Multiband Compression for Ambisonics

Toningenieur-Projekt

Markus Huber

Betreuung: DI Daniel Rudrich

Graz, May 12, 2020

institut für elektronische musik und akustik

Abstract

Multiband compressors are powerful and handy tools in the field of audio processing, and are frequently encountered within the mastering processing chain of audio productions. Yet up until now, no such tool existed in the world of the multichannel 3D-audio format Ambisonics. This may be due to the high number of channels, entailing rather tough computational demands. The goal of this work is to discuss the stages involved in designing a multiband compressor and to relate the findings to a multichannel setting. In particular, crossover filter designs, perfectly reconstructing linear-phase systems, compressor designs and the encompassing computational requirements are reviewed. Additionally, a VST Plug-in has been developed over the course of this project, which has been released as part of the *IEM Plug-in suite*.

Contents

1 Dynamic Range Compression

Dynamic range compression is one of the most widely used audio processing techniques and compressors are ubiquitously employed on arguably every kind of (at least commercial-grade) audio productions. In this introductory section, we want to explore the most common ways of controlling dynamic range, considering both single-band and frequency selective approaches, and point out when and why one would like to employ a particular tool in an audio production workflow.

1.1 Compressors and Limiters

While the amount of parameters as well as the adaptive and non-linear nature of dynamic processors introduce a certain degree of complexity to the topic, the notion of an "automated fader" describes the operation of a regular compressor quite well. In essence, the goal of compressing an audio signal is to reduce its dynamic range and thus bring its louder and quieter parts closer together, in order to obtain a more consistent and balanced sound. For instance, compressors can help a vocal recording, which naturally exhibits a wide dynamic range, to compete with its accompaniment by bringing the quieter parts up without overemphasizing the rest of the signal. Similarly, car entertainment systems for example may make use of a compressor to enhance only the lower volume parts of a whole mix during playback, which would otherwise be masked by background noise.

Although both of these examples indicate a boost of quiet signal components, compressors actually work by reducing peaks and high signal levels: if the input signal to a compressor exceeds a predetermined *threshold* level, compression kicks in and reduces it according to the *ratio* setting until the signal level falls below the threshold again. In order to maintain a smoothly transitioning signal level when the gain reduction sets in or out, the *attack* and *release* parameter control how fast the output should reach its target level after the input has crossed the threshold. In other words, these time constants determine how quickly the compressor should react. To compensate the volume level change, the so-called *make-up gain* increases the level of the compressed signal again. This way, quiet parts are raised higher compared to the louder signal components, in relative terms.

The widespread use of compressors stems from the fact that this type of dynamic control is a fairly typical requirement for basically all audio material - even more so when not just one but several audio tracks play in union. Hence, compressors are able to facilitate the creation of a coherent mix, which generally is the ultimate goal within the context of audio productions. Technically speaking, compressors in this sense help to adjust the crest factor of a certain signal to better match that of another one. The application of compressors is not limited to gain reduction though. Further objectives include creative effects (e.g. "pumping/breathing") as well as shaping the tonal character of a sound. However, it is this versatility of compressors that also makes them more prone to misuse.

A special form of the compressor is called limiter. While compressors gradually increase the amount of the applied gain reduction, limiters are designed to offer very high compression ratios (typically $\gtrapprox 12:1$) and are thus capable of protecting against distortion or clipping by preventing the signal to go over a certain threshold. Limiters are therefore most frequently utilized for mastering purposes, and often serve to boost the mix and squeeze out the loudest possible result, sometimes also in the form of "brick wall limiting" (with a ratio of $\infty:1$). For this reason, limiters typically also feature an input gain parameter (to drive the input against the threshold ceiling) as well as a look-ahead mode of operation (to guarantee to catch signal peaks before they cause distortion). Limiters also play an important role in broadcasting signals via frequency modulation (FM). Broadcasters have to ensure that the signals they transmit only occupy a certain bandwidth. As FM maps a signal and its amplitude onto the modulation of a carrier wave's frequency, with high amplitudes causing high frequency offsets, broadcasters can do so by limiting the signal's dynamic range. Another use case of limiters is to act as a safety net during recording or a live sound setting.

1.2 Multiband Compression

Although the common mode of operation acts upon the full frequency range at once, broadband signals often call for dynamics processing only within particular spectral regions. In a musical context, overhead microphone signals from a drum set recording might serve as a typical example - the individual sounds (i.e. cymbal vs kick-drum hits) are likely to be fairly imbalanced and compensation of these imbalances is not within the range of tasks standard single-band compressors are designed for. Even though the side-chain input of regular compressors can provide some sort of frequency selective compression - for example de-essing of sibilant consonants via a high-shelving boost of the side-chain input of a vocal recording - there is no getting around the fact that the compression will affect the entire frequency spectrum of the signal.

Multiband compressors allow to tackle problems that occur in both the time and frequency domain by separating the signal into multiple frequency ranges (*bands*) using a crossover filter network, before applying a dedicated compressor to each band and subsequently summing them up again, as illustrated in Figure 1.

Additional application examples of multiband compressors include bass instruments, which often require treatment in the low end whereas their higher frequency content may be left unchanged. Level inconsistencies in the low end might as well originate from a moving performer in combination with a microphone's proximity effect, and can be easily managed via a frequency selective processor. Disturbing guitar string noises or loud breaths may be reduced in a similar fashion, if the individual compressor section can be set such that it's only triggered by these occurrences (and the output/makeup gain doesn't compensate for the gain reduction).

The benefits of multiband compression are most commonly utilized by mastering engineers though, since it allows control and modification of specific parts in sum signals which would not be feasible otherwise. For example, if the low end has to be revised,

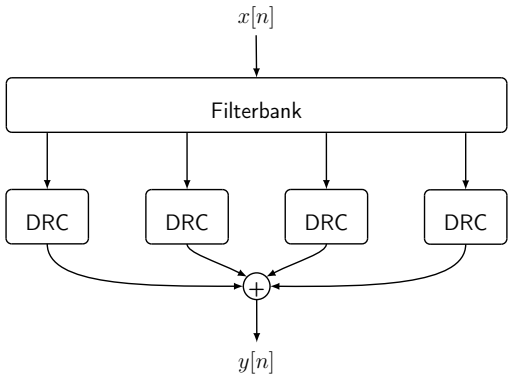


Figure 1: Very basic signal flow of a multiband compressor

regular compressors would cause the whole signal to "pump" on every threshold trigger, whereas multiband compressors eliminate this side effect and affect only the targeted frequency band individually. Similarly, the transient behaviour of individual spectral regions can still be tuned at a rather late stage by taking advantage of the individual compressors' time constants. Furthermore, multiband compressors enable the application of high overall gain reduction while maintaining an authentic sound, whereas full-band versions would typically yield unnatural outputs. To this regard, operating multiple compressors targeted on separate parts of the signal therefore also allows for squeezing out louder sounding results.

Although the notion of a multiband *compressor* suggests the possibility of using it as a *limiter*, this task is better left to its full-band counterpart. It's impracticable to do limiting with a multiband compressor, because it's hard to control the summed, maximum overall amplitude by operating on single bands in isolation.

Despite their powerful possibilities, some engineers have concerns about employing multiband compressors, since splitting up the signal into several bands tends to come with processing artefacts (also with thoroughly designed linear-phase designs, e.g. due to pre-ringing) and therefore also with the risk of losing fidelity. Naturally, a digital multiband compressor is heavier on the CPU compared to its full-band sibling, especially in multichannel scenarios.

Despite these reservations, multiband compression is a very useful and diverse tool, and complements the range of audio processing equipment very well.

2 Crossover filter design

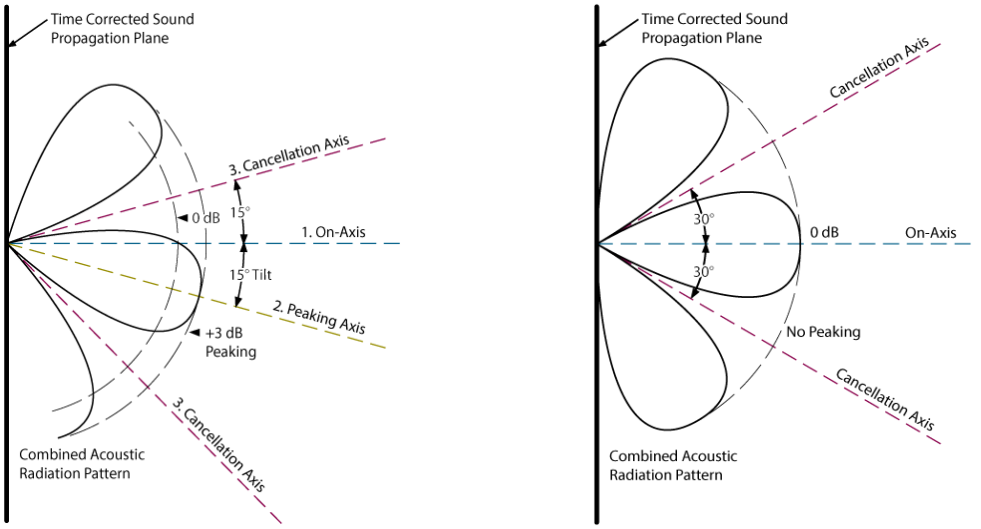
As stated above, multiband compressors split the signal into several bands before applying dynamic range compression to each band independently. This section deals with the design of appropriate filter networks for this task. In the context of multichannel audio it is crucial to bear in mind the design's computational impact, so generally IIR filters are preferred over FIR ones due to their efficiency. It is also desirable to keep processing artefacts as low as possible - the output of the multiband compressor ideally preserves the original signal, if no compression is triggered.

Digital filter design frequently adopts ideas from the analogue world (e.g. by making use of the bilinear transform), as analogue filter topologies have a long tradition, are well understood, and necessarily IIR. Although there are "native" digital filter banks relying on multirate processing techniques and analysis-synthesis stages which can offer perfect reconstruction (see e.g. [Smi11]), these typically require a filter prototype that is simply shifted in the frequency domain to separate the individual bands and are therefore less flexible to adjust. Another possible approach would be to apply the dynamic range compression entirely in the frequency domain, i.e. via a Short-Time Fourier Transform and a subsequent processing upon the frequency bins (see [MV17]), however this approach introduces additional latency due to the blockwise processing, and can lead to a whitening of the signal due to the high number of frequency bands.

