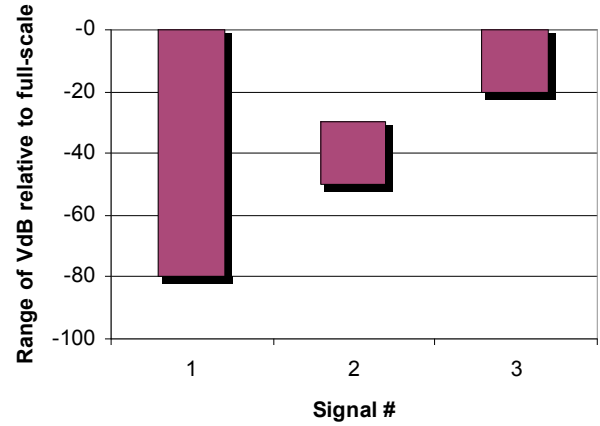


Figure 1. Long-term Loudness Matching by Mean vs. Maximum Level. Signal 2 attempts to match the overall loudness of signal 1 using the average frame level. Signal 3 tries to match signal 1 using the maximum frame level. Neither achieves an optimal perceptual match.

2 LONG-TERM LOUDNESS MATCHING LEVEL

We propose comparing the overall loudness of extended sound files using a measure we will refer to as the “long-term loudness matching level,” or LLML. The LLML defines a method for combining a series of individual (per-frame) level estimates, obtained using any of a variety of methods [5-9]. Our definition does not attempt to be compatible with standard definitions of loudness in sones or loudness level in phons; our primary interest is in creating a measure that is easily computed and manipulated and corresponds reasonably well with human judgments of the relative loudness of extended audio signals.

2.1 Desired Conditions

We would like our definition of the LLML to satisfy the following conditions:

1. The individual (per-frame) level estimates should reflect the fact that human hearing is more sensitive to certain frequencies than to others.
2. All non-silent frames should contribute to the value of the LLML.
3. Louder frames should optionally be weighted more heavily than softer frames, since the louder segments may more heavily influence our judgment of loudness.
4. A single emphasis parameter should control the amount of additional weighting given to louder frames.
5. If the optional loudness weighting of condition 3 is used, any number of additional silent frames should have no influence on the result.
6. The LLML of a sound file with a constant per-frame level should be that level.
7. The LLML of a full-scale square wave should be 0 dB.

8. Scaling the entire sound file by N dB should result in an N dB change in the LLML.

2.2 Hearing Curve Pre-emphasis

To satisfy condition 1, we need to take into account variations in the sensitivity of the ear at different frequencies. One efficient way of performing this frequency-weighting would be a simple pre-emphasis of the signal using an inverted equal-loudness curve; for example, the “B” weighting curve (see Figure 2) that applies to sound playback at moderate levels such as those encountered in home listening situations. This pre-emphasis has the additional advantage of making the result relatively insensitive to DC offsets.

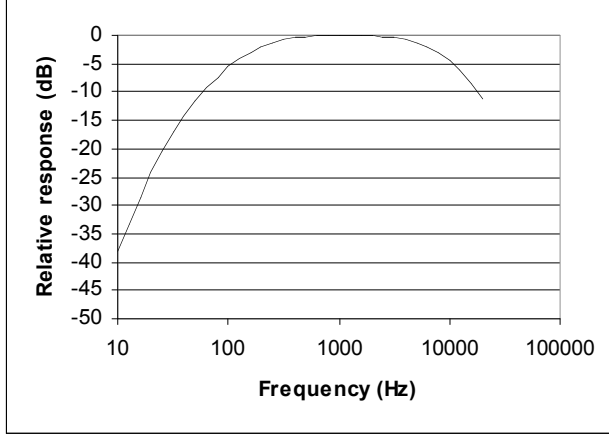


Figure 2. “B” frequency weighting curve.

The “B” curve pre-emphasis is only a rough approximation, not only because we may not know the actual playback level, but also because the equal loudness contours constitute a family of curves and cannot be duplicated using a single linear filter. Nevertheless, this approximation can be useful.

In situations where the computational power is quite limited, we could pre-filter the signal using a simplified approximation of the low end of the “B” weighting curve. Since most musical signals have much more energy in the bass and mid-range than in the high frequencies, the highs will tend to have minimal impact on the final loudness estimation. For example, for a 48 kHz signal, we could use a 200 Hz first-order Butterworth high-pass filter,

$$x_{filt}(n) = 0.9871x(n) - 0.9871x(n-1) \\ + 0.9742x_{filt}(n-1).$$

2.3 Level Extraction

We then extract a smoothed level from the pre-emphasized signal, using a level detector such as root-mean-square (RMS) [10]. The RMS level detector is preferred over the often used smoothed full-wave rectifier because it reduces the number of higher-order harmonics and the possibility of alias components folding back into the audible range [11, 12]. In a discrete-time system, the sub-sampled effective amplitude V_{rms} can be calculated using a running-average filter, as follows:

$$V_{rms}(i) = \sqrt{V_{ms}(Ni + N - 1)}, \text{ where}$$

$$V_{ms}(n) = cV_{ms}(n-1) + (1-c)x_{filt}^2(n), \\ c = e^{-\frac{1}{\tau F_s}}, \text{ and}$$

V_{ms} is a running average of the power of the equal-loudness-filtered signal, N is the number of samples per frame, τ is the RMS time constant (for example, 35 ms.), F_s is the sampling frequency and c is the smoothing coefficient.

To satisfy condition 7, V_{dB} is defined as a ratio to the RMS amplitude of a full-scale square wave (i.e., unity), converted to dB:

$$V_{dB}(i) = 20 \log_{10}(V_{rms}(i)).$$

The derivation of V_{dB} could be replaced with any reasonable method of obtaining individual per-frame level estimates. For example, the output of a smoothed Hilbert envelope [13] could be used in place of V_{rms} to further reduce unwanted ripple and higher-order harmonics.