2.1 Introduction to Linkwitz-Riley filters

Considerations concerning crossover filters for multiband compressors are closely connected to those employed in loudspeakers with multiple drivers, with each driver "processing" signals in its associated applicable frequency range and a subsequent "summation" of these signals in the acoustic domain. However, if the signal to be played back contains frequencies close to or at the crossover frequency, this summation of acoustic waves will inevitably cause constructive and destructive interferences, since drivers cannot be mounted coincidently (as shown in Figure 2). This has several implications on the design of (active) crossover filters: not only should they provide steep enough roll-off rates to feed sufficiently band-limited signals to the drivers, also the resulting summed frequency response should ideally be flat (on-axis) and the polar radiation pattern of the loudspeaker has to be acceptable as well. Moreover, the phase response (or rather the group delay) has to be considered.

Butterworth types are a popular choice for tackling this task - they exhibit a maximally flat passband and a $-3\,dB$ gain at the crossover frequency (see Figure 3). A 1st-order Butterworth filter ($-6\,dB/octave$) is considered inappropriate, as it provides too much out-of-band energy to the drivers. 2nd-order Butterworth crossovers ($-12\,dB/octave$) are $180°$ out of phase with each other at all frequencies. Flipping the polarity of one driver can remedy this problem (by convention typically the high-pass section is inverted), but this causes a $+3\,dB$ bump in the magnitude response at the crossover frequency. Though going to 3rd-order ($-18\,dB/octave$) appears to solve this problem because the

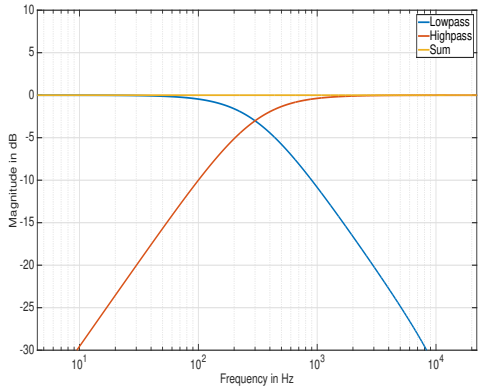


(a) 3rd-order Butterworth alignment *(b)* 4th-order Linkwitz-Riley alignment

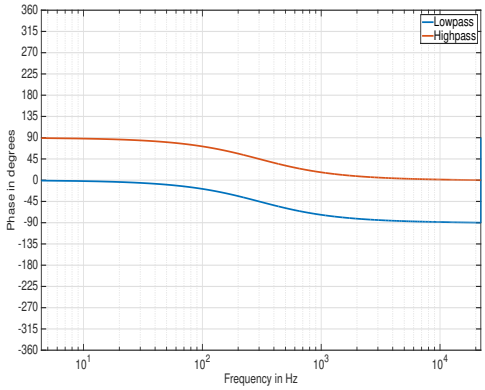
Figure 2: Polar radiation patterns of non-coincident drivers at the crossover frequency for different filter designs [ran]. Assuming the drivers are time aligned and mounted on top of each other along the vertical center, a 3rd-order Butterworth Allpass design (a) translates the crossover frequency signal correctly on-axis (midway between both drivers), but yields tilted lobes and an off-axis peak. While cancellations can not be avoided for non-coincident drivers, the in-phase relationships of the Linkwitz-Riley design (b) causes them to be symmetric about the on-axis plane and removes unexpected peaks and tilts of the lobes.

sections exhibit a phase shift of $270°$ to each other and therefore compensate the $+3\,dB$ bump, this introduces another problem since phase-shifts at the crossover frequency lead to bad off-axis polar response characteristics. The same kind of problems recur when higher orders are employed - each order increases the slope by $6\,dB/octave$ and the phase difference between the sections by $90°$.

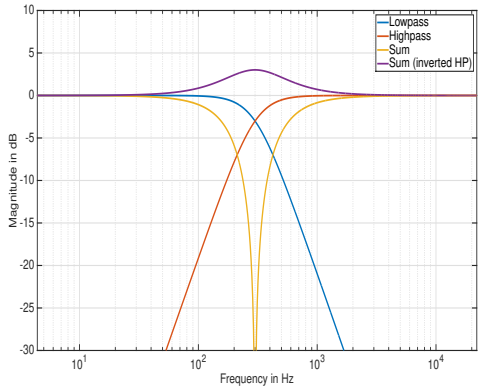
The *Linkwitz-Riley* design [Lin76] addresses these problems by cascading two Butterworth sections, yielding a $-6\,dB$ magnitude reduction at the crossover frequency and *in-phase* outputs of the high-pass and low-pass sections at *all* frequencies. As a result, the Linkwitz-Riley alignment shows an absolutely flat summed magnitude response, without introducing lobing errors or tilts in the loudspeaker's polar radiation pattern - it acts as an all-pass, only affecting the phase. As already stated, Linkwitz-Riley filters are constructed from a serial connection of Butterworth filters of the same order. The implication of this is that only even orders are possible for these designs - a n^{th}-order twofold Butterworth cascade yields a $2n^{th}$ order Linkwitz-Riley filter. Any chosen order complies with the previously described properties, though, for this to hold, the polarity of one crossover section has to be switched with every other order. Arguably the most common version is the 4^{th}-order Linkwitz-Riley filter, which is shown in Figure 4. Designs beyond 8^{th}-order exhibit increasing group delay non-linearity and are more costly and complex to implement (at least in the analogue domain) and therefore are considered ineligible.



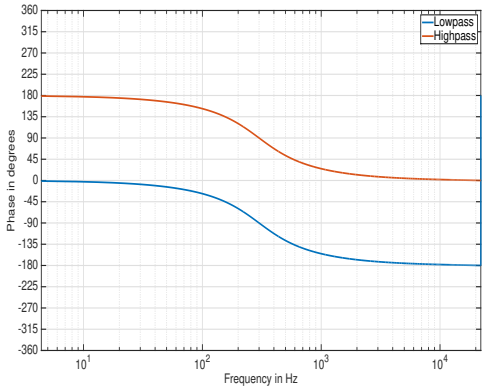
(a) 1st-order BW magnitude response ($-6\,dB/octave$)



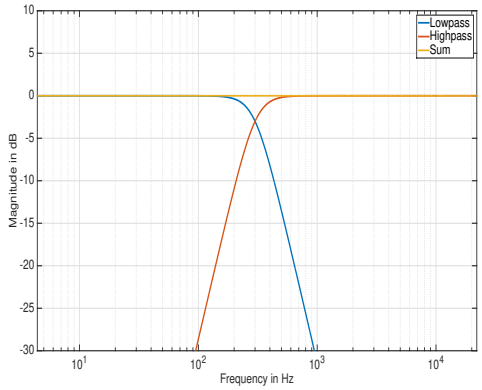
(b) 1st-order BW phase response



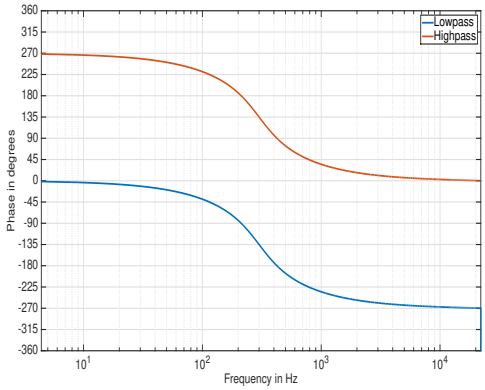
(c) 2nd-order BW magnitude response ($-12\,dB/octave$)



(d) 2nd-order BW phase response

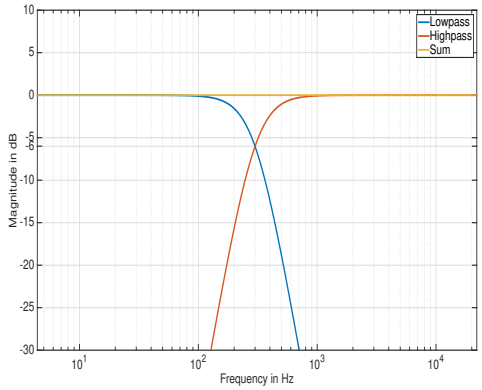


(e) 3rd-order BW magnitude response ($-18\,dB/octave$)

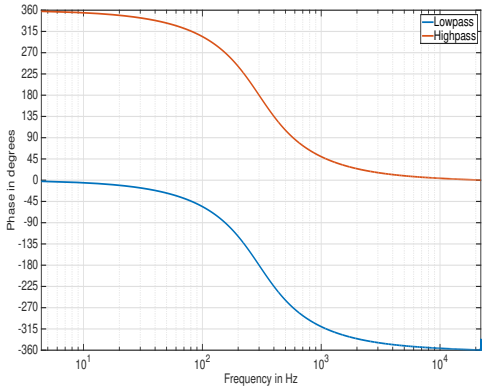


(f) 3rd-order BW phase response

Figure 3: Magnitude and phase responses for Butterworth alignments of different orders with a crossover frequency $f_c = 300\,Hz$.



(a) 4th-order LR magnitude response $(-24\,dB/octave)$



(b) 4th-order LR phase response

Figure 4: Magnitude and phase response for 4th-order Linkwitz-Riley alignment with a crossover frequency $f_c = 300\,Hz$.

Linkwitz-Riley filters offer ideal crossover behaviour, except for the non-constant group delay which will be discussed later on in section 2.4. The design of crossover filters for loudspeakers typically has to incorporate knowledge about the non-ideal behaviour of the loudspeaker components, so there is no go-to solution. This does not apply for the design of crossovers for digital multiband compressors though and Linkwitz-Riley filters, due to an absolutely flat magnitude response and in-phase outputs at all frequencies, seem very well suited to the task.

2.2 Calculation of Linkwitz-Riley filters

The Butterworth connection cascade that forms the Linkwitz-Riley filter is usually split into 2nd-order-sections, due to the favourable characteristics offered by Biquads (less risk of serious quantization errors / instability). The calculation of Butterworth filter coefficients in the digital domain is modelled on the analogue design and is usually derived by making use of the bilinear-transform (see appendix A). For a second-order transfer function of the form

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{1}$$

the digital filter coefficients of a 2nd-order Butterworth lowpass or highpass filter can be determined by the expressions presented in Table 1.

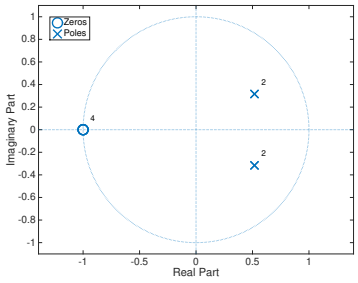
Note that both types (lowpass and highpass) share the same denominator - the poles of their transfer function are identical - and all zeros of lowpass and highpass are located at -1 and $+1$ in the unit circle, respectively. Cascading the Butterworth filters in order to yield Linkwitz-Riley behaviour doubles the order of the filter's corresponding poles and

Table 1: Coefficients for a 2nd order digital Butterworth filter, with $K = \tan(\frac{\pi f_c}{f_s})$. f_c and f_s denote the desired crossover and sampling frequency, respectively.

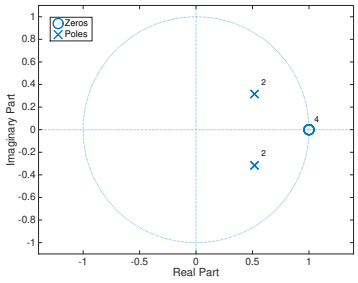
Type	b_0	b_1	b_2	a_1	a_2
Lowpass	$\frac{K^2}{1+\sqrt{2}K+K^2}$	$\frac{2K^2}{1+\sqrt{2}K+K^2}$	$\frac{K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$
Highpass	$\frac{1}{1+\sqrt{2}K+K^2}$	$\frac{-2}{1+\sqrt{2}K+K^2}$	$\frac{1}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$

Table 2: Coefficients for 2nd-order Allpass system equivalent to parallel connection of 4th-order Linkwitz-Riley Lowpass and Highpass filters. $K = \tan(\frac{\pi f_c}{f_s})$, with f_c and f_s denoting to cutoff and sampling frequency, respectively.

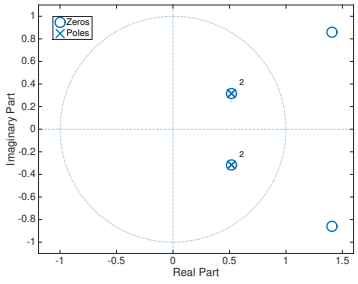
b0	*b1*	*b2*	*a1*	*a2*
$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	1	$\frac{2(K^2-1)}{1+\sqrt{2}K+K^2}$	$\frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}$



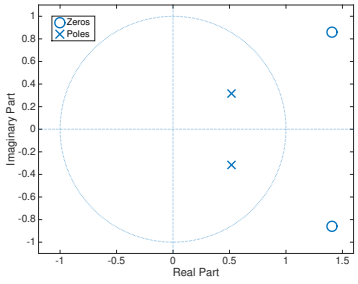
(a) 4th-order LR Lowpass



(b) 4th-order LR Highpass



(c) Resulting parallel 4th-order Allpass system



(d) Equivalent 2nd-order Allpass

Figure 5: Pole-zero plots for 4th-order Linkwitz-Riley filters (with $\omega_c = 2\pi \cdot \frac{5000\,Hz}{44100\,Hz} = = 0.7124$) and for the resulting crossover Allpass system after summation of the outputs.

zeros, but does not change the pole-zero locations. Upon summing the Linkwitz-Riley output sections, the poles still stay the same due to the equal denominator polynomials. However, the zeros shift and turn the overall system into an Allpass: half of the zeros cancel out the duplicate poles, reducing the system's order to that of the Butterworth filters it is composed of, whereas the other half moves to the reflections of the poles across the unit circle. Figure 5 depicts this pole-zero analysis for the 4$^{\text{th}}$-order Linkwitz-Riley alignment. The same kind of Allpass behaviour can be observed for arbitrary orders, with the difference being the number and the location of poles and zeros.

We can therefore describe the crossover system formed by the $2n^{\text{th}}$-order Linkwitz-Riley filters by simply constructing an equivalent Allpass of order n. The denominator polynomial is the same as that of the original Butterworth filter transfer functions and the coefficients of the nominator are obtained by reversing those of the denominator, placing the zeros at $z_{AP} = \frac{1}{p^*_{BW\_low}} = \frac{1}{p^*_{BW\_high}} = \frac{1}{p^*_{AP}}$ with * denoting the complex conjugate.

2.3 Extending to multiple bands

Higher numbers of individually processable frequency bands within a multiband compressor allow for more flexibility and increase its usefulness for the production of audio content. In general, additional bands can be obtained by running the output of one crossover through subsequent ones, resulting in a tree-like network structure. All crossovers have to be aligned so that their filters operate within the bounds of the frequency range of its respective input signal. Considering this, one could attempt to split an input signal $x[n]$ into 3 bands according to the structure depicted in Figure 6. In this visualization, the indexed LR-blocks denote Linkwitz-Riley Lowpass or Highpass filters and the resulting band-limited signals are summed up again without any processing in between.

Although the individual filter pairs show Allpass behaviour, the overall system response does not preserve the Allpass property. The transfer function of this system reads as follows

$$\begin{aligned} H(z) &= LR_{1,L}(z) \cdot (LR_{2,L}(z) + LR_{2,H}(z)) + LR_{1,H}(z) \\ &= LR_{1,L}(z) \cdot AP_2(z) + LR_{1,H}(z) \end{aligned} \tag{2}$$

with $AP_2(z)$ denoting an Allpass equivalent to the LR_2 system. The phase shift introduced to the low-band path by the second crossover disrupts the Linkwitz-Riley property of in-phase relationships between outputs at all frequencies, which in turn causes destructive interferences around the crossover frequencies - in this case, around the crossover frequency of the LR_1 filter pair. Note that the impact of these interferences are dependent on the ratio of the crossover frequencies, $\frac{f_{c_1}}{f_{c_2}}$, of the two independent LR-Allpass systems to each other, as well as the order of the chosen Linkwitz-Riley alignment. With $f_{c_1} \approx f_{c_2}$ and 4$^{\text{th}}$-order filters, the composite magnitude will exhibit a strong notch around f_{c_1}, since system LR_2 will shift the phase approximately 180° relative to LR_1

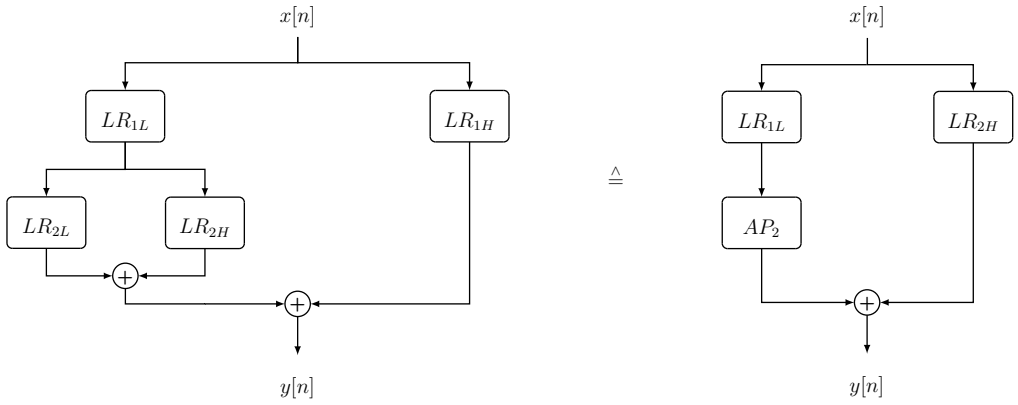


Figure 6: 3-band filterbank network structure and an equivalent structure that replaces one parallel Linkwitz-Riley system by a 2nd-order Allpass. Note that the resulting overall system will exhibit out-of-phase relationships and undesirable interferences, since the Allpass affects the phase of only one and not both signal paths. If the crossover frequency of the two Linkwitz-Riley systems are close to each other, these interferences become particularly severe.

and only one half of the overall system is affected by this change (see Figures 4 and 8). If $f_{c_1} \gg f_{c_2}$ (or $f_{c_1} \ll f_{c_2}$, depending on the desired signal path), then the deviation will turn out less significant, since the introduced phase shift will approach 360° (or 0°, respectively) for 4th-order LR systems.

Inserting an Allpass equivalent to the LR_2 system in the filterbank's upper-band path resolves this problem. We can now write the total resulting transfer function as

$$\begin{aligned} H(z) &= LR_{1,L}(z) \cdot (LR_{2,L}(z) + LR_{2,H}(z)) + LR_{1,H}(z) \cdot \mathbf{AP_2(z)} \\ &= (LR_{1,L}(z) + LR_{1,H}(z)) \cdot AP_2(z) \\ &= AP_1(z) \cdot AP_2(z) \end{aligned} \tag{3}$$

Figure 7 illustrates this solution.