2.4 Weighted Average Calculation

The LLML is obtained by taking a weighted average of the individual level estimates:

$$L = \sum_{i=0}^{M-1} w(i)V_{dB}(i),$$

where L is our long-term loudness matching level in dB and M is the number of frames in the file.

To satisfy conditions 2 through 5, we define the weighting function W as follows:

$$u(i) = k^{-V_{dB}(i)}, \quad 0 < k \leq 1 \\ w(i) = \frac{u(i)}{\sum_{j=0}^{M-1} u(j)}$$

Thus, the weight applied to each individual V_{dB} measurement is an emphasis parameter k raised to the negative $V_{dB}(i)$ power, normalized so that the sum of the weights is unity.

If $k = 1$, the LLML becomes a simple average of the individual V_{dB} measurements. As k approaches 0, the LLML approaches the level of the loudest single individual frame. At intermediate values, for example, $k = 0.85$, the LLML gives a somewhat greater emphasis to the louder frames, as desired. As long as $k < 1$, any number of additional silent frames will have no effect on the result (in compliance with condition 5), but all non-silent frames will be represented (as required by condition 2).

Preliminary tests have shown this definition of LLML to be useful for loudness matching of songs. As desired, scaling the entire sound file yields the expected overall change in dB. Additional listening tests may reveal the optimal value of k to best model the way humans judge loudness differences between extended sound recordings.

2.5 Gain Calculation

To perform loudness matching, we determine the amount of gain needed to convert the analyzed LLML to the target level. If no compression is applied, the desired gain is simply:

$$g = 10^{\frac{L_t - L_a}{20}},$$

where L_t is the target LLML and L_a is the analyzed LLML. If L_t is set to a high level, a peak limiter may be helpful to avoid the possibility of clipping. Alternatively, we can prevent clipping by constraining the gain such that

$$g \leq \frac{V_{max}}{V_{peak}},$$

where V_{max} is the full-scale amplitude and V_{peak} is the instantaneous peak amplitude of the signal.

3 COMPRESSION

While normalizers are used to adjust the overall loudness of an entire song or sound file, compressors are used to reduce the amount of loudness variation within a song.

Figure 3 shows a compressor block diagram [2], differing from the typical compressor only in the addition of an equal-loudness hearing curve filter. This filter is useful for ensuring that the compressor does not overreact to the beats of music containing heavy bass content, causing pumping of the midrange vocals or excessive attenuation of the bass.

Figure 4 depicts a typical compressor transfer function (or characteristic). The input signal level (along the x-axis) maps to an output level (along the y-axis). In the example shown, low signal levels will be unchanged due to the 45°-angle line segment, while signal levels above the breakpoint will be attenuated.

3.1 Three Problems with Traditional Compressors

Figure 4 also illustrates some problems inherent in compressors lacking advance knowledge of the signals they are about to process.

First, if an input signal has been normalized, a great deal of compression will be applied, whereas if the same signal is attenuated in advance of the compressor, it may receive no compression at all. We would like the dynamic range at the output of the compressor to be independent of whatever scaling may previously have been applied.

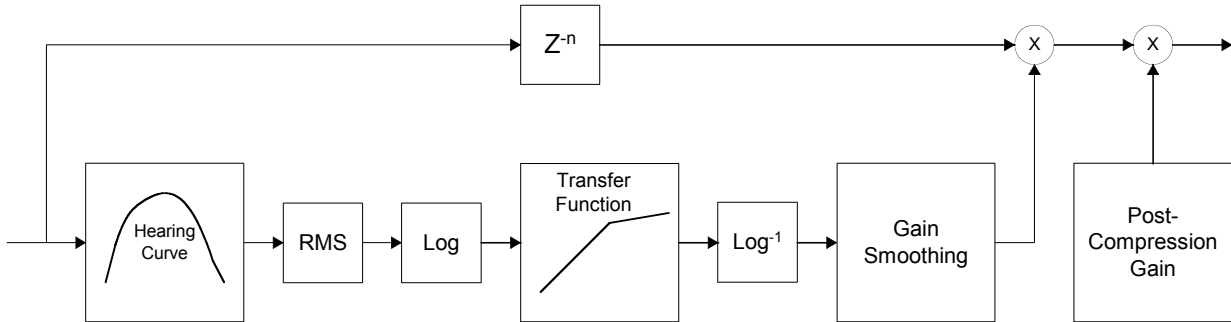


Figure 3. Compressor Block Diagram.

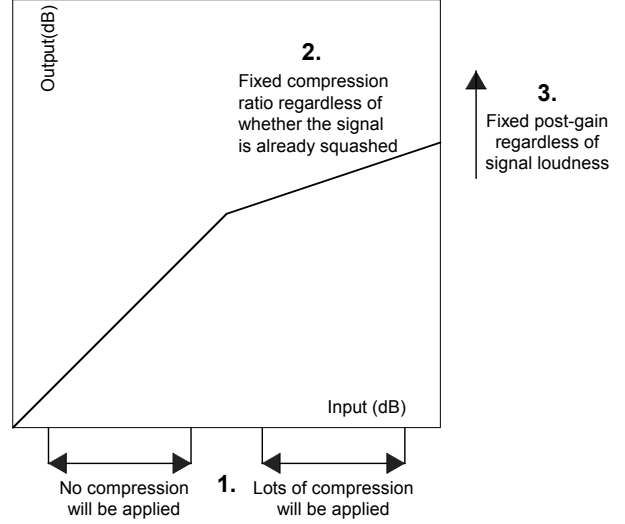


Figure 4. Compressor Transfer Function, illustrating three problem areas.

Secondly, without human intervention, traditional compressors use the same compression curve for each song, even if that song has already been squashed like a road-kill possum. This, of course, is because the compressor has no advance knowledge of the song's overall statistics.

Thirdly, traditional compressors use a fixed (i.e., often incorrect) post-gain in an attempt to compensate for the attenuation due to the transfer function. The correct post-gain depends not only on the compression curve, but also on the loudness of the input signal and the details of how its dynamics line up with the compression curve.