Following this intuition, we can extend the filterbank to multiple bands, by applying Allpass filters that compensate for phase shifts introduced by crossovers of different branches of the filter tree to all outputs. In the 4-band case as seen in Figure 9, the transfer functions $H_1(z)$ - $H_4(z)$ for each individual band (indexed from from lowest to highest according to the frequency range) would then correspond to:

$$\begin{aligned} H_1(z) &= LR_{1,L}(z) \cdot LR_{2,L}(z) \cdot AP_3(z) \\ H_2(z) &= LR_{1,L}(z) \cdot LR_{2,H}(z) \cdot AP_3(z) \\ H_3(z) &= LR_{1,H}(z) \cdot LR_{3,L}(z) \cdot AP_2(z) \\ H_4(z) &= LR_{1,H}(z) \cdot LR_{3,H}(z) \cdot AP_2(z) \end{aligned}$$

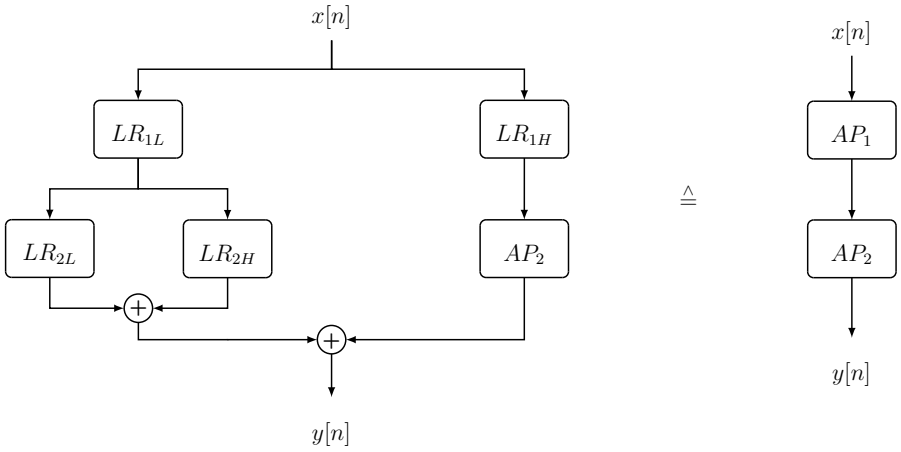
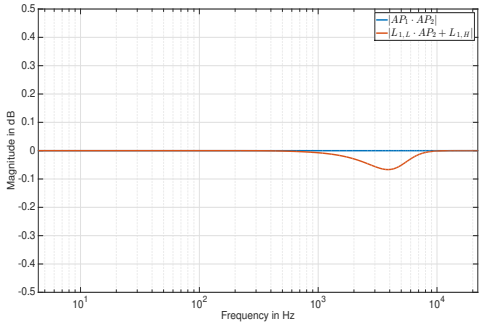
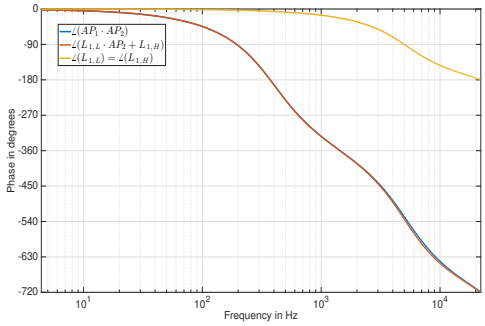


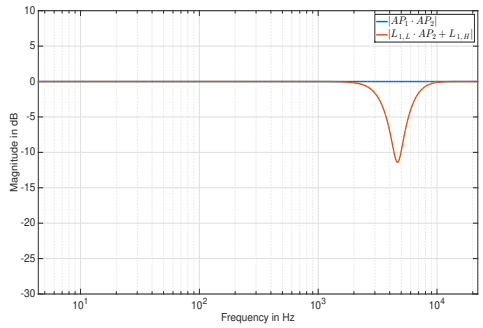
Figure 7: 3-band filterbank with compensating Allpass in the upper-band path and the equivalent description of the overall system as a serial Allpass connection.



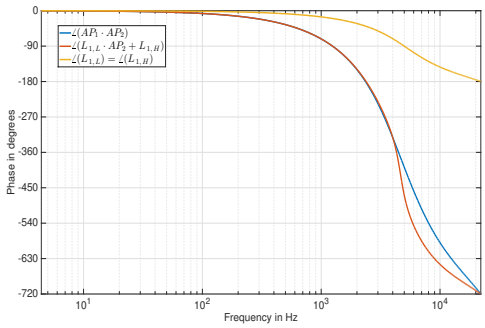
(a) Magnitude response,
$f_{c_1} = 5\,kHz, f_{c_2} = 400\,Hz$



(b) Phase response,
$f_{c_1} = 5\,kHz, f_{c_2} = 400\,Hz$

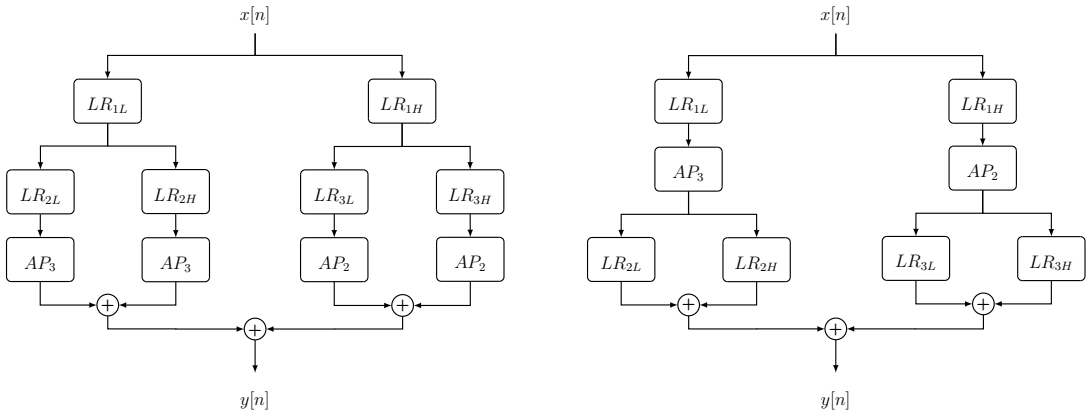


(c) Magnitude response,
$f_{c_1} = 5\,kHz, f_{c_2} = 4\,kHz$



(d) Phase response,
$f_{c_1} = 5\,kHz, f_{c_2} = 4\,kHz$

Figure 8: Total magnitude and phase responses for a 3-band filterbank system with 4^{th}-order Linkwitz-Riley filters, without a compensating Allpass filter. Note that the system exhibits a strong notch if the crossover frequencies are close to each other, due to the uncompensated phase shifts.



(a) *(b)*

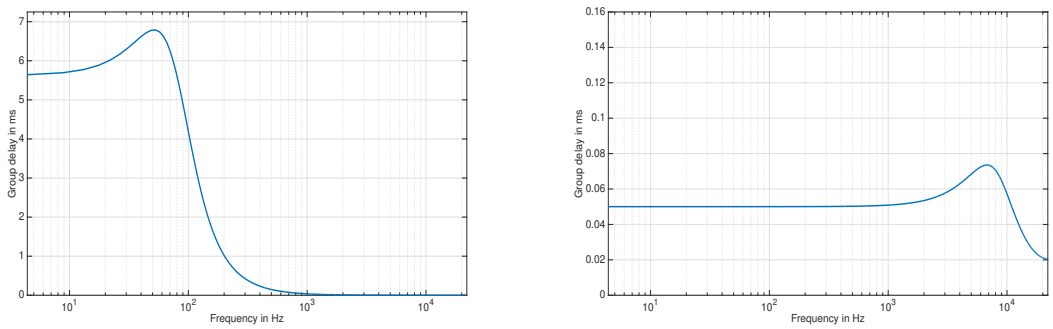
Figure 9: Equivalent 4-band filterbank structures with compensating Allpasses to ensure in-phase relationships of the resulting overall system. Since the Allpasses of structure (a) are inserted in both paths (high- and lowpass) of the corresponding Linkwitz-Riley (LR) system, we can push the Allpasses upwards and insert it just before the LR filters, resulting in the equivalent but computationally more efficient structure shown in (b).

If the same compensating Allpass terms apply to bands generated by the same 2-band complementary crossover, the Allpasses can be pushed up the tree, resulting in less filters, easing the computational requirements of the multiband compressor (see Figure 9b). A formalization of this method can be found in [FF10].

2.4 Dealing with phase distortion

The use of Linkwitz-Riley filters provides in-phase relationships in the passbands, yet their phase response is not linear and the group delay non-constant. This degrades the transparency of the overall system. Figure 10 shows the group delay displacement for Linkwitz-Riley Allpass systems. The distorting effect is more prominent at low crossover frequencies and drops with increasing f_c. The human auditory system is undoubtedly capable of perceiving phase distortions (see e.g. [LPV82], [Koy00]), however, the effects are found to be very subtle and audibility heavily depends on many factors, including mainly the nature of the source signal, the acoustic listening environment, and the extent of the phase distortion itself. Moderate phase deviation in high-complexity, real-world signals (like music) played back via an immersive, multichannel sound system like Ambisonics, will be practically inaudible. Nevertheless, keeping the phase response of the overall system flat is a meaningful goal, if the costs to reach this goal do not outweigh the benefits.

Despite considerable efforts to tackle this issue, phase compensation remains a problem difficult to solve, if real-time processing scenarios are of the concern. Even with the Allpass, that describes the phase shift introduced by the system, at hand (as in our



(a) $f_c = 80\,Hz$ *(b)* $f_c = 8000\,Hz$

Figure 10: Group delay in milliseconds for 4$^{\text{th}}$-order Linkwitz-Riley filters at different f_c

case), reversing its effect is not a trivial task. Due to their IIR nature (and the zeros lying outside the unit circle, see Fig. 5c), simple inversion of the Allpass filters, and thus yielding systems with an inverse phase response capable of compensating the phase shift perfectly, is impossible, as the systems obtained in this manner can in no way be stable and causal. Approximating a stable and causal IIR Allpass with inverse phase relationships is possible, but involve expensive optimisations. This is impractical for real-time applications, because the crossover frequencies of the multiband compressor should be easily adjustable on the fly. Furthermore, even if the used optimization schemes are very fast or the crossover frequencies are predefined, the required filter order will likely be too high for utilization in a multi-channel context. Needless to say, the employment of phase compensating FIR filters becomes virtually impossible as well, since the needed filter order will be considerably higher in this case.