4 DYNAMIC SPREAD

Compressors are popularly said to reduce the dynamic range of an audio signal, though the term “dynamic range” is also used to refer to the difference between the peak signal level and the noise floor or minimum signal level.

Even if we calculate the dynamic range based on a per-frame V_{dB} evaluation (for example, the difference between the V_{dB} values of the loudest and softest frames) instead of per-sample, the result still tells us little about the distribution of dynamics. The literature is overdue for a term that better describes what compressors are

intended to reduce. In this paper, we will use the term “dynamic spread.”

Range is one measure of the spread of a data set and is defined by the distance between the largest and smallest measurements. Because it is based on only two measurements, range is not always the most useful or robust measure of spread.

4.1 Desired Conditions

We would like our definition of dynamic spread to satisfy the following conditions:

1. It should be unaffected by a simple gain scaling.
2. If we were to scale all the distances from the mean by the same amount (essentially, compressing or expanding the dynamics around a central loudness), the spread should be scaled by the same factor.
3. For robustness, the dynamic spread should be based on substantially all of the per-frame V_{dB} values.

One way to satisfy these conditions would be to derive the dynamic spread from the generalized deviation of the per-frame V_{dB} values:

$$d = \left(\frac{1}{M} \sum_{i=0}^{M-1} |V_{dB}(i) - \bar{V}|^p \right)^{\frac{1}{p}}$$

where d is the dynamic spread and $\bar{V}$ is either the mean or the median of V_{dB} . The power p provides control over the relative emphasis given to outliers.

While $\bar{V}$ represents the central per-frame level, the dynamic spread d relates to how closely the values are clustered about that center; i.e., how “squashed” the audio is, either naturally (as in the case of the solo bassoon, with a dynamic range of perhaps 10 dB), or as a result of earlier compression.

If $p = 2$ and $\bar{V}$ is the mean of V_{dB} , we obtain the standard deviation (also called the root-mean-square deviation). However, because this equation squares the distance from the mean, it tends to over-emphasize the extremes of the V_{dB} array.

Our preferred definition of dynamic spread sets $p = 1$, which yields the mean absolute deviation,

$$d = \frac{1}{M} \sum_{i=0}^{M-1} |V_{dB}(i) - \bar{V}|.$$

This simply computes the average distance from the central per-frame level. Because it doesn’t square the distance, this equation is less sensitive to outliers such as the level of the noise floor. We do not want a small amount of tape hiss to have a large effect on the dynamic spread.

The perceptual correlate to a signal’s dynamic spread could be referred to as its loudness spread; this is the quantity the compressor is ultimately intended to reduce.

4.2 V_{dB} Histogram

Both the LLML and the dynamic spread can be approximated from a V_{dB} histogram (or statistical frequency function), such as the ones illustrated in the first column of Figure 5. The advantage is a

large reduction in the amount of metadata required. If V_{dB} is quantized to, say, 1 dB increments, an array containing the number of frames at each useful dB level would only require about 100 values, regardless of the length of the song.

In Matlab [14], the code for the V_{dB} histogram algorithm would be as shown in Listing 1 (using positive array indices to represent negative dB histogram bins):

```
Hist=zeros(100,1); % Allocate array

for i=1:100 % for each frame
    Bin = -round(VdB(i));
    if Bin < 1, Bin = 1; end;
    if Bin > 100, Bin = 100; end;
    Hist[Bin] = Hist[Bin] + 1;
End
```

Listing 1. Creating V_{dB} histogram.

5 DYNAMIC SPREAD MATCHING

Dynamic spread matching is a way of ensuring that compression results in audio files with similar distributions of dynamics. We want uniformity of the results, not of the process.

With traditional compressors [1, 2], once the parameter settings are selected, the same compression curve is applied to every song regardless of its original dynamics and scaling. This, again, is because the compressor does not know the original dynamics distribution in advance. If we can perform a pre-analysis of the audio data, this will no longer be the case.

Instead of blindly applying the same amount of compression to every song, we can match each song to a desired dynamic spread. If a song is already heavily compressed, it would be ridiculous to compress it further. However, if the next song is, say, a piece of classical music with a wide dynamic range, we may want to apply a suitable amount of compression.

By calculating the original dynamic spread and comparing that to a desired dynamic spread, we can intelligently apply whatever compression is needed.

5.1 Using a Single Line Segment

The simplest way of achieving the desired result is to use a single line segment as our compressor’s transfer function, as shown in Figure 6, with a slope determined by

$$S = \frac{d_d}{d_a},$$

where S is the slope, d_a is the analyzed dynamic spread of our sound file and d_d is our desired dynamic spread. (Note that in some articles [1, 2], the term “slope” is used to refer to the negative slope of the gain curve, or $1-1/R$, where R is the compression ratio. In this paper, “slope” simply means the inverse of the compression ratio; in other words, the slope of the transfer function line segment.)

Slopes in the range $0 \leq S < 1$ produce compression. If the desired dynamic spread is wider than that of the original, we may prefer to leave the original as is, rather than to apply expansion. If so, S can be limited to a maximum value of 1.