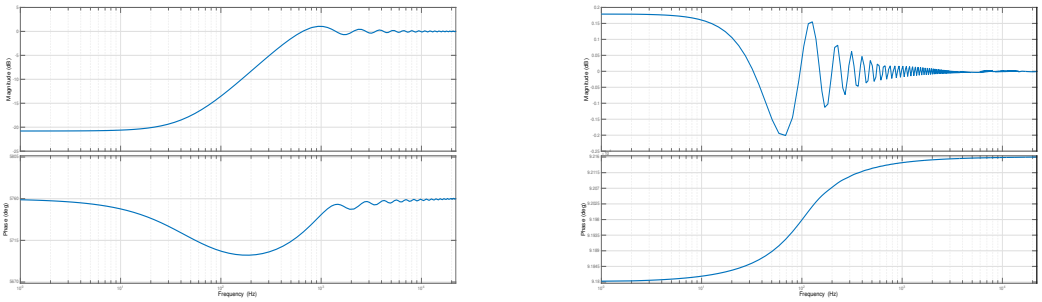
An alternative approach to mimic linear-phase IIR systems, is to filter twice - after a standard forward pass, the intermediate signal is time-reversed and run through the filter again, which compensates the filter's phase shift and group delay distortions. With regard to the double Butterworth sections that constitute Linkwitz-Riley filters, this seems convenient and works well for offline scenarios, however, the method becomes much less usable if real-time processing is considered. Applying this scheme in an online fashion has been proposed in [PC91] and various extensions to this approach exist (e.g. [DPL98], [AB07]). But the block-wise processing of the successively time-reversed signal and the reset of the filter states at every step inevitably induces corruptions of the magnitude response (alongside a fixed amount of processing delay).

Although utilizing linear-phase filters in the first place would avoid the whole issue entirely, they often pose new problems. If the phase response of the filter shall be linear, its impulse response must be symmetric. This has two major implications. First, the system has to be FIR, as symmetry is not a feasible requirement for one-sided (causal) infinite series. FIR filters typically need much higher filter orders to describe systems comparable to IIR ones, hence increasing the computational load and decreasing their applicability accordingly, which becomes especially critical if the rigorous demands of multichannel environments have to be met (e.g. 7$^{\text{th}}$-order Ambisonics has 64 channels per band). Secondly, since the axis of symmetry will usually carry the most prominently

weighted impulse, symmetric impulse responses frequently exhibit pre-ringing artefacts - audible side effects, particularly striking with transient sounds.

Reversing the phase distortions introduced by an IIR filter can also be addressed with an FIR Allpass that approximates the inverse phase response. Note that, apart from trivial perfect delays, constant-magnitude Allpass behaviour can only ever be approximated by FIRs. [Vic16] describes an implementation using a similar approach in order to attain linear phase systems. A simple intuition for the design of such an FIR is to truncate the original infinite impulse response, flipping it in time (rendering it acausal) and adding a sufficiently long bulk delay. However, to approximate FIR Allpasses, sufficiently high orders are needed, which becomes especially problematic if low crossover frequencies have to be captured, as illustrated by Fig. 11. Filter lengths of at least 512 samples are recommended, if the magnitude as well as the phase response are to be modelled appropriately. Apart from the immense computational requirements within a multichannel application, these filters introduce latencies that scale with the filter order, and the resulting delay will therefore be intolerably high. The time-domain filtering offers another point of view regarding the FIR Allpass approximation. The overall system can be described by convolving the impulse response of the filterbank network with the coefficients of the compensation filter, which is a time reversed version of this impulse response. Due to the time-reversal, this correspond to a cross-correlation. In the ideal case (with infinitely long response times), it can be seen as autocorrelation, yielding a symmetric (and therefore linear-phase) system, which basically acts as an ideal delay. By making the FIR longer, we can approach this ideal case. Note that pre-ringing will be less of an issue with Allpass FIRs, since the overall system approximates a perfect delay and we can expect a spike surrounded by very small values in its neighbourhood.

Given a described group delay, the filter might as well be constructed by integrating over the group delay and applying an inverse Fourier transform to a complex exponential constructed with the resulting phase response. Again, the applicability of such schemes to multichannel settings is limited by the computational requirements of FIRs, even if techniques like fast convolution are leveraged.



(a) $N = 31$ *(b)* $N = 511$

Figure 11: Magnitude responses of FIR Allpasses of different orders, approximating a 2nd-order Allpass with $f_c = 100\,Hz$, designed using a truncated, time-reversed IIR impulse response.

3 Compressor design

Among the standard audio effects (like equalization, panning, reverb, ...), dynamic range compression is often considered the most complex, in terms of usage as well as realization, probably due to the fact that compressors are both adaptive (depending on the input) and nonlinear processors. Also, the number of parameters is comparatively high, including threshold, ratio, knee, attack, release, make-up gain, optionally look-ahead and hold. Moreover, at least for the time constants, there's often a lack of clear definition and it's not agreed upon what the parameters exactly do. Many design choices have to be made, shaping the behaviour and sound, which can differ significantly between different compressors. This section, which is basically a summary of the analysis conducted in [GMR12], gives an overview of the main building blocks of digital compressors. In general, a compressor reduces a signal's dynamic range, with the degree of compression depending on the level of the so-called side-chain signal. Fig.12 shows a very basic compressor layout.

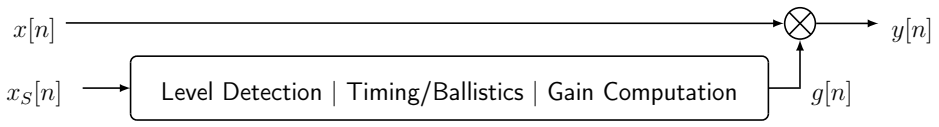


Figure 12: A basic feedforward compressor layout. $x_S[n]$ denotes the side-chain input, which is usually just a copy of the actual input signal $x[n]$.

3.1 Level detection

The gain computation depends on the estimated level of the side-chain signal. There are two basic methods to calculate this level: RMS-detection and peak-detection.

RMS-Detection. The root-mean-square measurement represents a smoothed average over the signal. It is also closely related to the perceived loudness of the signal, which is a desirable property of level detectors. However, it's calculation, given by

$$y_L[n] = \sqrt{\frac{1}{N} \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} x_S^2[n-m]}, \tag{4}$$

introduces an additional parameter N, as well as a latency of $N/2$ samples. In real-time implementations (as in many analogue RMS-based compressors) the RMS-estimate is therefore often approximated by exponentially smoothing the estimate via "leaky integration", a special case of a first order IIR lowpass filter

$$y_L[n] = \sqrt{\alpha\, y_L^2[n-1] + (1-\alpha)\, x_S^2[n]} \tag{5}$$

with α ($0 \leq \alpha < 1$) denoting a smoothing coefficient. We will see that this first order difference equation bears resemblance to smoothed level estimates using peak detectors. In fact, several studies suggest that the general behaviour for peak- and RMS-detection is the same, and differences only correspond to a scaling of smoothing coefficients / time-constants. For these reasons, the focus is on peak-detectors.

Peak-Detection. In the digital domain, a simple and intuitive estimate of the current level is the full-wave rectified signal value, corresponding mathematically to the absolute value function:

$$y_L[n] = |x_S[n]| \tag{6}$$

In order to avoid serious jumps in the output signal, the calculated gains are required to transition gradually across time. Continuous-time peak detectors of analogue compressors yield smoother estimates and are based on relatively simple *RC*-circuits, which is also how the attack and release time-constants are introduced to dynamic range controllers (with $\tau = RC$).

Time-constants. We can enforce the smoothing condition by filtering the estimates of the level detector, e.g. using an exponential moving-average filter (one-pole lowpass filter)

$$y[n] = \alpha y[n-1] + (1-\alpha)x[n] \tag{7}$$

with a step response of

$$y[n] = 1 - \alpha^n \quad \text{for } x[n] = 1,\, n \geq 1. \tag{8}$$

A time-constant τ is defined as the time it takes for a system to reach $1 - \frac{1}{e} \approx 63\%$ of its final value. With $y[\tau f_s] = 1 - \frac{1}{e}$ and the given step response, we can calculate the filter coefficient α via

$$\alpha = e^{\frac{-1}{\tau f_s}} \tag{9}$$

where f_s represents the sampling frequency.

As already stated, the time-constants originate from analogue RC components, and standard digital designs are derived from analogue ones (see [GMR12] for details on the derivation), all resulting in variants of this first-order filtering. Directly deriving from a simple analogue peak-detector consisting of a diode, two resistors (attack in serial and release in parallel), and a parallel capacitor, and assuming ideal components, is problematic, as this level detection scheme only gives correct estimates if the release time-constant τ_R is significantly higher than the attack parameter τ_A. Additionally, τ_A is slightly scaled by the release time, resulting in faster attack-times than expected when using a short τ_R.

Solving this problem in the analogue domain yields the so-called *decoupled* peak detector circuit, which, transferred to the digital world, writes as

$$y_1[n] = max(x_T[n],\, \alpha_R\, y_1[n-1]) \tag{10}$$
$$y_T[n] = \alpha_A\, y_T[n-1] + (1-\alpha_A)\, y_1[n] \tag{11}$$

where x_T and y_T respectively stand for input and output of the smoothing filter (also called *timing* or *ballistics* section). The parameters α_A and α_R denote the attack and release coefficients, respectively, and are calculated using relationship 9. However, as can be seen from the given equations, the release envelope is now conditioned on the attack envelope. A good estimate of the resulting effective release-time is $\tau_{R,eff} \approx \tau_A + \tau_R$.

Solving the problem solely in a digital environment, we can simply introduce a conditional operator, which fixes these problems:

$$y_T[n] = \begin{cases} \alpha_A\, y_T[n-1] + (1-\alpha_A)\, x_T[n] & x_T[n] > y_T[n-1] \\ \alpha_R\, y_T[n-1] & x_T[n] \leq y_T[n-1] \end{cases} \tag{12}$$

This *branching* detector not only corrects for the flawed level estimation, also the attack and release times now take on the intended values.

Nevertheless, both variants still suffer from an incorrect behaviour concerning the release phase, which is inherited from the analogue design. The full release-time is imposed upon the signal only if this signal experiences a drop to zero after a peak. If it settles on an intermediate value, the release envelope will be cut off at this point, resulting in a much shorter release-time. However, we can adjust both filter variants to compensate this behaviour and to always use the full release-time. The *smooth* decoupled peak detector can then be written as

$$y_1[n] = max(x_T[n],\, \alpha_R\, y_1[n-1] + (1-\alpha_R)\, x_T[n]) \tag{13}$$
$$y_T[n] = \alpha_A\, y_T[n-1] + (1-\alpha_A)\, y_1[n] \tag{14}$$

and the *smooth* branching detector as

$$y_T[n] = \begin{cases} \alpha_A\, y_T[n-1] + (1-\alpha_A)\, x_T[n] & x_T[n] > y_T[n-1] \\ \alpha_R\, y_T[n-1] + (1-\alpha_R)\, x_T[n] & x_T[n] \leq y_T[n-1] \end{cases} \tag{15}$$

The difference between these smooth and non-smooth filter versions can also be assessed via a total harmonic distortion (THD) measurement. Since the release envelope of the non-smooth versions ends abruptly, the THD can be significantly reduced by making use of the full release envelope via these smoothing modifications. Furthermore, the decoupled detectors slightly outperforms the branching ones in terms of THD, since the branching introduces a discontinuity, and therefore distortion, at the release phase inset. Also the so-called effective compression ratio is higher for the decoupled variants, as the area under their release envelope is slightly larger. As already discussed, the trade-off are incorrect effective release-times.