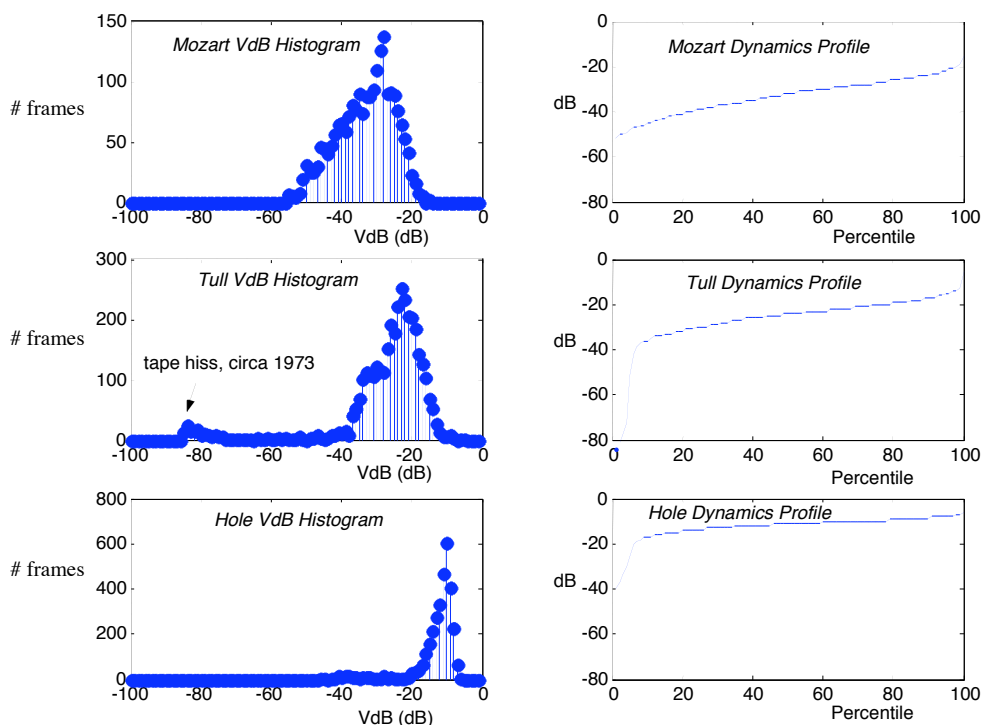


Figure 5. V_{dB} Histograms and Dynamics Profiles. Note how differently a 20 dB compressor threshold would affect the first and third recordings.

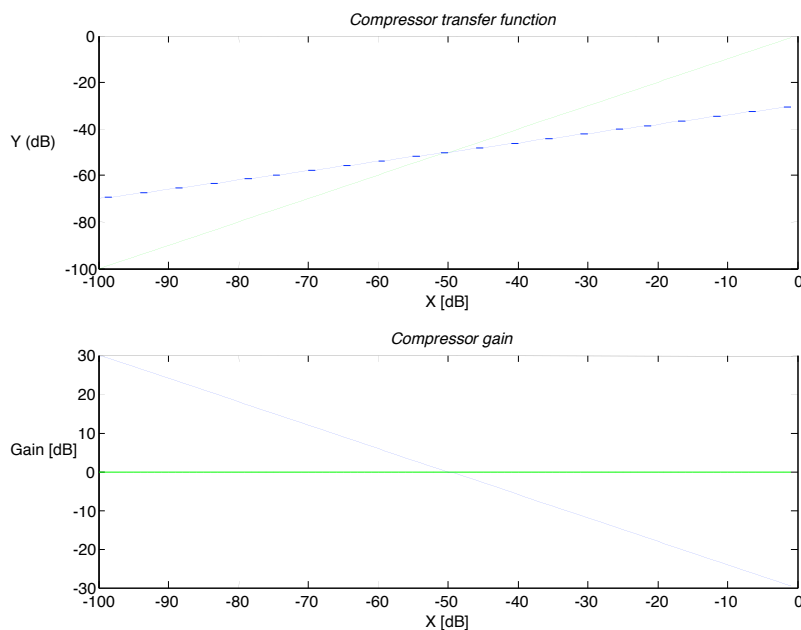


Figure 6. Compressor transfer function and gain, single line segment. Note that the compressor gain keeps increasing as the input signal level decreases.

While the use of a single line segment yields the desired dynamic spread, it has an unfortunate effect on the signal-to-noise ratio. As we see from Figure 6, as the input signal level decreases, the compressor gain keeps increasing. The result is a very noisy compressor, because we're applying a large amount of gain to those frames that already have the worst signal-to-noise ratio.

5.2 Using Multiple Line Segments

Therefore, we will probably want to use a traditional multi-segment compressor transfer function, which typically has a constant-gain region below the compressor threshold (as seen in Figure 4).

5.2.1 Specifying the Thresholds

If our characteristic uses two or more piecewise-linear segments, we will want to specify certain threshold levels in advance. For example, we will want to specify the location of the compressor threshold, and we may want an expander segment below a noise gate threshold to suppress low-level signals and minimize noise.

This poses a subtle yet serious problem: how can we specify a threshold so it will behave similarly with any sound file? We have seen that if we specify the threshold as an absolute position (for example, a certain number of dB below full-scale), the resulting transfer function will affect sound files differently depending on how they have been scaled. Since we want our compressor to achieve similar results regardless of scaling, this is clearly not the desired effect.

A common approach (in radio stations, for example) is to precede the compressor with an automatic gain control. This requires increased computational expense, and the additional layer of compression may increase distortion and compound transient problems such as overshoots. Another solution would be to perform a pre-compressor (x-axis) loudness normalization step, possibly including a limiter, in addition to our post-compressor (y-axis) normalization. This is cumbersome and inefficient.

A much better solution would be to specify the threshold locations in a signal-independent way and then translate those specifications into signal-dependent breakpoints.

For example, one might specify the thresholds in terms of multiples of the dynamic spread above or below the mean value, $\bar{V}_{dB}$. For example, we may want our noise gate threshold to be located at $\bar{V}_{dB} - 1.5d$, while our compressor threshold is at $\bar{V}_{dB} + 1.1d$. Unfortunately, if our dynamics distribution is significantly skewed (for example, by a large amount of tape hiss), the resulting breakpoint might end up being outside the range of the individual V_{dB} measurements.

A more robust method would be to specify the compressor threshold as a percentile. We can do this using the “dynamics profile.”

5.2.2 Dynamics Profile

The dynamics profile, illustrated on the right side of Figure 5, is a cumulative relative frequency plot with the x- and y-axes switched. This provides an overview of the statistical (but not temporal) distribution of dynamics within a song. For a given percentile value P on the x-axis, the dynamics profile gives us a dB value V on the y-axis, such that $P\%$ of the frames in the song are softer or equal in loudness to V .