3.2 Gain computation

In an analogue compressor, a VCA (voltage controlled amplifiers) attenuates the input signals to the compressor proportionally to an external control voltage coming from the side-chain. This control signal is generated by the gain computer according to the static compression characteristic defined by the threshold *T*, the ratio *R* and the knee width *W* (see Fig.13). When the input exceeds a threshold level, the gain is reduced corresponding to the ratio of input to output. Sometimes the inverse ratio is given instead - the *slope* (ratio of output to input). If *W*, given in dB, is greater than zero, the characteristic is said to have a soft knee (rather than a hard knee), which smooths the transition between the two operation modes at the threshold point via interpolation by "distributing" the knee width equally to both sides (should be at least 2nd-order interpolation). Consequently, the relationship between the input level of x_G and the target level y_G (both in dB, since the parameters are usually given in dB as well) is described by the following equations:

$$y_G = \begin{cases} x_G & (x_G - T) < -\frac{W}{2} \\ x_G + \frac{(\frac{1}{R}-1)\,(x_G - T + \frac{W}{2})^2}{2W} & |(x_G - T)| \leq \frac{W}{2} \\ T + \frac{x_G - T}{R} & (x_G - T) > \frac{W}{2} \end{cases} \tag{16}$$

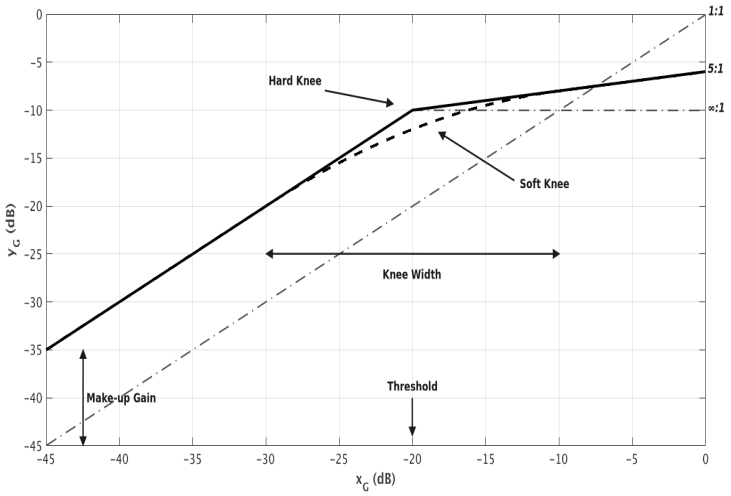


Figure 13: An exemplary compressor characteristic, with a threshold of $-20\,dB$, a ratio of $5:1$, a soft knee width of $20\,dB$ and a make-up gain of $+10\,dB$.

3.3 Compressor topologies

There are two basic system designs: feedforward and feedback topologies. Fig.12 depicts the basic feedforward case, with the gain $g[n]$ being applied after the calculations in the side-chain path. If the side-chain input $x_S[n]$ should equal the input signal $x[n]$,

then it's possible to reverse this relationship, so that the gain is applied first and the resulting signal is fed back to the side-chain input. This approach had been originally adopted in early analogue compressors, since the side-chain processing circuit has to be accurate over only a limited dynamic range compared to the forward path, minimizing possible inaccuracies in the gain stage. For digital implementations though, this aspect is irrelevant. Note that the backwards topology is not compatible with look-ahead designs and does not allow operation as a limiter either. Transferred to the dB domain, with $y_{dB} = g_{dB} + x_{dB}$, and using equation 16, the feedforward hard-knee gain calculation for the case $x_G > T$ can be formulated as

$$g_{dB} = (\frac{1}{R} - 1)\,(x_{dB} - T) \tag{17}$$

whereas the gain for the feedback topology follows

$$g_{dB} = (1 - R)\,(y_{dB} - T). \tag{18}$$

This implicates, that feedback topologies need to attain an impractical infinite negative amplification to act as a perfect limiter ($R = \infty : 1$), while this is poses no problem for the feedforward case, which instead even enables to apply over-compression. There exists also an alternative feedback topology, that relies on the assumption that the previous gain value will be close to the actual gain value, which allows to have an external side-chain signal in combination with a feedback design. However, most modern compressors are based on the feedforward approach, since they are more stable and predictable.

3.4 Timing placement

The sound of the compressor is also significantly influenced by the arrangement of the building blocks. Especially the placement of the ballistics smoothing filter has to be considered. The inputs and outputs of the level, timing and gain computation stage are labelled $x_L \rightarrow y_L$, $x_T \rightarrow y_T$ and $x_G \rightarrow y_G$, respectively. The make-up gain in dB is denoted by the character M.

A straightforward way to implement the side-chain processor is to do the level detection and timing in the linear domain and applying the compression characteristic in the log-domain.

$$\begin{aligned} x_T[n] &= y_L[n] \\ x_G[n] &= 20\,\log(y_T[n]) \\ g_{dB}[n] &= y_G[n] - x_G[n] + M \end{aligned} \tag{19}$$

The timing section operates on the full dynamic range and needs some time to charge up to the threshold level, as well as to fade out the signal beneath the threshold. However, the gain computer will be zero once the level is found to be below the threshold level. This creates a lag in the attack trajectory and a discontinuity in the release envelope.

In order to tackle this problem and turn the system back into a return-to-threshold type, we can instead smooth a signal which is biased at the threshold level:

$$\begin{aligned} x_T[n] &= y_L[n] - 10^{\frac{T}{20}} \\ x_G[n] &= 20\log(y_T[n] + 10^{\frac{T}{20}}) \\ g_{dB}[n] &= y_G[n] - x_G[n] + M \end{aligned} \tag{20}$$

On the downside, this causes troubles if soft knee characteristics are required.

An alternative placement of the ballistics in the linear domain, is to put it after the gain stage. This way, the control signal itself gets smoothed, yielding a return-to-threshold type without depending on a fixed threshold.

$$\begin{aligned} x_G[n] &= 20\log\left(y_L[n]\right) \\ x_T[n] &= 10^{\frac{y_G[n] - x_G[n] + M}{20}} \\ g[n] &= y_T[n] \end{aligned} \tag{21}$$

Placed in the linear domain, the smoothing filter lets the envelope decay exponentially. This produces a corresponding linear decrease in the log domain - a constant release rate of decibels per time. This implies that release times will differ for different degrees of compression. However, human perception in general tends to happen logarithmically, as is the case for the human ear. Thus, placing the timing section in the log domain creates a more subtle and smoother compression effect, and makes the envelope trajectory independent of the actual amount of compression.

$$\begin{aligned} x_G[n] &= 20\log(y_L[n]) \\ x_T[n] &= x_G[n] - y_G[n] \\ g_{dB}[n] &= M - y_T[n] \end{aligned} \tag{22}$$

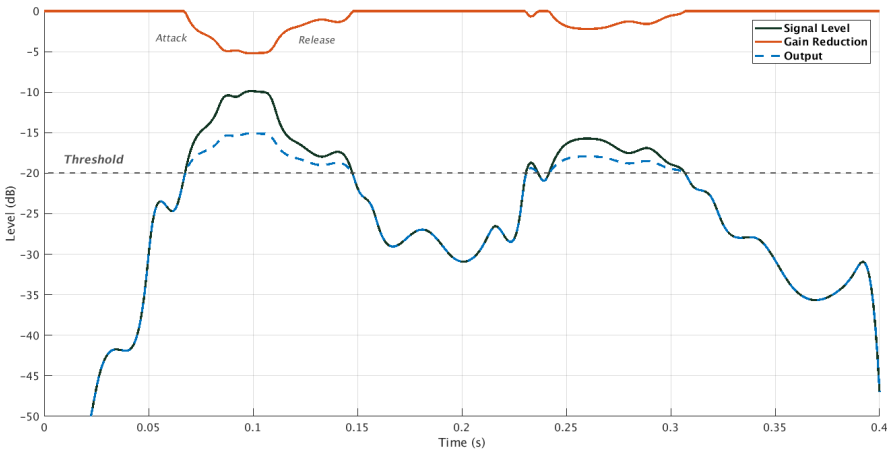


Figure 14: Visualization of a compressor's behaviour including the final gain reduction (ballistics already applied). The threshold is set to $-20\,dB$, the ratio to $2:1$, no make-up gain is applied.

4 Implementation

The main objective of this work was to develop a *VST* plug-in, that enables to do multiband compression for Ambsionic productions (up to 7th-order). This plug-in has been released under the IEM plug-in suite[1], which is publicly available under the GNU General Public License v3.0, and has been created with the JUCE[2] framework, which abstracts away basically all communication to the host and allows convenient development of multi-platform tools. This section reviews some of the design choices and implementation details.

4.1 Utilizing SIMD

Splitting the signal into, in this case, 4 bands, is achieved by running it through 4th-order Linkwitz-Riley filters and 2nd-order Allpass filters (as discussed in sections 2.2 and 2.3). The used 4th-order filters give a reasonable trade-off between computational complexity and roll-off rate, are frequently encountered in crossover networks, and can be easily implemented using a cascade of 2nd-order Butterworth IIR filters. The DSP module of JUCE eases the implementation of these filtering operations and provides handy classes for this purpose, facilitating the Transposed Direct-Form II structure.

However, upon investigating the filterbank topology of Fig.9b, one can anticipate heavy computational requirements if it is to be incorporated within a multichannel setting: successful separation of a 7th-order Ambisonic source signal, entailing 64 channels, with six 4th-order Linkwitz-Riley sections (= twelve 2nd-order Butterworth sections) and two 2nd-order Allpass filters per channel requires $64 \cdot (12 + 2) = 896$ 2nd-order IIRs. This corresponds to $(5{+}4){\cdot}896 = 17920$ multiply and addition operations per sample, ignoring the computational load of the subsequent dynamics processing and signal summation of the resulting $64 \cdot 4 = 256$ signals. A naive implementation would thus cause a serious computational bottleneck.

Fortunately though, since all filter operations on the individual channels equal each other, we can optimize this code by exploiting data parallelism mechanisms of hardware processors via *SIMD* instructions (Single Instruction - Multiple Data), which are supported by most CPUs nowadays. In contrast to multi-core processing which allows to do very different tasks at the same time, SIMD is a parallel computing method that facilitates vector operations by executing the same instruction over multiple data. Therefore, utilizing this technique enables to significantly speed up digital signal processing tasks. For example, the cost of element-wise addition of 8 floats to 8 other floats is similar to the cost of adding 2 floats. There are of course limitations to this method, e.g. a code full of control flow statements may hinder effective vectorization since the lock-step style processing prevents any inter-synchronization. However, as the filtering process is coherent for all channels, SIMD can be effectively leveraged for our purposes. So instead of fetching operands and executing multiply and addition operations of the filter bank

[1] `https://plugins.iem.at/`
[2] https://juce.com/

sequentially for all audio channels, we can direct the processor to load samples of different channels and process them simultaneously, with both instructions done in parallel by distributing data among several compute nodes.

But how can the CPU be instructed to utilize SIMD and vectorize the relevant operations? One possibility is to rely on the compiler to auto-vectorize the code. If certain optimization flags are set, compilers will try to recognize certain patterns and estimate the benefits and costs of vectorizing these. The compiler must prove that the vectorized code has the same semantics as the scalar code and it doesn't always succeed in doing so, even if the code is perfectly vectorizable for a human being. While there are ways to help the compiler finding this proof (e.g. sticking to conventional loop constructs or using certain keywords/intrinsics), it remains a compiler-specific black art and the generated code will be virtually unportable. Additionally, in order to make sure the code has eventually been optimized, the compiler report or resulting assembly code has to be checked. Therefore, explicit vectorization (e.g. via pragmas or SIMD-type libraries) is usually preferred - the programmer gains full control and the code can be made microarchitecture-agnostic.

Conveniently, JUCE offers a templated wrapper class `SIMDRegister<Type>` around a type, which is to be used with the target platform's native SIMD register, regardless of the specific processor and SIMD instruction set. All primitive types (as well as complex types) and most operations on these types are supported, and the processor classes of the DSP module are also able to operate on this wrapper. In our case, the signal samples and the filter coefficients should be declared to be of this wrapped type, if we want to utilize SIMD. We can query if the target machine's SIMD architecture with a macro provided by JUCE and get the size of the CPU's SIMD register too, like this:

```
#if JUCE_USE_SIMD
   using filterOperandType = SIMDRegister<float>;
   static constexpr int filterRegisterSize = SIMDRegister<float>::size();
#else
   using filterOperandType = float;
   static constexpr int filterRegisterSize = 1;
#endif
```

Load and store operations to the SIMD registers are most efficient if the respective data is aligned to the vector register boundaries. We can ensure this by interleaving the samples of the channels before filtering, putting one sample of `filterRegisterSize` successive channels next to each other into the SIMD register at once. If the number of channels is not a multiple of the `filterRegisterSize`, then the remaining slots in the interleaved channel vector are simply filled with zeros. After filtering and applying the dynamic range compression, the samples have to be deinterleaved in a similar fashion. Fig.15 depicts this procedure. The following code snippet demonstrates how this can be achieved within the JUCE framework. Note that we need $\lceil \frac{numChannels}{filterRegisterSize} \rceil$ distinct filter banks for this to work, with each filter object holding the states of $filterRegisterSize$ different channels (filter coefficients however are the same). Table 3 displays the significant increase in performance, when utilizing SIMD, with the plug-in being in 7^{th}-order mode.

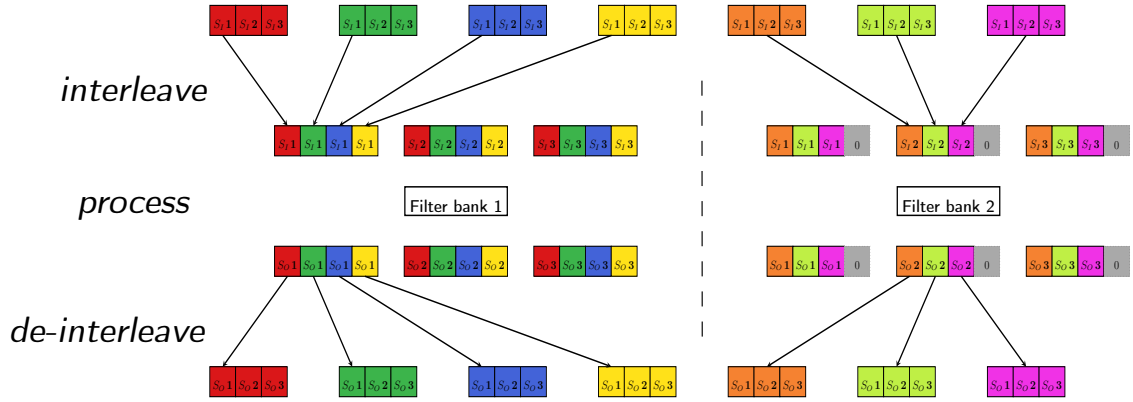


Figure 15: Illustration of the data interleaving procedure used to efficiently utilize SIMD. The first line depicts the input buffer consisting of 7 channels highlighted with different colors, each of which contains 3 samples. These are then interleaved to blocks of 4 samples, which corresponds to the size of the SIMD register (assuming SSE instruction set and single-precision floats). Since the number of channels is not a multiple of the SIMD register size, half of the SIMD registers (right side) have to be zero padded. After running them through the filter banks (which have the same filter coefficients but different states), the samples are de-interleaved accordingly and written to the output buffer.

```
// Interleave
const int numSimdFilters = 1 + (numChannels - 1) / filterRegisterSize;
int partial = numChannels % filterRegisterSize;

if (partial == 0)
{
   for (int i = 0; i < numSimdFilters; ++i)
   {
      interleaved[i]->clear();
      juce::AudioDataConverters::interleaveSamples (
         buffer.getArrayOfReadPointers() + i*filterRegisterSize,
         reinterpret_cast<float*> (interleaved[i]->getChannelPointer (0)),
         L, filterRegisterSize);
   }
}
else
{
   // - interleave as before until numSimdFilters-1
   // - interleave the remaining channels with zeros
   ...
}
// do filtering operations
...
```

Table 3: Comparison of performance metrics between implementations differing in the use of SIMD instructions. The results are obtained using *time profiling* for a period of 60 seconds, which takes a snapshot of the current call stack each millisecond. If a method happens to be in the snapshot, the count of the method is increased. The results are then multiplied by the time interval. *processBlock* is the main processing method of the audio callback, *Filter operations* include all processing concerning the filterbank. CPU usage was quantified using a performance meter within the digital audio workstation *Reaper*. The subsequent dynamic range compression within each band has been bypassed. Measurements were conducted on a Macbook Pro 6,2, which uses a 2,4 GHz Intel Core i5 with SSE4.1 and SSE4.2. For comparison, the average CPU usage of plug-ins of the IEM plug-in suite is (very roughly speaking) at about 2.6%, for 7$^{\text{th}}$-order Ambisonics.

		Time spent in methods	
SIMD	*CPU usage*	*processBlock*	*Filter operations*
no	$\sim 9.3\%$	$19.9\,s$	$18.6\,s$
yes	$\sim 3\%$	$4.16\,s$	$3.05\,s$

4.1.1 SIMD instruction set extensions

As stated above, basically all modern computers - at least all those one would want to produce audio on - support SIMD. In fact, the first instruction set for the x86 architecture to be widely deployed was introduced back in 1997. However, this instruction set (called *MMX*) provided only integer operations. Over time, the SIMD instruction set has been extended and the registers have grown. The Streaming SIMD Extensions (*SSE*), introduced in 1999, adds support for single-precision floating point operations by adding 8 or 16 (depending if a 32 or 64 bit system is used) 128-bit registers (called XMM), which allows to process 4 floats within one compute cycle. SSE2 extends the possible data types the XMM registers can hold and defines corresponding operations. SSE3 and SSSE3 add even more SIMD instructions, including support for "horizontal" operations - operations on distinct data that is stored in the same single register. SSE version 4 is another major enhancement, and is split into the subsets SSE4.1 and SSE4.2, which contribute 47 and 7 new instructions, respectively. In 2008, Intel proposed the Advanced Vector Extensions (AVX), which expands the width of the SIMD registers (now called YMM) from 128 bit to 256 bit and introduces many new features, e.g. 3 operand instructions which allow different destinations of results from 2 operands. Instruction prefixes allow backwards compatibility to SSE, which now uses the lower half of the 256 bit registers. AVX2 expands most integer commands to 256 bits and adds fused multiply-accumulate (FMA) operations. The newest version, introduced in 2013, is AVX-512, which widens the SIMD register size to 512 bits and is thus able to process up to 16 single-precision floats with a single instruction.

4.2 Phase compensation

Initially, the goal was to create a perfectly reconstructing multiband compressor, however, this conception has been dismissed, since it turned out to be quite difficult to preserve the original phase within the multichannel setting. As previously discussed in section 2.4, there are several approaches to phase compensation but none seems to be particularly well suited, because they either introduce imperfect magnitude responses or add extraordinarily heavy computational load and/or complexity. Given the low sensibility of the human ear to phase distortions and the relatively subtle impairments of the phase by Linkwitz-Riley crossovers, the costs seem to outweigh the benefits in the multichannel case.
Table 4 shows performance measures for an implementation that includes a 512-tap phase compensating FIR allpass filter, inserted into the processing chain right before writing the signal back into the output buffer of the audio callback. The allpass is derived from an impulse response, that describes the behaviour of the whole filterbank network, which is obtained from a cascade of 3 2^{nd}-order IIR allpasses (see 2.3). The impulse response is truncated after 512 samples and time-reversed. The FIR has been implemented with the fast convolution algorithm, which is more efficient than straightforward time-domain filtering, if the filter is of a high (≥ 64) order. Compared to the results of the SIMD implementation in Table 3, a striking increase in computational load can be observed, exemplifying the infeasibility of phase compensation. Moreover, this version of the plug-in produces serious audible glitches when the crossover frequencies are varied.