The conceptually simplest way to view the dynamics profile is to sort the original V_{dB} array in order of increasing level, then relabel the x-axis to display a range from the 0th to the 100th percentile.

A more computationally efficient method of calculating the dynamics profile, not requiring a large sort operation, is to derive it directly from the V_{dB} histogram. In Matlab, this could be done as shown in Listing 2:

```
% Allocate arrays
dBVsPercent = zeros(100, 1);
relFreqVsPercent = zeros(100, 1);
```

```
indx = 101;          % Start at -100 dB
sum = 0;

for i=1:100          % for each percentile
    % Find the lowest dB level whose
    % relative frequency exceeds this
    % percentile.
    while sum < 0.01*i
        indx = indx - 1;
        if indx < 1
            indx = 1;
            break;
        end;
        sum = sum + relFreqVsdB(indx);
    end

    dBVsPercent(i) = -indx;
end
```

Listing 2. Deriving dynamics profile from the V_{dB} histogram.

5.2.3 Using the Dynamics Profile

Figure 5 shows the V_{dB} histograms and dynamics profiles for three different sound files and helps illustrate the problem with using a fixed compressor threshold.

If we were to apply the same compression settings to each of these sound recordings, using a fixed threshold at -20 dB, the compressor would have almost no effect on the first signal (which has the most need for compression of any of the three). The third signal, which has relatively little loudness variation to begin with, would receive a great deal of compression.

To solve this problem, we propose specifying the compressor threshold as a percentile on the dynamics profile. For example, we may want to put the noise gate threshold at the 5th percentile, while the compressor threshold is placed at the 60th percentile. This method guarantees that our breakpoints will never be outside the range of the per-frame V_{dB} data, while automatically adapting the breakpoint locations to the dynamics of the audio data, regardless of how the audio may have been scaled or compressed.

In Figure 7, we can see how our use of the percentile domain helps to normalize differences in loudness and dynamic spread between sound files. The V_{dB} histograms (frames vs. dB) and dynamics profiles (dB vs. percentile) from Figure 5 have been combined into a single plot of frames vs. percentile, with all three data sets superimposed. Note the greatly improved similarity between the histograms in Figure 7 compared to their counterparts in Figure 5. By specifying the thresholds as a percentile, we achieve substantial independence from song-to-song variations in scaling and dynamics.

5.2.4 Determining the Slope

Once we specify the x-axis breakpoint locations, the next step is to determine the line segment slopes needed to yield the desired dynamic spread. A brute-force method would be simply to choose a slope, perform the actual compression on the entire song, measure the resulting dynamic spread and adjust the slope as needed. This is inefficient.

A better method is to predict the statistical results of the compression process. Assume for the moment that our characteristic has two line segments: a stationary 45° segment below the threshold and a compressor segment whose slope is to be determined. Figure 8 illustrates how we can estimate the effect of an arbitrary compressor characteristic on the dynamics profile of an arbitrary sound file, simply by applying the static transfer

function directly to the dynamics profile to create a new dynamics profile. In Matlab, this is done as shown in Listing 3.

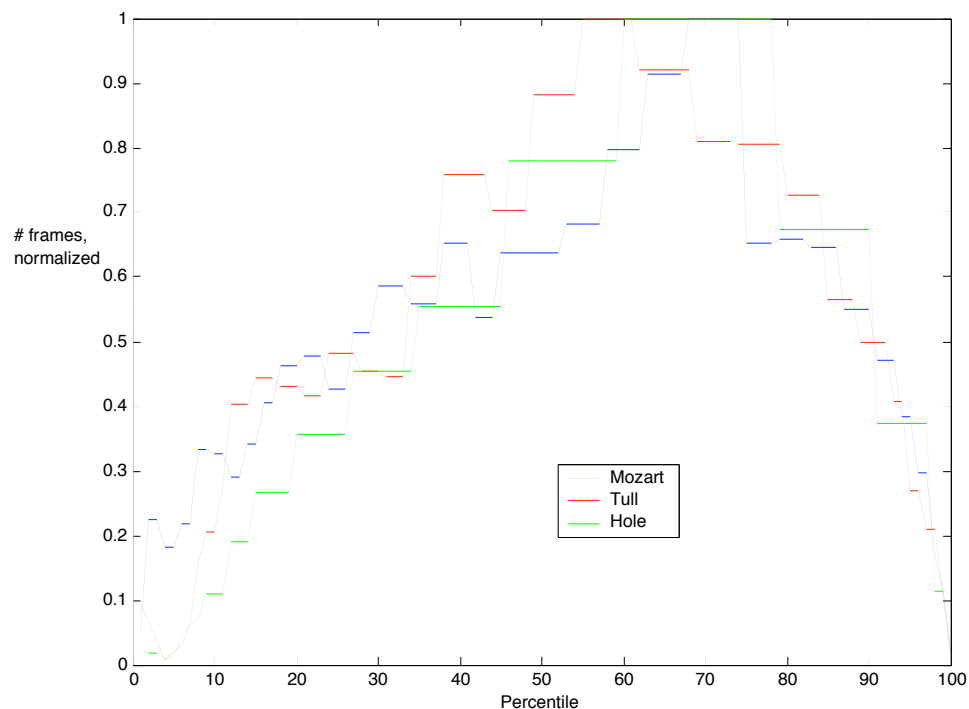


Figure 7. Normalizing histograms using percentiles.

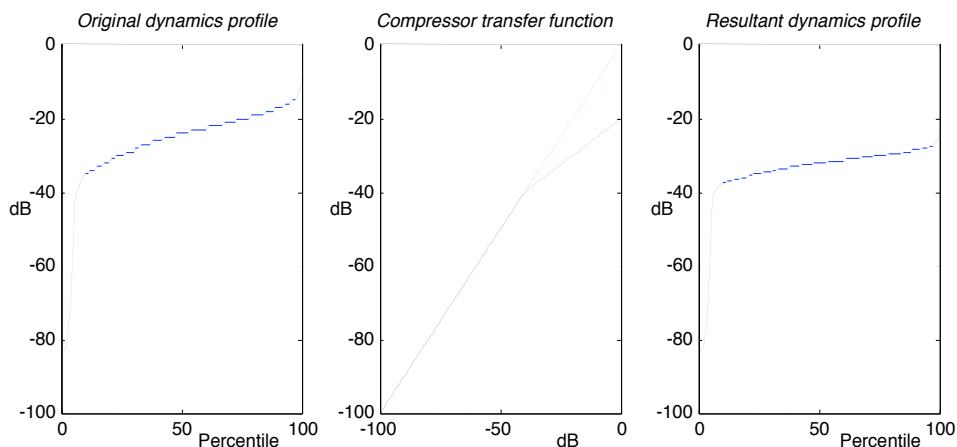


Figure 8. Application of the transfer function. The compressor's static transfer function is applied to the dynamics profile of the original sound file to yield the approximate dynamics profile of the resultant sound file.

```
% Allocate array
newRelFreqVsdB = zeros(100, 1);

% The indices of xferFcn represent
% input levels in negative dB; the
% array contents represent the
% corresponding output levels.

for i = 1:100      % for each orig. -dB
    newDBLevel = round(xferFcn(i));
```

```
newRelFreqVsdB(-newDBLevel) = ...
    newRelFreqVsdB(-newDBLevel) ...
    + relativeFreqVsdB(i);
end
```

Listing 3. Applying transfer function to dynamics profile.

If we initially set the slope of the compressor segment to 0° (horizontal), we can apply this static transfer function to the original dynamics profile and obtain an approximation of the dynamics profile that would result from this extreme compression (essentially, limiting). By analyzing the resulting dynamics profile, we can obtain an estimate of the dynamic spread that would be obtained if we were to perform the actual compression.

Next, we interpolate between this estimated dynamic spread and the dynamic spread of the original signal (which can be viewed as being compressed with unity slope; i.e., unchanged) to estimate the slope that will yield the desired dynamic spread. Assuming an approximately linear relationship between changes in slope and changes in dynamics spread, we find:

$$\frac{S_d - S_{\min}}{S_{\max} - S_{\min}} = \frac{d_d - d_{\min}}{d_{\max} - d_{\min}},$$

where S_d is the desired slope, $S_{\min}$ is the minimum slope (here, 0), $S_{\max}$ is the maximum slope (or 1), d_d is the desired dynamic spread, $d_{\max}$ is the original dynamic spread (at unity slope), and $d_{\min}$ is the dynamic spread obtained from applying compression with the compressor segment at minimum slope. (A minimum slope greater than zero might be desired in order to minimize detrimental sonic effects from extreme compression ratios.)

Solving for S_d , we obtain:

$$S_d = S_{\min} + (1 - S_{\min}) \left(\frac{d_d - d_{\min}}{d_{\max} - d_{\min}} \right).$$

If, after applying our new compressor curve to the original dynamics profile, the dynamic spread of the resulting dynamics profile is not sufficiently close to the desired value, we can iterate the interpolation process until we reach the desired precision. Similar processes can be devised in case there are additional line segments.

5.3 Temporal Behavior

The process of applying the static compressor curve directly to the dynamics profile does not take into account the compressor's temporal attack and release characteristics. Note that if our compressor were to use instantaneous attack and release times, relying solely upon the level detector for its smoothing, the estimated dynamics profile should match the actual result of the compression. Given sufficiently fast attack and release times (several hundred ms or less), the use of the static compressor curve in obtaining our dynamics profile estimate does not appear to cause significant skewing of the estimate over the course of an entire sound file. This might pose a larger problem for automatic gain controls, due to their slower time constants.

6 POST-COMPRESSOR LOUDNESS COMPENSATION

Dynamic compression changes the overall loudness of a sound file in a signal-dependent way. Traditional compressors try to compensate for their attenuation by applying a fixed post-compressor gain, but this is often too much or too little, depending on the song.

Without performing some sort of statistical analysis, we don't know in advance exactly how an arbitrary compression curve will affect the overall loudness of an arbitrary sound file, even if we know its original loudness, because the result depends on the exact distribution of the sound file's energy and how that lines up against the compression curve. If we want to apply loudness matching at the output of the compressor in real-time, we need a way of estimating the compressor's effect on a particular sound file.

We do this by again using the technique illustrated in Figure 8. By applying the compressor's static transfer function directly to the original dynamics profile, we obtain an estimate of the resulting dynamics profile. We then apply our LLML analysis process to the

new dynamics profile to predict the LLML of the song after the compressor. This in turn reveals the amount of post-gain needed for loudness matching.

The loudness normalization is calculated immediately before playback and then applied in the post-compression gain block shown in Figure 3. The equal-loudness filter used in the compressor should match the one used to generate the V_{dB} data during the song analysis phase.

7 SYSTEM OVERVIEW

A block diagram of the overall system, divided into song analysis, pre-playback, and playback phases, is shown in Figure 9. Figure 10 gives an overview of the song analysis (metadata generation) phase. A block diagram of the pre-playback (compressor parameter generation) phase is shown in Figure 11.

8 CONCLUSION

We have presented a method for normalizing the loudness of a sound file by comparing its long-term loudness matching level (LLML) to a desired target value. The LLML might provide a useful alternative to having a human operator attempt to match soundtrack levels for motion pictures; for example, it might provide an automatic way of generating the dialnorm metadata for normalizing dialogue levels in the AC-3 format [15].

The LLML might also be a useful basis for a standard to normalize the loudness of video games and other computer applications. Developers could measure the LLML of a nearly completed game and use that to set the overall loudness to an industry standard. An advantage of the LLML in this context is its ability to de-emphasize the weighting of periods of relative quiet.

In addition, we have presented a method for normalizing the dynamic spread of sound files, so that the desired compression is obtained without over-compressing audio that is already dynamically challenged. This technique could possibly be extended to incorporate other common compressor features, such as multi-band compression, etc.