Table 4: Performance measurements for a phase-compensating version of the plug-in, using a 512-tap FIR Allpass implemented with fast convolution. The filterbank itself is optimized with SIMD instructions. Again, time profiling is performed for 60 seconds of processing, like in Table 3, and compressors are bypassed. Additionally, the host has been set to enforce a block size of 512 samples.

		Time spent in methods	
Phase compensation	*CPU usage*	*processBlock*	*Filter operations*
yes	$\sim 8.1\%$	$16.75, s$	$3.1\, s$
no	$\sim 3\%$	$4.16\, s$	$3.05\, s$

4.3 Compressor design

The class *Compressor*, originally designed for the *OmniCompressor* and *DirectionalCompressor* plug-ins, holds the processing logic for the dynamic range compression. It comprises of a peak-detector and a smooth branching ballistics filter, placed in the log-domain after the actual gain computer. This ensures accurate attack and release times, and yields an envelope that decays independent from the current compression gain, as discussed in section 3. In order to preserve the spatial image of the Ambisonic signal, only the omni channel (*W*-channel) of each band serves as the side-chain input to the compressors and the calculated gain is applied to all channels within the respective frequency band.

4.4 User interface

Considerable amount of work also went into the design and implementation of the graphical user interface (GUI), which is shown in Fig. 16.

The frequency bands are visualized using the magnitude responses of the corresponding crossover filters. Since the signal processing does not actually need the 4^{th}-order Linkwitz-Riley coefficients, they have to be computed via a (hard-coded) convolution of the Butterworth coefficients before the magnitude responses can be obtained. These are then drawn to the screen using the *Path* classes provided by JUCE.

Furthermore, they are displaying the gain changes introduced by the individual compressors and are updated every 50 milliseconds. As redrawing the Paths is not the cheapest of operations, the Message Thread, which is responsible for rendering the GUI on the CPU, will be strained to an undesirable extent. Thus, the design of the *FilterBankVisualizer* strives to only update the visualizations if necessary. The GUI also features an option to display the resulting total magnitude, but it is hidden by default, because the total magnitude will change pretty much all of the time, resulting in higher CPU drain.

All individual compressor characteristics are visualized as well. For convenience, each frequency band has its own parameter section and also features a meter displaying the current amount of gain reduction. Additionally, master controls were added, which enable the user to adjust parameters of all bands simultaneously, without breaking the ratios between them.

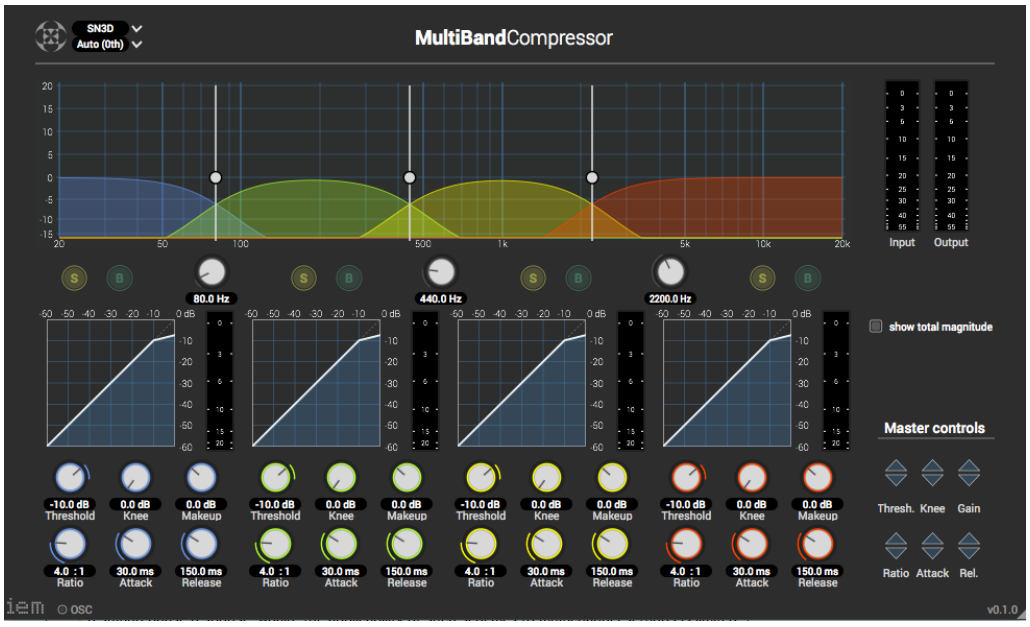


Figure 16: Screenshot of the GUI for the *MultiBandCompressor*.

5 Summary

This work described procedures with regard to the design of frequency selective dynamic range compression tools, in particular multiband compressors, for computationally demanding multi-channel environments, like Ambisonics. After introducing the corresponding concepts of dynamics processing in section 1, crossover filters for multiband compressors have been discussed, with a focus on the popular 4^{th}-order Linkwitz-Riley design. Additionally, it has been shown that linear-phase filter banks can hardly be realized in this multichannel framework, mostly due to their computational requirements, which has eventually been proven in the concluding section of this work. In the following, design considerations regarding digital dynamic range compression have been concisely discussed in section 3. Finally, the observations and choices made during implementing such a tool have been presented, with a focus on the utilization of single-instruction-multiple-data (SIMD) parallelism as well as its substantial effect on performance.

A Calculation of Digital Butterworth coefficients

The equation for an analogue normalized 2nd order Butterworth low-pass filter with crossover frequency Ω_c and unity DC gain can be written as [Orf95]

$$H(s) = \frac{1}{\frac{s^2}{\Omega_c^2} + \sqrt{2}\frac{s}{\Omega_c} + 1} \tag{23}$$

.

Multiplying by Ω_c^2 and pre-warping the crossover frequency by setting $\Omega_c \triangleq \omega_c = \frac{2}{T}K$ with $K = \tan(\frac{\pi f_c}{f_s})$ results in:

$$H(s) = \frac{(\frac{2}{T})^2 K^2}{s^2 + \frac{2}{T}K\sqrt{2}s + (\frac{2}{T})^2 K^2} \tag{24}$$

Now we can apply the bilinear transform given by $s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$.

$$H(s) = \frac{(\frac{2}{T})^2 K^2}{(\frac{2}{T})^2(\frac{1-z^{-1}}{1+z^{-1}})^2 + (\frac{2}{T})^2 K\sqrt{2}\frac{1-z^{-1}}{1+z^{-1}} + (\frac{2}{T})^2 K^2} \tag{25}$$

The $(\frac{2}{T})^2$ terms cancel out and after multiplying by $(1+z^{-1})^2$ and expanding the polynomials we arrive at

$$H(s) = \frac{K^2(1+2z^{-1}+z^{-2})}{(1-2z^{-1}+z^{-2}) + K\sqrt{2}(1-z^{-2}) + K^2(1+2z^{-1}+z^{-2})} \tag{26}$$

.

and we can again expand the terms and group them into coefficients with regard to the powers of z like so:

$$H(s) = \frac{K^2 + 2K^2 z^{-1} + K^2 z^{-2}}{1 + \sqrt{2}K + K^2 + 2(K^2-1)z^{-1} + (1 - \sqrt{2}K + K^2)z^{-2}} \tag{27}$$

Finally, we normalize for a unity a_0 coefficient.

$$H(s) = \frac{\frac{K^2}{1+\sqrt{2}K+K^2} + \frac{2K^2}{1+\sqrt{2}K+K^2}z^{-1} + \frac{K^2}{1+\sqrt{2}K+K^2}z^{-2}}{1 + \frac{2(K^2-1)}{1+\sqrt{2}K+K^2}z^{-1} + \frac{1-\sqrt{2}K+K^2}{1+\sqrt{2}K+K^2}z^{-2}} \tag{28}$$

The LR-Highpass filter coefficients can be derived in a similar fashion.

References

[AB07] V. Adam and S. Benz, "Correction of crossover phase distortion using reversed time all-pass iir filter," *AES Convention 2007*, vol. 2, 2007.

[DPL98] B. Djokic, M. Popovic, and M. D. Lutovac, "A new improvement to the powell and chau linear phase iir filters," *IEEE Trans. Signal Processing*, vol. 46, 1998.

[FF10] A. Favrot and C. Faller, "Complementary n-band iir filterbank based on 2-band complementary filters," 2010.

[GMR12] D. Giannoulis, M. Massberg, and J. D. Reiss, "Digital dynamic range compressor design - a tutorial and analysis," *JAES*, vol. 60, no. 6, 2012.

[Koy00] D. Koya, "Aural phase distortion detection," Master's thesis, University of Miami, 2000.

[Lin76] S. Linkwitz, "Active crossover networks for non-coincident drivers," *JAES*, vol. 24, no. 1, 1976.

[LPV82] S. P. Lipshitz, M. Pocock, and J. Vanderkooy, "On the audibility of midrange phase distortion in audio systems," *JAES*, vol. 30, no. 9, 1982.

[MV17] L. McCormack and V. Välimäki, "Fft-based dynamic range compression," 07 2017.

[Orf95] S. J. Orfanidis, *Introduction to Signal Processing*. Prentice-Hall, Inc., 1995.

[PC91] S. R. Powell and P. M. Chau, "A technique for realizing linear phase iir filters," *IEEE Trans. Signal Processing*, vol. 39, 1991.

[ran] "Linkwitz-riley crossovers: A primer." [Online]. Available: https://www.rane.com/note160.html

[Smi11] J. O. Smith, *Spectral Audio Signal Processing*, 2011. [Online]. Available: http://ccrma.stanford.edu/~jos/sasp/

[Vic16] M. Vicanek, "A new reverse iir filtering algorithm," 2016.