Finally, we have shown how to determine the correct post-gain to match a compressor's output to a desired loudness, even though the compressed signal is not yet available, by estimating the compressor's effect on the dynamics of the original signal.

The techniques presented here make use of an analysis of the original audio data. This analysis phase could take place while audio data is being "ripped" from compact disks, during download from a network, or as a background process. Since we are performing a statistical analysis, we have found that it generally suffices to analyze 500 or fewer frames, somewhat randomly chosen, independent of the length of the sound file.

If the dynamics profile is quantized to one-percentile increments, the song analysis process results in a very small amount of data, on the order of a hundred bytes per sound file. This data could easily be stored as metadata on CDs or DVDs, as sidestream data in streaming audio formats, in playlist tables, etc. This metadata can be generated without human intervention and does not force the playback system to use pre-determined compressor breakpoints or time constants. It is our hope that creators of new audio formats and standards will give strong consideration to including such data as part of their format definitions.

9 ACKNOWLEDGEMENT

Dr. Jean-Marc Jot made valuable remarks and asked intriguing questions which prompted the current research. He also read earlier versions of the manuscript and offered useful suggestions.

10 PATENT NOTICE

Some of the methods described in this paper are the subject of a patent application.

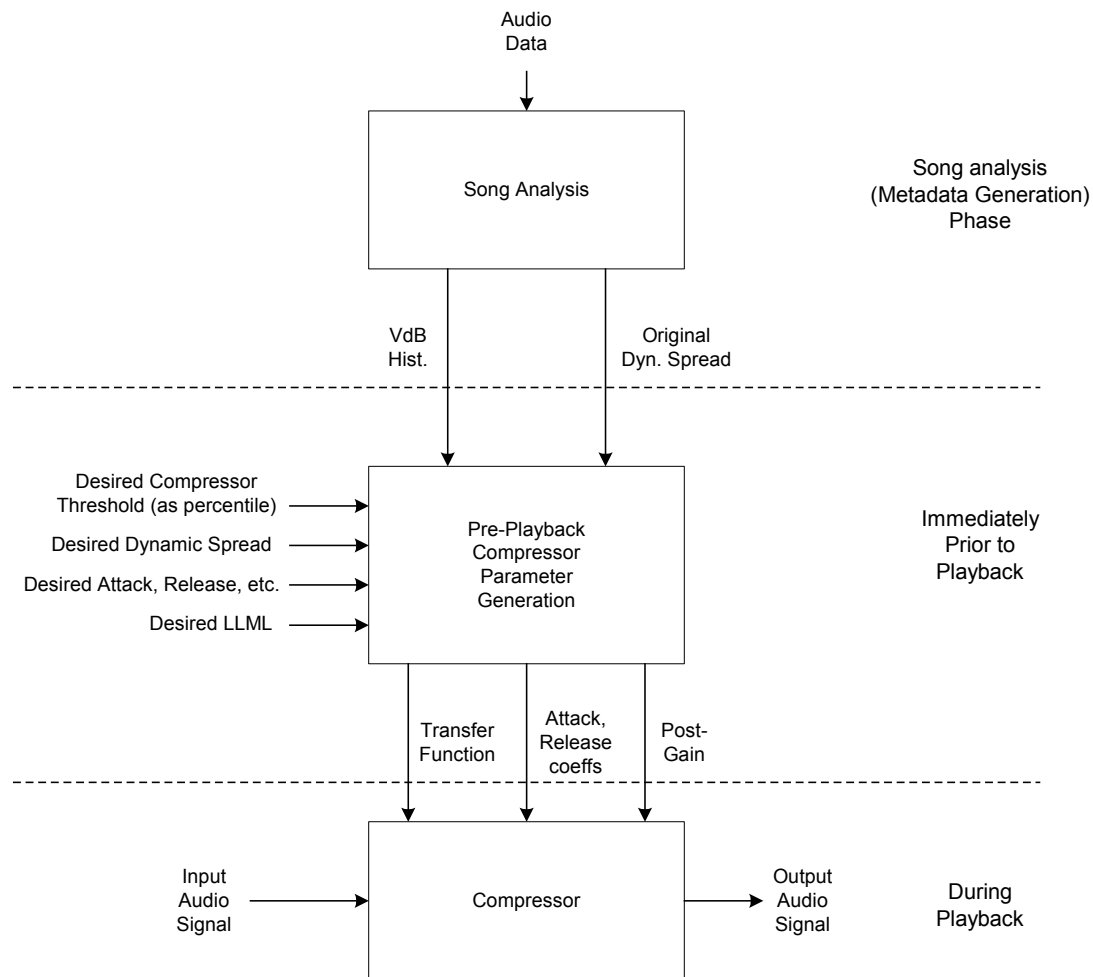


Figure 9. Block diagram of the overall system. The song analysis phase is shown in Figure 10, the parameter generation phase in Figure 11, and the compressor in Figure 3.

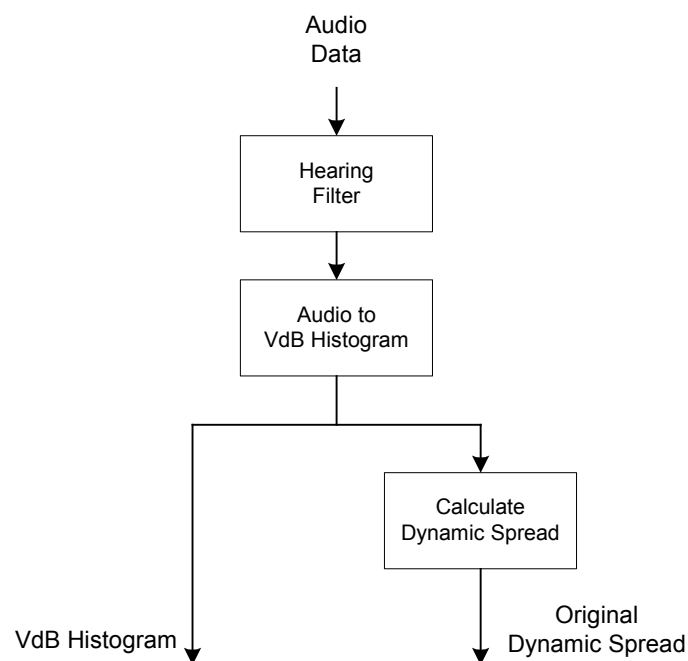


Figure 10. Block diagram of the Song Analysis (metadata generation) phase.

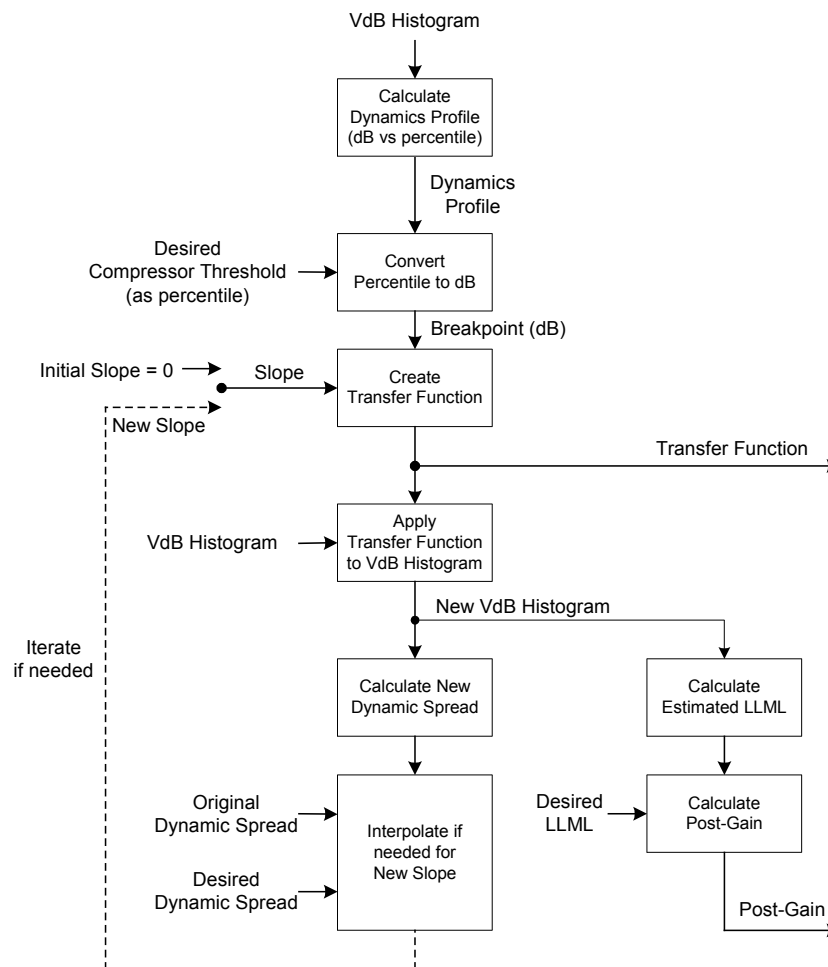


Figure 11. Block diagram of the Compressor Parameter Generation. This phase of processing typically occurs immediately prior to playback.

11 REFERENCES

- [1] G. W. McNally, "Dynamic Range Control of Digital Audio Signals," *J. Audio Eng. Soc.*, Vol. 32, No. 5, May 1984, pp. 316–327.
- [2] U. Zölzer, *Digital Audio Signal Processing*, John Wiley & Sons Ltd., 1997, pp. 207–219.
- [3] E. Zwicker, G. Flottorp, and S. S. Stevens, "Critical Band Width in Loudness Summation," *J. Acoust. Soc. Am.*, vol. 29, no. 5, pp. 548–557, 1957.
- [4] B. C. J. Moore, B. Glasberg, and T. Baer, "A Model for the Prediction of Thresholds, Loudness, and Partial Loudness," *J. Audio Eng. Soc.*, Vol. 45, No. 4, 1997, pp. 224–240.
- [5] B. C. J. Moore and B. R. Glasberg, "A Revision of Zwicker's Loudness Model," *Acoustica – Acta Acoustica*, vol. 82, 1996, pp. 335–345.
- [6] I. Allen, "Are Movies Too Loud?" *SMPTE Journal*, Vol. 107, p. 30, Jan. 1998, <<http://www.dolby.com/tech/tooloudp.pdf>>.
- [7] Sonic Foundry, Inc., Sound Forge (software).
- [8] Zwicker, E., Fastl, H., *Psychoacoustics*, Springer-Verlag 2nd Edition, 1999.
- [9] I. Neoran and M. Shashoua, "A Perceptive Loudness-Sensitive Leveler for Audio Broadcasting and Mastering," *105th Audio Eng. Soc. Convention*, Preprint No. 4852, 1998.
- [10] F. Floru, "Attack and Release Time Constants in RMS-Based Feedback Compressors," *J. Audio Eng. Soc.*, Vol. 47, No. 10, Oct. 1999, pp. 788–803.
- [11] A. Bateman, W. Yates, *Digital Signal Processing Design*, Computer Science Press, 1989, pp. 307–311.
- [12] P. Kraght, "Aliasing in Digital Clippers and Compressors," *J. Audio Eng. Soc.*, Vol. 48, No. 11, Nov. 2000, pp. 1060–1065.
- [13] P. Dutilleux, "Filters, Delays, Modulations and Demodulations: A Tutorial," *First COST-G6 Workshop on Digital Audio Effects (DAFX98)*, November 19–21, 1998
- [14] The MathWorks, Inc., *Matlab* (software).
- [15] Dolby Laboratories, Inc., "Dolby Digital Broadcast Implementation Guidelines," 1998, <<http://www.dolby.com/tech/ddbigsc.pdf>>.